Saisie avec cin en C++

> Ce document en PDF : saisie-cin.pdf

C++

C++ est un langage de programmation compilé (initialement créé par**Bjarne Stroustrup** dans les années 1980) permettant la programmation sous de multiples paradigmes (comme la programmation procédurale, **orientée objet** ou générique). Ses bonnes performances, et sa compatibilité avec le C en font un des langages de programmation les plus utilisés dans les applications où la performance est critique. [source : wikipedia.org]

Lien : www.cplusplus.com

Les exemples

saisie-cin-src.zip

cin

Il existe trois flux de base pour chaque processus :

- `stdin` (0) : Entrée standard (par défaut le clavier)
- `stdout` (1) : Sortie standard (par défaut l'écran)
- `stderr` (2) : Sortie erreur (par défaut l'écran également)

Un flux est un canal destiné à transmettre ou à recevoir de l'information. Il peut s'agir de fichier ou de périphériques.

`cin`, `cout` et `cerr` existent dans l'espace de nom `std` mais pourrait exister dans d'autres espaces de noms. Le nom complet pour y accéder est normalement `std::cin`. L'opérateur `::` permet la résolution de portée en C++.

Pour lire des entrées saisies au clavier (sur l'entrée standard `stdin`), on utilisera `cin` en C++.

Le nom `cin` (**c**haracter **in**put stream) désigne le flux d'entrée standard (le clavier par défaut). Les caractères saisis sont extraits du tampon clavier par `cin` au moyen de l'opérateur d'entrée `»` et affectés aux variables.

Liens :

- basic_io

- cin

L'opérateur d'entrée `»` agit sur un flux de type `istream`. Et L'opérateur de sortie `«` agit sur un flux de type `ostream`.

Liens :

- istream
- ostream

Exemple 1 : la saisie d'un entier

```
#include <iostream>

using namespace std;

int main()
{
    int i;

    cout << "Entrez un entier : ";

    cin >> i;

    cout << "La valeur de i est " << i << endl;

    return 0;
}
```

Les chaînes de caractères

`cin` est évidemment capable d'extraire des caractères pour les affecter à une variable de type `string` :

Exemple 2 : la saisie d'un mot

```
#include <iostream>

using namespace std;

int main()
{
    string mot;

    cout << "Entrez un mot : ";

    cin >> mot;

    cout << "Le mot est " << mot << endl;

    return 0;
}
```

La lecture des chaînes de caractère par `cin` se termine sur ce qu'on appelle un **espace blanc** (*whitespace*), c'est-à-dire le caractère espace, une tabulation ou un caractère de retour à la ligne (généralement la touche *Enter* ou Entrée). Les espaces sont ignorés par défaut ce qui signifie qu'on extrait un seul mot, pas une phrase ou une phrase entière.

Pour obtenir une ligne entière avec `cin`, il faut utiliser la fonction `getline()` qui prend le flux (`cin` ici) comme premier argument et la variable de type `string` comme second.

Exemple 3 : la saisie d'une phrase

```
#include <iostream>

using namespace std;

int main()
{
    string phrase;

    cout << "Entrez une phrase : ";

    getline(cin, phrase);

    cout << "La phrase est \"" << phrase << "\"" << endl;

    return 0;
}
```

Bonus : conversions

Le type `stringstream` permet à une chaîne de caractères d'être traitée comme un flux et permet ainsi des opérations d'extraction ou d'insertion de et vers des chaînes de la même manière qu'elles sont effectuées sur `cin` et `cout`.

Liens :

- sstream
- stringstream

Cette fonctionnalité est très utile pour convertir des chaînes en valeurs numériques et inversement.

