CHƯƠNG 1: CÁC KHÁI NIỆM CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH C/C++

1.1. KHÁI QUÁT VỀ NGÔN NGỮ LẬP TRÌNH C/C++

1.1.1. Giới thiệu

1.1.1.1. Các khái niệm cơ bản

Chương trình (program): Tập hợp tòan bộ mã nguồn với các mô tả, khai báo, các hàm thể hiện theo mộ giải thuật cụ thể nhằm giải quyết một vấn đề hay bài tóan cụ thể. Thông thường một chương trình viết bằng ngôn ngữ C được phân bố trên các file có đuôi .c, .cpp, .h. Sau khi được biên dịch thì sẽ tạo ra tập tin thi hành được dạng nhị phân với đuôi .exe hay .dll

Ngôn ngữ lập trình (programming language): Tập hợp các ký hiệu (symbol), mô tả (definition), khai báo (declaration), hệ thống cú pháp (syntax) và qui ước cho phép người lập trình mã hóa một giải thuật thành một chương trình dưới dạng văn bản (text) hay còn gọi là mã nguồn (source code). Một ngôn ngữ lập trình thường được tích hợp một công cụ cho phép biên dịch (compiling) hay thông dịch (interpreting) mã nguồn thành ngôn ngữ máy ở dạng nhị phân (binary).

C là một ngôn ngữ lập trình cấp cao nhưng cũng có khả năng thực hiện các thao tác can thiệp vào hệ thống máy tính như ngôn ngữ Assembler. Vì vậy C có tính tổng quát và rất linh hoạt nên còn được gọi là ngôn ngữ lập trình cấp trung gian.

Lập trình (programming): Viết chương trình dựa vào một giải thuật (algorithm) và được cài đặt bằng một ngôn ngữ lập trình (programming language) xác định nào đó nhằm giải quyết tự động một vấn đề hay bài tóan được đặt ra trong thực tế trên máy tính.

Phần mềm (software): Khái niệm rộng hơn chương trình. Là tập hợp các chương trình đơn lẻ tạo thành một hệ thống chương trình với sự liên kết chặt chẽ và logic. Trong thực tế thì một software cũng có thể là một chương trình đơn.

1.1.1.2. Giới thiệu ngôn ngữ lập trình C/C++

C là một ngôn ngữ lập trình cấp cao (high-level programming language) được phát triển bởi Brian Kernighan và Dennis Ritchie làm việc tại AT&T Bell Labs của Mỹ năm

1972. Vào thời điểm đó B là một ngôn ngữ lập trình sử dụng tập lệnh của máy DEC PDP-7. Với thế hệ tiếp theo DEC PDP-11 với một tập lệnh (instruction set) phức tạp hơn thì B không thể giải quyết được cho việc viết lại game Asteroids với một tập lệnh mới. Do đó, Brian Kernighan và Dennis Ritchie đã quyết định tạo ra ngôn ngữ mới phù hợp hơn, đó là ngôn ngữ C.

Ngay sau đó vào đầu năm 1980, ngôn ngữ C đã trở nên phổ biến đối với các lập trình viên.

Sau một thời gian ngắn, C được chuẩn hóa thành một ngôn ngữ lập trình thông dụng, và Viện tiêu chuẩn quốc gia Mỹ (American National Standards Institute) lấy tên gọi là ANSI C. Năm 1990, C được tổ chức tiêu chuẩn quốc tế ISO (International Standard Organization) chính thức công nhận và có tên chuẩn là ISO 9889:1990, hay còn gọi là "C chuẩn" (Standard C) để phân biệt với C trước đó của Brian Kernighan và Dennis Ritchie với tên gọi "K&R C".

C++ là một sự mở rộng của C chuẩn với mô hình lập trình mới: *lập trình hướng đối tượng (object-oriented programming)*. Theo mô hình này, dữ liệu và các thao tác xử lý dữ liệu được đóng gói chung với nhau tạo thành một thực thể thống nhất gọi là các đối tượng (object) thông qua việc định nghĩa lớp (class) các đối tượng. Bên cạnh đó dữ liệu nhập/xuất được định nghĩa là các dòng nhập/xuất (inputoutput streams).

