Université Joseph Fourier de Grenoble

Didacticiel
Bases de programmation en C++

Dernière modification fichier cours_1/cours_1.lyx : 18 septembre 2001

Auteurs :

Frédéric Faure (mailto:frederic.faure@ujf-grenoble.fr)
Jonathan Ferreira(mailto:jonathan.ferreira@ujf-grenoble.fr)
Philippe Peyla(mailto:philippe.peyla@ujf-grenoble.fr)

Table des matières

1 Introduction **2**

2 Que faire avant de programmer ? **3**
Quelques notions de qualité en informatique . . . 3

3 Le tout début. **4**
Quelques remarques : . . . 4

4 L'affichage. **5**
Quelques remarques : . . . 5
Remarques plus techniques . . . 5
Exercice . . . 6

5 Lire le clavier et afficher à l'écran. **6**
Remarque . . . 6
Exercice . . . 6

6 Déclaration et affectation des objets **6**
Les déclarations d'objets de base : . . . 6
Initialisation d'un objet de base . . . 7
Précision à l'affichage . . . 7
Conversions de classe . . . 8

7 Les instructions de base **8**
La boucle : for(départ ;tant que ; incrémentation) {instructions} . . . 8
Comment écrire une condition . . . 8
Le : do { instructions} while (condition) ; . . . 9
Le while (condition) { instructions } ; . . . 9
L'instruction if (condition) {instructions} else {instructions} ; . . . 10

8 Les fichiers **11**
Ecriture de données dans un fichier : classe ofstream . . . 11
Lecture de données depuis un fichier : classe ifstream . . . 11

9 Les tableaux **12**
déclaration . . . 13
Affectation . . . 13

10 Les fonctions **14**
Exemple de fonction qui ne renvoit rien. . . . 14
Exemple de fonction qui renvoit un objet . . . 14

11 Paramètres des fonctions par référence **15**
Exercice . . . 15

12 La surcharge des fonctions. **15**

13 Les pointeurs **16**
Déclaration et affectation d'un pointeur . . . 16
Allocation dynamique de la mémoire . . . 17

14 Création d'une classe **18**
Introduction . . . 18
Exemple : . . . 19
Commentaires : . . . 20
Exercices . . . 20
Suite du cours sur la création de classes . . . 21

15 L'héritage de classes **21**
La petite histoire : . . . 21

16 Graphisme avec la librairie ROOT **22**
important . . . 22
Exemple de départ . . . 22
Commentaires : . . . 23
Exercice : . . . 23
Exercice : . . . 23
Remarque . . . 23
Utilisation des pointeurs avec ROOT . . . 24

17 Micro-projet **25**
La fractale de Newton . . . 25
Un morpion pour 2 . . . 25
La fractale du dragon . . . 25

1 Introduction

- Ce didacticiel vous permettra d'appréhender le langage C++ et ses avantages. 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 que l'on a fabriqué soi-même. Par exemple, il existe la classe des nombres complexes (Complex). 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 Complex. On pourra alors directement écrire `a=b+c` ou n'importe quelle autre expression. On peut alors former une classe de matrice complexe, déclarer trois objets `A`, `B` et `C` appartenant à cette classe et écrire `A=B+C` ou n'importe quelle autre expression. Vers la fin de ce cours, nous aborderons l'aspect fabrication de classe. Pour l'instant nous allons apprendre la syntaxe du C++ qui s'apparente à celle du C.
- Noter que le langage C++ est très proche du langage Java qui est en plein essort. Nous préférons le C++ pour le moment car le langage Java est encore trop neuf pour le moment : il existe un grand nombre de librairies ou classes déja existantes en C/C++, spécialisées pour le calcul scientifique, qui n'existent pas en Java.

– **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 remplace peu à peu le langage *fortran*. D'un autre côté, les entreprises cherchent en général des gens ayant des compétences en visual basic, en création de pages web, puis en java, puis en C(++). Mais tout dépend du domaine précis, plus les compétences sont pointues plus c'est du C(++), et plus c'est généraliste plus ça s'oriente vers java et visual basic. Le problème principal de java, c'est que c'est un langage controlé par Sun que Microsoft essaye de modifier illégalement, on ne peut donc pas parler de standard ouvert comme le langage C ; rien ne dit que java va vraiment décoller chez la majorité des développeurs par rapportà d'autres langages (C, delphi et autres). Quant au domaine scientifique, C propose actuellement bien plus de possibilités ; Java sert surtout pour des interfaces et des applets web.
– De bonnes références de livres :
Le langage C de B.W. Kernighan et D.M. Ritchie edition Masson
Programmer en C++ de Claude Delannoy, editions Eyrolles

2 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. Vous pourrez avoir besoin de ces conseils dans quelques leçons, lorsqu'on vous demandera d'écrire un programme de votre propre conception.

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.
3. Tracer sur un papier un **organigramme** illustrant le fonctionnement de votre programme. Vérifiez que l'ordre d'éxecution 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 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.

