

BÀI 1. CĂN BẢN VỀ LẬP TRÌNH

I. Lập trình

I.1. Các bước để viết chương trình bằng C++

Để thực hiện việc viết và thực thi một chương trình đơn giản trong C++, người ta thường làm theo các bước sau:

- **Vào môi trường soạn thảo mã lệnh của C++:** để làm được việc này, trên máy tính phải được cài đặt phần mềm được gọi là IDE (thông thường ta sử dụng CodeBlock hoặc Dev-C++). Khi thực thi phần mềm này, nó cung cấp môi trường để soạn thảo và biên dịch mã lệnh.

- **Soạn thảo mã lệnh của chương trình:** IDE cung cấp một cửa sổ soạn thảo và hệ thống menu trợ giúp quá trình soạn thảo cũng như dịch và thực thi chương trình. Ta tiến hành soạn thảo mã lệnh của chương trình trong cửa sổ này theo đúng cú pháp của C++.

- **Soát lỗi, dịch chương trình:** Sau khi soạn thảo mã lệnh bằng ngôn ngữ C++, ta tiến hành dịch chương trình thành ngôn ngữ máy. Quá trình dịch chỉ thành công khi toàn bộ mã lệnh ta soạn thảo không có lỗi cú pháp. Vì vậy, trong quá trình dịch, trình biên dịch sẽ tiến hành soát lỗi. Quá trình soát lỗi được tiến hành lần lượt qua các dòng lệnh từ trên xuống. Khi gặp lỗi, chương trình dịch sẽ báo lỗi tại vị trí gần nơi xảy ra lỗi.

- **Thực thi chương trình:** Khi chương trình đã hết lỗi và được biên dịch thành công, ta có thể thực thi chương trình. Kết thúc quá trình thực thi sẽ quay về môi trường soạn thảo mã lệnh ban đầu.

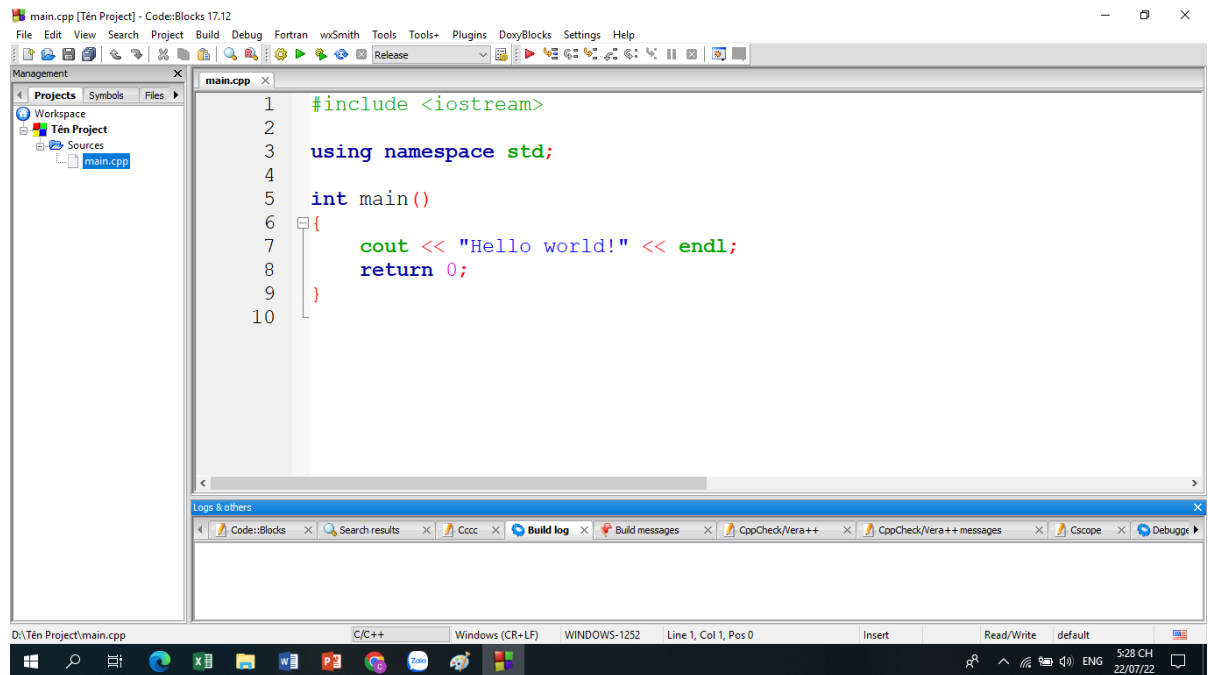
Cài đặt và sử dụng CodeBlock:

- **[1]. Tải bộ cài đặt CodeBlock:** trên trình duyệt internet, hãy gõ: Codeblock. Truy cập trang Codeblocks.org/ Downloads/ **Download the binary release/** Windows XP / Vista / 7 / 8.x / 10 (nếu bạn sử dụng hệ điều hành Windows)/ Download from và lựa chọn một trang web nơi có thể tải bộ cài đặt CodeBlock.

Chú ý: Thông thường, bạn cần cài đặt cả trình biên dịch đi kèm. Hãy lựa chọn và tải về file mà tên của nó có chứa “mingw-setup”. Ví dụ: “codeblocks-20.03mingw-setup.exe”.

- **[2]. Giao diện CodeBlock:**

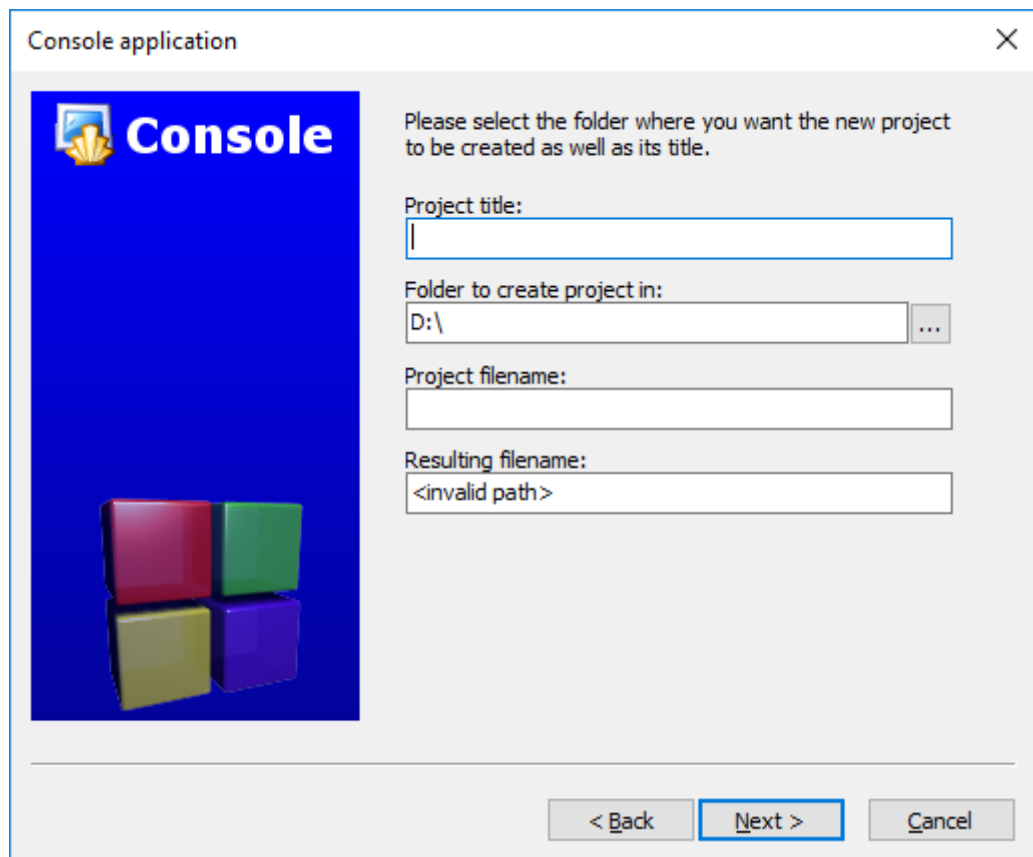
Giao diện codeblock gồm 4 phần chính:



Hình 1.1. Giao diện chính của CodeBlock




- Thanh menu: Được đặt phía trên cùng, nơi chứa các chức năng của CodeBlock như: File, Edit, View,...
- Thanh công cụ (Tool bars): Được đặt ngay dưới menu, nơi chứa các biểu tượng cho phép truy cập nhanh tới các chức năng. Để bật/ tắt thanh công cụ, ta chọn View/ Toolbars/...
- Cửa sổ chính: Nơi ta có thể soạn thảo mã lệnh của chương trình.
- Cửa sổ quản lý project/ file... Thông thường được đặt phía tay trái. Để bật/ tắt cửa sổ này, ta chọn View/ Manager hoặc bấm tổ hợp phím Shift + F2.
- Cửa sổ quản lý các thông báo khi biên dịch (như thông báo lỗi Build Log): thường được đặt phía dưới màn hình. Để bật tắt cửa sổ này, ta chọn View/ Logs hoặc bấm phím F2.
- **[3]. Tạo một project:** Thông thường, mỗi chương trình sẽ được viết trong một project riêng biệt để dễ quản lý. Để tạo một project, ta thực hiện như sau:
 - Chọn File/ New/ Project...
 - Chọn Console application/ Go/ Next/ C++.
 - Đặt tên Project trong ô "Project title".

- Chọn thư mục chứa project trong ô “Folder to create project in”. Hãy bấm vào biểu tượng “...” bên tay phải của ô để chọn thư mục.
- Chọn Next/ Finish.



Hình 1.2. Cửa sổ tạo project

Chú ý: Khi project được tạo thành công, cửa sổ quản lý project (bên tay trái) sẽ có thông tin về project vừa tạo. Hãy kích đúp (double click) vào tên project, chọn Sources, kích đúp vào main.cpp để bắt đầu soạn thảo chương trình.

- [4]. Soát lỗi, biên dịch và thực thi chương trình: Khi soạn thảo mã lệnh hoàn thành, hãy soát lỗi và biên dịch (hai thao tác này được thực hiện cùng nhau). Để làm việc này, trên thanh công cụ, hãy chọn:
 - Soát lỗi và dịch chương trình: 
 - Thực thi chương trình: 
 - Soát lỗi, dịch, và thực thi chương trình (nếu dịch thành công):  .

Ta cũng có thể thực hiện các chức năng trên trong menu Build: Build, Run, Build and run và các tổ hợp phím tắt kèm theo.

