Basic Input/Output

Until now, the example programs of previous sections provided very little interaction with the user, if any at all. Using the standard input and output library, we will be able to interact with the user by printing messages on the screen and getting the user's input from the keyboard.

C++ uses a convenient abstraction called *streams* to perform input and output operations in sequential media such as the screen or the keyboard. A stream is an object where a program can either insert or extract characters to/from it. We do not really need to care about many specifications about the physical media associated with the stream - we only need to know it will accept or provide characters sequentially.

The standard C++ library includes the header file `iostream`, where the standard input and output stream objects are declared.

Standard Output (cout)

By default, the standard output of a program is the screen, and the C++ stream object defined to access it is `cout`.

`cout` is used in conjunction with the *insertion operator*, which is written as `<<` (two "less than" signs).

```
cout << "Output sentence"; // prints Output sentence on screen
cout << 120;               // prints number 120 on screen
cout << x;                 // prints the content of x on screen
```

The `<<` operator inserts the data that follows it into the stream preceding it. In the examples above it inserted the constant string `Output sentence`, the numerical constant `120` and variable `x` into the standard output stream `cout`. Notice that the sentence in the first instruction is enclosed between double quotes (") because it is a constant string of characters. Whenever we want to use constant strings of characters we must enclose them between double quotes (") so that they can be clearly distinguished from variable names. For example, these two sentences have very different results:

```
cout << "Hello";  // prints Hello
cout << Hello;    // prints the content of Hello variable
```

The insertion operator (`<<`) may be used more than once in a single statement:

```
cout << "Hello, " << "I am " << "a C++ statement";
```

This last statement would print the message `Hello, I am a C++ statement` on the screen. The utility of repeating the insertion operator (`<<`) is demonstrated when we want to print out a combination of variables and constants or more than one variable:

```
cout << "Hello, I am " << age << " years old and my zipcode is " << zipcode;
```

If we assume the `age` variable to contain the value `24` and the `zipcode` variable to contain `90064` the output of the previous statement would be:

```
Hello, I am 24 years old and my zipcode is 90064
```

It is important to notice that `cout` does not add a line break after its output unless we explicitly indicate it, therefore, the following statements:

```
cout << "This is a sentence.";
cout << "This is another sentence.";
```

will be shown on the screen one following the other without any line break between them:

```
This is a sentence.This is another sentence.
```

even though we had written them in two different insertions into `cout`. In order to perform a line break on the output we must explicitly insert a new-line character into `cout`. In C++ a new-line character can be specified as `\n` (backslash, n):

```
cout << "First sentence.\n ";
cout << "Second sentence.\nThird sentence.";
```

This produces the following output:

```
First sentence.
Second sentence.
Third sentence.
```

Additionally, to add a new-line, you may also use the `endl` manipulator. For example:

```
cout << "First sentence." << endl;
cout << "Second sentence." << endl;
```

would print out:

```
First sentence.
Second sentence.
```

The `endl` manipulator produces a newline character, exactly as the insertion of `'\n'` does, but it also has an additional behavior when it is used with buffered streams: the buffer is flushed. Anyway, `cout` will be an unbuffered stream in most cases, so you can generally use both the `\n` escape character and the `endl` manipulator in order to specify a new line without any difference in its behavior.

Standard Input (cin).

The standard input device is usually the keyboard. Handling the standard input in C++ is done by applying the overloaded operator of extraction (`>>`) on the `cin` stream. The operator must be followed by the variable that will store the data that is going to be extracted from the stream. For example:

```
int age;
cin >> age;
```

The first statement declares a variable of type `int` called `age`, and the second one waits for an input from `cin` (the keyboard) in order to store it in this integer variable.

`cin` can only process the input from the keyboard once the `RETURN` key has been pressed. Therefore, even if you request a single character, the extraction from `cin` will not process the input until the user presses `RETURN` after the character has been introduced.