Voilà, vous pouvez maintenant en toute confiance traduire votre travail dans n'importe quel langage informatique. Les erreurs de structuration logique ayant été éliminées, il ne vous restera plus qu'à corriger les inévitables erreurs de syntaxe et de frappe.

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 (relativement) rapidement 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.

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.

3 Le tout début.

Un petit programme pour commencer :

Voici un petit programme C++ qui additionne deux nombres entiers a et b et affecte le résultat dans c. La classe `int` permet justement de stocker des nombres entiers. (Vient de integer en anglais).

```
/*
------------------------
ceci est
votre premier programme
--------------------------
*/
void main( ) // entête du programme principal
{
// début
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 la classe int.
c=a+b; // affectation : c prend la valeur a+b c'est à dire 3
// fin
}
```

Quelques remarques :

1. Le **point virgule** sépare les instructions.
2. 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 premier venu. Une autre possibilité est de commencer le commentaire par /* et de finir par */, le commentaire peut tenir alors sur plusieurs lignes.
3. Le début et la fin d'un bloc d'instructions sont respectivement "{" et "}"
4. "`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. Lorsque vous lancez votre programme, son éxécution commence toujours par cette fonction principale.

5. "`void`" signifie "vide", cela veut dire que main ne renvoie aucun paramètre.

Ces notions de fonction seront plus claires quand nous aborderons les fonctions à la leçon Les fonctions.

4 L'affichage.

Nous avons affecté la somme de deux entiers a et b dans un autre entier c. Mais nous n'avons pas affiché le résultat à l'écran. Pour cela on utilise l'objet `cout`. Cet objet représente l'écran. C'est un objet de la classe `ostream` (classe standard du C++) qui est définie dans le fichier : `iostream.h`. Il ne faut donc pas oublier d'inclure ce fichier avec la commande `#include` en début de programme pour que, lors de la compilation, l'ordinateur puisse savoir ce que veut dire `cout`.

```
#include <iostream.h>
void main( ) // entête du programme principal
{
// début
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 la 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;

// fin
}
```

Quelques remarques :

1. Le programme va afficher :
   `la somme de 1 et 2 vaut 3`
2. Les caractères entre " " 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. Le terme `cout` symbolise l'écran. Les signes << sont évocateurs : on envoit ce qui suit vers l'écran. `endl` signifie que l'on va à la ligne (fin de ligne).

Remarques plus techniques

1. `iostream.h` 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).
2. `cout` est **un objet** C++ de la **classe** ostream. Cet objet est associé à l'écran comme expliqué ci-dessus. Le signe << est un **opérateur** de cette classe à qui on a donné le sens précis d'afficher à l'écran. Tout cela est contenu dans le fichier iostream.h qui est lui même un programme C++. L'existence de cette classe simplifie donc la programmation pour les suivants. C'est l'esprit du C++. Vous apprendrez à écrire vous même des classes et des opérateurs par la suite.

Exercice

1. Recopier ce programme et exécuter le.
2. \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

5 Lire le clavier et afficher à l'écran.

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.h>
void main() // entête du programme principal
{
// début
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;

// fin
}
```

Remarque

1. Cette fois ci vous avez compris que 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 de 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 tout de suite à l'écran.

Exercice

Refaire le même travail que dans la leçon précédente.

6 Déclaration et affectation des objets

Les déclarations d'objets de base :

Pour stocker une information dans un programme, on utilise un objet qui est symbolisée 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, ou nombre à virgule, nombre complexe, ou série de lettres, ou matrice,...).

Voici les différentes classes de base qui existent en C++ :

Déclaration : `classe objet ;`	signification	Limites
`int a ;`	entier	-32767 à +32767
`float c ;`	réel	$\pm 10^{\pm 37}$ à 10^{-5} près
`double d ;`	réel double précision	$\pm 10^{\pm 37}$ à 10^{-9} près
`char e ;`	caractère	
`char *f ;`	chaîne de caractère	

Remarques :

1. Quand utiliser `float` plutôt que `double` ? puisque ce dernier est de toutes façons plus précis. Réponse : Stocker un objet de la classe `double` dans la mémoire de l'ordinateur (ou dans un fichier) prends plus de place. Par contre l'ordinateur calcule aussi vite avec `float` qu'avec `double`, car les processeurs actuels sont structurés pour cela. Donc si économiser de la place mémoire est crucial pour votre programme, et que la précision n'est pas nécessaire, (mais cela est rare) il est préférable d'utiliser `float`, sinon utilisez (en général) `double`.Ayez le reflexe `double`...
2. Les classes ci-dessus sont les classes de base standard en C++ qui existent en 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. Vous pouvez utiliser d'autres classes plus élaborés (qui ne sont plus des classes de base), comme les entiers à précision ilimitée, les complexes, les vecteurs, les matrices,.. à condition cette fois ci d'inclure le fichier `.h` approprié au début du programme.

Initialisation d'un objet de base

On peut déclarer un objet et l'initialiser en même temps, de différentes manières :

