

LẬP TRÌNH C++

Đây là những nội dung cơ bản SV cần nắm vững:

Chương I: Giới thiệu

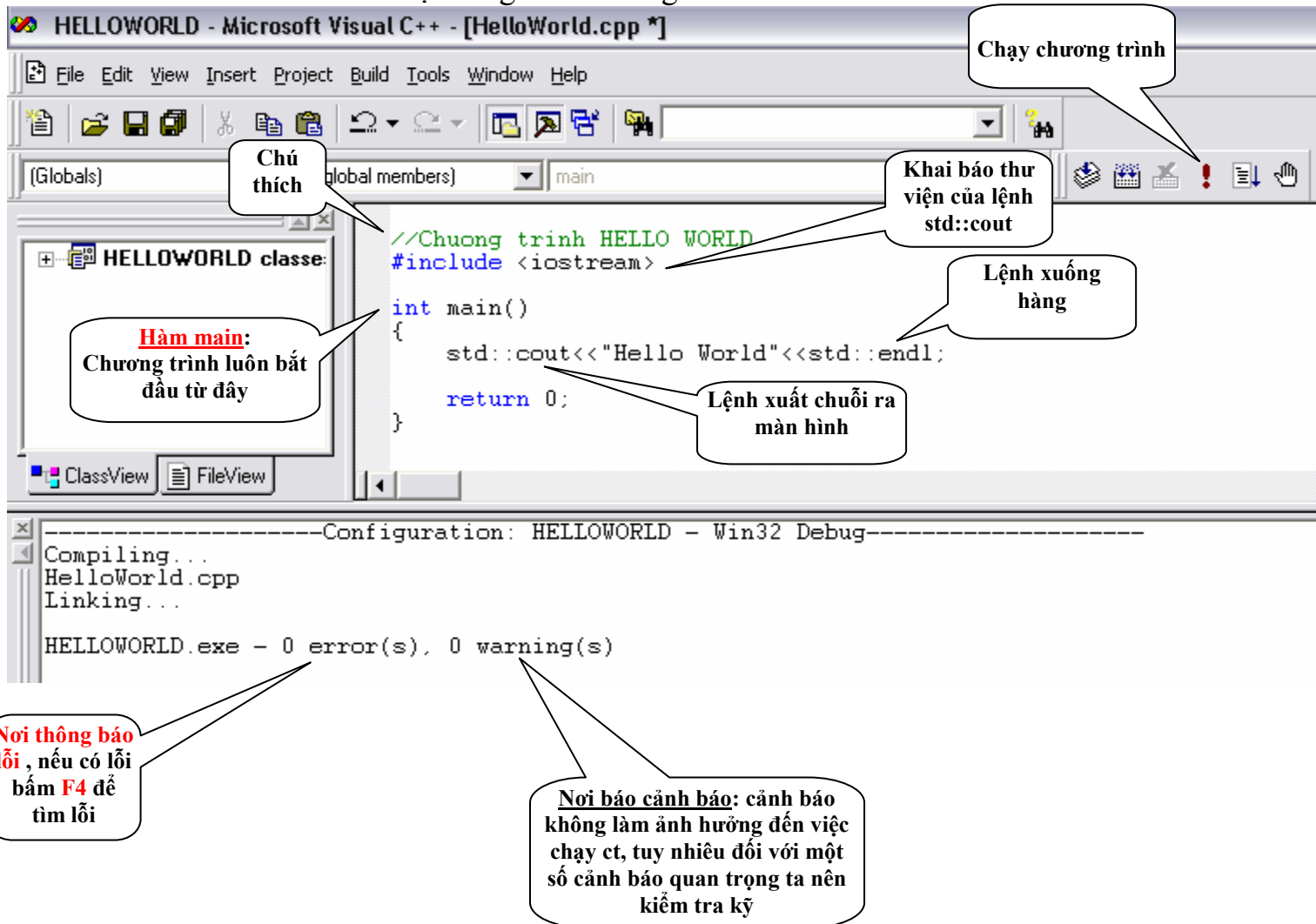
Tin học đại cương (hay lập trình C++), nói chung tinh thần của môn học là cũng dựa phần lớn vào cách lập trình mà chúng đã từng nghiên cứu ở tin học đại cương A1 (lập trình Pascal). Vì vậy đa số những lệnh có trong pascal chúng ta sẽ gặp lại trong ngôn ngữ C++, mặc dù có một số điểm sai khác (như cấu pháp), nhưng nhìn chung mục đích của những lệnh đó hoàn toàn giống với ngôn ngữ Pascal mà chúng ta đã học. Đối với những tập lệnh này chúng ta sẽ không nghiên cứu chi tiết về mặt lý thuyết, mà chủ yếu tập trung vào phần thực hành, vận dụng (có thể dùng phần mềm **chuyển ngôn ngữ từ pascal sang C++** để hiểu rõ hơn - download tại <http://fit.hcmup.edu.vn/~thqthu>).

Riêng với những tập lệnh đặc thù riêng của C++ chúng ta sẽ nghiên cứu kỹ hơn về phân lý thuyết.

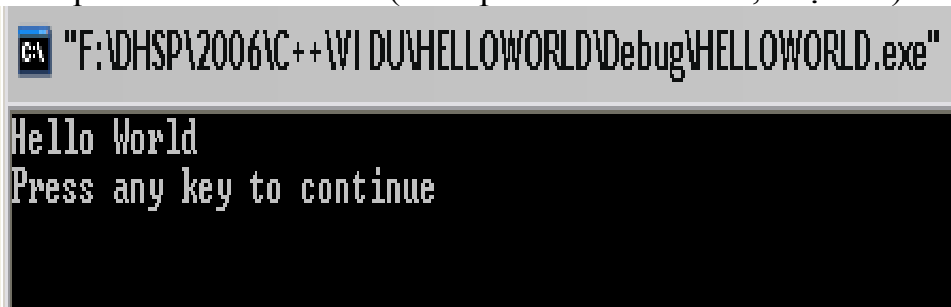
- + Ngôn ngữ lập trình: C++
- + Môi trường lập trình: Visual C++ 6.0 hoặc Dev-Cpp
- + Bộ giúp đỡ ngôn ngữ (help): MSDN 2001

1.1. Tạo và biên dịch một chương trình C++ trên môi trường Visual C++

Phân tích ví dụ đơn giản: Chương trình “Hello World”



Kết quả xuất ra màn hình (Bấm phím **CTRL + F5**, hoặc **F5**)



Một số yêu cầu khi trình bày soạn thảo một chương trình (bắt buộc)

The image shows a C++ IDE window with a code editor. The code is as follows:

```
//Chương trình HELLO WORLD
#include <iostream>
#include <conio.h>

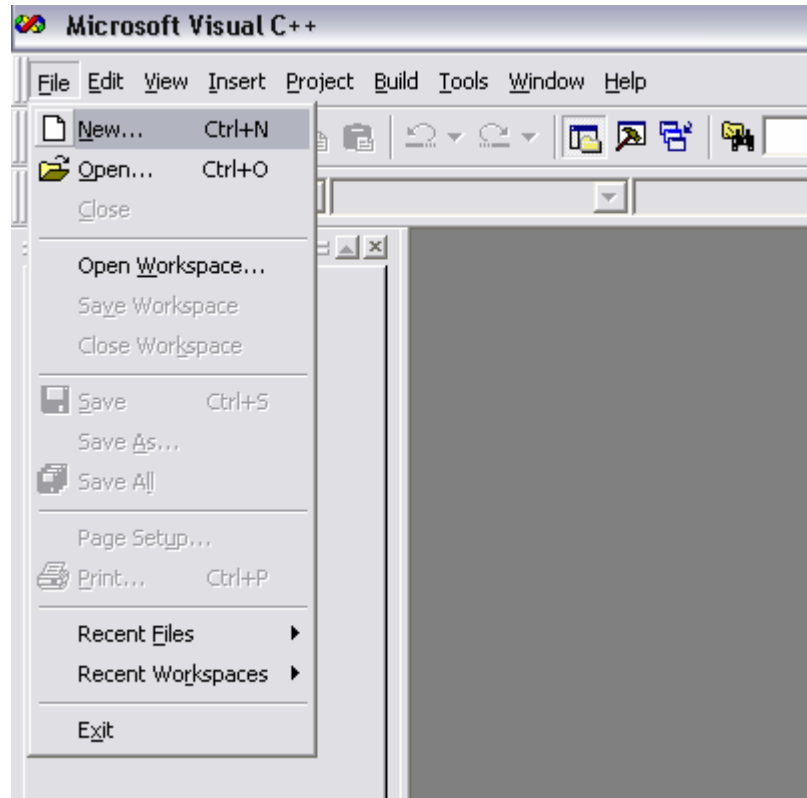
int main()
{
    //Xuất ra màn hình
    std::cout<<"Hello World"<<std::endl;
    std::cout<<"Chuc mung ban den voi C++"<<std::endl;
    getch();
    return 0;
}
```

Annotations (speech bubbles) provide the following instructions:

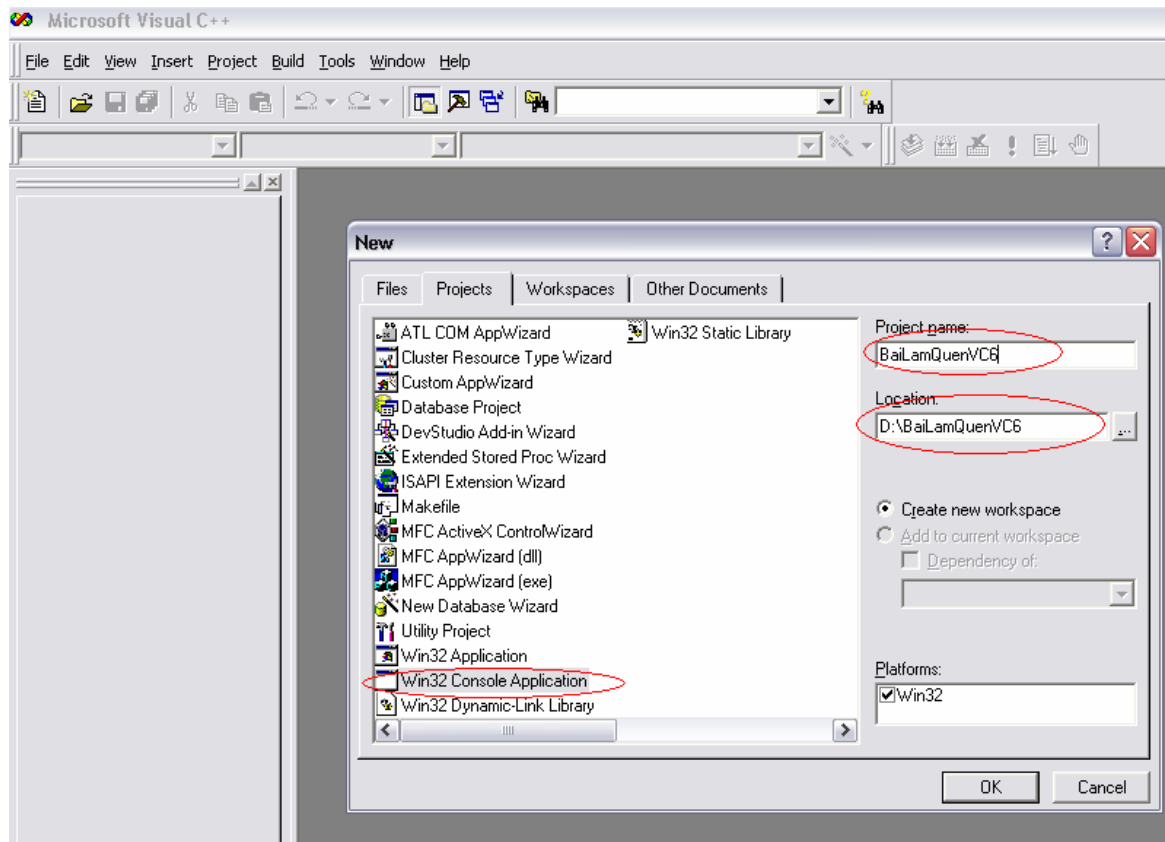
- Nơi khai báo thư viện và hàm main cách nhau **1 dòng****: Points to the space between the include statements and the main function definition.
- Nên có chú thích ở các dòng lệnh quan trọng**: Points to the comment line above the first cout statement.
- Sử dụng phím **Tab** để lồng các dòng lệnh với nhau**: Points to the indentation of the code inside the main function.
- Các dòng lệnh có ý nghĩa khác nhau nên cách nhau **1 dòng****: Points to the space between the getch() and return 0; statements.

1.2 Làm quen với môi trường lập trình Visual C++ 6.0

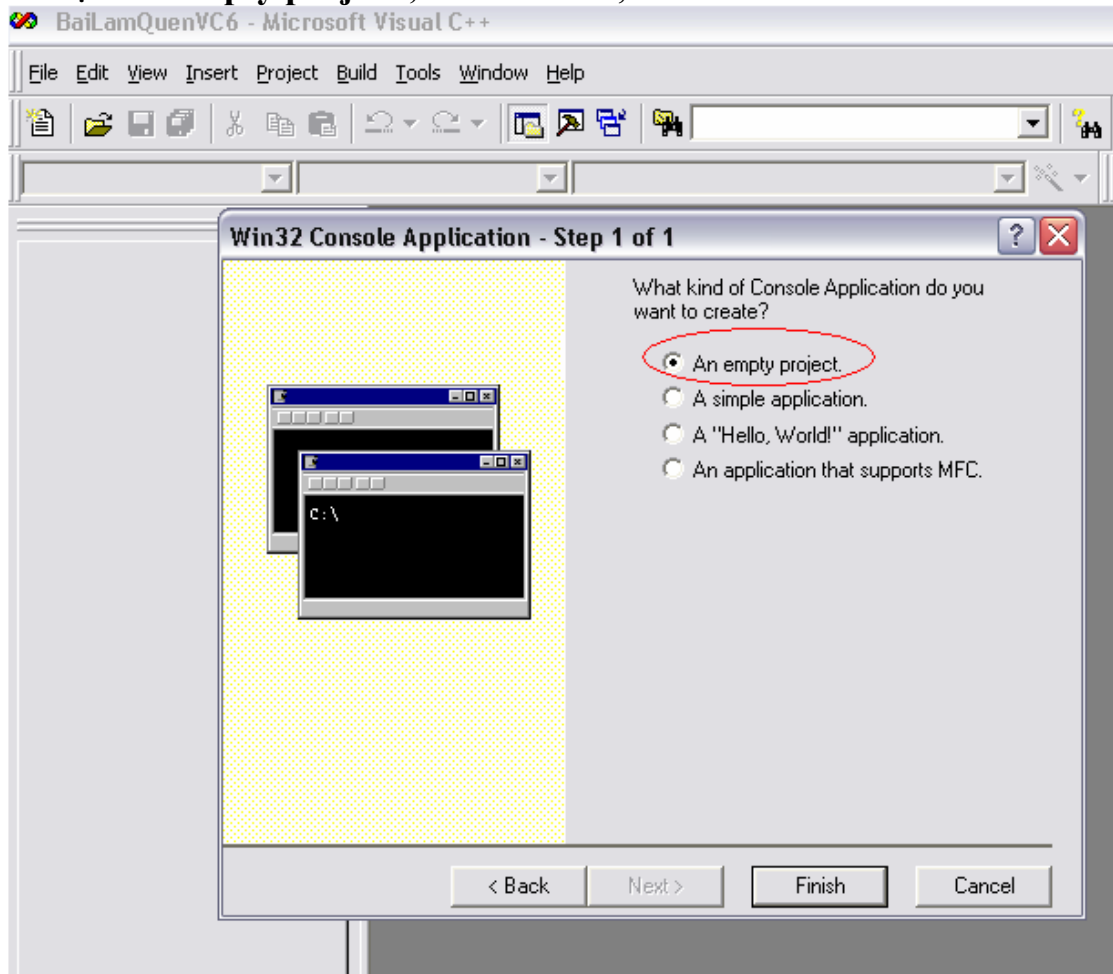
1. Chạy chương trình **Microsoft Visual C++ 6.0**
2. Vào File -> New



3. Tại thẻ **Projects**, chọn **Win32 Console Application**, đánh tên bài làm vào **project name** (ở đây là “BaiLamQuenVC6”), chọn nơi lưu trữ bài ở **Location** (ở đây chọn lưu ở D:\BaiLamQuenVC6) => **ok**