Exemple 4 : des conversions

```cpp
#include <iostream>
#include <string>
#include <sstream>

using namespace std;

int main()
{
    string chaine;
    float f = 0.f;
    int n = 0;

    // Conversion chaine -> numérique
    cout << "Entrez un entier : ";
    cin >> chaine;

    // Conversion chaine -> int
    stringstream(chaine) >> n;
    cout << "Après conversion, l'entier est : " << n << endl;

    cout << "Entrez un réel : ";
    cin >> chaine;

    // Conversion chaine -> float
    stringstream(chaine) >> f;
    cout << "Après conversion, le réel est : " << f << endl;

    // Conversion numérique -> chaine
    stringstream oss; // ou : ostringstream oss
    int base = 0;

    base = 8; // en octal
    oss << oct << n;
    cout << "Conversion numérique -> chaine : " << oss.str() << " en base " << base << endl;

    base = 10; // en décimal
    oss.str("");
    oss << dec << n;
    cout << "Conversion numérique -> chaine : " << oss.str() << " en base " << base << endl;

    base = 16; // en hexadécimal
    oss.str("");
    oss << hex << n;
    cout << "Conversion numérique -> chaine : 0x" << oss.str() << " en base " << base << endl;

    return 0;
}
```

Contrôle du flux

Les caractères saisis sont placés dans le flux. Le flux est un*buffer* (une zone mémoire) géré par le système. `cin` extrait des caractères du flux (donc du *buffer*) de manière “formatée” car il analyse le type de destination (un entier, un réel, etc …).

Il est possible de manipuler les caractères du flux de manière “non formatée” :

Unformatted input:

gcount	Get character count (public member function)
get	Get characters (public member function)
getline	Get line (public member function)
ignore	Extract and discard characters (public member function)
peek	Peek next character (public member function)
read	Read block of data (public member function)
readsome	Read data available in buffer (public member function)
putback	Put character back (public member function)
unget	Unget character (public member function)

Par exemple, si on souhaite simplement extraire un caractère du flux, on utilisera `get()`. Si on veut (re)placer un caractère dans le flux, on appellera `putback()`.

Exemple 5 : saisie d'un chiffre (pour montrer l'utilisation de `get()` et de `putback()`)

```cpp
#include <iostream>
#include <ctype.h> // pour isdigit

using namespace std;

int main()
{
    int i = 0;

    // on invite l'utilisateur
    cout << "Entrez un chiffre : ";

    // on extrait le caractère saisi
    char c = cin.get();

    // chiffre ?
    if(isdigit(c)) // équivalent à : (c >= '0') && (c <= '9')
    {
        // on replace le caractère précedemment extrait
        cin.putback(c);

        // on l'extrait sous la forme d'un int
        cin >> i;

        // on l'affiche
        cout << "Le chiffre saisi est : " << i << '\n';
    }

    return 0;
}
```

Remarque : Le positionnement dans le flux est controlable avec `tellg()` et `seek()`.

Contrôle de saisie

Il est possible de connaître (et de modifier) l'état courant du flux d'entrée `istream` et donc de `cin`.

- `rdstate()` retourne les indicateurs d'état (*flags*) d'erreur interne actuel du flux.

Remarque : en informatique, un *flag* (drapeau) est une valeur binaire (booléenne de type vrai ou faux) indiquant le résultat d'une opération ou le statut d'un objet. Ce terme provient du drapeau placé sur les boîtes aux lettres américaines et qui indique la présence ou non d'un courrier.

Ces indicateurs d'état sont automatiquement définis par les appels aux fonctions d'entrée/sortie sur le flux pour signaler certaines erreurs.

iostate value (member constant)	indicates	functions to check state flags				
		good()	eof()	fail()	bad()	rdstate()
goodbit	No errors (zero value iostate)	true	false	false	false	goodbit
eofbit	End-of-File reached on input operation	false	true	false	false	eofbit
failbit	Logical error on i/o operation	false	false	true	false	failbit
badbit	Read/writing error on i/o operation	false	false	true	true	badbit

- `setstate()` modifie les indicateurs d'état en combinant les valeurs actuelles avec ceux passés en argument (en réalisant une opération OU bit à bit).
- `clear()` définit une nouvelle valeur pour les indicateurs d'état du flux. La valeur actuelle est écrasée : tous les bits sont remplacés par ceux passés en argument (par défaut `goodbit`). Si le nouvel état est `goodbit` (qui vaut zéro), tous les indicateurs d'erreur seront effacés.

