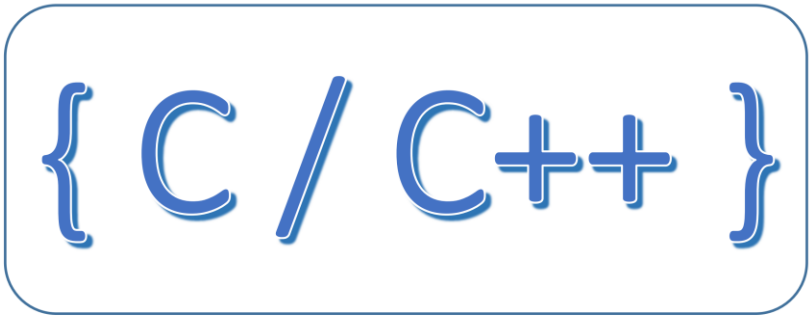
LYCÉE ST-JOSEPH HASPARREN BTS SN-IR	C++ ENTREES/SORTIES	COURS

Tâches professionnelles :	T7.2	Réaliser une maquette, un prototype logiciel/matériel
Compétences du référentiel	C4.4	Développer un module logiciel.
Savoirs - Savoir faires	S4.7	Langages de programmation C++

Table des matières



1 INTRODUCTION

Les programmes d'exemples des cours précédents ont fourni peu d'interaction avec l'utilisateur, et même pas du tout. Ils ont simplement imprimé des valeurs simples à l'écran, mais la bibliothèque standard fournit de nombreuses autres façons d'interagir avec l'utilisateur via ses fonctions d'entrée / sortie. Cette section présentera une courte introduction à certains des plus utiles.

C ++ utilise une abstraction pratique appelée ***flux*** pour effectuer des opérations d'entrée et de sortie dans les médias séquentiels tels que l'écran, le clavier ou un fichier. **Un *flux* est une entité vers/depuis laquelle un programme peut insérer/extraire des caractères**. Il n'y a pas besoin de connaître de détails sur le support physique associés au flux. Tout ce que nous devons savoir est que les *flux* sont une source/destination de caractères, et que ces caractères sont fournis/accepté séquentiellement (l'un après l'autre).

La bibliothèque standard définit une poignée d'objets de flux qui peuvent être utilisés pour accéder à ce que l'on considère les sources et les destinations standards de caractères (pour l'environnement dans lequel le programme fonctionne) :

Flux	Description
cin	flux d'entrée standard
cout	flux de sortie standard
cerr	flux erreur standard (sortie)
clog	flux journalisation standard (sortie)

Nous allons voir plus en détail seulement **`cout`** et **`cin`** (les flux de sortie et entrée standard); **`cerr`** et **`clog`** sont également des flux de sortie, de sorte qu'ils fonctionnent essentiellement comme **`cout`** , avec pour seule différence qu'ils identifient les flux à des fins spécifiques: messages d'erreur et enregistrement/journalisation d'évènements (dans de nombreux cas, ils font réellement exactement la même chose: ils impriment à l'écran, mais ils peuvent également être redirigés individuellement par exemple vers un fichier).

2 SORTIE STANDARD (COUT)

Sur la plupart des environnements de programmation, la sortie standard par défaut est l'écran, et l'objet de flux C++ défini pour y accéder est **`cout`**.

Pour les opérations de sortie formatée, **`cout`** est utilisé conjointement avec ***l'opérateur d'insertion***, qui est écrit comme **`<<`** (deux signes "inférieur").

```
1 cout << "Output sentence"; // prints Output sentence on screen
2 cout << 120;               // prints number 120 on screen
3 cout << x;                 // prints the value of x on screen
```

L'opérateur **`<<`** insère les données qui le suivent dans le flux qui le précède. Dans les exemples ci - dessus, il insère la chaîne littérale « **`Output sentence`** », le numéro **`120`**, et la valeur de la variable **`x`** dans le flux de sortie standard **`cout`**. Notez que dans la première instruction les deux mots sont entre guillemets (**`"`**) parce qu'elle est une chaîne littérale. Dans la dernière instruction, **`x`** n'est pas entre guillemets et donc c'est le contenu (la valeur) de la variable **`x`** qui sera imprimée.