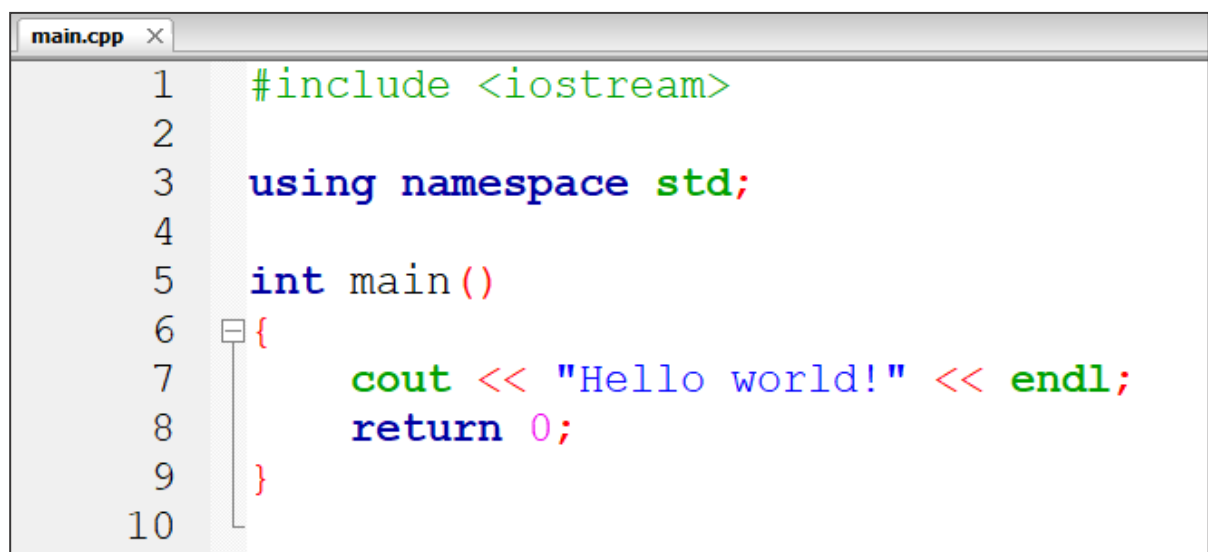
I.2. Cấu trúc một chương trình đơn giản trong C++

Một chương trình đơn giản trong C++ thường được viết trong một cửa sổ soạn thảo (điều này không hoàn toàn đúng trong các chương trình lớn).

Một chương trình bất kỳ trong C++ đều được tạo nên từ rất nhiều hàm (tạm gọi là những đơn vị chương trình), trong đó có một hàm đặc biệt được xem như “chương trình chính” và được gọi là hàm main. Thông thường hàm main được viết ra để gọi (sử dụng) các hàm khác nhằm giải quyết bài toán.

Khi thực thi chương trình, chương trình sẽ bắt đầu từ hàm main. Nếu hàm main có gọi tới các hàm khác, chúng sẽ được thực hiện.

Tuy nhiên, ở đây ta chỉ xét một chương trình đơn giản bao gồm duy nhất một hàm main, ta viết theo cấu trúc sau:

A screenshot of a code editor window titled 'main.cpp'. The code is as follows:

```
1  #include <iostream>
2
3  using namespace std;
4
5  int main()
6  {
7      cout << "Hello world!" << endl;
8      return 0;
9  }
```

Line numbers 1 through 10 are visible on the left margin.

Hình 1.3. Cấu trúc một chương trình đơn giản trong C++

Dòng 1: được gọi là các chỉ thị tiền xử lý. Dòng này có nhiệm vụ khai báo các thư viện sẽ sử dụng trong chương trình.

Chú ý: các thư viện của ngôn ngữ lập trình C thông thường có đuôi .h. Ví dụ conio.h, stdio.h, math.h..., trong khi các thư viện của C++ thường được viết theo hướng đối tượng sẽ không có đuôi nay. Ví dụ: iostream, iomanip, fstream,... Ta cũng có thể sử dụng duy nhất một thư viện “**bits/stdc++.h**” thay vì phải khai báo sử dụng nhiều thư viện khác nhau.

Dòng 3: Khai báo sử dụng “không gian tên” chuẩn của C++. Khai báo này giúp chúng ta có thể sử dụng một số hàm chuẩn có sẵn mà không cần phải đặt tiền tố std :: trước mỗi thứ.

Dòng 5 đến 9: Là khai báo hàm main, cũng còn được gọi là chương trình chính. Các dấu ‘{’ và ‘}’ báo hiệu bắt đầu và kết thúc thân hàm main hoặc bắt đầu và kết thúc một khối lệnh.

Chú ý: Trong C++ có phân biệt chữ hoa và chữ thường. Sau mỗi lệnh đều có dấu chấm phẩy “;” để báo kết thúc lệnh.