Il est possible de récupérer les indicateurs d'état individuellement en utilisant `good()`, `eof()`, `fail()` ou `bad()`.

Exemple 6 : contrôle d'une seule saisie

```
#include <iostream>

using namespace std;

int main()
{
    int i = 0;

    cout << "Entrez un entier : ";

    cin >> i;

    if(cin.good())
        cout << "La valeur de i est " << i << endl;
    else
        cout << "Erreur de saisie !" << endl;

    // ou :

    if(!cin.fail())
        cout << "La valeur de i est " << i << endl;
    else
        cout << "Erreur de saisie !" << endl;

    return 0;
}
```

Lorsqu'une erreur se produit, elle est indiquée dans les indicateurs d'état (*flags*) d'erreur interne. Le drapeau (*flag*) maintient evidemment son état jusqu'à ce qu'il soit modifié par un appele à `setstate()` ou `clear()` !

Exemple 7 : le problème d'enchaîner plusieurs saisies

```cpp
#include <iostream>

using namespace std;

int main()
{
    int i1;
    int i2;

    cout << "Entrez un premier entier : ";
    cin >> i1;

    if(cin.good())
        cout << "La valeur est " << i1 << endl;
    else
        cout << "Erreur de saisie !" << endl;

    cout << "Entrez un second entier : ";
    cin >> i2;

    if(cin.good())
        cout << "La valeur est " << i2 << endl;
    else
        cout << "Erreur de saisie !" << endl;

    return 0;
}
```

Le problème apparaît si il y a une erreur (la saisie du caractère 'a' qui n'est pas un entier) dans la première saisie. Tant que le drapeau associé à l'erreur maintient son état, les saisies suivantes échoueront :

```
$ g++ cin-7-probleme.cpp
$ ./a.out
Entrez un premier entier : a
Erreur de saisie !
Entrez un second entier : Erreur de saisie !
```

On pourrait penser qu'un simple appel à `clear()` suffirait à corriger le problème. Ce n'est pas suffisant car le caractère 'a' (qui a provoqué l'erreur) n'a pas été extrait du flux et il est donc encore présent. La prochaine extraction d'un entier par `cin` re-provoquera une erreur !

On pourrait penser alors qu'il suffirait de retirer le caractère qui a provoqué l'erreur en appelant `get()`. Ce n'est pas suffisant car on ne peut pas deviner ce qui a été réellement saisi (lesquels ? combien ? …). Généralement, on "vide" la totalité des caractères saisis jusqu'à l'espace blanc qui marque la fin de la saisie.

Il faut donc effacer l'erreur avec `clear()` et retirer du flux ce qui a provoqué l'erreur :

```cpp
cin.clear(); // on efface l'erreur
cin.ignore(numeric_limits<streamsize>::max(), '\n'); // on vide tout
```

Exemple 8 : correction du problème d'enchaîner plusieurs saisies

```cpp
#include <iostream>
#include <limits>

using namespace std;

int main()
{
    int i1;
    int i2;

    cout << "Entrez un premier entier : ";
    cin >> i1;

    if(cin.good())
    {
        cout << "La valeur est " << i1 << endl;
    }
    else
    {
        cout << "Erreur de saisie !" << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    cout << "Entrez un second entier : ";
    cin >> i2;

    if(cin.good())
        cout << "La valeur est " << i2 << endl;
    else
    {
        cout << "Erreur de saisie !" << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    return 0;
}
```

Saisie formatée

`cin` tente d'extraire du flux les caractères qu'il pourra traduire dans le type de la variable à affecter.

```
$ g++ cin-6-good.cpp

$ ./a.out
Entrez un entier : 1
La valeur de i est 1

$ ./a.out
Entrez un entier : a
Erreur de saisie !

$ ./a.out
Entrez un entier : 12b
La valeur de i est 12
```

Le dernier test montre que lorsqu'on saisit la séquence "12b", `cin` affecte la valeur entière 12 à la variable et que l'extraction effectuée par `cin` est valide. Par conséquence, le caractère 'b' n'a pas été extrait et il est donc toujours présent dans le flux. Il impactera alors la prochaine extraction réalisée par `cin` !

