Université Grenoble alpes.
Auteur :Frédéric Faure
Version : 22 septembre 2017

Didacticiel
Bases de programmation en C++

Table des matières

Déroulement des séances en Licence L3 de physique, 2017-2018

— Suivre le didacticiel **Partie I "Bases du langage C++"** I, en binome ou seul.
— Lire l'introduction 1, puis passer à la partie I, I.

- Faire tous les exercices demandés. Pour les exercices marqués par (*). Par exemple **"tabulation"** (*), écrire un programme tabulation.cc et le montrer à l'enseignant (ou montrer plusieurs exercices à la fois).
- On peut seulement lire rapidement les parties appelées "suppléments".
- Ce didacticiel se termine par la réalisation d'un microprojet en Section 9. Choisir parmi la liste proposée. Rédiger un compte rendu de 2 ou 3 pages avec lyx, avec des images et des formules. Exporter en pdf et en xhtml et le montrer à l'enseignant, avec un oral de 5mn-10mn.
- Optionellement, suivre le didacticiel **Partie II "Compléments du langage C++"** II.

— Choisir un **projet d'expérimentation numérique.** Voir la page **projets**.

- A la dernière séance, chaque étudiant (ou binome) présente son projet sur vidéo projecteur, pendant 15 à 20 mn, à l'ensemble de la classe.
- Cette présentation devra être préparée avec Lyx (qui permet de créer des documents scientifiques) et exportée en html (avec LyxHtml) ou en pdf.
- Pour apprendre Lyx, voici un document en pdf qui sert d'exemple et d'exercice. Voici le même document en html, qui vous servira éventuellement pour copier/coller.

Chapitre 1

Introduction

- Ce didacticiel vous permettra d'appréhender le langage de programmation C++ (version C++11) et ses avantages. Dans ce didacticiel, les exercices marqués par (*) seront vérifiés par l'enseignant pendant les séances. Il y a de nombreux "liens html" qui renvoie vers plus d'informations.
- Le grand avantage du langage C++ par rapport au langage C est **la programmation objet**. On peut utiliser des objets appartenant à des classes déjà existantes ou à des classes que l'on a fabriqué soi-même. Par exemple, il existe la classe des nombres complexes. Les objets appartenant à cette classe sont des nombres complexes. Il nous suffira de déclarer trois objets `a, b` et `c` comme appartenant à la classe des nombres complexes, et l'on pourra directement écrire `a=b+c` ou n'importe quelle autre expression mathématique. Comme autre exemple, il y a la classe de "matrice complexe" : on peut déclarer trois objets `A, B` et `C` appartenant à cette classe et écrire `A=B+C` ou n'importe quelle autre expression mathématique matricielle.
- **Pourquoi enseigner le C++ à l'université, dans la filière physique ?** d'une part c'est un langage qui est très bien adapté pour la simulation ou l'analyse de problèmes physiques et de façon plus générale pour le travail scientifique. Dans les laboratoires il a remplacé peu à peu le langage *fortran*. Le langage *python* est un autre langage qui utilise aussi des classes et est aussi très utilisé dans le domaine scientifique. Un programme en python est peut être plus simple à mettre en oeuvre qu'un programme en C++, **c'est une histoire de goût et d'habitude**, mais il est aussi beaucoup plus lent à l'exécution. Un avantage du langage python est qu'il est "interprété" : on peut exécuter un programme ligne après ligne alors que le langage C++ est "compilé" : un programme appelé compilateur le transforme en langage machine (fichier exécutable) avant qu'il ne soit exécuté et cela le rend rapide. La dernière version C++11 fait du C++ un langage aussi conviviale et simple d'utilisation que le langage python (et plus rapide !).
- Nouvelles version du C++:les récentes versions C++11 puis C++14, C++17, C++20 ont pour vocation de rendre le C++ plus facile à apprendre et toujours performant.
- Références :
  - **Sur le web :**

  - voici un site très utile : http://www.cplusplus.com/. Il contient un tutorial de très bonne qualité, où les exemples peuvent être essayé en ligne (avec la petite icône "*edit & run*" à droite de *example*, essayez par exemple sur setprecision) !. Il contient aussi une documentation complète au langage C++.
  - Un autre site en français :cppreference.com ou en anglais : cppreference.com.
  - Une façon pratique de se documenter sur une fonction précise est d'utiliser le moteur de recherche de google. Par exemple pour afficher un nombre avec une certaine précision numérique, si vous ne savez pas comment faire, tapez quelques mots clef comme "affichage precision c++" dans la fenêtre de recherche de google, qui vous amènera sur une page de forum avec un exemple. Vous y verrez que la fonction à utiliser est `setprecision`. Ensuite pour avoir une bonne documentation sur cette fonction et des exemples que l'on peut copier/coller, tapez toujours dans google "cplusplus setprecision" ce qui vous amène à une page de documentation.
  - Signalons le cours en ligne en francais sur le C++ de Coursera sur : ce sont des vidéos courtes, ou Cours de C++ de OpenClassRoom en français.
  - La page web de Bjarne Stroustrup, auteur du langage C++, et en particulier la FAQ sur le C++11
  - Un MOOC sur Linux. Un MOOC sur le C++. Un autre MOOC sur le C++.
- **Livre pour le langage C++ :**
  - *Programmer en C++* de Claude Delannoy, éditions Eyrolles
  - *The C++ programming langage* de Bjarne Stroustroup, 4eme édition.
- **Livre pour le langage C :** *Le langage C* de B.W. Kernighan et D.M. Ritchie edition Masson.

1.1 Choix du système d'exploitation

Ce didacticiel est indépendant du système d'exploitation. Cependant en séance de TP on travaille avec **linux** et plus particulièrement avec la distribution Xubuntu ou Ubuntu. On conseille cette distribution gratuite qui peut être installée sur votre ordinateur personnel en suivant ce lien : installer Ubuntu.

Il y a cependant possibilité d'utiliser d'autres systèmes comme windows, mac-os, etc...

Dans ce didacticiel, nous utiliserons les logiciels et librairies suivantes qui sont en général téléchargeables depuis votre distribution :

- Root.
- Armadillo.

1.1.1 Installer Xubuntu et l'environnement de programmation c++ sur son ordinateur

- Si votre ordinateur marche avec windows, vous pouvez installer rajouter ubuntu (linux) soit avec une machine virtuelle virtualbox ou installer Ubuntu en double

boot (Suivre les instructions sur installer Ubuntu.) ou installer directement ubuntu sous windows et doc sur WSL.

— Si ubuntu (linux) est déjà installé sur votre ordinateur, il faut installer les librairies requises pour ce didacticiel. Pour cela, ouvrir un terminal et écrire, l'une après l'autre les lignes suivantes :

```
sudo apt-get install -y root-system # librairie Root du Cern
sudo apt-get install -y emacs emacs-goodies-el global clang auto-complete-el
cscope-el # logiciel pour recherche de fonctions etc
sudo apt-get install -y libarmadillo-dev # C++ linear algebra
library
sudo apt-get install -y libboost-dev
sudo apt-get install -y libboost-all-dev
```

— Autre info:.

1.2 Environnement de programmation

Avec un petit exemple nous allons voir comment :

1. **écrire** le code d'un programme en C++
2. le **compiler**, c'est à dire que l'ordinateur traduit le programme C++ en un programme "exécutable" (en langage machine) qu'il pourra exécuter ensuite.
3. **exécuter le programme**
4. éventuellement de le **"débugger"** (cela signifie recherche d'erreurs) : si le programme ne fonctionne pas comme on le désire, on l'exécute instruction après instruction afin de suivre ce qu'il fait pas à pas, dans le but de trouver un "bug" (= une "erreur").

Dans la Section suivante nous effectuons ces taches avec l'éditeur emacs et une fenêtre de commandes (terminal).

1.2.1 Utilisation de commandes dans un terminal

Si vous avez une question sur l'utilisation de ubuntu, par exemple comment ouvrir et utiliser un terminal, écrire dans google : "ubuntu terminal".

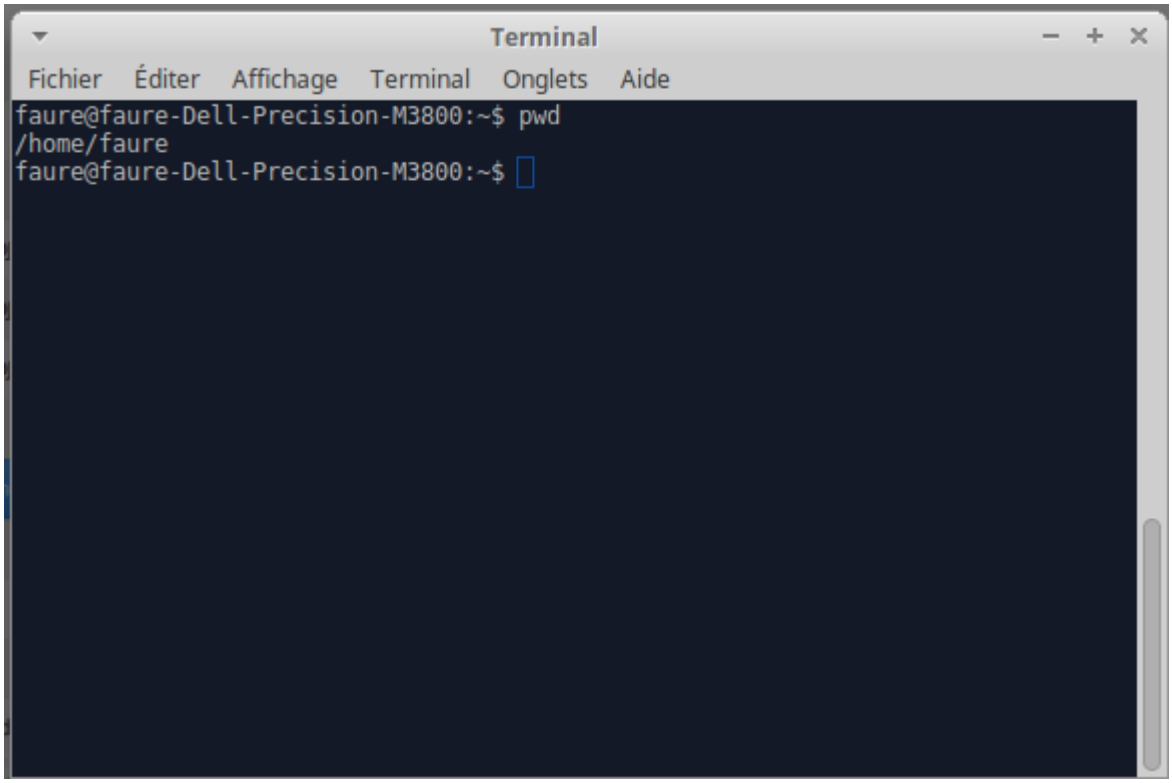
Dans ubuntu, ouvrir une fenêtre de **commandes** (ou **terminal**). Dans cette fenêtre de commandes, vous pouvez écrire des *commandes* à l'ordinateur. Ces commandes sont dans le langage UNIX, système avec lequel fonctionne l'ordinateur. Cette section vous explique les principales commandes de UNIX qu'il faut connaître : les commandes qui permettent de gérer l'emplacement et le nom de vos fichiers (qui seront par exemples vos programmes en C++).

Il est important de savoir que les fichiers sont classés dans l'ordinateur (sur le disque magnétique) selon une structure arborescente. Chaque noeud de l'arborescence s'appelle un **répertoire** ou **dossier** ou **directory** (en anglais). Dans ces répertoires se trouvent des **fichiers** (ou **file** en anglais).

Par exemple `/h/moore/u4/phyliphrec/aaal/toto.cc` signifie que le fichier `toto.cc` se trouve dans le répertoire `aaal` qui se trouve lui même dans le répertoire `phyliphrec`, qui se trouve lui même dans ...

1.2.1.1 Les répertoires

— Pour connaître le répertoire de votre fenêtre de commande (le répertoire courant) : `pwd`
sur la figure suivante par exemple, le répertoire courant est `/home/faure`



— Pour **afficher la liste des fichiers** et des sous répertoires du répertoire courant : `ls` ou `ls -als` (pour avoir plus de détails)
— Pour **changer de répertoire** :
  — pour aller dans le sous répertoire appelé rep : `cd rep`
  — pour aller dans le répertoire parent : `cd ..`
  — pour revenir à votre répertoire d'origine : `cd` ou `cd ~`
— Pour **créer un sous répertoire** appelé rep : `mkdir rep`
— Pour **détruire un sous répertoire** appelé rep : `rm -r rep`

1.2.1.2 Exercices sur les répertoires

A l'aide des commandes expliquées ci-dessus :

1. Placez vous dans votre répertoire d'origine : `cd`
2. Vérifiez que vous y êtes bien : `pwd`
3. Regardez la liste des fichiers par : `ls ,` puis avec : `ls -als`
4. Créer un répertoire du nom : `essai` Vérifier son existence (avec `ls`).
5. Aller dans ce répertoire : `cd essai` Vérifier que vous y êtes (avec pwd).
6. Revenir dans le répertoire d'origine par `cd ..` Vérifiez que vous y etes. Détruire le répertoire `essai`

1.2.1.3 Les fichiers (en exercice)

— **Créer un nouveau fichier** avec l'éditeur emacs : `emacs fichier.txt &`
écrire le texte : `salut`
Menu : `File/Save`
Menu : `File/Quit`
On **vérifie le contenu** du fichier dans la fenêtre de commandes : `more fichier.txt`
— Pour **changer le nom** du fichier.txt en fichier2.txt : `mv fichier.txt fichier2.txt`
— Pour **déplacer** le fichier2.txt dans le sous répertoire rep :
```
mkdir rep
mv fichier2.txt rep/fichier3.txt
```
— Pour **copier** le fichier3.txt dans le répertoire parent :
```
cd rep
cp fichier3.txt ../fichier1.txt
```
— Pour **détruire** le(s) fichiers :
```
rm fichier3.txt
cd ..
rm fichier1.txt
rm -r rep
```

1.2.1.4 La documentation sur les commandes unix

— Pour avoir la documentation sur la commande `ls` par exemple, taper dans une fenetre de commande : `man ls`
— Voici une documentation sur les commandes possibles bash pour la fenêtre terminal.

1.2.2 Programmer avec l'éditeur emacs et une fenêtre de commandes

1. Dans ubuntu, ouvrir une fenêtre de **commandes** (ou **terminal**). Se placer dans le bon répertoire de travail. Pour cela comme expliqué dans la section précédente, utiliser les commandes `pwd` (qui dit dans quel dossier on est situé), `ls` (qui liste les fichiers et dossiers) , `cd dossier` (: qui passe au dossier indiqué), `cd ..` (qui

passe au dossier parent), `cd .` (qui reste dans le même dossier, c'est inutile), `cd` (qui revient au dossier de départ),

2. Ecrire la commande `emacs &` qui lance l'éditeur. Le signe & signifie que le terminal reste utilisable. Dans emacs, une fois pour toutes, cocher dans le menu `Options/UseCUAKeys` et `Options/SaveOptions` qui vous permettra d'utiliser les raccourcis clavier C-c C-v pour "copier/coller".

3. Dans emacs, ouvrir un nouveau fichier `projet1.cc` qui sera vide la première fois (pour faire cela : `Menu/File/Visit New File` et ecrire `projet1.cc` , Valider). Le suffixe .cc signifie qu'il contiendra du code C++ (ou peut aussi choisir le suffixe .cpp). Dans ce fichier, Copier/coller le code suivant :

```
#include <iostream>
using namespace std;
int main()
{
cout << "Bonjour!" << endl;
}
```

   et sauvegarder ce fichier qui contient le **code C++ de votre programme**. (Observer les raccourcis clavier proposés par emacs dans ses menus pour toutes ces commandes : par exemple "`Save : C-x C-s`" signifie que les touches `Ctrl` avec `x` puis `Ctrl` avec `s`. "Shell Command M-!" signifie les touches `Alt` avec `!`).

4. Dans emacs ouvrir un nouveau fichier `Makefile` qui sera vide la première fois. Dans ce fichierCopier/coller le code de compilation suivant :

```
all:
    g++ projet1.cc -o projet1 -std=c++11 -O3 -Wall -lm
```

(1.1)

   et sauvegarder ce fichier qui contient les instructions pour compiler votre programme. **Attention :** au début de la deuxième ligne rajouter un **espace de tabulation** avant g++. Remarque : on peut basculer d'un fichier à l'autre dans emacs par le menu "Buffers" ou par le clavier avec `C-x C-`$\rightarrow$.

5. **Compiler le programme** par la commande du menu : Tools/Compile.. qui fait apparaitre en bas de page : `make -k` que l'on remplace par `make all` (puis faire entrée). Si tout se passe bien le dernier message est "compilation finished".

6. Remarques : cela a pour effet d'exécuter la commande de compilation qui se trouve dans le fichier Makefile : elle dit qu'il faut utiliser le compilateur g++ et qu'à partir du fichier `projet1.cc` il faut créer en sortie (-o = "output") un fichier exécutable appelé `projet1`. On a rajouté en option (inutiles dans cet exemple) `-std=c++11`

qui signifie que on utilise la syntaxe du c++11 et `-lm` qui signifie que on l'on souhaite utiliser la librairie mathématique. L'option `-Wall` signifie que le compilateur montrera "**all** the **W**arnings" , c'est à dire toutes les imperfections dans la syntaxe.

7. Dans le terminal, placez vous dans le répertoire qui contient votre projet, vérifiez avec la commande `ls` qu'il contient le fichier exécutable `projet1` , et **exécutez ce fichier** en écrivant : `./projet1`
   (remarque : `./` signifie le répertoire présent)

Exercice 1.2.1.

1. Dans le programme `projet1.cc` ajoutez une erreur, par exemple en enlevant un signe ; en fin de ligne. Recompiler le programme et observer que le compilateur détecte l'erreur et en cliquant sur le message d'erreur, emacs vous place à la ligne où est l'erreur à corriger.
2. Dans les instructions ci-dessus nous vous avons proposé d'utiliser un fichier Makefile qui sera surtout utile pour les projets de programmation en C++. Vous pouvez plus directement compiler votre programme en écrivant dans un terminal la commande de compilation :

   ```
   g++ projet1.cc -o projet1 -std=c++11 -O3 -Wall -lm
   ```

   le désavantage est que pour corriger l'erreur il faut soit même se placer sur la ligne indiquée.

   Une autre possibilité pour compiler est d'écrire la commande `make all` dans le terminal qui a pour effet d'exécuter le fichier `Makefile`.
3. `emacs` peut mettre votre programme "en forme" : pour cela sélectionnez tout le texte et appuyez sur la touche tabulation. Il apparaît des décalages de lignes convenables, mais non obligatoires.
4. Remarque : pour écrire le programme, à la place de emacs vous pouvez utiliser tout autre éditeur de texte.

1.2.2.1 Paramétrage de emacs

Voici quelques paramétrages utiles pour la suite :

— En haut du fichier .cc rajouter la ligne suivante :
`// -*- mode:C++ ; compile-command: "make all" -*-`
Cela a pour effet de préciser la commande de compilation lorsque l'on fait Menu/-Tools/Compile
On peut aussi remplacer `"make all"` par `"make all; projet1;"` ce qui a pour effet de compiler et de lancer l'exécutable `projet1`

— Dans le fichier ~/.emacs rajouter la ligne :
`(global-set-key (kbd "<f9>") 'compile)`
Ainsi il suffira dans emacs d'utiliser la touche f9 pour lancer la commande de compilation.

— Autres options, voir developper en c++ sous emacs.

Première partie

Bases du langage C++

Chapitre 2

Le tout début. Lire le clavier et afficher à l'écran

2.1 Affichage dans un terminal

Voici un petit programme en C++ qui additionne deux nombres entiers a et b , affecte le résultat dans c et affiche le résultat à l'écran.

```
#include <iostream>
using namespace std;

/* ==============
Programme principal
====================*/
int main()
{

    int a,b; // déclaration des objets a et b de la classe
 int
    a=1; // affectation: a prend la valeur 1
    b=a+1; // affectation b prend la valeur 2
    int c; // déclaration de l objet c de classe int
    c=a+b; // affectation: c prend la valeur a+b c'est à dire
 3

    // affichage à l'écran:
    cout << "la somme de " << a << " et " << b << " vaut "
 << c <<endl;

}
```

Exercice 2.1.1. Recopiez ce programme dans un fichier `prog.cc` , modifier la commande de compilation de `Makefile` pour le compiler et exécutez le.

Résultat du programme : `la somme de 1 et 2 vaut 3`

Remarque 2.1.2. sur la syntaxe du programme :

1. La classe `int` permet justement de stocker des **nombres entiers**. (Cela vient de *integer* en anglais).
2. Le **point virgule** sépare les instructions.
3. Les **commentaires** doivent débuter par //. Le reste de la ligne est alors considéré comme un commentaire. Un commentaire n'a aucune influence sur le programme, il sert en général à expliquer ce que fait le programme pour le rendre compréhensible au lecteur. Une autre possibilité est de commencer le commentaire par `/*` et de finir quelques lignes après par `*/`.
4. Le début et la fin d'un **bloc d'instructions** sont respectivement { et }
5. "`main` " veut dire "principal " en anglais. C'est le début du **programme principal** ou de la fonction principale. Les parenthèses () signifient que le programme principal n'utilise aucun paramètre dans cet exemple. Lorsque vous lancez votre programme, son exécution commence toujours par la première ligne de cette fonction principale qui s'appelle toujours `main`.

Remarque 2.1.3. **sur l'affichage :**

1. Pour l'affichage de texte dans un terminal, on `utilise` l'objet `cout`. Cet objet `cout` représente le terminal. Les signes << sont évocateurs : on envoit ce qui suit vers l'écran. On peut enchainer les objets que l'on envoit à l'écran en écrivant sous la forme : `cout << A<<B<<C<<D;` . Le symbole `endl` signifie que l'on va à la ligne (end of line = fin de ligne).
2. Les caractères entre guillements " " sont considérés comme une chaîne de caractères et écrits tels quels à l'écran . Par contre les caractères `a, b,c` ne sont pas entre " ". Cela signifie que ce sont des noms d'objets, et qu'il faut afficher le contenu de ces objets (et non pas leur nom).
3. Ce symbole `cout` appartient à la bibliothèque `iostream` qui est chargée grâce à la première ligne `#include <iostream>. iostream` est un fichier déjà présent sur l'ordinateur. Ce fichier contient des informations sur des commandes C++ d'affichage à l'écran et de saisie de touches appuyées au clavier. Iostream vient de l'anglais : Input (=entrée) , Output (=sortie) , Stream (=flux d'information). En principe pour l'affichage il faudrait écrire `std::cout` car le symbole `cout` appartient à "l'espace de nom" appelé `std` comme "standard" ; mais la deuxième ligne `using namespace std;` signifie que l'on pourra écrire plus simplement `cout.` (Il peut être utile de ne pas faire cette simplification si le même symbôle `cout` est utilisé pour autre chose par une autre bibliothèque dans un autre espace de nom comme `std2` . On pourra alors les distinguer en écrivant : `std::cout` et `std2::cout`).

Exercice 2.1.4. "tabulation" (*) `"\t"` permet d'afficher une **tabulation** (i.e. sauter un espace jusqu'à la colonne suivante). Modifier le programme précédent pour afficher :

```
objets:      a     b     c=a+b
valeurs:     1     2     3
```

Remarque 2.1.5. Voici la liste des autres caractères spéciaux comme la tabulation "\t", le retour à la ligne suivante "\n", le retour au début de ligne "\r", le caractère "\" lui même par "\\", etc.

2.2 Lire le clavier

Il peut être intéressant que l'utilisateur puisse lui-même entrer les valeurs de `a` et `b`. Pour cela l'ordinateur doit attendre que l'utilisateur entre les données au clavier et les valide par la touche *entrée*.

```
#include <iostream>
using namespace std;

int main()
{
    int a,b; // déclaration des objets a et b de la classe
 int
    cout <<"Quelle est la valeur de a? "<<flush;
    cin >> a; // lire a au clavier, attendre return
    cout <<"Quelle est la valeur de b? "<<flush;
    cin >> b; // entrer b au clavier puis return
    int c; // déclaration de la objet c de la classe int
    c = a + b; // affectation: c prend la valeur a+b

    // affichage à l'écran:
    cout <<"la somme de "<<a<<" et "<<b<<" vaut "<<c<<endl ;
}
```

Exercice 2.2.1. Recopier ce programme et exécuter le.

Remarque 2.2.2.

1. L'objet `cin` appartient à la classe `istream` et représente le clavier. Le signe `>>` est un opérateur associé à cet objet et qui a pour effet de transférer les données tapées au clavier dans l'objet qui suit (une fois la touche *entrée* enfoncée).
2. L'instruction `flush` à la fin de la phrase d'affichage à pour effet d'afficher la phrase à l'écran sans faire de retour à la ligne. Si on ne met rien (ni `flush`, ni `endl`) la phrase n'apparait pas forcément tout de suite à l'écran.

Chapitre 3

Déclaration et affectation des objets

3.1 Objets de base

3.1.1 Les déclarations d'objets de base

dans un programme, pour stocker une information, on utilise un objet qui est symbolisé par une lettre ou un mot. (Comme `a,b,c` précédement). On choisit la **classe** de cet objet, selon la nature de l'information que l'on veut stocker (nombre entier, nombre à virgule, nombre complexe, ou série de lettres, ou matrice, etc).

Voici quelques classes de base qui existent en C++. Il y en a d'autres, voir variables :

Déclaration : `classe objet;`	signification	Limites
`int a;`	nombre entier	-2147483648 à 2147483647
`double d;`	nombre réel double précision	$\pm 10^{\pm 308}$ à 10^{-9} près
`char e;`	caractère	
`bool v;`	booléen : vrai (true) ou faux (false)	
`char * t;`	chaîne de caractères	

Remarque 3.1.1. Les classes ci-dessus sont les classes de base standard du langage C++ qui existent aussi en langage C. Dans le vocabulaire du C, on dirait plutot **type** à la place de **classe**, et **variable** à la place de **objet**. Par exemple en écrivant `double d;` on dirait que `d` est une variable du type `double`.

3.1.2 Initialisation d'un objet de base

On peut déclarer un objet et l'initialiser en même temps de différentes manières. Voici un exemple.

```
#include <iostream>
```

```
using namespace std;

main ()
{
int i = 0;
double x = 1.23;
double x1(3.4),x2(6); // equivalent a x1=3.4
double y(x1), y2 = x2;
char Mon_caractere_prefere = 'X';
const char * texte = "que dire de plus?";
char c = texte[0]; // premier caractere
auto z = x + 2;
{
double x=5;
}
}
```

3.1.2.1 Remarques

1. i est un `int` qui vaut 0, `x` un float qui vaut 1.23, `x1` est un `double` qui vaut 3.4, etc..
2. Le terme `auto` signifie que le compilateur devine lui même la classe de l'objet, ici `z` est un `double`.
3. Dans le nom des objets, le langage C++ fait la différence entre les majuscules et les minuscules. On peut choisir ce que l'on veut et utiliser même des chiffres et le caractère `_`.
4. On peut initialiser un objet avec des paranthèses comme `x1(3.4)`. C'est équivalent que d'écrire `x1=3.4`. On peut initialiser un objet avec un objet déjà existant comme `y(x1)` ou `y2 = x2`.
5. Un caractère est entre le signe ', comme `'X'`.
6. Un texte (chaîne de caractère) est entre le signe ". Ici l'objet `texte` est du type `const char *` qui signifie que c'est une "chaine de caractères" (ou tableau de caractères) constante. Les indices du tableau commencent à `0`. C'est pourquoi `texte[0]` extrait le premier caractère qui est q. On expliquera plus tard que texte est en fait un pointeur sur charactères.
7. A la dernière ligne on déclare $x = 5$ dans un **bloc d'accolades** ou **bloc d'instructions** {..}. Cela est possible et signifie que dans ce bloc x est une nouvelle variable locale (qui masque la valeur de x précédente).
8. On pourra bien sûr modifier ces valeurs dans la suite du programme.

Exercice 3.1.2. "accolades" (*) Recopiez le programme, modifiez le pour qu'il affiche la valeur de chaque objet, et exécuter le. (si "auto" signale une erreur, vérifiez si l'option

c++11 est cochée, voir remarque de l'exercice 10.1.2(4).) Afficher la valeur de x dans le bloc d'accolades qui contient `double x=5;` et après ce bloc d'accolades. Dans un deuxième temps, essayer de ne pas mettre d'accolades et comprendre l'erreur de compilation.

3.1.3 Affichage des nombres avec une précision donnée

(il faut inclure `<iomanip>` et `<math.h>`)

```
#include <iostream>
using namespace std;
#include <math.h>
#include <iomanip>

//.................
main ()
{
double c = M_PI;
cout<<setprecision(3)<<c<<endl;
}
```

Résultat :

```
3.14
```

Référence : voir setprecision.

3.1.4 Attention à la division Euclidienne

Voici un exemple :

```
#include <iostream>
using namespace std;
//......
main ()
{
 int a=7, b=2;
 double c=7.;
 cout<<"7/2 = "<<a/b<<" = "<<7/2<<endl;
 cout<<"7./2 = "<<c/b<<" = "<<7./2<<" = "<<(double)a/b<<endl;
 cout<<"Le quotient de 7/2 = "<<7/2<<" Le reste de 7/2 est r= "<<7%2<<endl;
}
```

Résultat :

```
7/2 = 3 = 3
7./2 = 3.5 = 3.5 = 3.5
Le quotient de 7/2 = 3 Le reste de 7/2 est r= 1
```

Remarques :

- L'exemple précédent montre que l'écriture `7/2` signifie le quotient de la **division euclidenne** qui est bien 3 car `7=2*3+1`. Le reste 1 euclidien est obtenu par l'opération 7%2 qui signifie 7 **modulo** 2.
- Pour effectuer la division parmi les nombres réels (à virgule) il faut utiliser le type `double`.
- L'instruction `(double)a` permet de convertir la variable a qui est de classe `int` en une variable de classe `double`. Ce changement de classe est appelé "**cast**".
- Attention, cette convention que `7/2` est la division euclidienne en langage C et C++ est souvent source d'erreur.

3.1.5 Pour générer un nombre entier p aléatoire

```
#include <iostream>
using namespace std;

#include <time.h>
#include <stdlib.h>

main ()
{
 srand(time(NULL)); // initialise le hasard.(faire cela une seule
fois en debut de programme)
 int N = 100;
 int p = rand()%(N+1); // -> entier aléatoire entre 0 et N compris.
 cout<<" p="<<p<<endl;
}
```

Remarque 3.1.3. `rand()` génère un nombre entier aléatoire entre 0 et 32768. Ensuite `%` signifie modulo, `a%b` est le reste de la division de `a` par `b`. Ainsi `rand()%(N+1)` est le reste de la division du nombre aléatoire par `N+1`. C'est donc un entier compris entre 0 et N (inclus). En c++11 il y a une classe random plus élaborée pour générer des nombres aléatoires selon diverses lois.

3.2 Objets plus élaborés de la classe standard

Nous présentons ici quelques classes plus élaborées que les classes de base et très utiles. Pour une liste complète des classes d'objets possibles en C++, dans la librairie standard (STL, Standard Template Library) voir par exemple la référence.

3.2.1 Chaines de caractères (string)

Référence : string. Les chaines de caractères permettent de manipuler des suites de charactères, comme des phrases.

Voici un exemple de programme que vous pouvez essayer.

```
#include <iostream>
#include <string>
using namespace std;
main ()
{
    string texte1 = "J'aime le café";
    cout << texte1<<endl;
    string texte2 = texte1 + " sans sucre";
    cout << texte2<<endl;
    cout<<"Le premier caractère est du texte précédent est:
 "<<texte2[0]<<endl;
}
```

Résultat :

```
J'aime le café
J'aime le café sans sucre
Le premier caractère du texte précédent est:J
```

Remarque 3.2.1. L'indice du premier caractère est 0. (En informatique on numérote souvent à partir de 0). Voici plus d'informations et d'exemples sur la classe string.

Exercice 3.2.2. "texte" (*) Modifier le programme précédent pour qu'il affiche :

```
J'aime le café sans sucre
Le troisième caractère du texte précédent est:a
```

3.2.1.1 Conversion de chiffres en chaine de caractère et inversement. Solution simple.

Avec les fonctions stoi (penser "**S**tring **to** **I**nteger") stod (penser "**S**tring **to** **D**ouble") et to_string. On souhaite par exemple construire une chaine de caractères qui contient le résultat d'une opération numérique.

```
#include <iostream>
using namespace std;
#include <string>
#include <sstream>

//.....
main ()
{
//..... Conversion number to string.
int d=7; // chiffre
string S = to_string(d);
cout<<"d="<<d<<" S="<<S<<endl;

//..... Conversion string to number.
string s5 = "3"; // chaine contenant des chiffres
int i5 = stoi(s5);
string s6 = "3.14"; // chaine contenant des chiffres
double d6 = stod(s6);
cout<<"i5="<<i5<<" d6="<<d6<<endl;
}
```

Résultat :

```
d=7 S=7
i5=3 d6=3.14
```

3.2.2 Nombres complexes

Référence : complex.

```
#include <iostream>
using namespace std;
#include <complex>
typedef complex<double> Complex;

main ()
{
    Complex I(0,1); // on définit le nombre I
    cout<<"I.I="<<I*I<<endl;
    Complex a=2.+I*4., b=3.-2.*I;
    Complex c=a+b, d=a*b;
    cout<<" a="<<a<<" b="<<b<<endl;
    cout<<" a+b="<<c<<" a*b="<<d<<endl;
```

```
        cout<<" real(a)="<<a.real()<<" norme(a)="<<abs(a)<<endl;
}
```

Résultat :

```
I.I=(-1,0)
a=(2,4) b=(3,-2)
a+b=(5,2) a*b=(14,8)
real(a)=2 norme(a)=4.47214
```

Remarque 3.2.3.

- L'écriture `2.` est équivalente à `2.0` ou à `2`. Ainsi `a=2.+I*4.` est équivalente à `a=2+4*I`
- `typedef complex<double> Complex;` signifie que le mot `Complex` est un raccourci pour le type `complex<double>` qui est un nombre complexe avec `double` en partie réelle et imaginaire.

3.2.3 Liste d'objets (ou tableau de dimension 1) : vector

La classe `string` ci-dessus est une liste de caractères. Plus généralement la classe `vector` permet de manipuler des listes d'objets quelconques (des nombres ou autres). Voici les informations sur la classe vector.

Par exemple pour une liste de nombres entiers avec `vector<int>` :

```
#include <iostream>
using namespace std;
#include <vector>

main ()
{
vector<int> L; // liste vide qui contiendra des objets de type int
L.push_back(2); //rajoute l'element 2 a la fin de la liste
L.push_back(3); // rajoute 3 a la fin de la liste
L.push_back(5);
cout<<"La liste a la taille :"<<L.size()<<endl;
cout<<"Ses elements sont: "<<L[0]<<" , "<<L[1]<<" , "<<L[2]<<endl;
}
```

Résultat :

```
La liste a la taille :3
Ses elements sont: 2 , 3 , 5
```

Remarque 3.2.4. Les indices de la liste commencent à 0.

Autre exemple :

```
#include <iostream>
using namespace std;
#include <vector>
#include <algorithm>

main ()
{

//--- recherche la position de l'element maximal de la liste
vector<int> L={5,7,3,12,34,6}; // on initialise une liste d'entiers
cout<<"Le premier élément de la liste est "<<L[0]<<endl;
int p = max_element(L.begin(),L.end())-L.begin(); //renvoit la position
du maximum dans l'ordre (0,1,2..)
cout<<"L'élement le plus grand est "<<L[p]<<endl;
cout<<"Il est à la position "<<p<<endl;

//--- cherche la premiere/derniere position d'un element avec find()
/ rfind()
int e = 6;
int p2 = find(L.begin(), L.end(), e) - L.begin(); // cherche la
position de l'element e
if(p2<L.size())
cout<<"Le "<<e<<" est à la position "<<p2<<endl;
else
cout<<"Le "<<e<<" n'est pas dans la liste."<<endl;
}
```

Résultat :

```
Le premier élément de la liste est 5
L'élement le plus grand est 34
Il est à la position 4
Le 6 est à la position 5
```

Remarque 3.2.5. Dans l'exemple ci-dessus, l'objet `i` qui représente une position sur la liste est de la classe `vector<int>::iterator`. On utilise ici le terme `auto` qui détecte cette classe et est plus simple à écrire. Voici plus d'informations et d'exemples sur la classe vector. Pour plus d'informations et d'exemples sur les manipulations de listes (appelées `containers`) voir algorithms. Reference sur max_element.

3.3 Vecteurs et matrices pour l'algèbre linéaire avec la librairie Armadillo

Pour cela on utilise la " librairie armadillo" dont voici la documentation.

Commande de compilation : Cette librairie ne fait pas partie de la "librairie standard STL", (si vous utilisez votre propre ordinateur, il faut donc l'installer au préalable). Il faut **rajouter** `-larmadillo` dans les options de compilation : si vous utilisez un Makefile c'est à la suite de `-lm` dans la ligne de compilation (1.1). Si vous utilisez codeblocks, c'est à rajouter dans Menu/Settings/Compiler/LinkerSettings/OtherLinkerOptions).

Exemple :

```
#include <iostream>
using namespace std;
#include <armadillo>
using namespace arma;

main ()
{
 Col<int> V("1 2 3"); // vecteur colonne d'entiers
 cout<<"V="<<endl<<V<<endl;
 Mat<int> M("3 2 1; 4 5 0");// matrice d'entiers
 cout<<"M="<<endl<<M<<endl;
 Col<int> W = M * V; // calcule le produit
 cout<<"M*V="<<endl<<W<<endl;
 cout<<"element de matrice M(0,0)="<<M(0,0)<<endl;
}
```

Résultat :

```
V= 1
   2
   3

M= 3 2 1
   4 5 0

M*V= 10
     14

element de matrice M(0,0)=3
```

Remarque 3.3.1. Vous pourrez de la même façon utiliser d'autres classes selon vos besoins à condition cette fois ci d'inclure le fichier d'entête `#include <...>` approprié au début du programme et d'associer les librairies correspondantes à la compilation.

Exercice 3.3.2. Recopiez ce programme et exécuter le.

Exercice 3.3.3. "matrices" (*) Modifiez le programme précédent pour qu'il calcule et affiche :

```
    A = 1 1
        0 1

B = 1 0
    1 1

A*B = 2 1
      1 1
```

3.3.1 Quelques fonctions utiles sur les vecteurs et matrices

Veuillez consulter la page documentation pour toutes les fonctionnalités de armadillo sur les vecteurs et matrices.

Voici quelques exemples utiles.

```
#include <iostream>
using namespace std;
#include <armadillo>
using namespace arma;

main ()
{
//---- initialisations

//... vec est equivalent a Col<double>
vec V1 = randu<vec>(5); // vecteur colonne de 10 composantes aleatoires
 unif. dans [0,1]
cout<<"V1="<<endl<<V1<<endl;

//... mat est equivalent a Mat<double>
mat M1;
M1.zeros(3,3); // matrice 3*3 remplie de zeros
cout<<"M1="<<endl<<M1<<endl;
```

```
    //----- conversions de types
    vector<double> V2 = conv_to<vector<double>>::from(V1); // V1 ->
  V2
    cout<<"V2[0]="<<V2[0]<<endl;
    }
```

Résultat

```
V1=
0.7868
0.2505
0.7107
0.9467
0.0193

M1=
0 0 0
0 0 0
0 0 0

V2[0]=0.786821
```

3.4 Objets graphiques avec la librairie Root

Comme il est très utile et agréable de représenter des données et des résultats de façon graphique, nous présentons tout de suite l'utilisation d'une librairie graphique. La librairie root est développée au Cern (Laboratoire international de physique des particules). C'est une librairie en C++ gratuite et très performante, qui permet :

- de faire du **graphisme évolué** (dessiner des axes, des courbes, des surfaces, des objets en 3D,..), mais aussi du **graphisme simple** (lignes, points, ronds,..)
- de traiter des **données pour faire des statistiques** ; cela est très utile au CERN pour étudier les milliards de résultats issus d'une expérience de collisions entre particules.
- de faire des **interfaces graphiques** pour un programme (gestion de la souris, menus déroulants, boites de dialogues avec boutons, ...)
- et beaucoup d'autres choses. Voir la page web de presentation.
- Sur le réseau il y a une documentation complète. Il y a aussi Documentation générale. On peut aussi accéder à la liste des classes. Il y a une mailling list d'utilisateurs qui s'entraident. Quand on a un problème, il suffit d'envoyer un mail, la réponse nous revient quelques minutes ou heures plus tard. Le plus simple est souvent d'écrire dans le moteur de recherche de google des mots clefs comme "root cern ellipse" pour trouver comment dessiner une ellipse avec root.

— Si "root" n'est pas installé sur votre ordinateur, pour l'installer sous (X)Ubuntu, il faut écrire dans un terminal :

sudo apt-get install -y xorg-dev
`sudo apt-get install x-dev`
`sudo apt-get install root-system`

Remarque 3.4.1. Root peut s'utiliser aussi en mode interprété C++ : pour cela, lancer la commande **root** dans un terminal, et suivre les instructions...

3.4.1 Commande de compilation

— Si vous utilisez un Makefile, il faut rajouter à la ligne de compilation (1.1), à la suite de `-lm`, les instructions suivantes :

```
  -I/usr/include/root `root-config --cflags` `root-config --libs`
`root-config --glibs`
```

— Si vous utilisez l'éditeur codeblocks
  — Dans Settings/Compiler/Compiler_Settings/Other_Options, rajouter :

    `-I/usr/include/root`

  — Dans Settings/Compiler/Linker_Settings/Other_linker_Options, rajouter :

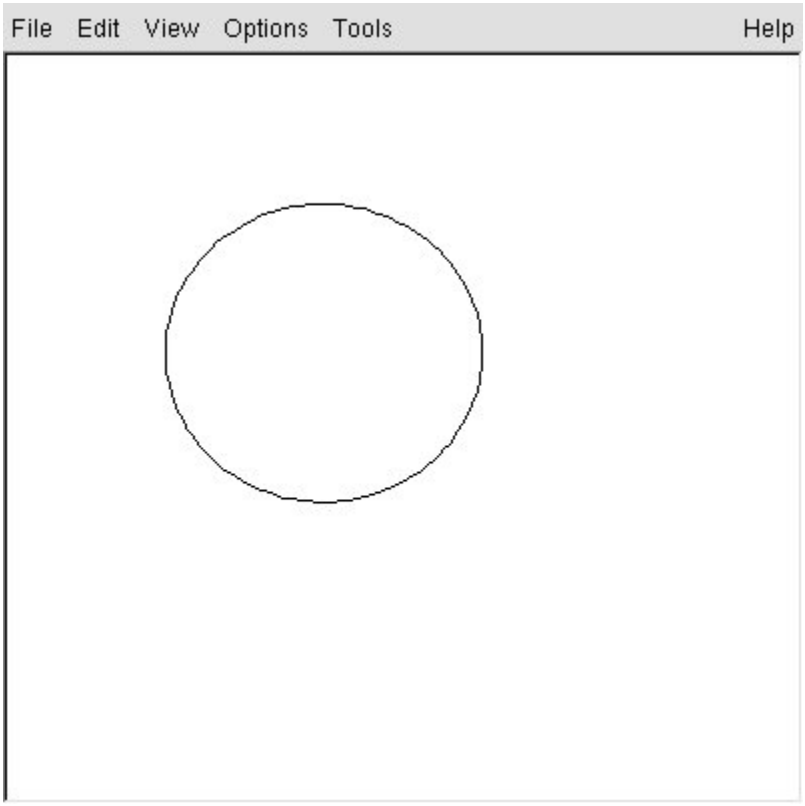
    `` `root-config --cflags` `root-config --libs` `root-config --glibs` ``

3.4.2 Exemple de départ qui dessine un cercle

```
#include <TApplication.h> // (A)
#include <TCanvas.h> //(B)
#include <TEllipse.h> // (C)

//------ fonction main (A) --------------------
int main()
{
 TApplication theApp("App", nullptr, nullptr); // (A)

 TCanvas c("c","fenetre",400,400); // (B) objet fenetre graphique.
Taille en pixels

 c.Range(0,0,5,5); // coordonnees de la fenetre c: x:0-5, y:0-5
(optionnel, par defaut ce sera 0-1)

//------ dessin d'une ellipse dans la fenetre c (C) -------
 TEllipse e(2,3,1); // on precise le centre (x=2,y=3) et le rayon=1
```

```
        e.Draw(); // dessine l'ellipse

        c.Update(); //(B) Montre le dessin

        theApp.Run(); // (A) garde la fenetre ouverte et permet a l'utilisateur
    d'interagir.
       }
```

]Résultat du programme :



Remarque 3.4.2. Le programme précédent est découpé en trois types de blocs :

(A) : dans tout programme utilisant Root, ces parties doivent toujours être réécrites. Il y a l'objet `theApp` de la classe `TApplication`.