Par exemple :

```
 9    int x{3};
10    cout << "x" << endl;  // imprime :
11    cout << x << endl;    // imprime :
```

.....

.....

Des opérations d'insertion multiple (<<) peuvent être enchaînés dans une seule instruction :

```
cout << "This " << " is a " << "single C++ statement";
```

Cette dernière instruction imprime le texte `This is a single C++ statement`.

Enchaîner des insertions est particulièrement utile pour mélanger les littéraux et les variables dans une seule instruction :

```
int age{23};
string prenom{"Jean"};

cout << "J'ai " << age << " ans et mon prénom est " << prenom << endl;
```

Va imprimer : .....

Pour insérer un saut de ligne on a deux solutions :

- Insérer un **`<< endl`** comme dans l'exemple ci-dessus
- Insérer un **`\n`** dans la chaîne comme vu précédemment (nous reviendrons dessus dans le prochain cours)

3 ENTREE STANDARD (CIN)

Dans la plupart des environnements de programme, l'entrée standard par défaut est le clavier, et **`cin`** est l'objet de flux défini par C++ pour y accéder.

Pour les opérations d'entrée formatées, **`cin`** est utilisé conjointement avec l'opérateur d'extraction **`>>`** (deux signes "supérieur"). Cet opérateur est alors suivi par la variable dans laquelle les données extraites sont stockées.

Par exemple :

```
1 int age;
2 cin >> age;
```

La première instruction déclare une variable de type **`int`** appelée **`age`**, et la seconde instruction extrait de **`cin`** une valeur à stocker dans cette variable.

Le programme attend une entrée depuis **`cin`**, donc il attendra aussi longtemps que nécessaire que l'utilisateur entre une certaine séquence avec le clavier.

Les caractères introduits en utilisant le clavier ne sont transmis au programme que lorsque la touche **`ENTRER`** (ou **`RETURN`**) est pressée.

L'opération d'extraction sur **`cin`** utilise le type de la variable après l'opérateur **`>>`** pour déterminer comment il interprète les caractères lus sur l'entrée : si c'est un entier, le format attendu est une série de chiffres, de même pour un réel, un caractère, etc.

L'opérateur d'extraction sur **`cin`** peut aussi être utilisé pour obtenir des chaînes de caractères de la même manière :

```
1 string mystring;
2 cin >> mystring;
```

Les extractions sur **`cin`** peuvent également être enchaînés en une seule instruction pour demander plus d'une donnée : `cin >> a >> b;` qui est équivalent à :

```
1 cin >> a;
2 cin >> b;
```

Dans les deux cas, l'utilisateur doit introduire deux valeurs, l'une pour la variable **`a`**, et un autre pour la variable **`b`**.

Tout type d'espacement est utilisé pour séparer deux opérations d'entrée consécutives ; cela peut être soit un espace, une tabulation, ou un caractère de nouvelle ligne.

Le programme suivant se déroule de la manière suivante :

- Le programme demande à ce que l'utilisateur rentre un entier
- L'utilisateur saisit 702 (par exemple) au clavier
- Le programme lit 702 sur **cin** et effectue la suite

```
1  // i/o example
2
3  #include <iostream>
4  using namespace std;
5
6  int main ()
7  {
8    int i;
9    cout << "Please enter an integer value: ";
10   cin >> i;
11   cout << "The value you entered is " << i;
12   cout << " and its double is " << i*2 << ".\n";
13   return 0;
14 }
```

Le programme va afficher :

.....

.....

Exercice :

Saisissez et testez un programme qui demande à un utilisateur de saisir, dans cet ordre, son prénom (dans une variable string), son âge (dans une variable int), sa moyenne au bac (dans une variable double), son nom (dans une variable string).

Ensuite le programme doit afficher: « Felicitations ***nom prenom***, vous avez ***age*** ans, et vous avez obtenu le bac avec une note moyenne de ***moyenne***. » (svp pas de minuscules accentuées pour l'instant).

Faites un essai en saisissant comme prénom : « Jean Yves » (avec un espace au milieu).

Que constatez-vous ? .....

4 CIN ET GETLINE

Pour **cin** les espacements (espace blanc, tabs \t , ligne nouvelle \n) provoquent la fin de l'extraction, et donc nous sommes limités à un seul mot à la fois, pas moyen de rentrer une phrase entière d'un coup.

