INFO0202 - REPETITION 1

Les bases de programmation en C++

Comme présenté dans les modalités pratiques des séances de répétitions présentes sur myULiege, les TPs se dérouleront via l'utilisation de la plateforme discord grâce au serveur "INFO0202" accessible en suivant ce lien:

`https://discord.gg/qG2eS6BDfD`

Veuillez à vous connecter sur celui-ci avant le début de cette séance qui aura lieue le 24/02/21. Pour tous problèmes, je reste à votre disposition à l'adresse mail: `thomas.ratz@uliege.be`. Pour plus de facilité et pour rendre les séances de TPs plus agréables pour tout le monde, n'hésitez pas à allumer votre caméra et à partager votre écran lors de nos discussions par petits groupes dans les canaux de discussions "Team X".

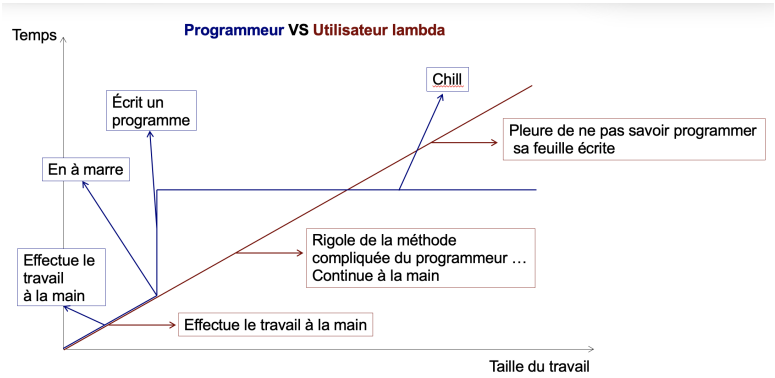
1.1 Pourquoi programmer ?

Premièrement, avant d'entamer l'étude d'une nouvelle matière, on peut se poser la question légitime de pourquoi apprendre cette nouvelle matière. Et donc dans le cadre de ces séances de répétition, de pourquoi programmer ?

A l'heure actuelle et de manière plus prononcée dans le domaine de la science, la programmation est un outil indispensable que tout chercheur utilise dans sa vie de tous les jours pour notamment:

- Etudier des phénomènes scientifiques grâce à des simulations construites sur bases d'équations théoriques (parfois non solvables analytiquement) et implémentées numériquement. Par exemple: étude de l'évolution de populations (Eq. de Lokta-Volterra), étude des propriétés des matériaux (molécules) à l'échelle atomique (density functional theory), ... *etc*
- Réaliser des scripts (séquence de commandes) permettant de réaliser automatiquement une série d'opérations sans devoir les répéter manuellement.
- Traiter et analyser ses données (fitting, ...)

La programmation est donc un outil essentiel permettant d'apporter des réponses à des problèmes complexes auxquels nous ne pourrions pas répondre sans elles. Ou de manière générale pour gagner du temps pour effectuer des séries de taches répétitives. Il est donc essentielle dans la formation de tout bon chercheur d'avoir des bases en algorithmique et en programmation permettant de développer un esprit logique et des compétences de codage informatique.



Dans ces séances de répétitions, nous allons aborder les bases de la programmation dans le langage C++. Première question, pourquoi un langage de programmation ? De manière physique, un ordinateur fonctionne des états binaires pouvant prendre deux valeurs: 0 et 1. Dés lors, pour communiquer et faire comprendre à l'ordinateur la la suite de séquence que nous voulons réaliser dans un programme, il nous faut donc communiquer avec un langage "connu" par l'ordinateur. La première étape est donc la création d'un code dans un langage de programmation (ici C++). Ensuite, il nous faudra **compiler** notre code dans un langage compréhensible par l'ordinateur (sous forme binaire 0 et 1). Cette étape consiste en la création d'un exécutable à partir de notre fichier (.cpp) contenant notre code C++. Enfin, il nous faut **exécuter** ce fichier pour réaliser la suite de commandes codées au départ. Il est important de garder en tête qu'un ordinateur reste une machine réalisant de manière séquentielle (successive) et plus ou moins rapide une suite d'opérations qui lui est fournie. L'ordinateur ne réfléchit pas à la place du programmeur.