(B) : c'est le code qu'il faut pour créer la fenetre graphique (objet c de la Classe TCanvas). Si votre programme fait du graphisme, il faut toujours créer une fenêtre graphique qui contiendra votre dessin.

(C) : c'est le code qu'il faut pour créer le cercle (objet e de la Classe TEllipse).

Exercice 3.4.3.

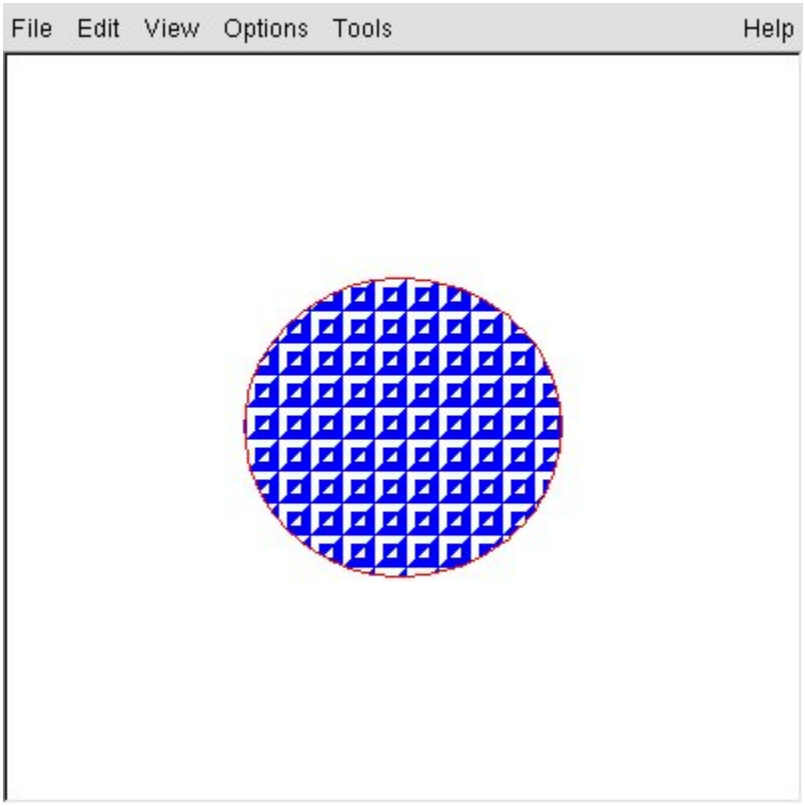
— Recopier le programme ci-dessus, le compiler et l'exécuter. Une fois que le dessin apparait, vous pouvez choisir dans le menu : **View/Event Status Bar** qui affiche la position de la souris.

— Modifier le programme afin d'afficher un cercle rouge au centre de la fenetre. Pour cela, rajouter la commande `e.SetLineColor(kRed);` à la bonne place. (voici la table des couleurs : table des couleurs).

Remarque 3.4.4. la commande `e.Draw()` appelle la **fonction membre** `Draw()` de la classe `TEllipse` qui dessine l'objet e. Pour connaitre toutes les opérations possibles que l'on peut effectuer sur l'ellipse, il faut regarder la liste des fonctions membres de cette classe TEllipse. On remarquera dans l'entête, que la classe TEllipse hérite d'autres classes comme la classe TAttLine qui concerne les propriétés des traits. Ainsi la classe TEllipse peut utiliser les fonctions de la classe TAttLine comme la fonction `SetLineColor()` qui permet de changer la couleur des traits.

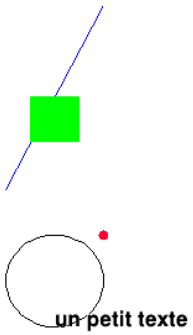
Exercice 3.4.5. Modifier le programme ci-dessus pour dessiner une ellipse remplie d'un motif bleu comme ci-dessous. Aide : d'après la documentation, TEllipse hérite de la classe TAttFill. On utilisera les fonctions membres :

```
e.SetFillColor(kBlue);
e.SetFillStyle(3025); // motif en carreaux
```



3.4.3 Quelques objets graphiques

Voici quelques exemples de graphisme 2D. Remarquez que au début du programme on ajoute une ligne spécifique par exemple `#include <TLine.h>` si on utilise la classe TLine.




```
#include <TApplication.h>
#include <TCanvas.h>
#include <TLine.h>
#include <TEllipse.h>
#include <TMarker.h>
#include <TBox.h>
#include <TText.h>

int main()
{
 TApplication theApp("App", nullptr, nullptr);

//.. La fenetre
 TCanvas c("titre","titre",10,10,400,400); //position x,y sur l'ecran
et taille X,Y en pixels
 c.Range(0,0,1,1); // xmin,ymin,xmax,ymax, systeme de coordonnees


//.. une ligne bleue
 TLine l(0.1,0.5,0.3,0.9); // x1,y1,x2,y2.
 l.SetLineColor(kBlue); // bleu
 l.Draw();

//.. Une Ellipse (ou cercle)
 TEllipse e(0.2,0.3,0.1); // (centre (x,y) et rayon r
e.Draw();
```

```
   //.. Un point
    TMarker m(0.3,0.4,8); // (x,y) et 6 ou 8: forme du Marker
    m.SetMarkerColor(kPink);
    m.Draw();

   //.. Un rectangle vert
    TBox p(0.15,0.6,0.25,0.7); // on precise les coins bas-gauche (x1,y1)
et haut droit (x2,y2) :
    p.SetFillColor(kGreen);
    p.Draw();

   //.. Du texte
    TText texte(0.2,0.2,"un petit texte"); // position x,y
    texte.Draw();

    c.Update(); // Montre le dessin
    theApp.Run(); // garde la fenetre ouverte et permet a l'utilisateur
d'interagir.
   }
```

3.5 Suppléments

3.5.1 Continuer une ligne de c++ à la ligne suivante avec \

```
#include <iostream>
using namespace std;

main ()
{
cout<<''abcdefghij''<<endl;

cout<<''abc\
def\
ghij''\
<<endl;
}
```

résultat :

```
abcdefghij
abcdefghij
```

3.5.2 Sur les types

3.5.2.1 Renommage

Dans la déclaration `vector<int> L;` on dit que l'objet `L` est du type `vector<int>`. Parfois un type peut être long à écrire. Pour simplifier les notations, voici deux commandes qui sont équivalentes et qui ont pour effet de définir le nouveau type `C` comme étant équivalent au type `vector<int>` (déjà existant). Reference.

```
#include <vector>
using C = vector<int>;
typedef vector<int> C;
```

3.5.2.2 Connaitre le type

pour connaitre le type de la variable `a` on peut utiliser typeid(a).name() et écrire

```
#include <iostream>
using namespace std;
#include <typeinfo>

main ()
{
auto a='A';
cout << typeid(a).name() <<endl;
auto x=3.14;
cout << typeid(x).name() <<endl;
auto s="ABC";
cout << typeid(s).name() <<endl;
}
```

résultat :

```
c
d
PKc
```

Dans ce résultat, c signifie caractère, d signifie double et PKc signifie chaine de caractères.

3.5.3 Quelques opérations élémentaires

```
#include <iostream>
using namespace std;

main ()
{
 int b=1;
 cout<<" b="<<b<<endl;
 b++; // equivalent a b=b+1
 cout<<" b="<<b<<endl;
 b-- ; // equivalent a b=b-1
 cout<<" b="<<b<<endl;
}
```

résultat

b=1
b=2
b=1

3.5.4 Sur les chaines de caractères

Référence : string.

3.5.4.1 Manipulations. Recherche dans une chaine, extraction ...

```
#include <iostream>
using namespace std;
#include <string>
int main()
{
string s1 = "ab&-e&-f";
string s2 = s1.substr(0, s1.find("&-")); // extrait sous chaine
entre le debut (indice 0) la premiere occurence de "&-"
string s3 = s1.substr(0, s1.rfind("&-")); // extrait sous chaine
entre le debut (indice 0) et la derniere occurence de "&-"
string s7 = s1.substr(s1.find("&-")); // extrait sous chaine entre
la premiere occurence de "&-" et la fin
string s4 = s1; // copie
s4.erase(0,2); // a la position 0, enleve 2 caracteres
string s5 = s1; // copie
```

```
s5.erase(s1.size()-2,2); // enleve les 2 caracteres de la fin
string s6 = s1; // copie
int p;
while(((p=s6.find("-"))>=0) && (p<=s6.size()))
s6.replace(p,1,"*"); // remplace tous les "-" par "*"
cout<<"s1="<<s1<<endl
<<"s2="<<s2<<endl
<<"s3="<<s3<<endl
<<"s7="<<s7<<endl
<<"s4="<<s4<<endl
<<"s5="<<s5<<endl
<<"s6="<<s6<<endl;
}
```

Résultat :

```
s1=ab&-e&-f
s2=ab
s3=ab&-e
s7=&-e&-f
s4=&-e&-f
s5=ab&-e&
s6=ab&*e&*f
```

3.5.4.2 Conversion de chiffres en chaine de caractère et inversement. Solution sophistiquée (permet des mélanges de type)

On a déjà un moyen élémentaire en Section 3.2.1.1.

On souhaite par exemple construire une chaine de caractère s qui contient le resultat d'une opération numerique. Pour cela il faut utiliser la classe `ostringstream`. Inversement pour extraire des chiffres d'une chaine de caractère il faut utiliser la classe `istringstream` .

```
#include <iostream>
using namespace std;
#include <string>
#include <sstream>

//.....
main ()
{
//..... Conversion chiffre -> string.
int c=7; // chiffre
```

```
ostringstream os;
os<<"resultat :"<<c; // on concatene du texte et le chiffre
string s = os.str(); // convertit os en string
cout<<"s= "<<s<<endl; // affichage pour verifier

//..... Conversion string -> chiffres.
string s2 = "3 4"; // chaine contenant des chiffres
istringstream is;
is.str(s2); // convertit chaine s2 en is
int a,b;
is>>a; // extrait le premier chiffre trouve dans is
is>>b; // extrait chiffre de is
cout<<"s2="<<s2<<endl;
cout<<"a="<<a<<" b="<<b<<endl; // affichage pour verifier
}
```

Résultat :

```
d=7 S=7
s= resultat :7
s2=3 4
a=3 b=4
```

3.5.4.3 Conversion nombre <-> chaine de caractere qui représente le nombre en base 2

Avec la classe bitset

```
#include <iostream>
using namespace std;
#include <string>
#include <bitset>
//.....
main ()
{

//... Conversion entier -> string (base 2)
int n4=117;
string s4 = bitset<8>(n4).to_string(); // nombre -> string (base
 2)
cout<<"n4="<<n4<<" s4="<<s4<<endl;

//.. Conversion string (nombre en base 2) -> entier
```

```
string s5="10011";
unsigned long n5 = bitset<8>(s5).to_ulong();
cout<<"s5="<<s5<<" n5="<<n5<<endl;
}
```

Resultat :

```
n4=117 s4=01110101
s5=10011 n5=19
```

3.5.4.4 Conversion de string vers char*

Il est parfois utile d'obtenir la chaine sous le format char*.
Le faire avec c_str().

3.5.5 Quelques opérations en base 2 (binaire, les bits) et en base 16 (hexadecimale)

Avec la classe bitset

```
#include <iostream>
using namespace std;
#include <bitset>
main ()
{
//... Declaration et Affichage en base deux:
int A=0b1001; // declaration en base 2 (prefixe 0b)
cout<<" A="<<A<<" en binaire = "<<bitset<8>(A)<<endl;
cout<<" Nombre de bits non nuls de A="<< bitset<8>(A).count()<<endl;

//.. decalages de bits
int b= A<<4; // equivalent: b= A * pow(2,4)
cout<<" b ="<<b<<" = "<< bitset<8>(b)<<endl;
int c = b>>3; // equivalent: c= b / pow(2,3)
cout<<" c ="<<c<<" = "<< bitset<8>(c)<<endl;

//.. operations bits a bits
int B = 0b1;
int d = A | B; // ou inclusif bit a bit cad 1|1 donne 1
int e = A ^ B; // ou exclusif bit a bit cad 1^1 donne 0
int f = A & B; // et bit a bit
int g = ~A; // non A, inversion des bits
```

```
cout<<" B="<<B<<" = "<<bitset<4>(B)<<endl;
cout<<" A|B = "<<d<<" = "<<bitset<4>(d)<<endl;
cout<<" A^B = "<<e<<" = "<<bitset<4>(e)<<endl;
cout<<" A&B = "<<f<<" = "<<bitset<4>(f)<<endl;
cout<<" ~A = "<<g<<" = "<<bitset<8>(g)<<endl;
}
```

Resultat :

```
A=9 en binaire = 00001001
Nombre de bits non nuls de A=2
b =144 = 10010000
c =18 = 00010010
B=1 = 0001
A|B = 9 = 1001
A^B = 8 = 1000
A&B = 1 = 0001
~A = -10 = 11110110
```

Remarque 3.5.1. De même on peut utiliser la base 16 (hexadécimale) avec le préfixe 0x. Par exemple :

```
a=0xFA;
cout<<a<<endl;
cout<<hex<<a<<endl; // pour afficher en base 16.
```

3.5.6 tuple : ensemble d'objets divers

Un objet de la classe tuple est un ensemble d'objets divers. Par exemple :

```
#include <iostream>
using namespace std;
#include <tuple>
main ()
{
 //... construction
 tuple<int,char> o1(10,'x'); // creation d'un objet constitué d'un
entier 10 et un caractère x
 auto o2 = make_tuple ("test", 3.1, 14, 'y'); // construction sans
preciser les types

 //... lecture des elements
 int a = get<0>(o1); // on extrait le premier element (position
0)
```

```
    cout<<"a="<<a<<endl;

    int b; char c;
    tie (b,c) = o1; // extraits les elements de o1
    cout<<"b="<<b<<" c="<<c<<endl;
    int d;
    tie (ignore, ignore, d, ignore) = o2; // extrait certains elements
de o2
    cout<<"d="<<d<<endl;

    //..... ecriture d'elements
    get<0>(o1) = 100; // on remplace l'element 0 de o1
    cout<<"o1[0]="<<get<0>(o1)<<endl;
    //.. concatenation de tuples
    auto o3 = tuple_cat(o1,o2);
    cout<<"o3[5]="<<get<5>(o3)<<endl;
   }
```

Résultat :

```
a=10
b=10 c=x
d=14
o1[0]=100
o3[5]=y
```

3.5.7 Vecteurs et matrices avec la librairie armadillo

3.5.7.1 Diagonalisation d'une matrice symétrique réelle

documentation. Exemple :

```
#include <iostream>
using namespace std;
#include <armadillo>
using namespace arma;
main ()
{
int N=3;
mat M = randu<mat>(N,N); // elements au hasard
M= 0.5*(M + trans(M)); // la rend symetrique
vec val_p;
mat vec_p;
```

```
    eig_sym( val_p, vec_p, M);
    cout<<"Matrice symetrique M="<<endl<<M<<endl;
    cout<<"valeurs propres="<<endl<<val_p<<endl;
    cout<<"vecteurs propres en colonnes="<<endl<<vec_p<<endl;
    cout<<"Verification M v0-l0 *v0 = 0 ? on trouve: "<<endl<<M*vec_p.col(0)
  - val_p(0) *vec_p.col(0) <<endl;
    }
```

Résultat :

```
Matrice symetrique M=
0.7868 0.5986 0.4810
0.5986 0.0193 0.2138
0.4810 0.2138 0.5206

valeurs propres=
-0.3109
0.2173
1.4203

vecteurs propres en colonnes=
0.5003 0.4080 0.7637
-0.8633 0.3037 0.4032
-0.0674 -0.8610 0.5041

Verification M v0-l0 *v0 = 0 ? on trouve:
-2.2204e-16
-2.2204e-16
-1.2143e-16
```

3.5.7.2 Diagonalisation d'une matrice hermitienne complexe

documentation. Exemple :

```
#include <iostream>
using namespace std;
#include <armadillo>
using namespace arma;
main ()
{
int N=3;
cx_mat M = randu<cx_mat>(N,N); // elements au hasard
M= 0.5*(M + trans(M)); // la rend symetrique
vec val_p;
```

```
    cx_mat vec_p;
    eig_sym( val_p, vec_p, M);
    cout<<"Matrice symetrique M="<<endl<<M<<endl;
    cout<<"valeurs propres="<<endl<<val_p<<endl;
    cout<<"vecteurs propres en colonnes="<<endl<<vec_p<<endl;
    cout<<"Verification M v0-l0 *v0 = 0 ? on trouve: "<<endl<<M*vec_p.col(0)
  - val_p(0) *vec_p.col(0) <<endl;
    }
```

Résultat :

```
Matrice symetrique M=
(+7.868e-01,+0.000e+00) (+4.810e-01,-4.620e-01) (+7.966e-02,+6.948e-02)
(+4.810e-01,+4.620e-01) (+5.206e-01,+0.000e+00) (+3.981e-01,+1.480e-01)
(+7.966e-02,-6.948e-02) (+3.981e-01,-1.480e-01) (+4.998e-01,+0.000e+00)

valeurs propres=
-0.1786
0.5407
1.4451

vecteurs propres en colonnes=
(+3.937e-01,-3.110e-01) (+4.834e-01,-2.058e-01) (+6.801e-01,-9.884e-02)
(-7.290e-01,-1.402e-01) (-1.696e-01,+6.247e-02) (+5.471e-01,+3.419e-01)
(+4.440e-01,+0.000e+00) (-8.315e-01,+0.000e+00) (+3.339e-01,-0.000e+00)

Verification M v0-l0 *v0 = 0 ? on trouve:
(-8.327e-17,+5.551e-17)
(-5.551e-17,+6.939e-18)
(-1.804e-16,+1.388e-17)
```

3.5.7.3 Diagonalisation d'une matrice complexe quelconque

documentation. Exemple :

```
    #include <iostream>
    using namespace std;
    #include <armadillo>
    using namespace arma;
    main ()
    {
    int N=3;
    cx_mat M = randu<cx_mat>(N,N); // elements au hasard
    cx_vec val_p;
```

```
   cx_mat vec_p;

   eig_gen( val_p, vec_p, M); // diagonalise

   //... ordonne les valeurs propres (et vect p) par module decroissant
   uvec indices = sort_index(abs(val_p),"descend"); // liste des indices,
"ascend" or "descend"
   val_p=val_p(indices); // vecteur ordonné
   vect_p=vect_p.cols(indices); // vecteur ordonné

   cout<<"Matrice complexe M="<<endl<<M<<endl;
   cout<<"valeurs propres="<<endl<<val_p<<endl;
   cout<<"vecteurs propres en colonnes="<<endl<<vec_p<<endl;
   cout<<"Verification: M v0-l0 *v0 = 0 ? on trouve: "<<endl<<M*vec_p.col(0)
- val_p(0) *vec_p.col(0) <<endl;
   }
```

Chapitre 4

Les instructions de base

4.1 La boucle "for"

L'instruction `for` permet de répéter un bloc d'instructions. Par exemple, la syntaxe de la ligne `for(int i=2; i<=8; i=i+2)` dans l'exemple ci-dessous signifie : **"Fait les instructions en partant de l'entier i=2, repête l'instruction i=i+2 et les instructions qui suivent dans le bloc `{...}` tant que i<=8 (inférieur ou égal)".**

```
#include <iostream>
using namespace std;

int main()
{
for(int i=2; i<=8; i=i+2)
    {
     int j=i*i;
     cout <<i << "\t "<<j <<endl;
    }
}
```

résultat :

```
2   4
4   16
6   36
8   64
```

Remarque 4.1.1. Remarquer que l'on a déclaré la variable `int i` dans la boucle for.

Exercice 4.1.2. Ecrire un programme qui affiche les valeurs 0.1 0.2 0.3 .. jusqu'à 10.0.

Remarque 4.1.3. Une autre possibilité d'utilisation de la boucle `for` en C++11 est donné avec l'exemple suivant. Dans ce cas la syntaxe est : `for(auto x:L)` et signifie **"pour chaque objet `x` dans la liste `L` fait l'instructions qui suit".**

```
#include <iostream>
using namespace std;
#include <vector>

int main()
{
  vector<int> L={5,7,3,12}; // une liste
  for(auto x:L) // la variable x parcourt la liste L
    cout<<x+1<<",";
}
```

résultat :

```
6,8,4,13,
```

Remarque 4.1.4. **"Bloc d'instruction"**. On a vu qu'un bloc d'instructions (= un ensemble d'instructions) est délimité par des accolades : "`{..}`". Par exemple dans une boucle "`for`", si vous avez une seule instruction, il n'est pas nécessaire de mettre des accolades. Si vous avez plusieurs instructions il faut mettre des accolades. Exemple :

```
#include <iostream>
using namespace std;
int main()
{
for(int i=2; i<=8; i=i+2)
   cout<<"i="<<i<<endl; // pas d'accolades

for(int i=2; i<=8; i=i+2)
  {   // accolade de debut de bloc
  int j=i*i;
  cout<<"i="<<i << "\t "<<"j="<<j <<endl;
  }  //accolade de fin

}
```

résultat :

```
i=2
i=4
i=6
i=8
```

```
i=2 j=4
i=4 j=16
i=6 j=36
i=8 j=64
```

Exercice 4.1.5. "Cercle qui tourne"(*) :

En faisant une boucle sur l'angle θ qui varie entre 0 et 2π par pas de 0.01 radian, faire un programme où l'on voit en animation un cercle de rayon $r = 1$, qui parcourt le cercle de rayon $R = 3$ dans le sens trigonométrique.

Aide : utiliser la librairie root, Section 3.4, et les formules de trigonométrie $x = R\cos\theta$, $y = R\sin\theta$. Pour utiliser les fonctions trigonométriques `cos` et `sin` il faut rajouter `#include <math.h>` en haut du programme. Le nombre π est connu en C++ en écrivant `M_PI`

Pour temporiser le programme, on pourra utiliser la Section 17.2.1 avec un délai de $10\,ms$ entre chaque dessin.

4.2 Ecrire une condition

On a utilisé ci-dessus la condition `i <= 30` qui signifie "i inférieur ou égal à 30". Voici la syntaxe pour écrire d'autres conditions :

4.2.1 Les opérateurs de comparaison :

Signification	symbole
supérieur à	`>`
inférieur à	`<`
supérieur ou égal à	`>=`
inférieur ou égal à	`<=`
égal à	`==`
différent de	`!=`

Attention à la confusion possible entre $==$ et $=$: `a == 2` sert à tester l'égalité de l'objet `a` avec 2 (cela ne change pas la valeur de a). Par contre `a = 2` met la valeur 2 dans l'objet `a`.

4.2.2 Les opérateurs logiques

Signification	symbole
et	&&
ou	||
non	!

Exercice 4.2.1. Que fait le programme suivant ? (deviner puis ensuite essayer pour vérifier)

```
#include <iostream>
using namespace std;

main ()
{
int a=1, b=2, c=3;
if( ((a <= b ) || (b >=c) ) && (a<c) )
  cout<<"Yes"<<endl;
else
  cout<<"No"<<endl;
}
```

4.2.3 Remarque

Lorsqu'une condition est évaluée comme `i<=30`, la valeur rendue est de la classe `boolean` (vrai ou faux).

4.3 La boucle "do {..} while (..) ;"

signifie **fait le bloc d'instructions {..} tant que la condition (..) est vraie.**

```
#include<iostream>
using namespace std;

main( )
{
int MAX(11);
int i(1),j;
do
 {
  j=i*i;
```

```
  cout<<i<<"\t "<<j<<endl;
  i=i+1; // Ne pas oublier l'incrémentation
 }
while(i<MAX); //condition
}
```

résultat :

```
1   1
2   4
etc...
10  100
```

4.4 La boucle "while (..) {..} ;"

signifie **tant que la condition (..) est vraie, fait le bloc d'instructions {..} .**

```
#include <iostream>
using namespace std;

main ( )
{
int max=11;
int i=1, j;
while(i<max) // condition
     {
      j=i*i;
      cout<<i<"\t"<<j<<endl;
      i=i+1; // ne pas oublier l'incrémentation
     }; // ne pas oublier le point virgule
 }
```

Résultat :

```
1   1
2   4
etc...
10  100
```

4.5 L'instruction "if"

```
#include<iostream>
```

```
using namespace std;

int main()
{
for(int i=1; i<=10; i=i+1)
 {
   if (i==4)
     cout<<"i= "<<i<<endl;
 }
}
```

Résultat `4`

Exemple plus général : On peut compléter avec "if.. else if.... else.." comme ceci :

```
#include<iostream>
using namespace std;
int main()
{
for(int i=1; i<=10; i=i+1)
{
if (i == 4)
 cout<<"i="<<i<<endl;
else if(i==5)
 cout<<"2*i= "<<2*i<<endl;
else
 cout<<"."<<endl;
}
}
```

Résultat :

.
.
.
i=4
2*i= 10
.
.
.
.
.

Exercice 4.5.1. "erreurs ?"(*) On souhaite afficher la suite 12345. Trouver l(es) erreur(s) et corriger le programme suivant :

```
#include<iostream>
using namespace std;

int main()
{
int i=1;
for(i=1; i<5; i=i+1);
     cout<<i;
}
```

4.6 *break :* pour sortir de la boucle. *continue* : pour sauter l'instruction.

```
#include <iostream>
using namespace std;
int main()
{
for(int i=2;i<=10;i=i+2)
    {
    if (i==4)
       continue; // on passe au suivant.
     if (i==8)
       break; // on sort de la boucle
     int j=i*i;
    cout<<i << "\t "<<j <<endl;
    }
}
```

Ce programme produira :

```
2   4
6   36
```

Exercice 4.6.1. "nombre au hasard"(*)

Faire un programme qui au départ choisit un nombre au hasard entre 0 et 1000 (Utiliser la Section 3.1.5), puis demande à l'utilisateur de le trouver, en répondant "trop grand " ou "trop petit " à chaque essai. L'utilisateur a le droit à 10 essais maximum.

Chapitre 5

Les fonctions

Une fonction est un petit sous programme qui effectue des instructions et utilise éventuellement certains paramètres donnés en entrée. Une fonction peut renvoyer un objet ou ne rien renvoyer du tout.

5.1 Exemple de fonction qui renvoit un objet

Dans l'exemple suivant, la fonction `carre` calcule `j=i*i`. Cette fonction prend en entrée un objet `int i` et renvoit `double j` en sortie. Pour cela on utilise la syntaxe `double carre(int i)`.

```
#include<iostream>
using namespace std;

//======declaration de la fonction carre ==================================

double carre(int i)
{
 double j=i*i;
 return j; //on renvoie le résultat au programme principal
}

//===declaration de la fonction principale =====
int main()
{
 int x;
 cout<<"entrer x "<<endl;
 cin>>x;
 double y=carre(x); // appel de la fonction carre
 cout<<"Le carré de "<<x<<" est "<<y<<endl;
```

```
}
```

résultat :

```
entrer x
2
le carré de 2 est 4
```

Remarque 5.1.1.

1. La fonction `carre` renvoit son résultat qui est un objet de la classe double. Pour cela on utilise l'instruction `return`. Pour appeler cette fonction, on utilise la syntaxe `y=carre(x)` si bien que le résultat renvoyé est tout de suite stocké dans l'objet `y`.
2. Dans le bloc `{..}` de la fonction `carre` on a déclaré l'objet `j`.Par conséquent, **cet objet `j` n'est connu que dans ce bloc et pas ailleurs.** On dit que c'est un objet **local**. On ne peut pas l'utiliser dans la fonction `main`. De même l'objet `x` déclaré dans le bloc de la fonction main n'est pas connu ailleurs : On ne peut pas l'utiliser dans la fonction `carre`.

5.2 Exemple de fonction qui ne renvoit rien

```
#include <iostream>
using namespace std;

//!========declaration de la fonction carre=============
void carre(int i)
{
 int j=i*i; // declaration d'un objet local
 cout<<"le carré de "<<i<<" est "<<j<<endl;
}

//!====declaration de la fonction principale ====================

int main()
{
 int x;
 cout<<" entrer x "<<endl;
 cin>>x;
 carre(x); // appel de la fonction carre
}
```

résultat :

```
entrer x
2
le carré de 2 est 4
```

Remarque 5.2.1.

1. La déclaration de la fonction `carre` montre que cette fonction prend un paramètre (l'objet i de la classe int) et ne renvoit rien (à cause du préfixe `void` qui signifie "vide"). La fonction `main` appelle la fonction `carre` et lui passe un parametre qui est l'objet x choisi par l'utilisateur.

5.3 Paramètres des fonctions par référence

Quand le programme principal fournit un objet à une fonction, on peut vouloir que celle-ci modifie le contenu de l'objet. Dans l'exemple précédent la procédure ou la fonction `carre` ne modifie pas la valeur de x. Dans l'exemple ci-dessous, on veut échanger le contenu de deux objets a et b. On doit alors dire à la fonction qu'elle a le droit de changer les objets qu'on lui donne en entrée. Pour cela, dans la déclaration des paramètres, on met le signe & devant les paramètres pouvant être modifié par la fonction. On dit que ce sont des **paramètres passés par référence**.

```
#include<iostream>
using namespace std;

//====== fonction permute ==================
void permute(int& i,int& j) // passage des parametres par reference
{
    int t=i; // stockage temporaire
    i=j;
    j=t;
}
//======== fonction main ===================
int main()
{
    int a=10, b=5;
    cout<<a<<" "<<b<<endl;
    permute(a,b);
    cout<<a<<" "<<b<<endl;
}
```

résultat :

```
10 5
5 10
```

Exercice 5.3.1. Executer ce programme, puis enlever les signes & (de **référence**) dans le passage des paramètres, et ré-essayer. Conclusion ?

5.4 La surcharge des fonctions.

Un aspect interessant du C++ est que l'on peut "surcharger " les fonctions : c'est à dire que l'on peut donner le même nom à des fonctions qui font des choses différentes. Ce sont les paramètres demandés lors de l'appel de la fonction qui permet à l'ordinateur de distinguer qu'elle est la fonction à appeler.

5.4.1 Exemple :

```
#include<iostream>
using namespace std;

//-----------------------------------------
void f(int i)
{
 cout<<"fonction 1 appelée"<<endl;
 cout<<" paramètre = "<<i<<endl;
}

//-----------------------------------------
void f(char *s,int i)
{
 cout<<"fonction 2 appelée"<<endl;
 cout<<" paramètre = "<< s <<endl;
 cout<<" paramètre = "<< i <<endl;
}

//-----------------------------------------
int main()
{
 f(10);
 f("Chaîne ",4);
}
```

Renvoit

```
fonction 1 appelée
  paramètre = 10
fonction 2 appelée
  paramètre = Chaîne
  paramètre = 4
```

Exercice 5.4.1. "factorielle" (*)

Ecrire une fonction que l'on appelera `factorielle` qui en entrée prend un objet `x` de la classe `long`, et en sortie renvoit sa factorielle x ! de la classe `long`. Dans la fonction principale on utilisera cette fonction avec le code suivant :

```
long a=5;
long b = factorielle(a);
cout<<" a="<<a<<" a!="<<b<<endl;
```

afin qu'il affiche comme résultat :

```
a= 5 a!=120
```

Remarque 5.4.2. Ici on programme la fonction factorielle ; c'est un exercice. Mais cette fonction factorielle est bien sûr déjà programmée comme d'autres fonctions spéciales que l'on trouve dans la librairie "boost".

Chapitre 6

Les pointeurs

6.1 Déclaration et affectation d'un pointeur

La notion de pointeur est importante dans le langage C et C++. Elle est réputée comme étant difficile et technique ; nous espérons que vous aurez néanmoins les idées claires après la lecture de cette section.

Nous introduisons rapidement la notion de pointeur, et montrons comme exemple, son intérêt pour créer des **tableaux de taille variable** au cours du programme.

Rappellons déjà ce qu'est un objet. Par exemple :

```
int i;
```

Cette instruction a pour effet de réserver un "objet" en mémoire de l'ordinateur, permettant de stocker un nombre entier. Cet objet s'appelle 'i' son type (ou sa classe) est `int`.

Bien sûr, cet objet se trouve quelque part dans la mémoire de l'ordinateur. Il a un certain emplacement, caractérisé par son "adresse mémoire", appellée son **pointeur**.

Il faut donc retenir que **pointeur d'un objet** signifie **adresse d'un objet** dans la mémoire de l'ordinateur.

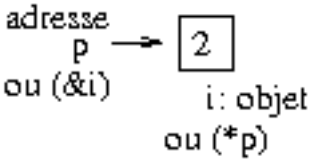
On peut avoir accès à l'adresse de l'objet `i` en faisant `&i` :

```
int i=2; // déclare l'objet i et affecte la valeur 2
int *p; // déclaration du pointeur p
p=&i; // p devient l'adresse de i
```

Grâce au signe *, la deuxième ligne déclare p comme étant un pointeur sur entier (c'est à dire une adresse d'un objet contenant un entier).

- **Le signe `&i` signifie l'adresse de l'objet `i`.**
- **(*p) signifie l'objet situé à l'adresse du pointeur p.**
- Si `fonction()` est une fonction appartenant à une classe et si `p` est un pointeur d'un objet de cette classe, alors
  l'instruction `p->fonction();` **est équivalente à** `(*p).fonction();`

Schéma qui résume ce que l'on vient d'expliquer :



Conséquences : pour afficher le contenu de l'objet i on a maintenant deux possibilités qui sont équivalentes :

```
cout<<i;
```

ou :

```
cout<<(*p);
```

Pour modifier le contenu de l'objet i on a maintenant deux possibilités qui sont équivalentes :

```
i=5;
```

ou :

```
(*p)=5;
```

Remarque 6.1.1. Attention : avant d'effectuer l'opération d'écriture : `(*p)=5` ; il faut être sur que l'adresse du pointeur correspond à un objet existant. Vous avez donc compris que avant d'écrire, il faut prendre le soin de réserver de la place mémoire. On parle d'allocation de la mémoire.

On comprends l'écriture `int *p;` comme la déclaration que `*p` est un objet de type int, donc `p` est un pointeur sur un objet de type int.

6.1.1 Paramètres de la fonction `main()`

Jusqu'à présent, nous avons écrit la fonction principale du programme par la déclaration `int main() {...}`

En fait cette fonction peut prendre des paramètres et renvoyer un entier. Cela peut être utile pour passer des paramètres à un programme et obtenir des paramètres en retour. Voici un exemple que l'on commente ensuite :

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
cout<<"nombres de parametres argc ="<<argc<<endl;
cout<<" Ce sont:"<<endl;

for(int i=0;i<argc; i++)
```

```
{
  cout<<"argv["<<i<<"] = "<<argv[i]<<endl;
}
return 0; // renvoit l'entier 0 (signifie habituellement OK)
}
```

Exécution

```
./test 2 toto 5
```

Résultat :

```
nombres de parametres argc =4
Ce sont:
argv[0] = ./test
argv[1] = 2
argv[2] = toto
argv[3] = 5
```

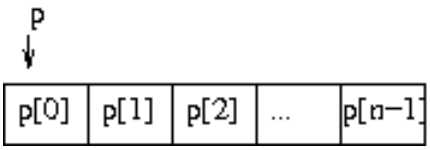
Commentaires On a lancé le programme par la commande `./test 2 toto 5` qui contient en effet 4 paramètres, le premier étant le nom du programme lui même. Cet exemple montre que au début du programme main(..) , la variable argc contient le nombre de paramètres et `argv` est un tableau de chaines de caractères pour chacun de ces paramètres.

6.2 Allocation dynamique de la mémoire

On peut réserver en mémoire de l'ordinateur une suite de n objets de type `double` par l'instruction :

```
p=new double[n];
```

Après cela, **p pointe sur le premier objet réservé.**



Remarque 6.2.1. De façon similaire `p=new string[13];` réserve en mémoire une suite de 13 objets de type `string`

Cela s'appelle une **allocation dynamique** de la mémoire, car elle se fait au cours de l'éxécution du programme.

Il y a n objets, numérotés de 0 à n-1.

on peut écrire dans la case du premier objet par l'instruction :

```
(*p)=1;
```

ou (ce qui est équivalent)

```
p[0]=1;
```

On peut de même écrire dans la case suivante par :

```
*(p+1)=2;
```

ou

```
p[1]=2;
```

etc... jusqu'à `p[n-1]`.

A la fin de l'utilisation, n'oubliez pas de **libérer l'emplacement mémoire** par l'instruction :

```
delete [ ] p; // on libere les cases mémoire
```

Exemple :

```
#include<iostream>
using namespace std;
int main()
{
int n;
cout <<"entrez nbre d'objets n >=0 ?"<< flush;
cin >> n;
double *p; //on déclare le pointeur p
p=new double[n]; // on réserve n cases mémoire de la classe double

//--- remplissage des cases memoires
for(int i=0; i<n; i++)
p[i]=2*i;

//---- affichage
for(int i=0; i<n; i++)
cout<<" p["<<i<<"]="<<p[i]<<endl;

delete [ ] p; // on libere les cases mémoire
}
```

Résultat : entrez nbre d'objets n >=0 ?3
p[0]=0
p[1]=2
p[2]=4

6.2.0.1 Remarques

1. On peut réserver un nombre fixe de cases en mémoire par l'instruction :

   ```
   double p[6]; // creation de 6 cases mémoires de type double, numérotées de 0 à 5.
   p[1] = 3.14; // on écrit dans la case 1.
   ```

   Il s'agit d'une allocation de mémoire dite statique (non dynamique) car le nombre de case est fixé à la compilation, et ne peut être variable. On peut aussi déclarer et initialiser en même temps :
   double p[6] = {1, 3.14, 3., 0.} ;

2. Si l'on veut ne réserver qu'une seule case mémoire au lieu d'un tableau, il suffit de faire :

   ```
   int *p;
   p=new int; // on réserve une case mémoire de classe int.
   (*p)=1;
   delete p; // pour libérer la place mémoire
   ```

3. Un tableau de caractères est aussi appelé une **chaîne de caractères**. Pour ce cas il y a une initialisation spéciale :

   ```
   char *chaine = ‘‘toto’’;
   ```

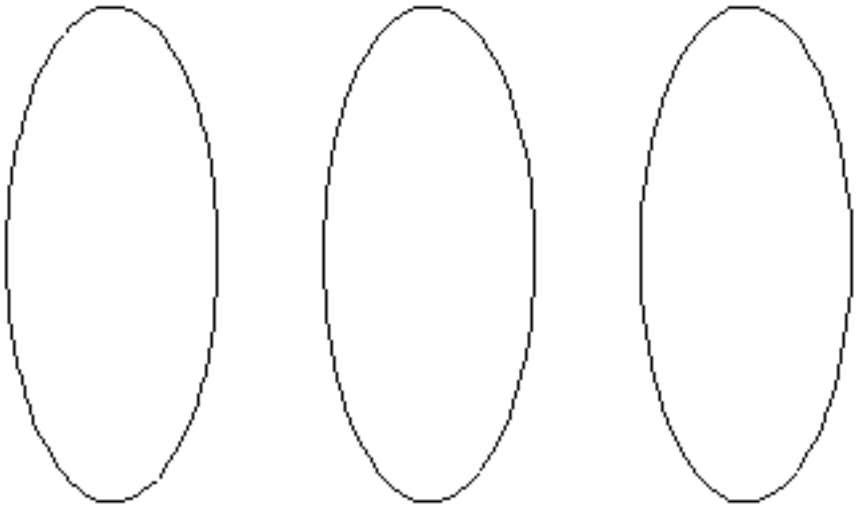
   qui déclare `chaine` comme étant un pointeur sur caractère, pointant sur les 4 caractères toto.

Exercice 6.2.2. “pointeurs” (*) Ecrire un petit programme qui demande à l'utilisateur un entier n compris entre 1 et 5, puis alloue de façon dynamique un tableau p de taille n de la classe string. Ensuite dans une bloucle on demande à l'utilisateur le contenu de chaque case du tableau. A la fin, on affiche le contenu du tableau. L'execution du programme doit donner cela (par exemple) :

```
entrez n? 3
entrez p[0] ? un
entrez p[1] ? petit
entrez p[2] ? programme
p[0] =un
p[1] =petit
p[2] =programme
```

6.3 Utilisation des pointeurs avec la librairie graphique ROOT

Dans l'exercice 4.1.5 vous avez dessiné une ellipse qui se déplace à l'aide d'une boucle "for". Comment dessiner plusieurs cercles simultanéments ? Nous vous proposons deux possibilités. La première utilise une liste (classe vector), la deuxième possibilité utilise des pointeurs.



6.3.1 Utiliser une liste d'ellipses

Ecrire le programme :

```
#include <TApplication.h>
#include <TEllipse.h>
#include <TCanvas.h>
#include <vector>
```

```
using namespace std;

int main()
{
 TApplication theApp("App", nullptr, nullptr);

 TCanvas c( "c","fenetre",400,400); // on precise la taille en pixels

c.Range(-2,-2,8,2); // coordonnees de la fenetre

//-------------
vector<TEllipse> tab_e(3); // liste de 3 ellipses

for (int i=0; i<3; i++)
    {
     tab_e[i].SetX1(3*i); // coord x du centre
     tab_e[i].SetY1(0); //coord y du centre
     tab_e[i].SetR1(1); // rayon selon x
     tab_e[i].SetR2(1); // rayon selon y
      tab_e[i].SetFillColor(kWhite);
      tab_e[i].Draw(); // dessin
    }
  //-------------
  c.Update();
 theApp.Run();
}
```

Ainsi l'ellipse numéro i+1 n'efface pas l'ellipse numéro i.

6.3.1.1 Remarques :

- Rappel : dans l'instruction "for", l'instruction `i++` est équivalente à `i = i+1`. De même `i--` est équivalente à `i = i-1`
- Pour enlever l'ellipse numéro 1 de la liste, il faut utiliser la fonction `erase()` de la classe `vector` :

  ```
  tab_e.erase(tab_e.begin()+1);
  ```

  Rajouter cette ligne au bon endroit, dans le programme précédent et observer le résultat.

6.3.2 Utiliser un pointeur sur l'ellipse

Ecrire le programme :

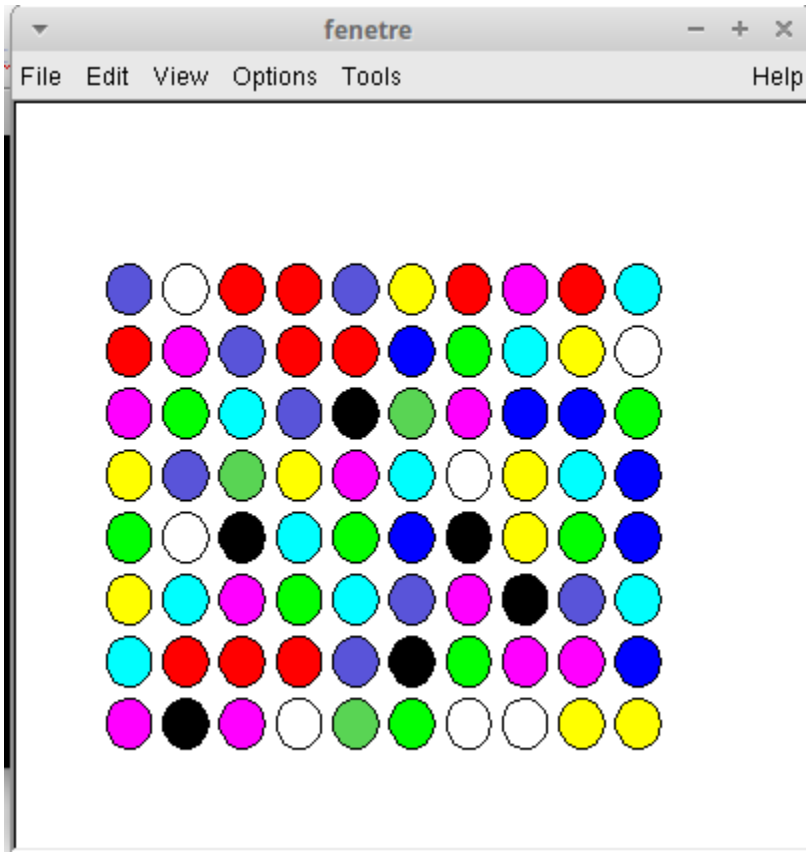
```
#include <TApplication.h>
#include <TEllipse.h>
#include <TCanvas.h>
#include <vector>
using namespace std;

int main()
{
 TApplication theApp("App", nullptr, nullptr);