```
int i=0;
float Pi=3.14;
double pi(3.1415),x1(3),x2(6);
double y(x1);
char mon_caractere_prefere = 'X';
char * texte=que dire de plus ?;
```

i est un `int` qui vaut 0, `Pi` un float qui vaut 3.14, `pi`est un `double` qui vaut 3.1415, etc..

Remarques

1. 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 _.
2. On peut initialiser un objet avec des parathèses comme `x1(3)`. On peut initialiser un objet avec un objet déjà existant comme `y(x1)`.
3. Un caractère est entre le signe ', comme `'X'`.Un texte (chaîne de caractère) est entre le signe .
4. On pourra bien sûr modifier ces valeurs dans la suite du programme.

Précision à l'affichage

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

```
double c=3.14159;
cout <<setprecision(3) << c << endl;
```

Affichera : 3.14

Conversions de classe

On peut affecter un objet à partir d'un objet d'une autre classe, **lorsque cela a un sens** (i.e. lorsque çà a été prédéfini) **.** On parle de conversion.

```
int i;
float f(3.14),g;
i= int(f); // convertion de float en entier. i contiendra 3.
cout <<i <<endl;
g=i; // conversion de int à float.
cout <<int('A'); //Conversion de char à int. Affichera 65 à l'écran qui est le code ASCII de A
cout <<char(65) <<endl; //Conversion de int à char. Affichera A à l'écran qui est le caractère qui a le code 65.
```

Remarques

Certaines conversions (comme celle de int à float), sont naturelles et ne font pas perdre de l'information. Dans ce cas, il n'est pas nécéssaire de préciser la classe de destination : on peut écrire `g=i;` au lieu de `g=float(i)`.

7 Les instructions de base

La boucle : for(départ ;tant que ; incrémentation) {instructions}

La syntaxe de la boucle *for* dans l'exemple ci-dessous signifie :
Fait les instructions en partant de i=10, fait i=i+2 et repête les instructions,..,tant que i<=30.
Les instructions à répéter se trouve dans le bloc `{...}` qui suit la ligne avec `for`.

```
#include<iostream.h>
void main( )
{
int i;
int j;
for(i=10;i<=30;i=i+2)
 {
  j=i*i;
  cout <<i <<"\t" <<j <<endl;
 }
}
```

Ce programme produira :
10 100
12 144
14 196
etc...
30 900

Comment écrire une condition

On a utilisé ci-dessus la condition `i<=30`. Voici la syntaxe pour écrire une condition plus générale :

Les opérateurs de comparaison :

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

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

Les opérateurs logiques

Signification	symbôle
et logique	&&
ou logique	||
non logique	!

Remarque

Lorsqu'une condition est évaluée comme `i<=30`, la valeur rendue est de classe entier (int). Elle est différente de 0 si la condition est vraie et 0 sinon.

Le : do { instructions} while (condition) ;

signifie **fait le bloc d'instructions tant que la condition est vraie.**

```
#include<iostream.h>
void 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); // i < MAX est la condition
}
```

produira :

1 1
2 4
etc...
10 100

Le while (condition) { instructions } ;

```
#include <iostream.h>
void 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;
}
```

Produira :

1 1
2 4
etc...
10 100

L'instruction if (condition) {instructions} else {instructions} ;

```
#include<iostream.h>
void main()
{
int i(5),j;
char test='N';
while(test!='O') // tant que la valeur de test est différente de 'O'
 {
  cout <<"entrer un nombre entre 1 et 10 " <<endl;
  cin >>j;
  if(i==j)
   {
   cout <<"Bravo vous avez trouvé le nombre mystérieux!" <<endl;
   test='O';
   }
  else
   cout <<"Non ce n'est pas le bon nombre" <<endl;
 }; // fin du while
} // du programme
```

Exercice

Faire un programme qui au départ choisit un nombre au hasard entre 0 et 1000 (se servir de la section suivante), puis demande à l'utilisateur de le trouver, en répondant "trop grand" ou "trop petit" à chaque essai. L'utilisateur a droit a un nombre limité d'essais.

Pour générer un nombre entier p aléatoire, entre 0 et N inclus. on écrit :

En début de programme :

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

Puis dans le programme, l'instruction suivante ne doit apparaitre qu'une seule fois ; elle permet d'initialiser les tirages à partir de la date.