You must always consider the type of the variable that you are using as a container with `cin` extractions. If you request an integer you will get an integer, if you request a character you will get a character and if you request a string of characters you will get a string of characters.

```
// i/o example

#include <iostream>
using namespace std;

int main ()
{
  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";
  return 0;
}
```

```
Please enter an integer value: 702
The value you entered is 702 and its double is
1404.
```

The user of a program may be one of the factors that generate errors even in the simplest programs that use `cin` (like the one we have just seen). Since if you request an integer value and the user introduces a name (which generally is a string of characters), the result may cause your program to misoperate since it is not what we were expecting from the user. So when you use the data input provided by `cin` extractions you will have to trust that the user of your program will be cooperative and that he/she will not introduce his/her name or something similar when an integer value is requested. A little ahead, when we see the `stringstream` class we will see a possible solution for the errors that can be caused by this type of user input.

You can also use cin to request more than one datum input from the user:

```
cin >> a >> b;
```

is equivalent to:

```
cin >> a;
cin >> b;
```

In both cases the user must give two data, one for variable `a` and another one for variable `b` that may be separated by any valid blank separator: a space, a tab character or a newline.

cin and strings

We can use `cin` to get strings with the extraction operator (`>>`) as we do with fundamental data type variables:

```
cin >> mystring;
```

However, as it has been said, `cin` extraction stops reading as soon as if finds any blank space character, so in this case we will be able to get just one word for each extraction. This behavior may or may not be what we want; for example if we want to get a sentence from the user, this extraction operation would not be useful.

In order to get entire lines, we can use the function `getline`, which is the more recommendable way to get user input with `cin`:

```
// cin with strings
#include <iostream>
#include <string>
using namespace std;

int main ()
{
  string mystr;
  cout << "What'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;
}
```

```
What's your name? Juan SouliÃ¯Â¿Â½
Hello Juan SouliÃ¯Â¿Â½.
What is your favorite team? The Isotopes
I like The Isotopes too!
```

Notice how in both calls to `getline` we used the same string identifier (`mystr`). What the program does in the second call is simply to replace the previous content by the new one that is introduced.

stringstream

The standard header file `<sstream>` defines a class called `stringstream` that allows a string-based object to be treated as a stream. This way we can perform extraction or insertion operations from/to strings, which is especially useful to convert strings to numerical values and vice versa. For example, if we want to extract an integer from a string we can write:

```
string mystr ("1204");
int myint;
stringstream(mystr) >> myint;
```

This declares a `string` object with a value of `"1204"`, and an `int` object. Then we use `stringstream`'s constructor to construct an object of this type from the string object. Because we can use `stringstream` objects as if they were streams, we can extract an integer from it as we would have done on `cin` by applying the extractor operator (`>>`) on it followed by a variable of type `int`.

After this piece of code, the variable `myint` will contain the numerical value `1204`.

```
// stringstreams
#include <iostream>
#include <string>
#include <sstream>
using namespace std;

int main ()
{
  string mystr;
  float price=0;
  int quantity=0;

  cout << "Enter price: ";
  getline (cin,mystr);
  stringstream(mystr) >> price;
  cout << "Enter quantity: ";
  getline (cin,mystr);
  stringstream(mystr) >> quantity;
  cout << "Total price: " << price*quantity <<
endl;
  return 0;
}
```

```
Enter price: 22.25
Enter quantity: 7
Total price: 155.75
```

In this example, we acquire numeric values from the standard input indirectly. Instead of extracting numeric values directly from the standard input, we get lines from the standard input (`cin`) into a string object (`mystr`), and then we extract the integer values from this string into a variable of type int (`quantity`).

Using this method, instead of direct extractions of integer values, we have more control over what happens with the input of numeric values from the user, since we are separating the process of obtaining input from the user (we now simply ask for lines) with the interpretation of that input. Therefore, this method is usually preferred to get numerical values from the user in all programs that are intensive in user input.


Input-Output

Output di Bahasa C

- Header stdio.h
- **printf**(<string>,[<variabel>]),
  puts(<string>) atau **putchar**(<char>)

Contoh di C:

```
#include <stdio.h>

void main(){

      char nama[50] = "anton";

      printf("Hallo saya bernama %s, saya sedang belajar C!",nama);

}
```

Output Tidak Terformat

- **putchar(char)** dan **puts(char[]).**
- puts diakhiri dgn enter

Contohnya:

```
#include <stdio.h>
void main()
{     char N,D[15] = "antonius rc";
      N = 'X';
      putchar(N);
      puts(D);
}
```

Hasilnya adalah : **Xantonius rc**

Output Tidak Terformat

- **(+)** Bentuknya sederhana

- **(-)** Tidak tidak dapat digunakan untuk menampilkan bentuk yang rumit
- **(-)** Hanya dapat menggunakan sebuah argumen saja.