Hiện nay, bên cạnh phong cách lập trình mới với C++ đối với các ứng dụng từ thông thường đến phức tạp và chuyên biệt, thì C chuẩn hay ANSI C cũng không thể bỏ qua vì tính uyển chuyển và tương thích của nó trong việc lập trình điều khiển thiết bị phần cứng, chẳng hạn như micro-controller.

1.1.1.3. Môi trường lập trình C/C++

Hiện nay có nhiều môi trường lập trình C/C++, tuy nhiên khá phổ biến là môi trường lập trình trên nền Windows. Trong đó ngòai trình biên dịch (compiler), còn được tích hợp công cụ soạn thảo mã nguồn (text editor), công cụ kiểm tra lỗi (syntax checker), công cụ chạy từng bước và gỡ rối (debuger), … giúp cho người lập trình nhanh chóng tạo các ứng dụng của mình.

Dưới đây là một số môi trường lập trình C/C++ quen thuộc:

- DevCpp (4.9.9.2) là phần mềm mã nguồn mở của hãng Bloodshed Software, có thể tải miễn phí từ http://www. Bloodshed.net
- Code::Blocks 17.12 là phần mềm mã nguồn mở, có thể tải miễn phí từ http://www.codeblocks.org/
- Microsoft Visual Studio.Net 2010/2015/2017/2019 chạy trên môi trường Windows

1.1.2. Các thành phần cơ bản của ngôn ngữ lập trình C/C++

C là một ngôn ngữ lập trình cấp cao nhưng cũng có khả năng thực hiện các thao tác can thiệp vào hệ thống máy tính như ngôn ngữ Assembler. Vì vậy C có tính tổng quát và rất linh hoạt nên còn được gọi là ngôn ngữ lập trình cấp trung gian. Một số đặc điểm nổi bật của ngôn ngữ C gồm:

1.1.2.1. Bộ ký tự của C

Ngôn ngữ C được xây dựng dựa trên một bộ các ký tự, các ký tự đó được kết hợp lại thành các từ, các từ tạo thành các câu, tất cả đều tuân theo một cú pháp rất chặt chẽ.

Bộ chữ cái Latin: A, B, C, ..., Z, …, a, b, c, …, z

Bộ chữ số thập phân: 0, 1, 2, …, 9

Các ký hiệu toán tử: +, -, *, /, %, =, !, …

Các ký hiệu khác: , ; : _ ! # [] (), % ,? & . . .

Dấu cách (space) dùng đề cách các từ.

Chú ý: Khi viết chương trình ta không được dùng các ký tự nào khác ngoài các ký tự trên. Ví dụ khi viết chương trình giải phương trình bậc 2: $ax^2+bx+c=0$, ta cần dùng ký hiệu khác để thay thế ký tự Δ.

1.1.2.2. Các từ khóa của C

Từ khóa (keyword) là các từ dành riêng của C, có ngữ nghĩa đã được xác định, không được dùng nó vào các việc khác như việc đặt tên biến, tên hàm, … mới trùng với tên các từ khóa. Thông thường các từ hóa của C đều dùng chữ thường

Ví dụ :

- Khai báo dữ liệu và dẫn hướng biên dịch: ifdef, ifndef, endif, define, include, typedef, struct, union, class, int, char, unsigned, float, double, long, short, void, const
- Điều khiển: if, else, switch, case, do, while, for, return, goto, break, continuous

1.1.2.3. Tên và cách đặt tên

Trong quá trình viết chương trình, điều thường gặp là đặt tên (name) hay danh hiệu (identifier) cho các biến (variable), hằng (constant), kiểu (type), hàm (function), …

- Tên/danh hiệu của C được bắt đầu bằng một chữ cái (letter) hoặc dấu gạch dưới (underline), tiếp theo có thể là chữ cái, chữ số, dấu gạch nối
- Không được có khoảng trống trong tên/danh hiệu
- C phân biệt chữ hoa chữ thường trong một tên/danh hiệu
- Khi viết chương trình ta nên đặt tên/danh hiệu sao cho chúng nói lên được ý nghĩa của đối tượng mà chúng biểu thị, điều này giúp ta viết chương trình được dễ dàng và người khác đọc chương trình được thoải mái, dễ hiểu, dễ kiểm lỗi

Ví dụ:

Tên đặt đúng: i, j, hoten, hoten, ho_ten, x3, x_3, _x3, __INTERFACE_H__, ...