```
srand(time(NULL));
```

Puis au moment de choisir un nombre au hasard :dans le programme

```
p=rand()%(N+1);
```

Explications : `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 choisit par `rand()` par `N+1`. C'est donc un entier compris entre 0 et N (inclus), ce que l'on veut.

8 Les fichiers

Jusqu'à présent, nous avons lu au clavier et affiché des données à l'écran. On peut aussi écrire des données dans un fichier. Pour cela il faut utiliser des fonctions qui manipulent les fichiers. Ces fonctions sont regroupées dans le fichier `fstream.h` 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..

Ecriture de données dans un fichier : classe 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.h>
#include<fstream.h> // utilisation des fichiers
void main( )
{
 ofstream f(hector.txt); // ouvre le nouveau fichier en ecriture. On lui
associe l'objet : f
 f <<3.1514 <<endl; // permet d'ecrire dans le fichier.
 f.close(); // fermeture du fichier
}
```

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 intermédiare de la classe `ofstream`.Cet objet a été initialisé avec le nom du fichier, et pour écrire dans le fichier, **c'est la même syntaxe que pour écrire à l'écran** (avec `f` à la place de `cout`).
3. L'ouverture du fichier s'est faite par l'initialisation de l'objet f avec le nom du fichier donné sous la forme d'une chaîne de caractères. On peut donc passer un objet intermédiare (tableau de caractères) de la façon suivante :
   `char * nom=hector.txt`
   `ofstream f(nom);`

Exercice

Ecrire le programme précédent et vérifier l'existence du fichier ; puis afficher son contenu sous UNIX par la commande : `more hector.txt`.

Lecture de données depuis un fichier : classe ifstream

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


```
#include<iostream.h>
#include<fstream.h> // utilisation des fichiers
void main()
{
ifstream g(hector.txt);// ouvre un nouveau fichier en lecture. On lui
associe l'objet : g
double pof;
g >>pof; //on lit ce qu'il y a au début du fichier, et on le copie dans
l'objet pof
cout <<pof <<endl; // on écrit à l'écran le contenu de pof
g.close(); // fermeture du fichier -lecture g
}
```

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 `pof`.
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. Lorsque l'on a effectué l'opération `g.close()`, g est un objet de la classe `ifstream` 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.
4. La lecture `g >>pof;`depuis le fichier est similaire à la syntaxe de lecture `cin >>pof`depuis le clavier.

Exercice

Créer avec un éditeur un fichier appelé `notes.txt` qui contient 20 notes comprises entre 0 et 20. Ecrire un programme qui lit ce fichier et affiche les notes à l'écran sous forme tabulée. On va réutiliser ce fichier plus tard.

Exercice

Que fait le programme suivant ? (essayer)

```
#include<iostream.h>
#include<fstream.h>
void main()
{
char c;
ifstream original("hector.txt");
ofstream copie("xerox.txt");
while (original >>c) copie <<c;
}
```

Remarquez que `original >>c` en plus de faire l'opération de lecture, renvoit un résultat (int) : 0 si échec, autre si lecture réussie. Et cela est bien utile dans ce programme.

9 Les tableaux

Un tableau est suite d'objets de même la classe.

déclaration

La façon de déclarer un tableau est la suivante :

`int tab1[100] ;` // tab1 est un tableau de 100 cases contenant chacune une objet de la classe entier

`float tab2[20] ;` // tab2 est un tableau de 20 cases contenant chacune une objet de classe réel

`char tab3[10] ;` // tab3 est un tableau de 10 cases contant chacune une objet de classe char. On dit aussi que c'est une chaîne de 10 caractères

Attention : la taille du tableau doit être un nombre constant ; ça ne peut pas être la valeur d'un objet (int). Si l'on veut créer un tableau de taille variable, voir la section pointeurs.

Affectation

Pour mettre la valeur 3 dans la case numéro 0, on écrit naturellement :

```
tab1[0]=3;
```

Attention : si on déclare `int tab1[N]`, **les N cases sont numérotées de 0 à N-1** .

Si on se trompe, si on écrit dans une case hors de limites, cela peut être la cause du plantage de votre programme, et créer *un bug de l'an 2000*. Sachez que la plupart des bugs que vous risquez de créer proviendront de ce problème.

Exercice

Complétez les signes ? dans le programme suivant

```
// pour calculer le produit scalaire de deux vecteurs
#include<iostream.h>
const dim=3;
void main()
{
double v1[dim], v2[dim],s;
v1[?]=1; v1[?]=2; v1[?]=3;
v2[?]=3; v2[?]=-2; v2[?]=1;
for (i=?;i<=?;i=i+1)
 s=s+v1[i]*v2[i];
cout <<"le produit scalaire est :" <<s <<endl;
}
```

Remarque

Dans le programme ci-dessus, on a déclaré une constante `dim`. Cela permet de changer facilement la taille des tableaux : il suffit de changer la valeur de `dim`.

Exercice

Ecrire un programme qui lit les notes dans le fichier `notes.txt` et les remet dans l'ordre croissant, dans le même fichier.

10 Les fonctions

Une fonction est un petit sous programme qui utilise (ou pas) certains paramètres que lui passe le programme principal (ou la fonction qui l'appelle). Une fonction peut modifier un objet, remplir un tableau, écrire une information à l'écran...

Une fonction peut renvoyer une valeur d'une certaine classe ou ne pas renvoyer de valeur du tout.

