

Chapter 4. Inputs / Outputs

1- Introduction :

In C++, inputs and outputs rely on the concepts of streams and operator overloading. The input/output streams of the C++ standard library represent one of the most interesting applications of operator overloading. Context: The overloading of the << and >> operators allows for intuitive interaction with these streams. Context: The overloading of the << and >> operators allows for intuitive interaction with these streams. In this section, the input/output functionalities are presented informally:

- Reading via standard input
- Writing via the standard output

Explanation

Streams are channels or "pipes" thru which data enters or exits the program. In C++, the most commonly used streams are input streams (for reading data) and output streams (for writing data). They are part of the C++ standard library and facilitate data management.

Context: • Input stream: Used to read data (e.g., from the keyboard). The standard input stream is called cin (console input).

Context: The standard input stream is called cin (console input). • Output stream: Used to send data (e.g., to the screen). The standard output stream is called cout (console output). The standard output stream is called cout (console output).

<< (insertion operator): This operator is used to insert data into an output stream. For example, we use cout << to display text or values on the screen.

Here, << inserts the text and the value of a into the standard output stream (cout), which displays it on the screen.

Context: >> (extraction operator): This operator is used to extract data from an input stream. For example, we use cin >> to read data entered by the user from the keyboard.

Reading via standard input: It is generally done with cin. The user can enter data via the keyboard, and this data is stored in program variables.

Writing via standard output: It is generally done with cout. The program sends information to the screen using insertion operators <<.

2- Generalities on flows

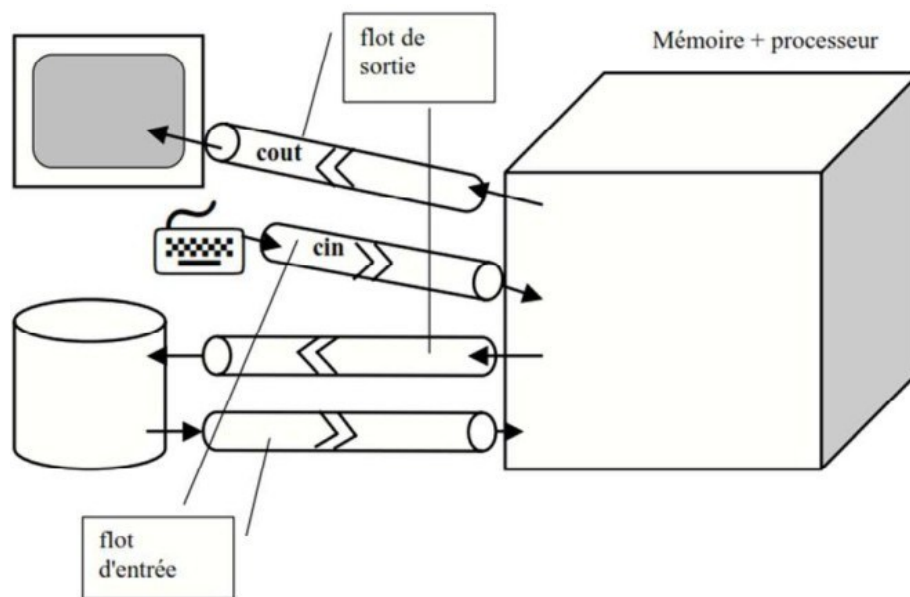
A program that does not interact with its environment is of little interest. It is essential that it can, at the very least, display a result on the screen or save it to a file. For this, the stream technique is used.

Definition of a flow: A stream is a transfer of information, starting from a sender, to a receiver, who consumes this information (this stream).

Inputs and outputs in C++ are always done thru streams, which can be considered as channels - receiving information in the case of an output stream. - providing information in the case of an input stream.

The standard output stream is `cout`. This stream is connected by default to the screen. The standard input stream is `cin`. This stream is connected by default to the keyboard.