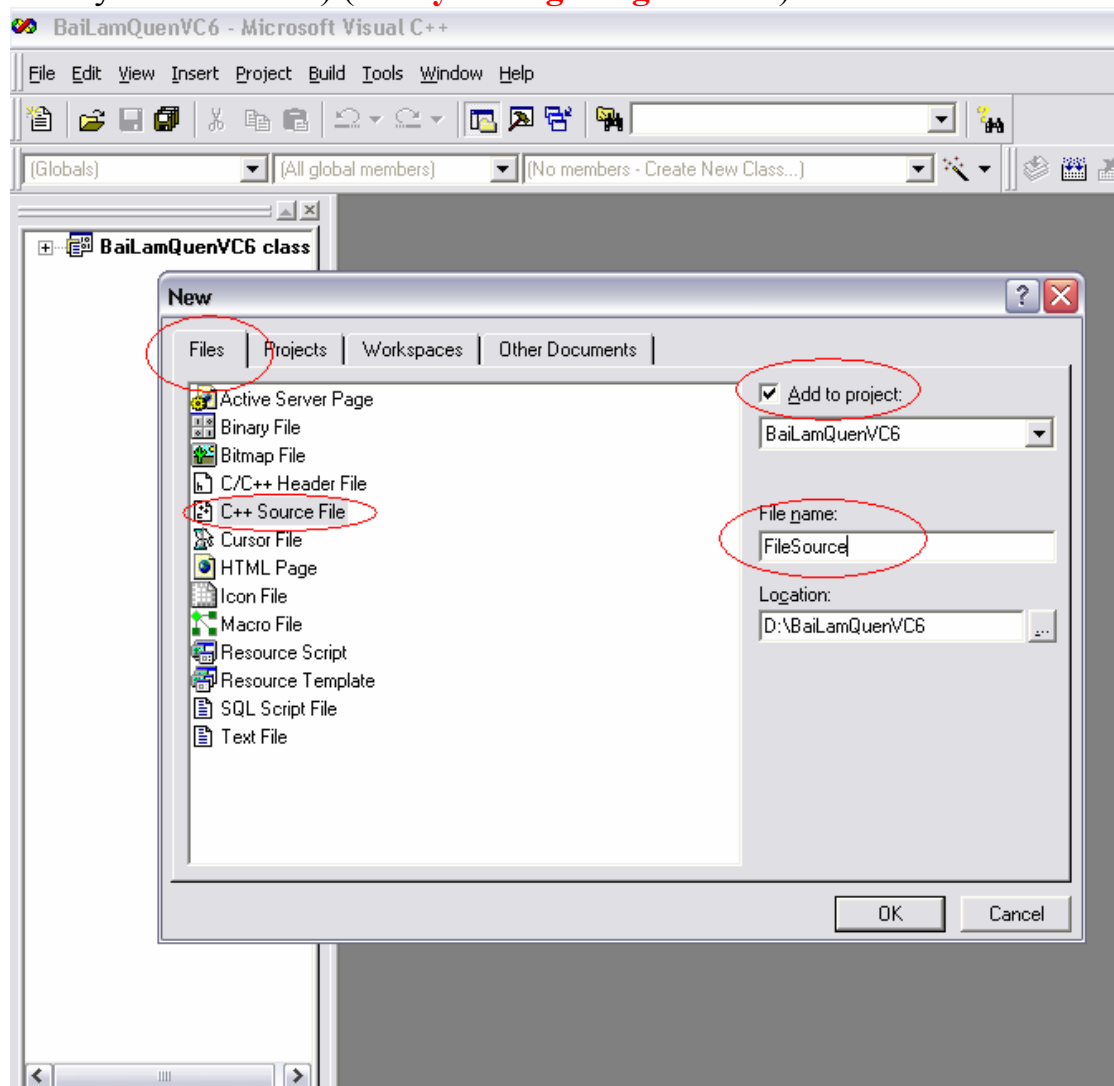


4. Chọn **An empty project** , nhấn **finish** , nhấn **ok**

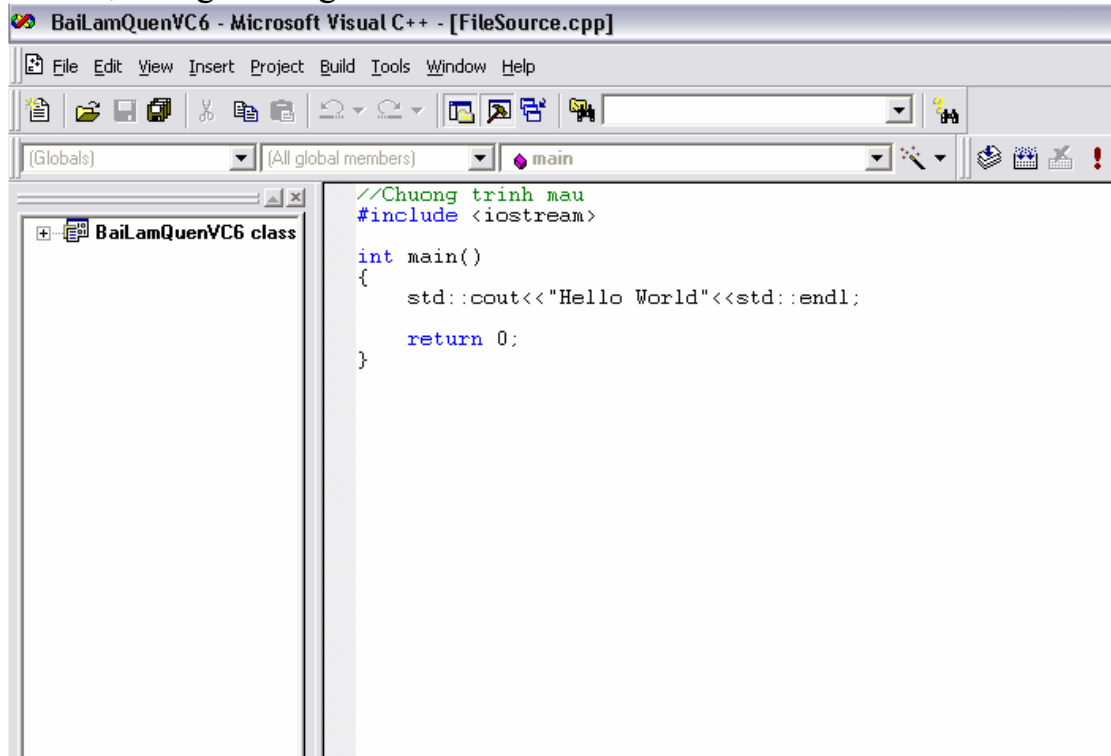


5. Vào **File -> New**

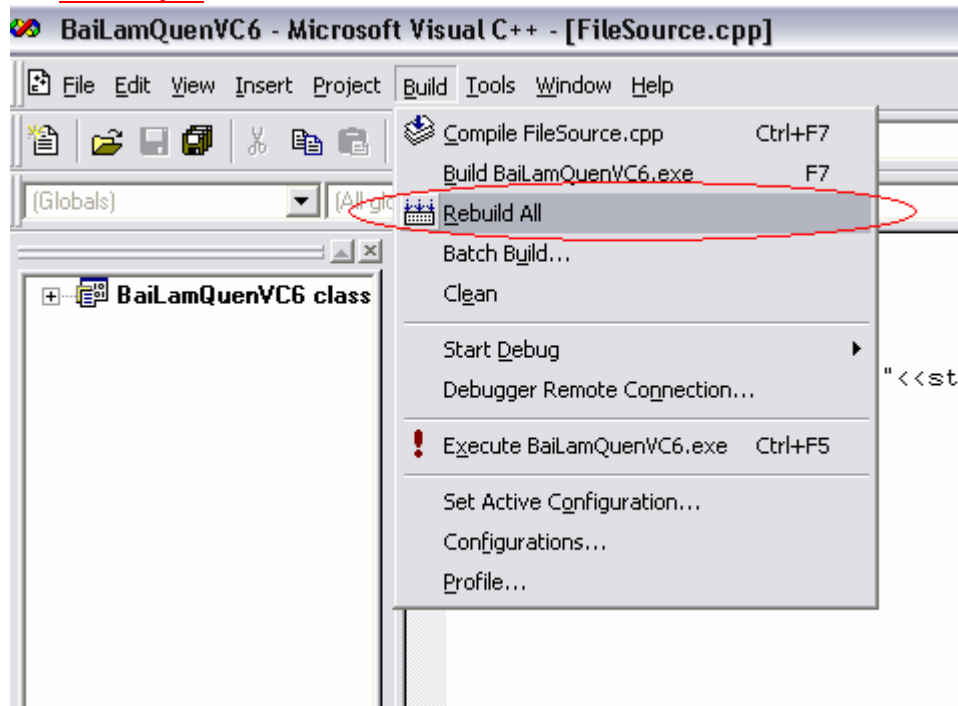
6. Tại thẻ **File**, ta chọn **C++ source file**, và đánh tên file vào **File Name** (ở đây là FileSource) (**Chú ý những vòng màu đỏ**)



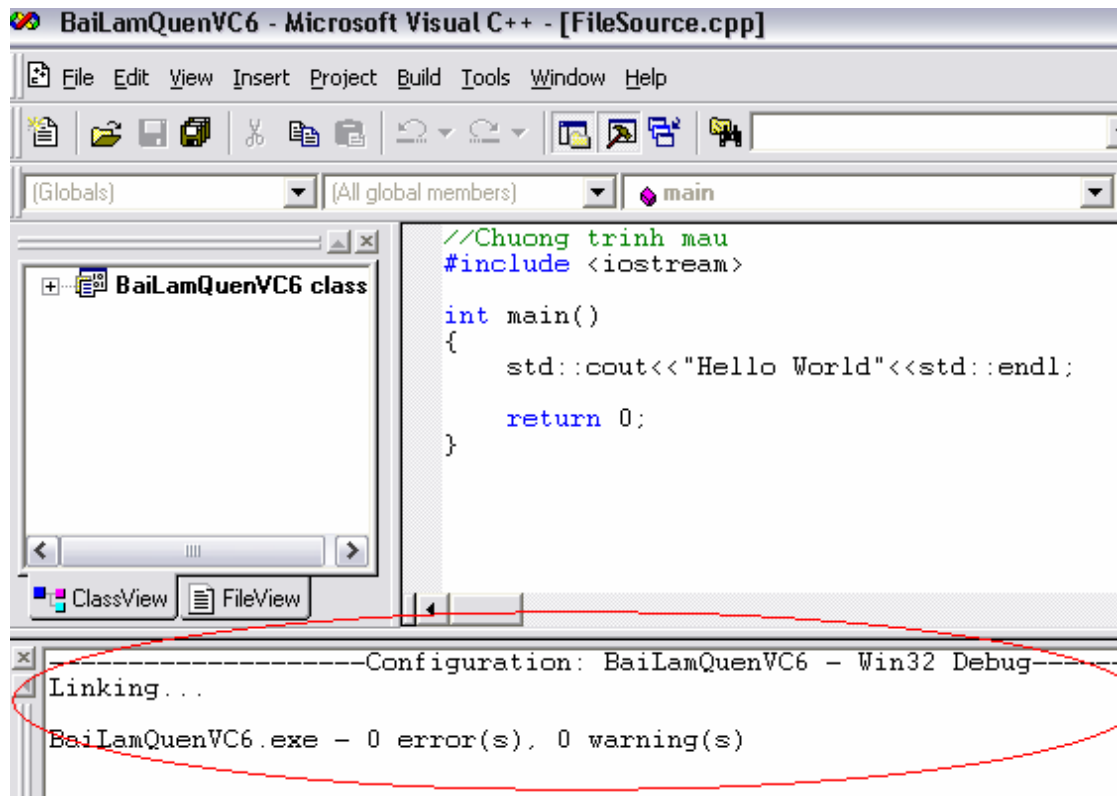
7. Gõ nội dung chương trình “Hello World”



8. Biên dịch : vào **Build -> Rebuild All**



Quan sát cuối màn hình, nếu biên dịch thành công ta sẽ thấy dòng chữ “0 error(s), 0 warning(s)”. Nếu không thành công thì cần xem lại đoạn mã mình đã gõ vào đúng chưa, **bấm F4 để tìm lỗi**



Lưu ý: Một thói quen tốt là nên ghi lại và hiểu các dòng báo lỗi để lần sau có thể tự sửa lỗi.

- Chạy chương trình, bấm **Ctrl + F5** (hoặc **F5**, nếu bấm **F5** Màn hình chạy chương trình có thể hiển thị rồi tắt rất nhanh, do đó cần quan sát kỹ.)

Lưu ý quan trọng:

- Trong một chương trình:

- Hàm main là nơi chương trình bắt đầu.
- **Một chương trình chỉ có một hàm main duy nhất**

- Khi viết xong một chương trình, để viết tiếp chương trình khác, ta phải đóng chương trình trước đi bằng cách vào menu **File - > Close Workspace**

- Để mở một chương trình có sẵn, trong môi trường VC6++, vào **File ->**

OpenWorkspace rồi chọn file ***.dsw**

- Trong thư mục chứa chương trình, ta luôn có 1 thư mục **Debug**, trong thư mục này có một file ***.exe** là file chạy trực tiếp, còn những file khác là những file biên dịch không quan trọng, **ta nên xóa chúng khi copy từ máy trường về nhà hoặc nộp bài tập cho giáo viên.**

Chương 2: Biến và Kiểu dữ liệu

2.1. Kiểu dữ liệu:

a. Các loại dữ liệu thông dụng:

Tên kiểu dữ liệu	Phạm vi	Ý nghĩa	Ví dụ
int	-32768 -> 32767	Kiểu số nguyên	-2
unsigned int	0 -> 65535	Số nguyên không âm	5
long	-2147483648->2147483647	Số nguyên dài	3445
float		Số thực	4.5
double		Số thực	7.8
char	-128 -> 127	Ký tự	'A', '+'
unsigned char	0 -> 255	Ký tự	'B', '-'
bool	true, false	Kiểu logic	

Chú ý:

- Kiểu ký tự cũng có thể là một dạng của kiểu số nguyên
- Kiểu bool Là kiểu thể hiện giá trị đúng sai
 - + Có hai giá trị: true hoặc false
 - + Các phép toán:
 - && à phép AND (và)
 - || à phép OR (hoặc)
 - ! à phép NOT (phủ định)

b. Kiểu dữ liệu do người dùng định nghĩa

- Từ khoá: typedef
- Cú pháp: typedef tên_kiểu_cũ tên_kiểu_mới
- Ý nghĩa: sau khi khai báo, ta có thể sử dụng tên_kiểu_mới như một kiểu dữ liệu bình thường khác.

Ví dụ:

- Định nghĩa lại tên kiểu dữ liệu số nguyên

```
typedef int so_nguyen;
```

Khi đó khi **so_nguyen** là một kiểu dữ liệu như kiểu int

2.2. Biến: Định nghĩa - Các thành phần của biến - Khai báo biến

- Mọi biến cần phải khai báo trước khi sử dụng.
- Việc khai báo biến được thực hiện theo cú pháp sau:

Kiểu_dữ_liệu tên_biến;

Kiểu tên_biến = giá_trị_ban_đầu_của_biến;

Kiểu tên_biến1, tên_biến2,...;

Ví dụ:

int i, j; //i và j là 2 kiểu số nguyên

char a; //a là kiểu ký tự

float f; // f là kiểu số thực

- Lưu ý:

+ Một tên hợp lệ là một dãy của các

- Chữ (hoa và thường)
- Số (Một tên không được bắt đầu bởi số)
- Dấu nối dưới (Không nên bắt đầu bằng dấu nối dưới)

+ Các tên có phân biệt chữ hoa và chữ thường

Ví dụ: Bien1 khác với BIEN1

+Tên biến **không được trùng** với tên của **các lệnh**, các **từ khoá** trong C++ (ví dụ không được đặt tên biến là int, return, cout,...)

