Input/Output Operation in C++

- The cout is not a C++ command. C++ has no built-in input/output commands.
- C++ uses a convenient abstraction called **streams** to perform input and output operations in sequential media such as the screen, the keyboard or a file.
- A stream is an entity where a program can either insert or extract characters to/from. There is no need to know details about the media associated to the stream or any of its internal specifications. All we need to know is that streams are a source/destination of characters, and that these characters are provided/accepted sequentially (i.e., one after another).
- The C++ **standard library** defines a handful of stream objects that can be used to access the standard sources and destinations of characters by the environment where the program runs.
- Using the C++ iostream library we will get the user's input from the keyboard and we will print messages onto the screen. The iostream library is part of the C++ standard library.
- The header file **iostream** must be included to make use of the input/output (cin/cout) operations. Do not use iostream.h, it is deprecated.

Standard output stream (cout):

- Usually the standard output device is the display screen.
- **cout** is the instance of the ostream class. cout is used to produce output on the standard output device which is usually the display screen. The data needed to be displayed on the screen is inserted in the standard output stream (cout) using the insertion operator (**<<**).

stream	description
cin	standard input stream
cout	standard output stream
cerr	standard error (output) stream
clog	standard logging (output) stream

- Some Standard stream objects already defined in standard library.
- We are going to see in more detail only cout and cin (the standard output and input streams); cerr and clog are also output streams, so they essentially work like cout, with the only difference being that they identify streams for specific purposes: error messages and logging;

- As **cout** is an object of class ostream, characters can be written to it either as formatted data using the insertion operator (operator<<) or as unformatted data, using member functions such as write.

- An output stream has a bunch of member variables that control the details of what output looks like, and these variables can be controlled by a set of member functions and special objects called **manipulators**. Most of these variables retain their values until they are changed, but a few only act temporarily - for the next output operation only.
- So **manipulators** are objects that cause the output stream object to do something, either to its output, or to its member variables.

- Operator **<<** has overloaded definitions for all of the built-in types. By default, characters and strings are simply output as is, and this is usually satisfactory. So the main complexity is controlling the format of numbers.
- We can control the format by using member functions of cout or manipulators to change a variety of member variables.

Example -1 : Calculate the area of circle (mainCircle01.cpp)

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

int mainCircle01()
{
   const float PI=3.14159;
   float radius = 5;
   float area;
   area = radius * radius * PI; // Circle area calculation
   cout << "The area is " << area << " with a radius of 5.\n";
   radius = 20; // Compute area with new radius.
   area = radius * radius * PI;
   cout << "The area is " << area << " with a radius of 20.\n";
   return 0;
}
```

The << operator inserts the data that follows it into the stream that precedes it. In this example it inserts all the three pieces one by one.

Actually, many return statements are optional. C++ would know when it reached the end of the program without this statement. It is a good programming practice, however, to put a return statement at the end of main routine.

EXAMPLE – 2 : Character Arrays Versus Strings (mainCharArray02.cpp)

NOTE: Strings must be stored in character arrays, but not all character arrays contain strings. The character array contains a string when it has a null zero at its end.

```
#include <iostream>
#include <string.h>
using namespace std;
//  Defining char array in different way
int mainCharArray02()
{
   char petname[20];    // Reserve space for the pet's name.
   petname[0]='A';      // Assign values one element at a time.
   petname[1]='l';
   petname[2]='f';       petname[3]='a';
   petname[4]='l';       petname[5]='f';
   petname[6]='a';
   petname[7]='\0'; // to ensure this is a string!
   cout <<petname << "\n";
```

How long is the string in petname? It is seven characters long because the length of a string never includes the null.
NOTE: Place an #include <string.h> line in programs that use **strcpy**.