The stream manipulation operators, which allow the transfer of information, are - `<<` which allows writing to an output stream - `>>` which allows reading from an input stream



Indeed, the C++ standard library defines extremely powerful classes in the `Iostream` header that allow for the manipulation of input/output streams. These classes particularly perform input/output operations from and to standard input and output devices (generally, the keyboard and the screen). Programming I/O in C++ is particularly easy for most of the programmer's needs.

3- Indeed, with the two operators '<<' and '>>' and the two words 'cin' and 'cout', we manage dialogs with the keyboard and the screen; and with two additional words, 'ofstream' and 'ifstream', we access files, because for these two classes, the opening is done in the constructor and the closing in the destructor.

The structuring of instructions on 'in-memory streams':

Context: The structuring of instructions on 'memory streams':

- `istream` which derives from `istream`
- `ostream` which derives from `ostream`

- `ostream` which derives from `ostream`

4-INPUT/OUTPUT (I/O) INSTRUCTIONS

The inputs are the data that the user provides to a program during execution, either thru the keyboard or a file. The outputs are the results that the program provides to the user during execution, either thru the screen or a file. Context: The outputs are the results that the program provides to the user during execution, either thru the screen or a file. We use the input-output instructions of C++ because they are simpler than those of C. In any case, you must add at the beginning of the file containing the program: `#include` Context: In any case, you must add the following at the beginning of the file containing the program: `#include` using namespace std; Namespaces.

A classic problem in C is that all global declarations share the same namespace. This can be problematic if, for example, you want to use two libraries that declare a `max()` function. C++ uses a namespace system to avoid this. All C++ functions are therefore included in the `std` namespace. In practice, to access an element of the standard library, you need to add `std::` in front of to indicate that we are trying to access an element from the `std` namespace. When a namespace is frequently used in a file, it is possible to add the declaration using namespace `std`; to avoid having to add `std::` everywhere.

Context: To be able to ask the computer to execute (i/o) functions – You need to include the C++ header file in the program.

`#include<iostream>`

4-1- READ ON THE KEYBOARD

The cin stream

Syntax: `cin >> value1 >> ... >> valuen;`

-cin is the keyboard input stream

- ">>" allows you to direct values typed on the keyboard to the variables valuei
 - cin allows you to read values of all types. In practice:
 - When executing a program, if a cin instruction is encountered, execution stops while waiting for the values to be entered on the keyboard. According to the previous syntax, you must enter n values val i separated by spaces, tabs, or line breaks. You must end the entry with a line break.
- The program will assign the value val i to each variable valuei: (valuei=vali)

Examples

Code c++:

```
#include <iostream>
#include <string> // pour utiliser std::string

using namespace std;

int main()
{
    int n;
    float x;
    string t; // Utilisation de std::string pour la chaîne de
    caractères

    do
    {
        cout << "Donner un entier, une chaîne et un flottant : ";
        cin >> n >> t >> x;
        cout << "Merci pour " << n << ", " << t << " et " << x <<
        "\n";
    }

    while (n != 0); // Boucle jusqu'à ce que 'n' soit égal à 0

    return 0;
}
```

```
Donner un entier, une chaine et un flottant : 14 bonjour 12.55  
Merci pour 14, bonjour et 12.55  
Donner un entier, une chaine et un flottant :
```

This exercise aims to teach you how to manage input and output in a C++ program, using standard streams (cin and cout), while manipulating multiple data types and creating simple but continuous interaction with the user. This is an important step in creating interactive programs in C++.

4-2- DISPLAY ON SCREEN

The cout stream

To write the contents of any variable to the screen, we use the cout object, which is an instance of the ostream_withassign class defined in the ostream input-output stream library (io stands for input-output, stream for stream). The cout stream should be thought of as a "pipe" that connects the program to the screen.

To display the contents of a variable on the screen, we proceed as follows:

Syntax:

```
cout << expr1 << ... << exprn << endl;
```

– cout is the console screen display stream

– "<<" directs expr1 expressions to cout;

– endl displays a line break if necessary; it is optional

– cout displays expressions of all types. – According to this syntax, the expression expr1, then the expression expr2, up to the expression expr n

will be displayed on the console one stuck to the other, then a line break will be displayed.