Lors de ces séances de répétitions, nous allons travailler à la création d'**algorithmes**. C'est-à-dire la création d'une suite de séquence logiques, d'opérations logiques ayant pour but de résoudre une problématique de départ. Prenons un exemple. En mathématique, une racine carré est définie telle que:

$$|y| = \sqrt{x} \leftrightarrow y^2 = x$$

Tandis qu'en algorithmique, la séquence d'opérations (algorithme de Héron) correspondant au calcul d'une racine carré est donnée par:

$$G_0 = x$$
$$G_n = \frac{1}{2}\left(G_{n-1} + \frac{x}{G_{n-1}}\right)$$
$$G_{n-1} = G_n$$

Tant que $|G_n - G_{n-1}| > \epsilon$. Cette séquence correspond à dire que tant que la différence entre valeur de sortie du programme G_n et la valeur de sortie du programme à l'itération précédente G_{n-1} est supérieure à une valeur donnée par l'utilisateur du programme, alors le programme continue de manière itérative à calculer G_n en remplaçant G_{n-1} par la dernière valeur de G_n obtenue.

De manière générale, il n'existe pas de solution unique pour la réalisation d'algorithme. Cependant, il reste à charge du programmeur de:

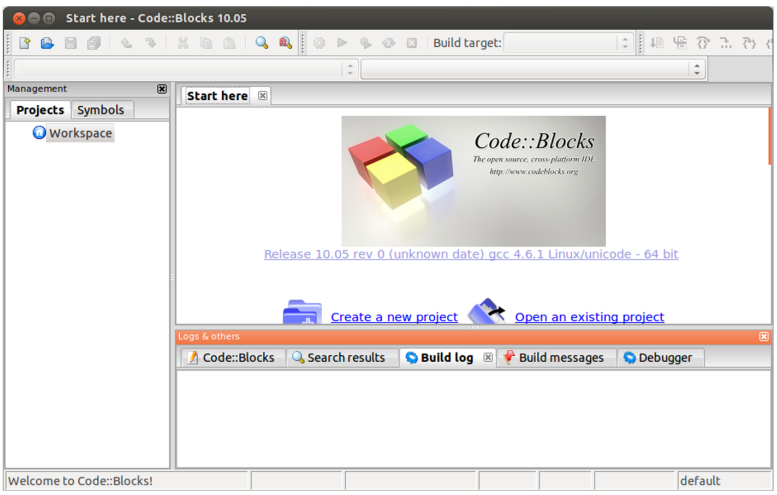
1. Vérifier que l'algorithme est correct, que le résultat obtenu est bien celui attendu: **justesse**.
2. Vérifier que l'algorithme fonctionne dans toutes les situations, c'est-à-dire que celui-ci soit **robuste** (Ex: division par 0).
3. Enfin, pour des utilisateurs extérieurs, de fournir les informations nécessaires au bon fonctionnement du programme.

1.2 Logiciel de programmation

Dans cette section, nous allons introduire le logiciel que vous allez utiliser (si vous le souhaitez) pour réaliser les différents programmes lors des séances de répétitions. Ce programme s'appelle **Code::Blocks** (`https://www.codeblocks.org/`. Il s'agit d'un logiciel gratuit permettant d'éditer un code source (un simple fichier texte) contenant les lignes de commandes écrites en C++. Celui-ci est constitué d'un IDE (integrated development environment). C'est-à-dire un éditeur de texte spécialisé en un langage de programmation (ici C++) contenant notamment:

- Une gestion de projets
- La coloration syntaxique
- Aide à la saisie intelligente
- Un raccourci vers le compilateur

Cet IDE permet de faciliter l'écriture de programmes, la compilation, l'exécution de programmes ou encore le débuggage de ceux-ci. De plus, afin de transformer votre fichier de code en un exécutable (**compilation**) qui sera ensuite exécuté, Code::Blocks contient un GCC (Gnu compiler collection).