2.3. Biểu thức và toán tử: Biểu thức - Toán tử và độ ưu tiên của toán tử

Tên toán tử	Ý nghĩa	Ví dụ
+	cộng	a+b
-	trừ	a-b
*	nhân	a*b

/	chia	a/b
%	chia lấy phần dư	a%b
>		a>b
<		a<b
<=	nhỏ hơn hoặc bằng	a<=b
>=	lớn hơn hoặc bằng	a>=b
!=	so sánh khác	a!=b
==	so sánh bằng	a==b
=	Lệnh gán	a = b
+=	Viết tắt của lệnh a=a+b	a+=b
-=	Viết tắt của lệnh a=a-b	a-=b
*=	Viết tắt của lệnh a=a*b	a*=b
/=	Viết tắt của lệnh a=a/b	a/=b
%=	Viết tắt của lệnh a=a%b	a%=b
++	Viết tắt của lệnh a=a+1	a++ ++a
--	Viết tắt của lệnh a=a-1	a-- --a

2.4. Câu lệnh gán: Cú pháp - Thi hành câu lệnh gán

Cú pháp: tên_biến = biểu thức (hoặc biến);

Ý nghĩa:

Giá trị của biến bên trái dấu “=” sẽ bằng giá trị của biểu thức *sau khi tính toán xong* (hoặc giá trị của biến) bên phải dấu “=”

Ví dụ:

```
int i, j;    //khai báo biến i và j là kiểu số nguyên
i = 3;    // i sẽ có giá trị là 3;
j = i + 7; // j sẽ có giá trị là 10 ( 3 + 7)
```

Lưu ý:

So sánh phép toán **a++** với **++a** (tương tự cho **a--** và **--a**)

Ví dụ:

```
/* khai báo biến a và b là kiểu số nguyên và cùng có giá trị ban đầu là 5
*/
```

```
int a = 5;
```

```
int b = 5;
```

```
int i, j; //khai báo biến i và j
```

```
i = ++a;    Ý nghĩa:
```

- Thực hiện phép toán **a=a+1 trước**, tức giá trị a tăng lên 1 đơn vị, hay a sẽ có giá trị là 6

- Sau đó **i sẽ được gán giá trị 6 của a**

- Giá trị của a sau lệnh này sẽ có giá trị là 6

```
j = b++;    Ý nghĩa:
```

- Thực hiện phép **gán j=b trước**, do giá trị của b hiện giờ là 5, **nên j sẽ có giá trị là 5**

- Sau đó phép toán **b=b+1** sẽ được thực hiện, tức giá trị của b sẽ được tăng lên 1 đơn vị, khi đó b sẽ có giá trị là 6

2.5. Chuyển đổi kiểu (ép kiểu)

Ép kiểu là một thủ thuật khá hay dùng để biến đổi kiểu của một biến, hay một biểu thức sang một kiểu dữ liệu phù hợp với nhu cầu tính toán của chương trình.

Cú pháp: (tên kiểu mới) tên_biến(hoặc biểu thức);

Lưu ý: giá trị của phép ép kiểu sẽ chuyển giá trị sang kiểu dữ liệu mới, nhưng **kiểu dữ liệu của những biến tham gia vào phép ép kiểu vẫn không bị thay đổi kiểu dữ liệu**

Ví dụ 1:

1.34 : là một số thực, thuộc kiểu số thực

(int) 1.34 : giá trị được ép kiểu sang kiểu số nguyên, khi đó kết quả sẽ là 1

Ví dụ 2:

float i = 2.34; /* i được khai báo là kiểu số thực, với giá trị ban đầu là 2.34 */

int j = (int) i ; // ép kiểu khi đó j sẽ có giá trị là 2, tuy nhiên giá trị của i vẫn là 2.34

Những lưu ý quan trọng:

- ☐ Tất cả lệnh đều kết thúc bằng dấu chấm phẩy ;
- ☐ Cách chú thích:
 - Chú thích 1 dòng: dùng ký hiệu //
 - Chú thích nhiều dòng: dùng ký hiệu mở /* và ký hiệu đóng */
 - ☐ Ví dụ:
/* đây là dòng chú thích dòng 1
Đây là dòng chú thích dòng 2 */
- ☐ Một số ký tự đặc biệt:
 - ‘\n’ là mã xuống dòng
 - ‘\t’ là mã dấu Tab
 - ‘\” ‘ là mã dấu nháy kép “
 - ‘\\’ là mã dấu \
- ☐ Chuỗi: bắt đầu và kết thúc bằng dấu “ và ”
 - Ví dụ:
 - “Đây là một chuỗi”
 - “Đây là một chuỗi có xuống dòng \n”

2.6 Lệnh xuất/nhập chuẩn:

Một số lệnh cơ bản cần nắm

- `std::cout` (xuất dữ liệu ra màn hình)

Ví dụ:

```
std::cout<< a ; // xuất giá trị của a ra màn hình
```

```
std::cout<<a<<b; // xuất giá trị của a và b ra màn hình
```

- `std::cin` (nhập giá trị vào vào biến)

Ví dụ:

```
std::cin>>a; /*May tính sẽ yêu cầu người dùng nhập một giá trị từ bàn phím, sau khi nhập từ phím 1 giá trị và bấm enter, thì a sẽ mang giá trị mà người đó vừa nhập*/
```

- Để việc lập trình thuận tiện, thay vì phải đánh chữ `std` trước một số lệnh, ta khai báo tên vùng `std` ở phía trước hàm `main` (bằng cú pháp **using namespace std**), khi đó ta không cần phải đánh lại `std` trước các lệnh đó

Ví dụ:

```
//Chương trình nhập xuất giá trị của biến
#include <iostream>    //Khai báo thư viện hàm cout, cin
#include <conio.h>     //khai báo thư viện hàm getch()

using namespace std; //khai báo tên vùng lệnh std

int main()
{
    int i; //Khai báo biến i là kiểu số nguyên
    int j = 10; /*khai báo biến j là kiểu số nguyên với giá trị ban đầu là 10 */

    cin>>i; //Nhập giá trị và biến i
    cout<<i<<j; //Xuất giá trị của biến i và j ra màn hình
```

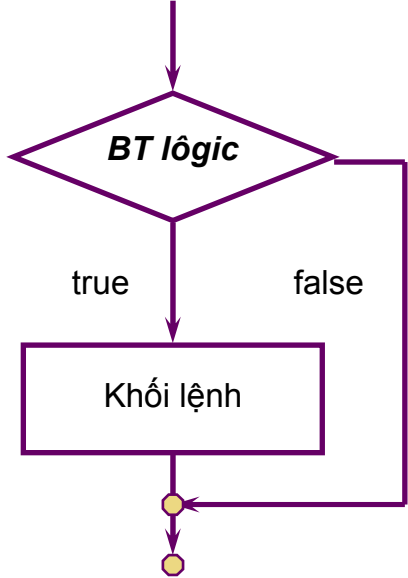
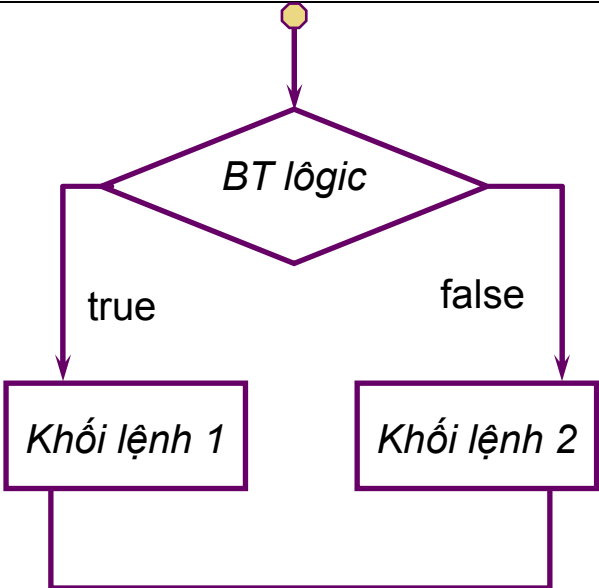


```
    getch(); //Dừng màn hình  
    return 0;  
}
```

Chương 3: Cấu trúc điều kiện

3.1. Điều kiện if:

a. Cú pháp:

<pre> 1. if (BT logic) { //Khối lệnh } </pre>	 <pre> graph TD Entry(()) --> Decision{BT logic} Decision -- true --> Block[Khối lệnh] Decision -- false --> Merge(()) Block --> Merge Merge --> Exit(()) </pre>
<pre> 2. if (BT logic) { //Khối lệnh 1 } else { //Khối lệnh 2 } </pre>	 <pre> graph TD Entry(()) --> Decision{BT logic} Decision -- true --> Block1[Khối lệnh 1] Decision -- false --> Block2[Khối lệnh 2] Block1 --> Exit(()) Block2 --> Exit </pre>

<pre> 3. if (BT logic) { //Khối lệnh } else if (BT logic) { //Khối lệnh } </pre>	
---	--

Chú ý:+ C++ qui ước:

true != 0

false == 0

+ Kiểu Logic:

- ☐ Là kiểu thể hiện giá trị đúng sai
- ☐ Có hai giá trị: true hoặc false
- ☐ Các phép toán:
 - && à phép AND (và)
 - || à phép OR (hoặc)
 - ! à phép NOT (phủ định)

b. Bài tập:

- i. Bài tập 1: Nhập vào một số x, xác định tính chẵn lẻ của x
- ii. Bài tập 1: Nhập vào một số x, xác định số x là số dương, hay số âm hoặc bằng 0 ?
- iii. Bài tập 2: Nhập vào 2 số nguyên a,b. Tìm số lớn nhất.
- iv. Bài tập 2: Nhập vào 3 số nguyên a,b,c. Tìm số lớn nhất và số nhỏ nhất trong 3 số đó.

v. Giải phương trình $Ax + B = 0$. Với A, B là hai hệ số được nhập vào từ bàn phím

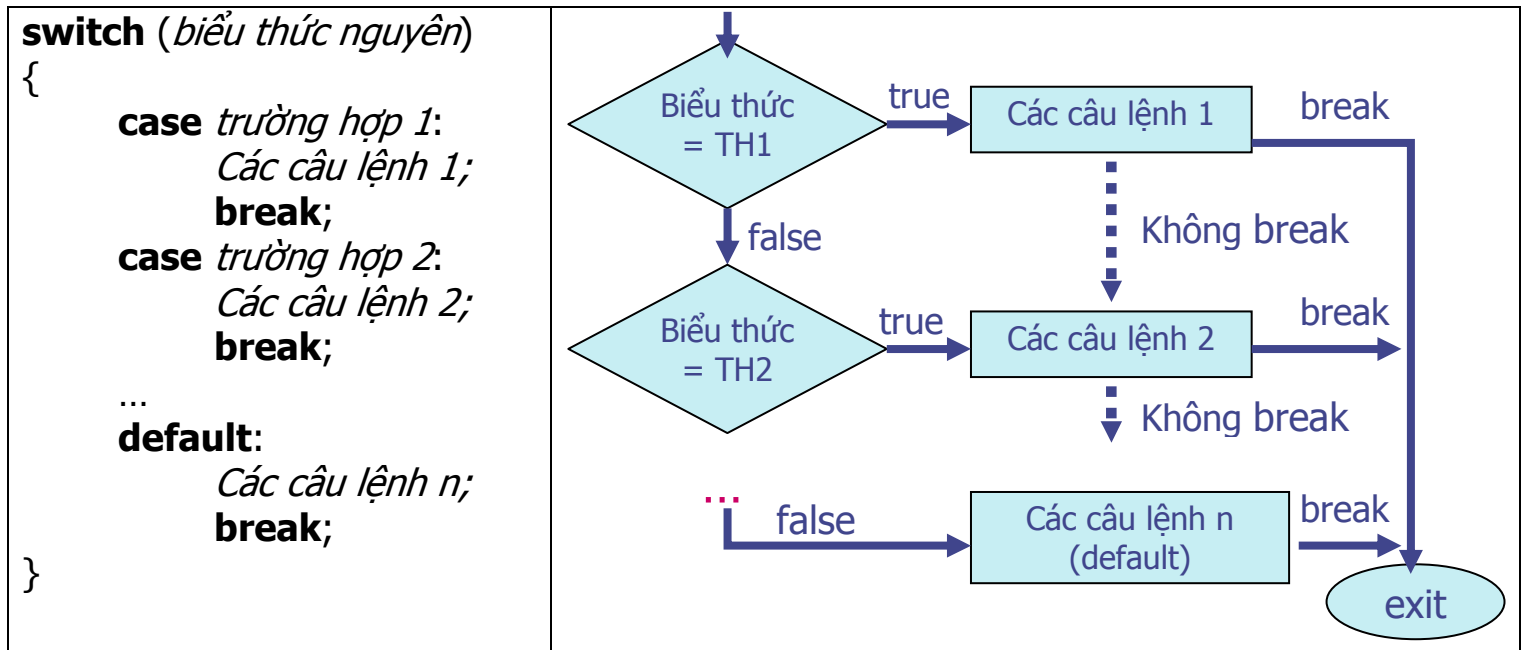
vi*. Nhập vào một số n, xác định xem có phải là số chính phương hay không?

3.2. Cấu trúc lựa chọn switch ... case ...

a. Trường hợp sử dụng

- Khi có quá nhiều lựa chọn dẫn đến nếu dùng cấu trúc if ... else ... if sẽ dài dòng và khó hiểu.
- Khi sự so sánh chỉ là so sánh bằng, không phải so sánh thứ tự.

b. Cú pháp:



Lưu ý:

- Biểu thức nguyên: Biến kiểu nguyên, biểu thức có giá trị nguyên, hoặc kiểu ký tự, kiểu logic.
- Trường hợp ... : là số nguyên, hằng ký tự, hoặc biểu thức hằng, giá trị logic.
- default: (trường hợp ngoại lệ), sẽ được thực thi nếu các trường hợp ở trên không xảy ra (thành phần không bắt buộc).

c. Bài tập:

Bài tập 1: Nhập vào một số x, xác định tính chẵn lẻ của x

Bài tập 2: Viết chương trình cho trò chơi thi trắc nghiệm

- Xuất ra màn hình một câu hỏi và 4 đáp án a, b, c, d.
- Nếu người chơi chọn đáp án sai thì xuất câu thông báo “sai rồi”
- Còn nếu người chơi chọn đáp án đúng thì báo “đúng rồi”

Bài tập 3: Xuất kết quả của phép toán

A phép_toán B

Với A, B là 2 số thực nhập từ bàn phím

Và phép toán là một trong những phép toán sau: +, -, *, /

Bạn hãy viết chương trình xuất ra kết quả của phép toán trên phù hợp với phép toán mà người dùng nhập vào.

Ví dụ:

Người dùng nhập A = 2

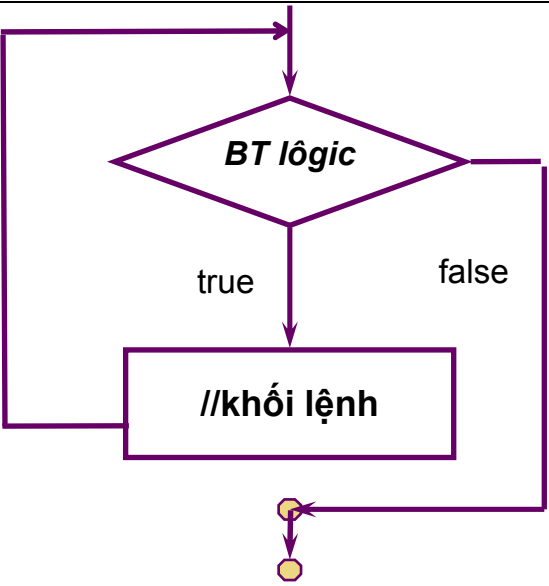
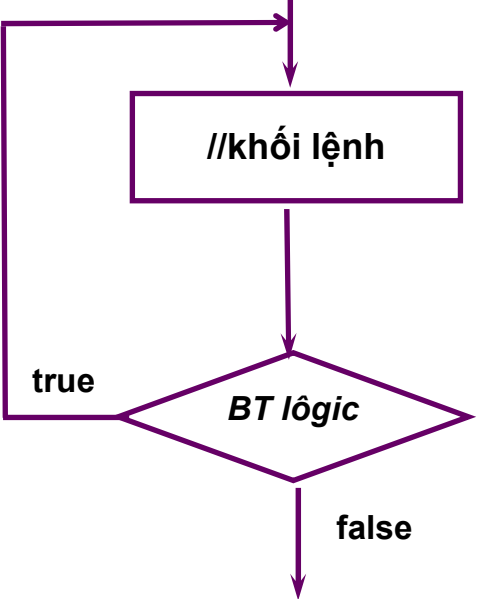
B = 3.5

Phép toán: *

Khi đó kết quả của phép toán là: $2 * 3.5 = 7$

Chương 4: Cấu trúc lặp

4.1. Các cấu trúc lặp:

<p>1. while (BT logic)</p> <pre> { //Khởi lệnh } </pre>	 <pre> graph TD Entry(()) --> Decision{BT logic} Decision -- true --> Process[///khởi lệnh/] Process --> Entry Decision -- false --> Exit(()) </pre>
<p>2. do</p> <pre> { //Khởi lệnh } while (BT logic) ; </pre> <p><u>Lưu ý:</u></p> <ul style="list-style-type: none"> - có dấu ; ở cuối lệnh while - Khởi lệnh luôn được thực hiện ít nhất 1 lần 	 <pre> graph TD Entry(()) --> Process[///khởi lệnh/] Process --> Decision{BT logic} Decision -- true --> Entry Decision -- false --> Exit(()) </pre>

3. for (khởi tạo ; điều kiện; lệnh thực hiện sau)

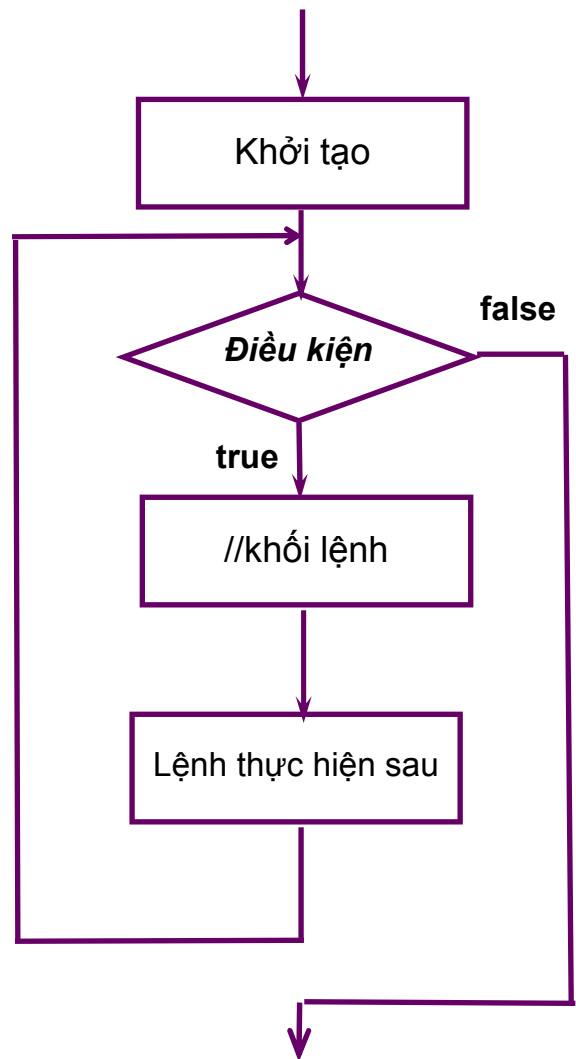
```
{
    ..... //Khởi lệnh
}
```

Ví dụ:

```
for(int i=0; i< 5; i++)
    cout<<"Hello world"<<endl;
int j=10;
for (; j >=0; --j)
    cout<<"Hello world"<<endl;
```

+ Lưu ý: các thành phần trong lệnh for có thể được khuyết

+ Câu hỏi gợi mở: khi kết thúc vòng for ở trên, thì các biến i và j sẽ có giá trị là bao nhiêu?