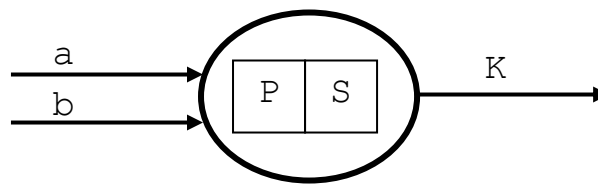
II. Biến, biểu thức, các lệnh nhập xuất

II.1. Biến

Để hiểu khái niệm và bản chất biến, ta xét chương trình đơn giản sau:

Nhập vào các số nguyên a, b. Gọi P là tổng của a và b, S là tích của a và b. Tính $K = S * P / (S + P)$.

Ta dễ dàng biểu diễn bài toán bằng mô hình sau:



Có thể coi đầu vào của bài toán là a, b và đầu ra là K. Các giá trị P và S là các giá trị trung gian.

Khi người dùng nhập vào các giá trị a và b, chúng cần được lưu trữ trong bộ nhớ. Tương tự như vậy các giá trị P, S, K nếu cần cũng sẽ được lưu trữ trong bộ nhớ. Vậy tối đa ta cần sử dụng 5 ô nhớ để lưu trữ chúng. Các ô nhớ này gọi là các biến.

Biến là một vùng nhớ được đặt tên.

Mỗi ô nhớ (biến) đều được đặt 1 tên tuân theo quy ước đặt tên như sau:

- Tên biến bao gồm chữ cái, chữ số hoặc dấu gạch nối ‘_’
- Ký tự đầu tiên của tên biến không được là một chữ số.
- Không được trùng với từ khoá.
- Trong cùng một phạm vi không có 2 biến trùng tên.

Trong chương trình, ta sử dụng tên biến để truy cập vào ô nhớ của biến. Khi đó tên biến chính là giá trị đang chứa trong ô nhớ của nó.

Tất cả các biến khi sử dụng đều phải khai báo. Cú pháp như sau:

<kiểu biến> <tên biến>;

Trong đó:

<kiểu biến>: mỗi biến dùng để chứa một loại giá trị khác nhau như: số nguyên, số thực, ký tự .v.v., do vậy chúng phải có kiểu tương ứng. Một số kiểu cơ bản như sau:

Kiểu số: bao gồm

+ *Số nguyên int/ short int*: là kiểu dữ liệu có độ dài 2 byte, dùng để khai báo các biến nguyên có giá trị trong khoảng $-32768 \rightarrow 32767$

+ *Số nguyên không dấu: unsigned int*: độ dài 2 byte, khai báo các biến nguyên có giá trị từ 0 tới 65535.

+ *Số nguyên dài long*: là kiểu dữ liệu có độ dài 4 byte, dùng khai báo các biến nguyên có giá trị trong khoảng $-2.147.483.648 \rightarrow 2.147.483.647$.

+ *Số nguyên dài không dấu: unsigned long*: độ dài 4 byte, khai báo các biến có giá trị từ 0 tới 4.294.967.295.

+ *Số thực (dấu phẩy động) float*: kích thước 4 byte khai báo các biến thực từ $3.4 \cdot 10^{-38} \rightarrow 3.4 \cdot 10^{38}$.

+ *Số thực (dấu phẩy động, độ chính xác kép) double*: kích thước 8 byte, có phạm vi từ $1.7 \cdot 10^{-308} \rightarrow 1.7 \cdot 10^{308}$

+ *Số thực (dấu phẩy động, độ chính xác kép) dài long double*: kích thước 10 byte, khai báo các biến từ $3.4 \cdot 10^{-4932}$ tới $1.1 \cdot 10^{4932}$.

Kiểu ký tự: char, khai báo biến chứa một ký tự.

Biểu Boolean: bool, biến có kích thước 1 bit, chỉ nhận một trong hai giá trị **true** (1) hoặc **false** (0).