1.2.1 Installer Code::Blocks

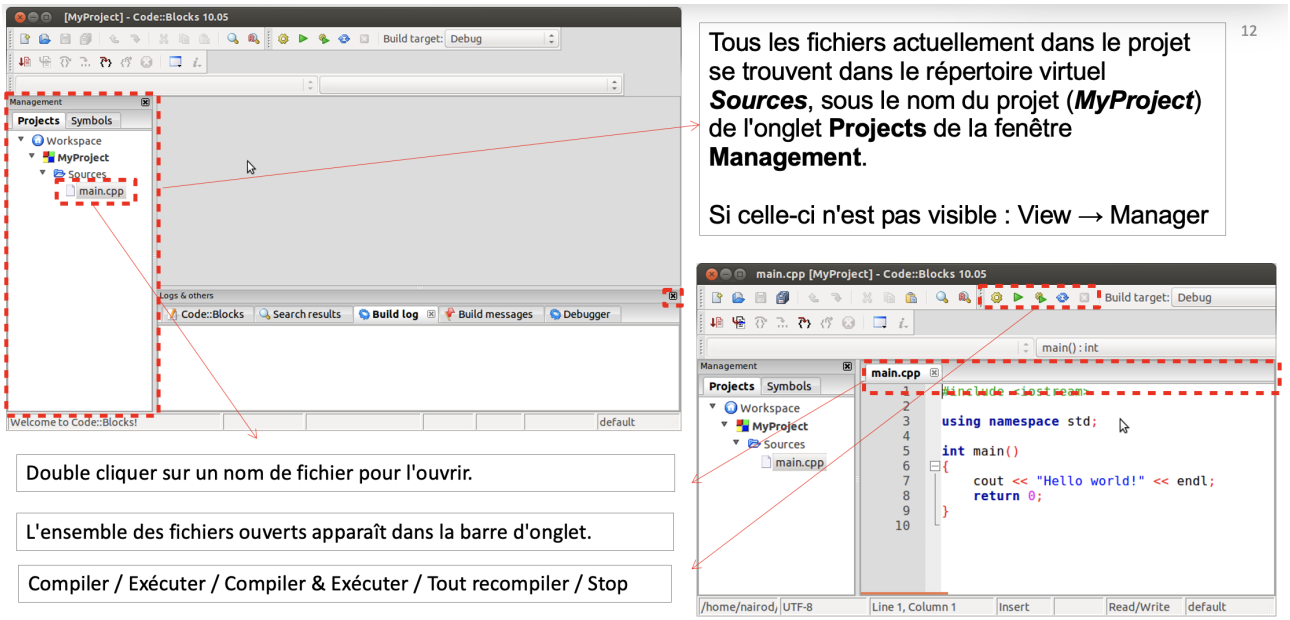
Commencez par installer Code::Blocks sur vos ordinateurs en vous rendant sur le lien suivant:

`https://www.codeblocks.org/downloads/binaries/`

Une fois le logiciel installé sur votre ordinateur, ouvrez Code::Blocks. Vous devriez obtenir la fenêtre suivante:

Ensuite, pour chaque séance de répétition, je vous propose de créer un projet qui contiendra l'ensemble des fichiers (codes) relatifs à la séance. Pour ce faire, allez dans File/New/Project. C:B vous propose ensuite de sélectionner un type de projet. Nous n'utiliserons uniquement le type "Console application". Enfin, sélectionnez le type de langage à utiliser: C++. Une fois ces étapes réalisées, donnez un nom à votre projet. Sélectionnez le dossier où votre projet sera créé. Pour ce faire, créez un dossier dédié aux séances de répétitions sur votre ordinateur et placez l'ensemble des projets dans ce dossier. Pour des informations complémentaires, référez vous aux slides 8-12 du fichier powerpoint accompagnant cette séance de répétition.