Nhận xét:

- Vòng lặp while, do ... while, for đều sẽ **không thực thi** những khối lệnh bên trong nữa khi **Biểu thức logic** hay **điều kiện vòng lặp** có giá trị là **false** (hay có giá trị bằng 0)

Lưu ý:

Trong C++ có 2 từ khoá là **break** và **continue** được dùng phổ biến trong vòng lặp (**đặc biệt là từ khoá break**)

- **break**: Trong vòng lặp khi gặp từ khoá này thì chương trình sẽ nhảy ra khỏi vòng lặp hiện thời, mà không cần phải thực thi những lệnh tiếp theo của vòng lặp

- **continue** (thường thấy trong vòng lặp for): Trong vòng lặp khi gặp từ khoá này thì chương trình bắt đầu một vòng mới của vòng lặp.

4.2. Mối liên quan giữa for và while:

a. Chuyển lệnh for thành while:

Khởi_tạo_for;

while (điều_kiện_của_for)

{

.... //Khởi lệnh trong thân lệnh for

Lệnh_thực_hiện_sau_của_for;

}

b. Chuyển lệnh while thành for:

for (; điều_kiện_của_while;)

{

..... //Khởi lệnh trong thân lệnh while

}

2. Bài tập:

Thực hiện bằng cấu trúc while, do... while, for cho cả 3 câu i, ii, iii

i. Tính tổng $1 + 2 + 3 + 4 + \dots + n$

ii. Tính tổng $1 + 1/2 + 1/3 + 1/4 + \dots + 1/n$

iii. Tính tổng $1^2 + 2^2 + 3^2 + 4^2 + \dots + n^2$

iv. Nhập vào một số n, xác định xem có phải là số nguyên tố không?

v. Xuất ra các số nguyên tố từ 1 đến n, với n là số nguyên dương nhập từ bàn phím

Chương 5: Cấu trúc

5.1. Nhắc lại cách khai báo một kiểu dữ liệu mới (typedef):

a. Cú pháp:

```
typedef Kiểu_dữ_liệu_cũ Kiểu_dữ_liệu_mới;
```

b. Ý nghĩa:

Khi khai báo một kiểu dữ liệu mới bằng **typedef**, tức là

- **Kiểu_dữ_liệu_mới** là 1 kiểu dữ liệu như kiểu **Kiểu_dữ_liệu_cũ**
- Ta có thể sử dụng kiểu dữ liệu mới như những kiểu dữ liệu bình thường khác (sử dụng giống như kiểu dữ liệu int, float, double, char, ...)

c. Ví dụ:

```
typedef int so_nguyen;           //Định nghĩa so_nguyen la kiểu dữ liệu mới
so_nguyen i;                    //Sử dụng kiểu so_nguyen như kiểu dữ liệu bình
int j;                          //thường khác
for (so_nguyen t = 0; t<5; t++)
    cout<<"Hello world"<<endl;
```

5.2. Cách khai báo một kiểu dữ liệu mới bằng cú trúc:

a. Mục đích:

Trên thực tế, nhiều đối tượng có chứa nhiều thành phần riêng lẻ. Ví dụ, PHÂN SỐ có 2 thành phần là TỬ và MÃU, hoặc ĐIỂM thì có HOÀN HẠCH X và TUNG ĐỘ Y. Xu hướng lập trình hiện nay là gộp các thành phần riêng lẻ thành một kiểu dữ liệu mới, mà kiểu dữ liệu này sẽ chứa thông tin của các thành phần đó. Ví dụ, ta sẽ xây dựng kiểu dữ liệu PHÂN SỐ, với 2 trường (field) là TỬ và MÃU, trong đó TỬ và MÃU là 2 trường kiểu số nguyên

b. Cú pháp:

- i.
- Cách 1:**
- dùng typedef để định nghĩa kiểu mới

```
typedef struct{
    kiểu_dữ_liệu    trường_1;
    kiểu_dữ_liệu    trường_2;
    .....
}tên_cấu_trúc;
```

- ii.
- Cách 2:**
- không dùng typedef để định nghĩa kiểu mới

```
struct tên_cấu_trúc{
    kiểu_dữ_liệu    trường_1;
    kiểu_dữ_liệu    trường_2;
    .....
};
```

khai báo tên cấu trúc đồng thời có thể khai báo cùng lúc các biến kiểu cấu trúc.

```
struct tên_cấu_trúc{
    kiểu_dữ_liệu    trường_1;
    kiểu_dữ_liệu    trường_2;
    .....
}biến_1, biến_2,...;
```

Lưu ý:

- Có dấu chấm phẩy (;) ở cuối khai báo cấu trúc
- Ta sử dụng kiểu cấu trúc như mọi kiểu bình thường khác

c. Ví dụ:

Xây dựng kiểu cấu trúc PHÂN SỐ, với 2 trường là TỬ và MÃU kiểu số nguyên

Cách 1:

```
typedef struct
{
    int tu;
    int mau;
} PHANSO;
```

Cách 2:

```
struct PHANSO
{
    int tu;
    int mau;
};
```

khi đó ta có thể sử dụng kiểu cấu trúc PHANSO như là một kiểu bình thường

```
PHANSO ps;
```

c. Cách truy cập vào các trường của cấu trúc:

Ta dùng cú pháp sau để truy cập vào trường của cấu trúc

Tên_biến_kiểu_cấu_trúc.trường

(lưu ý: có dấu chấm (.) ngăn cách giữa biến_kiểu_cấu_trúc và trường được gọi)

Ví dụ, với cấu trúc PHANSO được khai báo như ở trên, ta có thể nhập và xuất các trường của biến ps có kiểu cấu trúc PHANSO như sau:

```
PHANSO ps;
cin>>ps.tu>>ps.mau;
cout<<ps.tu<<"/"<<ps.mau;
```

5.2. Bài tập cấu trúc

1) Viết chương trình nhập vào 2 phân số ps1, ps2

- Tính tổng, hiệu, tích, nhân 2 phân số đó, kết quả trả về là phân số

(Lưu ý: phải kiểm tra tính đúng đắn của dữ liệu nhập vào, ví dụ mẫu của phân số phải luôn khác không)

2) Viết chương trình nhập vào 3 điểm A, B, C trong hệ trục tọa độ Oxy, hãy:

- Kiểm tra xem 3 điểm đó có thẳng hàng hay không
- Tính khoảng cách từ A đến B, A đến C, và B đến C
- Nếu A,B,C không thẳng hàng hãy tính diện tích, chu vi của tam giác ABC

3) Viết chương trình quản lý điểm của sinh viên, với

- Số lượng sinh viên là n, với n nhập từ bàn phím
- Thông tin về sinh viên gồm:
 - ☐ Họ và tên
 - ☐ Ngày tháng năm sinh
 - ☐ MSSV
 - ☐ Điểm Toán, Lý, Hoá
 - ☐ Điểm trung bình

Chương 6: Mảng

6.1. Định nghĩa:

Mảng là một tập hợp nhiều phần tử có cùng một kiểu giá trị và có cùng chung một tên

6.2. Khai báo:

Kiểu_dữ_liệu Tên_mảng[spt][spt]....;

Cụ thể:

a. Mảng một chiều:

Kiểu_dữ_liệu Tên_mảng[spt];

b. Mảng hai chiều:

Kiểu_dữ_liệu Tên_mảng[spt dòng][spt cột];

Lưu ý quan trọng:

- Chỉ số của mảng luôn luôn được đánh từ **chỉ số 0**
- Các phần tử của mảng ta có thể sử dụng như một biến bình thường

Ví dụ: Gán một phần tử của mảng cho 10

$a[2] = 10;$

Ví

du:

-

Mã

ng

1

chi

ều kiểu số nguyên có 10 phần tử: `int a[10];`

`a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]`

- Mảng 1 chiều kiểu ký tự có 5 phần tử:

`char s[5];`

//Lưu ý: Đây cũng có thể xem là một chuỗi có 5 ký tự

- Mảng 1 chiều kiểu chuỗi có 5 phần tử: `string chuoi[5];`

- Mảng 2 chiều kiểu số nguyên có 3 dòng, 4 cột: `int a[3][4];`

a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2][0]	a[2][1]	a[2][2]	a[2][3]

6.2. Bài tập:

i. Thực hiện các phép toán của mảng (1 chiều và 2 chiều):

- Khởi tạo một mảng
- Xuất các giá trị của mảng
- Thêm một phần tử vào mảng (1 chiều)
- Xoá một phần tử trong mảng (1 chiều)

ii. Nhập vào thông tin của một sinh:

- Họ tên
- Năm sinh
- Điểm các môn học (lưu vào mảng)

Xuất ra điểm trung bình của sinh viên đó

iii. Nhập vào một dãy số nguyên, tìm phần tử lớn nhất, nhỏ nhất của dãy (1 chiều, 2 chiều)

iv. Nhập vào một dãy số nguyên,

Xuất ra 2 dãy số:

- Dãy những số âm của mảng
- Dãy những số dương của mảng

v. Nhập vào một dãy số nguyên,

Xuất ra 2 dãy số:

- Dãy những số nguyên tố
- Dãy những số chính phương.

Chuyên đề: Mảng – ký tự - chuỗi

1. Ký tự

- Ví dụ: 'c' , 'b' , 'd'
- Tên kiểu dữ liệu: char
- Ví dụ:

```
char c;  
cin >> c;
```

2. Mảng

- Là một tập hợp các biến có cùng kiểu dữ liệu
- Từ phần tử của mảng ta có thể xem là một biến, ví dụ a[i], a[1], a[2][3],.... ta có thể thực hiện mọi lệnh, mọi phép toán trên biến áp dụng vào từ phần tử của mảng
- **Lưu ý: Mảng luôn luôn được đánh số từ 0**
- Cách khai báo:
 - Mảng 1 chiều:
 Tên_dữ_liệu Tên_mảng[spt];
 - Mảng 2 chiều:
 Tên_dữ_liệu Tên_mảng[số_dòng][số_cột]
- **Ví dụ:**

```
int n;  
int a[100]; //Khai báo mảng 1 chiều kiểu số nguyên có 10 phần tử  
  
cout<<"nhập vào số phần tử của mảng"<<endl;  
cin>>n;  
  
//Nhập vào từng phần tử của mảng  
cout<<"Mời bạn nhập vào từng phần tử của mảng"<<endl;  
for (int i = 0 ; i<n; ++i)  
    cin>>a[i];  
  
//Xuất các phần tử của mảng ra màn hình  
for (i = 0 ; i<n; ++i)  
    cout<<a[i];
```

3. Chuỗi

- Có thể nói **chuỗi là mảng 1 chiều kiểu ký tự** (từ phần tử của mảng là ký tự) – tức những thao tác về mảng, ta có thể áp dụng vào cho chuỗi
- **Chú ý:** Phần tử (ký tự) đầu tiên của chuỗi s là s[0], phần tử (ký tự) cuối cùng của chuỗi s là s[L – 1] , với L là chiều dài chuỗi

a. Chuỗi - mảng ký tự

- Khai báo: `char tên_chuỗi[chiều_dài_chuỗi];`
- Với `char s[10] = "Hello"` thì `s[0] = 'H'`, `s[1] = 'e'`, `s[2] = 'l'`, `s[3] = 'l'`, `s[4] = 'o'`, **đặc biệt s sẽ có thêm ký tự kết thúc chuỗi là `s[5] = '\0'`**. Khi đó chiều dài chuỗi là 5 (không tính ký tự kết thúc chuỗi)
- Ví dụ:
`char s[10]; //khai báo một chuỗi có tối đa 10 ký tự`
`//hay một mảng gồm 10 phần tử kiểu ký tự`
- Nhập dữ liệu vào chuỗi
 - i. **Cách 1:** Nhập từ phần tử giống như mảng
`//Với L là chiều dài của chuỗi`
`for (int i = 0; i < L; ++i)`
`cin>>s[i];`
 - ii. **Cách 2:** Nhập chuỗi **không có khoảng trắng**
`cin>>s;`
 - iii. **Cách 3:** Nhập chuỗi **có khoảng trắng**
`gets(s);`
- Cách xuất chuỗi
 - i. **Cách 1:** Xuất từ phần tử giống như mảng
`//Với L là chiều dài của chuỗi`
`for (int i = 0; i < L; ++i)`
`cout<<s[i];`
 - ii. **Cách 2:**
`cout<<s;`
- Các lệnh có liên quan đến chuỗi dạng kiểu ký tự

Cú pháp lệnh	Ý nghĩa	Thư viện phải khai báo
<code>int toupper(int c)</code>	Chuyển ký tự c thành ký tự hoa	<code>#include <ctype.h></code>
<code>int tolower(int c)</code>	Chuyển ký tự c thành ký tự thường	<code>#include <ctype.h></code>
<code>char* strupr(char* s)</code>	Chuyển s sang chuỗi hoa	<code>#include <string.h></code>
<code>char* strlwr(char* s)</code>	Chuyển s sang chuỗi thường	<code>#include <string.h></code>
<code>int strlen(const char* s)</code>	Trả về độ dài chuỗi	<code>#include <string.h></code>
<code>char* strcat(char* dest, const char* src)</code>	Cộng 2 chuỗi, kết quả trả về cho chuỗi dest	<code>#include <string.h></code>
<code>char* strcpy(char* dest, const char* src)</code>	Copy nội dung của chuỗi src về cho chuỗi dest	<code>#include <string.h></code>
<code>int strcmp(const char* s1, const char* s2)</code>	Giá trị trả về giá trị: - <0 : nếu s1 "nhỏ hơn" s2 - 0 : nếu s1 bằng s2 - >0 : nếu s1 "lớn hơn" s2	<code>#include <string.h></code>

int atoi(const char* s)	Đổi chuỗi số sang số nguyên (nếu không đổi được kết quả sẽ trả về giá trị 0) Ví dụ: <pre>char s[10] = "123"; int i = atoi(s); cout<<i; //Khi đó i sẽ bằng 123</pre>	#include <stdlib.h>
double atof(const char* s)	Đổi chuỗi số sang số thực (nếu không đổi được kết quả sẽ trả về giá trị 0.0) Ví dụ: <pre>char s[10] = "12.3"; float f = atof(s); cout<<f; //Khi đó f sẽ bằng 12.3</pre>	#include <stdlib.h>
char* itoa(int value, char *str, int hệ_cơ_số);	Đổi số value thành chuỗi, tương ứng với cơ số được khai báo Ví dụ: <pre>int i = "123"; char s[100]; itoa(i, s, 10); //đổi sang cơ số 10 cout<<s; //Khi đó s sẽ bằng "123"</pre>	#include <stdlib.h>
<u>*Định dạng chuỗi theo ngôn ngữ C</u> sprintf(char* s,chuỗi_định_dạng_của_C, tập_giá_trị); chuỗi định dạng là tập các ký tự cờ: "%c" – ký tự "%i" – số nguyên "%f" – số thực	Ví dụ: <pre>char s[100]; int i = 10; float f = 132.124; sprintf(s, "%i", i); sprintf(s, "%f", f); sprintf(s, "%.2f", f); //lấy 2 chữ số //sau dấu . //của số thực</pre>	#include <stdio.h>

b. **Chuỗi - kiểu string** (sẽ được học kỹ ở chương 14)