Chú ý: Kích thước của một (ô nhớ dành cho) biến được tính trên hệ điều hành 16 bits. Với các hệ điều hành 32 hoặc 64 bits, kích thước này không còn đúng nữa.

<tên biến>: được đặt tùy ý tuân theo các quy ước đặt tên biến ở trên.

Ví dụ: `int a, b; float c;`

Ở đây, ta khai báo 2 biến a và b có cùng kiểu số nguyên và biến c có kiểu số thực. Khi đó chương trình sẽ cấp phát 2 ô nhớ có kích thước 2 byte mỗi ô và đặt tên là a, b; một ô nhớ kích thước 4 byte được đặt tên c.

Vị trí khai báo: Có thể khai báo biến tại bất kỳ đâu trong chương trình, trước khi sử dụng.

II.2. Biểu thức

Một biểu thức bao gồm 2 thành phần: các toán tử (phép toán) và các toán hạng (số hạng).

Các toán tử:

Trong C++, các toán tử được tạm phân chia làm 4 loại theo chức năng của chúng. Sau đây là một số toán tử hay dùng:

- Các toán tử số học:

Stt	Toán tử	Cách viết
1	Cộng	+
2	Trừ	-
3	Nhân	*
4	Chia	/
5	Đồng dư	%
6	Tăng 1 đơn vị	++
7	Giảm 1 đơn vị	--

- Các toán tử Logic:

Stt	Toán tử	Cách viết
1	Và	&&
2	Hoặc	
3	Phủ định	!

- Các toán tử so sánh:

Stt	Toán tử	Cách viết
1	Lớn hơn	>
2	Nhỏ hơn	<
3	Lớn hơn hoặc bằng	>=
4	Nhỏ hơn hoặc bằng	<=
5	Bằng	==
6	Không bằng	!=

- Toán tử gán:

Stt	Toán tử	Cách viết
1	Gán	=

Các toán hạng:

Có thể chia các toán hạng làm 3 loại gồm: hằng, biến và hàm.

Hằng: gồm hằng số, hằng xâu ký tự và hằng ký tự. Hằng xâu ký tự khi viết cần được đặt giữa hai dấu nháy kép “”; hằng ký tự được đặt giữa hai dấu nháy đơn ‘’ còn hằng số thì không.

Hàm: gồm những hàm trả về một giá trị nào đó và giá trị này được sử dụng trong biểu thức. Có rất nhiều hàm có sẵn trong các thư viện mà ta có thể sử dụng cho biểu thức. Sau đây là một số hàm toán học (thư viện math.h) thường dùng:

STT	Tên hàm	Cách viết
1.	Sin(x)	sin(x)
2.	Cos(x)	cos(x)
3.	\sqrt{x}	sqrt(x)
4.	e^x	exp(x)
5.	Ln(x)	log(x)
6.	Log ₁₀ (x)	log10(x)
7.	x	fabs(x)

Ví dụ: xét biểu thức toán học sau

$$((2e^x + |x|.Ln(x)) > \sqrt{x} / \sin(x)) \wedge (x < 5)$$

Biểu thức được viết dưới dạng ngôn ngữ C++ như sau:

$$(2*\exp(x) + fabs(x)*\log(x) > \sqrt{x} / \sin(x)) \&\& (x < 5)$$

Trong biểu thức này, các toán tử số học gồm: +, *. Toán tử logic: &&. Các toán tử so sánh gồm: >, <. Các toán hạng là hằng số gồm: 2, 5. Toán hạng là biến: x. Các toán hạng là hàm gồm: exp(x), fabs(x), log(x), sqrt(x), sin(x).

II.3. Các lệnh nhập-xuất

a. Các lệnh nhập xuất trong IOStream

- **Lệnh xuất:**

cout<< <Dữ liệu>;

Trong đó:

<<: được gọi là toán tử xuất.

<Dữ liệu>: có thể là Hằng ký tự, Hằng xâu ký tự, Biến, Hàm, biểu thức hoặc phương thức định dạng.