```
    char book_name[27]="An Introduction to Angular";  // Total length of array 1 more than total char
    char writer_name[]="Mr Ruth Ann Cooper";
    cout << book_name << "\n" << writer_name << "\n";
    strcpy(petname, "PushiCat");      // Copies PushiCat into the array.
    cout << petname << "\n";
    return 0;
}
```

Two ways of assigning value to string during declaration.

Note :

- Manipulator **endl** is an object, which when supplied to operator<<, causes a newline character to be put into the output stream, followed by a call of cout's flush function, which causes the internal buffer to be immediately emptied.
- This makes sure all of the output is displayed before the program goes on to the next statement. So manipulators are objects that cause the output stream object to do something, either to its output, or to its member variables.
- The manipulators with no arguments, like **endl**, are included in <iostream>. If you use manipulators that take arguments (like setprecision and setw) you need to #include <iomanip>.

- Default output format for integers and doubles All of the digits of an integer will be printed using decimal (base 10), with no leading zeros and a leading minus if it is negative.
- The basic amount of space used is determined by the precision. The default precision is 6, which means up to 6 significant digits are used to represent the number. This counts digits both to the left and the right of the decimal point.
- As the number increases in size, places to the right of the decimal point will be dropped, and the result rounded off as needed to stay within 6 digits. If the number cannot be represented with six digits to the left of the decimal point after all to the right have been dropped, the output will flip into exponential (scientific) notation showing 6 significant digits and the exponent of 10.
- You can increase or decrease the precision, and this will change the number of significant digits shown in the output.
- **Saving and restoring stream settings**
  If you are going to change the format settings of the stream, you normally will want to restore them to what they were before you changed them, so that different parts of your program will not interfere with each other. You can get the current settings (called "**flags**") from the output stream with a member function named **flags**. The settings can be stored in a type of object called **fmtflags** object, defined in a class called ios, which is included with <iostream>. You can declare one of these objects, but you have to declare it using the scope resolution operator. Example of getting current settings:
  ios::fmtflags old_settings = cout.flags();

  Then, after doing the output using the new setting, you can restore the old setting by calling the same function with the old settings as an argument: **cout.flags(old_settings);**

  Other settings can be obtained and restored with member functions. For example,
  int old_precision = cout.precision(); will save the current precision specification. Then
  cout.precision(old_precision); will restore the precision to the original value.

- **cout.precision(0);** or **cout << setprecision(0);** restores the 6-digit default.

- **Precision and the fixed floating-point format for neat output of doubles**, the fixed format is most useful. In fixed format, the precision specifies the number of digits to the right of the decimal point, and a precision of zero means zero places to the right of the decimal point (i.e. round to the nearest unit).

Example- 3 : Control printing using manipulator (mainControlPrinting03.cpp)

- All floating-point numbers print with too many decimal places for most applications, but we can specify how many print positions to use in printing a number and decimal places.
- We typically use the **setw** manipulator when you want to print data in uniform columns. Be sure to include the **iomanip** header file in any programs that use manipulators.

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

int mainControlPrinting03()
{  // The output appears in comment
   // C++ right-justifies the number by the width you specify.

   cout << 456 << 456 << 456 << "\n";     // Prints 456456456
   cout << setw(5) << 456 << setw(5) << 456 << setw(5) << 456 << "\n";     // Prints 456 456 456
   cout << setw(7) << 456 << setw(7) << 456 << setw(7) << 456 << " \n";    // Prints 456 456 456

   cout << setw(2) << 1234567890 << " \n";   //  If you don't specify enough width to output the full
                                                string, C++ ignores the width.

   //  Print values in tabular fashion
   //  Instead of using the tab character, \t, which is limited to eight spaces, this program uses the width
   //  specifier to set the tabs. It ensures that each column is 10 characters wide.

   cout << setw(10) << "Parrots" << setw(10) << "Rams" << setw(10) << "Kings" << setw(10) <<
        "Titans" << setw(10) << "Chargers" << endl ;
   cout << setw(10) << 3 << setw(10) << 5 << setw(10) << 2 << setw(10) << 1 <<
         setw(10) << 0 << "\n";
   cout << setw(10) << 2 << setw(10) << 5 << setw(10) << 1 << setw(10) << 0 <<
         setw(10) << 1 << "\n";
   cout << setw(10) << 2 << setw(10) << 6 << setw(10) << 4 << setw(10) << 3 <<
         setw(10) << 0 << "\n";

   //  Compute and print payroll data
```