Output Terformat

Perintah untuk menampilkan hasil terformat adalah **printf()**

Kode Format	Kegunaan
%c	Menampilkan sebuah karakter
%s	Menampilkan nilai string
%d	Menampilkan nilai desimal integer
%i	Menampilkan nilai desimal integer
%u	Menampilkan nilai desimal integer tak bertanda
%x	Menampilkan nilai heksa desimal integer
%o	Menampilkan nilai oktal integer
%f	Menampilkan nilai pecahan
%e	Menampilkan nilai dalam notasi saintifik
%g	Sebagai pengganti %f atau %e tergantung yg terpendek
%p	Menampilkan suatu alamat memori untuk pointer

Menampilkan Karakter

- Menampilkan karakter di C secara terformat, kita dapat menggunakan **“%c”.**
- Untuk menampilkan sebuah karakter dengan lebar 3 posisi (tiga karakter di depan, karakternya blank), maka gunakan **“%3c”**
- Untuk membuat rata kiri (blank ada di sebelah kanan karakternya) dapat digunakan simbol (*flag*) minus, misalnya “**%-3c**”.

```
#include <stdio.h>

void main(){
      char c = 'a';
      printf("%3c\n",c);
      printf("%-3c\n",c);
}
```

Hasilnya:

```
  a
a
```

Menampilkan String Terformat

FORMAT	KETERANGAN
"%s"	Menampilkan semua nilai karakter pada nilai string
"%Ns"	Menampilkan semua karakter rata kanan dengan lebar N posisi; N adalah konstanta numerik bulat.
"%-Ns"	Menampilkan semua karakter rata kiri dengan lebar N posisi; N adalah konstanta numerik bulat.
"%N.Ms"	Menampilkan rata kanan hanya M buah karakter pertama saja dengan lebar N posisi; M dan N adalah konstanta numerik bulat.
"%-N.Ms"	Menampilkan rata kiri hanya M buah karakter pertama saja dengan lebar N posisi; M dan N adalah konstanta numerik bulat.

Contoh Output Terformat

```
#include <stdio.h>

main()
{
      char D[15] = "Antonius Rachmat C";
      printf("12345678901234567890\n");
      printf("%s\n",D); /* semua karakter, rata kiri */
      printf("%20s\n",D); /* lebar 20, rata kanan */
      printf("%-20s\n",D); /* lebar 20, rata kiri */
      printf("%20.5s\n",D); /* 5 karakter lbr 20, rata kanan */
      printf("%-20.5s\n",D); /* 5 karakter lbr 20, rata kiri */
}
```

Hasil

```
12345678901234567890
Antonius Rachmat C
  Antonius Rachmat C
Antonius Rachmat C
               Anton
Anton
```

Integer Terformat

FORMAT	ARTI
"%d", "%i"	signed int
"%u"	unsigned int
"%ld","%li"	long int
"%hi"	short int
"%hu"	unsgined short int
"%lu"	unsigned long int

Contoh Integer Terformat

```
#include <stdio.h>
main()
{
            int i=1234;
            printf("%i\n",i);
            printf("%5i\n",i);
            printf("%7d\n",i);
            printf("%07d\n",i);
            printf("%-7d\n",i);
}
```

Hasilnya:

```
1234
 1234
   1234
0001234
1234
```

Menampilkan Bilangan Pecahan

FORMAT	ARTI
"%f"	float dgn nilai pecahan
"%e"	float dgn notasi saintifik
"%g"	terpendek dari "%f" atau "%e"
"%lf", "%le" atau "lg"	double
"%Lf", "%Le" atau "Lg"	long double

Contoh Pecahan

```
#include <stdio.h>
void main()
{
   float x=123.4567;
   printf("%3f %15f %020f\n",x,x,x);
   printf("%3e %15e %020e\n",x,x,x);
}
```