Ví dụ: cout<<"Sin(x) ="; cout<<sin(x);

Có thể sử dụng liên tiếp nhiều toán tử xuất trên một dòng cout, chẳng hạn:

cout<<"Sin(x) ="<<sin(x);

Nếu muốn xuất dữ liệu trên nhiều dòng ta có thể sử dụng toán tử endl để xuống dòng. Ví dụ: cout<<"Sin(x) ="<<endl<<sin(x); sẽ xuất dữ liệu trên 2 dòng.

- **Định dạng dữ liệu trước khi xuất:**

Ta có thể sử dụng một trong 2 cách sau:

Cách 1: sử dụng phương thức định dạng:

cout.width(int n): chỉ định tối thiểu n vị trí dành cho việc xuất dữ liệu.

Nếu giá trị xuất chiếm ít hơn n vị trí thì mặc định là các ký tự trống sẽ chèn vào phía trước. Nếu giá trị xuất chiếm nhiều hơn n vị trí, số vị trí dành cho giá trị xuất đó sẽ được tăng lên sao vừa đủ thể hiện giá trị xuất.

cout.fill(char ch): Chỉ định ký tự ch sẽ được điền vào những vị trí trống (nếu có).

cout.precision(int n): chỉ định độ chính xác của giá trị số khi xuất là n ký tự phần thập phân.

Ví dụ: giả sử ta có biến thực a: float a = 123.4523; Nếu muốn xuất a ra màn hình dưới dạng: 000123.45 ta có thể định dạng như sau:

```
cout.width(9);
cout.fill('0');
cout.precision(2);
cout<<a;
```

Cách 2: sử dụng các cờ định dạng:

Tương tự như các phương thức định dạng, các cờ định dạng tương ứng là:

setw(int n) – tương tự như cout.width(int n).

setfill(char ch) – tương tự như cout.fill(char ch).

setprecision(int n) – tương tự như cout.precision(int n).

Cách dùng: sử dụng các cờ định dạng ngay trên các dòng cout, trước khi đưa ra giá trị xuất. Với ví dụ trên, ta có thể viết:

```
cout<<setw(9)<<setfill('0')<<setprecision(2)<<a;
```

- **Lệnh nhập:**

```
cin >> (Biến);
```

Trong đó:

>>: được gọi là toán tử nhập.

Dòng cin dùng để nhập các giá trị (thông thường là) từ bàn phím vào các biến.

Ví dụ: để nhập giá trị cho biến a, ta viết:

```
cout<< "a=";      cin>>a;
```

Có thể dùng liên tiếp nhiều toán tử nhập trên một dòng cin để nhập giá trị cho nhiều biến, chẳng hạn:

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

Lệnh `cin` chỉ thích hợp cho việc nhập các biến kiểu số. Với các biến kiểu xâu ký tự thì xâu nhập vào phải không chứa dấu cách vì lệnh `cin` sẽ kết thúc khi ta nhập vào dấu cách hoặc phím Enter.

Ví dụ: Viết chương trình nhập vào một số thực x , in ra màn hình giá trị của $F(x) = \sin^2(x) + |x| + e^{\ln(x)}$ với độ chính xác 2 chữ số sau dấu phẩy.

```
#include <conio.h>
#include <math.h>
#include <iostream>
using namespace std;
int main()
{
    float x, F;
    cout<<"nhập số thực x: "; cin>>x;
    cout.precision(2);
    cout<<"Giá trị F("<x<<") =";
    cout<<sin(x)*sin(x) + fabs(x) + exp(log(x));
    return 0;
}
```

b. Các lệnh nhập xuất trong `stdio.h`

- Lệnh xuất: `printf("chuỗi cần xuất" , <Biến 1>, <Biến 2>...);`

Trong đó:

- “chuỗi cần xuất” có thể gồm:
- Hằng ký tự, hằng xâu ký tự: Là các ký tự cần in lên màn hình.
- Các đặc tả hay ký tự đại diện, bao gồm:

`%d`: đại diện cho biến nguyên.