 TCanvas c( "c","fenetre",400,400); // on precise la taille en pixels

c.Range(-2,-2,8,2); // coordonnees de la fenetre

//-------------
TEllipse *pe; // crée un pointeur sur TEllipse
for (int i=0;i<3;i++)
{
    pe= new TEllipse(3*i,0,1); // pe pointe sur une ellipse
 crée en position x=3*i, y=0, rayon 1
    pe->Draw(); // dessin
}
     //-------------
  c.Update();
 theApp.Run();
}
```

6.3.2.1 Remarques :

Cette dernière méthode est simple, et voici ce qu'il se passe : `pe` est un pointeur sur un objet de la classe `TEllipse` autrement dit l'adresse de l'objet. L'instruction `pe= new TEllipse()` crée un objet ellipse et affecte `pe` à l'adresse de cet objet. La deuxième fois, cette même instruction crée un nouvel objet, sans détruire l'ancien, et `pe` devient l'adresse de ce nouvel objet. A la fin on a la situation suivante où les trois ellipses existent (numéros 0,1,2), et `pe` est l'adresse du dernier objet :

L'avantage de cette dernière méthode est que l'on peut créer un nombre indéfini d'ellipses. Pour modifier une ellipse déjà existante il faudrait stocker au fur et à mesure les adresses `pe` dans un tableau (ou utiliser une fonction de ROOT qui permet de récupérer les adresses des objets existants).

Exercice 6.3.1. "Tableau coloré" (*) Ecrire un programme qui dessine un tableau de 10*8 cercles colorés au hasard comme cela :



Chapitre 7

Création d'une classe

Programmer **une classe** consiste à définir un nouveau type d'objets avec des opérations précises que l'on pourra faire avec. Vous avez utilisé par exemple la classe des **Complexes** déjà existante (écrite par d'autres informaticiens avant vous). Avec cette classe, on peut déclarer des variables complexes et effectuer directement des opérations sur les nombres complexes comme z3=z1*z2*exp(z4),...sans devoir décomposer l'opération sur les parties réelles et imaginaires.

Dans cette section, on va apprendre à écrire soi-même une classe, et l'on prendra l'exemple des vecteurs et des matrices. Il existe déjà des classes de vecteurs et de matrices prêtes à l'emploi et très performantes, comme *armadillo*, voir Section 3.3. Considérez donc cette Section comme étant à but pédagogique.

7.1 Exemple (étape1)

Un vecteur de $\mathbb{R}^3$ est définit par ses trois composantes (x, y, z). Voici un programme de départ pour définir un vecteur en C++. Essayez ce code. Lisez ensuite les commentaires.

Code :

```
#include <iostream>
using namespace std;
#include <math.h>
// ===============déclaration de la classe ===============
class Vect
{
public:
 double x,y,z;
};
//====== Programme principal ======================
```

```
int main()
{
 Vect v; // declaration
 v.x=1; v.y=2; v.z=3;
 cout << "composante v.y = " << v.y << endl;
}
```

Résultat :

```
composante v.y = 2
```

7.1.1 Commentaires

— Dans la première ligne du programme principal on a déclaré un objet `v` de la classe `Vect`. Dans la partie déclaration, il est écrit qu'un objet de la classe Vect possède trois variables `x,y,z` de type double appelées variables membres. A la deuxième ligne du programme principal, le code `v.x=1;` met la valeur 1 dans la variable membre x de l'objet `v`. De façon générale `objet.variable` est la façon de désigner une variable associée à un objet.

— L'annonce `public:` en haut des déclarations permet à ce qui suit d'être connu et accessible hors de la déclaration de la classe. Au contraire on rend les déclarations non accessibles (si besoin est) par l'annonce `private:`

— Remarquez dans le programme la présence ou non de points virgule ; en particulier à la fin de la déclaration de la classe.

Exercice 7.1.1. En modifiant le programme précédent, créer une classe appelée `Musicien` qui contient les variables membres : `string nom; string instrument; int age; string style;` . Dans le programme principal on crée un objet de cette classe : `Musicien robert_J;` on initialise les variables et on affiche la variable instrument.

7.2 Exemple (étape2)

On complète le programme précédent. Essayez ce code. Lisez ensuite les commentaires.

Code :

```
#include <iostream>
using namespace std;
#include <math.h>

// ===============déclaration de la classe ===============
class Vect
{
```

```
public:
//---- variables membres
 double x,y,z;
//-- le constructeur---
 Vect()
 {
 x=0; y=0; z=0;
 cout << "vecteur construit... " << endl;
 }
//-- le destructeur---
 ~Vect()
 {
 cout << "vecteur détruit.... " << endl;
 }
//--la fonction Norme---
 double Norme()
 {
 double n=0;
 n=x*x+y*y+z*z;
 return sqrt(n);
 }
};

//====== Programme principal ======================
int main()
{
 Vect v; // declaration
 v.x=1; v.y=2; v.z=3;
 cout << "Norme = " << v.Norme() << endl;
}
```

Résultat :

```
vecteur construit...
Norme = 3.74166
vecteur détruit....
```

7.2.1 Commentaires

— Commençons par la première ligne du programme principal : on a déclaré un objet `v` de la classe `Vect`. Cela a pour effet d'appeler la fonction `Vect()` déclarée dans la classe, appelée **constructeur**. Dans la partie déclaration, il est écrit qu'un objet de la classe Vect possède trois variables `x,y,z` de type double appelées variables

membres. Dans notre exemple la fonction constructeur a pour effet de mettre les variables x,y,z à zéro et d'afficher le texte "vecteur construit...".

— Ainsi à la deuxième ligne du programme principal, le code `v.x=1;` met la valeur 1 dans la variable membre ***x*** de l'objet `v`. De façon générale `objet.variable` est la façon de désigner une variable associée à un objet.

— A la troisième ligne du programme principal, le code `v.Norme()` appèle la fonction `Norme()` qui est déclarée dans la classe. Cela a pour effet de calculer et afficher $\sqrt{x^2+y^2+z^2} = \sqrt{1+4+9} = 3.74..$ De façon générale `objet.fonction()` est la façon d'appeler une fonction membre associée à un objet.

— a la fin du programme principal, l'objet v va être détruit et le programme appelle la fonction ~Vect() appelée destructeur, même si cela n'est pas explicitement écrit. Dans notre exemple cela a pour effet d'afficher le texte "vecteur détruit.... ".

— Dans la déclaration d'une fonction membre comme `Norme()` les variables membre `x,y,z` s'accèdent directement (sans la syntaxe v.x) car ce sont sans ambiguité les variables membre de l'objet en cours pour qui la fonction a été appelée, ici `v`.

— La fonction membre **constructeur** `Vect()` est appelée chaque fois qu'un objet Vect est déclaré au cours du programme. Comme son nom l'indique, le constructeur sert à initialiser le nouvel objet créé. Le constructeur *doit toujours porter le nom de la classe* en question (ici Vect). Cette fonction particulière ne renvoit rien, pourtant on n'écrit pas `void Vect()`.

— la fonction membre **destructeur** ~`Vect()` est appelée chaque fois qu'un objet Vect est détruit au cours du programme. Le destructeur *doit toujours porter le nom de la classe* en question (ici `Vect`) avec `~` devant . Cette fonction particulière ne renvoit rien, pourtant on n'écrit pas `void ~Vect()`.

Exercice 7.2.1. "Classes" (*)

1. Ecrire le programme ci dessus dans un fichier **vecteur.cc** (dans un nouveau répertoire /classes/), éxécutez le, et vérifiez le comportement attendu.
2. Rajouter une fonction membre de la classe `Vect` que l'on appelera `Affiche()`, et qui affiche à l'écran le contenu du vecteur sous la forme :
   `Vect: x=1 | y=2 | z=3`
   Appelez cette fonction dans le programme principal et testez le bon fonctionnement.
3. Rajouter une fonction membre de la classe `Vect` que l'on appelera `double Produit(Vect v2)`, et qui permet de calculer le produit scalaire entre deux vecteurs. Le résultat est un `double`. Dans le programme principal, l'appel doit se faire sous la forme :
   ```
   Vect v,w;
   double d=v.Produit(w);
   ```
   Aide : dans la fonction `double Produit(Vect v2),` on écrira `return x*v2.x+y*v2.y+z*v2.z;`
4. Appelez cette fonction dans le programme principal et testez le bon fonctionnement. Remarque : essayez de comprendre la cause des messages construit et détruit. Il manque un construit. La raison de cela sera expliquée ultérieurement au paragraphe Constructeurs par recopie, et affectations.

5. Rajoutez un constructeur qui permet d'initialiser directement les variables membres `x,y,z` du vecteur. Il aura la syntaxe `Vect(double xi,double yi,double zi).` Dans le programme principal, on pourra alors déclarer un objet par `Vect v(1,2,3);` Remarque : vous gardez le constructeur précédent `Vect()` qui ne prend pas d'argument. N'oubliez pas en effet qu'en C++ des fonctions différentes peuvent avoir le même nom et ne différent que par leurs arguments.

7.3 Utilisation de pointeurs sur objets

Nous avons expliqué les pointeurs dans la section 6. Dans l'exemple 7.2 ci-dessus, remplacer les quelques lignes du programme principal par

```
//====== Programme principal ======================
int main()
{
 Vect *w = new Vect(); // declaration
 w->x=0; w->y=3; w->z=4;
 cout << "Norme de w = " << w->Norme() << endl;
}
```

Résultat :

```
vecteur construit...
Norme de w = 5
```

Commentaires

— La déclaration Vect *w signifie que w est un pointeur (c'est à dire une adresse) sur un objet de la classe Vect. le code w = new Vect() ; a pour effet de créer un objet à cette adresse.

— A la deuxième ligne le code `w->x=0;` met la valeur 0 dans la variable membre x de l'objet pointé par w. De façon générale `pointeur->variable` est la façon de désigner une variable associée à un objet qui est pointé par le pointeur.

— A la troisième ligne du programme principal, le code `w->Norme()` appèle la fonction `Norme()` qui est déclarée dans la classe. De façon générale `pointeur->fonction()` est la façon d'appeler une fonction membre associée à un objet pointé par le pointeur `pointeur`.

Remarque 7.3.1.

— Il n'y a pas d'appel automatique de la fonction destructeur à la fin du programme. En fait seule la variable pointeur est détruite. Si l'on souhaite appeler le destructeur de l'objet pointé il faut le faire explicitement par `w->~Vect();` ou `delete(w);` qui a le même effet.

Chapitre 8

Les fichiers

Jusqu'à présent, nous avons lu des données au clavier et nous les avons affiché à l'écran. On peut aussi écrire des données dans un fichier (sur le disque dur de l'ordinateur ou sur un disque dur ou clef USB externe). Pour cela il faut utiliser des fonctions qui manipulent les fichiers. Ces fonctions sont regroupées dans le fichier `fstream` et font partie des classes `ofstream` pour écrire dans un fichier ou `ifstream` pour lire dans un fichier.

Traduction : Output (=sortie) File (=fichier) Stream (= flux de données), ou Input(=entrée),etc..

Reference :

8.1 Ecriture de données dans un fichier

reference : ofstream.

Par exemple pour écrire un chiffre dans un nouveau fichier que l'on appellera `hector.txt`, il suffit de faire 3 étapes :

```
#include <iostream>
using namespace std;
#include <fstream> // utilisation des fichiers

int main()
{
 ofstream f("hector.txt"); // ouvre le fichier hector.txt pour y
ecrire. On lui associe l'objet: f
 f<<3.1514<<endl; // permet d'ecrire dans le fichier.
 f.close(); // fermeture du fichier f
}
```

8.1.0.1 Remarque :

1. On a choisit de terminer le nom du fichier `hector.txt` par le suffixe `.txt`. Ce n'est pas une obligation, mais une convention : `txt` signifie texte, et précise que ce fichier

peut se lire comme un texte (même si il y a des chiffres). De la même façon le fichier qui contient votre programme se termine par `.cc` pour signifier que c'est le texte d'un programme C++.

2. Pour écrire dans le fichier, on a utilisé un objet intermédiaire de la classe `ofstream` (abréviation de output-file-stream). Cet objet a été initialisé avec le nom du fichier, et pour écrire dans le fichier on utilise **la même syntaxe que pour écrire à l'écran** (avec `f` à la place de `cout`).
3. L'ouverture du fichier se fait par l'initialisation de l'objet f avec le nom du fichier donné sous la forme d'une chaine de caractères. On peut donc passer un objet intermédiaire (chaine de caractères) de la façon suivante :
`string nom=”hector.txt”`
`ofstream f(nom);`
4. Lorsque l'on a effectué l'opération `f.close()`, f est un objet de la classe `ofstream` et `close()` est une fonction de cette classe. On parle de **fonction membre**. Remarquez la syntaxe : l'objet et la fonction membre sont reliés par un point.
5. Attention, la commande `ofstream f("hector.txt");` ouvre le fichier mais efface les données existantes si il y en avait. Pour rajouter des données à un fichier existant (ou non existant) il faut l'ouvrir avec l'instruction `ofstream f("hector.txt",ios::app);` (“app” signifie “append”, “ajouter”)

8.1.0.2 Exercice

Ecrire le programme précédent et vérifier l'existence du fichier. Ouvrir le fichier dans codeblocks ou avec un autre éditeur et vérifier son contenu.

8.2 Lecture de données depuis un fichier

reference : ifstream.
Le programme suivant permet de lire le chiffre écrit précédement dans le fichier `hector.txt`.

```
#include<iostream>
using namespace std;
#include<fstream> // utilisation des fichiers

int main()
{

    ifstream g("hector.txt");// ouvre un nouveau fichier en
 lecture. On lui associe l'objet: g
    double x;
    g>>x; //on lit ce qu'il y a au début du fichier, et on
 le copie dans l'objet x
    cout<<x<<endl; // on écrit à l'écran le contenu de x
```

```
        g.close(); // fermeture du fichier g
}
```

8.2.0.1 Remarques :

1. Si le début du fichier n'avait pas contenu de chiffre (mais des lettres ou rien du tout) le programme n'aurait rien mit dans l'objet `x`.
2. On peut de la même façon écrire ou lire plusieurs données à la suite dans un fichier : il est important de savoir que le programme est au départ positionné au début du fichier, puis après avoir lu (ou écrit) une donnée, il se trouve positionné juste après cette donnée, prêt à lire la donnée suivante.
3. La lecture `g>>x` depuis le fichier est similaire à la syntaxe de lecture `cin>>x` depuis le clavier.
4. A la place de `cout<<x<<endl;` on peut écrire les instructions :

   ```
   if(g.good())
   cout<<x<<endl;
   ```

   En effet `g.good()` renvoit true si la lecture précédente s'est bien faite.

Exercice 8.2.1. (*) Reprendre et modifier le programme de l'exercice 4.6.1 pour que les numéros proposés par l'utilisateur soient ecrit dans un fichier.

8.3 Suppléments sur les fichiers

Voici quelques exemples qui montrent comment lire des lignes, des caractères dans un fichier.

```
#include <iostream>
using namespace std;
#include <string>
#include <fstream>
int main()
{
//... ecrit des lignes de texte dans un fichier
ofstream f("test.txt");
f<<"ligne1: abc"<<endl;
f<<"ligne2: def"<<endl;
f.close();
//... ouvre le fichier en lecture
ifstream g("test.txt");
string l;
while(g.good())
```

```
   {
   getline(g, l);// -> l. copie la ligne dans l (sans le code retour
a la ligne)
   cout<<l<<endl; // affiche ligne l
   }
   g.clear(); // remet les indicateurs à zero pour continuer la recherche.

   //....
   g.seekg(0, g.beg); // se place au debut du fichier (begining + 0)
   int p;
   p=g.tellg(); // -> position p dans le fichier
   cout<<"p="<<p<<endl;
   char c;
   for(int i=0; i<=5; i++)
   {
   g.get(c); // -> c. Lit caractere
   cout<<c;
   }
   cout<<endl;
   p=g.tellg();
   cout<<"p="<<p<<endl;
   }
```

résultat

```
ligne1: abc
ligne2: def
p=0
ligne1
p=6
```

Remarque 8.3.1. `g.seekg(0, g.beg);` est équivalent à `g.seekg(g.beg);` Comme autre option il y a `g.seekg(0, g.end);` (position fin de fichier + 0) ou `g.seekg(0, g.cur);` (position actuelle + 0)

Chapitre 9

Micro-projets 1

En guide de conclusion de cette première partie, nous vous proposons de réaliser un micro-projet parmi la liste suivante. Ce sont des programmes rapides à réaliser (quelques lignes de code), afin de vous montrer l'aspect ludique et créatif de la programmation, et aussi pour résumer différentes notions vues dans ce didacticiel.

Choisissez et réalisez un de ces micro-projet, selon votre motivation et l'aisance que vous ressentez en informatique.

Pour le graphisme, utiliser la Section 16.

Nous demandons de produire un compte rendu sous forme pdf et html, que l'on peut rédiger par exemple avec lyx. Voir Section 14.6.

La Section suivante donne quelques recommandations générales et prélables pour effectuer un projet de programmation.

9.1 Que faire avant de programmer ?

Avant de se retrouver devant un ordinateur pour écrire un programme, il faut avoir établi clairement ce qu'on désire lui faire faire. Voici l'ensemble des étapes à suivre, **indispensables**, à faire au préalable devant une feuille de papier.

1. Définir clairement **l'objectif du programme** (surtout s'il y a plusieurs personnes à y collaborer).
2. Définir l'ensemble des tâches qui doivent être accomplies dans un **ordre logique** pour atteindre cet objectif. Pour un projet scientifique il faut que toutes les formules mathématiques soient bien écrites.
3. Tracer sur un papier un **organigramme** illustrant le fonctionnement de votre programme. Vérifiez que l'ordre d'exécution des diverses tâches va effectivement faire ce que vous attendez.

   En fait, votre organigramme se traduira directement dans la partie principale de votre programme (fonction int main()). A chaque tâche correspondra l'appel d'une sous partie (**une fonction**).

4. Déterminez la méthode (**algorithme**) permettant de remplir chaque tâche. Il est possible qu'une sous-tâche soit commune à plusieurs fonctions. Dans ce cas, faites-en une nouvelle fonction. Cependant, pour éviter une multiplication de ces fonctions, ne le faites que pour celles faisant plusieurs lignes de programme. L'art de la programmation consiste à trouver un compromis entre la simplicité, la longueur, et la rapidité d'éxécution de votre programme.
5. Si le programme effectue des taches complexes et "imprévisibles", imaginer des situations simples où l'on connait la solution et qui permettront de tester le programme.

9.1.1 Quelques notions de qualité en informatique

La qualité d'un programme ne se traduit pas simplement en termes **d'efficacité** mais également en termes de **sécurité (fiabilité) et d'exportabilité**. Autrement dit, vous devez écrire votre programme de telle sorte (1) que n'importe qui puisse le lire et comprendre ce qu'il fait et (2) qu'il puisse être aisément testé. Voici quelques indications :

— **modulaire :** chaque fonction devrait être indépendante en vue d'être éventuellement utilisée dans un autre programme. Dans ce but n'hésitez pas à créer une nouvelle classe dès que le besoin se fait sentir.

— **utilisable par autrui :** chaque fonction (et classe) devrait avoir en entête des commentaires indiquant,

1. son objectif,
2. l'algorithme ou méthode utilisée,
3. la liste des objets d'entrée,
4. la liste des objets de sortie,
5. (éventuellement) des instructions sur la manière de l'appeler et un exemple d'utilisation.

Cet ensemble d'informations correspond à une sorte de cahier des charges d'une fonction. Lors d'un travail en équipe, il appartient à chacun de travailler sur des fonctions différentes. Il faut donc se mettre d'accord au préalable sur la manière dont chaque fonction va dialoguer avec les autres.

— **lisible :** tant que possible, utiliser des noms évocateurs pour les objets (mais pas trop longs) et les fonctions.

— **fiable :** éviter au maximum l'utilisation d'objets globaux, sauf si ils permettent un véritable gain en lisibilité (passage d'arguments moins fréquents). Dans ce cas, indiquer dans un commentaire le (ou les) objet(s) global(aux) qu'une fonction attend.

9.2 Suite de Syracuse

Difficulté : facile.

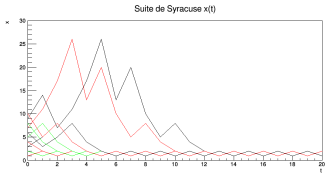
Soit un nombre entier $x_0 \geq 1$. Par récurrence, on définit x_{t+1} à partir de x_t par :

$$x_{t+1} = \begin{cases} \frac{1}{2}x_t & \text{si } x_t \text{ est pair} \\ \frac{1}{2}(3x_t + 1) & \text{si } x_t \text{ est impair} \end{cases}$$

Observer que la valeur de départ $x_0 = 1$ donne la suite infinie périodique $1, 2, 1, 2, 1, \ldots$

La valeur de départ $x_0 = 7$ donne la suite $7, 11, 17, 26, 13, 20, 10, 5, 16, 8, 4, 2, 1, 2, 1, 2, 1, \ldots$

La fameuse **conjecture de Syracuse** non démontrée à ce jour est que partant de toute valeur x_0 la suite arrive forcément au bout d'un moment T (qui dépend de x_0) à la suite périodique $1, 2, 1, 2, \ldots$



Exercice 9.2.1. Ecrire un programme qui demande x_0 à l'utilisateur et affiche la suite $(x_t)_t$ jusqu'à ce que la valeur $x_t = 1$ soit atteinte.

Solution 9.2.2. Faire :

1. Ecrire une fonction $S(x)$ qui prend un entier x en entrée et renvoit $\frac{1}{2}x$ si x est pair ou $\frac{1}{2}(3x + 1)$ sinon.
2. Dans le programme principal, on demande x à l'utilisateur. Tant que $x \neq 1$, faire
   (a) $x = S(x)$
   (b) Affiche x

Exercice 9.2.3. Pour x_0 donné, on note $T(x_0)$ la plus petite valeur de $t \in \mathbb{N}$ telle que $x_t = 1$. Tracer T en fonction de $x_0 \in \{1, 2, \ldots 1000\}$.

Solution 9.2.4. Voici l'algorithme. Faire une fonction $T = L(x)$ qui prend un entier x en entrée et renvoit T :

1. On pose $T = 1$.
2. Tant que $x \neq 1$ faire :
   (a) on calcule $x = S(x)$.
   (b) $T = T + 1$

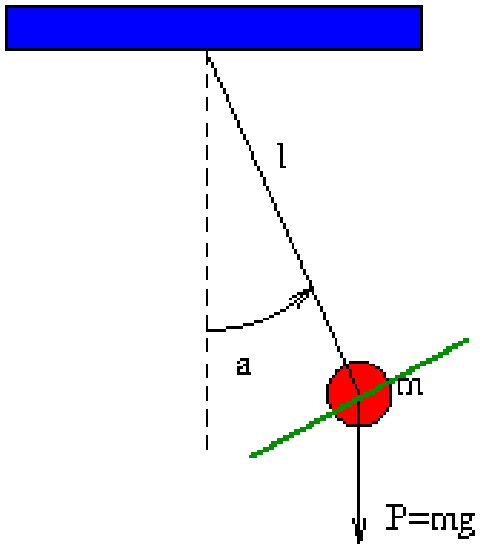
Ensuite,

1. Créer une fenetre graphique, avec des axes $x_0 = 1 \to 1000$ et $T = 0 \to 100$,
2. on parcourt $x_0 = 1 \to 1000$,
   (a) on calcule $T = L(x_0)$
   (b) on dessine un point en (x_0, T)

9.3 Pendule simple

Difficulté : moyenne.

Dessiner (en animation) dans l'espace réel et dans l'espace des phases, le mouvement d'oscillation d'un pendule simple de masse m, longeur l, fixées, soumis à la force de pesanteur $P = mg$ avec $g = 9.8m/s^2$ et à une force de frottement de coef γ.



Equations physiques : si $a(t)$ est la position angulaire et $b(t) = \frac{da}{dt}$ la vitesse angulaire au temps t on a

$$\frac{da}{dt} = F_a(a,b) := b$$
$$\frac{db}{dt} = F_b(a,b) := -\frac{g}{l}\sin(a) - \frac{\gamma}{m}b$$

où $F_a(a,b), F_b(a,b)$ sont des fonctions de a, b. (vérifier ces formules en appliquant la loi de Newton sur la droite verte tangente au cercle.).

Position du pendule :

$$x = l\sin(a), \quad y = -l\cos(a)$$

Méthode de résolution numérique Avec la **méthode de Euler**, on considère un petit intervalle de temps dt ; au premier ordre en dt on a :

$$a(t+dt) = a(t) + dt \cdot F_a$$
$$b(t+dt) = b(t) + dt \cdot F_b$$

En effet ces formules découlent de $\frac{da}{dt} = F_a \Leftrightarrow \frac{a(t+dt)-a(t)}{dt} = F_a \Leftrightarrow a(t+dt) = a(t) + dt\, F_a$, et de même pour b.

Voici comment on utilise ces formules. On choisit un pas de temps dt très petit. Par exemple $dt = 0.01$. On discrétise le temps avec un pas de temps dt : la date de départ est t_0. ensuite il y a $t_1 = t_0 + dt$, $t_2 = t_1 + dt$, etc..

On suppose que $a_0 = a(t_0), b_0 = b(t_0)$ sont les conditions initiales connues. Avec les formules de Euler on calcule $a(t_1), b(t_1)$, puis $a(t_2), b(t_2)\ldots$

On fait le dessin du pendule au fur et à mesure du calcule.

Structure du programme Ecrire une fonction :

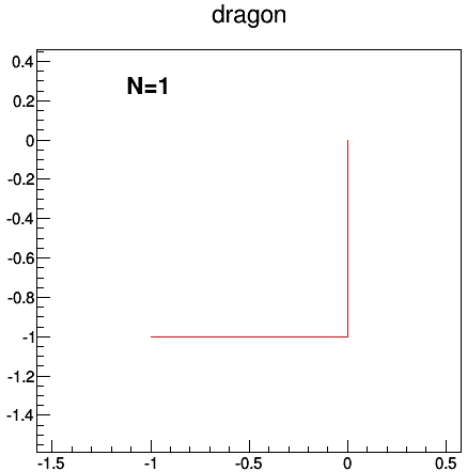
```
/* Formule du champ de vecteur sur le plan (a,b)
entree: a,b
sortie: Fa,Fb
*/
void F(double a, double b, double & Fa,double & Fb)
{
//... code ici
}
```

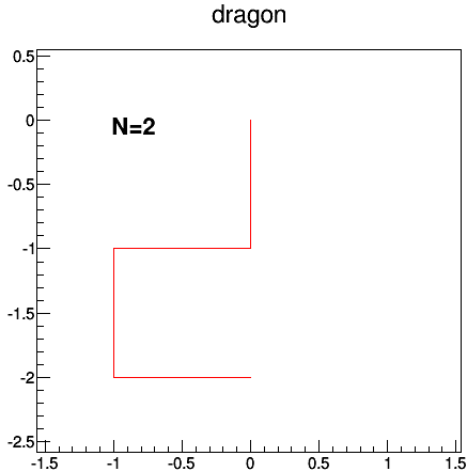
Dans la fonction main, partir de valeurs a, b initiales, calculer a, b à chaque pas de temps et dessiner le pendule.

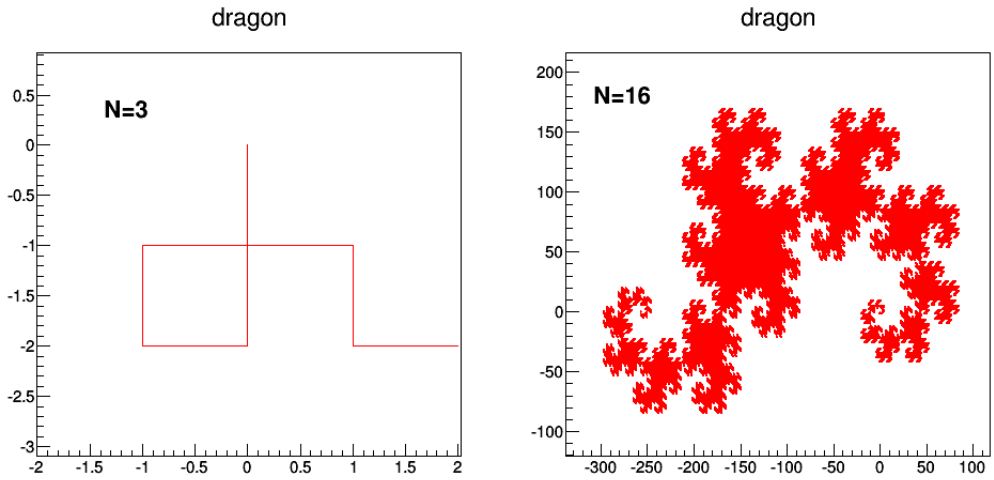
9.4 La fractale du dragon

Difficulté : moyenne.

Si on prend une feuille de papier que l'on plie $N = 1$ fois et que l'on impose au pli de faire un angle droit (90°) on obtient la forme suivante. Si on plie $N = 2$ fois ou $N = 3$ fois ou ...$N = 16$ fois on obtient les formes suivantes assez surprenantes. La forme pour $N \gg 1$ s'appelle la fractale du dragon.







Ecrire un programme qui demande une valeur $N \geq 1$ et dessine la forme obtenue si l'on plie N fois. Qu'observez vous de surprenant ?

Exercice 9.4.1. On peut coder la forme à N fixé, comme une suite S_N de $2^N - 1$ valeurs ± 1, où $+1$ (respect. -1) signifie que le pli est à droite (respect. à gauche). Par exemple $S_1 = \{1\}$, $S_2 = \{-1, 1, 1\}$, $S_3 = \{-1, -1, 1, 1, -1, 1, 1\}$. Montrer que la suite S_{N+1} se déduit facilement de la suite S_N.

Solution 9.4.2. Si la suite $S_N(i)$ est connue alors

$$S_N = \overline{S_{N-1}} \cup \{1\} \cup S_{N-1}$$

où $-\overline{S_{N-1}}$ est la suite S_{N-1} écrite à l'envers avec l'opposé des éléments.

Exercice 9.4.3. En programation, on peut utiliser la classe `vector<int> S;` qui est une liste d'entiers, avec les instructions `S.push_front(j);` pour rajouter la valeur j au début, et `S[k];` qui donne la valeur à la position k. Ecrire l'algorithme de cette suite.

Solution 9.4.4. $N \geq 1$ est donné. On part de la liste $S = \{1\}$. On pose $n = 1$. Tant que $n < N$ faire :

1. On copie $S' = S$. On pose $d = S.size()$.
2. On rajoute 1 au début de la liste S'.
3. Pour $i = 0 \to d - 1$, on rajoute $-S[i]$ au début de la liste S'.
4. On copie $S = S'$. On fait $n = n + 1$.

Exercice 9.4.5. Ensuite, pour la partie graphique, il suffit de lire cette liste et dessiner des segments à la suite. Ecrire l'algorithme.

Solution 9.4.6. On code une direction par un chiffre : dir=0,1,2,3 pour respectivement : droite, haut, gauche, bas.

1. On initialise $(x_1, y_1) = (0, 0)$ et dir $= 0$ et $i = 0$.

2. On calcule (x_2, y_2) à partir de (x_1, y_1) de la façon suivante :
   (a) Si dir $= 0$ alors $(x_2, y_2) = (x_1 + 1, y_1)$
   (b) Si dir $= 1$ alors $(x_2, y_2) = (x_1, y_1 + 1)$
   (c) etc
3. On change de direction : $dir = (\text{dir} + S_N(i) + 4)\,\%4$. (Remarque : on a écrit $+4$ car il y a un "bug" sur le modulo en langage C/C++.)
4. On dessine une droite de (x_1, y_1) à (x_2, y_2).
5. On pose $(x_1, y_1) = (x_2, y_2)$ et $i = i + 1$.
6. Si $i < S_N.size()$ on retourne en (2).

9.5 Images subliminales

Dans "le code de la conscience" de Stanilas Dehaene, p.66, il est expliqué qu'une image de durée <0.33 sec et entourée d'autres images ne sera pas percue conscienment. Effectuer ce type d'expériences avec des formes géométriques.

9.6 Equation de la chaleur

Difficulté : moyenne.

Notons $x = (x_1, \ldots x_n)$ les variables d'espace et t la variable de temps. Joseph Fourier a découvert en 1822 que la propagation de la température $T(x,t)$ dans un matériau est régit par l'équation d'évolution

$$\frac{\partial T(x,t)}{\partial t} = (\Delta T)(x,t)$$

appelée équation de la chaleur, avec l'opérateur Laplacien :

$$\Delta T := \frac{\partial^2 T}{\partial x_1^2} + \ldots + \frac{\partial^2 T}{\partial x_n^2}$$

On va modéliser cette évolution en dimension $n = 2$. On discrétise l'espace avec un pas $h \ll 1$ très petit. On note donc $x_1 = ih$, $x_2 = jh$ avec $i, j = 0 \ldots N-1$. Le champ de températures à l'instant t est modélisé par une matrice $T(i,j)$ de taille $N \times N$. On discrétise le temps avec un pas $\epsilon \ll 1$ très petit et on note T' le champ de température à l'instant $t + \epsilon$.

On remplaçant les dérivées par des différences finies montrer que l'équation de la chaleur ci-dessus devient :

$$T'(i,j) = T(i,j) + \frac{\epsilon}{h^2}\left(T(i+1,j) + T(i-1,j) + T(i,j+1) + T(i,j-1) - 4T(i,j)\right).$$

Partant d'un champ de températures quelconque, en utilisant cette modélisation, écrire un programme qui fait évoluer et dessine le champ de température. Discuter le choix des paramètres ϵ, h. Pour les matrices, on utilisera la classe `mat` de `armadillo`.

9.7 Le jeu puissance 4

niveau facile (difficile si option de stratégie).

— Faire un programme qui permet de jouer à deux personnes au jeu de puissance 4, et affiche le résultat sur le terminal (avec cout). Stocker l'état du jeu dans une matrice d'entiers.

— En option, on peut jouer contre l'ordinateur. Développer une stratégie par recherche arborescente.

9.8 Le jeu de la vie

Difficulté : moyenne.

En utilisant une matrice d'entiers de taille 10*10 avec des conditions périodiques, faire évoluer et dessiner les coefficients de matrice selon les règles simples du jeu de la vie.

9.9 Algorithme de Monte-Carlo pour le modèle d'Ising

Difficulté : moyenne à difficile.

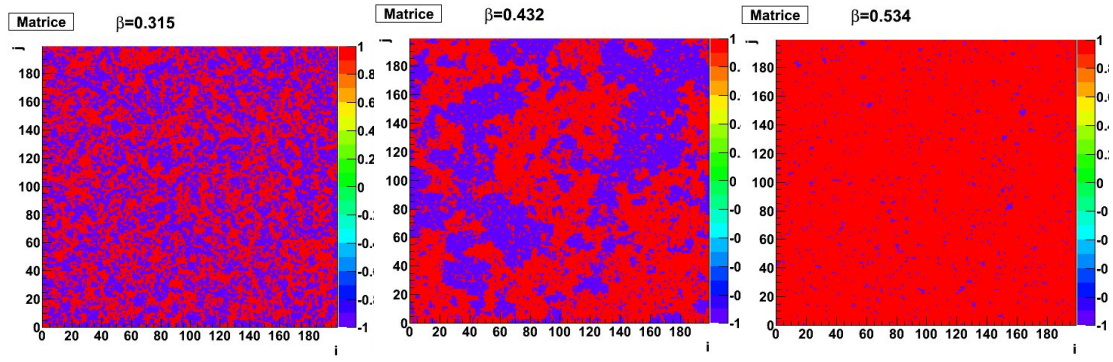


FIGURE 9.1 – Résultat de l'algorithme de Monte-Carlo pour le modèle d'aimantation de Ising, aux températures $\beta = 1/(k_b T) = 0.3$ (désordre), 0.4 (transition de phase) et 0.5 (ordre magnétique).

(Explication physique : voir Cours de Master 1: système dynamiques, chapitre "Dynamique probabiliste sur les graphes").

On considère un réseau $N \times N$ périodique dont les sites sont notés $X = (x, y)$ et dont les variables sont $f_X = \pm 1$ et modélisent des spins (up/down). Si deux sites sont voisins

(chaque site a 4 voisins) on note $X \sim Y$. Une configuration des spins est un champ donné : $f = (f_X)_X$. Son énergie ferromagnétique est :

$$E(f) := \sum_X \sum_{Y, X \sim Y} (-f_X . f_Y)$$

Remarque 9.9.1. Il y a N^2 sites et 2^{N^2} configurations possibles.

Les configurations d'énergie minimale sont celles donnant $(-f_X.f_Y)$ minimal soit $f_X.f_Y = 1$ pour tous X, Y. Il y a donc deux configurations d'énergie minimale qui sont : f_{up}=$\{f_X = 1$ pour tous site $X\}$ et f_{down}=$\{f_X = -1$ pour tous site $X\}$.

Les configurations d'énergie minimale sont celles donnant $(-f_X.f_Y)$ maximal soit $f_X.f_Y = -1$ pour tous X, Y. Cela est possible si Nest pair avec des spins dont les valeurs sont alternées (il y a deux configurations).

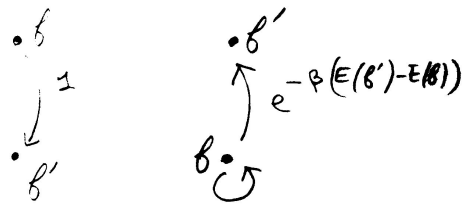
9.9.1 Algorithme de Monte-Carlo

L'algorithme d'évolution suivant est appelé **algorithme de montecarlo**. $\beta \geq 0$ est un paramètre fixé qui est l'inverse de la température : $\beta = 1/(k_b T)$.

1. A l'instant $n = 0$, on part d'une configuration f choisie au hasard.
2. On choisit un site X_0 au hasard, et on note f' la configuration identique à f sauf au site X_0 où le spin est opposé : $f'_{X_0} = -f_{X_0}$ (spin opposé).
3. Choix de la configuration à l'instant $n + 1$,
   (a) Si $E(f') < E(f)$ on choisit la configuration f'.
   (b) Si $E(f') \geq E(f)$ on choisit la configuration f' avec la probabilité $P_{f \to f'} = \exp(-\beta.(E(f') - E(f)))$ (et on choisit f avec la probabilité complémentaire).
4. On revient en (2) pour poursuivre l'évolution du champ de spins.

Remarque 9.9.2. Cet algorithme correspond à une matrice stochastique réversible pour la mesure de Boltzmann :

$$u(f) = \frac{1}{Z} \exp(-\beta E(f))$$



Exercice 9.9.3.

1. Montrer que la condition $E(f') < E(f)$ s'écrit simplement en fonction du site X et de ses voisins.

2. Ecrire un programme qui fait évoluer et représente la configuration du champ de spins f_X selon l'algorithme de Monte-Carlo. (Utiliser une matrice d'entiers avec armadillo et le dessin de TH2D expliqué en Section 16.3.1 avec l'option "`col`").
3. L'aimantation de la configuration f est $M(f) := \sum_X f_X$. Représenter l'aimantation au cours du temps. Quand l'équilibre statistique est atteind, calculer la moyenne $\langle M \rangle$ et la variance $V = \left\langle (M - \langle M \rangle)^2 \right\rangle$. Représenter $\langle M \rangle (\beta)$ et $V(\beta)$ en fonction de β.

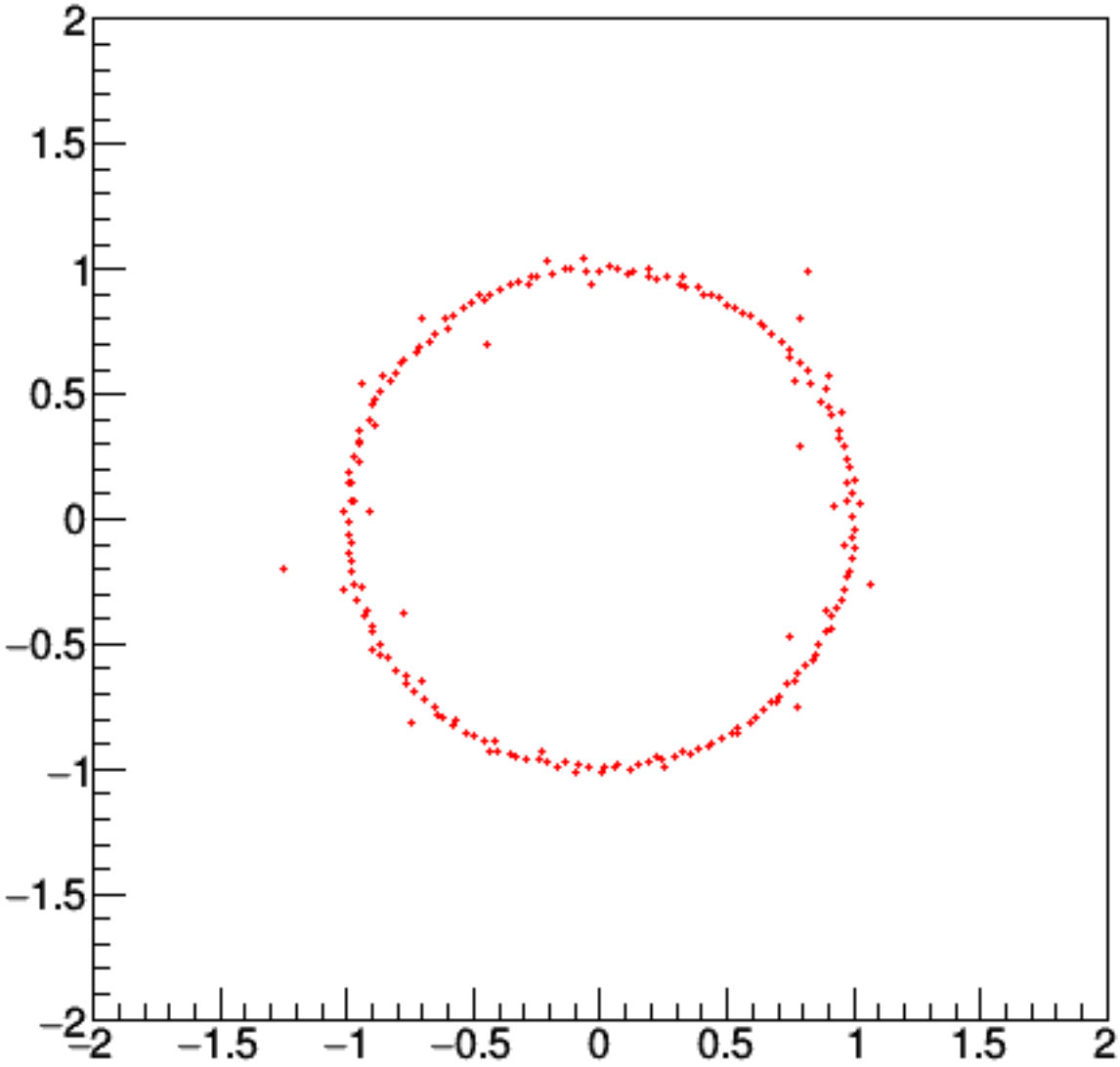
9.10 Zeros d'un polynome aléatoire

On considère un polynome $P(x)$ de degré $N \geq 1$ et à coefficients P_k aléatoires et indépendants, selon une loi uniforme $P_k \in [-1, 1]$:

$$P(x) = \sum_{k=0}^{N} P_k x^k$$

On sait que P a N zéros $(z_i)_{i=0 \to N}$ dans le plan complexe. Dessiner les N zéros du polynome et observez. Quelle conjecture avez vous ? Consulter cet article de Terence Tao et al.. On peut aussi considérer des coefficients complexes aléatoires : $P_k = R_k + iI_k$ avec $R_k, I_k \in [-1, 1]$ aléatoires et indépendants.

Aide : les zéros du polynome sont les valeurs propres de la "matrice compagnon" du polynome. Construire cette matrice et utiliser la fonction eigen() de armadillo pour trouver les valeurs propres.



sls sls d

Deuxième partie

Compléments sur le langage C++

Chapitre 10

Autres environnements

10.1 Programmer avec l'environnement *"codeblocks"*

Au lieu d'utiliser emacs comme ci-dessus vous pouvez utiliser un environnement de programmation plus convivial ou évolué comme codeblocks. Le désavantage est qu'il nécessite un certain apprentissage, qu'il effectue automatique certaines taches à votre place et vous perdez un peu le contrôle de votre programmation.

Remarque 10.1.1. Il existe aussi d'autres environnements de développement sophistiqués comme eclipse ou emacs.

Exercice 10.1.2.

1. Lancer le programme codeblocks.
2. Choisir "create a new project"
3. Faire les choix : projet de type "Console application", C++ ; lui donner un nom, comme "projet1" et choisir le répertoire où il sera hébergé.
   Dans le répertoire "Sources", il apparaît un fichier appelé **main.cpp** contenant le code C++ suivant :

```
#include <iostream>
using namespace std;
int int main()
{

   cout << "Hello world!" << endl;
    return 0;
}
```

4. Une fois pour toute on va choisir la version 11 du C++. Pour cela, dans le menu Settings/Compiler/Compiler Flags, cocher l'option de compilation **-std=c++11**
5. La commande du menu Plugins/SourceCodeFormatter vous permet de mettre votre programme en forme standard. Faites le souvent.

6. Pour **compiler et exécuter** ce programme, taper F9 (ou *build and run* dans le menu). Il apparait alors une fenêtre (console) avec le message "***Hello world !”.***

— Dans la suite, vous pouvez modifier le code de ce programme pour l'adapter aux exemples donnés ci-dessous. Vous pouvez aussi créer d'autres projets de la même manière.
— Pour **copier un exemple**, vous pouvez "copier” le code donné en exemple dans ce tutoriel et le "coller” dans votre éditeur, sans le retaper.

Chapitre 11

Ecrire et lire des données en binaires sur un fichier (ou un flux)

11.1 Bits, octets, bytes, taille d'un objet

Or dans la mémoire de l'ordinateur une objet numérique (ex : `int i=17`) est stockée en **binaire**, i.e. en base 2, sous forme d'une suite de 0 et 1 appelés des **bits**. Une séquence de 8 bits est appellée **un octet**.

Le byte est un ensemble de bits qui définit l'unité de stockage. En général un byte contient 8 bits, mais cela peut varier selon l'ordinateur. La variable système CHAR_BIT contient le nombre de bits contenus dans un byte. En général CHAR_BIT=8.

La commande `sizeof(type)` indique le nombre byte occupé par un type donné. Le programme :

```
#include<iostream>
using namespace std;
#include <limits.h>
int main()
{
cout<<"sizeof(int)="<<sizeof(int)<<endl;
cout<<"CHAR_BIT="<<CHAR_BIT<<endl;
}
```

affiche :

```
sizeof(int)=4
CHAR_BIT=8
```

Par exemple l'objet `int i=17` est stocké en mémoire en base 2 par une suite de 4*8=32 bits soit 4 octets :

```
00010001 00000000 00000000 00000000
```

Voici le taille des principales types de base :

Classe	char	short int	int	long	float	double
Taille (en byte)	1	2	4	4	4	8

Exemple :

```
#include<iostream>
using namespace std;
#include <bitset>
int main()
{
int i=17;
cout<<"La variable int i="<<i<<" est de taille "<<sizeof(i)<<" bytes."<<endl;
unsigned char *p=(unsigned char*)&i; // Le pointeur p est l'adresse
memoire de i
cout<<"Contenu des bytes, en ecriture base 10:"<<endl;
for(int i=0; i<sizeof(i);i++)
  cout<<" "<<(int)p[i];
cout<<endl<<"Contenu des bytes, en ecriture base 2:"<<endl;
for(int i=0; i<sizeof(i);i++)
  cout<<" "<<bitset<8>(p[i]);
cout<<endl;
}
```

Affiche :

```
La variable int i=17 est de taille 4 bytes.
Contenu des bytes, en ecriture base 10:
17 0 0 0
Contenu des bytes, en ecriture base 2:
00010001 00000000 00000000 00000000
```

11.2 Fichiers en format texte

Dans les leçons précédentes, nous avons vu comment écrire des données numériques dans un fichier. Ces données étaient écrites sous forme de chiffres (ex : 17) et sont lisibles depuis un éditeur. On dit qu'il s'agit d'une écriture en **format texte** ou format ASCII ; car un objet numérique est représenté en base dix par un ou plusieurs chiffres dans le fichier.

Lorsque l'on écrit dans un fichier en format texte (par la commande `fichier<<i<<endl;`) l'ordinateur convertit le nombre en base dix et écrit la suite des chiffres, ce qui donne 17. (Il faut savoir que chaque caractère est codé par un octet en mémoire, selon le codage ASCII).

L'avantage du format texte est que l'on peut visualiser le contenu du fichier à l'aide d'un éditeur comme emacs.

Rappels sur le code ASCII :

```
   cout <<(int)'A' << endl; // --affiche 65 qui est le code ASCII de
A
   cout <<(char)(65) << endl; //--affiche A qui a pour code ASCII:
65
```

11.3 Fichiers en format binaire

Il y a cependant la possibilité d'écrire les données dans un fichier de façon brute et compacte, en format binaire. L'avantage est la rapidité, et l'économie de place mémoire. C'est aussi utile (mais non nécéssaire) si l'on veut accéder en lecture ou écriture à un endroit quelconque du fichier.

11.3.0.1 Exemple

Le programme suivant montre comment écrire et lire en format binaire, et comment se positionner dans un fichier.