Une fois votre projet créé, ouvrez le fichier main.cpp contenant un premier programme. Voici la fenêtre que vous devez obtenir:



1.2.2 Premier programme

Commençons par détailler le contenu de ce premier programme présent dans le fichier "main.cpp".

```
1 #include <iostream>
2
3 using namespace std;
```

```
4
5 int main()
6 {
7    cout <<"Hello world!"<< endl;
8
9    return 0;
10 }
```

- En C++, une instruction se termine toujours par ;
- Dans ce programme les "" délimitent une partie de code (un bloc) nommée ici "main". La fonction "main" est la fonction principale de votre fichier de code contenant la suite d'instructions à réaliser. Cette fonction est définie comme retournant un "entier" par la déclaration "int main". Celle-ci doit donc donner en fin de programme un entier. C'est le rôle de la ligne "return 0".
- Au dessus de la fonction main, la commande "$\#include < iostream >$" permet d'inclure la bibliothèque "iostream" au programme. Une libraire est un répertoire de fonctions codées en C++ pouvant être utilisées à condition d'inclure la libraire à laquelle appartient la fonction.
- Enfin, "using namespace std;" est un nom d'espace standard pour l'appel de fonctions. Il s'agit ici d'un nom d'espace permettant au compilateur d'identifier correctement les fonctions appelées et provenant de certaines bibliothèques. Par exemple pour utiliser l'objert "std::cout", il est nécessaire de spécifier le nom d'espace de celui-ci. Par simplicité, lors des répétitions, nous inclurons le nom d'espace via l'instruction "using namespace std;" en début de code. Ce qui nous permettra d'écrire "cout" directement. En programmation "plus poussée", il faut cependant rester vigilant quant à cette pratique qui peut induire des confusions dans l'appel de différentes fonctions.

1.3 Rappels théoriques

Lors de cette première répétition, nous allons introduire différentes notions essentielles de la programmation en C++ :

- **Les variables** permettant de stocker des données dans la mémoire de l'ordinateur à une adresse connue.
- **Les opérateurs** permettant de réaliser des opérations sur le contenu des variables.
- **Les interactions avec le terminal** permettant à l'utilisateur d'interagir avec le programme lors de son exécution (visualisation, écriture).
- **Les conditions et boucles** permettant de réaliser des opérations sous conditions ou également de répéter de manière itérative une suite d'opération.

Enfin, nous verrons tout au long de cette répétition, **les bonnes pratiques** à connaitre dès nos premiers pas en programmation.

1.3.1 Variables

De manière physique, une variable représente un espace mémoire alloué dans la mémoire de l'ordinateur pour contenir une information (nombre, caractère, booléen). Chaque variable possède un **nom** choisi avec soin afin de s'y référer. Une **adresse** correspondant au lieu (virtuel) dans la mémoire de l'ordinateur où la variable est stockée et en fin un **type** dépendant de la nature de la donnée stockée. Le type de variable détermine également la taille de l'espace mémoire dédié au stockage de cette donnée. On distingue différent types de variables:

- nombres entiers (int, long, ...)
- nombres réels (double, float, ...)
- booléens (bool)
- caractères (char)
- ...

Ces différents types de variables peuvent stocker une information avec une taille et une précision différente:

type de variable	Signification	Octets	Plage de valeurs acceptées
char	Caractère	1	0 à 255
int	Entier	2	-32 768 à 32 767
long int	Entier long	8	-2 147 483 648 à 2 147 483 647
float	Flottant	4	3,4 10^{-38} à 3,4 10^{38} env.
double	Flottant double	8	1,7 10^{-308} à 1,7 10^{308} env.