Pour extraire une phrase entière de **cin**, il existe une fonction (terme qui sera étudié plus tard), appelée **getline**, qui prend le flux (cin) comme premier argument, et la variable **string** qui récupère la saisie en tant que deuxième argument.

Chaque fois que le programme interroge l'utilisateur pour saisir une entrée, celui-ci introduit le champ, puis appuie sur **ENTRÉE** (ou **RETURN**) : c'est la lecture de **\n** qui provoque le transfert de **cin** vers la variable **string**.

Par conséquent, pour obtenir les saisies texte de l'utilisateur dans vos programmes console, vous devriez toujours utiliser **getline** au lieu d'extraire directement de **cin**.

Nous reprenons le programme précédent en remplaçant les cin par des getline pour le nom et le prénom :

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

void main()
{
    string prenom{ "Peio" };        // Déclarations et initialisations
    int age{ 19 };
    double moyenne{ 20.0 };
    string nom{ "Dupond" };


    cout << "Bonjour, quel votre prenom ? ";
    getline(cin, prenom);           // getline pour les string
    cout << "Votre Age ? ";
    cin >> age;                     // cin pour les int, char, float, etc.
    cout << "Votre moyenne au bac ? ";
    cin >> moyenne;                 // cin pour les int, char, float, etc.
    cout << "Votre nom ? ";
    getline(cin, nom);              // getline pour les string

    cout << "Felicitations " << nom << " " << prenom << "  vous avez " << age
        << " ans, vous avez passé le bac avec une note moyenne de " << moyenne
        << " ." << endl;
}
```

Faites un essai avec prénom « Jean Yves » (avec un espace au milieu). Que constatez-vous ? .....

.....

Et pour le nom ? .....

5 COMMENT PURGER LE BUFFER CLAVIER ?

cin extrait les caractères jusqu'au premier espacement, mais laisse dans le flux ce caractère d'espacement. Ces caractères qui « trainent » dans le flux vont venir perturber la lecture par un getline.

Pour supprimer une ligne saisie par l'utilisateur qui se trouve dans le buffer d'entrée, il faut utiliser **`cin.ignore()`** :

```
cin.ignore( numeric_limits<streamsize>::max(), '\n' );
```

Le code précédent demande d'extraire mais d'ignorer le *maximum* de caractères, jusqu'à rencontrer un caractère `'\n'` (ligne nouvelle). Ce *maximum* possible dépend du système et est obtenu grâce **à `numeric_limits (on peut aussi prendre un maximum de 32767)`**.

Donc après chaque cin, ou après une série de cin, il faut purger le buffer, comme par exemple :

```
cin >> age;                         // cin pour les int, char, float, etc.
cin.ignore(numeric_limits<streamsize>::max(), '\n'); //purge buffer
```

ou