Exemple de fonction qui ne renvoit rien.

on met `void` : (=vide en anglais).

```
#include <iostream.h>
//========declaration de la fonction carre============
void carre(int i)
{
int j ; // declaration d'un objet local
j=i*i;
cout <<"le carré de " <<i <<" est " <<j <<endl;
}
//====declaration de la fonction principale ====================
void main()
{
int x;
cout <<" entrer x " <<endl;
cin >>x;
carre(x); // appel de la fonction carre
}
```

Remarques :

1. 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 n'est pas connu ailleurs : On ne peut pas l'utiliser dans la fonction `carre`.
2. La fonction main appelle la fonction `carre` et lui passe un parametre qui est l'objet x choisi par l'utilisateur.
3. La déclaration de la fonction `carre` montre que cette fonction prend un paramètre (un objet de la classe int) et ne renvoit rien (à cause du préfixe `void`). De même la fonction `main` ne prend aucun paramètre, et ne renvoit rien.

Exemple de fonction qui renvoit un objet

```
#include<iostream.h>
//======declaration de la fonction carre ==================================

int carre(int i)
{
int j;
j=i*i;
return(j); //on renvoie le résultat au programme principal
}
//===declaration de la fonction principale =====
void main()
```

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

Remarques

La seule différence avec l'exemple précédent est que la fonction `carre` renvoit son résultat, par un objet de la classe int. Pour cela on utilise l'instruction `return`.Et pour appelé cette fonction, on utilise une la syntaxe `y=carre(x)`si bien que le résultat est tout de suite stocké dans l'objet `y`.

11 Paramètres des fonctions par référence

Quand le programme principal fournit une objet à une fonction, celle-ci peut éventuellement vouloir modifier le contenu de la 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 affecte.

Pour cela, dans la déclaration des paramètres, on met le signe & devant les paramètres pouvant être modifié par la fonction :

```
#include<iostream.h>
//====== fonction permute ==================
void permute(int& i,int& j)
{
int t=i; // stockage temporaire
i=j;
j=t;
}
//======== fonction main ===================
void main()
{
int a=10,b=5;
cout <<a <<" " <<b <<endl;
permute(a,b);
cout <<a <<" " <<b <<endl;
}
```

Exercice

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

12 La surcharge des fonctions.

Un des aspects les plus puissants 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 qui permet au langage de distinguer qu'elle est la fonction à appeler, c'est les paramètres demandés lors de l'appel de la fonction.

Exemple :

```
#include<iostream.h>
//-------------------------------------------
void func(int i)
{
cout <<"fonction 1 appelée" <<endl;
cout <<" paramètre = " <<i <<endl;
}
//-------------------------------------------
void func(char *s,int i)
{
cout <<"fonction 4 appelée " <<endl;
cout <<" paramètre = " <<s <<endl;
cout <<" paramètre = " <<i <<endl;
}
//---------------------------------------------
void main()
{
func(10);
func("Chaîne",4);
}
```

13 Les pointeurs

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 une "case" en mémoire de l'ordinateur, permettant de stocker un nombre entier. Cette case s'appelle 'i'.

Bien sûr, cette case se trouve quelque part dans la mémoire de l'ordinateur. Elle a un certain emplacement, caractérisée par son adresse, appellée son **pointeur**.

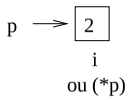
Il faut donc retenir que **pointeur d'un objet** signifie **adresse d'un objet** dans la mémoire de l'ordinateur.

On peut avoir accès à l'adresse de la "case" i en faisant :

```
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 (désignant une case mémoire qui est sensée contenir un entier).

Le signe `&i` signifie l'adresse de l'objet i. Donc la troisième ligne met dans p l'adresse de l'objet i.



Pour afficher le contenu de l'objet i on a maintenant deux possibilités :

```
cout <<i;
```

ou :

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

qui est équivalent car :
(*p) signifie le contenu de la "case" où pointe p.
Pour modifier le contenu de l'objet i on a maintenant deux possibilités :

```
i=5;
```

ou :

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

qui est équivalent.
Attention : avant d'effectuer l'opération d'écriture : `(*p)=5`; il faut être sur que l'adresse du pointeur correspond à une "case" existente.
Vous avez donc compris que avant d'écrire, il faut prendre le soin de réserver de la place mémoire.

Allocation dynamique de la mémoire

Tant qu'à faire, on peut réserver une suite de n cases mémoires, par l'instruction :

```
int n;
cout <<entrez n <<flush;
cin >>n;
float *p; //on déclare le pointeur p
p=new float[n]; // on réserve n cases mémoire de la classe float
```

Cela s'appelle une **allocation dynamique** de la mémoire, car elle se fait au cours de l'éxécution du programme.
A près cela, **p pointe sur la première case réservée :** on peut légitimement écrire dans cette première case par l'instruction :

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

ou (ce qui est équivalent)

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

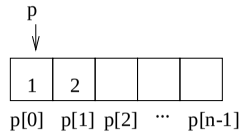
On peut de même écrire dans la case suivante par :

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

ou

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

etc... jusqu'à `p[n-1]`.



Remarquer qu'il y a n cases, numérotées de 0 à 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
```


Remarques

1. Lorsqu'on a utilisé des tableaux, vous avez compris qu'il s'agissait de pointeurs. Dans cette leçon, on a vu que l'on peut réserver 6 cases en mémoire par l'instruction :

   ```
   float p[6];
   ```

   Il s'agit d'une allocation de mémoire mais dite statique (non dynamique) car le nombre de case est fixé, et ne peut être variable.