Tên đặt sai: ho ten, x 3, -x, 100, ...

1.1.2.4. Cách ghi lời giải thích trong chương trình

Khi viết chương trình nên đưa vào các câu chú thích (comment) để làm cho chương trình rỏ ràng, dễ hiểu. Điều này cũng giúp cho việc sửa đổi và nâng cấp chương trình sau này được dễ dàng. Có hai cách chú thích:

- Đặt dấu // trên 1 dòng đơn hoặc ngay sau một dòng mã nguồn
- Bao tòan bộ đoạn cần chú thích giữa hai dấu /* và */

Ví dụ 1.1: Sau đây là mã nguồn của một chương trình đơn giản cho phép người dùng nhập vào 2 số thực và in ra màn hình tổng, hiệu của chúng

```
#include <stdio.h>  //Thư viện chứa hàm printf(), scanf()
```

```
#include <conio.h>  //Thư viện chứa hàm getch()
int main()
{
      float a, b, add, sub;
      //Nhập các số a và b
      printf("a = ");
      scanf("%f", &a);
      printf("b = ");
      scanf("%f", &b);
      // Thực hiện các phép tính cộng và trừ
      add = a + b;
      sub = a - b;
      // In kết quả ra màn hình
      printf("%5.2f + %5.2f = %5.2f\n", a, b, add);
      printf("%5.2f - %5.2f = %5.2f\n", a, b, sub);
      getch(); // Dừng màn hình kết quả chờ nhấn phím bất kỳ
      return 0;
}
```

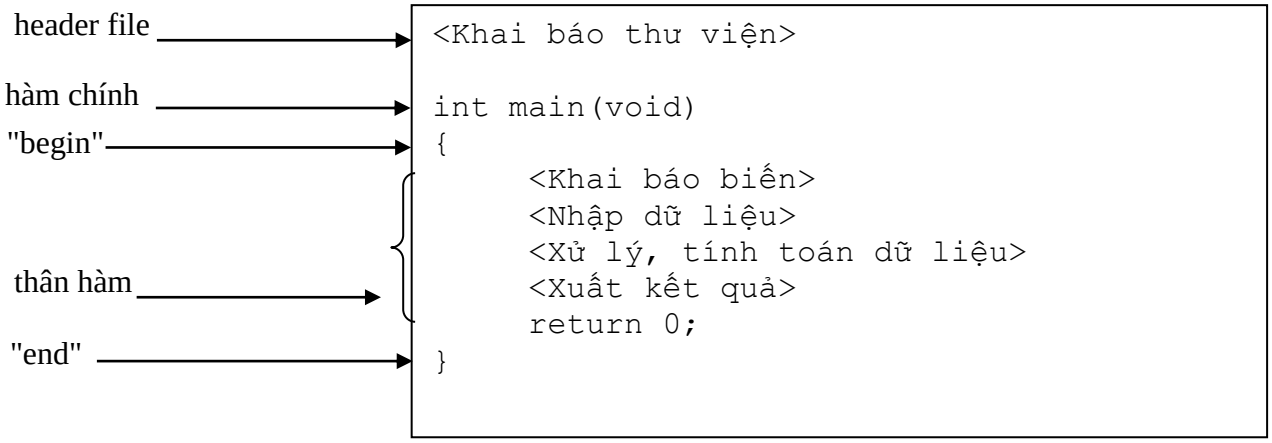
Lời chú thích có thể đặt ở bất kỳ vị trí nào trong chương trình, nếu chương trình dài phức tạp thì nên đưa chú thích vào trước mỗi đoạn lệnh để giải thích. Tuy vậy cũng không nên quá lạm dụng chú thích, nghĩa là ghi quá nhiều chú thích trong chương trình ở những nơi không cần thiết, hoặc chú thích quá rườm rà sẽ làm cho chương trình trở nên khó đọc và đôi khi có thể gây ra lỗi.