```
   #include <stdlib>
   #include <iostream>
   using namespace std;
   #include <fstream>
   #include <iomanip>

   int main()
   {
   double x=3.14;
   int i=17;
   int nx=sizeof(double); //nombre de byte occupés par un objet de
classe double
   int ni=sizeof(int); //nombre de byte occupés par un objet de classe
int
   cout<<"nx= "<<nx<<" ni= "<<ni<<endl;
   //==============================
   ofstream sortie( "donnees.dat ",ofstream::binary); //ouverture d un
fichier en ecriture --
   sortie.write(&x, nx); //--- ecriture de x ---
```

```
   sortie.write(&i, ni); //--- ecriture de i ---
   sortie.close(); //- fermeture du fichier --
   //==============================
   ifstream entree( "donnees.dat ", ifstream::binary); //-- ouverture
d un fichier en lecture --
   entree.read(&x, nx); //--- lecture de x ---
   entree.read(&i, ni); //--- lecture de i ---
   cout<<" x= "<<x<<" i= "<<i<<endl;
   entree.seekg(nx); // on se positionne après nx octets en partant
du début, soit après la valeur de x
   entree.read(&i, ni); //--- lecture de i ---
   cout<<" i= "<<i<<endl;
   int pos=entree.tellg(); // donne la position du curseur dans le
fichier (en byte)
   cout<<"la position du curseur est maintenant: "<<pos<<endl;
   }
```

11.3.0.2 Remarques

Pour se positionner en lecture, il faut utiliser la fonction `seekg`. Pour se positionner en écriture, il faut utiliser la fonction `seekp`. Le premier argument de ces fonctions indique le déplacement relatif du pointeur (en byte). Le deuxième argument (facultatif) indique l'origine du déplacement. Ce peut-être :

- le début du fichier : `ios::beg`
- la position actuelle du curseur : `ios::cur`
- la fin du fichier `ios::end`

exemple : `entree.seekg(nx,ios::beg); // on se positionne après nx octets en partant du début`

Chapitre 12

Quelques commandes utiles

12.0.1 La commande system

La commande system() permet de lancer d'autres commandes ou programmes sur l'ordinateur. Par exemple pour lancer la commande "ls" qui affiche la liste des fichiers :

```
#include <iostream> // pour cout
using namespace std;

#include <stdlib.h> // pour system


//----Fonction principale ----------------
int main()
{
 system("ls"); // execute la commande ls. (affiche les fichiers)
}
```

Chapitre 13

Classes (suite)

Cette Section fait suite à la Section 7.

13.1 Surdéfinitions d'opérateurs

13.1.1 Exemple d'opérateur binaire

Pour effectuer le produit scalaire entre deux vecteurs, plutôt que d'utiliser la fonction *Produit* ci dessus, on aimerait utiliser la syntaxe plus agréable : *d=v*w;* cela est possible en redéfinissant l'opérateur * de la façon suivante.

Code à rajouter dans le code des fonctions membres :

```
double operator*(Vect v2)
{
    double p=0;
    p=x*v2.x+y*v2.y+z*v2.z;
    return p;
}
```

Utilisation dans le programme principal :

```
int main()
{
    double d;
    Vect v(1,2,3);
    Vect w(2,3,4);
    d=v*w;
    cout <<"d=" << d << endl;
}
```

13.1.2 Commentaire

— La syntaxe de l'opération * est semblable à celle de la fonction membre `Produit`. Comme pour `Produit`, Il est important de remarquer `Vect v2` est un paramètre de l'opération *. Lorsque l'on écrit d=v*w, l'opération * est appelée pour l'objet v qui se trouve à gauche de * (et non w) ; à l'appel de cette opération, le vecteur v2 prend la valeur de w qui se trouve à droite de *.

— On dit que * est **un opérateur binaire** car lors de l'appel `d=v*w` il y a deux entrées pour *. Il y a à sa gauche l'objet de la classe, et à sa droite un autre objet (à priori quelconque, ici `Vect v2`).

— Les **symboles possibles** pour un opérateur binaire sont :
`(), [ ], -> , *, /, %, +, -, << , >>, <, <=, >, >=, ==, !=, &, ^, ||, &&, |, =, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=,` et la virgule `,` .

Exercice 13.1.1. (*)

1. Ecrivez le code pour effectuer la somme (termes à termes) et la différence de deux vecteurs par la syntaxe : `w=u+v;` `w=u-v;` et testez.
2. Ecrivez le code pour effectuer le produit et la division externe d'un réel a avec un vecteur v (produit de chaque élément) par la syntaxe `w=v*a;` ou `w=v/a;` et testez.
3. On aimerait accéder aux variables membres `v.x, v.y, v.z` respectivement, par la syntaxe `v[1], v[2], v[3]`. Cela est possible en définissant l'opérateur [ ] avec la déclaration `double & operator [ ](int i);` A l'utilisation, on écrit par exemple `d=v[1];`. Ecrivez le code de cet opérateur et testez.
   Rem : le signe & permet d'écrire aussi une affectation comme `v[1]=3;`. Cela sera mieux expliqué au paragraphe "affectations".

13.1.3 Exemple d'un opérateur unaire

Comme en mathématiques, on aimerait donner un sens à $(-v)$ qui est l'opposé du vecteur v. Pour cela il faut écrire :

Code à rajouter dans le code des fonctions membres :

```
Vect operator-()
{
    return (*this)*(-1);
}
```

Utilisation dans le programme principal :

```
int main()
{
    Vect v(1,2,3);
    Vect w;
    w=-v;
    w.Affiche();
```

```
}
```

13.1.4 Commentaires

— On dit que - est **un opérateur unaire** car lors de l'appel `w=-v` car la seule entré de - est l'objet v de la classe qui se trouve à droite. (il n'y a pas d'entrée à gauche).
— Les **symbôles possibles** pour un opérateur unaire sont :
`+,-,++,--,!,~,*,&,new,new[],delete,(cast)`
— **(*this)** dans la fonction membre signifie l'objet en cours. Ici il s'agit du vecteur v. Remarquez la façon économique de calculer le résultat : on utilise la multiplication externe déjà programmée.

Exercice 13.1.2.
— Simplifiez le code de l'opérateur binaire w=u-v déjà écrit en utilisant l'opérateur binaire + et l'opérateur unaire - seulement.
— Simplifiez le code de l'opérateur binaire w=v/a déjà écrit en utilisant l'opérateur binaire * .

13.2 Fonctions amies

13.2.1 Exemple

La syntaxe que permet les fonctions membres ci-dessus a une petite limitation. Dans la classe Vect, on aimerait par exemple définir le produit externe * d'un double a avec un Vect v, et écrire : `w=a*v;`

Mais d'après la disposition des objets, il faudrait que * soit un opérateur binaire de l'objet `a` qui se trouve à gauche . On peut cependant associer cette opération à la classe Vect de l'objet `v` en écrivant :

Code à rajouter dans le code des fonctions membres :

```
friend Vect operator*(double a,Vect v2)
{
    return (v2*a);
}
```

Utilisation dans le programme principal :

```
int main()
{
    double d(2);
    Vect v(1,2,3);
    Vect w;
    w=d*v;
}
```

13.2.2 Commentaire

— Le préfixe *friend* signifie que cet opérateur n'est pas exactement une fonction membre de la classe Vect, mais pas non plus indépendante puisque elle est déclarée avec la classe. Remarquez l'ommission du préfixe de rattachement *Vect::* .
— Les paramètres de l'opération * dans la déclaration sont *(double a,Vect v2)*. Ils se trouvent de part et d'autre de * lors de l'utilisation : *w=a*v;*
— Comme en mathématiques, on a définit a*v=v*a. Mais on pourrait écrire ce que l'on veut et ne pas respecter la commutativité.
— la déclaration *double d(2)* est équivalente à *double d=2.* (La signification est que le constructeur *double(int)* est appelé.)

13.2.3 Exemple de cout <<

On aimerait pouvoir afficher le contenu du vecteur v au moyen de la commande : *cout <<v;* Remarquons que l'objet à gauche de l'opérateur << est "cout" appartenant à la classe ostream (ici "o" signifie output donc affichage, et cette classe standard du C++ est obtenue par *#include <iostream>*). Si l'on veut définir cette opération dans notre classe Vect, il faudra donc la déclarer comme une fonction amie. Exactement comme l'exemple précédent, cela se fait en écrivant :

Code à rajouter dans le code des fonctions membres :

```
friend ostream & operator <<(ostream & sortie, Vect v2)
{

    sortie <<"Vecteur: "<<v2.x <<"/"<<v2.y <<"/"<<v2.z
 <<"/";
    return sortie;

}
```

Utilisation dans le programme principal :

```
int main()
{

    Vect v(1,2,3);
    cout <<v <<endl;

}
```

Exercice 13.2.1. (*)

Ecrivez le code nécessaire afin de pouvoir demander le contenu d'un vecteur à l'utilisateur par la syntaxe *cin >>v;* L'utilisateur doit entrer les composantes à la suite des questions *Vect x=?,y=?,z=?.* La syntaxe est cette fois-ci :

istream & operator >>(istream & entree,Vect & v2);

Le symbole *&* s'appelle passage par références, et est nécessaire puisque le vecteur *v* est modifié à la sortie de la fonction.

13.3 Vecteurs de taille variable

On va améliorer la classe de vecteurs de $\mathbb{R}^3$ précédente, en permettant d'avoir des vecteurs de $\mathbb{R}^N$ avec N quelconque, au choix de l'utilisateur.

Important : avant de commencer, nous conseillons vivement de (re)lire le paragraphe **sur les pointeurs**.

Il apparaît que les variables membres de la nouvelle classe, doivent être :

(Code à écrire dans la déclaration de la classe :)

```
//---- variables membres

    int N; //taille
    double *element; //pointeur
```

Important : la valeur de N nous renseigne à tout moment sur la taille de l'espace mémoire réservé.

Cette modification nous obligera à réécrire les fonctions membres de la classe Vect, mais aussi d'introduire deux nouvelles fonctions membres : le constructeur par recopie, et l'opérateur d'affectation =.

Exercice 13.3.1. Créer une nouvelle classe de vecteurs, qui contient **les variables membres ci-dessus**, avec les **fonctions membres** suivantes :

- Une fonction membre *Libere()* qui libère l'espace mémoire de taille N.
- Une fonction membre *Init(Ni)* qui fait dans l'ordre les opérations suivantes :
  (1) Libère l'emplacement mémoire déjà existante (si N>0)
  (2) :Réserve l'emplacement mémoire de taille Ni , (si Ni>0).
  (3) :met N=Ni.
  (4) met les composantes du vecteur à zéro.
- Un **constructeur** *Vect(),* qui met seulement $N = 0$, et ne réserve pas de place mémoire. (il appelle la fonction : Init(0) ;)
- Un **constructeur** *Vect(int Ni),* pour créer un vecteur de taille Ni (il appelle la fonction : Init(Ni) ;).
- Un **destructeur** *~Vect()* qui libère l'espace mémoire. (il appelle la fonction Libere())

13.3.1 Constructeur par recopie

Dans un exercice précédent, nous avons écrit une fonction membre *double Produit (Vect v2)* . Lors de l'appel de cette fonction par la syntaxe *v.Produit(w)* dans le programme principal, un objet v2 est créé, et l'objet w est **copié** dedans. Mais il était apparu qu'aucun Constructeur n'était explicitement appelé pour cette recopie. La raison est que le compilateur C++ a appelé un **constructeur de recopie** par défaut, qu'il a créé lui-même. Dans ce constructeur par défaut, il est effectuée une recopie simple des variables membres (il s'agissait de x,y,z).

Dans le cas actuel, une recopie simple des variables membres ne suffit pas. A la création du nouvel objet Vect v2, il faut en effet réserver l'emplacement mémoire, et donc écrire nous même **le constructeur par recopie.** Le code est le suivant :

Code à rajouter dans le code des fonctions membres :

```
Vect (const Vect & v2)
{
    // ... code à écrire (exercice)...
}
```

13.3.2 Opérateur d'affectation =

Dans la classe Vect de R^3, si v,w sont deux vecteurs, et que l'on écrit l'opération d'affectation :

```
v=w;
```

alors les composantes de w sont copiées dans celles de v. Cette **opération d'affectation** a été faite automatiquement par le compilateur. Il y a recopie des variables membres x,y,z. Dans notre cas actuel de vecteur à taille variable, on ne peut laisser faire le compilateur, puisqu'il faut libérer l'ancien emplacement mémoire de v, puis lui réserver un nouvel emplacement mémoire, et ensuite copier les composantes de w dans celles de v.

On va écrire nous même le code de **l'opérateur d'affectation** =

Code à rajouter dans le code des fonctions membres :

```
Vect & operator=(const Vect & v2)
{
    // ... code à écrire (exercice)...
}
```

Exercice 13.3.2.

1. Ecrivez le code du **constructeur par recopie** et de l'**opérateur d'affectation** = ci dessus, et testez. (en affichant des messages).
2. Reprenez les **fonctions membres** de la classe Vect de R^3 précédente, et écrivez les versions "Vect de R^N " correspondantes.
   Concernant spécialement **l'opérateur [ ]**, avec N éléments, on décide d'y accéder de la façon suivante :
   `Vect v(N);`
   `v[0],v[1],...,v[N-1],`
   Rem : Dans le code de l'opérateur [ ](int i), il est bien de tester si l'indice i est dans l'intervalle autorisé, et sinon d'afficher un message d'erreur (i.e. si i<0, ou i>N-1).

3. Dans un deuxième temps, on aimerait que **le premier indice** i_1 soit quelconque (et pas forcément $i_1 = 0$). Par exemple, que les trois composantes d'un vecteur de R^3soit `v[2],v[3],v[4],` i.e. $i_1 = 2, N = 3$;
   Précisement, on voudrait qu'un vecteur soit caractérisé par **sa taille** N, et par **son premier indice** i_1, afin d'accéder aux éléments du vecteur par la commande `v[i]` avec $i \epsilon [i_1, i_1 + 1, ..., i_1 + N]$.
   Il faut donc rajouter $i_1$comme variable membre de la classe, et modifier les constructeurs et destructeurs, et fonctions membres si besoin est. A la fin, on aura une classe de vecteur assez sophistiquée, et prête à l'emploi.
4. (Long) Sur le même modèle que la classe vecteur, créer une **classe matrice**, avec les opérations usuelles.
   Rem : on pourra définir une matrice N*M comme étant un tableau de vecteurs, et prendre comme variables membres :
   ```
   int N,M; // taille
   Vect * ligne;
   ```
   Rem : attention aux fonctions `Init() et Libere()` !

Chapitre 14

Gestion de projets avec fichiers .h, Makefile, Git et Doxygen

14.1 Projet avec plusieurs fichiers, fichier .h

Si on developpe un projet il est parfois utile d'écrire le code c++ dans différents fichiers. D'une part pour l'organisation, pour la réutilisation ultérieure, etc..

Voici un projet informatique en C++, très simple (et sans trop de sens) qui est fait de 3 fichiers :

- un fichier main.cc qui contient la fonction main() par laquelle commence le programme.
- un fichier autres.cc qui contient une classe Vect et une fonction f utilisées dans la fonction main.
- un fichier main.h qui contient les déclarations de la classe Vect et de la fonction f. Le suffixe .h signifie header (comme "entêtes").

14.1.0.1 Résultat du programme :

```
Norme de v=1
3*3=9
```

14.1.0.2 Code c++

```
//== fichier main.cc ==============
#include <iostream>
using namespace std;

#include "main.h"

int main()
```

```
{
        Vect v;
        v.x=1;
        cout<<"Norme de v="<<v.Norme()<<endl;

        cout<<"3*3="<<f(3)<<endl;

}
```

Remarque 14.1.1. Au début du fichier main.cc, l'instruction #include "main.h" a pour effet d'inclure le contenu du fichier main.h à cet endroit. Il y a les déclarations (mais pas le code) de la classe Vect et de la fonction f qui sont utilisées dans la fonction main().

```
//== fichier autres.cc ==============
#include "main.h"
#include <math.h>

//--- contenu du constructeur
Vect::Vect()
{
        x=0; y=0; z=0;
}

//-----contenu de la fonction Norme
double Vect::Norme()
{
        return sqrt(x*x+y*y+z*z);
}

//--- code de la fonction f
double f(double x)
{
        return x*x;
}
```

Remarque 14.1.2. Le fichier autres.cc contient le code des fonctions membres de la classe Vect et le code de la fonction f(). Noter la syntaxe particulière Vect : :Norme() qui signifie fonction Norme de la classe Vect.

```
//== fichier main.h ==============
#if !defined(__MAIN_H)
#define __MAIN_H

//------ d'une classe
```

```
class Vect
{
public:
    double x,y,z;
        Vect(); // constructeur
        double Norme();
};

// ----- declaration d'une fonction
double f(double x);

#endif
```

Remarque 14.1.3. Le fichier main.h contient les déclarations (et non pas le code) de la classe Vect et de la fonction f(). Les lignes particulières :

```
#if!defined(__MAIN_H)
#define __MAIN_H
...
#endif
```

ne sont pas nécessaires ici mais sont souvent utiles : ce sont des instructions du "préprocesseur" c'est à dire que au moment de la compilation, si le compilateur voit le code intermédiaire ... la première fois, il le considère (et met la variable __MAIN_H à 1). Les fois suivantes, il ne considère par cette partie de code. L'intérêt de cela est que si le fichier main.h se trouve répété à cause d'inclusions multiples (par exemple le fichier A.cc inclut main.h et B.h, et le fichier B.h inclut main.h), alors le code n'est écrit qu'une seule fois, ce qui évite une erreur de "code déjà déclaré".

14.1.0.3 Commandes de compilation

```
g++ -c main.cc -o main.o
g++ -c autres.cc -o autres.o
g++ main.o autres.o -o main
```

Remarque 14.1.4. Le première ligne signifie "on compile le fichier c++ main.cc et on fabrique un fichier en langage machine main.o". De même pour la deuxièmre ligne. La troisième ligne signifie "à partir des fihciers main.o et autres.o on fabrique le fichier exécutable final main".

14.2 Un programme Makefile pour gérer la compilation

Dans l'exemple de la section 14.1.0.3, si les fichiers sont importants et si on effectue une modification du code que dans le fichier `main.cc`, on souhaiterait de pas refaire la commande de compilation "`g++ -c autres.cc -o autres.o`". Le programme `Makefile` permet de gérer cela automatiquement (il regarde les dates de modification des fichiers).

Voici un tutoriel sur le Makefile. Voici le manuel de make.

```
# --- fichier Makefile
main.o: main.cc main.h
	g++ -c main.cc -o main.o

autres.o: autres.cc main.h
	g++ -c autres.cc -o autres.o

final : main.o autres.o
	g++ main.o autres.o -o main
```

Remarque 14.2.1. Attention le décalage en début de ligne est une **tabulation**.

Commande de compilation on écrit :

```
make final
```

résultat :

```
g++ -c main.cc -o main.o
g++ -c autres.cc -o autres.o
g++ main.o autres.o -o main
```

Commentaires :

- Dans le fichier Makefile, les deux lignes
  `main.o: main.cc main.h`
  `g++ -c main.cc -o main.o`
  signifient que si on souhaite "`main.o`", cela dépend des fichiers `main.cc` et `main.h`, et si un de ces derniers a été modifié il faut éxécuter la commande `g++ -c main.cc -o main.o`
- la commande make final signifie que l'on souhaite "`final`". Cela a pour effet d'enclencher les commandes de compilation nécessaires. Essayer par exemple de modifier seulement le fichier `main.cc` et refaites `make final`. Le résultat est :
  `g++ -c main.cc -o main.o`
  `g++ main.o autres.o -o main`

14.2.1 Un programme Makefile plus pratique

Le programme suivant est totalement équivalent au précédent mais il est plus pratique car on a placé des instructions de compilation dans des variables (en lettres majuscules). Ainsi si vous rajoutez des librairies, il suffit de la faire à un seul endroit. De plus grâce au symbols $@ et $* on évite des réécritures.

```
# --- fichier Makefile

# .... librairies
LIBR = -lm

# ... instructions de compilation
CC =g++ -std=c++11

OBJETS= main.o autres.o

main.o: main.cc main.h
	$(CC) -c   $*.cc -o $@

autres.o: autres.cc main.h
	$(CC) -c   $*.cc -o $@

final : main.o autres.o
	$(CC) $(OBJETS) -o main $(LIBR)

lib:
	ar r main

clean:
	rm *.o
```

Remarque 14.2.2.

- Par exemple dans ce tutorial, on utilise souvent beaucoup de librairies et au total on écrira :
  `LIBR=$(shell root-config --libs) $(shell root-config --glibs) -lm -lpthread -lsndio -larmadillo -std=c++11 -lboost_system`
  `CC =g++ -std=c++11 $(shell root-config --cflags) # compilateur`
- On a rajouté l'entrée clean qui permet d'effacer tous les fichiers en langage machine .o par la commande `make clean`
- On a rajouté l'entrée lib pour créer une librairie.

14.2.2 Generation automatique du fichier makefile ?

Voir generation avec GNU

14.3 Gestion de versions d'un projet avec Git

Le logiciel Git permet de gérer différentes versions d'un projet. Il se souvient de l'historique. On lui indique une liste de fichiers ("le dépot") et régulièrement on lui demande de faire une sauvegarde (par l'instruction commit)

14.3.0.1 Utilisation de Git sur son ordinateur

]La premiere fois :
— Dans le répertoire de travail, taper
`git init`
Cela initialise un répertoire .git/
— Ecrire (en mettant votre nom et adresse) :
`git config --global user.name "Frederic Faure"`
`git config --global user.email frederic.faure@univ-grenoble-alpes.fr`
Cela modifie le fichier `~/.gitconfig`

A faire au cours du projet :
— `git status` :montre l'état des fichiers par rapport au dépot.
— **git add fichier.txt** ou **git add '*.cc'** ou **git add *** pour ajouter un (des) fichier(s) au dépot
— **git commit -m "first commit"** fait une sauvegarde des fichiers du dépot avec le titre indiqué
— **git log** pour voir l'historique des opérations
— **git rm fichier** :enleve un fichier du dépot
— **git mv file1 file2** : deplace ou change le fichier.
— **git diff** pour lire les differences
— Dans le fichier **.gitignore** il y a les extensions et repertoires à ignorer.

14.4 Collaboration et partage de projets sur internet avec GitHub

Si vous développez un projet avec des collaborateurs, vous pouvez déposer (gratuitement) votre projet sur le site github (par exemple).
Au départ il faut vous créer un compte sur github qui va héberger votre projet.

]Pour poser/ retirer l'archive sur le site web de github :
Si votre projet s'appelle `projet`, et votre compte s'appelle `fdrfaure`

Commandes :

- `git remote add projet https://github.com/fdrfaure/projet.git`
  Cela associe votre projet local au dépot sur le site web
  - `git push -u projet master`
    Cela dépose les dernières modifications de votre projet sur le site web
- `git pull expanding_map master`
  Cela permet de récupérer les dernières modifications (posées par les collaborateurs) qui sont sur le site web.
- `git clone https://github.com/fdrfaure/projet.git repertoire_local`
  Cela copie le dépot du site web sur votre répertoire local. (a faire une premiere fois pour copier un projet)

14.5 Commenter et documenter un projet avec Doxygen

Nous expliquons l'utilisation de base de doxygen pour documenter automatiquement un projet en c++.

14.5.1 Création et consultation de la documentation

14.5.1.1 Création de la documentation html

- Se placer dans le répertoire du projet.
- Dans une console, lancer le programme doxywizard et configurer les paramètres. On génère la documentation par "Run". Cela crée des répertoires : html et latex.

14.5.1.2 consultation de la documentation

- On lit le fichier hml/index.html
- Pour la documentation pdf, dans le répertoire latex, executer la commande make. Cela génère le fichier latex/refman.pdf

14.5.2 conventions pour écrire les commentaires

14.5.2.1 Page principale :

(voir mainpage)
Ecrire :

```
/*! \mainpage Titre
\section Nom_section
texte...
*/
```

14.5.2.2 Le résultat suivant :

Header 1

Header 2

Link text **Link text**

- Item 1

  More text for this item.

- Item 2
  - nested list item.
  - another nested item.
- Item 3

1. First item.
2. Second item.

lignes:

---

single asterisks*

single underscores

double asterisks**

double underscores

The link text](http://example.net/) The link text The link text The link text The link text

Caption text
Caption text
Caption text
img def

http://www.example.com> https://www.example.com ftp://www.example.com address@example.com address@example.

First Header	Second Header
Content Cell	Content Cell
Content Cell	Content Cell

Right	Center	Left
10	10	10
1000	1000	1000

This is a paragraph introducing:

```
a one-line code block
```

```
int func(int a,int b) { return a*b; }
```

```
also a fenced code block
```

est obtenu avec le code suivant (syntaxe markdown) :

```
Header 1 {#labelid}
========
## Header 2 ## {#labelid2}
[Link text](#labelid)
[Link text](@ref labelid)
- Item 1
More text for this item.
- Item 2
+ nested list item.
+ another nested item.
- Item 3
1. First item.
2. Second item.
lignes:
- - -
------
single asterisks*
_single underscores_
double asterisks**
__double underscores__
The link text](http://example.net/)
[The link text](http://example.net/ "Link title")
[The link text](/relative/path/to/index.html "Link title")
[The link text](somefile.html)
[The link text](@ref MyClass)
![Caption text](/path/to/img.jpg)
![Caption text](/path/to/img.jpg "Image title")
![Caption text][img def]
![img def]
[img def]: /path/to/img.jpg "Optional Title"
http://www.example.com>
<https://www.example.com>
<ftp://www.example.com>
<mailto:address@example.com>
<address@example.com>
First Header | Second Header
------------- | -------------
Content Cell | Content Cell
Content Cell | Content Cell
| Right | Center | Left |
| ----: | :----: | :---- |
```

```
    | 10 | 10 | 10 |
    | 1000 | 1000 | 1000 |
    This is a paragraph introducing:
    ~~~~~~~~~~~~~~~~~~~~

    a one-line code block
    ~~~~~~~~~~~~~~~~~~~~

    ~~~~~~~~~~~~~~~{.c}
    int func(int a,int b) { return a*b; }
    ~~~~~~~~~~~~~~~

    ```
    also a fenced code block
    ```
    Mathematical Inline Formula:
    \f$\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}\f$.

    or

    \f[
    |I_2|=\left| \int_{0}^T \psi(t)
    \left\{
    u(a,t)-
    \int_{\gamma(t)}^a
    \frac{d\theta}{k(\theta,t)}
    \int_{a}^\theta c(\xi)u_t(\xi,t)\,d\xi
    \right\} dt
    \right|
    \f]

    or

    \f{eqnarray*}{
    g &=& \frac{Gm_2}{r^2} \\
    &=& \frac{(6.673 \times 10^{-11}\,\mbox{m}^3\,\mbox{kg}^{-1}\,
    \mbox{s}^{-2})(5.9736 \times 10^{24}\,\mbox{kg})}{(6371.01\,\mbox{km})^2}
\\
    &=& 9.82066032\,\mbox{m/s}^2
    \f}
```

14.5.3 Fichier de commentaires avec la syntaxe markdown

On peut écrire le fichier README.md du projet avec la syntaxe “markdown”. Voici une référence sur Markdown. avec tutorial.

14.6 Compte rendu de projet en pdf et html avec Lyx

Lyx est un logiciel qui produit des documents scientifiques en différents formats (pdf, html, etc).

La documentation est bien faite. Voici quelques informations précises.

— On peut exporter en html : Exporter en xhtml avec Lyxhtml. Insertion de videos sous format gif animé.

— On peut insérer des schémas techniques avec des notations en Latex, grâce au logiciel xfig ou inkscape (dans ce logiciel, on a les formules latex par extensions/rendu/Latex).

— Pour des présentation on peut choisir le style **Beamer**

Chapitre 15

Interface interactive pour le C++

— Il y a le projet cling. Voir Cling.
— Il y a le projet SWAN qui permet d'effectuer du C++ scientifique de façon interactive sur le "cloud".

Troisième partie

Compléments sur les librairies du langage C++

Dans les chapitres suivants nous présentons quelques librairies C++ utiles.
Nous conseillons de consulter la page de Boost qui présente d'autres librairies.
Voici en particulier une liste de librairies pour l'audio, le graphisme.

Chapitre 16

Librairie Root : graphisme et autre.. (compléments)

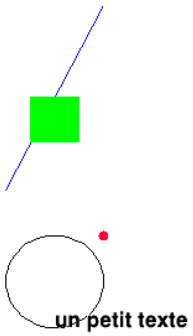
Nous avons déjà vu une introduction à cette librairie en Section 3.4. Cette Section apporte quelques compléments.

Voici aussi la Documentation générale.

16.1 Graphisme de base 2Dim

16.1.1 Objets graphiques

Voici quelques exemples de graphisme 2D. Remarquez que au début du programme on ajoute une ligne spécifique par exemple `#include <TLine.h>` si on utilise la classe TLine.




```
#include <TApplication.h>
#include <TCanvas.h>
#include <TLine.h>
#include <TEllipse.h>
```

```
   #include <TMarker.h>
   #include <TBox.h>
   #include <TText.h>

   int main()
   {
    TApplication theApp("App", nullptr, nullptr);

   //.. La fenetre
    TCanvas *c=new TCanvas("titre","titre",400,400);
    c->Range(0,0,1,1);

   //.. une ligne bleue
    TLine *l=new TLine(0.1,0.5,0.3,0.9); // x1,y1,x2,y2.
    l->SetLineColor(kBlue); // bleu
    l->Draw();

   //.. Une Ellipse (ou cercle)
    TEllipse *e= new TEllipse(0.2,0.3,0.1); // (centre (x,y) et rayon
r
   e->Draw();

   //.. Un point
    TMarker *m=new TMarker(0.3,0.4,8); // (x,y) et 6 ou 8: forme du
Marker
    m->SetMarkerColor(kPink);
    m->Draw();

   //.. Un rectangle vert
    TBox *p=new TBox(0.15,0.6,0.25,0.7); // on precise les coins bas-gauche
(x1,y1) et haut droit (x2,y2) :
    p->SetFillColor(kGreen);
    p->Draw();

   //.. Du texte
    TText *texte=new TText(0.2,0.2,"un petit texte"); // position x,y
    texte->Draw();

    c->Update(); // Montre le dessin
    theApp.Run();
   }
```

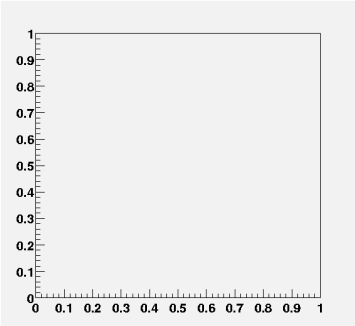
16.1.2 Dessiner dans plusieurs fenêtres

Avec la classe TCanvas qui hérite de TPad.

```
#include <TApplication.h>
#include <TCanvas.h>
#include <TMarker.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
TCanvas c1("c1","fenetre1",400,400); // creation de la fenetre c1
TCanvas c2( "c2","fenetre2",400,400); // creation de la fenetre
c2
TMarker *m1=new TMarker(0.3,0.4,8); // un point
TMarker *m2=new TMarker(0.3,0.4,8); // un point
m2->SetMarkerColor(kRed);
c1.cd(); // choix de la fenetre c1 pour le prochain dessin
m1->Draw(); // dessin du point 1
c2.cd(); // choix de la fenetre c2 pour le prochain dessin
m2->Draw(); // dessin du point 2
theApp.Run();
}
```

16.1.3 Un cadre avec des axes gradués : DrawFrame

Avec la classe TPad.

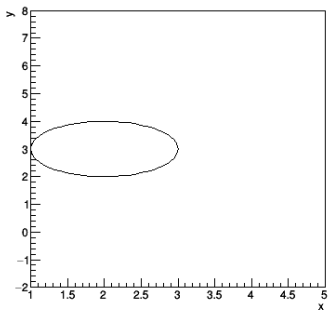



```
#include <TApplication.h>
#include <TCanvas.h>
int main()
{
 TApplication theApp("App", nullptr, nullptr);
```

```
 TCanvas *c=new TCanvas("titre","titre",400,400);
 double x1(0),y1(0),x2(2),y2(1); // position du cadre
 c->DrawFrame(x1,y1,x2,y2);

 theApp.Run();
}
```

On peut compléter le dessin du cadre de la façon suivante. On récupère un pointeur sur un histogramme de la classe TH1, et on a accès aux fonctions membres pour modifier le cadre, par exemple :

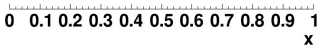
```
#include <TApplication.h>
#include <TCanvas.h>
#include <TEllipse.h>
#include <TH1F.h>

int main()
{
 TApplication theApp("App", nullptr, nullptr);

 TCanvas *c=new TCanvas("titre","titre",400,400);
 double x1(0),y1(0),x2(10),y2(10); // position du cadre
 TH1F *h =c->DrawFrame(x1,y1,x2,y2);
 TEllipse e(2,3,1);
 e.Draw();
 h->SetXTitle("x");
 h->SetYTitle("y");
 h->SetMinimum(-2); // change les bornes en y
 h->SetMaximum(8);
 h->SetBins(1000,1,5); // change les bornes en x: 1->5
 c->Update(); // Montre le dessin

 theApp.Run();
}
```



16.1.4 Un seul axe gradué : TGaxis

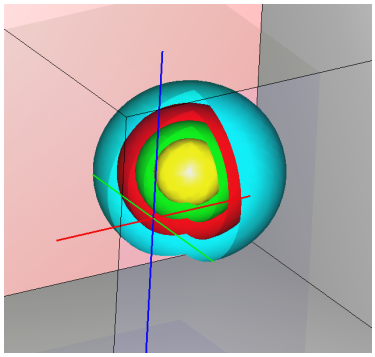
Avec la classe TGaxis.

```
#include <TApplication.h>
#include <TCanvas.h>
#include <TGaxis.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
TCanvas *c=new TCanvas("titre","titre",400,400);
double x1(0.1),y1(0.5),x2(0.9),y2(0.5); // position de l'axe
double v1(0),v2(1); // graduations extremes
TGaxis axe(x1,y1,x2,y2,v1,v2,510,"");
axe.SetTitle("x"); // met le titre sur l'axe
axe.Draw();
theApp.Run();
}
```



16.1.5 Couleurs

voir Couleurs



```
#include <TApplication.h>
#include <TCanvas.h>
#include <TStyle.h>
#include <TColor.h>
#include <TH3.h>
//====================
main ()
```

```
   {
   TApplication theApp("App", nullptr, nullptr);
   //---- on defini une palette de 5 couleurs
   const int n = 5; //nbre de couleurs
   float rgb[n * 3] =
   {0.9f, 0.60f, 0.f,
   0.f, 1.f, 1.f,
   1.f, 0.f, 0.f,
   0.f, 1.f, 0.f,
   1.f, 1.f, 0.f};
   int palette[n] = {0};
   for (Int_t i = 0; i < n; ++i)
   palette[i] = TColor::GetColor(rgb[i * 3], rgb[i * 3 + 1], rgb[i
* 3 + 2]);
   gStyle->SetPalette(n, palette);
   //------------------------
   gStyle->SetCanvasPreferGL(kTRUE); // permet d'utiliser openGL
   TCanvas *c = new TCanvas("glc","TH3 Drawing", 400,400);
   int N=50;
   double x1=-3,x2=3,y1=-3,y2=3,z1=-3,z2=3;
   TH3D *h = new TH3D("h", "h", N, x1, x2, N, y1, y2, N, z1, z2);
   for(int i=0;i<N;i++)
   for(int j=0;j<N;j++)
   for(int k=0;k<N;k++)
   {
   double x= (x2-x1)*i*1./N+x1;
   double y= (y2-y1)*j*1./N+y1;
   double z= (z2-z1)*k*1./N+z1;
   double f= exp(-x*x-y*y-z*z); // formule f(x,y,z)
   h->SetBinContent(i+1,j+1,k+1, f);
   }
   h->SetFillColor(kBlue);
   h->SetContour(n);
   h->Draw("gliso");// glbox1 glbox glcol gliso
   theApp.Run();
   }
```

16.2 Graphisme 1D

Voici quelques exemples d'utilisation des Histogrammes. Pour utiliser ces exemples, il faut remplacer les parties (C) de l'exemple de la Section 3.4.2, par le code ci-dessous.

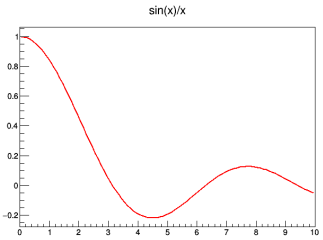
Attention : la ligne `#include <..>` mentionnée chaque fois est à mettre en haut de votre programme.

La page de THistPainter montre les nombreuses possibilités de représentation avec les histogrammes. Nous présentons quelques une d'entre elles.

16.2.1 Courbe a partir de l'expression d'une formule $y = f(x)$: TF1

Avec la classe TF1.

16.2.1.1 Exemple à partir d'une formule littérale :




```
#include <TApplication.h>
#include <TF1.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
//------------
TF1 *f1 = new TF1("f1","sin(x)/x",0,10);
f1->Draw();
//-------------------
theApp.Run();
}
```

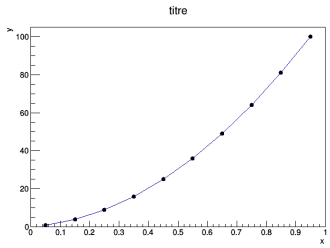
16.2.1.2 Même résultat à partir d'une fonction C++

```
#include <TApplication.h>
#include <TF1.h>
//---------------------
double F(double *px, double *p)
{
double x= px[0];
return sin(x)/x;
```

```
}
//---------------
int main()
{
TApplication theApp("App", nullptr, nullptr);
//------------
TF1 *f1 = new TF1("f1",F,0,10);
f1->Draw();
//--------------------
theApp.Run();
}
```

16.2.2 Histograme 1Dim ou Courbe $y(x)$ à partir de tableau de valeurs $y(i)$: TH1D

Avec la classe TH1D.

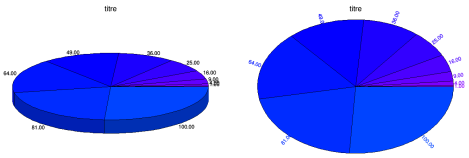



```
#include <TApplication.h>
#include <TH1.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
//------------
int nx(10); // nombre de points
double xmin(0.0),xmax(1.0); // graduation de l'axe x
TH1D *h1=new TH1D("nom","titre",nx,xmin,xmax);
h1->SetXTitle("x");
h1->SetYTitle("y");
for(int i=1;i<=nx;i++) // remplissage
{
double z=i*i;
h1->SetBinContent(i,z);
}
h1->SetStats(0);// pas de boite de resultats stat
```

```
h1->SetMarkerStyle(8);//8: gros point
h1->Draw("L P"); // option L: ligne ou P: marker
//--------------------
theApp.Run();
}
```

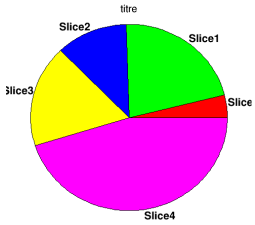
16.2.2.1 Représentation en diagramme “Camembert” (Pie Chart)

Voir TPie.
Même exemple ci-dessus avec l'option `h1->Draw("PIE 3d");` et `h1->Draw("PIE rsc");`



16.2.3 Représentation d'un tableau de valeurs $y(i)$ sous forme de “camembert” ou “Pie”

Voir TPie.