Voici un exemple:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7     cout <<"Hello world!"<< endl;
8
9     char myCharactere;
10    myCharactere = 'c';
11
12    int a;
13    a = 1;
14
15    double myDouble = 4.5;
16
17    return 0;
18 }
```

Dans ce dernier, "myCharactere" représente une variable de type caractère (char). Celle-ci est déclarée ligne 9 et ensuite évaluée ligne 10 par la lettre c. De même, la varaible de nom "a" est déclarée comme entière et ensuite on place le nombre 1 dans celle-ci. "myDouble" quant à elle est déclarée réelle et ensuite évaluée à 4.5. Avant d'être évaluée, tout variable est d'abord déclarée. Cette déclaration permet à l'ordinateur de connaitre la place dans la mémoire à allouer à la variable. Le choix de la déclaration dépend du type de variable ainsi que de la précision souhaitée.

Certaines variables peuvent également être déclarée comme constante. Ce type de déclaration permet de placer dans une variable une valeur type qui ne pourra pas être modifiée (si c'est le cas, votre programme retournera une erreur lors de la compilation). Les variables déclarées comme constantes sont généralement écrites en majuscules.

```
#include <iostream>

using namespace std;

const double PI = 3.141592;

int main()
{
   cout <<"Hello world!"<< endl;

   const int SIZEMAX = 5000;

   PI = 3.14; // RETURN ERROR

   return 0;
}
```

1.3.2 Opérateurs

Les opérateurs sont des outils s'appliquant sur un ou deux éléments/variables. On retrouve parmi ces opérateurs:

- Des opérateurs arithmétiques: +,-,*,/,%
- Des opérateurs d'affection: =, +=, -=, *=, /=, %=, ++, –
- Des opérateurs de comparaison: >, >=, <=, ! = (différent), == (comparable)

- Des opérateurs logiques: && (et), || (ou), ! (non)

Notez ici la différence entre affecter et comparer. "a=5" signifie que l'on affecte la valeur 5, on place la valeur 5 dans la variable "a". Tandis que "a==5", on compare la valeur contenue dans la variable "a" avec 5.

Quelques exemples:

```
#include <iostream>

using namespace std;

const double PI = 3.141592;

int main()
{
   cout <<"Hello world!"<< endl;

   int a = 7, b = 5, c = 3, inbr;
   double pi = 3.14159, e = 2.71828, phi = 1.6180, dnbr;
   bool result;

   dnbr = (pi+phi)*e;              // dnbr vaut 12.93789
   inbr = a%b-c;                   // inbr vaut -1
   inbr += b*c;                  // inbr vaut -5
   result = !((b > a) || (b > c));     // result vaut false/0

   return 0;
}
```

Ensuite, il est également parfois nécessaire de changer le type d'une variable. Par exemple, lors d'une opération de division. Pour ce faire, il faut spécifier le nouveau type à adopter et placer le résultat dans une variable de type semblable. En cas de doute, il vaut mieux forcer le typ d'une variable. Cependant attention à ne pas perdre des données (manque de précision).

```
#include <iostream>

using namespace std;

const double PI = 3.141592;

int main()
{
   cout <<"Hello world!"<< endl;

   int a = 27, b = 5, c = 3, bigint = 2000000000;
   double dnbr;
   bool result;

   dnbr = (double)a/(double(b); // Division rationnelle
   result = (bool)(b-5);
   result = (bool)(c-5);

   short littleint = (short)bigint; // perte de donnees ...
   return 0;
}
```

1.3.3 Interactions avec le terminal

La libraire "iostream" contient des éléments permettant à l'utilisateur d'interagir avec la compilation du code. Notamment l'objet **cout** permettant d'afficher quelque chose dans le terminal pendant l'exécution du programme. Ou encore l'objet **cin** permettant à l'utilisateur d'entrer une valeur dans une variable lors de l'exécution du programme. L'objet **cout** s'utilise avec l'opérateur "de sortie": "¡¡" tandis que l'objet **cin** s'utilise avec l'opérateur "d'entrée": "¿¿".

L'utilisation de l'objet **cout** se fait comme suit:

```
#include <iostream>

using namespace std;

int main()
{
```

```
7     int a = 1;
8     cout << "La valeur de a est" << '\t' << a << endl; // Affiche: La valeur de a est 1
9
10    return 0;
11 }
```

Dans cet exemple, chaque élément de type distinct est séparé par l'opérateur <<. L'objet **cout** affiche d'abord "La valeur de a est" ensuite la tabulation '^' et enfin la valeur contenue dans la variable a. La commande "endl" signifie un retour à la ligne.

Ensuite, l'objet **cin** permet de lire des données tapées dans le terminal par l'utilisateur et de stocker celle-ci dans une variable définie préalablement. Celle-ci s'accompagne de l'opérateur >> signifiant "entrée".

```
1 #include <iostream>
2
3 using namespace std;
4
5 int main()
6 {
7     int a;
8     cout <<"Entrez un nombre entier :";
9     cin >> a;
10    cout << "La valeur de a est" << '\t' << a << endl; // Affiche: La valeur de a
11    cout << a*a << endl; // Affiche la valeur contenue dans a au carre
12
13    return 0;
14 }
```

Ces deux objets (cout, cin) appartiennent à la bibliothèque iostream et sont définis dans l'espace de nom standard (std, d'où le "using namespace std;").

1.3.4 Bonnes pratiques

Afin d'éviter les erreurs, de permettre à un autre utilisateur de comprendre votre programme ou même à vous même de vous replonger dans un programme réalisé précédemment, il est plus que nécessaire de suivre des règles de bonnes pratiques (celles-ci seront prises en compte dans l'évaluation de fin d'année).

Premièrement, les **commentaires**. Les commentaires représentent du texte ignoré par le compilateur mais permettent de documenter et/ou d'expliquer le code. Ceux-ci améliorent la lisibilité et la compréhension du code. Pour ajouter un commentaire, utilisez: "//" ou encore "/* ... */ " pour commenter plusieurs lignes:

```
1 #include <iostream> // inclus la bibliotheque iostream
2
3 using namespace std;
4
5 int main() // fonction principale
6 {
7     int a;
8     cout <<"Entrez un nombre entier :"; // Demande
9     cin >> a;
10    /*
11    cout << "La valeur de a est" << '\t' << a << endl; // Affiche: La valeur de a
12    cout << a*a << endl;  */
13    // Tout ceci est ignore
14
15    return 0;
16 }
```

Un second élément essentiel est l'indentation. C'est-à-dire, l'utilisation de tabulation lors de l'ouverture de fonctions, boucles, conditions pour "définir" de manière visuelle ce qui appartient à cette fonction, boucle, condition. L'indentation permet une meilleure lisibilité et compréhension du code.

```
void a_function(void)
{
   if (x == y) {
      something1();
      something2();
   } else {
      somethingelse1();
      somethingelse2();
   }
   finalthing();
}
```