1.1.2.5. Câu lệnh và dấu chấm câu

Mỗi câu lệnh trong phần thân chương trình (body) phải được kết thúc bằng dấu chấm phẩy (;) tuy nhiên có một số lệnh đặc biệt thì không có dấu chấm phẩy (;) ở cuối câu.

1.1.12.6. Cấu trúc của chương trình

Một chương trình C/C++ đơn giản sẽ có cấu trúc như sau :



1.1.2.7. Các bước lập trình cơ bản

- Tìm hiểu và phân tích và mô hình hóa bài toán
- Xây dựng thuật toán phù hợp để giải quyết bài toán
- Cài đặt thuật toán bằng ngôn ngữ lập trình
- Biên dịch chương trình và sửa lỗi
- Chạy thử nghiệm và hòan thiện chương trình
- Tối ưu chương trình

1.1.3. CÁCH TẠO FILE MÃ NGUỒN CHƯƠNG TRÌNH C/C++ BẰNG VISUAL STUDIO.NET

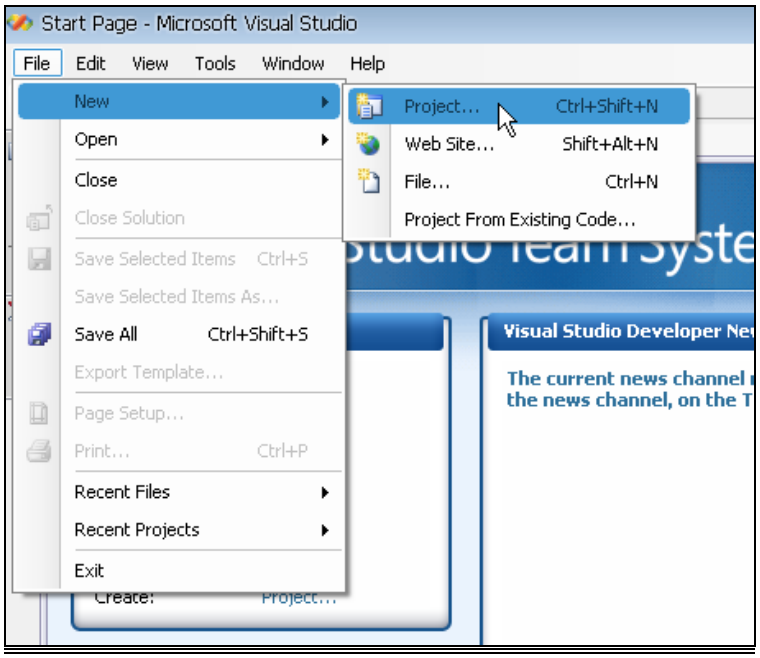