```
#include <TApplication.h>
#include <TPie.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
//------------
int n= 5; // nbre de valeurs
double valeurs[5] = {.2, 1.1, .6, .9, 2.3};
int couleurs[5] = {2,3,4,5,6};
TPie *p = new TPie("p", "titre", n, valeurs, couleurs);
p->Draw();
//--------------------
theApp.Run();
}
```

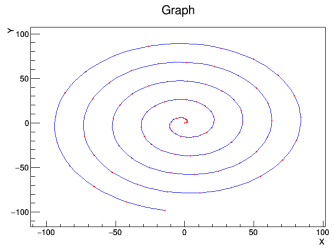
16.2.4 Courbe paramétrée $(x(i), y(i))$ avec axes à partir de tableaux de valeurs $x(i), y(i)$: TGraph

Attention : si les valeurs $x(i)$ son équidistribuées il est préférable d'utiliser la classe TH1 ci-dessous. La classe TGraph est préférables si les valeurs $x(i)$ et $y(j)$ ne sont pas équidistribuées.

Avec la classe TGraph.

Voir Options de dessin.

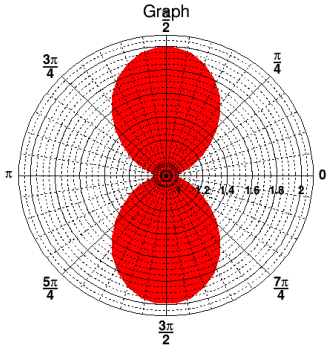
Exemple :




```
#include <TApplication.h>
#include <TGraph.h>
#include <TPad.h>
#include <TAxis.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
//---- creation de tableaux
int n = 100;
double x[100], y[100];
for (int i=0;i<n;i++) // remplissage
{
double r = i;
x[i] = r*cos(0.3* i);
y[i] = r*sin(0.3 * i);
}
//--- dessin
TGraph *gr = new TGraph(n,x,y);
gr->SetLineColor(kBlue);
gr->SetMarkerColor(kRed);
gr->SetMarkerStyle(6); // 6:petit cercle 8: gros
gr->GetXaxis()->SetTitle("X");
gr->GetYaxis()->SetTitle("Y");
```

```
gr->Draw("A C P"); // AC: trace un segment entre les points
// gPad->BuildLegend(); // rajoute la legende
//-------------------
theApp.Run();
}
```

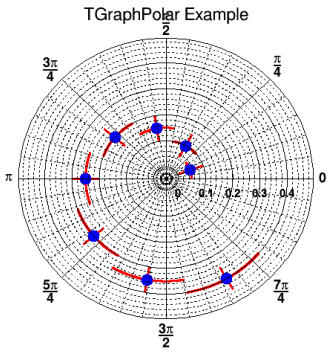
16.2.5 Courbe paramétrée $(r(i), \theta(i))$ représentée en coordonnées polaire à partir de points $r(i), \theta(i)$



Voir Options de dessin.

```
#include <TApplication.h>
#include <TCanvas.h>
#include <TGraphPolar.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
TCanvas * c = new TCanvas("CPol","TGraphPolar Example",500,500);
int N=50;
double theta[N];
double radius[N];
for (int i=0; i<N; i++)
{
theta[i] = i*(2*M_PI/N);
radius[i] = sin(theta[i])* sin(theta[i]) +1 ;
}
TGraphPolar * grP1 = new TGraphPolar(N, theta, radius);
grP1->SetFillColor(kRed);
grP1->Draw("f");
c->Update();
theApp.Run();
}
```

Autre exemple :

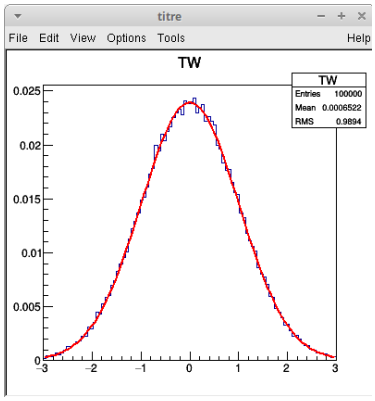


Voir Options de dessin.

```
#include <TApplication.h>
#include <TCanvas.h>
#include <TGraphPolar.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
TCanvas * c = new TCanvas("CPol","TGraphPolar Example",500,500);
double theta[8];
double radius[8];
double etheta[8];
double eradius[8];
for (int i=0; i<8; i++) {
theta[i] = (i+1)*(M_PI/4.);
radius[i] = (i+1)*0.05;
etheta[i] = M_PI/8.;
eradius[i] = 0.05;
}
TGraphPolar * grP1 = new TGraphPolar(8, theta, radius, etheta, eradius);
grP1->SetTitle("TGraphPolar Example");
grP1->SetMarkerStyle(20);
grP1->SetMarkerSize(2.);
grP1->SetMarkerColor(kBlue);
grP1->SetLineColor(kRed);
grP1->SetLineWidth(3);
grP1->Draw("PE");
c->Update();
theApp.Run();
}
```

16.2.6 Crée histogramme ou fonction 1D $y = f(x)$ par tirage aléatoire et ajustement par une formule : TF1

Remarque 16.2.1. voir HowtoFit.html



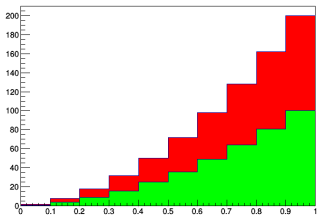
Code :

```
#include <TRandom.h>
#include <TApplication.h>
#include <TCanvas.h>
#include <TF1.h>
#include <TH1.h>
#include <iostream>
using namespace std;
int main()
{
 TApplication theApp("App", nullptr, nullptr);
 TCanvas *c=new TCanvas("titre","titre",400,400);
//------- cree histo avec tirage aleatoire avec loi gaussienne ------------
 int nbin=100; double xmin=-3,xmax=3;
 TH1D *h=new TH1D("TW","TW",nbin,xmin,xmax);
gRandom->SetSeed(); // initialise sur l'horloge
for(int i=1;i<=1e5;i++) //tirages
 {
 double x= gRandom->Gaus(0,1); // c'est P(x)=exp(-0.5*(x/sigma)^2)
 h->Fill(x);
 }
 h->Draw();

//---- calcule integrale et normalise l'histogramme -----------
double I= h->Integral();
```

```
   h->Scale(1/I);
   h->Draw();

   //--- Fit l'histogramme avec une Gaussienne --------------
   TF1 *f1 = new TF1("f1","gaus"); // ou pol0, pol1,...polN pour polynome
degre N, ou expo
   h->Fit("f1");
   cout<<"Ecart type ="<<f1->GetParameter(2)<<endl;

   //--- montre fenetres de commandes ---
   h->DrawPanel();
   h->FitPanel();
    //------
   c->Update();
    //------
    theApp.Run();

   }
```

16.2.7 Superposition d'histogrammes : THStack

Classe THStack.



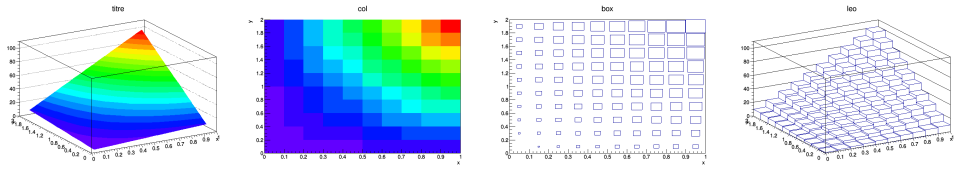
```
   #include <TApplication.h>
   #include <TH1.h>
   #include <THStack.h>
   int main()
   {
   TApplication theApp("App", nullptr, nullptr);
   //---- cree un histogramme --------
   int nx(10); // nombre de points
   double xmin(0.0),xmax(1.0); // graduation de l'axe x
   TH1D *h1=new TH1D("nom","titre",nx,xmin,xmax);
   h1->SetXTitle("x");
   h1->SetYTitle("y");
   for(int i=1;i<=nx;i++) // remplissage
```

```
   {
   double z=i*i;
   h1->SetBinContent(i,z);
   }
   h1->SetFillColor(kGreen);
   //-------------- une copie ----------
   TH1D *h2 = (TH1D*)h1->Clone();
   h2->SetFillColor(kRed);
   //-------------------------
   THStack *hs = new THStack("hs","");
   hs->Add(h1);
   hs->Add(h2);
   hs->Draw(); // dessin les uns sur les autres. nostack: superposes
, nostackb, lego1
   //-------------------
   theApp.Run();
   }
```

16.3 Graphisme 2D

16.3.1 Surface 2Dim : $z = f(x, y)$ à partir d'un tableau de valeurs $z(i, j)$: TH2D

Avec la classe TH2D.



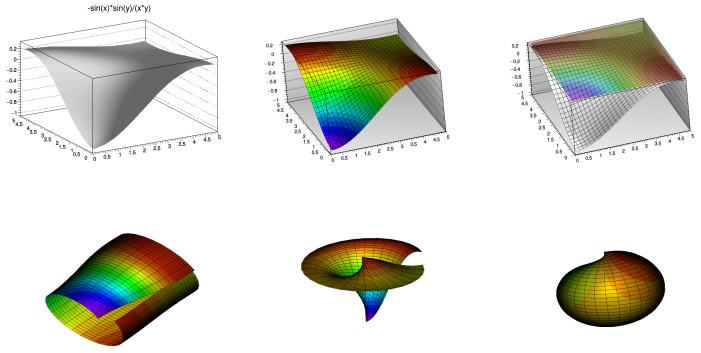

```
#include <TApplication.h>
#include <TH2D.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);

//------------
int nx(10),ny(10);
double xmin(0),xmax(1),ymin(0),ymax(2);
TH2D *h= new TH2D("h1","titre",nx,xmin,xmax,ny,ymin,ymax);
h->SetXTitle("x");
```

```
h->SetYTitle("y");
h->SetStats(kFALSE); // pas de panneau stat
for(int i=1;i<=nx;i++) // remplissage
for(int j=1;j<=ny;j++)
{
 double z=i*j;
 h->SetCellContent(i,j,z);
}
h->Draw("surf2"); // essayer options: surf1, surf3,surf4,lego
//--------------------

theApp.Run();
}
```

16.3.2 Surface 2D a partir de l'expression d'une formule $z = f(x, y)$: TF2

Avec la classe TF2.

Voici le résultat du programme suivant avec différentes options, en particulier, représentation cylindrique, polaire, spherique et en utilisant openGL.

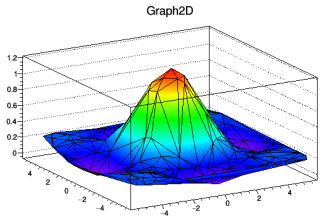


```
#include <TApplication.h>
#include <TF2.h>
#include <TStyle.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
//------------
gStyle->SetCanvasPreferGL(kTRUE); //si option gl.. cidessous
TF2 *f2 = new TF2("f2","-sin(x)*sin(y)/(x*y)",0,5,0,5);
```

```
   f2->Draw("surf4"); // glsurf1, glsurf2, glsurf3, glsurf3, glsurf1cyl,
glsurf1pol, glsurf1sph
   //-------------------
   theApp.Run();
   }
```

16.3.3 Surface 2D $z(x,y)$ à partir d'un ensemble arbitraire de points $x(i), y(i), z(i)$: TGraph2D

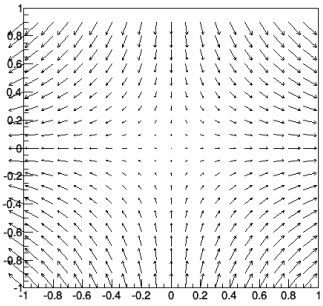
La classe TGraph2D permet de dessiner une surface $z(x,y)$ à partir de la donnée d'un ensemble de N points (x_i, y_i, z_i).




```
   #include <TApplication.h>
   #include <TGraph2D.h>
   #include <TCanvas.h>
   #include <TRandom.h>
   #include <TStyle.h>
   using namespace std;
   main ()
   {
   TApplication theApp("App", nullptr, nullptr);
   TCanvas *c = new TCanvas("c","Graph2D example",0,0,600,400);
   double x, y, z, P = 6.;
   int N = 200;
   TGraph2D *g = new TGraph2D();
   TRandom *r = new TRandom();
   for(int i=0; i<N; i++)
   {
   x = 2*P*(r->Rndm(N))-P; // choix de (x,y) au hasard dans le carré
[-P,P]
   y = 2*P*(r->Rndm(N))-P;
   z = (sin(x)/x)*(sin(y)/y)+0.2; // formule de la surface lisse
   g->SetPoint(i,x,y,z); // ajoute point (x,y,z) a l'ensemble
   }
   gStyle->SetPalette(1); // couleurs. autres choix: 0, 1, 2
   g->Draw("tri1"); // autres choix: tri, triw, tri1, tri2, p,po, pcol,line
```

```
theApp.Run();
}
```

16.3.4 Champ de vecteur en dimension 2



/// Ce code dessine un champ de vecteur spécifié à deux dimensions.

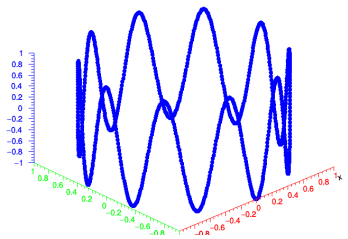
```
#include <TApplication.h>
#include <TCanvas.h>
#include <TArrow.h>
#include <TLine.h>
#include <math.h>
#include <iostream>
using namespace std;
///--- definition du champ de vecteur
void vect(double x, double y, double &vx, double &vy)
{
 vx = x ;
 vy= -y;
}
//-----------------------------------
int main()
{
TApplication theApp("App", nullptr, nullptr);
//------------
double xmin=-1, xmax=1, ymin=-1,ymax=1; // coordonnées
TCanvas *c=new TCanvas("c","c",500,500); // fenetre
c->DrawFrame(xmin,ymin,xmax,ymax);
//------------------
double N=20; // nbre d'echantillons
double dx=(xmax-xmin)/N;
double dy=(ymax-ymin)/N;
 for (int i=0; i<N; i++)
```

```
for (int j=0; j<N; j++)
{
 double x=xmin + dx*i;
double y=ymin + dy*j;
double vx,vy;
vect(x,y,vx,vy);// calcul les coordonnées du champ de vecteur
vx *= dx *1; // normalise
vy *= dy *1;
 double n=sqrt(vx*vx+vy*vy); // norme
TArrow *t= new TArrow(x,y,x+vx,y+vy,dx*n);
 // TLine *t= new TLine(x,y,x+vx,y+vy);
t->Draw();
 } // for i,j
//--donne la main a l'utilisateur (A)-
theApp.Run();
}
```

16.4 Graphisme 3D

16.4.1 Des points dans $\mathbb{R}^3$: TPolyMarker3D

Doc : TPolyMarker3D.
Remarque : ensuite avec le menu on peut régler leur opacité etc



```
#include <TApplication.h>
#include <TPolyMarker3D.h>
int main()
{
TApplication theApp("App", nullptr, nullptr);
//------------
int N=1000;//nbre de points
TPolyMarker3D *p = new TPolyMarker3D(N);
for( int i = 0; i < N; i++ )
{
```

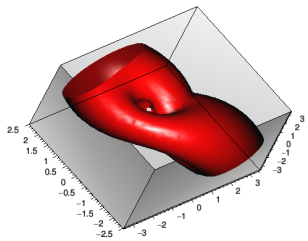
```
double theta=i/double(N)*2*M_PI;
double x=cos(theta), y=sin(theta), z=cos(10*theta);
p->SetPoint( i, x, y, z );
}
p->SetMarkerSize( 1 );
p->SetMarkerColor( kBlue);
p->SetMarkerStyle( 8 );
p->Draw();
//--------------------
theApp.Run();
}
```

16.4.2 Surface déterminée par $f(x, y, z) = 0$: TF3

Ici la surface dans $\mathbb{R}^3$ est déterminée par l'équation $f(x, y, z) = 0$ avec f donnée par son expression.

On utilise la classe TF3.

the options "FB" and "BB" suppress the "Front Box" and "Back Box" around the plot.



```
#include <TApplication.h>
#include <TCanvas.h>
#include <TStyle.h>
#include <TPaveLabel.h>
#include <TFormula.h>
#include <TF3.h>
// utiliser la mollette de la souris pour zoomer
// et touche 's' pour changer la couleur
main ()
{
TApplication theApp("App", nullptr, nullptr);
//--- prepare la fenetre 3D
gStyle->SetCanvasPreferGL(1);
```

```
   TCanvas *cnv = new TCanvas("glc", "TF3: Klein bottle", 200, 10,
600, 600);
   TPad *tf3Pad = new TPad("box", "box", 0.04, 0.04, 0.96, 0.8);
   tf3Pad->Draw();
   //--- definition et dessin de l'objet 3D
   TFormula f1 = TFormula("f1", "x*x + y*y + z*z + 2*y - 1");
   TFormula f2 = TFormula("f2", "x*x + y*y + z*z - 2*y - 1");
   TF3 *tf3 = new TF3("Klein Bottle","f1*(f2*f2-8*z*z) + 16*x*z*f2",
   -3.5, 3.5, -3.5, 3.5, -2.5, 2.5);
   tf3->SetFillColor(kRed);
   tf3Pad->cd();
   tf3->Draw("gl"); // gl, default, kMaple0, kMaple1, kMaple2 (touche
's')
   theApp.Run();
   }
```

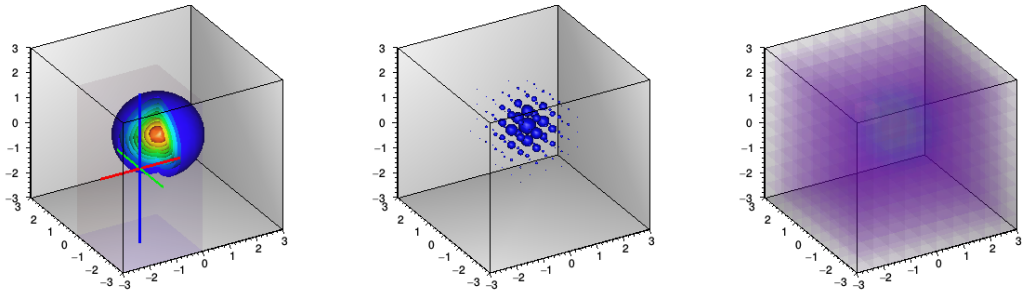
16.4.3 Fonction $f(x, y, z) \in \mathbb{R}$: TH3

Doc : TH3.

Dessin avec openGL et en option : dessin de la surface de niveau de f ("gliso") ou boules montrant la valeur de f ("glbox") ou densité transparente "glcol".

Remarques :

— avec l'option surface de niveau "gliso" la touche c permet de voir des coupes et les déplacer avec la souris.



```
   #include <TApplication.h>
   #include <TCanvas.h>
   #include <TStyle.h>
   #include <TH3.h>
   main ()
   {
   TApplication theApp("App", nullptr, nullptr);
```

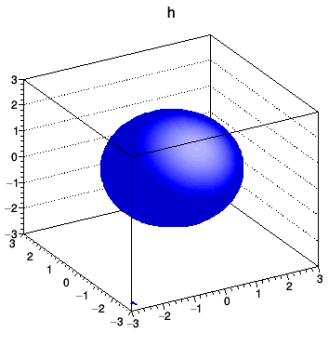
```
   gStyle->SetCanvasPreferGL(kTRUE); // mettre avant declaration de
la fenetre TCanvas
   TCanvas *c = new TCanvas("glc","TH3 Drawing", 400,400);
   int N=30;
   double x1=-3,x2=3,y1=-3,y2=3,z1=-3,z2=3;
   TH3D *h = new TH3D("h", "h", N, x1, x2, N, y1, y2, N, z1, z2);
   for(int i=0;i<N;i++)
   for(int j=0;j<N;j++)
   for(int k=0;k<N;k++)
   {
   double x= (x2-x1)*i*1./N+x1;
   double y= (y2-y1)*j*1./N+y1;
   double z= (z2-z1)*k*1./N+z1;
   double f= exp(-x*x-y*y-z*z); // formule f(x,y,z)
   h->SetBinContent(i+1,j+1,k+1, f);
   }
   h->SetFillColor(kBlue);
   double levels[2] = {2, 2.5}; // valeurs des niveaux
   h->SetContour(2,levels); // on choisit deux niveaux pour le dessin
iso
   h->Draw("gliso");// glbox1 glbox glcol gliso
   theApp.Run();
   }
```

16.4.3.1 Dessin d'une surface de niveau souhaitée

Sans l'option "gl", mais seulement "iso", la surface de niveau dessinée est pour la valeur $C = \frac{1}{N_x N_y N_z} \sum_{i,j,k} f(x_i, y_j, z_k)$. Par conséquent si on souhaite montrer la ligne $f(x, y, z) = C_0$ avec C_0 arbitraire, il faut par exemple modifier le contenu du point $i, j, k = 1, 1, 1$ en écrivant $f(x_1, y_1, z_1) = f(x_1, y_1, z_1) + \delta$ avec $\delta = -\sum_{i,j,k} f(x_i, y_j, z_k) + N_x N_y N_z C_0$ de sorte que

$$C = \frac{1}{N_x N_y N_z} \left(\sum_{i,j,k} f(x_i, y_j, z_k) + \delta \right) = C_0$$

Voici un exemple qui dessine la surface $\sqrt{x^2 + y^2 + z^2} = 2$ c'est à dire la sphère de rayon 2 :

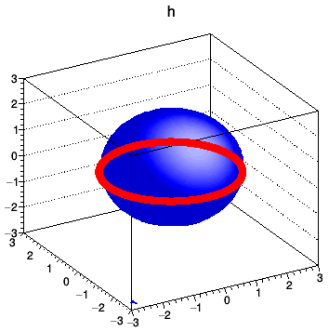



```
#include <TApplication.h>
#include <TCanvas.h>
#include <TStyle.h>
#include <TH3.h>
main ()
{
TApplication theApp("App", nullptr, nullptr);
//--------------------
double C0=2; // niveau souhaite
TCanvas *c = new TCanvas("glc","TH3 Drawing", 400,400);
int N=30;
double x1=-3,x2=3,y1=-3,y2=3,z1=-3,z2=3;
TH3D *h = new TH3D("h", "h", N, x1, x2, N, y1, y2, N, z1, z2);
double sum=0;
for(int i=0;i<N;i++)
for(int j=0;j<N;j++)
for(int k=0;k<N;k++)
{
double x= (x2-x1)*i*1./N+x1;
double y= (y2-y1)*j*1./N+y1;
double z= (z2-z1)*k*1./N+z1;
double f= sqrt(x*x+y*y+z*z); // fonction
h->SetBinContent(i+1,j+1,k+1, f);
sum+=f;
}
double delta = -sum + (N*N*N) *C0;
h->SetBinContent(1,1,1, h->GetBinContent(1,1,1)+delta);
h->SetFillColor(kBlue);
h->Draw("iso");
theApp.Run();
}
```

16.4.3.2 Dessin d'une courbe sur une surface

On peut combiner le dessin d'une iso-surface de la Section 16.4.3.1 avec le dessin d'une courbe de la Section 16.4.1 (mais sans l'option GL malheuresement).

Voici un exemple : un grand cercle équateur sur la sphère de rayon 2. Malheuresement, la partie arrière de la courbe n'est pas cachée.




```
#include <TApplication.h>
#include <TCanvas.h>
#include <TStyle.h>
#include <TH3.h>
#include <TPolyMarker3D.h>
main ()
{
TApplication theApp("App", nullptr, nullptr);
//--------------------
double C0=2; // niveau souhaite
TCanvas *c = new TCanvas("glc","TH3 Drawing", 400,400);
int N=30;
double x1=-3,x2=3,y1=-3,y2=3,z1=-3,z2=3;
TH3D *h = new TH3D("h", "h", N, x1, x2, N, y1, y2, N, z1, z2);
double sum=0;
for(int i=0;i<N;i++)
for(int j=0;j<N;j++)
for(int k=0;k<N;k++)
{
double x= (x2-x1)*i*1./N+x1;
double y= (y2-y1)*j*1./N+y1;
double z= (z2-z1)*k*1./N+z1;
double f= sqrt(x*x+y*y+z*z); // fonction
h->SetBinContent(i+1,j+1,k+1, f);
sum+=f;
}
double delta = -sum + (N*N*N) *C0;
```

```
h->SetBinContent(1,1,1, h->GetBinContent(1,1,1)+delta);
h->SetFillColor(kBlue);
h->Draw("iso");
//------------
int N2=1000;//nbre de points
TPolyMarker3D *p = new TPolyMarker3D(N2);
for( int i = 0; i < N2; i++ )
{
double theta=i/double(N2)*2*M_PI;
double x=2*cos(theta), y=2*sin(theta), z=0;
p->SetPoint( i, x, y, z );
}
p->SetMarkerSize( 1 );
p->SetMarkerColor( kRed);
p->SetMarkerStyle( 8 );
p->Draw();
//-------------------
theApp.Run();
}
```

16.5 Utilisation de la souris

Dans une fenetre gaphique, chaque objet capture la souris si elle passe à proximité. Pour l'observer, vous pouvez activer l'option "EventStatus" dans le menu de la fenetre.

Voici un code complet qui definit une classe que l'on appelle **Fenetre** (héritée de la classe TCanvas). La nouveauté est la fonction `HandleInput()` qui est appelée chaque fois que la souris bouge devant la fenetre. Lorsque le programme execute l'instruction `theApp.Run()`; il est en attente d'évènements provenant du clavier ou de la souris. Si un tel évènement se produit, il appelle la fonction `void HandleInput(EEventType event, Int_t px, Int_t py)`, en passant les paramètres de l'évènement. C'est donc dans cette fonction que vous allez rajouter du code (ou faire appel à d'autres fonctions).

```
#include <iostream>
using namespace std;
#include <TCanvas.h>
#include <TMarker.h>
#include <TApplication.h>
#include <TEllipse.h>
//====Ma Classe Fenetre================================
class Fenetre: public TCanvas
{
 public:
 // constructeur ......
```

```
    Fenetre(Text_t* name, Text_t* title, Int_t ww, Int_t wh) : TCanvas(name,title,w
    { }
    //-- Cette fonction est appelle si la souris s'agitte. --
    void HandleInput(EEventType event, Int_t px, Int_t py)
    {
    //- affiche l'etat de la souris ---
    cout<<" position: px="<<px<<" py="<<py<<" code du bouton="<<event<<endl;
    //- convertit la position en pixel px,py en coordonnees (x,y) --
    double x = gPad->AbsPixeltoX(px);
    double y = gPad->AbsPixeltoY(py);
    // si bouton gauche appuye dessine un point -
    if(event==1)
    {
    TMarker *m=new TMarker(x,y,8);
    m->Draw();
    Update();
    }
    //------ Event clavier -------
    if (event==kKeyPress) // touche appuyée
    {
    cout<<"touche px="<<px<<endl;
    switch (px)
    {
    //--- change de mode (tempéré <-> juste) ---
    case 'm':
    cout<<"Touche m appuyée"<<endl;
    break;
    }; // switch px
    } // if event
    }
   };
   ///-----------------------------------------
   int main()
   {
    TApplication theApp("App", nullptr, nullptr);
    Fenetre *c = new Fenetre("c", "fenetre1", 400,400); // crée une
fenetre
    double xmin(0),ymin(0),xmax(5),ymax(5);
    c->Range(xmin,ymin,xmax,ymax); // coordonnees de la fenetre (optionnel,
par defaut: 0-1)
    theApp.Run();

   }
```

16.5.1 Pour qu'un objet graphique capture les evenements souris

Dans l'exemple suivant une ellipse change d'état selon les evenement souris (clique gauche). Pour cela il faut créer une classe dérivée qui possède la fonction `ExecuteEvent(int event, int px, int py)`.

```
#include <iostream>
using namespace std;
#include <TCanvas.h>
#include <TApplication.h>
#include <TEllipse.h>
//==============
class Cercle: public TEllipse
{
public:
// constructeur ......
Cercle(double x, double y, double r): TEllipse(x,y,r)
{ }
int col = kRed;
//-- Cette fonction est appelle si la souris s'agitte. --
void ExecuteEvent(int event, int px, int py)
{
//--- change l'epaisseur du cercle
SetLineWidth(3);
Draw();
gPad->Update();
SetLineWidth(1);
Draw();
//--- si clique gauche
if(event == 1)
{
if(col == kRed)
col = kBlue;
else
col = kRed;
SetFillColor(col);
Draw();
}
}
};
```

```
   ///-----------------------------------------
   int main()
   {
   TApplication theApp("App", nullptr, nullptr);
   TCanvas *c = new TCanvas("c", "fenetre1", 400,400); // crée une
fenetre
   double xmin(0),ymin(0),xmax(4),ymax(4);
   c->Range(xmin, ymin, xmax, ymax); // coordonnees de la fenetre (optionnel,
par defaut: 0-1)
   Cercle *C1= new Cercle(3,2,0.5);
   C1->SetFillColor(kRed);
   C1->Draw();
   Cercle *C2= new Cercle(1,2,0.5);
   C2->SetFillColor(kRed);
   C2->Draw();
   theApp.Run();
   }
```

16.5.2 Pour gérer la souris lorsque le programme calcule

Si votre programme est lancé dans un calcul interminable, et que vous vouliez cependant modifier des paramètres par l'intermédiaire de la souris ou d'un commande clavier, il faut dans votre boucle mettre la commande suivante :

```
gSystem->ProcessEvents(); // avec #include <TSystem>
```

Ainsi si la souris est actionnée, dans l'exemple ci-dessus, la fonction `Fenetre::ExecuteEvent` sera appelée.

16.6 Comment créer un fichier gif animé

Voici un exemple complet qui permet de créer un fichier "film.gif" animé à partir d'images créées par root.

Travail préalable :

1. Fichiers à télécharger et sauver dans son répertoire : bib_fred.h et bib_fred.cc.
2. Il faut avoir installé le logiciel **gifsicle**,

```
#include <TApplication.h>
#include <TCanvas.h>
#include <TEllipse.h>
#include "bib_fred.cc"
```

```
///-----------------------------------
int main()
{
TApplication theApp("App", nullptr, nullptr);
 TCanvas *c= new TCanvas("c","fenetre",400,400);
 double xmin(-1),ymin(-1),xmax(2),ymax(2);
 c->Range(xmin,ymin,xmax,ymax);
 int periode = 10; // temps entre chaque image en 1/100e sec.
int imin=1, imax=30;
for(int i =imin; i<=imax ; i++)
{
 double x = i/(double)imax;
 TEllipse e(x,0.5,1);
 e.Draw(); // dessine l'ellipse
c->Update();
Animation(c,i,imin,imax,"film",periode); // -> crée fichier film.gif
 }
theApp.Run();
}
```

Ensuite pour visualiser le film vous pouvez utiliser la commande `firefox film.gif`.

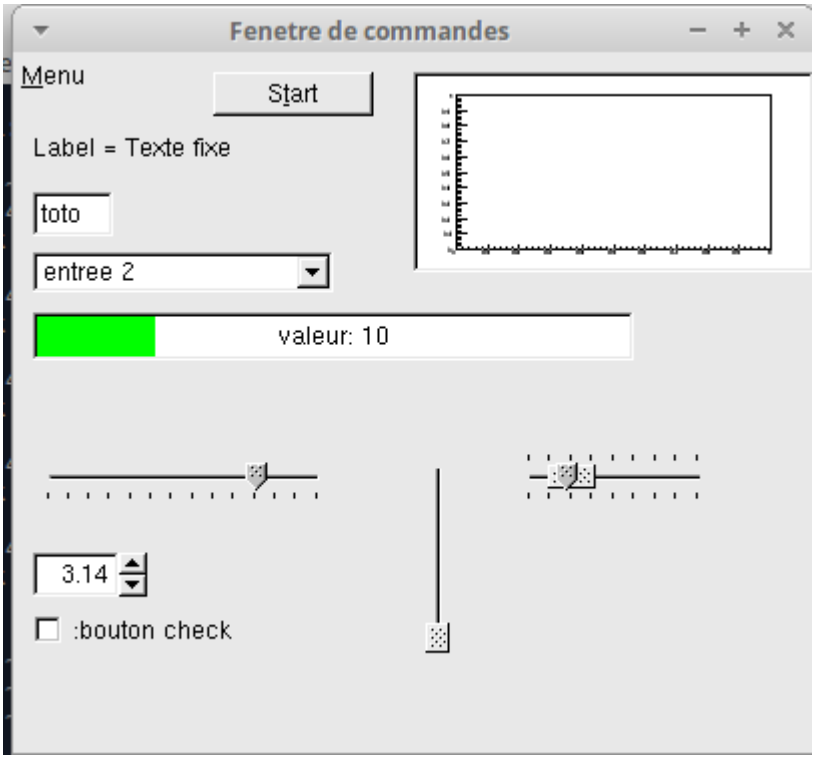
16.7 Boutons de commandes (Widgets et GUI)

Le mot "widget" est une contraction de "windows" et "gadget". GUI signifie "Graphical User Interface".

- Documentation
- Exemple très complet qui montre presque toutes les possibilités et comment le faire : guitest.cc à copier/coller. (attention, il y a une erreur : remplacer (char **) par (const char **)).
- Voir aussi les nombreux exemples dans /usr/share/doc/root/tutorials/gui que l'on execute avec l'interpréteur et la commande .x nom.C

16.7.1 Exemple simple avec quelques widgets

Voici un exemple plus simple, à copier/coller, donnant l'interface :




```
/*! ==========================
* Exemple simple d'une fenetre de commande.
* Il y a deux classes: la classe Com qui est la fenetre de commande
*  et qui permet de commander une autre classe A
*  Chaque classe a un lien vers l'autre afin de pouvoir communiquer.
*/

#include <TCanvas.h>
#include <TGFrame.h>
#include <TGButton.h>
#include <TGComboBox.h>
#include <TGTextEntry.h>
#include <TGLabel.h>
#include <TGProgressBar.h>
#include <TPad.h>
#include <TFrame.h>
#include <TH1F.h>
#include<TSystem.h>
#include <TGNumberEntry.h>
#include <TGSlider.h>
#include <TGDoubleSlider.h>
#include <TGTripleSlider.h>
```

```
#include <TGMenu.h>
#include <TApplication.h>
#include <TRootEmbeddedCanvas.h>
#include <thread>
#include <chrono>
#include <iostream>

using namespace std;

class A; // classe definie aprÃÍs

/*!==========fenetre de commandes ======================
 * Description de la classe Com
 */
class Com: public  TGMainFrame  // (derivee d une fenetre GUI)
{
public :

    TGMenuBar *fMB; // barre de menu
    TGPopupMenu *fM; // menu
    TGTextButton  *fB; // bouton
    TGTextEntry    *fT; // zone texte
    TGLabel      *fL; //  texte fixe ecrit sur la fenetre
    TGComboBox *fC;  //  liste deroulante
        TGCheckButton *fCB;

    TGHProgressBar *fH; // barre de progres
    TGNumberEntry *fNE;
    TGHSlider *fS;
    TGDoubleVSlider *fS2;
    TGTripleHSlider *fS3;
    TRootEmbeddedCanvas *fEC; // fenetre

    //--------------------
    A *a;  // lien vers un objet a d'une classe A
    //--------------------
    Com(A *a_i);  // constructeur
    ~Com(); // destructeur


    Bool_t ProcessMessage(Long_t msg, Long_t parm1, Long_t); // fonction appe
    //=============================================
```

```
    // met a jour les valeurs de la fenetre

    void  Met_a_jour(); // fonction qui met Ãă jour les valeurs de la fenetre
};

/*!================
* Une classe pour effectuer un calcul ...
 */

class A
{
public:
    double x;
    //--------------
    void Calcul()
    {
        cout<<"calcul:";
        for(x=1; x<=50; x++)
        {
            this_thread::sleep_for(chrono::milliseconds {100}); // attend
0.5 ms// attente,
            com->Met_a_jour();
            cout<<x<<","<<flush;
        }
        cout<<endl;
    }

    //-------------
    Com *com; // pointeur vers l'objet fenetre de control

    void Lance_commandes()
    {
        TApplication theApp("App", NULL,NULL);
        com=  new Com(this); // cree fenetre de control et l'associe a
l'objet present
        theApp.Run(); //----donne la main a l'utilisateur ---
    }


};

/*!====================
```

```
Constructeur
 */
Com::Com(A *a_i) : TGMainFrame(gClient->GetRoot(),400,350)  // taille de la f
{
    a=a_i; // initialise le lien vers l'objet Ãă controller (sera utile pour
    a->com=this;
    //------------------
    SetWindowName("Fenetre de commandes"); // nom de la fenetre

//---------- Menu ------------
    fMB = new TGMenuBar(this, 35, 50, kHorizontalFrame);
    AddFrame(fMB, new TGLayoutHints(kLHintsTop | kLHintsExpandX, 2, 2, 2, 5))
    fM=fMB->AddPopup("&Menu");
    fM->AddEntry("&Option 1",3);
    fM->AddSeparator();
    fM->AddEntry("O&ption 2",4);
    fM->Associate(this);


    //-----boutons --------------------------

    fB= new TGTextButton(this,"S&tart",2);  // cree bouton, lettre t, et
code message  2
    fB->Move(100,10); // position x,y
    fB->SetToolTipText("Demarre un calcul ..."); // texte d'aide
    fB->Associate(this);
    fB->Resize(80, fB->GetDefaultHeight()); // taille  (x,y)

    //..Label = texte fixe.................
    fL= new  TGLabel(this, new TGString("Label = Texte fixe"));
    fL->Move(10,40); // position x,y

    //...Zone de texte ................

    string text="toto";
    fT = new TGTextEntry(this,text.c_str(),1); //code 1
    fT->Move(10,70); // position x,y
    fT->Resize(40,fT->GetDefaultHeight()); // taille  (x,y)
    char buf[100]="texte";
    fT->Associate(this);

    //------- Liste Combo ----
```

```
    fC = new TGComboBox(this, 88);
    fC->AddEntry("entree 1",1);
    fC->AddEntry("entree 2",2);
    fC->Resize(150, 20);// taille (x,y)
    fC->Select(2); // selection a priori
    AddFrame(fC, new TGLayoutHints(kLHintsTop | kLHintsLeft,5,5,5,5)); // mis
    fC->Move(10,100); // position

    //----- Progress Bars----------------

    fH = new TGHProgressBar(this,TGProgressBar::kFancy,300);
    fH->ShowPosition();
    fH->SetBarColor("green");
    fH->Move(10,130); // position
    fH->SetRange(0.0,50.); // min, max
    fH->SetPosition(10.); // position
    fH->ShowPosition(kTRUE,kFALSE,"valeur: %.0f");


//---------- Sliders ----------
    fS = new TGHSlider(this,150,kSlider1|kScaleDownRight,2); // simple, Id=2
    fS->SetRange(0,50);
    fS->SetPosition(39);
    fS->Move(10,200);

    fS2 = new TGDoubleVSlider(this,100,kDoubleScaleNo,3); //double
    fS2->SetRange(-10,10);
    fS2->Move(200,200);

    fS3 = new TGTripleHSlider(this,100,kDoubleScaleBoth,4, //triple
                              kHorizontalFrame);
    fS3->SetConstrained(kTRUE);
    fS3->SetRange(0,10);
    fS3->SetPosition(2,3);
    fS3 ->SetPointerPosition(2.5);
    fS3->Move(250,200);



//-------- entree numerique ------
    double val=3.1415;
    fNE = new TGNumberEntry(this, val, 5, 400,(TGNumberFormat::EStyle) 2); //
```

```
    fNE->Associate(this);
    fNE->Move(10,250); // position

//----- Fenetre Canvas ----------------

    fEC = new TRootEmbeddedCanvas("ec1", this, 200, 100);
    AddFrame(fEC, new TGLayoutHints(kLHintsTop | kLHintsLeft | kLHintsExpandX
                                    kLHintsExpandY, 5, 5, 5, 5)); // les opti
    fEC->Move(200,10);
    TCanvas *c = fEC->GetCanvas();


    c->DrawFrame(0,0,1,1);
    c->Update();
  //  c->Modified();



        //....Check Button
        fCB= new TGCheckButton(this,":bouton check",420);
        fCB->SetToolTipText("texte d'aide pour bouton check");
        fCB->Associate(this);
        fCB->Resize(120, fCB->GetDefaultHeight());
        fCB->Move(10,280);

//-------------

    MapSubwindows(); // dessin des objets (bouttons ,...)
    MapWindow(); // dessin de la fenetre
}


/*!==========================================
*  Destructeur
*/
Com::~Com()
{
    delete fB;
    delete fT;

    delete fH;
    delete fC;
```

```
}


/*!==========================================
 *  Fonction appelee lorsque l'utilisateur agit  sur la fenetre de commandes
 *  -> Ici on gÃÍre les actions
 */
Bool_t Com::ProcessMessage(Long_t msg, Long_t p1, Long_t p2)
{

    int M = GET_MSG(msg), S=GET_SUBMSG(msg);


    cout<<" M = "<<M<<"  S = "<<S<<"  p1="<<p1<<"  p2="<<p2<<endl;


    switch (M)
    {
    case 1:

        switch (S)
        {
        case 3:

            switch(p1)
            {
            case 2:
                cout<<"bouton 2"<<endl;
                a->Calcul();
                break;
            }
                        break;
                case 4:
                        if(p1==420)
                        {

                                cout<<"bouton check .. "<<endl;

                                if(fCB->IsOn())
                                        cout<<"  on."<<endl;
                                else
                                        cout<<"  off."<<endl;
```

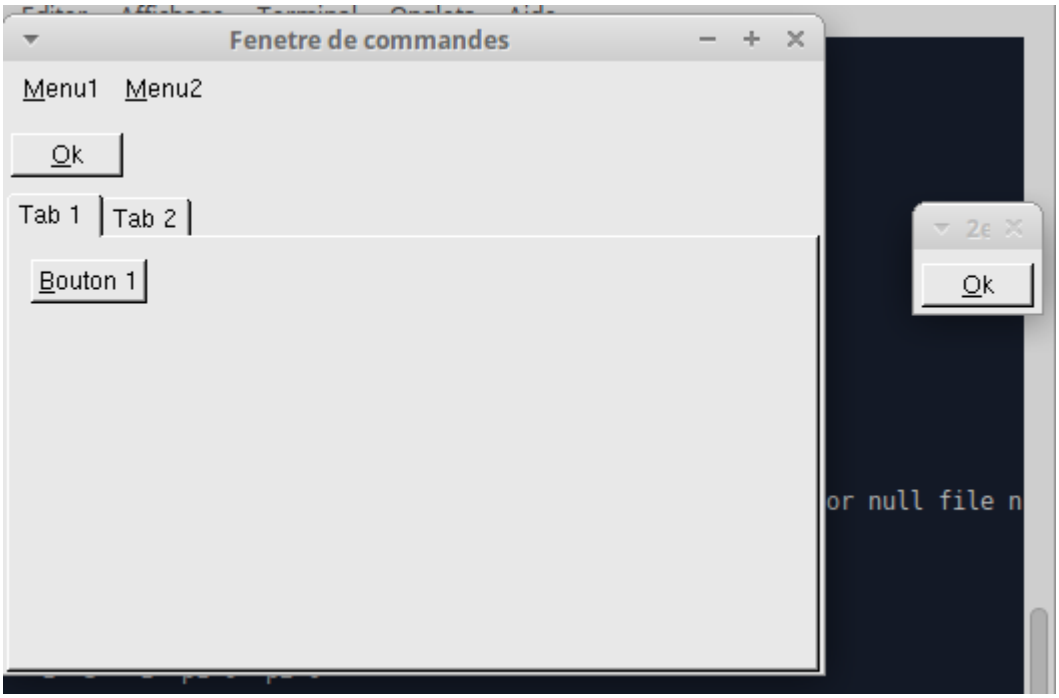
```
                        }
                        break;
        case 7:
            cout<<"combo choisi: "<<fC->GetSelected()<<endl;
            break;
        }

        case 4:

        switch (S)
        {
        case 1:
            cout<<"texte change"<<endl;
            switch (p1)
            {
            case 1:
                cout<<"Le nouveau texte est: "<<((fT->GetBuffer())->GetString
                break;
            case 400:
                cout<<"Le nouveau chiffre est: "<<((fNE->GetNumber()))<<endl;
                break;
            }
            break;
        }
        break;
    default:
        break;
    }

    Met_a_jour();
    return kTRUE;
}

/*! ========================================
 * Fonction  appelÃ©e par une fonction externe
 *  pour mettre a jour les valeurs de la fenetre,
 */
void  Com::Met_a_jour()
{
    fH->SetPosition(a->x); // progress Bar
    //  gClient->NeedRedraw(fH);
```

```
    gSystem->ProcessEvents();  // handle GUI events
}

/*!===Main=============================================================
*/
int main(int argc, char **argv)
{
    A *a=new A;
    a->Lance_commandes();
    return 0;
}
```

Remarque 16.7.1.

1. fNumber->GetNumberEntry()->SetToolTipText("Whatever text you want to see here...") permet de rajouter un texte d'aide à un widget.

16.7.2 Exemple simple avec un menu, des zones, des tabs et une fenetre secondaire




```
/*! ===================================
 * Exemple simple d'une fenetre de commande.
 * avec Menu et Tab et sous fenetre libre
 */
```

```
#include <TCanvas.h>
#include <TGFrame.h>
#include <TGButton.h>
#include <TGMenu.h>
#include <TApplication.h>
#include <TGTab.h>

#include <iostream>

using namespace std;

//---- identifieurs (int) pour les messages de commandes
enum Command_id
{
        M_1, M_2, M_3, M_4, B_1, B_2
};

//=============================================
// Fenetre secondaire
class Fen2 : public TGTransientFrame
{

public:
        //---- constructeur
        Fen2(const TGWindow *p, const TGWindow *main, UInt_t w, UInt_t h,
UInt_t options = kVerticalFrame): TGTransientFrame(p, main, w, h, options)
                {

                        SetCleanup(kDeepCleanup);

                        //--------- zone avec  bouton

                        auto fFrame1 = new TGHorizontalFrame(this, 60, 20, kF

                        auto fOkButton = new TGTextButton(fFrame1, "&Ok", B_1
                        fOkButton->Associate(this);

                        fFrame1->AddFrame(fOkButton);


                        AddFrame(fFrame1);
```

```
                        //---------

                        MapSubwindows();
                        Resize();   // resize to default size
                        CenterOnParent();                       // position r
                        SetWindowName("2eme fenetre");
                        MapWindow();
                        //fClient->WaitFor(this);     // otherwise canvas cont
                }

        //---- destructeur
    virtual ~Fen2()
                {
                        // Delete test dialog widgets.
                }

        //------
        virtual void CloseWindow()
                {
                        cout<<"fenetre Fen2 fermeeï£?"<<endl;

           // Add protection against double-clicks
                        //              fOkButton->SetState(kButtonDisabled);

                        DeleteWindow();
                }
        //------
        virtual Bool_t ProcessMessage(Long_t msg, Long_t p1, Long_t p2)
                {
                        int M = GET_MSG(msg), S=GET_SUBMSG(msg);
                        cout<<" Fen2:  M = "<<M<<"  S = "<<S<<"  p1="<<p1<<"
p2="<<p2<<endl;
                }
};

/*!==========fenetre principales ========================
 */
class Com: public  TGMainFrame
{
public :

        int X=300, Y=150; // taille en pixels
```

```
//===== Constructeur

Com() : TGMainFrame(gClient->GetRoot(), X,Y)
        {

                SetCleanup(kDeepCleanup); //??