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**. On utilise souvent la déclaration :

   ```
   char *chaine=toto;
   ```

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

Exercice

Modifier le programme de la leçon tableaux sur le produit scalaire de deux vecteurs, en créant maintenant ces deux vecteurs de façon dynamique : le programme demande à l'utilisateur la taille n des vecteurs, et les composantes des vecteurs.

14 Création d'une classe

Introduction

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 dans les fichiers `Complex.h`, `Complex.cc`). Avec cette classe, on peut déclarer des variables complexes par `Complex z;` 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.
Point de vocabulaire : on parle de **la classe** `Complex`. Lors d'une déclaration d'un nombre complexe par `Complex z;` on dit que la variable `z` est **un objet** de la classe `Complex`.
La notion de **classe** est fondamentale en C++. On dit que le C++ est **un langage orienté objets**.
Par rapport à des langages traditionnels comme le *Pascal* ou le *C*, la *programmation objet* est une nouvelle façon de programmer, car on va créer une classe (ou utiliser une classe existante) pour chaque concept qui semble ressortir du problème que l'on traite. Par exemple si l'on résoud un probleme d'algèbre linéaire, ce sera bien de programmer au préalable une classe de vecteurs et de matrices avec toutes leurs opérations standard. Le programme principal sera ainsi beaucoup plus simple en apparence, puisque on pourra écrire les opérations sous la forme W=M*N*V par exemple où V,W sont des vecteurs et M,N des matrices.
Un avantage de la programmation objet par rapport à la programmation habituelle, est donc que le code est **mieux structuré** et **plus lisible**. Un autre avantage est que le code est plus facilement **réutilisable** (Vous pouvez vous confectionner une bibliothèquee de classes usuelles et/ou trouver des classes sur le web prêtes à l'emploi).
Dans cette section, on va apprendre à écrire soi-même une classe, et l'on prendra l'exemple des vecteurs et des matrices.

On va découvrir progressivement les différentes possibilités. Pour plus de détails, on vous conseille tout ouvrage sur le C++ comme :
Programmer en langage C++ de Claude Delannoy (edts Eyrolles).

Exemple :

Un vecteur de R^3est définit par ses trois composantes (x, y, z).
Voici le code de départ pour définir un vecteur en C++. Lisez ensuite les commentaires.

```
#include <iostream.h>
#include <math.h>
// ===============déclaration de la classe ===============
class Vecteur
{public :
//---- variables membres
    double x,y,z; // composantes du vecteur
//--- constructeurs
    Vecteur();
//--- destructeur
    ~Vecteur();
//----- fonctions membres
    double Norme();
};
// =====code des fonctions membres =======================
//-- le constructeur---
Vecteur : :Vecteur()
{
    x=0; y=0; z=0;
    cout <<vecteur construit... <<endl;
}
//-- le destructeur---
Vecteur : :~Vecteur()
{
    cout <<vecteur détruit.... <<endl;
}
//--la fonction Norme---
double Vecteur : :Norme()
{
    double n=0;
    n=x*x+y*y+z*z;
    return sqrt(n);
}
```

On l'utilise cette classe Vecteur dans le programme principal de la sorte :

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


Commentaires :