```
cin >> age;     cin.ignore(32767, '\n');    // cin et purge buffer
```

Modifiez le programme précédent pour purger le clavier après chaque cin .

6 COMMENT VERIFIER LES VALEURS SAISIES AVEC CIN ?

Comme vous pouvez le voir, l'extraction de **cin** semble assez simple. Mais cette méthode a aussi un gros inconvénient. Qu'est-ce qui se passe si le programme demande de saisir un entier **i** et que l'utilisateur saisi autre chose qui ne puisse pas être interprété comme un entier ? Eh bien, dans ce cas, l'opération d'extraction échoue. Et cela, par défaut, laisse le programme continuer sans fixer une valeur pour la variable **i**, produisant des résultats indéterminés si la valeur de **i** est utilisée plus tard.

On s'attend à ce que les programmes se comportent d'une manière plus robuste, même si les utilisateurs saisissent des valeurs invalides. Seuls les programmes très simples doivent reposer sur des valeurs extraites directement à partir de **cin** sans vérification.

L'opérateur **>>** utilisé pour la saisie permet aussi de vérifier la validité de celle-ci via le test de l'exemple suivant :

- D'abord une boucle **while** que nous étudierons bientôt, et qui permet de redemander la saisie tant qu'elle est incorrecte
- On rentre dans le corps du while si :
  - la lecture depuis cin vers nombre n'est pas ok (!)
  - Ou (||) si nombre < 1
  - Ou (||) si nombre > 6
- Si cin.fail() : erreur ce n'est pas un int qui a été saisi
  - Cin.clear() : on efface les erreurs
  - Cin.ignore() : on purge le buffer clavier de tout ce qu'il y a dedans
  - On peut recommencer
- Sinon : c'est un int qui a été saisi mais hors plage

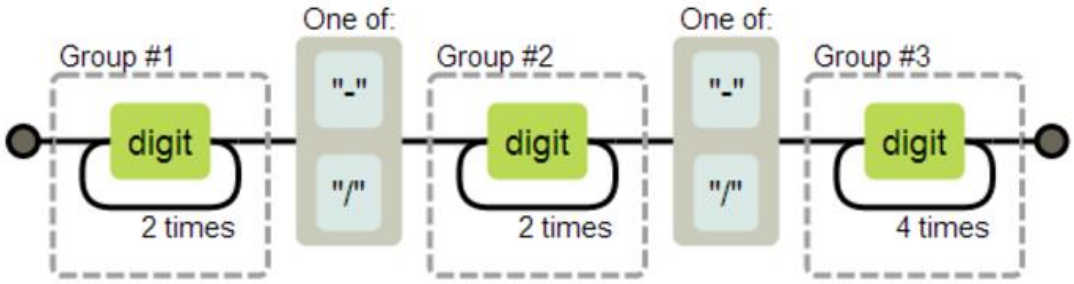
```
// Vérification saisie clavier
#include <iostream>
#include <string>
using namespace std;

int main()
{
    int nombre;
    cout << "Entrez un chiffre entre 1 et 6 : ";

    // lit et teste cin pour voir si c'est un int, vérifie si cin dans la plage requise
    // recommence tant que c'est pas ok
    while (!(cin >> nombre) || nombre < 1 || nombre > 6)
    {
        if (cin.fail())                                    // erreur sur le type de saisie
        {
            cout << "Saisie incorrecte, recommencez : ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
        else                                               // erreur sur la plage de saisie
        {
            std::cout << "Le chiffre n'est pas entre 1 et 6, recommencez : ";
        }
    }
    cin.ignore(numeric_limits<streamsize>::max(), '\n');  // purge le buffer

    cout << "nombre = " << nombre << '\n';                 // si ok affiche la saisie

    cout << "Taper Entrée pour continuer";                 // équivalent à system("pause")
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
}
```

7 LES EXPRESSIONS REGULIERES REGEX

Les versions récentes du C++ permettent de définir un « motif » (pattern) à rechercher/vérifier dans une saisie opérateur.

La définition d'un motif utilise une syntaxe particulière

Par exemple, l'expression régulière "(\d{2})[-/](\d{2})[-/](\d{4})", permet de valider une date, et peut être représentée sous la forme suivante :



Pour plus d'informations :

http://guillaume.belz.free.fr/doku.php?id=expressions_regulieres

https://msdn.microsoft.com/fr-fr/library/bb982727.aspx

8 UN PEU D'ANGLAIS…

Flux	Stream
Enregistrement	Logging
Guillemets "	double quotes

9 EN LANGAGE C

Le langage C ne bénéficie pas de std::cin et std::cout. A leur place il faut utiliser les fonctions **scanf** et **printf** que nous étudierons plus tard.

10 MISE EN FORME DES ENTREES/SORTIES

Pour formater plus précisément l'affichage de nombres, dates, etc. : http://en.cppreference.com/w/cpp/io/manip

11 EXERCICE

Ecrivez un programme en utilisant cin, cout, getline, afin de demander à un utilisateur de saisir son nom, son année de naissance, son adresse complète (numéro et nom rue, code postal, ville, etc. peu importe la forme de l'adresse, l'utilisateur saisit ce qu'il veut).

Il faut tester que la saisie de l'année de naissance soit correcte.

Le programme affichera ensuite « Vous êtes **`nom`** , vous avez **`age`** ans, et vous habitez à **`adresse`**.

Saisissez ce programme sur **http://cpp.sh/ ou https://repl.it/languages/cpp11** , ou **Visual Studio** puis testez-le.