```
void a_function (void)
{
if (x == y)
    {
    something1 ();
    something2 ();
    }
    else
  {
    somethingelse1 ();
    somethingelse2 ();
  }
    finalthing ();
}
```

De manière générale, veillez aux différents points suivants:

- Le programme doit être correct (justesse)
- Le programme doit fonctionner dans toute situation (robustesse)
- Il doit être lisible (bien indenté et commenté)
- Les noms des variables et des fonctions doivent être pertinents !
- Le programme ou une fonction doit être concis (plus un programme est cours moins il y a d'erreurs)
- Comprenez les lignes que vous écrivez
- Compilez votre programme régulièrement afin d'identifier des erreurs éventuelles
- Ajouter des lignes de codes de vérifications
- En cas de doutes, référez-vous à la documentation (internet, forums, livres de références, ...)

Enfin, quelles sont les types d'erreurs et les bugs que vous pouvez rencontrer ? Il existe deux grands types:

1. Les **erreurs de syntaxes** menant à une erreur lors de la compilation de votre code par Code:Blocks (Ex: oubli ";", ...)
2. Les **erreurs logiques** menant à une erreur lors de l'exécution de votre code. Votre code ne réalise pas correctement ce que vous imaginiez. Afin d'éviter ces erreurs, compilez régulièrement votre code, ajoutez des lignes de codes de vérifications (cout) pour vérifier que votre code donne bien la solution attendue à différentes étapes de celui-ci. Également, avant d'entamer la rédaction de votre code, il est utile d'écrire sur une feuille de papier les étapes logiques à réaliser pour atteindre votre but.

Réaliser la première série d'exercices guidés.

1.3.5 Branchements conditionnels

Dans ces dernières sections, nous allons aborder les derniers éléments de bases essentiels à la programmation que sont les boucles et les branchements conditionnels. Premièrement, les branchements conditionnels permettent d'exécuter une partie du code si une condition est vérifiée (true).

if ... else

```
1  #include <iostream> // inclus la bibliotheque iostream
2
3  using namespace std;
4
5  int main() // fonction principale
6  {
7     int a = 2, b = 5;
8
9     if(a > b)
10       cout << a << " est plus grand que " << b << endl;
11
12    if(b>a)
13    {
```

```
14        cout << b << " est plus grand que " << a << endl;
15    }
16
17    if(b==a)
18    {
19        cout << b << " est egal a " << a << endl;
20    }
21
22    if (a > b)
23        cout << a << " est plus grand que " << b << endl;
24     else if (a==b)
25        cout << a << " est egal a " << b << endl;
26    else
27        cout << b << " est plus grand que " << a << endl;
28
29    return 0;
30 }
```

Sur l'exemple ci-dessus, la première condition compare le contenu de la variable "a" avec celui de la variable "b". Si "a" est bien plus grand que "b" alors le résultat est true, le code sous la condition est exécuté. Si le code à exécuter est constitué d'une seule ligne, les crochets ne sont pas nécessaires. Il est possible d'imbriquer des conditions dans des conditions.

switch

Si une variable numérique doit être comparée à plusieurs valeurs, l'instruction **switch** est plus élégante à utiliser:

```
1  #include <iostream> // inclus la bibliotheque iostream
2
3  using namespace std;
4
5  int main() // fonction principale
6  {
7      int choix;
8      cout << "Rentrez votre choix: ";
9      cin >> choix;
10     switch (choix)
11     {
12         case 0:
13         {
14             cout << "Bon choix" << endl;
15             break;
16         }
17         case 1:
18             cout << "Choix ok" << endl;
19             break;
20         default:
21             cout << "Mauvais choix" << endl;
22     }
23    return 0;
24 }
```

Remarque, les variables de type `char` sont des variables numeriques et peuvent être utilisés dans les `switch`

```
1  char monChar = 'c';
2  switch (monChar) {
3      case 'a':
4      ...
5      break;
6      case 'c':
7      ....
8      break;
9      default:
10     ...
11 }
```

1.3.6 Boucles

Enfin nous allons décrire deux types de boucles. Les boucles "while" et les boucles "for". Les boucles sont des instructions de programmations permettant de répéter le contenu de code qu'elles contiennes.

while

```
#include <iostream> // inclus la bibliotheque iostream

using namespace std;

int main() // fonction principale
{
   int a = 2, b = 5;

   while(a < b)
   {
      a++;
      cout << a << endl;
   }

   return 0;
}
```

Dans cet exemple simple, l'instruction while (tant que) est utilisée pour incrémenter la valeur contenue dans la variable "a". La condition (contenue entre parenthèses après while) étant que si le contenue de la variable "a" est plus petit que le contenu de la variable "b" alors on exécute ce qui est contenu dans la boucle et on réévalue la condition après exécution. De sorte que la boucle s'arrête lorsque a devient égale à b. On utilise généralement la boucle while quand l'on ne sait pas combien d'itération l'on veut exécuter.

for

Le second type de boucle est la boucle "for" (pour). Un bloc de code ou une instruction est exécuté itérativement suivant une variable incrémentée. Dans cette boucle for, on retrouve donc l'initialisation de la variable d'itération, une condition d'arrêt et une incrémentation. (Ex: for(int i = 0 *initialisation*; i ¡= 10 *condition* ; i ++ *incrément*))