`%f`: đại diện cho biến thực.

`%c`: đại diện cho biến kiểu ký tự (mặc định).

...

- Mỗi biến cần đưa ra màn hình cần có một đặc tả tương ứng tại vị trí muốn đưa ra, như vậy số lượng biến bằng số lượng các đặc tả.

Ví dụ: Cần đưa ra các giá trị của các biến a , b , c kiểu nguyên, ta viết:

```
printf ("Giá trị của a b c là %d %d %d", a, b, c);
```

- Lệnh nhập: `scanf("chuỗi các đặc tả", &<Biến 1>, &<Biến 2>...);`

Trong đó, mỗi biến cần phải có một đặc tả tương ứng. Lệnh scanf nhập giá trị vào các biến thông qua địa chỉ của biến. Toán tử & được đặt trước tên biến để lấy địa chỉ của biến.

c. Các lệnh nhập xuất trong Conio.h

- Lệnh xuất: **puts(p);**

Trong đó p là một con trỏ chuỗi ký tự, tức p có kiểu char* hoặc là một mảng kiểu char. Lệnh puts(p) sẽ đưa các ký tự được con trỏ p trỏ tới lên màn hình.

- Lệnh nhập: **gets(p);**

Trong đó p là một con trỏ chuỗi ký tự, tức p có kiểu char* hoặc là một mảng kiểu char. Lệnh gets(p); có chức năng cho phép người sử dụng nhập vào một chuỗi ký tự và chứa chuỗi vừa nhập vào biến p.

Nếu sử dụng liên tiếp nhiều lệnh gets thì xen giữa các lệnh gets ta cần làm sạch bộ đệm bàn phím bằng lệnh: **fflush(stdin);**

Các lệnh gets, puts thích hợp cho việc nhập xuất các biến kiểu chuỗi ký tự.

Ví dụ sau đây minh họa các sử dụng các lệnh nhập xuất một cách phù hợp cho từng loại biến:

Nhập vào Họ tên, quê quán, số ngày công của một công nhân. In các thông tin vừa nhập lên màn hình kèm theo tiền công, biết rằng mỗi ngày công được trả 50.000Đ.

```
#include "iostream"
#include "stdio.h"
#include "conio.h"
using namespace std;
int main()
{
    char HoTen[30]; char Que[50];
    int NgayCong;
    cout<<"Nhập họ tên: "; gets(HoTen);
    fflush(stdin);
    cout<<"Nhập quê quán: "; gets(Que);
    cout<<"Nhập ngày công: "; cin>>NgayCong;
    long Luong = (long) NgayCong*50000;
    cout<<"Thông tin vừa nhập là: "<<endl;
    cout<<"Họ tên:"<<HoTen<<endl<<"Que:"<<Que<<endl;
    cout<<"Ngày công:"<<NgayCong<<endl<<"Luong:"<<Luong;
    return 0;
}
```

Vì `NgayCong` là biến kiểu `int` nên khi ta nhân với 50000 sẽ được một con số thuộc kiểu `int`. Tuy nhiên con số này quá lớn so với dữ liệu kiểu `int` nên khi gán sang biến `long Luong` ta cần chuyển nó về kiểu `long` bằng cách viết: `(long) NgayCong*50000`; Cách viết này gọi là ép kiểu.

Để ép kiểu một biểu thức ta viết: **(<kiểu>) <Biểu thức>;**

Đặc biệt trong C++ luôn quy định phép chia một số nguyên cho một số nguyên sẽ thu được thương cũng là một số nguyên. Vì vậy muốn thu được thương là số thực ta cần ép kiểu thương số này.

Ví dụ: với `n` nguyên dương, phép chia `1/n` sẽ cho ta kết quả là một số nguyên (lấy phần nguyên của thương). Để lấy kết quả là số thực ta viết: `(float) 1/n`.