- il y a deux parties pour définir la classe : la **partie déclaration de la classe** et la **partie code des fonctions membres**. Ainsi, la syntaxe d'appel d'une fonction membre (comme `Norme()`) se trouve dans la partie déclaration de la classe mais le code se trouve dans la partie code des fonctions membres. Remarquez le préfixe `Vecteur : :` qui se trouve pour préciser la classe d'appartenance de la fonction membre.
- Remarquez dans le programme la présence ou non de points virgule ;.
- Dans la partie déclaration, les **variables membres** montrent qu'un vecteur est défini par trois variables réelles x, y, z de type double.
  Au cours du programme principal, on a déclaré un objet `v` de la classe `Vecteur`, et on accède à ses variables membres *par un point*. Ainsi `v.x=1;` met la valeur 1 dans la variable membre x.
  A l'intérieur d'une fonction membre comme `Norme()` par contre les variables membre `x,y,z` s'accèdent directement. 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** `Vecteur()` est appelée chaque fois qu'un objet Vecteur est déclaré au cours du programme. Dans notre fonction main(), ce sera à la première ligne `Vecteur v;` Cette première ligne appelle donc le code du constructeur, qui se trouve dans la partie code. On peut voir que l'effet produit est de mettre à zéro les variables x, y, z et d'afficher le texte vecteur construit.
  Comme son nom l'indique, le constructeur sert à initialiser le nouvel objet créé.
  Rem : Le constructeur *doit toujours porter le nom de la classe* en question (ici Vecteur).
- la fonction membre **destructeur** `~Vecteur()` est appelée chaque fois qu'un objet Vecteur est détruit au cours du programme. Dans notre fonction `main()`, l'objet `v` est détruit tout à la fin, à la fermeture de l'accolade }. L'effet produit est ici d'afficher le texte vecteur détruit.
  Rem : Le destructeur *doit toujours porter le nom de la classe* en question (ici `Vecteur`) avec ~ devant .
- La **fonction membre** `Norme()` calcule la norme du vecteur. Elle est appellée dans le programme principal, et le résultat est affiché. Remarquez que l'appelle de la fonction `Norme()` pour l'objet `v`, se fait *par un point* : `v.Norme()`.
- L'annonce `public :` en haut des déclarations permet à ce qui suit d'être connu et accessible depuis le programme principal. On rend les déclarations non accessibles (si besoin est) par l'annonce `private :`.

Exercices

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 `Vecteur` que l'on appelera `Affiche()`, et qui affiche à l'écran le contenu du vecteur sous la forme :
   `Vecteur : 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 `Vecteur` que l'on appelera `double Produit(Vecteur 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 :
   `double d;`
   `Vecteur v,w;`
   `d=v.Produit(w);`
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 `Vecteur(double xi,double yi,double zi).` Dans le programme principal, on pourra alors déclarer un objet par `Vecteur v(1,2,3);`
   Remarque : vous gardez le constructeur précédent `Vecteur()` 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.

Suite du cours sur la création de classes

On continue à améliorer cette classe vecteur dans la partie **Approndissement du C++** `../cours_2/cours_2.html#tth_sEc1`

15 L'héritage de classes

exemple point coloré @@
exercice : dessin du point coloré @@
rem : héritage multiple possible @à
root utilise l'héritage multiple

La petite histoire :

Careless code recycling causes killer kangas – Mutant Marsupials Take Up Arms Against Australian Air Force.

The reuse of some object-oriented code had caused tactical headaches for Australia's armed forces.

As virtual reality simulators assume larger roles in helicopter combat training, programmers have gone to great lengths to increase the realism of their scenarios, including detailed landscapes and – in the case of the Northern Territory's Operation Phoenix – herds of kangaroos (since disturbed animals might well give away a helicopter's position).

The head of the Defense Science & Technology Organization's Land Operations/Simulation division reportedly instructed developers to model the local marsupials' movements and reactions to helicopters.

Being efficient programmers, they just re-appropriated some code originally used to model infantry detachment reactions under the same stimuli, changed the mapped icon from a soldier to a kangaroo, and increased the figures' speed of movement. Eager to demonstrate their flying skills for some visiting American pilots, the hotshot Aussies "buzzed" the virtual kangaroos in low flight during a simulation.

The kangaroos scattered, as predicted, and the visiting Americans nodded appreciatively... then did a double-take as the kangaroos reappeared from behind a hill and launched a barrage of Stinger missiles at the helpless helicopter. (Apparently the programmers had forgotten to remove THAT part of the infantry coding.)

The lesson ? Objects are defined with certain attributes, and any new object defined in terms of an old one inherits all the attributes. The embarrassed programmers learned to be careful when reusing object-oriented code, and the Yanks left with a newfound respect for Australian wildlife.

Simulator supervisors report that pilots from that point onward have strictly avoided kangaroos, just as they were meant to.

>From June 15, 1999 _Defense Science and Technology Organization Lecture series_, Melbourne, Australia, and staff reports.

Item taken from _Software Testing and Quality Engineering_ magazine, Volume 1, Issue 6 (November/December 1999)

16 Graphisme avec la librairie ROOT

La librairie Root est développée au Cern. C'est une librairie en C++ 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 pour un programme (gestion de la souris, menus déroulants, boites de dialogues avec boutons, ...)
- et beaucoup d'autres choses. Voir la page sur le web de présentation (ci-dessous).

Cette librairie est conçue au départ pour les physiciens des particules travaillant au Cern et participant à la grande expérience du LHC (Large Hadron Collider) qui est une collaboration de milliers de physiciens et démarrera en 2005 (dont l'objectif est de découvrir de nouvelles particules élémentaires). Cette librairie est gratuite, et disponible sur le réseau internet à l'adresse *root.cern.ch*

Sur le réseau il y a une documentation complète (voir le lien dans la page de documentation), voir `http://root.cern.ch/root/RootDoc.html`, pour une documentation au format PDF.

Sur le réseau 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.

La suite est une introduction succinte à l'utilisation de root en mode compilé. Pour connaitre plus de possibilités, voir le document plus de graphismes avec Root.