- Thư viện cần khai báo: **#include <string>**
- Khai báo: string tên_chuỗi[chiều_dài_chuỗi];
- Ví dụ: string s;
- Nhập dữ liệu vào chuỗi
 - i. **Cách 1**: Nhập chuỗi **không có khoảng trắng**
cin>>s;
 - ii. **Cách 3**: Nhập chuỗi **có khoảng trắng**
getline(cin,s);
- Cách xuất chuỗi
cout<<s;

Lưu ý:

- **Cách khai báo hằng:** (hằng là những biến là giá trị sẽ không bao giờ bị thay đổi trong chương trình)

+ **Cách 1: (dùng macro #define)**

```
#define biến_hằng giá_trị //Không có dấu ; cuối lệnh
```

- Ý nghĩa: nếu dùng từ khoá marco **#define**, trong khi biên dịch, chương trình sẽ thay tất cả **biến_hằng** bằng **giá_trị**

- **Ví dụ:**

```
#define bien_hang 123
```

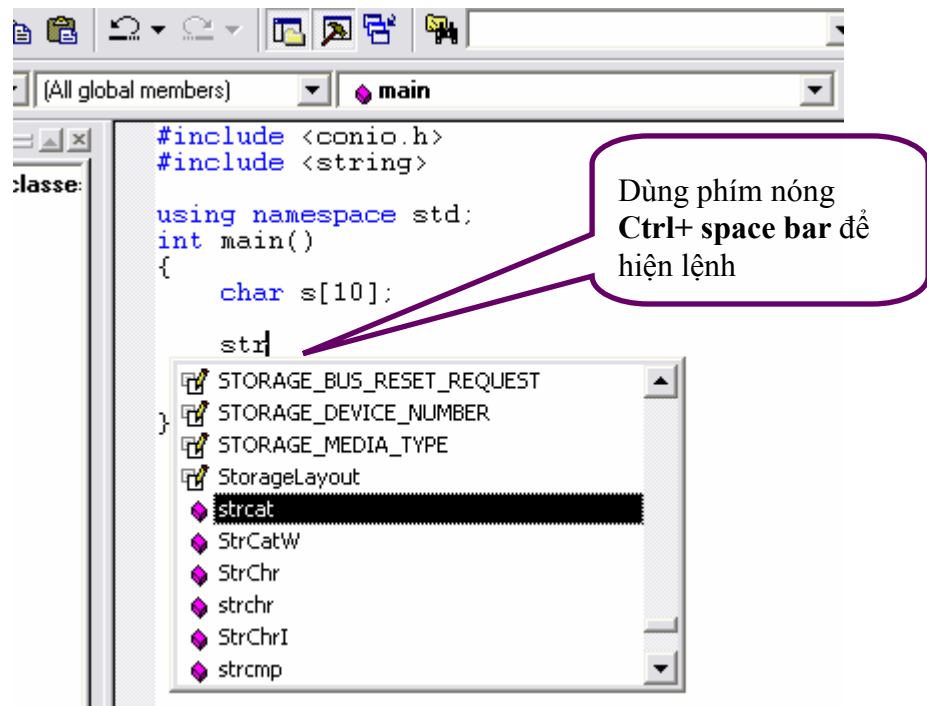
+ **Cách 2:**

```
const kiểu_dữ_liệu tên_hằng = giá_trị; //có dấu ; cuối lệnh
```

Ví dụ:

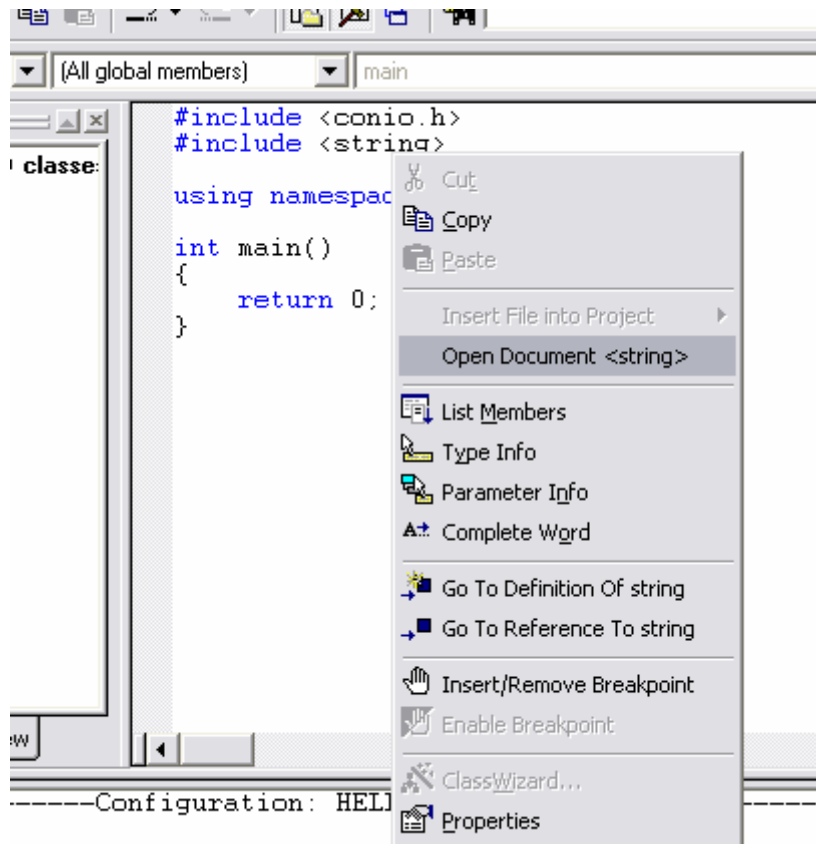
```
const int bien_hang = 123;
```

- Khi lập trình trên môi trường Visual C++, trong khi đánh lệnh ta có thể dùng phím nóng **Ctrl + Space bar** để hiển thị các lệnh của trong C++



Do hàm getline trong Visual C++ bị lỗi, ta sẽ sửa lỗi bằng cách sau:

+ Bước 1: Bấm chuột phải lên chỗ khai báo thư viện **<string>**, bấm chuột phải chọn **“Open Document <string>”**



+ Bước 2: Bấm phím nóng Ctrl + F

+ Bước 3: gõ vào hộp thoại tìm kiếm dòng chữ:

else if (_Tr::eq((_E)_C, _D))

+ Bước 4: Thay đổi một số lệnh

Ban đầu	Sau khi sửa
<pre>else if (_Tr::eq((_E)_C, _D)) { _Chg = true; _I.rdbuf()->snextc(); break; } </pre>	<pre>else if (_Tr::eq((_E)_C, _D)) { _Chg = true; //_I.rdbuf()->snextc(); //Phần thêm vào _I.rdbuf()->sbumpc(); break; } </pre>

```

const _Myis::sentry _Ok(_I, true);
if (_Ok)
{
    _TRY_IO_BEGIN
    _Tr::int_type _C = _I.rdbuf()->sgetc();
    for (; ; _C = _I.rdbuf()->snextc())
        if (_Tr::eq_int_type(_Tr::eof(), _C))
        {
            _St |= ios_base::eofbit;
            break; }
        else if (_Tr::eq((_E)_C, _D))
        {
            _Chg = true;
            //_I.rdbuf()->snextc();

            //Phần thêm vào
            _I.rdbuf()->sbumpc();
            break; }
        else if (_X.max_size() <= _X.size())
        {
            _St |= ios_base::failbit;
            break; }
        else
            _X += _Tr::to_char_type(_C), _Chg = true;
    _CATCH_IO_(_I); }
if (!_Chg)
    _St |= ios_base::failbit;

```

+ Bước 5: Bấm Ctrl + S (lưu nội dung đã thay đổi), sau đó đóng cửa sổ lại.

Bài tập bổ sung:

1. Nhập vào một mảng 2 chiều $n \times n$ (ma trận vuông)
 - ☐ Tính tổng các số hạng trên đường chéo chính
 - ☐ Tính tổng các số hạng trên đường chéo phụ
 - ☐ Duyệt phân nửa trên của đường chéo chính
2. Sắp xếp tăng dần/giảm dần các giá trị của một mảng 1 chiều
3. Bài tập về phân số
 - ☐ Nhập vào 2 phân số (phân số gồm có tử và mẫu)
 - ☐ Tính tổng, hiệu, tích, thương của 2 phân số
 - ☐ Rút gọn phân số
4. Xuất ra màn hình các hình sao (với chiều cao của mỗi hình là một số nguyên dương h được nhập từ bàn phím)

```

      *
     **
    ***
   ****
  *****

          *
         **
        ***
       ****
      *****

          *
         **
        ***
       ****
      *****

          *
         **
        ***
       ****
      *****
  
```

5. Nhập vào một chuỗi s và một ký tự c , kiểm tra xem trong s có ký tự c không, nếu có thì xuất vị trí của c trong s , nếu không xuất ra giá trị -1
6. Tương tự như bài tập 4, nhưng thay vì là ký tự c , ta sẽ thay bằng một chuỗi $s2$

Chuyên đề: Hỗ trợ kỹ thuật lập trình

A. Phạm vi hoạt động của một biến

1. Phân loại biến:

- Biến toàn cục: Là những biến có phạm vi hoạt động xuyên suốt từ đầu đến cuối chương trình
- Biến cục bộ:
 - + Là những biến có phạm vi hoạt động tại một thời điểm của chương trình.
 - + Phạm vi hoạt động của biến cục bộ là từ lúc khai báo biến cho đến **dấu đóng khối gần nhất chứa nó**.

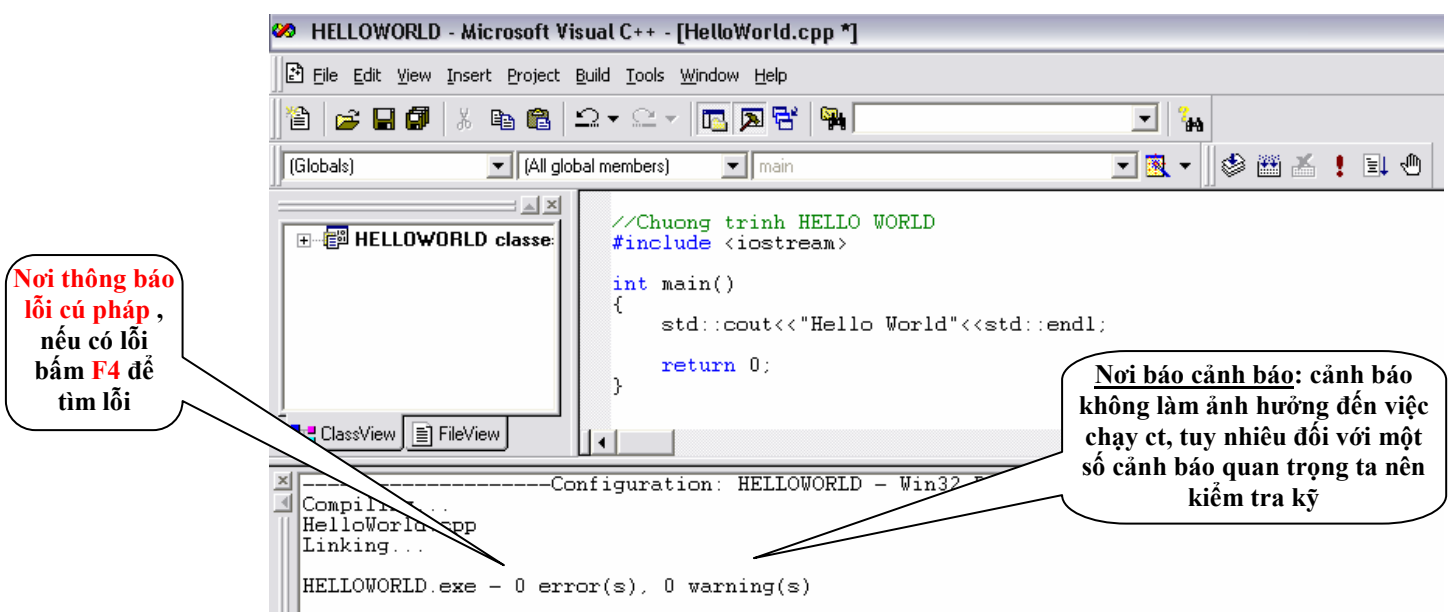
2. Nơi khai báo biến:

- Biến toàn cục: khai báo ngoài hàm main
- Biến cục bộ: khai báo trong hàm main

B. Debug

1. Phân loại các loại lỗi (Error):

- Lỗi cú pháp: Là những lỗi do người lập trình không ghi đúng cú pháp lệnh, những lỗi này sẽ được chương trình biên dịch thông báo.



- Lỗi logic: Là những lỗi chương trình do người lập trình viết không thực hiện đúng với nội dung yêu cầu được đặt ra (ví dụ: yêu cầu của bài toán, yêu cầu của khách hàng,...)

Đối với những lỗi logic chúng ta phải sử dụng **Debug** để tìm ra lỗi.

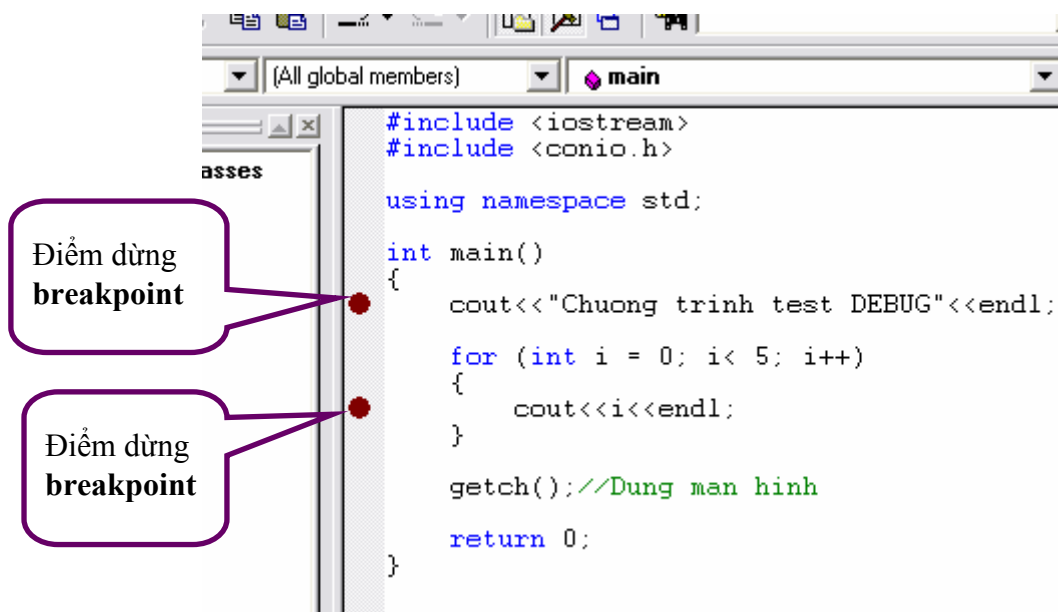
2. Debug là gì?:

- Là cách tìm những lỗi logic bằng cách để chương trình chạy từng dòng lệnh (chạy từng bước từ trên xuống), qua từng dòng lệnh ta sẽ xác định giá trị của từng biến thay đổi như thế nào có đúng với thuật toán đưa ra hay không => từ đó ta có thể xác định được lỗi logic của chương trình.