Hasilnya:

```
123.456703      123.456703 0000000000123.456703
1.23457e+02     1.23457e+02 0000000001.23457e+02
```

Contoh2:

```
#include <stdio.h>
void main()
{
    float F=12345.6789;
    printf("%15f\n",F);
    printf("%15.2f\n",F);
    printf("%015.2f\n",F);
    printf("%-15.2f\n",F);
    printf("%15.0f\n",F);
}
```

Hasilnya:

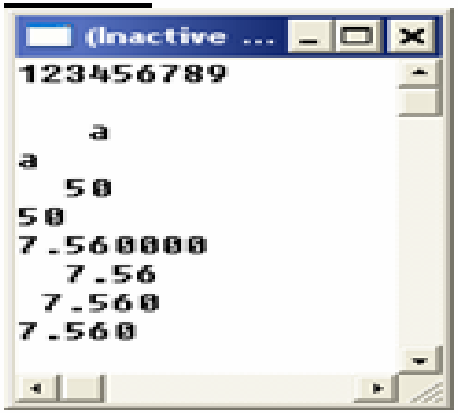
```
    12345.678711
        12345.68
000000012345.68
12345.68
             123456
```

Contoh Pecahan

Contoh3:

```
#include <stdio.h>
void main(){
                printf("123456789\n\n");
                printf("%4c\n",'a');
                printf("%-4c\n",'a');
                printf("%4d\n",50);
                printf("%-4d\n",50);
                printf("%6f\n",7.56);
                printf("%6.2f\n",7.56);
                printf("%6.3f\n",7.56);
                printf("%-6.3f\n",7.56);
}
```

Hasil:



Hexadesimal & Octal

```
#include <stdio.h>
void main()
{
    int x = 1234;
    printf("%x\n",x);     //hasil = 4d2
}
```

```
#include <stdio.h>
void main()
{
      int O=1234;
      printf("%o\n");      //Hasil = 2322
}
```

Membersihkan Layar & Meletakkan Kursor

Menggunakan header **conio.h** dengan fungsi yang bernama **clrscr()**

Contoh :

```
#include <stdio.h>
#include <conio.h>

void main()
{     clrscr();
      printf("Layar sudah bersih...");
}
```

Mengggunakan header **conio.h** dengan fungsi yang bernama **gotoxy()**

Contoh :

```
#include <stdio.h>
#include <conio.h>

void main()
{    clrscr();
     gotoxy(20,1);printf("Layar sudah bersih...");
     gotoxy(20,3);printf("Ini baris ke 3, kolom 20");
     gotoxy(20,5);printf("Ini baris ke 5, kolom 20");
     gotoxy(25,6);printf("Ini kolom 25, baris ke 6");
     gotoxy(20,7);printf("Ini kolom 20, Baris ke 7");
}
```

Input Data

- Header **stdio.h:**
  - **gets()**
  - **scanf()**
- Header **conio.h:**
  - **getche()**
  - **getchar()**
  - **getch()**

Input Data Karakter Tidak Terformat

- getche(): Tanpa Enter, karakter terlihat
- getchar(): Dengan Enter, Karakter terlihat
- getch(): Tanpa Enter, karakter tdk terlihat

Contoh 1:

```
#include <stdio.h>
#include <conio.h>

void main()
{ char hrf;
  printf("Masukkan sebuah karakter : "); hrf = getche();
  printf("\nNilai yang dimasukkan : %c\n",hrf);
}
```

Hasilnya :

```
Masukkan sebuah karakter : N
Nilai yang dimasukkan : N
```

Contoh Input

Contoh 2:

```
#include <stdio.h>
#include <conio.h>

void main()
{       char hrf;
        printf("Masukkan sebuah karakter : ");
        hrf = getchar();
        printf("\nNilai yang dimasukkan : %c\n",hrf);
}
```

Hasilnya :

```
Masukkan sebuah karakter : N↵
Nilai yang dimasukkan : N
```

Contoh 3:

```
#include <stdio.h>
#include <conio.h>

void main()
{       char hrf;
        printf("Masukkan sebuah karakter : ");
        hrf = getch();
        printf("\nNilai yang dimasukkan : %c\n",hrf);
}
```