Example :

```
#include <iostream> // Indispensable pour utiliser cout
```

```
using namespace std; // Pour éviter d'écrire std:: à chaque fois
```

```
int main()
{
    cout << "Bonjour"<< endl; // Équivalent à printf("bonjour"); en
C

    system("PAUSE");
    return 0;
}
```

- cout is a predefined output stream associated with standard output (stdout in C).
- << is an operator whose left-hand operand (cout) is a stream and the right-hand operand is an expression of any type.

The preceding "cout << ..." instruction can be interpreted as follows: the stream cout receives the value "hello".

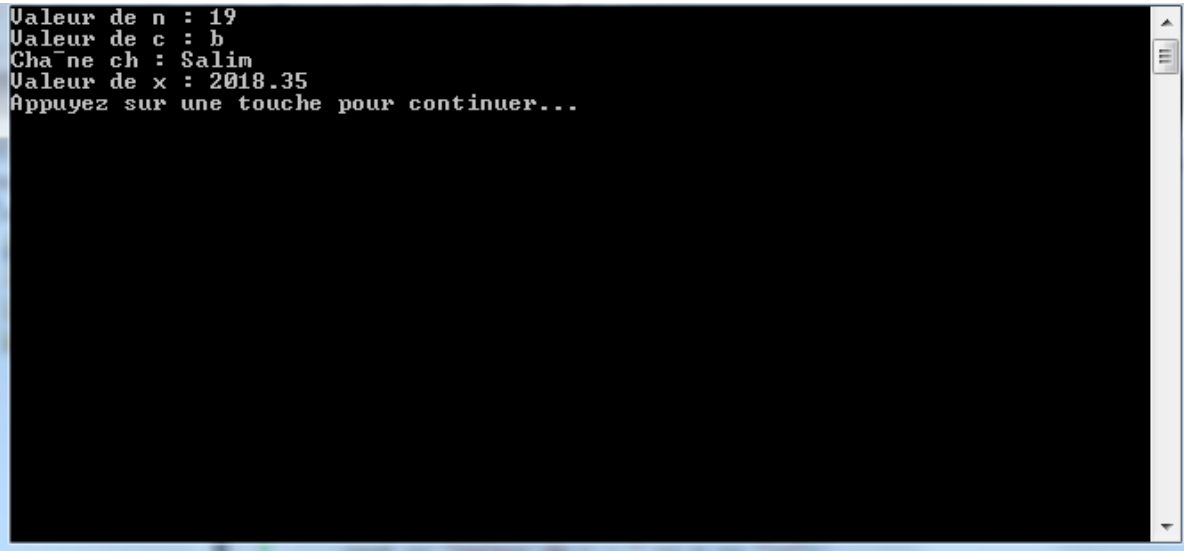
```
#include <iostream>
#include <string> // Inclure la bibliothèque pour std::string

using namespace std;

int main()
{
    int n = 19;
    char c = 'b';
    string ch = "Salim"; // Utilisation de std::string
    double x = 2018.3456789;

    cout << "Valeur de n : " << n << "\n";
    cout << "Valeur de c : " << c << "\n";
    cout << "Chaîne ch : " << ch << "\n";
    cout << "Valeur de x : " << x << "\n";

    system("PAUSE");
    return 0;
}
```



```
Valeur de n : 19
Valeur de c : b
Chaîne ch : Salim
Valeur de x : 2018.35
Appuyez sur une touche pour continuer...
```