(root peut s'utiliser aussi en mode interprété : lancer la commande **root** dans une fenêtre *xterm*, et suivre les instructions...)

important

Dans la suite, on suppose que vous compilez votre programme comme indiqué dans l'introduction : `../cours_0/intro.html#tth_sEc4`

Exemple de départ

Ecrire le programme suivant :

```
//--------- (A) -------------
#include <TROOT.h>
#include <TApplication.h>

//----------- (B) ----------------
#include <TCanvas.h>

//----------- (C) ----------------
#include <TEllipse.h>

//------ lignes de declarations globales de ROOT ----(A)-----------
extern void InitGui();
VoidFuncPtr_t initfuncs[] = { InitGui, 0 };
int Error;
TROOT root("hello","hello", initfuncs);

 //------ fonction main (A) --------------------
int main(int argc, char **argv)
{
  TApplication theApp("App", &argc, argv);
```

```
//---- Creation d'une fenetre graphique (B)--------------------
  TCanvas c("c", "fenetre", 400, 400); // on precise la taille en pixels

  double xmin(0),ymin(0),xmax(5),ymax(5);
  c.Range(xmin,ymin,xmax,ymax); // coordonnees de la fenetre (optionnel)

//------ 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

//------- met à jour la fenetre -- (B)--------
  c.Update();

//----donne la main a l'utilisateur (A)---
  theApp.Run();
  return 0;
}
```

Commentaires :

Le programme précédent est découpé en trois types de blocs :

(A) : dans tout programme utilisant Root, ces parties doivent toujours être réécrites.

(B) : c'est le code qu'il faut pour créer la fenetre graphique (Classe TCanvas) .

(C) : c'est le code qu'il faut pour créer l'ellipse (Classe TEllipse).

Si votre programme fait du graphisme, il faut toujours créer une fenêtre graphique qui contiendra votre dessin.

Exercice :

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

Exercice :

Modifier le programme ci-dessus pour faire apparaitre un cercle de rayon 1 qui tourne autour de l'origine, à la distance $R = 2$.

aide : une possibilité est d'utiliser les fonctions membre `SetX1(x1)`, `SetY1(y1)` de la classe `TEllipse` pour positionner son centre.

Remarque

La commande `e.Draw()`appelle une fonction membre de la classe `TEllipse` qui dessine celle-ci. Pour connaitre toutes les opérations possibles que l'on peut effectuer sur l'ellipse, il faut regarder la liste des fonctions membres de cette classes, qui se trouve sur la documentation du Cern : `http://root.cern.ch/root/html/TEllipse.html`.

La page suivante contient la liste des classes proposées par ROOT :`http://root.cern.ch/root/html/ClassIndex.html`

Utilisation des pointeurs avec ROOT

Dans l'exercice précédent, vous avez remarqué que lorsque vous redessiner l'ellipse (objet `e`) **l'ancien dessin disparait**. Cela est pratique, pour faire de l'animation (faire déplacer un cercle par exemple), mais comment dessiner plusieurs cercles simulanéments ?

Nous vous proposons deux possibilités :

1) Utiliser un tableau d'ellipses

Il suffit d'écrire :

```
   c.Range(-2,-2,8,2); // coordonnees de la fenetre
   TEllipse tab_e[3]; // crée un tableau de 3 ellipses
   for (int i=0;i<3;i++)
   {
   tab_e[i]=TEllipse(3*i,0,1); // l'ellipse numéro i se trouve en x=3*i,
y=0
   tab_e[i].Draw(); // dessin
   }
```

Ainsi l'ellipse numéro i+1 n'efface pas l'ellipse numéro i. Mais cette solution est possible que si l'on connait à priori le nombre d'ellipses que l'on veut dessiner.

Remarques :

pour modifier la position `y` de l'ellipse numéro 1, c'est possible, il suffit de faire :

```
   tab_e[1].SetY1(0.5);
```

2) Utiliser un pointeur sur l'ellipse

Il suffit d'écrire :

```
   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
   pe->Draw(); // dessin
   }
```

Remarques :

Cette dernière méthode est simple, et voici ce qu'il se passe : `pe` est un pointeur sur un objet de la classe `TEllipse`.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, et `pe` n'est l'adresse que de la dernière :

L'avantage de cette dernière méthode est que l'on peut créer un nombre indéfini d'ellipses. Le désavantage est que l'on ne peut plus modifier une ellipse déjà existante (il y a quand même une possibilité avec ROOT).

Exercice

Ecrire un programme pour tester ces deux méthodes.

Suite du cours sur la librairie ROOT

Pour plus d'informations, aller à l'adresse : `../intro-root/intro-root.html`

17 Micro-projet

En guide de conclusion, nous vous proposons trois micro-projets qui 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 motiviation et l'aisance que vous ressentez en informatique.

La fractale de Newton

@@

Un morpion pour 2

@@

La fractale du dragon

@@