Hasilnya :

```
Masukkan sebuah karakter :
Nilai yang dimasukkan : N
```

Input Data String tidak terformat

- Untuk memasukkan nilai string dapat dipakai fungsi **gets()**

Contoh:

```
#include <stdio.h>

void main()
{     char kata[10];
      printf("Masukkan suatu nilai string : ");
      gets(kata);
      printf("Nilai string yang dimasukkan : %s\n",kata);
}
```

Hasilnya :

```
Masukkan suatu nilai string : anton
Nilai string yang dimasukkan : anton
```

Input Data Terformat

- **Menggunakan scanf(kodeformat,variabel)**

Kode Format	Kegunaan
%c	Membaca sebuah karakter
%s	Membaca sebuah data string
%d	Membaca sebuah nilai desimal integer
%i	Membaca sebuah nilai desimal integer
%x	Membaca sebuah nilai heksa desimal
%o	integer
%f	Membaca sebuah nilai oktal integer
%e	Membaca sebuah data pecahan
%g	Membaca sebuah data pecahan
	Membaca sebuah data pecahan

Input Data Karakter Terformat

Contoh:

```
    #include <stdio.h>

    void main()
    {    char c1,c2,c3;
         printf("Masukkan 3 nilai karakter : ");
         scanf("%c%c%c",&c1,&c2,&c3);
         printf("\nAnda memasukkan : %c %c %c\n",c1,c2,c3);
    }
```

Hasilnya :

```
Masukkan 3 nilai karakter : abcde
Anda memasukkan : a b c
```

Input Data String Terformat

Contoh:

```
#include <stdio.h>

void main()
{    char kata[10];
     printf("Masukkan suatu nilai string : ");
     scanf("%s",kata);
     printf("Nilai string yang dimasukkan : %s\n",kata);}
```

Hasilnya :

```
Masukkan suatu nilai string : Anton
Nilai string yang dimasukkan : Anton
```

Perhatian

Scanf(<format>,<variabel>):

- Jika string yang dimasukkan memiliki whitespace karakter, maka input string hanya akan dibaca sampai dengan karakter sebelum whitespace saja!

Solusi:

- kode format **“%s”** dapat diganti dengan “**%[^\n]**”
  - Berarti bahwa karakter nilai string akan dibaca terus sampai ditemui penekanan tombol Enter (bentuk ‘^’ menunjukkan maksud ‘tidak’ dan karakter ‘\n’ artinye Enter). Sehingga dengan demikian semua karakter temasuk spasi dan tabulasi akan dibaca sampai ditemui penekanan tombol Enter.
- Atau dengan **gets(<string>)**

Contoh

```
#include <stdio.h>

void main()
{    char kata[20];
     printf("Masukkan suatu nilai string : ");
     scanf("%[^\n]",kata);
     printf("Nilai string yang dimasukkan : %s\n",kata);
}
```

Hasilnya :

```
Masukkan suatu nilai string : Antonius RC
Nilai string yang dimasukkan : Antonius RC
```

Memasukkan Nilai Numerik

- Menggunakan %d untuk integer
- Menggunakan %i untuk integer
- Menggunakan %ld atau %li untuk long integer
- Menggunakan %f untuk double dan float
- Menggunakan %le atau %lf dan %lg untuk long double

Soal-soal

- Buatlah program menghitung luas persegi panjang!
- Buatlah program menghitung luas lingkaran!
- Buatlah program penghitung rumus sebagai berikut:
  - $E = mc^2$
- Buatlah program konversi suhu, dari Celcius, Reamur, dan Farenheit.
  - F = 9/5 * C + 32
  - R = 4/5 * C
- Buatlah program konversi **detik** ke hari, jam, menit, detik!
  - Rumus : 1 hari = 86400 detik; 1 jam = 3600 detik dan 1 menit = 60 detik.

Soal - soal

- Hitung jarak tempuh, dengan kec v, dan waktu t (detik)!
  - S = v * t
- Perkalian 2 pecahan:
  - P1 = ¾
  - P2 = 2/3
  - Hasil = (3 x 2) / (3 x 4)
- Program konversi dolar ke rupiah
  - Gunakan konstanta!
- Menghitung upah gaji per jam seorang pegawai, jika per jam @ 5000!