3. Cách hoạt động của Debug:

Bước 1: Đặt những điểm dừng **breakpoint** bằng bấm phím nóng **F9**

(breakpoint sẽ đánh dấu những dòng lệnh, khi đó chương trình sẽ dừng lại tại những dòng lệnh chứa breakpoint)



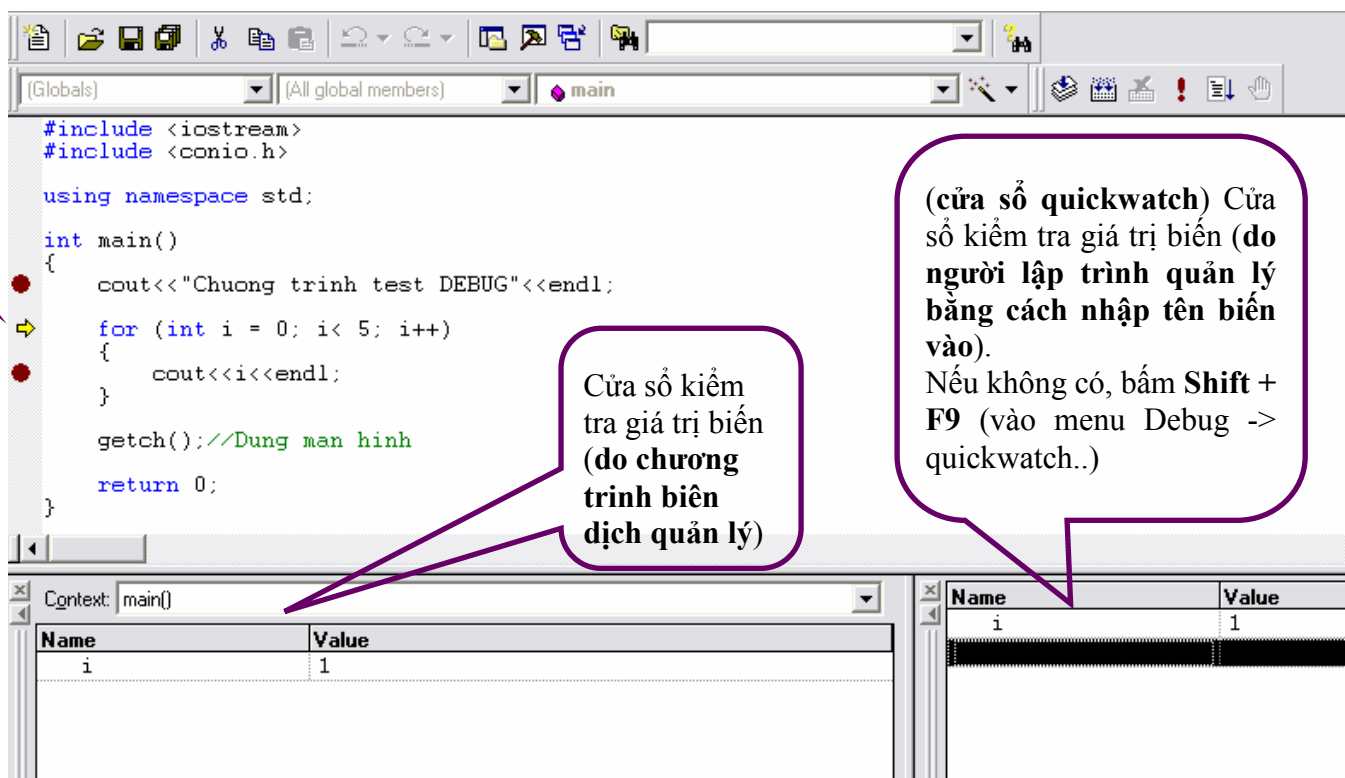
Bước 2: Bấm **F5** để chương trình sẽ bắt đầu chế độ Debug (**chương trình sẽ dừng lại ở vị trí breakpoint**).

Tuỳ theo mục đích của người kiểm tra lỗi, ta có các phím nóng để hỗ trợ

- + **F10**: chương trình chạy từng bước, bỏ qua chương trình con
- + **F11**: chương trình chạy từng bước, nếu gặp chương trình con sẽ vào kiểm tra (không bỏ qua chương trình con).
- + **F5**: đến điểm dừng breakpoint gần đó nhất
- + **F9**: thêm điểm dừng breakpoint
- + **Shift + F5**: Dừng chế độ Debug

Lưu ý:

1 .Màn hình trong chế độ debug gồm 3 cửa sổ chính



2. **Ctrl + F5** : chạy chương trình không Debug

F5: chạy chương trình ở chế độ Debug

Chương 7:

Hàm - Lập trình hướng cấu trúc

7.1. Giới thiệu:

Ví dụ ta có chương trình sau: Nhập một mảng 1 chiều n phần tử, và xuất mảng đó ra màn hình sau khi tăng giá trị từng phần tử lên 1 đơn vị

```

1  #include <iostream>
2  #include <conio>
3
4  using namespace std;
5
6  int a[100];
7  int n;
8  int main()
9  {
10     cout<<"Nhập dữ liệu của mảng 1 chiều: "<<endl;
11     cout<<"nhập vào số lượng phần tử: ";
12     cin>>n;
13     for (int i = 0; i < n; i++)
14     {
15         coutt<<"Nhập vào phần tử thứ "<<i<<": ";
16         cin>>a[i];
17     }
18
19     cout<<"Mảng vừa nhập"<<endl;
20     for (i = 0; i < n; i++)
21         cout<<a[i]<<" ";
22     cout<<endl;
23
24     //Tăng các phần tử của mảng lên 1 đơn vị
25     for (i = 0; i < n; i++)

```

```

35         ++a[i];
26
27         cout<<"Mảng sau khi xử lý: "<<endl;
28         for (i = 0; i < n; i++)
29             cout<<a[i]<<" ";
30         cout<<endl;
31
32
33         getch();
           return 0;
           }

```

Phân tích:

- ☐ Ta nhận thấy đoạn chương trình từ **18 -> 20** và **28->30** rất giống nhau và cùng thể hiện là xuất nội dung của mảng ra màn hình.
- ☐ Làm sao để không lặp lại những đoạn chương trình giống nhau như vậy?

7.2. Định nghĩa hàm (hay còn gọi là đơn thể):**Mục đích của việc xây dựng hàm:**

- Giải quyết vấn đề phải lặp lại nhiều đoạn chương trình giống nhau
- Chia bài toán lớn thành nhiều bài toán nhỏ hơn. Khi đó để giải quyết bài toán lớn ta sẽ đi giải quyết các bài toán nhỏ đó

a. Giải quyết bài toán ví dụ:

```

1  #include <iostream>
2  #include <conio>
3
4  using namespace std;
5
6  int a[100];
7  int n;

```

```

8 //Khai báo hàm (prototype)
9 void  xuat_mang_1chieu();
10
11 int main()
12 {
13     cout<<"Nhập dữ liệu của mảng 1 chiều: "<<endl;
14     cout<<"nhập vào số lượng phần tử: ";
14     cin>>n;
16     for (int i = 0; i < n; i++)
17     {
18         cout<<"Nhập vào phần tử thứ "<<i<<": ";
19         cin>>a[i];
20     }
21
22     cout<<"Mảng vừa nhập"<<endl;
23     xuat_mang_1chieu(); //Gọi hàm xuất mảng 1 chiều
24
25     //Tăng các phần tử của mảng lên 1 đơn vị
26     for (i = 0; i < n; i++)
27         ++a[i];
28
29     cout<<"Mảng sau khi xử lý: "<<endl;
30     xuat_mang_1chieu(); //Gọi hàm xuất mảng 1 chiều
31
32     getch();
33     return 0;
34 }
35
36 void  xuat_mang_1chieu();
37 {
38     for (i = 0; i < n; i++)
39         cout<<a[i]<<" ";
40     cout<<endl;
41
42 }

```

b. Hàm không có giá trị trả về:

Cú pháp:

i. Hàm không có truyền tham số

Cú pháp	Ví dụ
<pre>void tên_hàm() { //Khởi lệnh; }</pre>	<pre>void Hàm_Chào() { cout<<"hello world"; cout<<endl; }</pre>

ii. Hàm có truyền tham số

Cú pháp	Ví dụ
<pre>void tên_hàm(kiểu_thamsô1, kiểu & thamsô2, ..) { //Khởi lệnh; }</pre>	<pre>void Xuất_Tổng(int a, int b) { int s = a + b; cout<<"Tổng "<<a<<"và"<<b<<"là ", cout<<s; }</pre>

c. Hàm có giá trị trả về:

Cú pháp:

i. Hàm không có truyền tham số

Cú pháp	Ví dụ
<pre>kiểu_trả_về tên_hàm() { //Khởi lệnh; return giá_trị_trả_về; }</pre>	<pre>int Tổng_1_đến_10() { int s = 0; for (int i = 1; i<=10 ; i++) s += i; return s; // Giá trị trả về }</pre>

ii. Hàm có truyền tham số

Cú pháp	Ví dụ
<pre>kiểu_trả_về tên_hàm(kiểu_thamsô1, kiểu & thamsô2, ..) { //Khởi lệnh; return giá_trị_trả_về; }</pre>	<pre>int Tính_Tổng(int a, int b) { int s = a + b; return s; // Giá trị trả về }</pre>

7.3. Khai báo hàm – khai báo prototype:

Cú pháp:

void Tên_Hàm(danh sách **kiểu** các tham số);

kiểu_trả_về Tên_Hàm(danh sách **kiểu** các tham số);

Chú ý:

- *Danh sách kiểu tham số*: chỉ ra các kiểu của tham số (có thể có hoặc không có tên biến)

- Vị trí đặt prototype thường là phía **trên hàm main**
- Có dấu **chấm phẩy (;)** ở cuối prototype

Ví dụ:

//Khai báo prototype

void **Xuat_Tong**(int , int) ;

void main()

{

//Khởi lệnh trong hàm main

}

//Định nghĩa hàm

void **Xuat_Tong**(int a, int b)

{

int s = a + b;

cout<<"Tổng của "<<a<<" và "<<b<<" là "<<s;

cout<<endl;

}

7.4. Lời gọi hàm:a. Ví dụ với hàm không có giá trị trả về*//Khai báo prototype***void** Xuat_Tong(int , int) ;**void** main()

{

Xuat_Tong(2 , 3); *//Gọi hàm Xuat_Tong*

}

*//Định nghĩa hàm***void** Xuat_Tong(int a, int b)

{

int s = a + b;

cout<<"Tổng của "<<a<<" và "<<b<<" là "<<s;

cout<<endl;

}

Khi đó có sự so khớp như sau:

Hàm

void Xuat_Tong(int a , int b)

Lời gọi hàm

Xuat_Tong(2 , 3);



Hay tại thời điểm gọi hàm Xuat_Tong *a sẽ có giá trị 2, b sẽ có giá trị là 3*
 Vậy hàm Xuat_Tong sẽ xuất ra màn hình:

*Tổng của 3 và 5 là 8*b. Ví dụ với hàm có giá trị trả về*//Khai báo prototype***int** Tinh_Tong(int , int) ;**void** main()

{

int i;

i = Xuat_Tong(2 , 3); *//Gọi hàm Tinh_Tong*

cout<<i;

}

*//Định nghĩa hàm***int** Tinh_Tong(int a, int b)

{

int s = a + b;

return s;

}

Tương tự ta có sự so khớp như sau:

Hàm	int	Tinh_Tong(int a , int b)
	↓ 5	↑ ↑
Lời gọi hàm	i	= Tinh_Tong(2 , 3);

Hay tại thời điểm gọi hàm `Xuat_Tong` *a* sẽ có giá trị 2, *b* sẽ có giá trị là 3
Và giá trị 5 sẽ được **trả về** cho biến *i*, khi đó *i* sẽ có giá trị là 5

Chú ý:

Lệnh **return** ngoài ý nghĩa là trả về giá trị của hàm, nó còn có ý nghĩa là thoát khỏi chương trình con chứa nó

7.5. Tham trị và tham chiếu (quan trọng):

Tham trị	Tham chiếu
- khi khai báo không có dấu & giữa kiểu và tham số Ví dụ: void ham(int a) { //Khởi lệnh }	- khi khai báo có dấu & giữa kiểu và tham số void ham(int &a) { //Khởi lệnh }
- Không làm thay đổi biến được gọi	- Làm thay đổi biến được gọi
void binh_phuong(int); void main() { int i = 5; binh_phuong(i); cout<<i; // i = 5 // Không làm thay đổi i } void binh_phuong(int a) { a *= a; cout<<a; // a = 25 }	void binh_phuong(int &); void main() { int i = 5; binh_phuong(i); cout<<i; // i = 25 // Làm thay đổi i } void binh_phuong(int &a) { a *= a; cout<<a; // a = 25 }

7.6. Giá trị mặc định của tham số:**a. Ý nghĩa:**

Cho phép gọi hàm một cách linh hoạt:

- Nếu có truyền tham số thì hàm sẽ lấy tham số được truyền
- Nếu không có truyền tham số thì hàm sẽ lấy giá trị mặc định

b. Các khai báo:

Ta chỉ cần khai báo các giá trị mặc định của tham số lúc khai báo prototype (khai báo hàm)

kiểu_trả_về tên_hàm(kiểu_thamsố1 = **giá trị 1**, kiểu_thamsố2=**giá trị 2**, ...);

Chú ý:

- Giá trị mặc định của tham số phải được khai báo từ **phải sang trái**

Ví dụ 1:

```
int Tinh_Tong(int , int b = 1); //Giá trị mặc định của b là 1
```

```
int main()
{
    int i;
    i = Tinh_Tong ( 2 , 3);
    cout<<i;           //i =5

    int j;
    j = Tinh_Tong( 2 ); //Chỉ truyền có 1 tham số
                        //Khi đó b sẽ lấy giá trị mặc định là 1
    cout<<j;           //j = 2 + 1 = 3
}
```

//Định nghĩa hàm vẫn như bình thường

```
int Tinh_Tong(int a, int b)
{
    int s = a + b;
    return s;
}
```

Ví dụ 2:

Khai báo giá trị mặc định như sau là **sai**

```
int Tinh_Tong(int a = 1, int b);
```

Do a được khai báo có giá trị mặc định trong khi b không có giá trị mặc định (*nhắc lại: Giá trị mặc định của tham số phải được khai báo từ phải sang trái*)

7.7. Quá tải hàm:**a. Các thành phần của hàm:**

Hai hàm khác nhau khi chúng có ít nhất 1 trong 3 thành phần khác nhau, gồm:

- Tên hàm
- Kiểu của tham số
- Số lượng tham số

b. Quá tải hàm:

Quá tải hàm là những hàm có cùng tên hàm, nếu rơi vào 1 trong 2 trường hợp sau:

- Số lượng tham số khác nhau
- Cùng số lượng tham số, khác nhau ở một kiểu dữ liệu nào đó của tham số

Ví dụ 1:

```
void  Xuat_Tong(int a, int b);
void  Xuat_Tong(int a, int b, int c);

int main()
{
    Xuat_Tong( 1 , 2);

    Xuat_Tong( 1 , 2, 3);
}

void  Xuat_Tong(int a, int b)
{
    cout<<a+b;
}

void  Xuat_Tong(int a, int b, int c)
{
    cout<<a + b + c;
}
```

Ví dụ 2:

```
void  Xuat_Tong(int a, int b);
void  Xuat_Tong(int a, float b);
```

7.8. Quá tải toán tử:**a. Các toán tử trong C++:**

Các toán tử quen thuộc trong C++

Toán tử	Ví dụ
+	<code>a = b + c;</code>
-	<code>a = b - c;</code>
*	<code>a = b * c;</code>
/	<code>a = b / c;</code>
%	<code>a = b % c;</code>
=	<code>a = b;</code>
==	<code>if (a == b) {}</code>
[]	<code>a[1];</code> //với <i>a</i> là mảng
<<	<code>cout<<i;</code>
>>	<code>cin>>i;</code>

Các toán trong C++ đều có chung một cách đặt **tên hàm** theo mô tả sau:

operator <ký_hiệu_phép_toán>

b. Quá tải toán tử:

Ứng dụng: ta có thể sử dụng các toán tử của C++ đối với các biến cấu trúc mà ta tự nghĩa như những kiểu bình thường trong C++

Ví dụ:

với kiểu cấu trúc PHANSO được khai báo như sau:

```
struct PHANSO
```

```
{
```

```
    int x;
```

```
    int mau;
```

```
};
```

```
int main()
```

```
{
```

```
    PHANSO ps1, ps2;
```

```
    cin>>ps1>>ps2; //Nhập như kiểu dữ liệu bình thường
```

```
    PHANSO ps;
```

```
    ps = ps1 + ps2; //Thực hiện phép cộng như kiểu dữ liệu
                    //bình thường khác
```

```
    cout<<ps;      //Xuất như kiểu dữ liệu bình thường
```

```
}
```

Để làm áp dụng các toán tử vào kiểu PHANSO như ví dụ trên thì ta phải **định nghĩa các hàm quá tải toán tử**

Ví dụ:**1. Quá tải toán tử + của 2 phân số**

```
PHANSO operator + (PHANSO a, PHANSO b)
{
    PHANSO ps;
    ps.tu = a.tu * b.mau + a.mau * b.tu;
    ps.mau = a.mau * b.mau;
    return ps;
}
```

2. Quá tải toán tử << cho phân số

```
ostream& operator(ostream& os, PHANSO ps)
{
    os<<ps.tu<<" / "<<ps.mau;

    return os;
}
```

3. Quá tải toán tử >> cho phân số

```
istream & operator(istream & is, PHANSO ps)
{
    is>>ps.tu;
    is>>ps.mau;

    return is;
}
```

Bài tập về hàm:

1. Sử dụng hàm và quá tải hàm, quá tải toán tử để viết lại bài tập phân số
 - a. kiểm tra tính đúng đắn của phân số
 - b. Nhập / Xuất phân số
 - c. Rút gọn phân số
 - d. Cộng trừ nhân chia phân số
 - e. Quá tải toán tử nhập, xuất, =, ==, +, -, *, /
2. Sử dụng hàm và quá tải hàm, quá tải toán tử viết lại bài tập về mảng 1 chiều kiểu số nguyên
 - a. Nhập / xuất mảng 1 chiều
 - b. Tìm các số nguyên tố trong mảng
 - c. Tìm các số chính phương trong mảng
 - d. Thêm 1 phần tử vào đầu dãy
 - e. Thêm 1 phần tử vào cuối dãy
 - f. Thêm 1 phần tử vào vị trí k
 - g. Xoá 1 phần tử ở vị trí k
 - h. Tìm kiếm phần tử có giá trị x
 - i. Tìm min
 - j. Tìm max
 - k. Tính tổng
 - l. Tính trung bình các số hạng
 - m. Đảo ngược dãy
 - n. Sắp xếp theo thứ tự tăng/giảm
3. Sử dụng hàm và quá tải hàm, quá tải toán tử viết lại bài tập về mảng 2 chiều kiểu số nguyên
 - a. Nhập / xuất mảng
 - b. Thêm 1 dòng vào đầu ma trận
 - c. Thêm 1 phần tử vào cuối ma trận
 - d. Thêm 1 phần tử vào dòng thứ k
 - e. Xoá dòng thứ k
 - f. Tìm min
 - g. Tìm max
 - h. Tính tổng
 - i. Tính trung bình các số hạng
 - j. Tìm kiếm phần tử có giá trị x

Chương 8: Tổ chức thư viện

8.1. Ý nghĩa của việc tổ chức thư viện:

Khi lập trình, lập trình viên luôn có xu hướng sử dụng lại những hàm mà mình đã viết, không phải tốn công viết lại như ban đầu. **Vì vậy họ tổ chức các hàm thường sử dụng lại thành thư viện**, khi nào cần sử dụng lại thì chỉ cần gọi hàm đó ra, mà không cần phải định nghĩa lại hàm.