En contrôlant la saisie, il est maintenant possible de réaliser des saisies formatées.

Exemple 9 : saisir les coordonnées d'un point sous la forme x,y

```
#include <iostream>
#include <limits>

using namespace std;

int main()
{
    int x = 0;
    int y = 0;

    cout << "Entrez les coordonnées d'un point sous la forme \"x,y\" : ";

    // on tente d'extraire x
    cin >> x;

    // x est valide ?
    if(cin.good())
    {
        // on extrait le délimiteur
        char delimiteur = cin.get();

        // est-ce une virgule ?
        if(delimiteur == ',')
        {
            // on tente maintenant d'extraire y
            cin >> y;

            // y est valide ?
            if(cin.good())
            {
                // on a saisi un point !
                cout << "x = " << x << endl;
                cout << "y = " << y << endl;
            }
            else
            {
                cout << "Erreur de saisie !" << endl;
                cin.clear();
                cin.ignore(numeric_limits<streamsize>::max(), '\n');
            }
        }
        else
        {
            // on replace le caractère dans le flux
            cin.putback(delimiteur);

            // et on arrête la saisie du point
            cout << "Erreur de saisie !" << endl;
        }
    }
    else
```

```
    {
        cout << "Erreur de saisie !" << endl;
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
    }

    return 0;
}
```

On obtient :

```
$ g++ cin-9-point.cpp

$ ./a.out
Entrez les coordonnées d'un point sous la forme "x,y" : 2,5
x = 2
y = 5

$ ./a.out
Entrez les coordonnées d'un point sous la forme "x,y" : a,b
Erreur de saisie !

$ ./a.out
Entrez les coordonnées d'un point sous la forme "x,y" : 2,b
Erreur de saisie !

$ ./a.out
Entrez les coordonnées d'un point sous la forme "x,y" : 2;5
Erreur de saisie !
```

Fin de fichier

L' indicateur `eofbit` est défini par toutes les opérations d'entrée standard lorsque la fin de fichier est atteinte dans la séquence associée au flux.

Remarque : EOF signifie *End Of File* soit fin de fichier

La valeur renvoyée par `eof()` dépend de la dernière opération effectuée sur le flux (et non de la suivante).

Les opérations qui tentent de lire à la fin du fichier échouent et donc à la fois l `eofbit` et le `failbit` sont positionnés. `eof()` peut être utilisée pour vérifier si l'échec est dû à l'atteinte de la fin du fichier ou à une autre raison.

Pour savoir si on a atteint la fin de fichier, il faut faire une opération sur le flux qui provoque donc une erreur !

Remarque : pour générer un EOF avec le clavier, il suffit de taper `Ctrl+z` sous DOS/Windows et `Ctrl+d` sous UNIX.

Exemple 10 : contrôle d'une saisie d'un entier

```cpp
#include <iostream>
#include <limits>

using namespace std;

bool saisirEntier(int &n, int min, int max)
{
    while(!(std::cin >> n) || (n < min || n > max))
    {
        if(cin.eof())
        {
            return false;
        }
        else if(cin.fail())
        {
            cout << "Entrée invalide ! Recommencez" << endl;
            cin.clear();
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
        }
        else
        {
            cout << "Entrée incorrecte ! Recommencez" << endl;
        }
    }
    return true;
}

int main()
{
    int jour;
    int mois;

    cout << "Quel jour ? ";
    if(saisirEntier(jour, 1, 31))
    {
        cout << "Le jour est " << jour << endl;
    }

    cout << "Quel mois ? ";
    if(saisirEntier(mois, 1, 12))
    {
        cout << "Le mois est " << mois << endl;
    }

    return 0;
}
```

Voir aussi

- Affichage avec cout