```
#include <iostream> // inclus la bibliotheque iostream

using namespace std;

int main() // fonction principale
{
   int a = 2;

   for(int i=0; i <=10; i += 2)
   {
      a++;
      cout << a << endl;
   }

   return 0;
}
```

Réaliser la seconde série d'exercices guidés.

1.4 Exercices guidées

1.4.1 Première série

1. Déclarez deux variables de type entières et deux variables de type réels. Affectez ces variables à des valeurs saisies au clavier et afficher le contenu de ces variables dans la console. Effectuez les opérations '+','-','*','/' sur chacun des couples de variables et affichez les résultats dans la console.

2. Créez un programme qui affiche dans la console la table de vérité de l'opérateur logique AND (&&)

true	&&	true	=	true
true	&&	false	=	false
false	&&	true	=	false
false	&&	false	=	false

3. Déclarez deux variables de types entières auxquelles vous affecterez une valeur saisie au clavier. Echanger ensuite le contenu des deux variables et affichez dans la console le contenu des variables avant et après l'échange des valeurs. Afin de réalier cette tâche, il est nécessaire de créer une variable tampon.

1.4.2 Seconde série

1. Créez un programme permettant de sommer les n premiers multiples d'un nombre k. Par exemple, la somme des 6 premiers multiples de 3 est 3 + 6 + 9 + 12 + 15 + 18 = 63. Le programme demandera à l'utilisateur d'entrer les deux nombres entiers n et k.

2. La suite de Fibonacci 1, 1, 2, 3, 5, 8, 13, 21, ... est une suite de nombres entiers dont chaque termes est la somme des deux termes qui le précédent. Le rapport de deux termes consécutifs de cette suite tend vers le nombre d'or ϕ. Créez un programme qui demande à l'utilisateur d'entrer une précision et estime le nombre d'or à cette précision. Dans ce but, calculez les éléments de la suite de Fibonacci jusqu'à ce que la différence entre les valeurs de ϕ calculées à deux étapes consécutives soit inférieur à la précision souhaitée. La fonction valeur absolue (fabs) est définie dans la bibliothèque cmath (qu'il ne faut pas oublier d'inclure au début du programme).

3. Créez un programme qui demande à l'utilisateur d'entrer deux nombres entiers et calcule leur PGCD (Plus Grand Commun Diviseur). Si les deux nombres sont premiers entre eux – c'est-à-dire leur PGCD vaut 1 - le programme affichera " nombre1 (valeur) et nombre2 (valeur) sont premiers entre eux " et sinon il affichera leur PGCD. Pour calculer le PGCD, on se base sur les propriétés suivantes. Le PGCD de x,y est le PGCD de x-y,y si $y <= x$ et est le PGCD de x,y-x si $x < y$. Le principe de l'algorithme est le suivant : on modifie à chaque étape x ou y de façon à obtenir un nouveau couple de valeurs (x',y') plus petites et toujours positives. Une fois que l'une de ces deux valeurs est nulle, la valeur non nulle restante est le PGCD

1.5 Exercice supplémentaire

1. Créez un programme qui affiche dans la console les nombres entiers compris entre 1 et 1000 qui sont multiples de 5 et de 11 mais pas de 17. Donnez le nombre total de ces multiples (utilisation d'un compteur).

2. Créez un programme qui affiche dans la console les 100 premiers nombres entiers qui sont multiples de 5 et de 11 mais pas de 17.