8.2. Các bước tạo thư viện

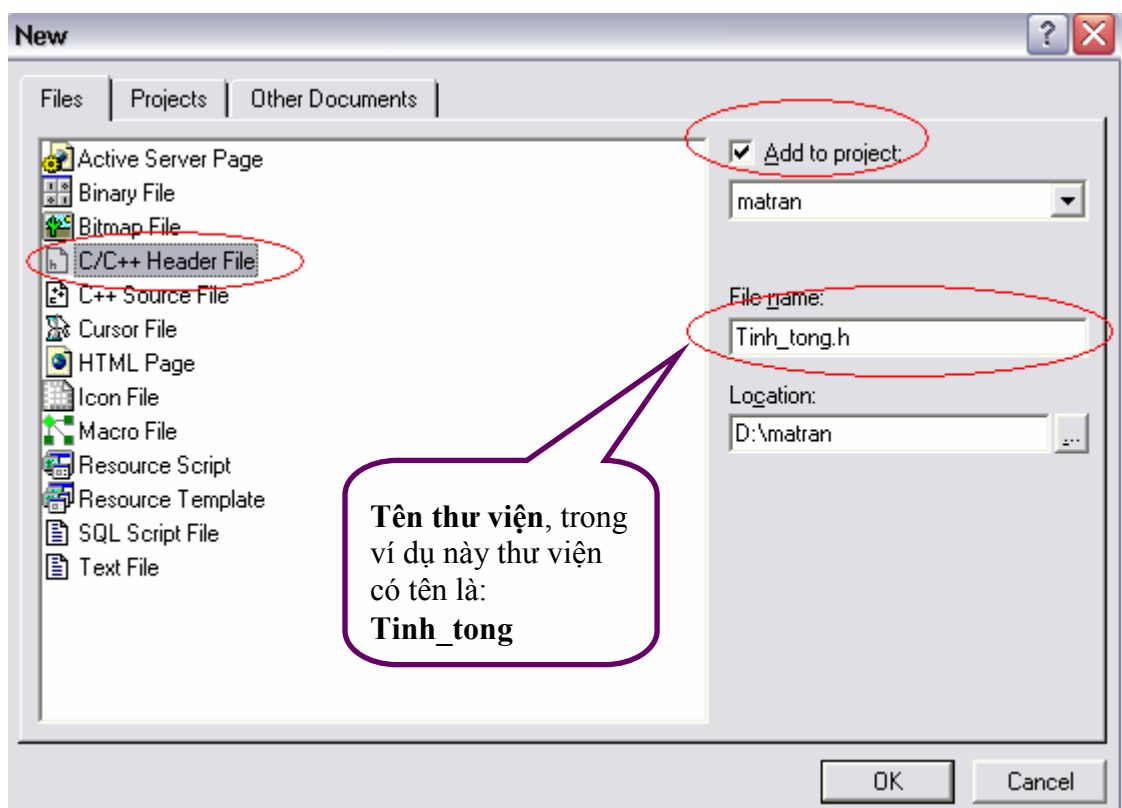
Các bước tạo thư viện mang tên **tenthuvien**

i. Giai đoạn 1: Tạo file tenthuvien.h

Bước 1: Vào menu Project -> Add to project -> New

Bước 2: Tại ther File, chọn C/C++ header file

Bước 3: Đánh vào: **tenthuvien.h** vào ô **file name**



Bước 4: bấm OK

Bước 5: Khi đó ta có một file là **tenthuvien.h** , trong flie này ta soạn nội dung như sau:

- Đầu file:

```
#ifndef __TENTHUVIEN_H__  
#define __TENTHUVIEN_H__
```

- Giữa file:

- Khai báo hàm (prototype của hàm) của các hàm mà ta muốn tổ chức thành thư viện

- Cuối file:

```
#endif
```

ii. Giai đoạn 2: Tạo file tenthuvien.cpp

Bước 1: Vào menu Project -> Add to project -> New

Bước 2: Tại ther File, chọn C/C++ Source File

Bước 3: Đánh vào: **tenthuvien.cpp** vào ô **file name**

Sau đó chọn ok

Bước 4: Khi đó ta có một file là **tenthuvien.cpp** , trong flie này ta soạn nội dung:

- Đầu file:

```
#include "tenthuvien.h"
```

- Các **định nghĩa hàm** đã được khai báo trong **tenthuvien.h**

iii. Giai đoạn 2: Sử dụng thư viện tenthuvien

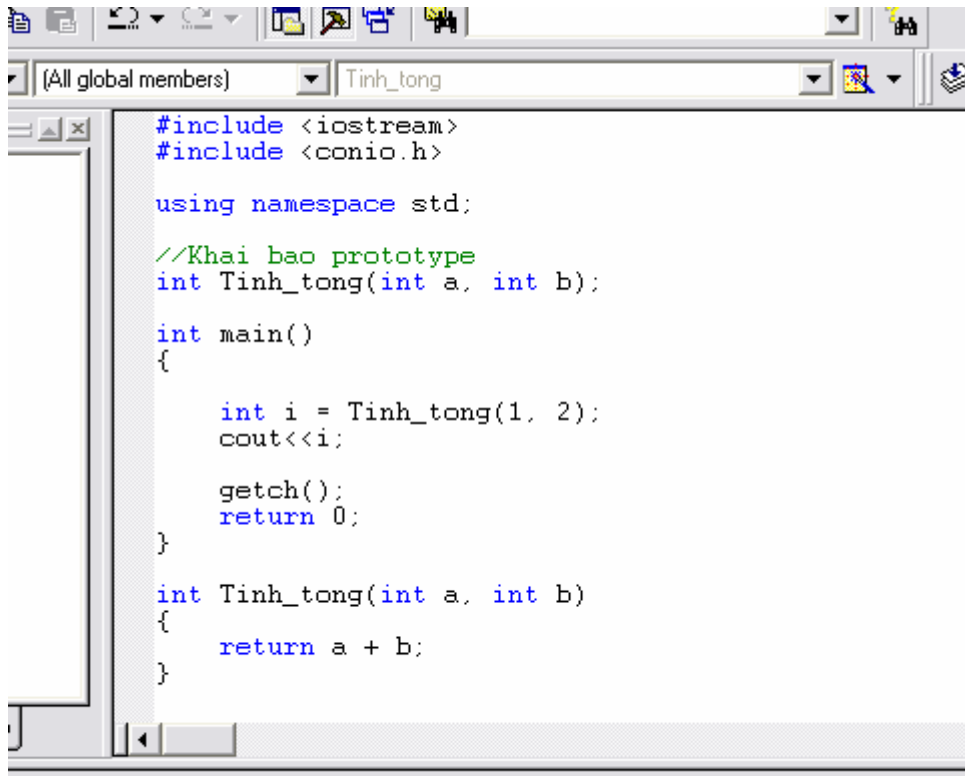
Ở những file nào có sử dụng các hàm trong thư viện **tenthuvien** ta đều phải khai báo thư viện ở đầu file

```
#include "tenthuvien.h"
```

Sau đó ta có thể sử dụng các hàm trong thư viện một cách bình thường như các hàm trong C++.

8.3. Phân tích ví dụ

Ví dụ ta có chương trình tính tổng 2 số a và b được viết một cách bình thường như sau:



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

using namespace std;

//Khai báo prototype
int Tinh_tong(int a, int b);

int main()
{
    int i = Tinh_tong(1, 2);
    cout<<i;

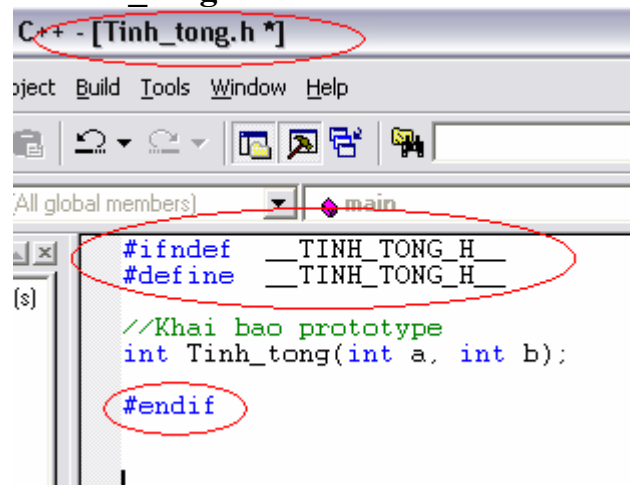
    getch();
    return 0;
}

int Tinh_tong(int a, int b)
{
    return a + b;
}
```

Viết lại chương trình trên với cách tổ chức thư viện – xây dựng thư viện

Tinh_tong

i. Nội dung file **Tinh_tong.h**



```

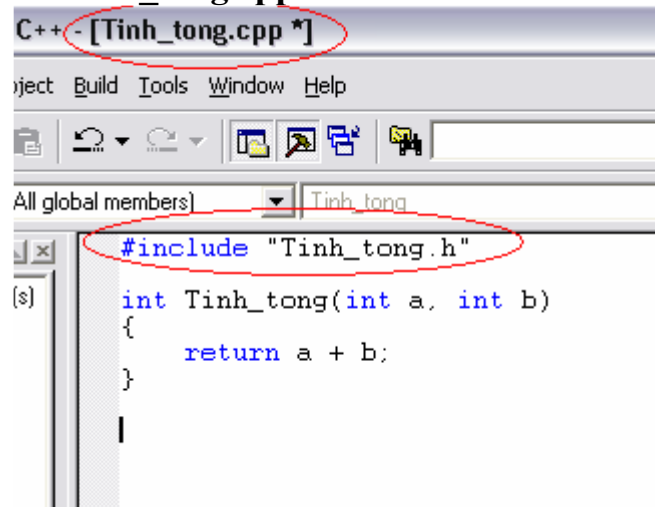
C++ - [Tinh_tong.h *]
Project Build Tools Window Help

[All global members] | main
(s)
#ifndef __TINH_TONG_H__
#define __TINH_TONG_H__

//Khai báo prototype
int Tinh_tong(int a, int b);

#endif

```


ii. Nội dung file **Tinh_tong.cpp**


```

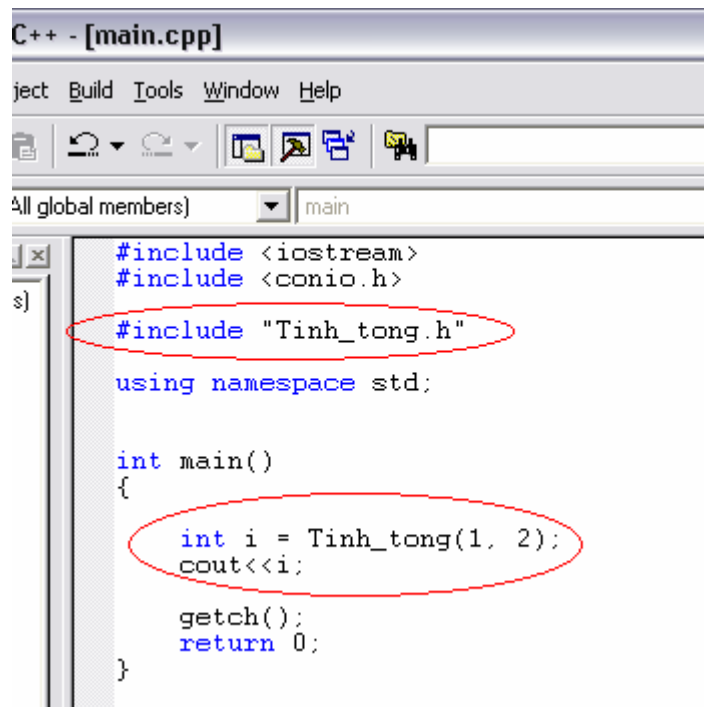
C++ - [Tinh_tong.cpp]
Project Build Tools Window Help

All global members) Tinh_tong

#include "Tinh_tong.h"

(s) int Tinh_tong(int a, int b)
    {
        return a + b;
    }
  
```

iii. Nội dung file sử dụng thư viện Tinh_tong



```

C++ - [main.cpp]
Project Build Tools Window Help

All global members) main

#include <iostream>
#include <conio.h>
#include "Tinh_tong.h"
using namespace std;

int main()
{
    int i = Tinh_tong(1, 2);
    cout<<i;

    getch();
    return 0;
}
  
```

8.4. **Chỉ thị #include:** so sánh 2 cách khai báo thư viện

#include <thư_viện.h>	#include "thư_viện.h"
Chương trình sẽ tìm thư_viện trong thư mục mặc định chứa các thư viện trong C++	<ul style="list-style-type: none"> - Chương trình sẽ tìm thư viện trong thư mục gốc (nên ct đang chạy) - Nếu không tìm thấy thư viện, chương trình sẽ tìm trong thư mục mặc định chứa các thư viện trong C++
Thường là các thư viện chuẩn của C++	Thường là các thư viện do người lập trình tự định nghĩa

Bài tập:

Viết lại các tất cả bài tập trong chương hàm (chương 10) bằng cách tổ chức thành thư viện

Chương 9: Quản lý bộ nhớ động

9.1. Khái niệm về ô nhớ - địa chỉ trong lập trình:

Khi ta khai báo một biến, điều đó có nghĩa là chương trình sẽ tạo cho ta một ô nhớ để lưu trữ biến đó.