    //-- regles de positionnement
                TGLayoutHints *fLH_TX =  new TGLayoutHints(kLHintsTop
                TGLayoutHints *fLH_C = new TGLayoutHints(kLHintsCente
                TGLayoutHints *fLH_L = new TGLayoutHints(kLHintsLeft,
                TGLayoutHints *fLH_R = new TGLayoutHints(kLHintsRight



    //---------- Menu ----------
                TGMenuBar *fMB = new TGMenuBar(this, X, Y);//, kHoriz
                AddFrame(fMB, fLH_TX);

                TGPopupMenu *fM=fMB->AddPopup("&Menu1");
                fM->AddEntry("&Fenetre",M_1);
                fM->AddSeparator();
                fM->AddEntry("O&ption 2",M_2);
                fM->Associate(this);

                TGPopupMenu *fM2=fMB->AddPopup("&Menu2");
                fM2->AddEntry("&Option 1",M_3);
                fM2->AddSeparator();
                fM2->AddEntry("O&ption 2",M_4);
                fM2->Associate(this);



                //---------- zone superieure


                auto  * fFrame1 = new TGHorizontalFrame(this, X, Y);/

                AddFrame(fFrame1,  fLH_L);


                //.. bouton 1
                TGButton    *fOkButton = new TGTextButton(fFrame1, "&C
```

```
                fOkButton->Associate(this);
                fFrame1->AddFrame(fOkButton, fLH_L);

        //.. bouton 2
                TGButton    *fOkButton2 = new TGTextButton(fFrame1, "&
                fOkButton2->Associate(this);
                fOkButton2->Move(50,50);
                fFrame1->AddFrame(fOkButton2, fLH_L);




                //----- Tab --------------------


                TGTab *fTab = new TGTab(this, X,Y);
                AddFrame(fTab);


                //.. Tab1   ------------

                TGCompositeFrame *tf1 = fTab->AddTab("Tab 1");

                //.. bouton  1
                TGTextButton *  fB1 = new TGTextButton(tf1, "&Bouton
                fB1->Associate(this);
                tf1->AddFrame(fB1);

                //.. bouton  2
                TGTextButton *  fB2 = new TGTextButton(tf1, "&Bouton
                fB2->Associate(this);
                tf1->AddFrame(fB2);


                //.. Tab2 ................
                TGCompositeFrame *tf2 = fTab->AddTab("Tab 2");


                //-- fenetre  generale----------

                MapSubwindows();
```

```
                        Resize();
                        MapWindow();
                        SetWindowName("Fenetre de commandes"); // nom de la f



                }

//=============================
        ~Com() // destructeur
                {
                }

//=============================================
        Bool_t ProcessMessage(Long_t msg, Long_t p1, Long_t p2)
                {
                        int M = GET_MSG(msg), S=GET_SUBMSG(msg);
                        cout<<"Fen:  M = "<<M<<"  S = "<<S<<"  p1="<<p1<<"
p2="<<p2<<endl;



                        if(M==1 && S==1 && p1 ==M_1 && p2== 0)
                                new Fen2(fClient->GetRoot(), this, 400, 200);



                        return kTRUE;
                }

}; // fin de la classe




/*!===Main=============================================================
*/
int main(int argc, char **argv)
{
        TApplication theApp("App", nullptr, nullptr);
        Com *com=  new Com(); // cree fenetre de control et l'associe a
```

```
l'objet present
        theApp.Run(); //----donne la main a l'utilisateur ---

    return 0;
}
```

16.7.3 Construire une interface de widgets à la souris avec GuiBuilder

Root propose un logiciel pour construire le code à la souris : GuiBuilder. Cela permet au moins de voir la syntaxe du code pour réaliser une construction.

Voici comment l'utiliser.

1. Dans un terminal de commandes lancer la commande `root`. Dans la ligne de commande qui apparait lancer la commande `new TGuiBuilder`. Cela lance le logiciel de creation d'interface.

16.8 Autres trucs

16.8.0.1 Nombre aléatoire avec Root

Voir TRandom.

```
#include <TRandom.h>
gRandom->SetSeed(0); // initialise le hasard sur l'horloge
double x= gRandom->Uniform(2); // nombre aléatoire dans l'intervalle
(0,2)
```

16.8.0.2 Quelques Options générales de Root

avec

```
#include <TROOT.h>
```

- Mode Batch (sans sortie écran + rapide et permet de le lancer sur un serveur.) :
  - `gROOT->SetBatch(kTRUE); // met en mode Batch`
  - `gROOT->IsBatch(); // permet de savoir si on est en mode batch.`
- `gROOT->SetStyle("Plain"); // style standard`
- Rendre les dessins éditables (modifiables) :
  - `gPad->SetEditable(0); // rend les dessins non modifiables`
  - `gPad->SetEditable(1); // rend les dessins modifiables`
- `TCanvas *c=h->GetCanvas(); //`Pour obtenir la fenetre TCanvas à partir d'un histogramme TH1 *h
- `gStyle->SetPadGridX(kTRUE); // grilles en X,Y`
  `gStyle->SetPadGridY(kTRUE);`

Chapitre 17

Mesure du temps

On présente ici la classe chrono.
Mais on peut aussi conseiller la classe Boost.Date_Time.

17.1 Mesure de la date et du temps écoulé

17.1.0.1 Mesure très précise du temps :

```
#include <chrono>
using namespace std::chrono;
#include <iostream>
using namespace std;

int main()
{

auto t1=high_resolution_clock::now(); // mesure du "time point 1"

//....... instruction qui prend du temps .....
for(int i=0;i<1e5;i++)
cout<<" \t"<<i<<flush;
//.............

auto t2=high_resolution_clock::now(); // mesure du "time point 2"

auto dt1=duration_cast<nanoseconds>(t2-t1).count(); // calcul de
la durée en ns
auto dt2=duration_cast<milliseconds>(t2-t1).count(); // calcul de
la durée en ms
```

```
    auto dt3=duration_cast<duration<double>>(t2-t1).count(); // calcul
de la durée en s

    cout<<"durée: "<<dt1<< "ns"<<" = "<<dt2<< "ms"<<" = "<<dt3<<" s."<<endl;
    }
```

Affiche :

```
0 1 2 3 ...99996 99997 99998 99999
durée: 491264417ns = 491ms = 0.491264 s.
```

17.1.0.2 Affichage de la date

Au programme précédent, si on rajoute :

```
    auto tt=system_clock::to_time_t(t1); // convertit au type time_t
    cout << "Maintenant c'est: " << ctime(&tt);
```

On obtient :

```
Maintenant c'est: Mon Aug 25 19:37:30 2014
```

17.2 Attendre

17.2.1 Attendre une certaine durée

```
#include <thread>
#include <chrono>
using namespace std::chrono;
using namespace std::this_thread;

sleep_for(seconds{2}); // attend 2 secondes
sleep_for(milliseconds{3}); // attend 3 millisecondes
```

17.2.2 Attendre jusqu'à une certaine date

```
#include <iostream>
#include <thread>
#include <chrono>
using namespace std::chrono;
using namespace std::this_thread;
using namespace std;
```

```
main ()
{
auto t1 = high_resolution_clock::now(); // date actuelle
cout<<" J'attends 3 secondes"<<endl;
auto t2 = t1 + seconds {3}; // date future
sleep_until(t2); // attente
auto t3 = high_resolution_clock::now(); // date mesurée
auto dt = duration_cast<duration<double>>(t3-t1).count(); // mesure
dt = t3-t1
cout<<"La durée a été dt = "<<dt<<" s."<<endl;
}
```

On obtient :

```
J'attends 3 secondes
La durée a été dt = 3.00016 s.
```

Chapitre 18

Intégrer des équations différentielles ordinaires (EDO) avec la librairie ODEINT

On propose d'utiliser la librairie **odeint** de boost. Voir aussi la documentation ODEINT.

Introduction : Une équation différentielle ordinaire (E.D.O.) est une équation de la forme :

$$\frac{dX}{dt} = F(X, t)$$

où $X(t) \in \mathbb{R}^n$ est un vecteur qui dépend du temps $t \in \mathbb{R}$, et $F : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ est une fonction connue. On connait $X(0)$ et F l'on cherche $X(t)$ pour d'autres valeurs de $t \in \mathbb{R}$. Le théorème de Cauchy Lipchitz garantit l'existence et l'unicité de $X(t)$ mais on a rarement une expression explicite de la solution. L'ordinateur est utile pour trouver des valeurs approchées de $X(t)$.

Exemple Pour intégrer avec la methode "runge_kutta_dopri5 with standard error bounds 10-6 for the steps", le champ de vecteur dans $X(t) = (x_0(t), x_1(t), x_2(t)) \in \mathbb{R}^3$ du système de Lorenz défini par :

$$\begin{aligned}
\frac{dx_0}{dt} &= \sigma \cdot (x_1 - x_0) \\
\frac{dx_1}{dt} &= Rx_0 - x_1 - x_0 x_2 \\
\frac{dx_2}{dt} &= -bx_2 + x_0 x_1
\end{aligned}$$

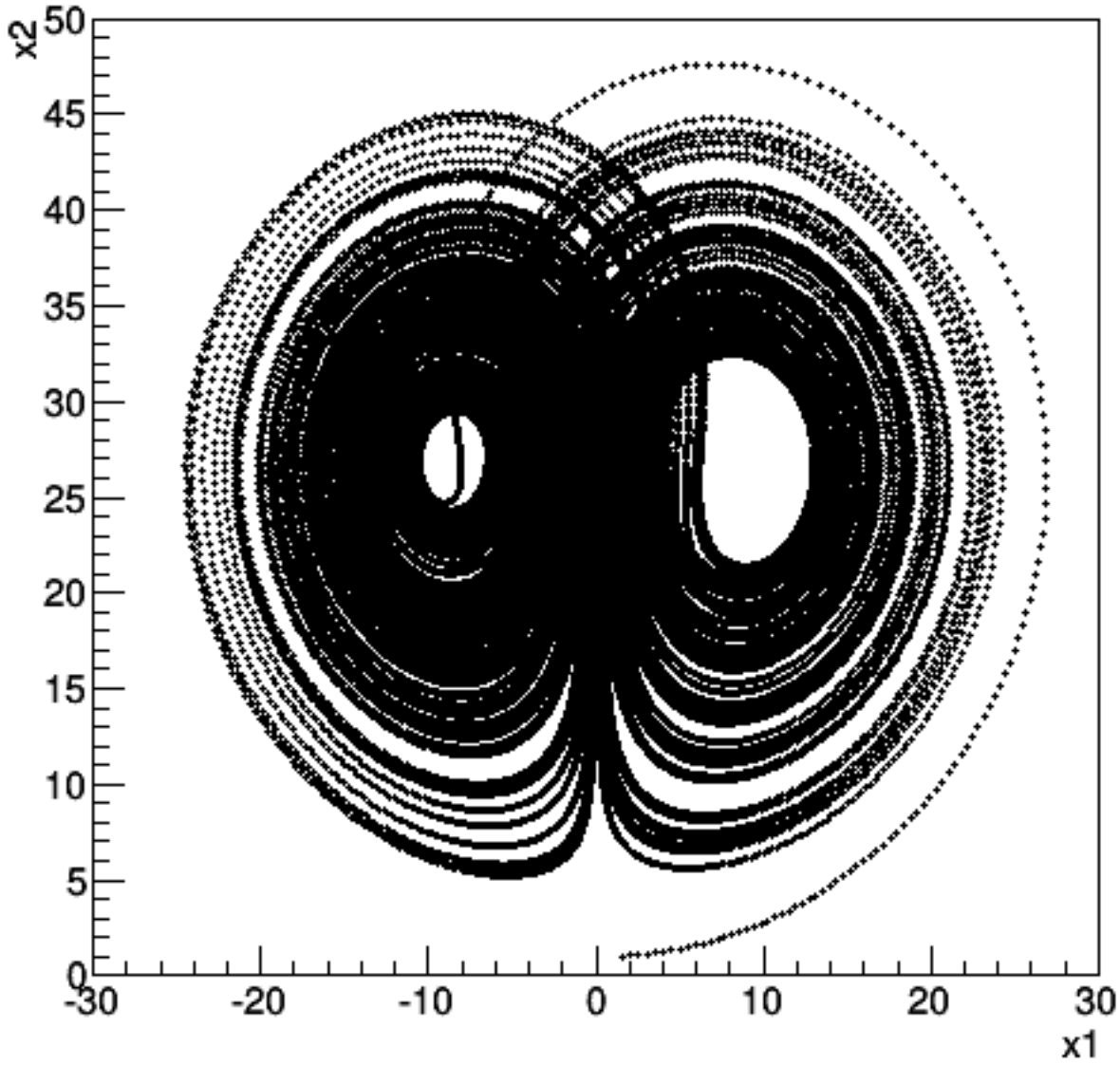
avec les paramètres : $\sigma = 10$, $R = 28$, $b = 8./3.$, on écrit :

```
#include <iostream>
#include <boost/array.hpp>
#include <boost/numeric/odeint.hpp>
```

```
   using namespace std;
   using namespace boost::numeric::odeint;
   const double sigma = 10.0;
   const double R = 28.0;
   const double b = 8.0 / 3.0;
   const int N=3 ; //nbre de variables de l'EDO
   ///-- definition de la fonction à intégrer---------
   void lorenz( const array<double,N> &x , array<double,N> &dxdt , double t
)
   {
   dxdt[0] = sigma * ( x[1] - x[0] );
    dxdt[1] = R * x[0] - x[1] - x[0] * x[2];
    dxdt[2] = -b * x[2] + x[0] * x[1];
   }
   ///----------
   int main()
   {
   array<double,N> x = { 10.0 , 1.0 , 1.0 }; // conditions initiales
    double t1 =0 ,t2=25; // temps initial et final
    double dt =0.1; //pas de temps
   for(double t=t1; t<=t2; t=t+dt)
    {
    integrate( lorenz , x , t, t+dt, dt); // intégre entre t et t+dt
   cout << t << '\t' << x[0] << '\t' << x[1] << '\t' << x[2] << endl;
   }
   }
```

Remarque 18.0.1. Pour plus d'options ou autres méthodes d'intégration voir la documentation.

Exercice 18.0.2. Modifier le code ci-dessus pour dessiner la trajectoire dans le plan x_1, x_2 avec un point tous les pas de temps $dt = 0.02$ et $t = 0 \to 100$ et obtenir l'image suivante :



Solution :

```
#include <iostream>
#include <boost/array.hpp>
#include <boost/numeric/odeint.hpp>
#include <TApplication.h>
#include <TCanvas.h>
#include <TMarker.h>
#include <TH1F.h>
```

```
using namespace std;
using namespace boost::numeric::odeint;



const double sigma = 10.0;
const double R = 28.0;
const double b = 8.0 / 3.0;
                const int N=3; //nbre de variables de l'EDO


///- definition de la fonction à intégrer

void lorenz( const array<double,N> &x , array<double,N> &dxdt , double t )

{
    dxdt[0] = sigma * ( x[1] - x[0] );
    dxdt[1] = R * x[0] - x[1] - x[0] * x[2];
    dxdt[2] = -b * x[2] + x[0] * x[1];
}



///

int main()

{


    TApplication theApp("App", nullptr, nullptr);
    TCanvas *c = new TCanvas( "c","fenetre",500,500); // on precise la taille

    double xmin(-30),ymin(0),xmax(30),ymax(50);
    TH1F *h=c->DrawFrame(xmin,ymin,xmax,ymax); // coordonnees de la fenetre (
    h->SetXTitle("x1");
    h->SetYTitle("x2");

    int cpt=0;

    array<double,N> x = { 10.0 , 1.0 , 1.0 }; // conditions initiales
    double t1 =0 ,t2=100; // temps initial et final
    double dt =0.002; //pas de temps
```

```
    for(double t=t1; t<=t2; t=t+dt)

    {
        integrate( lorenz , x , t, t+dt, dt); // integre entre t et t+dt
        //  cout << t << '\t' << x[0] << '\t' << x[1] << '\t' << x[2] << endl
        cpt++;


        TMarker *p=new TMarker(x[1],x[2],6);
        p->Draw();

        TMarker p2(x[1],x[2],8);
        p2.SetMarkerColor(kRed);
        p2.SetMarkerSize(2);
        p2.Draw();

        if(cpt%10==0)
            c->Update();

    }

    c->Update();
    theApp.Run();


}
```

Chapitre 19

Lecture et écriture d'un fichier son (audio) de format WAV

19.0.0.1 Introduction

— Un **signal audio** est une fonction $t \in \mathbb{R} \to s(t) \in \mathbb{R}$ qui représente les variations de pression sonores en fonction du temps t.

— Un **fichier audio** au format WAV représente le signal audio pour des valeurs discrètes du temps :

$$t_j = j.\delta t, \quad j = 1, 2, \ldots$$

avec un pas de temps δt^{-1} appelée fréquence d'échantillonage. Les valeurs standard sont : $\delta t^{-1} = 8000Hz$, $\delta t^{-1} = 44100Hz$, $\delta t^{-1} = 48000Hz$, etc. Chaque valeur $s(t_j)$ est codée sur 8 bits ou 16 bits. Le nombre de caneaux 1 ou 2 indique si le fichier peut contient un (mode mono) ou deux signaux (mode stéréo). Le fichier WAV contient les valeurs $s(t_j)$ à la suite, mais au début du fichier il y a les paramètres du codage. Regarder le code source pour plus d'informations.

19.0.0.2 Travail préalable pour le projet C++

1. Fichiers à télécharger et sauver dans son répertoire : bib_fred.h et bib_fred.cc.
2. Dans codeblocks, il faut ajouter ces fichiers au projet en faisant dans le menu : Projet/add_files ...
3. Il faut aussi **rajouter** `-larmadillo` dans les options de compilation (pour codeblocks c'est dans Setting/Compiler/LinkerSettings/OtherLinkerOptions)

19.0.1 Programme pour écrire un fichier WAV :

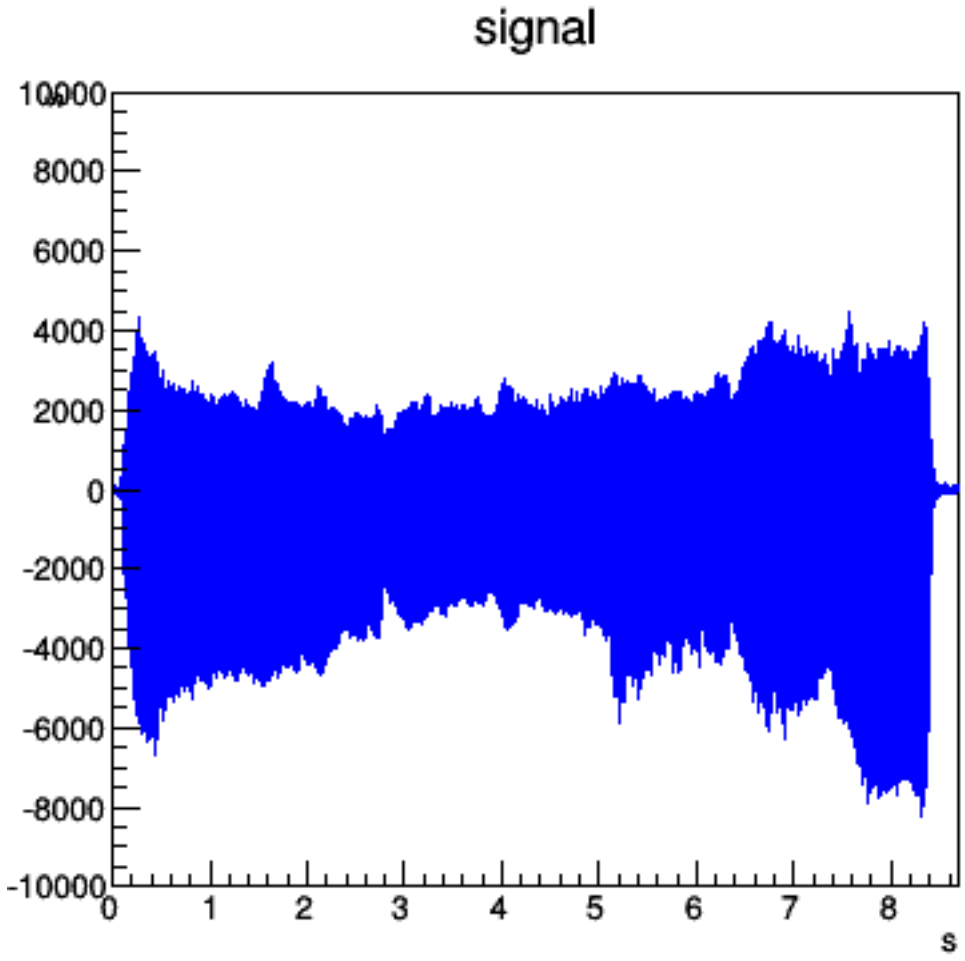
Pour tester, télécharger ce fichier Voyelles_Par_Malik.wav et sauvegarder le dans le répertoire de votre projet.

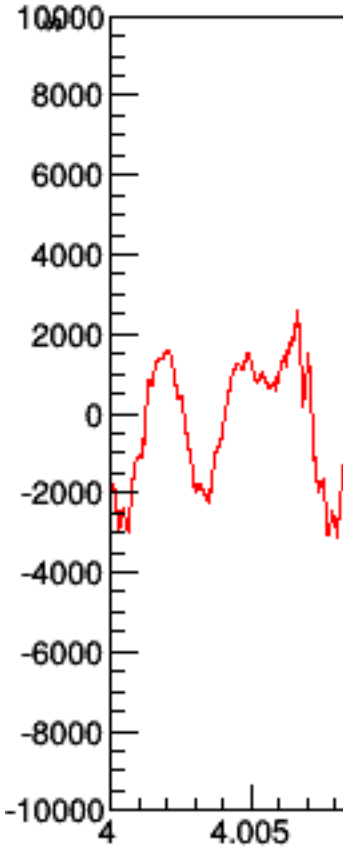
Le programme suivant permet

— de lire et afficher les paramètres du codage audio de ce fichier sonore $s(t_j) \in \mathbb{R}$ (nombre de canaux, taux d'echantillonage etc..),
— de mettre les amplitudes $s(t_j)$ dans un vecteur de nombres
— de dessiner ce vecteur avec la librairie ROOT.
— De plus il lance le logiciel **vlc** qui permet d'entendre le fichier (il envoie le fichier sur la carte son).

```
#include <iostream>
using namespace std;
#include <TCanvas.h>
#include <TMarker.h>
#include <TApplication.h>
#include "bib_fred.cc"
///-------------------------------------------
int main()
{
TApplication theApp("App", nullptr, nullptr);

//---- lecture audio du fichier wav avec le logiciel vlc
system("vlc Voyelles_Par_Malik.wav");

//--- lecture de tout le fichier
vec signal;
double f=Lecture_fichier_wav("Voyelles_Par_Malik.wav", signal);
// transfert les données du fichier dans le vecteur
double dt=1./f; // pas en temps
//Dessin(signal, "signal");
Dessin(signal,"s","s",0,signal.size()*dt,"signal",0,"L",-1e4,1e4,kBlue,0);

//--- lecture d'une partie
vec signal2;
double t1=4, t2=4.03; // intervalle de temps en secondes
f=Lecture_fichier_wav("Voyelles_Par_Malik.wav", signal2, 1,t1,t2);
dt=1./f;
//Dessin(signal2, "s2");
Dessin(signal2,"s","s",t1,t1+signal2.size()*dt,"signal2",0,"L",-1e4,1e4,kRed,0);

//---------
theApp.Run();
}
```

Sortie du programme :





Remarquer que vous pouvez zoomer le dessin etc...
Sortie du programme sur le terminal :

```
=====Crée tableau à partir du fichier: Voyelles_Par_Malik.wav
ChunkID=RIFF
format=WAVE
Nombre de canaux=1
Frequence d'echantillonage=48000 Hz
Bits par echantillon=16
Nombre de données =834402
Nombre d'octets par echantillon=2
Pas de temps dt =2.08333e-05 s.
Durée T =8.69169 s.
Lecture fichier finie.
--------------------
```

19.0.2 Programme pour lire un fichier WAV :

Le programme suivant crée un signal échantilloné dans un vecteur de double (de type vec) et ensuite écrit ce vecteur dans un fichier au format WAV. De plus il lance le logiciel **vlc** qui permet de jouer le fichier (envoie sur la carte son).

```
#include <iostream>
using namespace std;
#include <TCanvas.h>
#include <TMarker.h>
#include <TApplication.h>
#include "bib_fred.cc"
///-----------------------------------------
int main()
{
TApplication theApp("App", nullptr, nullptr);
double dt=1./48000; // pas de temps en s.
double T=5;// durée en s.
int N=T/dt;
vec signal(N);
double f0=440; // frequence de la note que l'on veut en Hz

for(int i=0; i<N; i++)
{
double t=i*dt;
double f= f0 *(1+0.01*cos(2.*M_PI*t/1.)); // modulation de frequences
double s= sin( 2*M_PI * f * t ); // note pure de frequence f
s = s + 0.5 * sin( 2*M_PI * 2*f * t ); // rajoute harmonique 2
s = s + 0.3 * sin( 2*M_PI * 3*f * t ); // rajoute harmonique 3
signal(i) = s;
}

Dessin(signal,"s","s",0,signal.size()*dt,"signal",0,"L",-4,4,kBlue,0);

// ecrit le vecteur signal dans le fichier son.wav. On indique dt
Ecriture_fichier_wav(signal, dt,"son.wav");

//---- lecture audio du fichier wav avec le logiciel vlc
system("vlc son.wav");

//---------
theApp.Run();
}
```

Sortie du programme : ce fichier son.wav, l'image du signal et la sortie sur terminal :

```
=====Crée un fichier wav à partir du tableau ====fichier:son.wav
Nombre de canaux=1
Frequence d'echantillonage=48000 Hz
Bits par echantillon=16
Nombre d'octets par echantillon=2
Nombre d' echantillons =240000
Pas de temps dt =2.08333e-05 s.
Durée T =5 s.
```

Chapitre 20

Gestion de la carte son (audio temps réel) avec la librairie sndio

On utilisera la librairie sndio développée pour OpenBSD mais utilisable aussi avec linux. Cette librairie permet de gérer le son audio et aussi les évènements midi en temps réel (entrée/sortie).

Remarques

— Pour avoir la liste des cartes sons faire `cat /proc/asound/cards` ou `aplay -l`, `cela donne par exemple`

```
0 [HDMI ] : HDA-Intel - HDA Intel HDMI
 HDA Intel HDMI at 0xf7a1c000 irq 38
1 [PCH ] : HDA-Intel - HDA Intel PCH
 HDA Intel PCH at 0xf7a18000 irq 37
```

20.1 Installation (à faire la première fois)

— Si vous êtes administrateur, suivre cette documentation. Puis tester :
  — Pour envoyer (écouter) le fichier `test.wav` sur la carte 0 (carte son par default) par exemple il faut écrire : `aucat -i ~/test.wav`.
— Si il n'y a pas de son et le message "snd0: rsnd/0: failed to open audio device" c'est que la carte son 0 n'existe pas. Vérifier la liste des cartes sons de votre ordi avec la commande `aplay -l` et :
  — Pour envoyer (écouter) le fichier `test.wav` sur la carte 1 par exemple il faut écrire : `aucat -f rsnd/1 -i ~/test.wav`.
  — Si la carte 1 est la carte son par defaut (au lieu de la carte 0) il faut éditer `/etc/default/sndiod` et ajouter l'option `DAEMON_OPTS="-f rsnd/1"`. Voir remarques de configuration ci-dessous pour plus d'options.

- Sinon si vous n'êtes pas administrateur et que cette librairie n'est pas déjà installée, il vous faut l'installer sur votre compte. Pour cela :
  - Télécharger le source et décompresser l'archive.
  - Dans le répertoire de sndio ainsi créé, taper :
    - `configure --prefix=$HOME`
    - `make`
    - `make install`
  - Remarque : cela a pour effet de compiler la bibliothèque et d'installer les librairies dans HOME/lib et HOME/include.
- Dans les options de compilation, il faudra rajouter :
  - `-I/$HOME/include` (: dans codeblocks, c'est à rajouter dans Settings/Compiler/CompilerSetting/OtherOptions),
  - `-L/$HOME/lib` et `-lsndio` (: dans codeblocks, c'est à rajouter dans Settings/-Compiler/LinkerSettings/OtherOptions)
- Dans le fichier `.bashrc` il faut rajouter `export LD_LIBRARY_PATH=$HOME/lib` et executer dans une fenêtre de commande : `. .bashrc`

Autres options de configuration de sndio : Dans le fichier /etc/default/sndio on peut ecrire par exemple :

```
DAEMON_OPTS="-z 128 -q rmidi/0 -q rmidi/1 -q rmidi/2 -f rsnd/1 -s default
-m play,mon -s fred"
```

puis faire

```
service sndiod enable
service sndiod start
```

Rappel : `amidi -l` : enumere les ports midi utilises par les appareils"

`aplay -l` : detecte les cartes son -> numero 1"

Ces options signifient :

- -q : signifie configurer & exposer ce port midi pour que plusieurs programmes puissent l'utiliser"
- -z latence
- -f rsnd/1 : demande utiliser carte son numero 1. L'ordre est important. plus d'info avec : xterm -e sudo pasuspender -- SNDIO_DEBUG=2 etc..
- -m play,mon -s fred : signifie redirige la sortie audio vers l'entree. Cela permet d'enregistrer ce qui est joué. Il y a une association entre les connecteurs entree et en sortie.
- -s default :permet de garder la configuration par default, cad utiliser aussi le micro. le choix du device se fait dans le programme qui utlise sndio.

20.1.0.1 Travail préalable pour le projet C++

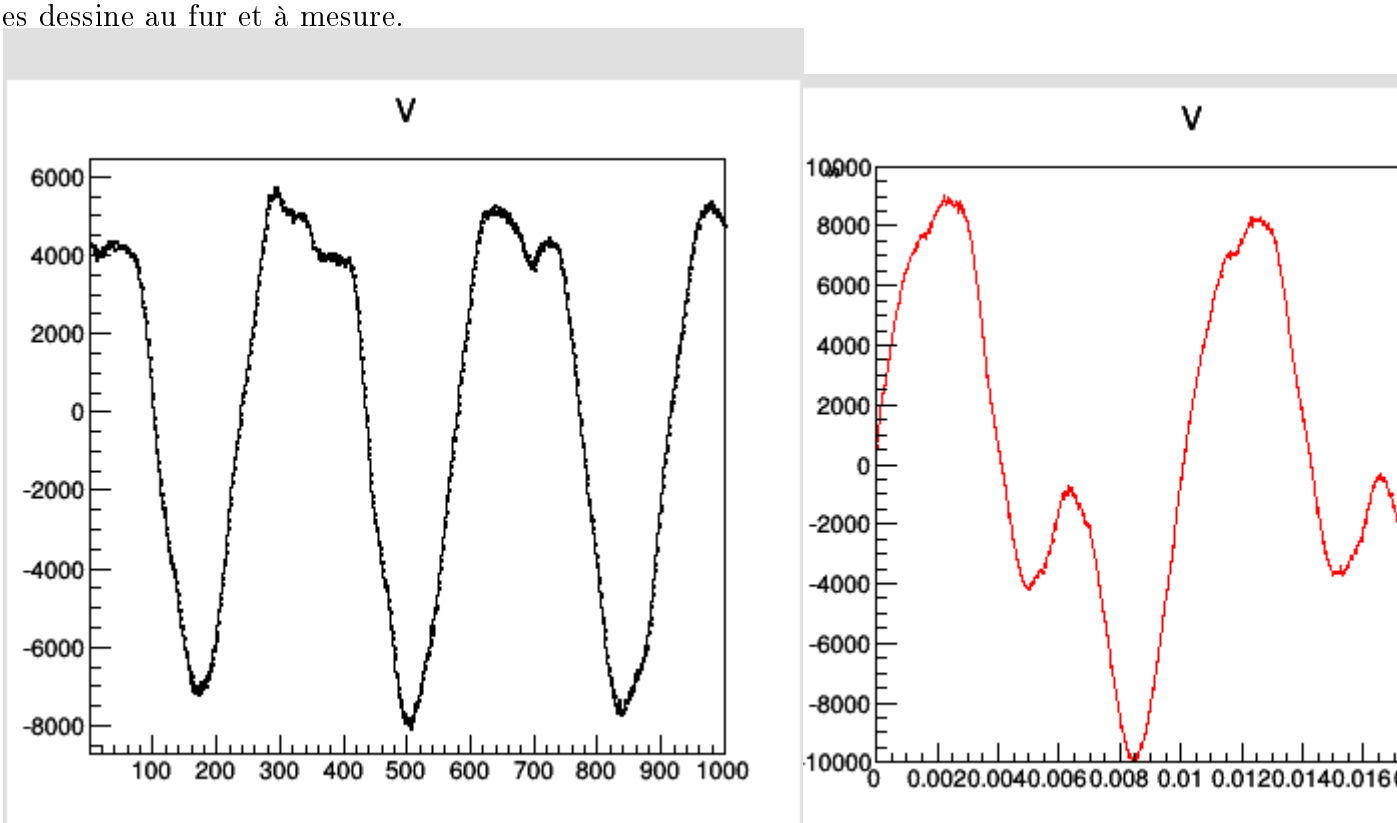
1. Fichiers à télécharger et sauver dans son répertoire : bib_fred.h et bib_fred.cc, audio.h et audio.cc.

2. Dans codeblocks, il faut ajouter ces fichiers au projet en faisant dans le menu : Projet/add_files ...
3. Il faut aussi **rajouter** `-larmadillo` dans les options de compilation (pour codeblocks c'est dans Setting/Compiler/LinkerSettings/OtherLinkerOptions)
4. Démarrage du serveur sndiod, si vous êtes administrateur, écrire dans un terminal : `sudo sndiod -dd -z480 -f rsnd/0` et le laisser ouvert,
   sinon, si sndio a été installé sur votre compte, écrire dans un terminal : `sndiod -dd -z480 -f rsnd/0` et le laisser ouvert.

20.2 Utilisation du micro (entrée) et du haut-parleur (sortie)

20.2.1 Le micro

Le micro échantillonne le son avec un pas de temps $Dt = 1/48000sec.$ et envoie les données à la carte son qui les convertit sous forme numérique. Cet exemple lit les données du micro depuis la carte son par bloc de $N = 1000$ donnée, soit de durée $T = N.Dt = 0.02$sec., et les dessine au fur et à mesure.




```
#include "bib_fred.cc"
```

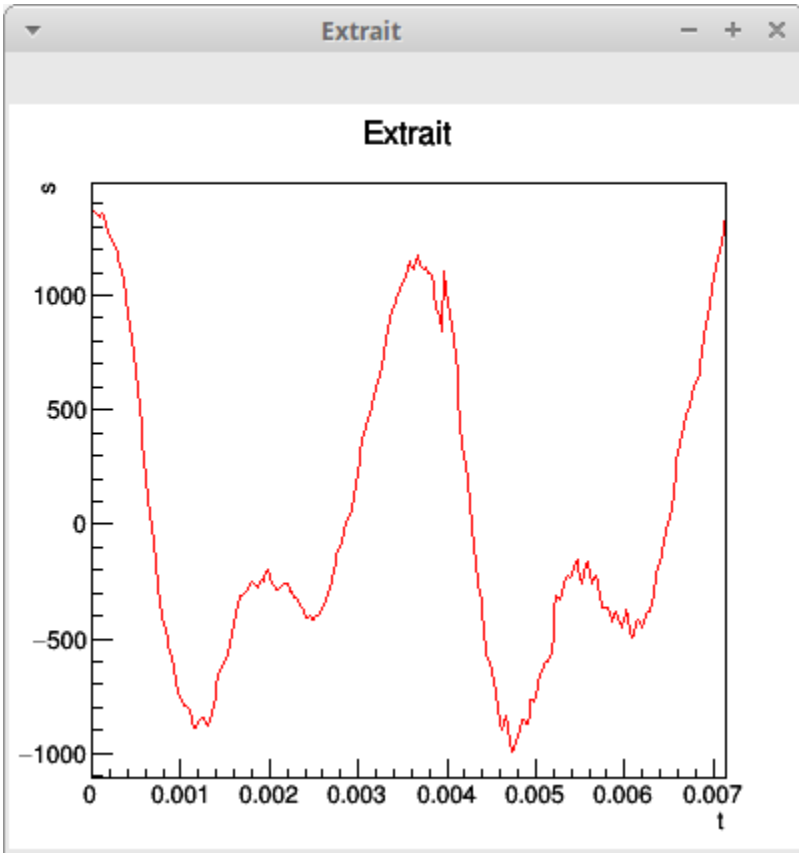
```
#include "audio.cc"
#include <iostream>
using namespace std;
#include <TROOT.h>
#include <TApplication.h>

//===Main=============================================
int main()
{
  TApplication theApp("App", nullptr, nullptr);
  Audio s;
  s.Initialise_audio(1); // initialise la carte son en entree (1)
et affiche ses informations
  int N=1000; // on donne la taille que l'on veut (nombre d'echantillons)
  vec V(N);

  while(1) // boucle infinie
  {
      s.Lit_donnees_audio(V); // remplit V depuis le micro
      Dessin(V,"V");
      // Dessin(V,"t","s",0,N*s.Dt,"V",0,"L",-1e4,1e4,kRed,0); //
dessin normalisé en rouge
  }

  /// ----Donne la main a l'utilisateur ---
  theApp.Run();

}
```

20.2.1.1 Exemple : détection de note de musique en temps réel

Une note de musique est un signal périodique de période T, et fréquence $f = 1/T$ dans l'intervalle audible $30Hz \leq f \leq 3000Hz$. Le programme suivant utilise un algorithme de détection de période dans un signal, et affiche le signal sur une (ou plusieurs périodes) ainsi que le nom de la note jouée. (Pour information, la fonction utilisée `Signal::Detecte_Periode()` se trouve dans le fichier `signal.cc`.)

Résultat :

```
Periode : T=0.0034036s. <-> Frequence f=1/T=293.806 Hz
Note =Re, Ecart = 0.00833494 demitons, Octave=0.
```




```
#include "bib_fred.cc"
#include "audio.cc"
#include <iostream>
using namespace std;


#include <TROOT.h>
#include <TApplication.h>

string Notes[12] = {"Do","Do#", "Re","Re#", "Mi", "Fa", "Fa#", "Sol", "Sol#",

//===Main=========================================================

int main()
{
    TApplication theApp("App", nullptr, nullptr);

    Audio s;
    s.Initialise_audio(1); // initialise la carte son en entree (1) et affich
```

```
    int N=2000; // on donne la taille que l'on veut (determine la fenetre tem
    vec V(N);

    //parametres pour la detection de la periode
    int NP=2; // 1,2: nbre de periodes à afficher
    double f_A = 440; // diapason
    s.seuil_norme_par_I=1e4;
    s.seuil_C =0.1;
    s.verbose =0; //1 affiche messages erreur

    //-----------------------
    while(1) // boucle infinie
    {
        s.Lit_donnees_audio(V); // remplit V  depuis le micro

        double t0 =0; // date de debut d'analyse
        int opt_dessin_C=0; // 0 ou 1
        double T=s.Detecte_Periode(V, t0, opt_dessin_C); // -> Periode T ou 0
        if(T>0)
        {
            double f=1/T;
            cout<<" Periode : T="<<T<<"s. <->  Frequence f=1/T="<<f<<" Hz"<<e

            //--- conversion en note de musique
            double nu = 12*log(f/f_A)/log(2.) + 69;
            int nu_i = floor(nu+0.5);
            cout<<" Note ="<<Notes[nu_i%12]<<" Ecart = "<<nu-nu_i<<" demitons


            //---- affiche NP periodes extraites
            vec Extrait = V.rows( t0/s.Dt, (t0+NP*T)/s.Dt); // on extrait NP
            uword i = Extrait.index_max();
            Extrait =shift(Extrait, -i ); // place le max de l'extraitau debu
            Dessin(Extrait,"t","s",0,NP*T,"Extrait",0,"L",-1111,-1111,kRed,0)

        }

    }


/// ----Donne la main a l'utilisateur ---
    theApp.Run();
```

```
}
```

20.2.2 Le haut-parleur

Voici un exemple qui lit les données sur le micro et les envoie sur le haut-parleur après un délai de 200ms ("la latence"). Mettre un casque.

```
#include "bib_fred.cc"
#include "audio.cc"
#include <iostream>
using namespace std;


#include <TROOT.h>
#include <TApplication.h>


int main()
{
    TApplication theApp("App", nullptr, nullptr);

    Audio s;
    s.Initialise_audio(3); //  initialise la carte son en entree et sortie (3

    vec V;

    while(1) // boucle infinie
    {
        s.Lit_donnees_audio(V); // micro -> V
        Dessin(V,"t","s",0,V.size()*s.Dt,"V",0,"L",-1e4,1e4,kRed,0); // dessi
de V
        s.Ecrit_donnees_audio(V); // V -> Haut Parleur
    }

/// ----Donne la main a l'utilisateur ---
    theApp.Run();


}
```

Chapitre 21

Plusieurs programmes en parallèle qui communiquent, avec *"thread"*

Il peut être utile dans un projet de répartir le travail entre plusieurs programmes qui fonctionnent ensemble et communiquent. Un ordinateur est capable de gérer plusieurs programmes qui fonctionnent. Si l'ordinateur ne possède qu'un processeur, il va faire avancer chaque programme tour à tour en alternant sur des intervalles de temps très court. Cela donne l'impression que plusieurs programmes fonctionnent en parallèle. On va s'interesser à la communication entre ces programmes.

On peut imaginer cela comme des travailleurs qui fabriquent une maison. Chacun travaille sur sa partie mais il y a des moments où ils doivent échanger des informations, parfois un travailleur doit attendre qu'un autre ait fini son travail. Alors c'est le chef de projet (le processeur) qui se charge de superviser tout cela, de mettre en attente certains et réactiver d'autres.

Il y a donc trois concepts importants dont nous allons voir la mise en oeuvre :

1. Les threads : au niveau de la terminologie, chaque programme s'appelle un "thread" ou "process" ou "context". On va tout d'abord apprendre à **démarrer des threads**.
2. Les mutex : lorsque ces programmes doivent "communiquer" c'est à dire échanger des données (et c'est souvent le cas) il y a des précautions à prendre afin que tout ce passe bien : par exemple il faut éviter qu'une variable soit manipulée par deux threads en même temps. Ce sera le rôle des **mutex** qui sont simplement des indicateurs booleen (vrai/faux) que l'on interprète comme (occupé/libre).
3. Les variables conditionnelles, cond_var : c'est comme une sonnette. Cela permet au processeur de mettre en attente certains thread et d'en réactiver d'autres sous certaines conditions.

références : article sur C++11 threads, locks and condition variables.

21.1 Lancement de threads ou processus

```
#include <iostream> // std::cout
using namespace std;
#include <thread>

//---- Une petite fonction qui affiche des ‘‘a’’----------------
void f(int n)
{
for(int i=0;i<n;i++)
  cout<<"a"<<flush;
cout<<endl;
}

//----Fonction principale ----------------
int main()
{ // ici la fonction main est lancée
int n=10;
thread t(f,n); // lance la fonction f(n) dans un thread

cout << "Les fonctions main et f sont maintenant en concurrence.\n";
for(int i=0;i<n;i++) // affiche des ‘‘b’’
  cout<<"b"<<flush;
cout<<endl;

t.join(); // attend ici que la fonction f soit finie
cout << "f est finie.\n";
}
```

21.1.0.1 Commande de compilation :

- Si on utilise codeblocks, il faut rajouter `-pthread -std=c++11` dans Settings/Compiler/Compiler_Settings/Other_Options et dans Settings/Compiler/Linker_Settings/Other_linke
- Pour une compilation en ligne de commande :
  `g++ main.cpp -o main.out -pthread -std=c++11`

Résultat :

```
Les fonctions main et f sont maintenant en concurrence.
bbbbbaaaaaaaaaabbbbb
f est finie.
```

21.1.0.2 Commentaires

1. Les fonctions `f` et `main` fonctionnent en parallèle et leur affichages 'a' ou 'b' sur l'écran se mélangent de façon incontrollée. Un deuxième lancement du programme donnerait éventuellement autre chose.
2. On peut passer (ou pas) des variables à la fonction lancée en parallèle, ici on passe la variable n.
   (a) On peut **passer un pointeur** (adresse d'une variable) à la fonction `f` en écrivant par exemple, `void f(int * pn) {..}` pour la déclaration et `thread t(f,&n);` pour le lancement.. Cela permet de transmettre un résultat de la fonction `f` vers la fonction `main`.
   (b) Mais pour **passer une variable par référence**, il faut écrire par exemple, `void f(int & n) {..}` pour la déclaration et `thread t(f,ref(n));` pour le lancement.
3. Il peut être utile de créer une certaine attente dans un programme. Utiliser Section 17.2.
4. Si vous oubliez `t.join()` à la fin, il y aura une erreur à l'execution du programme.
5. Documentation complète sur thread.

21.1.0.3 Lancer un thread d'une fonction membre de classe

```
#include <iostream>
using namespace std;
#include <thread>

class A
{
 public:
 void f(int n ) { cout << n << endl; }
};

int main()
{
 A a;
 thread t1(&A::f, &a, 100);
 t1.join();

}
```

Commentaires

— Si on lance le thread depuis une autre fonction membre, il faut bien sûr remplacer le pointeur `&a` par `this`.

21.1.0.4 Lancer une liste de thread (tableau)

```
#include <iostream> // std::cout
using namespace std;
#include <thread>
//---- Une petite fonction qui affiche n fois n ----------------
void f(int n)
{
 for(int i=0; i<n; i++)
 cout<<n<<flush;
}
//----Fonction principale ----------------
int main()
{
int n=7;
thread L_t[n]; // tableau de threads
for (int i=0; i<n; i++)
 L_t[i] = thread(f,i); // lance de thread i qui affiche i fois i
for(int i=0; i<n; i++)
 L_t[i].join(); // attend ici que le thread i soit fini
}
```

Résultat : 315524563656666254443

21.1.0.5 Lancer une liste de thread (vector)

Même chose que précédement mais avec un **vector**.