NOTE: If you do not specify a width large enough to hold the number, C++ ignores your width request and prints the number in its entirety.

```
    char emp_name[ ] = "Larry Payton";
    char pay_date[ ] = "03/09/92";
    int hours_worked = 43; float rate = 7.75; // Pay per hour
    float tax_rate = .32;                     // Tax percentage rate

    float gross_pay, taxes, net_pay;
    gross_pay = hours_worked * rate;      taxes = tax_rate * gross_pay;
    net_pay = gross_pay - taxes;

    // While prining number by default scientific notation is used. In this case setprecision indicates the
       number of significant digits.
    cout << "As of: " << pay_date << "\n";
    cout << emp_name << " worked " << hours_worked << " hours\n";
    cout << "and got paid " << setw(2) << setprecision(2) << gross_pay << "\n";
    cout << "After taxes of: " << setw(6) << setprecision(2) << taxes << "\n";
    cout << "his take-home pay was $" << setw(8) << setprecision(2) << net_pay << "\n";

    //  In fixed point mode, setprecision indicates  the number of places after the decimal.
    cout << "\n Printing in fixed point mode \n";
    cout << "and got paid " << setw(2) << setprecision(2) << fixed << gross_pay << "\n";
    cout << "After taxes of: " << setw(6) << setprecision(2) << fixed << taxes << "\n";
    cout << "his take-home pay was $" << setw(8) << setprecision(2) << fixed << net_pay << "\n";
    return 0;
}
```

manipulato

Example- 4 : Accepting Keyboard Inputs (mainUserInputs04.cpp)

- In most program environments, the standard input by default is the keyboard, and the C++ stream object defined to access it is **cin**.
- Before every cin, print a prompt that explains exactly what you expect the user to type. cin never check the correctness of inputs. So if wrong input provided program will raise exception.
- You can also use cin to request more than one datum (a piece of information) input from the user:
  cin >> a >> b; **is equivalent to:** **cin >> a; cin >> b;**

- **String Input** **:** When inputting keyboard strings into character arrays with cin, you are limited to receiving one word at a time. The cin does not enable you to type more than one word in a single character array at a time.
- In order to get entire lines in a string variable , we can use the function **getline**, which is the more recommendable way to get user input with cin. getline takes two Parameters :
  - is - istream object from which characters are extracted.
  - str - string object where the extracted line is stored.
  - Return type - type istream. This function return the same input stream as is which it accepts as parameter.

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

int mainUserInputs04 ()
{
   int i; cout << "Please enter an integer value: ";
   cin >> i;       cout << "The value you entered is " << i;
   cout << " and its double is " << i*2 << ".\n";

   char first[20], last[20];
   cout << "What is your first name? ";
   cin >> first;
   cout << "What is your last name? ";
   cin >> last;
   cout << "\n\n"; // Prints two blank lines.
   cout << "In a phone book, your name would look like this:\n";
   cout << last << ", " << first;


   cin.ignore();   //  If you do not give it, next input will be
                          ignored as stated
   string mystr;    cout << "\nWhat's your name? ";
   getline (cin, mystr);
   cout << "Hello " << mystr << ".\n";
   cout << "What is your favorite team? ";
   getline (cin, mystr);
   cout << "I like " << mystr << " too!\n";

   return 0;

}
```

Some problem while using cin and getline

- If you're using getline() after cin >> something, you need to flush the newline character out of the buffer in between. You can do it by using cin.ignore().
- Here we have a cin >> last before the getline(cin, mystr). Then you will see it's not stopping to get the input value from getline. To work it properly you should use cin.ignore()
- This happens because the >> operator leaves a newline \n character in the input buffer and getline(), which reads input until a newline character is found. It will stop reading immediately.

getline() extracts characters from is and stores them into str until the delimitation character delim is found (or the newline character, '\n. Each extracted character is appended to the string.