```
int main()
{
```

```
    int a;
    int b;
    int c;
```

```
    a = 10;
    b = 20;
    c = 30;
```

```
    return 0
```

```
}
```

8	30	c
4	20	b
0	10	a
địa chỉ	Giá trị hiện thời của ô nhớ	biến

Một ô nhớ gồm có 2 thành phần cơ bản sau:

- Địa chỉ của ô nhớ (do hệ điều hành quản lý)
- Giá trị hiện thời của ô nhớ đó

Với một biến a, ta muốn biết địa chỉ ô nhớ lưu trữ của nó là bao nhiêu thì ta dùng toán tử **&** để xác định địa chỉ.

Ví dụ:

```
int a = 20;
cout<<&a; //xuất địa chỉ ô nhớ chứa biến a
```

9.2. Tổng quan biến con trỏ (biến động)

- Là biến lưu địa chỉ của ô nhớ
- Cú pháp khai báo
kiểu_dữ_liệu * tên_biến;
- Ý nghĩa: tạo một biến lưu trữ địa chỉ các biến thuộc kiểu đã chỉ ra

Ví dụ:

```
int * contro; //Biến con trỏ lưu trữ địa chỉ các biến kiểu số nguyên
float* contro; //Biến con trỏ lưu trữ địa chỉ các biến kiểu số thực
```

9.3. Các thao tác trên con trỏ:

- i. Con trỏ lưu địa chỉ ô nhớ, nên

```
cout<<p; //Xuất địa chỉ ô nhớ mà p đang lưu
```

ii. Xuất giá trị hiện thời tại ô nhớ do con trỏ lưu trữ

- Toán tử : *

- Ví dụ:

```
int a = 5;
```

```
int * contro;
```

```
contro = &a; //contro lưu (trỏ) đến địa chỉ của a
```

```
cout<< *contro; //xuất giá trị của địa chỉ mà contro đang lưu
//trữ (tức là 5)
```

iii. Tạo ô nhớ của một kiểu dữ liệu xác định

- Toán tử :
- new**

- Cú pháp:

```
new kiểu_dữ_liệu;
```

- Ví dụ:

```
int * p = new int;
```

iv. Hủy biến con trỏ (hủy ô nhớ do con trỏ lưu trữ địa chỉ)

- Toán tử :
- delete**

- Cú pháp:

```
delete biến_kiểu_con_trỏ;
```

- Ví dụ:

```
delete p;
```

v. NULL:

- NULL – là một hằng của mọi con trỏ

- Một con trỏ có giá trị là NULL, tức là nó chưa trỏ đến địa chỉ của ô nhớ nào cả

Ví dụ:

```
int * p;
```

```
if ( p == NULL)
```

```
    p = new int;
```

	Biến tĩnh	Biến động
Kích thước	cố định	không cố định
Phạm vi sử dụng	từ lúc khai báo đến hết khối gần nhất chứa nó.	từ lúc được tạo ra và kết thúc khi bị hủy.
Tên của biến dùng để	Lưu giá trị	Lưu địa chỉ
Cách truy xuất giá trị	tên_biến	*tên_biến
Cách truy cập địa chỉ	& tên_biến	tên_biến
	Tự động giải phóng khi hết phạm vi sử dụng	Không tự động giải phóng. Lập trình viên phải lo việc này.

9.4. Mảng động (mảng con trỏ)

- Khai báo
kiểu_dữ_liệu* tên_biến = **new** kiểu_dữ_liệu[spt];
- Hủy mảng động:
delete [] tên_biến;
- Ưu điểm: sử dụng được mảng với số phần tử rất lớn
- Ví dụ:

```
#define MAX 100000
int * a = new int [MAX];
for (int i = 0 ; i< MAX ; i++)
    a[i] = i;
delete[] a;
```

9.5. Chuỗi con trỏ

- Khai báo
- `char* tên_chuỗi = new char[chiều_dài_tối_đa_của_chuỗi];`
- Huỷ chuỗi:
- `delete[] chuỗi;`
- Khi đó ta có thể sử dụng các lệnh về chuỗi áp dụng cho chuỗi con trỏ một cách bình thường

9.6. Con trỏ kiểu cấu trúc:

Ví dụ:

```
struct PHANSO
{
    int tu;
    int mau;
};

int main()
{
    PHANSO *ps; //con trỏ lưu địa chỉ của biến kiểu cấu trúc
    ps = new PHANSO;

    //truy cập đến trường của cấu trúc bằng dấu chấm (.)
    *ps . tu = 1;
    *ps . mau = 2;
    cout<<"Xuất phân số: ";
    cout<< *ps.tu << " / " << *ps.mau;

    //truy cập đến trường của cấu trúc bằng dấu mũi tên (->)
    ps->tu = 1;
    ps->mau = 2;
    cout<<"Xuất phân số: ";
    cout<< ps->tu << " / " << ps->mau;

    delete ps;
}
```

*** Cách truy cập vào các trường của biến con trỏ cấu trúc:**

Cách 1:

`* tên_biến_con_trỏ . tên_trường`

Cách 2:

`tên_biến_con_trỏ -> tên_trường`

9.7. Mối quan hệ giữa mảng và con trỏ:**Tên mảng là một hằng địa chỉ**

```
int a[10];
```

a	tương đương với &a[0] (địa chỉ của phần tử a[0])
a + i	tương đương với &a[i] (địa chỉ của phần tử a[i])
*(a+i)	tương đương với a[i]

Vì vậy ta có thể trỏ con trỏ đến biến mảng như sau:

```
int *p;
```

```
int a;
```

```
p = a;
```

p	tương đương với &a[0] (địa chỉ của phần tử a[0])
p + i	tương đương với &a[i] (địa chỉ của phần tử a[i])
*(p+i)	tương đương với a[i]

hoặc:

```
int *p;
```

```
int a;
```

```
p = &a[k];
```

p	tương đương với &a[k] (địa chỉ của phần tử a[k])
p + i	tương đương với &a[k + i] (địa chỉ của phần tử a[k+i])
*(p+i)	tương đương với a[k+i]

9.8. Một số ví dụ đơn giản:

<pre>#include <iostream.h> #include <conio.h> using namespace std; int main() { int * p; int a = 1; p = &a; cout<<*p<<endl; *p = 2; cout<<*p<<endl; cout<<a<<endl; int j = 3; p = &j; cout<<*p; cout<<a; getch(); return 0; }</pre>	<pre>#include <iostream.h> #include <conio.h> using namespace std; int main() { int * p = new int; int * q; q = new int; *q = 1; p = q; cout<<*p<<endl; cout<<*q<<endl; *p = 2; cout<<*p<<endl; cout<<*q<<endl; int* t = new int; *t = 3; p = t; }</pre>
--	---

```
cout<<*t<<endl;  
cout<<*p<<endl;  
cout<<*q<<endl;
```

```
*q = 10;  
*p = *q;  
cout<<t <<endl;  
cout<<p<<endl;  
cout<<*q<<endl;
```

```
delete p,q;
```

```
getch();  
return 0;
```

```
}
```


Chương 10: Thư viện STL

(Standard Template Library)

10.1. **iostream**: #include <iostream>

các lệnh liên quan đến việc xuất nhập

Lệnh	Ý nghĩa	Ví dụ
cout	xuất chuẩn	cout<<"Hello world";
cerr	Xuất lỗi chuẩn	cerr<<"Error";
cin	nhập chuẩn	cin>>n;

10.2. **iomanip**: #include <iomanip>

Lệnh	Ý nghĩa	Ví dụ
setw(n)	<ul style="list-style-type: none"> - Định độ dài tối thiểu khi xuất - Khi độ dài của dữ liệu xuất không đủ thì nó sẽ tự động thêm vào khoảng trắng phía trước để đủ độ dài 	<pre>cout<<"Hello"; cout<<setw(10)<<"Hello"; cout<<setw(15)<<"Hello";</pre> <p><i>//Kết quả xuất ra là:</i></p> <pre>Hello Hello Hello</pre>

10.3. **cmath**: #include <cmath>

Lệnh	Ý nghĩa	Ví dụ
abs	x , với x là số nguyên	int i = -12; int a = abs(i);
ceil	cận trên	int j = ceil(i);
floor	cận dưới	int j = floor(i);
fabs	x , với x là số thực	float f = -12.34; float b = fabs(i);
sqrt	Hàm lấy căn bậc 2	float f = sqrt(i);
pow	Hàm lũy thừa	pow(a,8); // a ⁸
sin		cout<<sin(x);
cos		cout<<cos(x);
tan		cout<<tan(x);
log	ln	cout<<log(x);

***Làm quen với các đối tượng cơ bản được xây dựng sẵn trong C++**

Khi viết một chương trình người lập trình thường mô hình hoá các đối tượng trong thực tế thành đối tượng cụ thể trong lập trình. Ví dụ, nếu bạn viết game đá banh, thì các đối tượng ngoài thực tế cầu thủ, trọng tài, trái banh, ... đều được mô hình thành các đối tượng.

Đối với mỗi đối tượng đều có những **thuộc tính riêng**, ví dụ đối với đối tượng cầu thủ thì thuộc tính là chiều cao, cân nặng,... và có một số cách thức hoạt động như dẫn bóng, sút bóng, đội đầu, gọi là **phương thức của đối tượng**

Lập trình hướng đối tượng cho phép ta xây dựng các đối tượng chứa đựng gói gọn các thuộc tính, phương thức.

Trong C++ đã xây dựng cho chúng ta một số đối tượng cơ bản giúp người lập trình có thể thao tác dễ dàng với các dữ liệu phức tạp.

Ta sẽ lần lượt đi qua các đối tượng cơ bản sau:

string : đối tượng chuỗi
fstream : đối tượng quản lý tập tin
vector : đối tượng vector (tương tự mảng)

10.4. Đối tượng string: #include <string>

+ khai báo đối tượng string

string tên_đối_tượng_string;

+ Nhập và xuất

string s; //Khai báo một đối tượng string

cin>>s; //Nhập vào chuỗi không có khoảng trắng

getline(cin, s); //Nhập vào một chuỗi có khoảng trắng

+ Các phương thức của đối tượng string

Phương thức	Ý nghĩa	Ví dụ
size()	- Lấy chiều dài chuỗi	string s = "hello"; cout<<s.size(); // 5
erase()	Xoá chuỗi thành chuỗi rỗng	s.erase();
erase(int chỉ_số_bắt_đầu, int slượng_phần_tử_bị_xoá)	Xoá một khoảng bắt đầu từ chỉ số đầu	string s = "hello"; s.erase(1,3); cout<<s; // s = ho
empty()	Kiểm tra chuỗi có rỗng không	if (! s.empty()) { //Khởi lệnh }
compare()	So sánh 2 chuỗi string	string s1 = "hello"; string s2 = "world"; s1.compare(s1); Trả về 0: s1 "bằng" s2 1: s1 "lớn hơn" s2

		-1: s1 “nhỏ hơn” s2
find(char ký tự , int chisố_bđầu) find(string chuỗi, int chisố_bđầu)	Tìm ký tử (hoặc chuỗi) trong dãy bắt đầu từ chỉ_số_bđầu của chuỗi. - nếu tìm thấy sẽ chỉ ra vị trí của ký tự (chuỗi) trong chuỗi - nếu không tìm thấy sẽ trả về -1	string s = “hello”; int i = s.find(‘ e ’); //i=1 int j = s.find(‘ t ’); //i=-1
Toán tử +	cộng 2 chuỗi	string s1 =”Hello” string s2 =” World” string s = s1 + s2; //s=”Hello World”
Toán tử []	Truy cập đến phần tử của chuỗi	cout<<s[0]; cout<<s[1];

10.5. Đối tượng vector

a. Mô tả: Là một đối tượng được mô tả tương tự như mảng 1 chiều, được xây dựng nhiều phương thức giúp ta thao tác dễ dàng và tiện lợi đối với đối với các bài toán liên quan đến dãy

b. Khai báo thư viện

```
#include <vector>
```

c. Cú pháp

```
vector<kiểu_dữ_liệu> tên_vector;
```

Khai báo một vector lưu trữ các thành phần là thuộc kiểu_dữ_liệu, tương tự như mảng 1 chiều

Nhập dữ liệu từ bàn phím	Nhập dữ liệu từ file
<pre>#include <iostream> #include <conio.h> using namespace std; int main() { int a[100]; //khai báo mảng 1 chiều int n = 3; a[0] = 2; a[1] = 1; a[2] = 3; cout<<”Xuất mảng:”<<endl; for (int i =0 ; i<n; i++) cout<<a[i]<<” “;</pre>	<pre>#include <iostream> #include <conio.h> #include <vector> using namespace std; int main() { vector<int> vec; //Khai báo vector vec.push_back(2); vec.push_back(1); vec.push_back(3); cout<<”Xuất vector:”<<endl; for (int i =0 ; i< vec.size(); i++) cout<<vec[i] <<” “;</pre>

<pre> getch(); return 0; } </pre>	<pre> getch(); return 0; } </pre>
-----------------------------------	-----------------------------------

Các phương thức cơ bản của vector

Phương thức	Ý nghĩa	Ví dụ
push_back	Thêm phần tử vào cuối vector	<pre> vector<int> vec; vec.push_back(2); vec.push_back(1); vec.push_back(3); //Khi đó //vec[0]=2,vec[1]=2,vec[3]=3 cout<<vec.size(); // = 3 </pre>
size()	- kích thước (số lượng phần tử) của vector	cout<<vec.size(); // = 3
clear()	Làm rỗng vector	vec.clear();
empty()	Kiểm tra vector có rỗng không	<pre> if (! vec.empty()) { //Khởi lệnh } </pre>
Toán tử []	Truy cập đến phần tử của vector	<pre> cout<<vec[0]; cout<<vec[1]; </pre>

10.6. fstream: #include <fstream>

a. Mục đích:

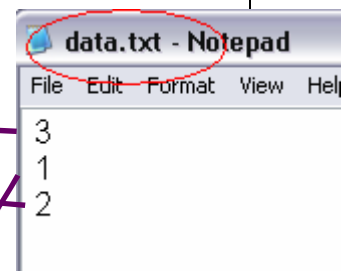
- + Quản lý việc nhập dữ liệu từ File (đối tượng **ifstream**)
- + Quản lý việc xuất dữ liệu vào File (đối tượng **ofstream**)

b. Nhập dữ liệu từ File:

Nhập dữ liệu từ bàn phím	Nhập dữ liệu từ file
+ Khó quản lý dữ liệu nhập	+ Dữ liệu nằm trong file, nên dễ quản lý
+ Mỗi lần chạy chương trình là phải nhập dữ liệu lại	+ Mỗi lần chạy chương trình ta không cần nhập dữ liệu lại
+ Không cần file lưu trữ dữ liệu	+ Phải tạo file lưu trữ dữ liệu trước khi chạy chương trình (thường là *.txt)

Ví dụ viết chương trình nhập vào ba số nguyên a, b, c

Nhập dữ liệu từ bàn phím	Nhập dữ liệu từ file
<pre>#include <iostream> #include <conio.h> using namespace std; int main() { int a, b, c; cin>>a; cin>>b>>c; getch(); return 0; }</pre>	<pre>#include <iostream> #include <conio.h> #include <fstream> using namespace std; int main() { int a, b, c; ifstream ifile; //Tạo đối tượng ifstream ifile.open("data.txt"); //Mở file data.txt ifile>>a; ifile>>b>>c; //Đóng file ifile.close(); getch(); return 0; }</pre>



Chú ý:

Người ta thường dùng phương thức **eof()** của **ifstream** để kiểm tra là đã đọc hết file chưa (thường dùng việc nhập dữ liệu cho mảng từ file)

```
ifstream ifile; //Tạo đối tượng ifstream
ifile.open("data.txt"); //Mở file data.txt
```

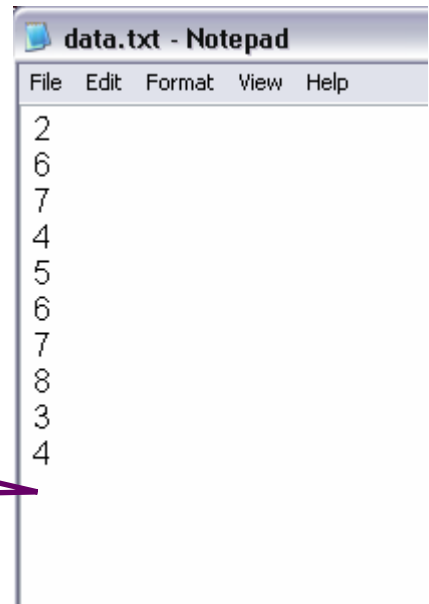
```
int n = 0;
```

```
//Trong khi chưa đọc hết file
```

```
while (!ifile.eof())
{
    ifile>>a[n++];
}
```

```
//Đóng file
ifile.close();
```

Vị trí cuối file:
ifile.eof() == true

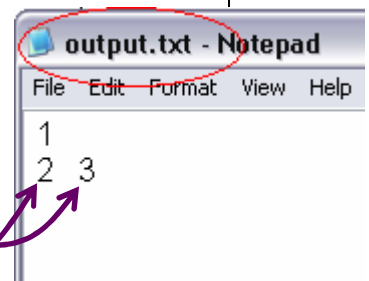


c. Xuất dữ liệu vào File:

Xuất dữ liệu ra màn hình	Xuất dữ liệu ra file
+ Khó quản lý dữ liệu xuất	+ Dữ liệu được xuất ra file, nên dễ quản lý

Ví dụ viết chương trình nhập vào ba số nguyên a, b, c

Nhập dữ liệu từ bàn phím	Nhập dữ liệu từ file
<pre>#include <iostream> #include <conio.h> using namespace std; int main() { int a, b, c; a= 1; b= 2; c= 3; cout<<a<<endl; cout<<b<<" "<<c<<endl; getch(); return 0; }</pre>	<pre>#include <iostream> #include <conio.h> #include <fstream> using namespace std; int main() { int a, b, c; a= 1; b= 2; c= 3; ofstream ofile; //Tạo đối tượng ofstream ofile.open("out.txt"); //Mở file output.txt ofile<<a; ofile<<b<<" "<<c; //Đóng file ofile.close(); getch(); return 0; }</pre>



Lưu ý: Khi mở một file thì ta phải nhớ đóng file **đóng** lại khi đã sử dụng xong