```
#include <iostream> // std::cout
using namespace std;
#include <thread>
//---- Une petite fonction qui affiche n fois n ----------------
void f(int n)
{
 for(int i=0; i<n; i++)
 cout<<n<<flush;
}
//----Fonction principale ----------------
int main()
{
```

```
int n=7;
vector<thread> L_t; // tableau de threads
for (int i=0; i<n; i++)
 L_t.push_back(thread(f,i)); // lance de thread i qui affiche i fois i
for(int i=0; i<n; i++)
 L_t[i].join(); // attend ici que le thread i soit fini
}
```

Résultat : 213332444456666665555

21.2 Mutex pour annoncer occupé/libre

21.2.0.1 Introduction

Un mutex est simplement une variable booléenne dont on pense que la valeur est "occupé/libre". Une analogie possible est qu'un mutex est comme le panneau devant une salle de bain (sans serrure) qui annonce une valeur "occupé/libre". Par exemple, on peut utiliser un mutex avant de modifier une variable commune à plusieurs threads. On déclare un mutex par

```
mutex mtx;
```

Ensuite il y a deux commandes de base :

1. La commande
   `mtx.lock();`
   qui signifie : si `mtx` est "libre" on le met à "occupé" et on continue, sinon, si il est "occupé" alors on attend jusqu'à ce qu'il soit libre.
2. La commande
   `mtx.unlock();`
   qui signifie : on met `mtx` à "libre"

21.2.0.2 Remarques

— Quel intéret d'utiliser un mutex `mtx` plutot qu'une variable boolenne `mtx` ordinaire pour effectuer la suite d'instructions "`si mtx est "libre" on le met à "occupé" et on continue, sinon on attend`" ?
réponse : en utilisant une variable ordinaire, il y a le risque qu'un autre processus viennent modifier la valeur de la variable pendant cette suite d'instructions, ce que l'on ne souhaite absolument pas.En utilisant un mutex il est garantit que cette suite d'instruction est faite tout à la suite par l'ordinateur, sans être interrompu (on dit de façon "atomique").

21.2.0.3 Exemple

```
#include <iostream> // std::cout
using namespace std;
#include <thread>
#include <mutex>

mutex mtx;

//---------------------
void f(int n,char c)
{
mtx.lock(); // si mtx est libre on le bloque, sinon on attend

for(int i=0;i<n;i++)
  cout<<c<<flush;
cout<<endl;

mtx.unlock(); // débloque mtx
}

//---------------------
int main()
{
int n=10;
thread t1(f,n,'a');
thread t2(f,n,'b');

t1.join();
t2.join();

}
```

21.2.0.4 Résultat :

```
aaaaaaaaaa
bbbbbbbbbb
```

Dans cet exemple le thread t1 est arrivé le premier et a bloqué le mutex. Le thread t2 a donc attendu qu'il soit débloqué pour le bloquer à son tour.

21.3 Communications : variables conditionnelles pour réveiller un programme en attente

Introduction Dans la communication entre programmes (process), il peut arriver qu'un programme 2 attende le resultat d'un autre(s) programme(s) 1.

Voici une solution naive et maladroite pour programmer cette solution :

— Il y a une variable qui est a=0 au départ et que le programme met à la valeur a=1 dès qu'il a obtenu sont résultat. (Cette variable est protégée par un mutex). Le programme 2 effectue une boucle infinie dans laquelle il lit la variable a et si a==1 alors il sort de la boucle et récupère le résultat. Cette solution est maladroite car le programme 2 consomme du CPU (processeur de la machine) pour rien.

Une meilleure solution est que le programme 2 soit mis en attente (en veille, il ne consomme pas de CPU) et réveillé à l'arrivée du résultat. Pour cela on utilise une **variable conditionelle**.

Instructions de base : Une variable conditionnelle se déclare par

`condition_variable cv;`

Il y a la commande

`cv.wait(lck);`

qui met le thread en attente (endormi), et

`cv.notify_one();`

qui reveille un parmi les autres thread qui se sont mis en attente, (ou `cv.notify_all();` qui reveille tous les autres thread qui se sont mis en attente.)

21.3.0.1 Exemple de base où un thread endormi est reveillé par un autre.

```
#include <iostream>
using namespace std;
#include <thread>
#include <chrono>
#include <mutex>
#include <condition_variable>

mutex mtx;
condition_variable cv;

//--------------------
void f()
{
unique_lock<mutex> lck(mtx); // bloque le mutex mtx. Il sera debloque
automatiquement par wait(lck), ou a la destruction de lck.
```

```
   cv.wait(lck); // met le thread en attente (endormi). Il sera réveillé
par un signal extérieur.
   cout<<"FIN"<<endl;
   }

   //--------------------
   int main()
   {
   thread t(f);

   cout<<"On attend 2sec."<<endl;
   this_thread::sleep_for(chrono::seconds{2});
   cout<<"voilà."<<endl;

   // ... reveille cv
   cv.notify_one(); // reveille le thread endormi

   //.........
   t.join();
   }
```

Résultat :

```
On attend 2sec.
voilà.
FIN
```

21.3.0.2 Remarques

— L'instruction wait() sauvegarde tous les registres du processus (tout son environnement mémoire) et notify_one() les restore.

— dans cet exemple l'utilisation du mutex ne sert à rien, mais le langage C++ oblige à l'utiliser dans l'instruction wait(lck). Voir ci-dessous l'exemple 21.3.4 où l'utilisation du mutex est utile et obligatoire. Pourquoi faut il utiliser le mutex `mtx` ? L'utilisation d'un mutex est obligatoire, car les thread se partagent une variable `cv` qui spécifie l'état sommeil/reveillé et donner l'instruction de réveil. La fonction wait() utilise cette variable en faisant en fait de façon séquentielle : lecture de cv, si signal je dois me réveiller.

Exemple 21.3.1. "où un thread endormi attend des évènements provenant de differents thread extérieurs".

```
#include <iostream>
using namespace std;
```

```
#include <thread>
#include <chrono>
#include <mutex>
#include <condition_variable>
mutex mtx;
condition_variable cv;
//-------
void f1()
{
 while(1)
 {
this_thread::sleep_for(chrono::milliseconds {1000});
cout<<"1"<<flush;
 cv.notify_one(); // reveille le thread main() endormi
 }
}
//-------
void f2()
{
while(1)
{
this_thread::sleep_for(chrono::milliseconds {1502});
cout<<"2"<<flush;
cv.notify_one(); // reveille les thread main() endormi
}
}
//-------
int main()
{
thread t1(f1);
thread t2(f2);
unique_lock<mutex> lck(mtx); // bloque le mutex mtx. Il sera debloque
automatiquement par wait(lck), ou a la destruction de lck.

while(1)
{
cv.wait(lck); // met le thread main() en sommeil
cout<<","<<flush;
}
//.........
t1.join();
t2.join();
}
```

Résultat : 1,2,1,1,2,1,2,1,1,2,1,2,1,1,2,1,2,1,1,2,...

Remarque 21.3.2.

— Pour expliquer le résultat, il faut remarquer que la fonction f1() affiche "1" aux dates suivantes (en ms) : 1000, 2000, 3000, 4000, 5000,.... Après chaque affichage, la fonction f1() reveille la fonction main() qui affiche ",". De même, la fonction f2() affiche **"2"** aux dates : **1502, 3004, 4506, 6008**, ... et de même à chaque affichage elle reveille la fonction main() qui affiche ",". La suite chronologique de ces dates est : 1000,**1502**,2000,3000,**3004**,4000,**4506**,5000,6000,**6008**, ... ce qui explique le résultat 1,2,1,1,2,1,2,1,1,2,..

— Plusieurs threads peuvent être mis en attente par `cv.wait(lck);` et la commande `cv.notify_one();` va en reveillé un seul (on ne sait pas lequel). Par contre la commande `cv.notify_all();` reveillerait tous les threads mis en attente.

Exemple 21.3.3. "où deux thread endormi sont reveillés par une notification venant d'un 3eme thread".

```
#include <iostream>
using namespace std;
#include <thread>
#include <chrono>
#include <mutex>
#include <condition_variable>
mutex mtx;
condition_variable cv;
//-------
void f1()
{
unique_lock<mutex> lck(mtx); // bloque le mutex mtx. Il sera debloque
automatiquement par wait(lck), ou a la destruction de lck.
cout<<"a"<<endl;
cv.wait(lck);
cout<<"b"<<endl;
}
//-------
void f2()
{
unique_lock<mutex> lck(mtx); // bloque le mutex mtx. Il sera debloque
automatiquement par wait(lck), ou a la destruction de lck.
cout<<"A"<<endl;
cv.wait(lck);
cout<<"B"<<endl;
}
//-------
```

```
int main()
{
thread t1(f1);
thread t2(f2);
for(int t=3; t>=0; t--) // compte à rebour
{
this_thread::sleep_for(chrono::seconds {1});
cout<<t<<","<<flush; // \r permet de re-ecrire au debut de la ligne
}
cout<<"partez!"<<endl;
cv.notify_all();
//.........
t1.join();
t2.join();
}
```

Exemple 21.3.4. "Un thread remplit une liste. Le deuxieme thread vide la liste".

```
#include <iostream>
using namespace std;
#include <thread>
#include <chrono>
#include <mutex>
#include <condition_variable>
#include <list>
mutex mtx;
condition_variable cv;
list<int> L;
int N=10;
//----------------------
void f()
{
    while(1) // boucle infinie
    {
        {
            unique_lock<mutex> lck(mtx); // bloque le mutex mtx. Il sera debl
            if(L.size()< N) // si liste non pleine
                cv.wait(lck); // debloque le mutex lck et met le thread en at
        } // ici lck est detruit donc mtx est debloque
        mtx.lock();
        cout<<"Liste pleine. On vide la liste par le debut:"<<flush;
        while(L.size()>0)
        {
```

```
            cout<<L.front()<<","<<flush;
            this_thread::sleep_for(chrono::milliseconds {100});
            L.pop_front(); // enleve le premier element
        }
        cout<<endl;
        mtx.unlock();
        cv.notify_one(); // reveille le thread endormi
    }
}
//----------------------
int main()
{
    thread t(f); // lance la fonction f dans un thread
    while(1) // boucle infinie
    {
        mtx.lock(); // bloque le mutex
        if(L.size()==0) // si liste vide
        {
            cout<<"Liste Vide. Remplit la liste:"<<flush;
            for(int j=0; j< N; j++)
            {
                cout<<j<<","<<flush;
                L.push_back(j);
                this_thread::sleep_for(chrono::milliseconds {100});
            }
        cout<<endl;
        }
        mtx.unlock(); // debloque
        cv.notify_one(); // reveille le thread endormi
// se met en attente
        unique_lock<mutex> lck(mtx);
        cv.wait(lck);
    }
 //.........
    t.join();
    return 0;
}
```

Résultat :

```
Liste Vide. Remplit la liste:0,1,2,3,4,5,6,7,8,9,
Liste pleine. On vide la liste par le debut:0,1,2,3,4,5,6,7,8,9,
Liste Vide. Remplit la liste:0,1,2,3,4,5,6,7,8,9,
```

```
Liste pleine. On vide la liste par le debut:0,1,2,3,4,5,6,7,8,9,
...
```

Chapitre 22

Messages MIDI Musical temps réel avec la librairie sndio

La convention sur les messages MIDI (Musical Instrument Digital Interface) entre ordinateurs sont apparus dans les années 80 pour la **musique électronique, l'informatique musicale et les synthétiseurs**. Référence sur les messages MIDI.

Dans ce chapitre nous voyons comment envoyer et/ou recevoir des messages midi depuis un programme en utilisant la librairie sndio. On verra aussi comment utiliser ces messages midi en musique.

Configuration préalable

22.1 Installation (à faire la première fois)

— Suivre les instructions de la Section 20.1.

— De plus il faut télécharger les fichiers suivants et les sauver dans son répertoire : midi.h et midi.cc.

1. Si vous utilisez codeblocks, il faut ajouter ces fichiers au projet en faisant dans le menu : Projet/add_files ...
2. Si vous utilisez un Makefile, rajouter les fichiers au projet selon les explications de la Section 14.
3. Il faut aussi **rajouter** `-lsndio` dans les options de compilation (pour codeblocks c'est dans Setting/Compiler/LinkerSettings/OtherLinkerOptions)

22.2 Exemple élémentaire sur l'envoi et reception de messages MIDI

Ce premier exemple montre comment envoyer des octets (chiffres) entre deux programmes via un port midi virtuel de l'ordinateur. En pratique les messages MIDI ont

un contenu très particulier qui ont une signification musicale comme “début de telle note”, “changement de volume”,.., et dans la Section suivante on précisera comment utiliser ces messages pour l'informatique musicale (reception des notes d'un clavier numérique et envoie de notes à un synthétiseur).

Instructions

- Copier et compiler les programmes `test_midi_in.cc` et `test_midi_out.cc` avec le programme Makefile donnés ci-dessous.
- Lancer le programme `test_midi_in` dans un terminal puis le programme `test_midi_out` dans un autre terminal.

Résultat :

- Le programme `test_midi_in` se met en attente et lorsqu'il recoit un message midi (provenant du programme `test_midi_out`) il l'affiche à l'écran :

```
port ouvert.
Attente de message ..
1 messages recus.
Message recu sur le port=0: 144, 60, 50
Midi::On ferme les ports midi..
```

- Le programme `test_midi_out` affiche :

```
port ouvert.
J'ai envoye le message 144, 60, 50 sur le port 0
Midi::On ferme les ports midi..
```

Code C++ des programmes : Voici le code C++ ainsi que le Makefile permet de compiler les deux programmes par l'instruction `make all`.

```
//===== Programme test_midi_in.cc ==========
#include <iostream>

using namespace std;
#include "midi.h"


int main()
{

        Midi midi;


        //--- ouvre un port midi en entree
        int port=0; // numero de port
```

```
        int     res = midi.Initialise_port_midi_in(port);
        if(res==0)
                cout<<"port est encore ferme."<<endl;
        else
                cout<<"port ouvert."<<endl;


        //--- attente de messages
        cout<<"Attente de message .."<<endl;
        midi.Attente_Message();
        cout<<"Messages recus."<<endl;

        //--- on affiche les messages recus dans vector<vector<unsigned char>
        for(auto message : midi.L_message) // analyse chacun des messages mid
        {
                int port_in=(int)message[0]; // numero de port entrant
                message.erase(message.begin()); // enleve le 1er
octet. Il reste le message midi standard

                cout<<"Message recu sur le port="<<port_in<<": "<<flush;

                for(int i=0;i<message.size();i++)
                        cout<<(int)message[i]<<", "<<flush;
                cout<<endl;
        }


//--- ferme port midi
        midi.Ferme_port_midi_in(port);



}

//===== Programme test_midi_out.cc ==========
#include <iostream>

using namespace std;
#include "midi.h"


int main()
```

```
{

        Midi midi;

        //--- ouvre port midi out
        int port = 0;

        int res =  midi.Initialise_port_midi_out(port);
        if(res==0)
        {
                cout<<"port est encore ferme."<<endl;
                exit(1);
        }
        else
                cout<<"port ouvert."<<endl;


        //---- envoie message: 144, 60, 50

        vector<unsigned char> message;
        message.push_back(144);
        message.push_back(60);
    message.push_back(50);
        midi.Envoi_Message(port, message);

    cout<<"J'ai envoye le message 144,60,50 sur le port "<<port<<endl;

    //-- ferme les ports midi ouverts
        midi.Ferme_port_midi_out(port);



}

#=== Fichier Makefile
LIBR= -lm  -lpthread -lsndio  -std=c++11

CC =g++  -std=c++11


#=============================================
OBJETS_MIDI_O=  test_midi_out.o midi.o
```

```
midi.o : midi.cc midi.h
	$(CC) $(CFLAGS) -c $*.cc -o $@


test_midi_out.o : test_midi_out.cc bib_fred.h
	$(CC) $(CFLAGS) -c $*.cc -o $@


test_midi_out: $(OBJETS_MIDI_O)
	$(CC) -o test_midi_out $(OBJETS_MIDI_O) $(LIBR)


#=============================================
OBJETS_MIDI_I= test_midi_in.o midi.o



midi.o : midi.cc midi.h
	$(CC) $(CFLAGS) -c $*.cc -o $@


test_midi_in.o : test_midi_in.cc bib_fred.h
	$(CC) $(CFLAGS) -c $*.cc -o $@


test_midi_in: $(OBJETS_MIDI_I)
	$(CC) -o test_midi_in $(OBJETS_MIDI_I) $(LIBR)

#=============================
all : test_midi_out test_midi_in
```

22.3 Exemple d'envoie de message MIDI à un synthétiseur

Comme application de l'exemple précédent, le programme `gamme.cc` suivant envoie des notes à un synthetiseur ou à un instrument électronique et cela joue la gamme chromatique. Pour cela il envoit un premiere message MIDI qui sélectionne le son de l'instrument sur le canal 0, puis envoie une série d'instruction correspondant à jouer une note/ eteindre une note, avec un espace de 0.2 seconde.

Le numéro de la note do est 60, ensuite do# = 61, etc... Voir messages midi.

Travail préalable

- **Installer un synthétiseur MIDI** comme `midisyn` et le lancer. Il sera alors à l'écoute du port `midithru/0` qui est le port `0` dans notre programme. Par exemple on lance midisyn dans une fenetre de commande à part, par l'instruction :

  ```
  midisyn /home/faure/alex_ratchov/patches/midisyn.conf
  ```

- **Brancher un piano électrique** ou autre instrument midi sur l'ordinateur, et observer le numéro de port par l'instruction `amidi -l` dans un terminal. Cela donne le numéro de port à mettre dans le programme suivant d'après la liste dans le fichier `midi.h`. Par exemple `rmidi/2` (port 2 de l'instruction amidi -l) est sur le port 4.
- Lancer le programme suivant qui va jouer la gamme chromatique.

```
//===== Programme gamme.cc ==========
#include <iostream>

using namespace std;
#include "midi.h"


int main()
{

        Midi midi;

        //--- ouvre port midi out
        int port = 0;

        int res =  midi.Initialise_port_midi_out(port);
        if(res==0)
        {
                cout<<"port est encore ferme."<<endl;
                exit(1);
        }
        else
                cout<<"port ouvert."<<endl;


    //.....
        int ch = 0; // canal
        int inst = 0; // instrument, 0: piano
        midi.Program_Change(port, ch, inst);// associe l'instrument inst au c

        for(int key=60; key<=72; key++) //montee de la gamme chromatique
        {
```

```
                //.......  joue une note midi ..........
                int vol =100;
                cout<<"joue note "<<key<<endl;
                midi.Joue_note(port, ch, key, vol); // message pour jouer une

                midi.Sleep(200); // attente en ms

                //.... eteind une note midi .........
                midi.Eteind_note(port, ch, key); // message pour eteindre la

        }

    //-- ferme les ports midi ouverts
        midi.Ferme_port_midi_out(port);



}
```

```
#======== Fichier Makefile=========================================
OBJETS_MIDI_G=  gamme.o midi.o


midi.o : midi.cc midi.h
        $(CC) $(CFLAGS)  -c  $*.cc -o $@


gamme.o  : gamme.cc
        $(CC) $(CFLAGS)  -c  $*.cc -o $@


gamme:  $(OBJETS_MIDI_G)
        $(CC)  -o gamme  $(OBJETS_MIDI_G)    $(LIBR)
```

Remarque 22.3.1. La liste des codes des instruments est donnée d'après la convention "general midi". Le canal 9 est réservé aux percussions. Mais si vous utilisez le synthetiseur midisyn, pour le moment il n'y a que les possibilités suivantes :

code program_change	0	1	2	3	4	33	73	74	75	76
instrument	piano	guitare	guitare	guitare	guitare	bass	flute	flute	flute	flute

et spécialement sur le canal 9 des percussions :

instrument :	kick	stick	snare1	snare2	hihatc	hihatf	hihato	crash1	ride1_bell	ride1	sp
key :	36	37	38	40	42	44	46	49	51	53	

Le détail des sons est configurable dans les fichiers `patches/midisyn.conf patches/piano/piano.con`

`etc...` On peut par exemple rajouter de nouveaux sons en ajoutant les fichiers raw correspondant et un fichier de configuration associé.

```
//===== Programme gamme.cc ==========
#include <iostream>

using namespace std;
#include "midi.h"


int main()
{

        Midi midi;

        //--- ouvre port midi out
        int port = 0;

        int res =  midi.Initialise_port_midi_out(port);
        if(res==0)
        {
                cout<<"port est encore ferme."<<endl;
                exit(1);
        }
        else
                cout<<"port ouvert."<<endl;


    //.....
        int ch = 0; // canal
        int inst = 0; // instrument, 0: piano
        midi.Program_Change(port, ch, inst);// associe l'instrument inst au c

        for(int key=60; key<=72; key++) //montee de la gamme chromatique
        {
                //.......  joue une note midi ..........
                int vol =100;
                cout<<"joue note "<<key<<endl;
                midi.Joue_note(port, ch, key, vol);

        //... ride
                midi.Joue_note(port, 9, 53, vol);
```

```
                midi.Sleep(200); // attente en ms

                //... stick
                midi.Joue_note(port, 9, 37, vol);

                midi.Sleep(200); // attente en ms

                //.... eteind une note midi .........
                midi.Eteind_note(port, ch, key);

        }
                //... splash
                midi.Joue_note(port, 9, 55, 100);


    //-- ferme les ports midi ouverts
        midi.Ferme_port_midi_out(port);



}
```

22.4 Reception de notes midi depuis un piano électrique

Comme application de l'exemple précédent, le programme reception_notes.cc suivant ecoute les messages midi et les analyse. Si il reconnait "Note on" et "Note off", il affiche les notes identifiées.

Travail préalable

- **Brancher un piano électrique** ou autre instrument midi sur l'ordinateur, et observer le numéro de port par l'instruction `amidi -l` dans un terminal. Cela donne le numéro de port à mettre dans le programme suivant d'après la liste dans le fichier `midi.h`. Par exemple `rmidi/2` (port 2 de l'instruction amidi -l) est sur le port 4.

```
//===== Programme reception_notes.cc ==========
#include <iostream>

using namespace std;
```

```
#include "midi.h"



int main()
{

        Midi midi;


        //--- ouvre un port midi en entree
        int port=0; // numero de port

        int     res = midi.Initialise_port_midi_in(port);
        if(res==0)
                cout<<"port est encore ferme."<<endl;
        else
                cout<<"port ouvert."<<endl;


        //--- attente de messages

        midi.Attente_Message();


        //---analyse des messages recus
        for(auto message : midi.L_message)
        {
                int port_in=(int)message[0]; // numero de port entrant
                message.erase(message.begin()); // enleve le 1er
octet. Il reste le message midi standard


                if(((int)message[0])/16 == 0x9 && ((int)message[2]) > 0 )
// note on avec vel>0
                {
                        int ch=(int)message[0]-0x90;
                        int key=(int)message[1];
                        int vel=(int)message[2];
                        cout<<"Note on recue : port="<<port_in<<" canal="<<ch
                }
                else    if(((int)message[0])/16==0x8   || (((int)message[0])/1
// note off
```

```
                {
                        int ch=(int)message[0];
                        if(ch/16==0x8)
                                ch-= 0x80;
                        else if(ch/16==0x9)
                                ch-= 0x90;


                        int key=(int)message[1];
                        int vel=(int)message[2];
                        cout<<"Note off recue : port="<<port_in<<" canal="<<c
                }
        }


//--- ferme port midi
        midi.Ferme_port_midi_in(port);



}


#=============================================
OBJETS_MIDI_R=  reception_notes.o midi.o


midi.o : midi.cc midi.h
        $(CC) $(CFLAGS)  -c  $*.cc -o $@


reception_notes.o : reception_notes.cc
        $(CC) $(CFLAGS)  -c  $*.cc -o $@


reception_notes:  $(OBJETS_MIDI_R)
        $(CC)  -o reception_notes  $(OBJETS_MIDI_R)    $(LIBR)
#=============================
all : reception_notes
```

Chapitre 23

Messages MIDI (Musique) sur fichier

Les messages musicaux MIDI peuvent être écrit dans un fichier. Ce format s'appelle "Standard Midi File" S.M.F.

ref : ici ou ici ou la référence.

La convention des messages est très similaire aux messages en temps réel. Il y a peu de différences (il y a "Message méta évènement" et plus de "Message temps réel")

23.1 Structure d'un fichier SMF

Il est constitué de blocs de données appelés "chunk" (=gros morceau). Le premier bloc annonce les suivants.

Conventions sur le codage des nombres :

— CLF "Codage à longueur fixe". Un nombre entier positif X est codé sur n octets $x_1 \dots x_n$ avec $x_j \in [0, 255]$ au format big-endian, c'est à dire que le nombre associé X est

$$X = x_1 256^{n-1} + x_2 256^{n-2} + \dots x_{n-1} 256 + x_n$$

Sauf mention du contraire on utilise CLF.

— CLV "Codage à longueur variable": pour n octets $x_1, x_2, \dots x_n$ on utilise seulement les 7 derniers bits de chaque octet. Le **premier bit** quand à lui est 1 (et donc octet $x_j \geq 128$) pour les octets $0 \leq j < n$, et pour le dernier octet utilisé $j = n$ le premier bit est 0 (et donc octet $x_n < 128$). Le nombre associé X est (format big-endian)

$$X = \tilde{x}_1 128 + \dots \tilde{x}_{n-1} 128 + x_n, \quad \text{avec } \tilde{x}_j = x_j - 128$$

23.1.1 Premier bloc (header chunk)

Il est composé de

— 4 octets : "`MThd`" = 0x4D546864. Ces 4 caractères invariables en début de fichier signalent que c'est un fichier MIDI.

— L : 4 octets : 0x00000006 : longueur invariable en octets de ce bloc, comptée après ces 4 octets.
— C : 2 octets :
  — 0x0000 : si le fichier contient un seul bloc après celui ci. Il contiendra éventuellement plusieurs canaux (pour avoir polyphonie)
  — 0x0001 : (le plus courant) si le fichier contient plusieurs blocs après celui ci. Ils seront joués simultanément. (polyphonie). Les métadonnées sont sur le premier bloc.
  — 0x0002 : si le fichier contient plusieurs blocs après celui ci. Ils seront joués les un après les autres dans le temps.
— N : 2 octets : nombre de blocs qui vont suivre ce premier bloc.
— F : 2 octets : "Delta time" ou "unité de temps".
  — Si le premier bit de F est 0, cad $F < 2^{15} = 32768$, alors l'unité de temps est

$$\delta t = \frac{T}{F}$$

où T qui est le tempo cad la durée entre deux pulsation. ex : $T = \frac{1}{92} min.$ est "Andante" donnée après dans les méta données (ou pas donnée). En général, $F = 120, 240, 480, 960$
  — Si le premier bit de F est 1, cad $F \geq 2^{15} = 32768$, alors F est codé en "SMPTE compatible units. Midi time code" cad

$$F = (100bbbbb \quad cccccccc)_{base2}$$

avec $(bbbbb)_{base2}$ est le nombre d'images par secondes (ex : $24, 25, 29, 30$) et $(cccccccc)_{base2}$est le nombre de subdivisions d'une image ("frame")

23.1.2 Bloc de piste (Track chunk)

Il est composé de
— 4 octets : "`MTrk`" = 0x4D54726B
— L : 4 octets : longueur en octets de ce bloc. (comptés après ces 4 octets)
— Une suite d'"évènements midi" (Midi event). Chaque évènement midi est composé de
  — N_t : nombre de pas de temps δt écoulés depuis l'évènement précédant, codé au format CLV.
  — M : Un **message midi** (le nombre d'octets dépend du premier quartet) :
    — Si le premier quartet q (4 bits) a le premier bit 1, cad $q \geq 8$ alors M a 2 ou 3 octets.

    — Si le premier quartet q (4 bits) a le premier bit 0, cad $q < 8$ alors M a deux octets, c'est à dire que la dernière action est répétée, meme 1er octet, "running status". Par exemple pour coder l'accord C-E-G sur le canal 3

— (x00 - x93 - x3C - x40) (x00 - x93 - x40 - x40) (x00 - x93 - x43 - x40) : sans running status
— ou (x00 - x93 - x3C - x40) (x00 - x40 - x40) (x00 - x43 - x40) : avec running status

La **fin de bloc** est annoncée par le message avec 3 octets: xFF, x2F, x00

23.1.3 Message Midi standard M

M a 2 ou 3 octets $M = x_1, x_2, x_3$. Le premier octet est décomposé en 2 quartet (4 bits chacuns)

$$x_1 = qc$$

où$c \in [0, 15]$ est le numéro de canal. Les octets suivants sont

$$0 \leq x_1, x_2 < 128$$

x_1	signification et nombres d'octets à suivre	x_2
	Messages standard	
x8c	Note est éteinte sur un canal c	note : $C_0 \rightarrow$
x9c	Note est jouée sur canal c	note
xAc	Aftertouch sur canal c(vibrato sur une note)	note
xBc	Control Change sur canal c (paramétrage d'un canal) cf dessous	type
xCc	Program Change sur canal c (changement d'instrument sur un canal)	instrume
xDc	Canal pressure sur canal c (vibrato sur un canal)	vibrato
xEc	Pitch Wheel Change (glissando sur un canal) décale de $\delta = x_2 + 128x_3 - 64.128$.	x_2

23.1.4 Message méta évènement M (pour les signaux midi sur fichier et non pas temps réel)

x_1	signification	x_2	$x_3, \ldots$
xFF	Méta-évènement. cf Section 23.1.8	type	Lenght (+sieurs octets)

23.1.5 Message du système (System Exclusive Message (SysEx)) M

x_1	signification	x_2	$x_3, \ldots$
xF0	System Exclusive Message (SysEx)		
xF7			

Le message se termine par le même octet 0xF7 :
— 0xF0 + <data_bytes> 0xF7
— ou: 0xF7 + <data_bytes> 0xF7

23.1.6 Message temps réel M (pour les signaux midi temps réel et non sur fichier)

x_1	signification
xF8	Timing Clock: Envoyé 24 fois par noire en cas d
xFA	Start: Démarre une séquence. Il est possible souvent d'autoriser un clavier à déclencher le dén
xFB	Continue : Reprend une séquence à l'endroit où el
xFC	Stop : Arrête la séquence en cour
xFE	En cas de premier envoi, ce message doit être répété avec un délai maximum de 0,3 seconde, s
xFF	Reset Réinitialisation

23.1.7 Control Change pour les message $x_1 = xBc$

(copié de jchr)

Selon la valeur de x_2= type : x00 Bank Select : sélection d'une banque de sons
x01 Modulation Wheel : roulette de modulation (à 0 si réinitialisation)
x02 Breath Controller : contrôle par le souffle
x04 Foot Controller : contrôle au pied
x05 Portamento Time : rapidité du portamento
x06 Data Entry MSB : octet fort de donnée supplémentaire
x07 Channel Volume : volume principal d'un canal
x08 Balance
x0A Pan : panoramique, 64 est le milieu
x0B Expression Controller : contrôle de l'expression (127 si réinitialisation)
x0C Effect Control 1 : effet 1
x0D Effect Control 2 : effet 2
x10 à x13 General Purpose 1-4 : contrôleurs tout usage
x20 à x3F octet de poids faible pour les contrôleurs 0-31
x40 Damper Pedal : maintient toutes les notes (0 si réinitialisation)
x41 Portamento On/Off (0 si réinitialisation)
x42 Sostenuto On/Off : sustain pour la note jouée au moment du déclenchement (0 si réinitialisation)
x43 Soft Pedal On/Off : pédale douce (0 si réinitialisation)
x44 Legato Footswitch : contrôle de legato au pied
x45 Hold 2 : autre sustain
x46 Sound Controller 1 / Sound Variation : variation du timbre
x47 Sound Controller 2 / Timbre/Harmonic Intens. : contenu harmonique du timbre
x48 Sound Controller 3 / Release Time : temps de release par défaut
x49 Sound Controller 4 / Attack Time : temps d'attaque
x4A Sound Controller 5 / Brightness : brillance
x4B Sound Controller 6 / Decay Time : temps d'extinction du son

x4C Sound Controller 7 / Vibrato Rate : fréquence de vibrato
x4D Sound Controller 8 / Vibrato Depth : profondeur du vibrato
x4E Sound Controller 9 / Vibrato Delay : délai avant le vibrato
x50 à x53 General Purpose Controller 5-8
x54 Portamento Control
x58 High Resolution Velocity Prefix
x5B Effects 1 Depth / Reverb Send Level : intensité de réverbération
x5C Effects 2 Depth / formerly Tremolo Depth : intensité de trémolo
x5D Effects 3 Depth / Chorus Send Level : intensité de chorus
x5E Effects 4 Depth / formerly Celeste Detune Depth : profondeur du désaccord
x5F Effects 5 Depth / formerly Phaser Depth : intensité de phaser
x60 Data Increment (Data Entry +1) : incrémentation de donnée (pas d'octet de valeur)
x61 Data Increment (Data Entry -1) : décrémentation de donnée (pas d'octet de valeur)
x62 NRPN / Non-Registered Parameter Number - LSB : octet de poids faible de numéro de paramètre non enregistré
x63 NRPN / Non-Registered Parameter Number - MSB : octet de poids fort de numéro de paramètre non enregistré
x64 RPN / Registered Parameter Number - LSB : octet de poids faible de numéro de paramètre enregistré
x65 RPN / Registered Parameter Number - MSB : octet de poids fort de numéro de paramètre enregistré
x78 - x00 All Sound Off : éteint tous les sons, même les fins d'enveloppe
x79 - x00 Reset All Controllers : réinitialise tous les contrôles
x7A - xnn Local Control On/Off : désactive (x00) ou réactive (x7F) la production du son par le clavier maître
x7B - x00 All Notes Off : éteint toutes les notes (mais pas les enveloppes), comme le font les quatre suivant
x7C - x00 Omni Mode Off - recommandé explanations
x7D - x00 Omni Mode On : envoie les événements sur tous les canaux
x7E - xnn Mono Mode On : ne permet qu'une note à la fois sur un canal, ce qui permet de mieux simuler la production de son d'un instrument monophonique (portamento) ; xnn est le nombre de canaux ? ? ?
x7F - x00 Poly Mode On : permet plusieurs notes simultanées sur un même canal

23.1.8 type pour les Meta-Evenement

Un meta-evenement contient
— 0xFF
— 1 octet : "meta-type event" : (cf cidessous)
— 1 octet L : length of meta event data ou en codage CLV ?
— the actual event data.

Par exemple: 3 octets: xFF, x2F, x00 en **fin de bloc**.

type (en hexa)	signification
00	Sequence Number
01	Text Event
02	Copyright Notice
03	Sequence Name
04	Instrument Name
05	Lyric/Display
06	Marker text
07	Cue point
08	Program Name
09	Peripheric Name
20	MIDI channel prefix assignment
$2F$	End of track (cf + haut)
51	Tempo Setting
84 (dec)	SMPTE Offset
88(dec)	Time Signature
89 (dec)	Key Signature
127 (dec)	Sequencer Specific Event

23.1.9 Systeme exclusif Messages (Sysex)

Constructeurs Ces identifiants servent lors de messages spécifiques au matériel (SysEx, DLS and XMF). Les membres historiques du MMA disposent d'un identifiant d'un octet ; les plus récents consistent en deux octets préfixés de l'octet nul.

x00 (préfixe) x0E Matthews Research x24 Hohner x46 Kamiya Studio
x01 Sequential Circuits x10 Oberheim x25 Crumar x47 Akai
x02 Big Briar x11 PAIA x26 Solton x48 Victor
x03 Octave / Plateau x12 Simmons x27 Jellinghaus Ms x4B Fujitsu
x04 Moog x13 DigiDesign x28 CTS x4C Sony
x05 Passport Designs x14 Fairlight x29 PPG x4E Teac
x06 Lexicon x15 JL Cooper x2F Elka x50 Matsushita
x07 Kurzweil x16 Lowery x36 Cheetah x51 Fostex
x08 Fender x17 Lin x3E Waldorf x52 Zoom
x09 Gulbransen x18 Emu x40 Kawai x54 Matsushita
x0A Delta Labs x1B Peavey x41 Roland x55 Suzuki
x0B Sound Comp. x20 Bon Tempi x42 Korg x56 Fuji Sound
x0C General Electro x21 S.I.E.L. x43 Yamaha x57 Acoustic Technical Lab.
x0D Techmar x23 SyntheAxe x44 Casio

Chapter 24

Autres librairies conseillées (?)

— Trinilos
— Math Kernel Library

24.1 algèbre linéaire

— SimuNova
— Eigen pour l'algèbre linéaire.

24.2 Images, vidéos, graphisme et boites de dialogues

— SDL pour le graphisme de base (avec OpenGL) et le son audio. Librairie bien adapté pour les jeux.
— FLTK pour les boites de dialogues et le graphisme. Bibliothèque "légère" et performante.
— QT bibliothèque très générale, performante, multiplateformes mais assez complexe.

Chapitre 25

Le traitement d'images et de vidéos avec la librairie OpenCV

OpenCV est spécialisée pour le traitement d'image et de vidéos (CV signifie "Computer-Vision"). Un Tutoriel. Livre conseillé : "OpenCV by example" de P. Joshi, D.M. Escriva et V. Godoy.

25.1 Commande de compilation

Avant d'écrire un programme utilisant les librairies de OpenCV, il faut, une fois pour toutes,

— Dans l'éditeur codeblocks

— Dans Settings/Compiler/Linker_Settings/Other_linker_Options, rajouter :

```
`pkg-config --cflags --libs opencv`
```

Remarque 25.1.1. Si vous n'utilisez pas codeblocks, mais Makefile par exemple, la compilation en ligne de commande du programme `main.cpp` pour donner l'executable `main` est :

```
g++ main.cpp -o main -g -std=c++11 `pkg-config --cflags --libs opencv`
```

25.2 Lecture et ecriture de fichiers images

Les commentaires expliquent les fonctions. Il faut avoir le fichier Image.jpg dans le répertoire du programme.