5- CASE OF CHARACTER STRINGS

Example

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

int main() {
    string str;
    cout << "Entrez une chaîne sans espaces : ";
    cin >> str; // Ne lit que jusqu'au premier espace
    cout << "Vous avez saisi : " << str << endl;

    system("PAUSE");
    return 0;
}
```

- **Explanation :**

`cin >> str;`: This extraction operator reads characters until it encounters a space, a tab, or a line break. If the user enters multiple words separated by spaces, only the first word will be captured.

For example:

If the user enters: Hello World, the program will only display Hello because `cin >>` stops at the first space.

To solve the problem of partially capturing strings containing spaces with `cin >>`, two methods are available:

- 1- Using a character array (`char[]`)
- 2- Using `std::string` with `getline()`:

1- Using a character array (`char[]`)

Example with a character array (without spaces):

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

int main() {
    char ch[50]; // Déclaration d'un tableau de 50 caractères
    cout << "Entrez une chaîne sans espaces : ";
    cin >> ch; // Ne lit que jusqu'au premier espace
    cout << "Vous avez saisi : " << ch << endl;

    system("PAUSE");
    return 0;
}
```

Example with character table (with spaces):

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

int main() {
    char ch[50]; // Déclaration d'un tableau de 50 caractères
    cout << "Entrez une chaîne avec espaces (max 50 caractères) : ";
    cin.get(ch, 50); // Lit toute la ligne, y compris les espaces
    cout << "Vous avez saisi : " << ch << endl;

    system("PAUSE");
    return 0;
}
```

1- Using `std::string` with `getline()`

Example with `cin >>` (without spaces):


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

int main() {
    string str;
    cout << "Entrez une chaîne sans espaces : ";
    cin >> str; // Ne lit que jusqu'au premier espace
    cout << "Vous avez saisi : " << str << endl;

    system("PAUSE");
    return 0;
}
```

Example with getline() (with spaces):

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

int main() {
    string str;
    cout << "Entrez une chaîne avec espaces : ";
    getline(cin, str); // Lit toute la ligne, y compris les espaces
    cout << "Vous avez saisi : " << str << endl;

    system("PAUSE");
    return 0;
}
```

- For strings without spaces, use `cin >>`.
- For strings with spaces, use `cin.get()` with `char[]` or `getline()` with `std::string`.

The `getline()` method with `std::string` is the most recommended method because it is safer and more flexible.

Strings are a group of letters that can be used to represent words, sentences, etc. A string is an array of chars whose last value is 0 (zero). In C++, there are two ways to manage strings:

- an array of characters: `char ch[50];`
- an object: `stringst;`

5-1- Using a character array (`char[]`):

When using a character array (`char[]`), if you want to allow the user to enter a string with spaces, you cannot simply use `cin >> ch;`, as this only captures the first word (up to a space). Context: When you use a character array (`char[]`), if you want to allow the user to enter a string with spaces, you cannot simply use `cin >> ch;`, as this only captures the first word (up to a space). To capture a complete string with spaces, it is recommended to use the `cin.get()` function, as in the following example:

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

int main() {
    char ch[50]; // Déclaration d'un tableau de 50 caractères
    cout << "Entrez une chaîne de caractères (max 50 caractères) : ";
    cin.get(ch, 50); // Capture jusqu'à 50 caractères, y compris les
    espaces
    cout << "Vous avez saisi : " << ch << endl;
    system("PAUSE"); // Pause pour voir les résultats (sur Windows)
    return 0;
}
```

Explanation:

- `cin.get(ch, 50);` allows reading up to 50 characters, including spaces, without exceeding the size of the array. This also protects against buffer overflows.
- This method is useful when working with character arrays, but it is riskier due to possible size limitations.

5-1- Using `std::string` with `getline()`:

The use of the `std::string` class is recommended because it is safer and more flexible. Context: The use of the `std::string` class is recommended because it is safer and more flexible. You can capture strings containing spaces by using the `getline()` function, which reads the entire line until the end (until you press "Enter").

Example

```
#include <iostream>
#include <string> // Nécessaire pour utiliser std::string
using namespace std;

int main() {
    string str;
    cout << "Entrez une chaine de caracteres : ";
    getline(cin, str); // Lecture de la chaîne avec espaces
    cout << "Vous avez saisi : " << str << endl;
    system("PAUSE");
    return 0;
}
```

Explanation:

- `getline(cin, str);` reads the entire line until the user presses "Enter", including spaces.
- Context: • With `std::string`, there is no risk of overflow because the size of the string is managed dynamically.