```
#include <iostream>
using namespace std;
```

```
#include <string>

#include <opencv2/core/core.hpp>
#include <opencv2/highgui/highgui.hpp>
using namespace cv;


//============================
int main()
{

//..... Lecture des images -> img
Mat img= imread("Image.jpg");


//..... Ecriture d'une image: img -> fichier Image2.jpg
imwrite("Image2.jpg", img);


//..... Lecture d'un pixel
int x=img.cols/2; // selectionne point (x,y) au centre de l'image
int y=img.rows/2;
Vec3b p= img.at<Vec3b>(x,y);
cout << "La valeur du pixel en x="<<x<<", y="<<y<<" est (B,G,R): (" << (int)p

//..... dessin de l'image
imshow("Image", img);

//..... Attend qu'une touche soit appuyee.
int c = waitKey(0); // rem: montre le dessin. Renvoit le code de la touche. C


}
```

Resultat :

- Affiche deux images : une en couleur et l'autre en noir et blanc.
- `Pixel value (B,G,R): (3,7,12)`

25.2.0.1 Fonctions utiles sur les images

Copie d'une image :

— La commande `Mat image2 = image1;` ne fait pas une copie des images mais seulement de l'adresse (c'est à dire que `image1` et `image2` vont désigner la même image). Pour **dupliquer l'image** il faut faire `image1.copyTo(image2);`

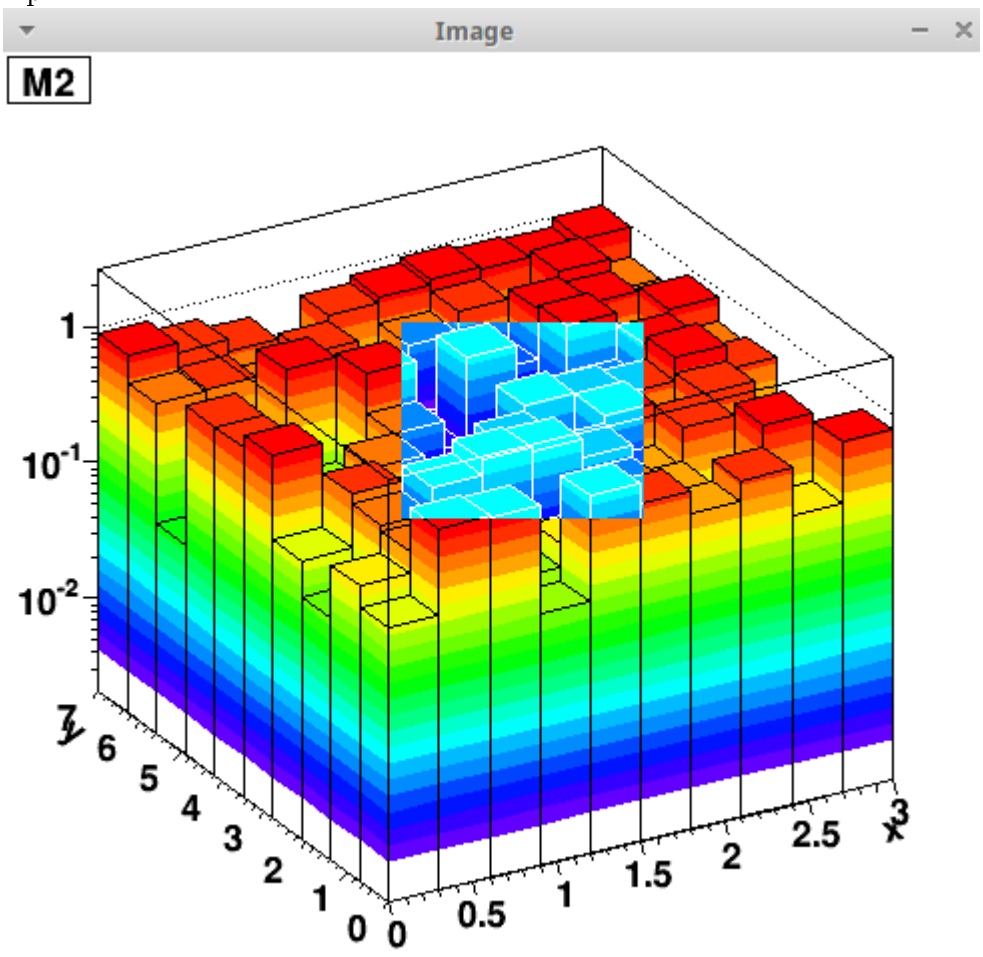
Recadrage d'une image :

— resize(frame, frame, Size(), scalingFactor, scalingFactor, INTER_AREA) ;

— Mat img= imread("Image.jpg", 0) ; // 0 : convertit en gris

25.3 Utilisation de la souris

Le programme suivant montre comment utiliser la souris. Dans cet exemple on selectionne une partie de l'image avec clic gauche (down/up) et on revient à l'image de départ avec clic droit (down). Le codesuivant utilise le fichier image M2.png à télécharger dans le même répertoire.



—

```
#include <iostream>
```

```
using namespace std;


#include <string>


#include <opencv2/core/core.hpp>

#include <opencv2/highgui/highgui.hpp>

using namespace cv;


//-----------------------------------
Mat img, image, roi;

int etat_select = 0;// 0: selection pas faite, 1: selection en cours, 2: sele

Rect selection; // taille de la selection
Point origin; // point de depart de la selection

//---- fonction appelee si la souris est actionnee

static void onMouse( int event, int x, int y, int, void* )
{
//      cout<<"event="<<event<<" x="<<x<<" y="<<y<<endl;

        //--- si action du clic souris

    switch( event )
    {
        case 1: // clic gauche down
                if(etat_select==0)
                {
                        origin = Point(x,y);
                        selection = Rect(x,y,0,0);
                        etat_select = 1;
                }
        break;
    case 4: // clic gauche up
        if( etat_select==1 && selection.width > 0 && selection.height > 0 )
                {
                        etat_select =2; // selection finie
```

```
                        bitwise_not(roi, roi); // inversion de roi (dans imag
                        imshow("Image", roi);
                }
        break;

        case 2: //clic droit down
                if(etat_select==2)
                {
                        etat_select = 0;
                                imshow("Image", img);
                }
    }

        //--- si deplacement souris,  traitement de l'image
    if( etat_select == 1 )
    {
        selection.x = MIN(x, origin.x);
        selection.y = MIN(y, origin.y);
        selection.width = std::abs(x - origin.x);
        selection.height = std::abs(y - origin.y);
        selection &= Rect(0, 0, img.cols, img.rows); // pour ne pas depasser

                if(selection.width > 0 && selection.height > 0 )
                {

                        img.copyTo(image);
                        roi = Mat(image, selection); // la matrice roi est un
                        bitwise_not(roi, roi); // inversion de roi (dans imag
                        imshow("Image", image); // montre image avec sa selec
                }
    }


}


//=========================

int main()

{

//..... Lecture des images
```

```
img= imread("M2.png");


//..... Affiche l'image a l'ecran
namedWindow( "Image", 1 );
setMouseCallback( "Image", onMouse, 0 ); // associe la fonction de lecture de


imshow("Image", img);

waitKey(0);



}
```

25.4 Lecture et écriture d'un fichier video ou de la webcam

```
#include <iostream>
#include <string>
#include <sstream>
using namespace std;


#include "opencv2/core.hpp"
#include "opencv2/highgui.hpp"
using namespace cv;


//-----------------------------------------
int main()
{
    //.... ouverture du fichier video ou webcam
    String nom = "video_pendule.mp4"; // fichier video
    VideoCapture cap;
    cap.open(nom);
    //cap.open(0);// ouvre la webcam.
    if(!cap.isOpened()) // si erreur d'ouverture
        return -1;
```

```
//...ouverture d'un fichier video en ecriture
    String nom2= "video_pendule2.avi"; // fichier video
    Size frameSize(static_cast<int>(cap.get(CV_CAP_PROP_FRAME_WIDTH)), static_
    VideoWriter out (nom2, CV_FOURCC('P','I','M','1'), 20, frameSize, true);
    if ( !out.isOpened() )
        return -1;



//.... boucle infinie
    while(true)
    {
        Mat img;
        cap >> img; // extrait la prochaine image
        imshow("Video", img); // dessin de img dans la fenetre Video

        out.write(img); // ecrit l'image dans fichier

        if(waitKey(30) >= 0) break; // Montre dessin. Attente de 30 ms et arr
    }


    cap.release(); //libere la memoire allouee

}
```

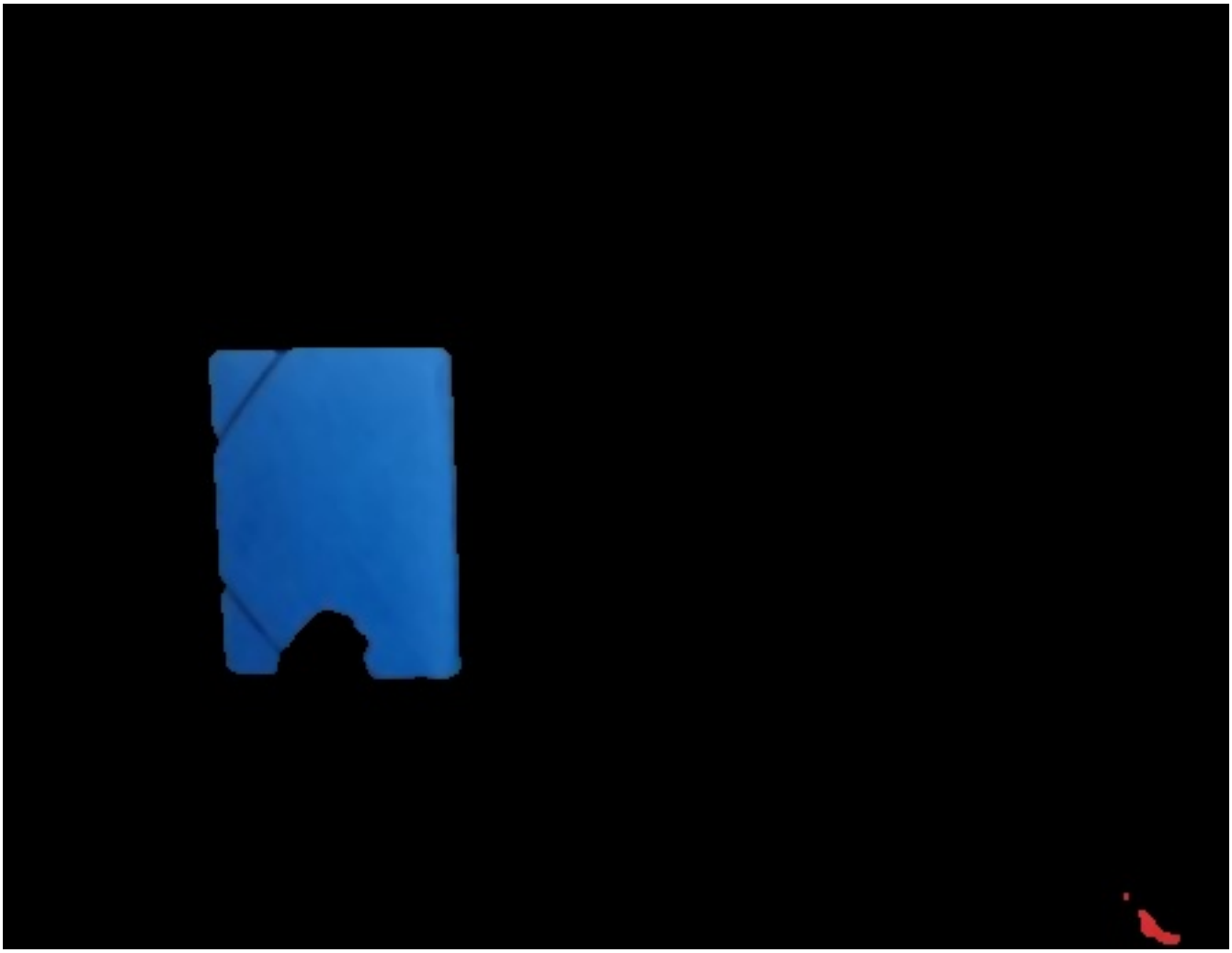
Remarque 25.4.1. La commande waitKey(0) permettrait de faire avancer la video image par image.

25.5 Détection d'objet par leur couleur

Voici un exemple de programme qui lit une image et selectionne les objets bleus. Pour afficher les images intermediares, il faut décommenter les lignes correspondantes.

- Voici une documentation sur les formats de couleurs. On utilise le format HSV (voir aussi HSV sur wikipedia) qui est proche de la perception humaine : $H \in [0,1]$ est la teinte de la couleur, par exemple $H \in [0.5, 0.7]$ est une palette de bleus. $S \in [0,1]$ est la saturation entre $S = 0$ (blanc) et $S = 1$ (couleur vive). $V \in [0,1]$ est la luminosité entre $V = 0$ (noir) et $V = 1$
- Sauver l'image "objet_bleu.jpg" ci-dessous dans le répertoire du programme.

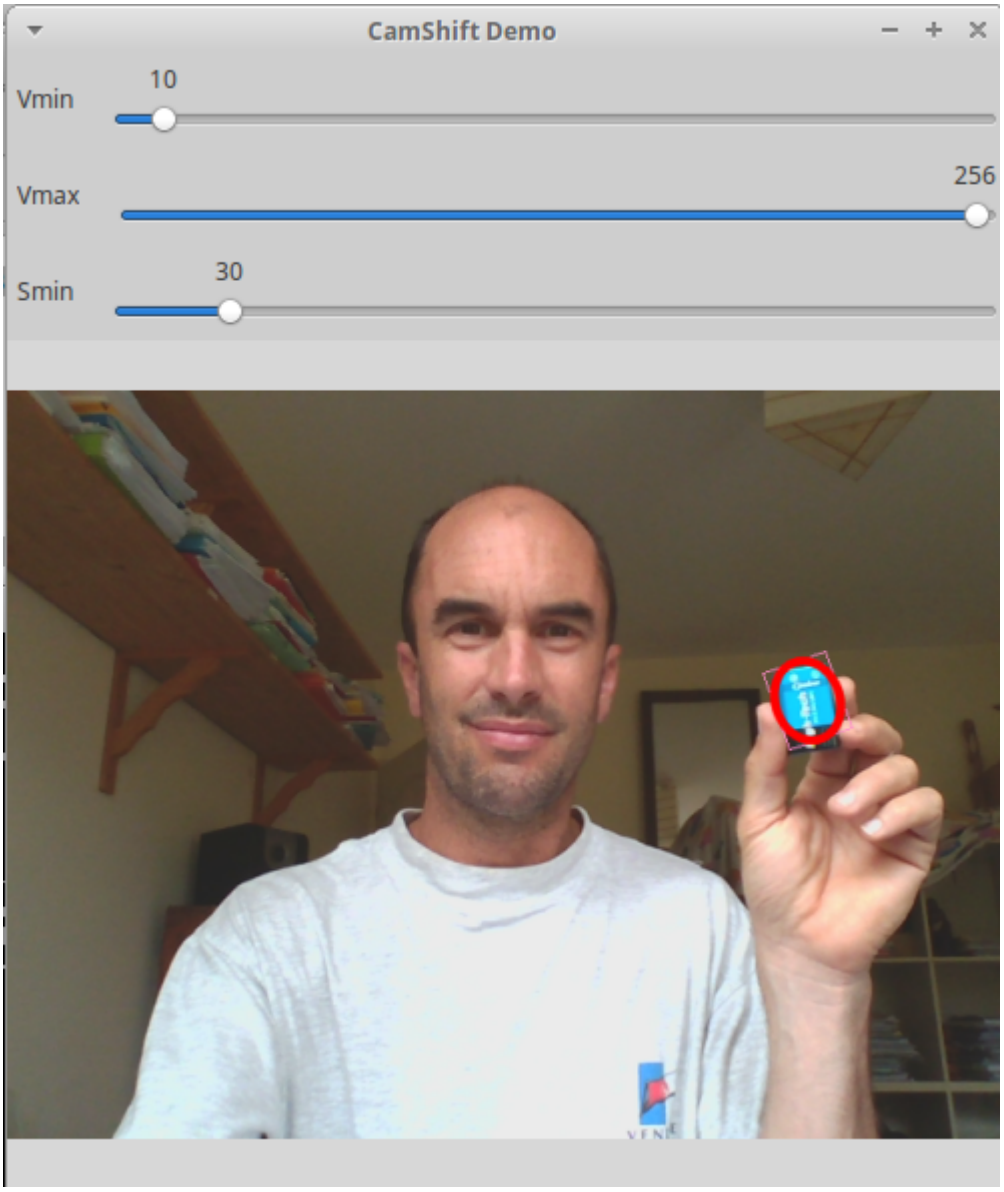
— Résultat du programme :



#**include** <iostream>**using namespace** std;#**include** <string> #**include** <opencv2/core/core.hp

25.6 Détection et suivit d'objet de couleur avec l'algorithme CamShift

Voici des informations, sur les algorithmes MeanShift et CamShift , ou sur wikipedia.



Le code ci-dessous est tiré du site web de opencv.

```
#include "opencv2/video/tracking.hpp"
#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"

#include <iostream>
#include <ctype.h>

using namespace cv;
using namespace std;
```

```
//-----------------------------------
Mat image;


bool backprojMode = false; // true: si on montre l'image seuillee
bool showHist = false; // true: on montre l'histogramme

bool selectObject = false; // true: si on est en train de selectionner
int trackObject = 0; //0: pas de selection faite. -1: selection faite mais Ãă
Rect selection; // taille de la selection
Point origin; // point de depart de la selection

//---- Si l'utilisateur selection un rectangle -> selection et le flag trackC

static void onMouse( int event, int x, int y, int, void* )
{
    //  cout<<"event="<<event<<" x="<<x<<" y="<<y<<endl;

    if( selectObject )
    {
        selection.x = MIN(x, origin.x);
        selection.y = MIN(y, origin.y);
        selection.width = std::abs(x - origin.x);
        selection.height = std::abs(y - origin.y);

        selection &= Rect(0, 0, image.cols, image.rows); // pour ne pas depas
    }

    switch( event )
    {
        cout<<"event="<<event<<endl;
    case CV_EVENT_LBUTTONDOWN: // coin de depart
        origin = Point(x,y);
        selection = Rect(x,y,0,0);
        selectObject = true;
        break;
    case CV_EVENT_LBUTTONUP:
        selectObject = false;
        if( selection.width > 0 && selection.height > 0 )
            trackObject = -1;
        break;
    }
```

```
}

//-------------------------------------
static void help()
{
    cout << "\nThis is a demo that shows mean-shift based tracking\n"
         "You select a color objects such as your face and it tracks it.\n"
         "Usage: \n"
         "    ./camshiftdemo \n";

    cout << "\n\nHot keys: \n"
         "\tESC - quit the program\n"
         "\tc - stop the tracking\n"
         "\tb - switch to/from backprojection view\n"
         "\th - show/hide object histogram\n"
         "\tp - pause video\n"
         "To initialize tracking, select the object with mouse\n";
}


///-------------------------------------
int main( int argc, const char** argv )
{
    help();

    VideoCapture cap;
    Rect trackWindow;
    int hsize = 16; // pour histogramme
    float hranges[] = {0,180};
    const float* phranges = hranges;



    String videoFile= "pendule_jaune.mp4"; // fichier video
//    cap.open(videoFile); // ouvre le fichier video

    cap.open(0); // ouvre camera

    if( !cap.isOpened() )
    {
        cout << "***Could not initialize capturing...***\n";
        return -1;
    }
```

```
//... initialise les fenetres

namedWindow( "Histogram", 0 );
namedWindow( "CamShift Demo", 0 );
setMouseCallback( "CamShift Demo", onMouse, 0 ); // associe la fonction d


//.... pose les sliders
int vmin = 10, vmax = 256, smin = 30;
createTrackbar( "Vmin", "CamShift Demo", &vmin, 256, 0 );
createTrackbar( "Vmax", "CamShift Demo", &vmax, 256, 0 );
createTrackbar( "Smin", "CamShift Demo", &smin, 256, 0 );

Mat frame, hsv, hue, mask, hist, histimg = Mat::zeros(200, 320, CV_8UC3),
bool paused = false;

//------ boucle infinie ----------
for(;;)
{
    if( !paused )
    {
        cap >> frame;
        if( frame.empty() )
            break;
    }

    frame.copyTo(image); // -> image

    if( !paused )
    {
        cvtColor(image, hsv, COLOR_BGR2HSV); // convertit -> hsv

        if( trackObject )
        {
            int _vmin = vmin, _vmax = vmax;

            // seuillage -> mask (fenetre de 0 et 1)
            inRange(hsv, Scalar(0, smin, MIN(_vmin,_vmax)), Scalar(180, 2
            //imshow( "mask", mask);
```

```
                    // Mix the specified channels
                    int ch[] = {0, 0};
                    hue.create(hsv.size(), hsv.depth()); // -> hue: nouvelle fene

                    mixChannels(&hsv, 1, &hue, 1, ch, 1); // -> copie hsv dans hu


                    // on traite la selection -> histogramme
                    if( trackObject < 0 )
                    {
                        cout<<"ici"<<endl;

                        // Create images based on selected regions of interest (r
                        Mat roi(hue, selection), maskroi(mask, selection);
// imshow( "hue", hue);
//imshow( "mask", mask);
//waitKey(0);

                        // Compute the histogram and normalize it -> tableau hist
                        calcHist(&roi, 1, 0, maskroi, hist, 1, &hsize, &phranges)
                        normalize(hist, hist, 0, 255, CV_MINMAX);


                        trackWindow = selection;
                        trackObject = 1;

                        histimg = Scalar::all(0);
                        int binW = histimg.cols / hsize;
                        Mat buf(1, hsize, CV_8UC3);
                        for( int i = 0; i < hsize; i++ )
                            buf.at<Vec3b>(i) = Vec3b(saturate_cast<uchar>(i*180./
                        cvtColor(buf, buf, CV_HSV2BGR);

                        // dessin des rectangles de l'histogramme sur la fenetre
                        for( int i = 0; i < hsize; i++ )
                        {
                            int val = saturate_cast<int>(hist.at<float>(i)*histim
                            rectangle( histimg, Point(i*binW,histimg.rows), Point
                        }
                    }

                    // Compute the histogram backprojection
                    calcBackProject(&hue, 1, 0, hist, backproj, &phranges); // ->
```

```
                backproj &= mask; // ameliore backproj


                //... trackwindow, backproj -> trackBox
                RotatedRect trackBox = CamShift(backproj, trackWindow,
TermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ));

                // Check if the area of trackingRect is too small
                if( trackWindow.area() <= 1 )
                {
                    // Use an offset value to make sure the trackingRect has
                    int cols = backproj.cols, rows = backproj.rows, r = (MIN(
                    trackWindow = Rect(trackWindow.x - r, trackWindow.y - r,
                                       trackWindow.x + r, trackWindow.y + r)
                                  Rect(0, 0, cols, rows);
                }

                if( backprojMode )
                    cvtColor( backproj, image, COLOR_GRAY2BGR );



                // draw rotated rectangle trackBox
                Point2f rect_points[4];
                trackBox.points( rect_points );

                RNG rng(12345);
                Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255
                for( int j = 0; j < 4; j++ )
                    line( image, rect_points[j], rect_points[(j+1)%4], color,



                // Draw the ellipse on top of the image
                ellipse( image, trackBox, Scalar(0,0,255), 3, CV_AA );
            }
        }
        else if( trackObject < 0 )
            paused = false;
```

```
        // Apply the 'negative' effect on the selected region of interest
        if( selectObject && selection.width > 0 && selection.height > 0 )
        {
            Mat roi(image, selection);
            bitwise_not(roi, roi);
        }

        imshow( "CamShift Demo", image );
        imshow( "Histogram", histimg );

        char c = (char)waitKey(10);
        if( c == 27 ) // esc.
            break;
        switch(c)
        {
        case 'b':
            backprojMode = !backprojMode;
            break;
        case 'c':
            trackObject = 0;
            histimg = Scalar::all(0);
            break;
        case 'h':
            showHist = !showHist;
            if( !showHist )
                destroyWindow( "Histogram" );
            else
                namedWindow( "Histogram", 1 );
            break;
        case 'p':
            paused = !paused;
            break;
        default:
            ;
        }
    }

    return 0;
}
```

Chapitre 26

Conversion d'un programme C++ en programme Java Script avec emscripten

— Le **langage javascript** est reconnu par tous les navigateurs internet et donc les programmes écrits en javascript peuvent tourner sur toutes les machines (ou smartphone ou tablettes). Javascript fait partie des trois langages de base pour la programmation des pages web :

1. HTML qui définit le contenu des pages web. Tutorial. Reference. Voici un site qui permet de valider votre code html, ou trouver les erreurs.
2. CSS qui spécifie le style des pages web. Tutorial. Reference.
3. JavaScript qui permet de programmer le comportement des pages web. Tutorial. Reference.

— Le **logiciel emscripten** permet de convertir automatiquement du code C++ en code javascript. Cela évite de convertir soi même le code à la main, ou d'écrire en javascript (si on ne connait pas ce langage). De plus le code covertit et compilé peut être plus rapide que du code javascript natif.

— Il y a une alternative : Cheerp. Comparaison entre cheerp et emscripten

Dans cet Section, on explique comment faire cela avec des exemples très simples. Voici des exemples sophistiqués : Box2d ou Bullets

Documentations :

— Voici des slides.

26.1 Installation

Voir Installation avec Portable SDK.

On vérifie que l'installation est correcte par `./emcc -v`

26.2 Conversion C++ -> Javascript

Exemple de base : Copier le code suivant dans le fichier `test.cc`

```
#include <iostream>
using namespace std;
double somme(double a,double b)
{
return a+b;
}
int main()
{
double a=2, b=3.1;
cout<<" a="<<a<<", b="<<b<<", a+b="<<a+b<<endl;
}
```

Compilation : `emcc test.cc -std=c++11 -O2 -o test.js`
Cela produit l'executable javascript `test.js`

Execution dans un terminal : `node test.js`
Cela affiche :
`a=2, b=3.1, a+b=5.1`

Pour créer un fichier html : `emcc test.cc -std=c++11 -O2 -o test.html`
cela crée un fichier test.html et le script test.js à part, que l'on execute par :
`firefox test.html`

26.2.1 Autres exemples

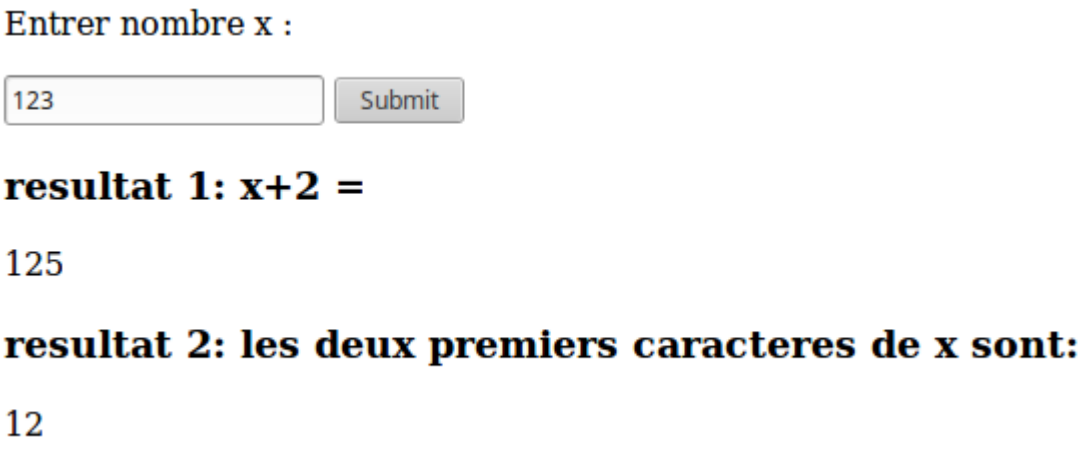
Voir /emsdk_portable/emscripten/master/tests/
Voir tutorial

26.3 Pour connecter un code C++ et une interface html

On décrit ici l'utilisation de Cwrap. Autre possibilité serait une utilisation de WebIDL (voici un exemple).

26.3.1 Exemple de base

Résultat : Lien : code.html.



Code C++ : Copier le code suivant dans le fichier `test.cc`

```
#include <math.h>
extern "C"
{
//-------------
int f1(int x)
{
return x+2;
}
//-------------
char* f2(char*s)
{
char *u;
u = new char[3];
u[0]=s[0];
u[1]=s[1];
u[2]='\0'; // fin de chaine
return u;
}
}
```

Compilation : dans un terminal écrire :

```
emcc test.cc -std=c++11 -O2 -o test.js -s EXPORTED_FUNCTIONS="['_f1','_f2']"
```

Cela produit l'executable javascript `test.js`

Remarque 26.3.1. Dans cette commande on rajoute _ au nom de la fonction.

Code html : Ecrire le code suivant dans un fichier `code.html`

```
<!DOCTYPE html>
<html>
<body>
<p>Entrer nombre x :</p>
<input id="n">
<button type="button" onclick="F()">Submit</button>
<h3>resultat f1(x)=x+2:</h3>
<p id="zone1"></p>
<h3>resultat f2(x): 2 premiers caracteres de x sont:</h3>
<p id="zone2"></p>
<script src="test.js"></script>
<script>
function F()
{
var x = document.getElementById("n").value; // recupere entree (string)
//........
f1 = Module.cwrap('f1', 'number', ['number']); // declaration de
la fonction qui est definie dans test.js
var y1 = f1(x);
document.getElementById("zone1").innerHTML = y1; // envoie resultat
(string)
//........
f2 = Module.cwrap('f2', 'string', ['string']);
var y2 = f2(x);
document.getElementById("zone2").innerHTML = y2; // envoie resultat
(string)
}
</script>
</body>
</html>
```

Execution : avec la commande `firefox code.html`

Attention il faut que le script test.js soit présent dans le même répertoire.

Partage du programme : Si vous posez les fichiers `code.html` et `test.js` dans le même répertoire sur un serveur web, alors un utilisateur extérieur qui charge le fichier code.html pourra exécuter votre programme.

Cela marchera depuis toute plateforme (ordinateur sous linux, windows, mac ou même iphone, android etc).

Par exemple le programme précédent est utilisable ici : code.html.

Remarque 26.3.2.

— Dans le fichier `code.html`, on écrit `f1 = Module.cwrap('f1', 'number', ['number','number'])` pour une fonction avec deux entrée et une sortie. En sortie : on met null si il n'y a pas de variable. array si on renvoit un tableau d'octets.
— Il est important de savoir (et parfois utile) que si il y a des **variables globales** (ex : des tableaux etc) dans le code c++, alors leur **contenu est conservé** à l'appel suivant.

26.3.2 Exemple avec des échanges plus complexes

Dans l'exemple précédent, l'interface html et le programme c++ échangent seulement une variable de type `int` ou une chaine de caractère.

Pour échanger des variables diverses (provenant de "forms" et qui en retour peuvent modifier le contenu de la page html), on pourra tout simplement passer l'information codée dans une unique chaine de caractères.

26.3.2.1 Exemple

Code C++ : (fichier test.cc)

```
#include <math.h>
#include <sstream>
#include <string>
#include <iostream>
using namespace std;
#include <vector>
#include <string.h>

//=======================================
// Decompose la chaine "| .. | ..|" en sous chaines des caractères
entre les '|' =delim
vector<string> Decompose(string ligne,char delim)
{
vector<string> instructions;
while(ligne.find(delim)!=string::npos) // il reste une instruction
{
auto pos= ligne.find(delim);
string ligne2=ligne.substr(0,pos);
//.................
instructions.push_back(ligne2);
ligne=ligne.substr(pos+1);
// cout<<" I="<<ligne2<<flush;
```

```
    }
    if(ligne.size()>0)
    {
    // cout<<" I="<<ligne<<flush;
    instructions.push_back(ligne);
    }
    return instructions;
    }
    //=========================
    extern "C"
    {
    const char* f(char*s)
    {
    //--- parse la chaine en entree:
    vector<string> vs=Decompose(string(s),'%');

    //--- lecture des donnees
    double x = stod(vs[0]);
    string m=vs[1];

    //--- traitement
    double y=x+2;
    string m2="Salut "+m;

    //----fabrique la chaine de sortie
    string str_out;
    str_out = to_string(y) + "%" + m2; // on choisit le signe % pour
separer les données
    char * s_out = new char [str_out.length()+1];
    strcpy(s_out, str_out.c_str());
    return s_out;
    }
    }
    //============================
    int main()
    {
    //... test ...
    char s[]="3.14%fred faure";
    const char*u = f(s);
    }
```

Compilation :

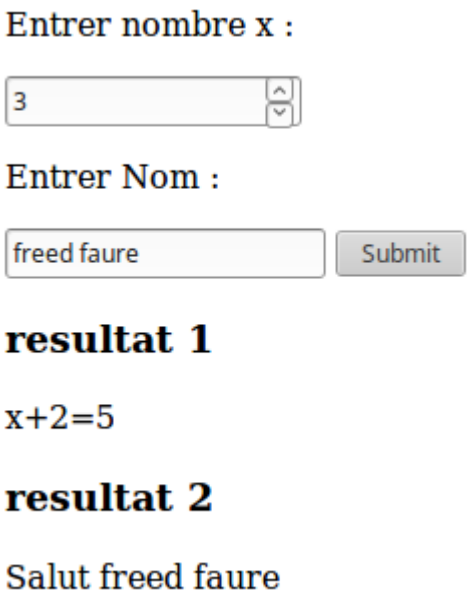
```
emcc test.cc -std=c++11 -O2 -o test.js -s EXPORTED_FUNCTIONS="['_f']"
```

Code html (fichier code.html) :

```
<!DOCTYPE html>
<html>
<body>
<p>Entrer nombre x :</p>
<input type="number" id="x">
<p>Entrer Nom :</p>
<input id="m">
<button type="button" onclick="F()">Submit</button>
<script src="test.js"></script>
<script>
function F()
{
var x = document.getElementById("x").value;
var m = document.getElementById("m").value;
var s = x.toString() + "%" + m.toString();
//........
f = Module.cwrap('f', 'string', ['string'])
var u = f(s);
document.getElementById("zone2").innerHTML = "debug";
var tu = u.split("%"); // on choisit le signe % pour separer les
données
document.getElementById("zone1").innerHTML = "x+2=" + tu[0];
document.getElementById("zone2").innerHTML = tu[1];
}
</script>
<h3>resultat 1</h3>
<p id="zone1"></p>
<h3>resultat 2</h3>
<p id="zone2"></p>
</body>
</html>
```

Execution : `firefox code.html`

Résultat : Lien : code.html.



Pour tester le code C++ seulement : Cela utilise la fonction main() de test.cc.
Compilation :

```
g++ test.cc -std=c++11 -O2 -o test
```

Execution :

```
./test
```

26.4 Exemples de "forms" en html

Dans l'exemple 26.3.1, on utilisé une entrée de type "button" qui permet de demander une chaine de caractères à l'utilisateur et de la transférer au script. Cette entrée est un cas particulier de "form" en html. Voici d'autres exemples.

Résultat : Lien : code.html.

Entrées

Field Set (groupe):

Une entrée texte (R1):

aa

Boutons radio (R2):

A B C

Check box (R3):

Slider (range) (R4):

Entree nombre (R5):

2

Selection de couleur (R6):

Choix multiple (R7):

France

Choix de fichier (R8):

Browse... GoPanda2

Bouton (action):

Submit

Résultats:

R1 = aa R2 =B R3=1 R4=6 R5=2 R6=#ff0000 R7=france R8=GoPanda2

Sorties

Barre de progrès (progress) (R4):

Une jauge (meter) (R4):

Code html :

```
<!DOCTYPE html>
<html>
<body>
<h2>Entrées</h2>
<br>
<fieldset>
<legend>Field Set (groupe):</legend>
Une entrée texte (R1): <br>
<input id="n">
<br>
Boutons radio (R2):
<br>
<form id="test">
<label><input type="radio" name="test" value="A"> A</label>
<label><input type="radio" name="test" value="B" checked> B</label>
<label><input type="radio" name="test" value="C"> C</label>
</form>
</fieldset>
<br>
Check box (R3):
<br>
<input type="checkbox" id="cb">
<br>
Slider (range) (R4):
<br>
<input type="range" id="r" name="points" min="0" max="10">
<br>
Entree nombre (R5):
<br>
<input type="number" id="nb" name="quantity" min="1" max="5">
<br>
Selection de couleur (R6):
<br>
<input type="color" id="col" name="favcolor" value="#ff0000">
<br>
Choix multiple (R7):
<br>
<select name="pays" id="pays">
<optgroup label="Europe">
<option value="france">France</option>
<option value="espagne">Espagne</option>
```

```
<option value="italie">Italie</option>
<option value="royaume-uni">Royaume-Uni</option>
</optgroup>
<optgroup label="Amérique">
<option value="canada">Canada</option>
<option value="etats-unis">Etats-Unis</option>
</optgroup>
<optgroup label="Asie">
<option value="chine">Chine</option>
<option value="japon">Japon</option>
</optgroup>
</select>
<br>
Choix de fichier (R8):
<br>
<input type="file" id="sel" name="img" multiple>
<br>
Bouton (action):
<br>
<button type="button" onclick="F()">Submit</button>
<br>
<hr>
<h2>Résultats:</h2>
<p id="zone"></p>
<script>
function F()
{
var x = document.getElementById("n").value; // recupere entree (string)
var text1 = "R1 = "+ x;
var form = document.getElementById("test");
var text2 = " R2 =" + form.elements["test"].value;
var text3 =0;
if (document.getElementById("cb").checked)
text3=1;
text3 = " R3=" + text3;
var r4 = document.getElementById("r").value;
var r5 = document.getElementById("nb").value;
var r6 = document.getElementById("col").value;
r6 = " R6=" + r6;
var r7 = document.getElementById("pays").value;
r7 = " R7=" + r7;
var r8 = document.getElementById("sel").value;
r8 = " R8=" + r8;
```

```
   document.getElementById("zone").innerHTML = text1 + text2 + text3
+ " R4=" + r4 + " R5=" + r5 + r6 + r7 + r8 ; // envoie resultat (string)
sur la zone
   var m = document.getElementById("m1");
   m.setAttribute("value", r4);
   var m2 = document.getElementById("m2");
   m2.setAttribute("value", r4);
   }
   </script>
   <h2>Sorties</h2>
   Barre de progrès (progress) (R4):
   <br>
   <progress id="m1" value="2" max="10">
   </progress>
   <br>
   Une jauge (meter) (R4):
   <br>
   <meter id="m2" value="2" min="0" max="10">
   </meter>
   <br><hr>
   </body>
   </html>
```

26.5 Appeler des fonctions js depuis c++ et des fonctions c++ depuis js

Code c++ :

```
   // -*- mode:C++ ; compile-command: "emcc test.cc -std=c++11 -O2
-o test.js -s EXPORTED_FUNCTIONS="['_f']"" -*-
   #include <emscripten.h>
   //========================
   extern "C"
   {
   void f()
   {
   //.. on passe du code js dans la chaine de caractere ".."
   emscripten_run_script(" var y=3; document.getElementById(\"zone1\").innerHTML
= y;");

   // on passe du code js dans (..)
```

```
   EM_ASM(
   alert('hello world!');
   );

   // on passe du code js dans (.. , x0 , x1 ) avec des variables qui
sont recuperees par $0,$1 etc
   double x0=1.2, x1=4.3;
   EM_ASM_(
   var x0= $0; var x1 = $1; document.getElementById("zone2").innerHTML
= "x0 x1 = " + x0 + x1;
   , x0, x1);
   }
   }
```

Code html :

```
   <!DOCTYPE html>
   <html>
   <body>
   <button type="button" onclick="F()">Submit</button>
   <script src="test.js"></script>
   <script>
   function F()
   {
   f = Module.cwrap('f', 'string', ['string'])
   f();
   }
   </script>
   <h3>resultat 1</h3>
   <p id="zone1"></p>
   <h3>resultat 2</h3>
   <p id="zone2"></p>
   </body>
   </html>
```

Resultat

26.6 Autres remarques

— Le préprocesseur reconnait __EMSCRIPTEN__ qui indique que emcc est le compilateur.

26.7 Graphisme

— Utiliser la librairie C SDL qui est utilisable en C++ et que emscripten peut exporter.
— Voir exemples sur /emsdk_portable/emscripten/master/tests/
— Utilisation de Root avec Javascript.
— Utilisatio de Qt avec emscripten.
— Librairie OpenGL qui est utilisable en C++ et que emscripten peut exporter en WebGL. Voir GEOGRAM.

26.8 Notes musicales en html5

Voir : Vexflow. ou Alphatab ou midi.js
Voir ,
— Midi :
  — web-midi-api
  — tutorial
— Audio :
  — webaudio
  — http://www.keithwhor.com/music/
  — http://blog.teamtreehouse.com/building-a-synthesizer-with-the-web-audio-api

26.9 Utilisation des capteurs d'un smartphone

Voir capteurs en javascript.

26.10 Utilisation de fichiers

Supposons que dans le répertoire courant il y a un fichier "file.txt" contenant "abcd".

Code c++ :

```
   // -*- mode:C++ ; compile-command: "emcc test.cc -std=c++11 -O2
-o test.js -s EXPORTED_FUNCTIONS="['_f']" --preload-file file.txt "
-*-
   #include <fstream>
   using namespace std;
   #include <emscripten.h>
   //========================
   extern "C"
   {
   void f()
```

```
    {
    //... lecture depuis fichier
    ifstream g("file.txt");
    string res;
    g>>res;

    //.. affiche le contenu
    string s = "document.getElementById(\"zone1\").innerHTML = \"" +
res + "\";";
    emscripten_run_script(s.c_str());
    }
    }
```

Commande de compilation `emcc test.cc -std=c++11 -O2 -o test.js -s EXPORTED_FUNCTIONS=
--preload-file file.txt`

Code html :

```
    <!DOCTYPE html>
    <html>
    <body>
    <button type="button" onclick="F()">Submit</button>
    <script src="test.js"></script>
    <script>
    function F()
    {
    f = Module.cwrap('f')
    f();
    }
    </script>
    <p id="zone1"></p>
    </body>
    </html>
```

Resultat

Chapitre 27

Création automatique de Makefile et d'interface graphique utilisateur (GUI)

Dans ce chapitre, je présente un logiciel que j'ai développé pour usage personnel au départ. Il écrit automatiquement :

— Un fichier Makefile pour un projet c++

— Du code c++ correspondant à des fenêtres de commandes GUI, utilisant la librairie root.

Cela est très utile lorsque on développe un projet, et permet de gagner du temps.

27.1 Mode d'emploi

ce programme sert a creer un fichier makefile automatiquement.
en partant d'un fichier "`nomfichier.cc`" qui contient le `main()`.
Pour l'utiliser, il faut

1. écrire le fichier makef.config
2. lancer : makef repertoire nomfichier.cc makef.config <nom_executable>
3. ensuite pour compiler (lancer le Makefile), il
   il faut faire :
   `make clean` (pour detruire les fichiers .o precedants)
   `make all`
   voir par exemple le script "compp"

27.1.0.1 Fonctionnement final des commandes

— Une fenetre de commande GUI est lancée dans un autre thread (process) "com" et utilise root. Cette fenetre de commande contient tous les widgets des variables (demandees) des classes que utilise le projet main().

— Communication :

- Si un widget est activé (changé), le process "com" change les variables du main avec une protection prealable de type mutex.
- depuis le process "main", on appelle explicitement p_com->Met_a_jour() pour que l'affichage de "com" corresponde aux variables de "main".
- optionnellement, le process "com" appelle Met_a_jour() de facon periodique (tous les 0.1 sec), pour eviter cet appel explicite.
- Dans le programme "main", avant/apres d'acceder aux variables partagees, il faut mettre/ enlever le mutex mtx : mtx.lock() ; mtx.unlock() ;

— -si besoin, on a acces aux fenetres de commandes par le pointeur Com *p_com ;

27.1.0.2 sortie du programme makef

fichiers c++ : `com.cc, com.h` dans le meme repertoire que fichier.h qui sont des classes c++ d'un panneau de commandes

27.1.1 Instructions pour placer les widgets

La fenetre de commande peut contenir les zones suivantes :

— un menu,
— une zone commune : ZC
— des zones de tabs : ZT1, ZT2, ...

Les widgets sont placés à la suite horizontalement dans la zone demandee.

— **Retour à la ligne** : On peut revenir à la ligne si on commence par l'instruction "nl" :
ex :
```
double x1=4; // make_gui = nl N(ZT("tab1"), "x1=")
```
— **Texte d'aide** : On peut rajouter un texte d'aide (qui sera un "Tip") :
ex :
```
double x1=4; // make_gui = N(ZT("tab1"), "x1=") help = ''texte d'aide''
```
— **N :** Numerique entre/sortie pour int, double
ex :
```
int a; // make_gui = N(ZC,"a=")
double x1=4; // make_gui = N(ZT("tab1"), "x1=")
```
— **T :** Texte entre/sortie pour string
ex :
```
string text = "hello" ; //make_gui = T(ZT("tab1"))
```
— **HS** : Horizontal Slider entre/sortie pour int/double
ex :
```
double x = 1 ; // make_gui = HS(ZT("tab1"), 0, 5)
```
— **VS** : Vertical Slider : entre/sortie pour int/double
— **PB** : Progress Bar : sortie pour int,double
ex :
```
double y = 5 ; // make_gui = PB(ZT("tab2"), 0,5)
```

— **C** : Check entree/sortie pour int x=0,1
ex :
`int opt = 1 ; //make_gui = C(ZT("tab2"), ":opt")`
— **L** : Liste entre/sortie pour int x \in [0,N-1]
ex :
`int choix = 2 ; //make_gui = L(ZT("tab2"), {"a","b","c","d"})`
— **B** : Bouton d'appel pour fonction void f()
`void f() ; // make_gui = B(ZC,"C1_f")`
— **M** : Menu d'appel pour fonction void f()
`void g() ; // make_gui = M("menu2","C1_g")`
— projet : - H1 : Histogramme 1D pour pour vector<int>, vector<double>, vec
les valeurs de ces widgets sont en permanence couplées à la variable c++.

Pour cela il y a un pointeur pour chaque classe.
ex : pointeur `p_C1` pour la classe `C1`
dans la fonction main(), il faut rajouter (avec la typo) :

```
#includ "com.h"
Com *com= new Com(&c1,&c2);
//thread t(Lance_com, &c1, &c2); // lance fenetre de commande dans autre
thread. Donne pointeurs sur objets
```

On met une typo car cela ne doit pas empecher le fonctionnement de fichier.h sans les commandes
(on doit pouvoir desactiver en option la compilation de com.cc, com.h)
et les fichiers existants ne doivent pas etre modifiés.

Chapitre 28

A propos des licences

Lire licences GPL.

Le témoignage d'un programmeur : Avec la licence tu donnes legalement le droit aux gens de utiliser le soft ; sinon, ils ont juste le droit de le télécharger. Donc il faut toujours une licence.

Elle doit être dans l'entête de chaque fichier. Si le texte de la license est trop long, dans l'entête du fichier on met juste un paragraphe type indiquant comment obtenir la license ; alors celui-ci est mis à la racine dans un fichier COPYING ou LICENSE.

Fais bien attention à ne rien altérer (meme pas les espace et les allés à la ligne), sinon on ne peut plus utiliser des outils (grep et diff) pour savoir quelle est la license ; et surtout on se demandera pourquoi le texte a été altéré.

Pour que le soft soit libre, en pratique il faut trouver un texte de license existant, connu et éprouvé. Sinon l'utilisateur ne sait pas exactement quel droits il a (au sens juridique) et doit payer un avocat si il veut être sur. Donc en pratique c'est comme si il n'y avait pas de license pour le commun des mortels.

Pour de l'open-source & libre il y a 2 types de licences :

1. Celles où l'auteur donne le soft et ne veut rien en retour ; un peu comme quand on donne un cadeau : on ne se soucie plas si il finit utilisé vendu ou à la poubelle. Ce sont les licenses ISC, BSD, X, MIT, ... Elles sont courtes (puisqu'elle ne disent pas grand chose) on les retrouve en entête des .c et .h.
2. GNU public license (et autres semblables peu connus). Dans ce cas l'auteur demande des choses à l'utilisateur, typiquement de ne pas revendre le soft et publier les modifs qu'il y fait, etc. Le texte est en conséquence assez long et ibitable. Mais des juristes s'y sont déjà penchés et "il est bon".

On trouve aussi des license de grands industriels, typiquement 30 pages de charabia de juriste que personne ne comprend ; elles sont probablement piegés.

On trouve aussi des licences que des gens ont écrit en toute naiveté et qui n'ont pas de valeur (ie un bon avocat pourrait leur faire dire tout et son contraire).

Perso, j'attends rien en retour, donc j'utilise la license ISC pour toute publication.

Bibliographie