1.1.3.1. Khởi động Visual Studio.Net

Start/Programs/Microsoft Visual Studio

1.1.3.2. Tạo Project mới

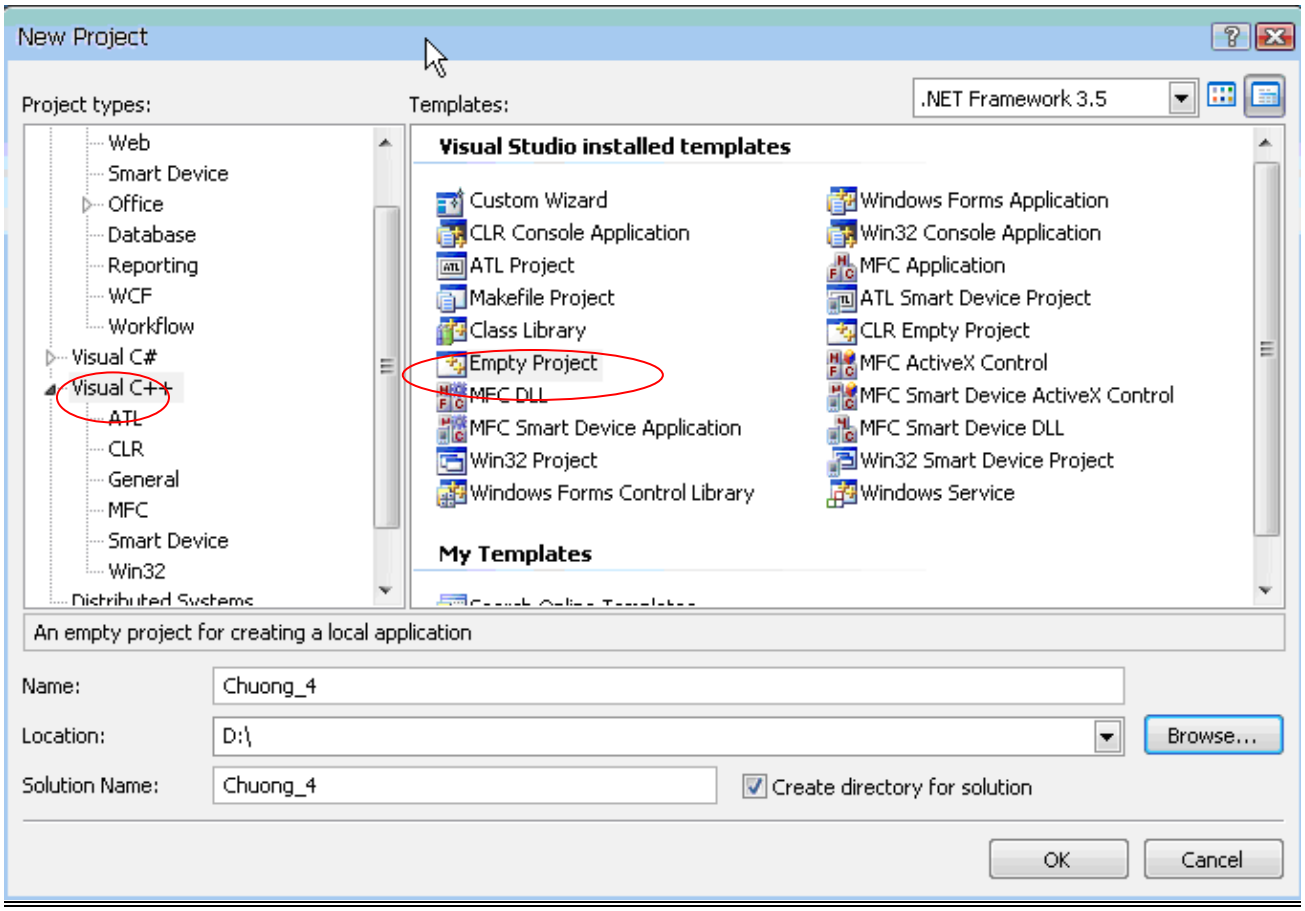
B1:

- Cách 1: Trong khung Recent Projects, chọn Create Project
- Cách 2: Vào menu File/New/Project



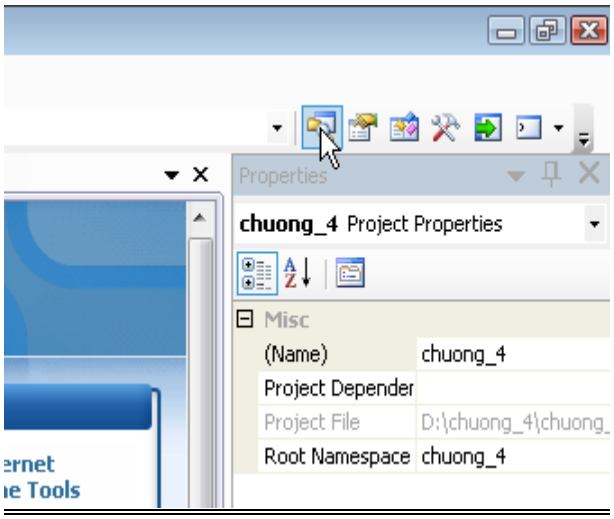
B2: Trong cửa sổ NewProject

- Trong khung Project types: Chọn Visual C++
- Trong khung Templates: Chọn Empty Project
- Trong ô Name: Gõ vào tên project
- Click Browse để chọn vị trí của Project (vị trí này sẽ được hiển thị trong mục Location)
- Gõ vào tên của Solution trong ô Solution Name (mặc định là cùng tên với project, trường hợp solution chứa nhiều project thì nên đặt lại tên của solution)
- Click OK

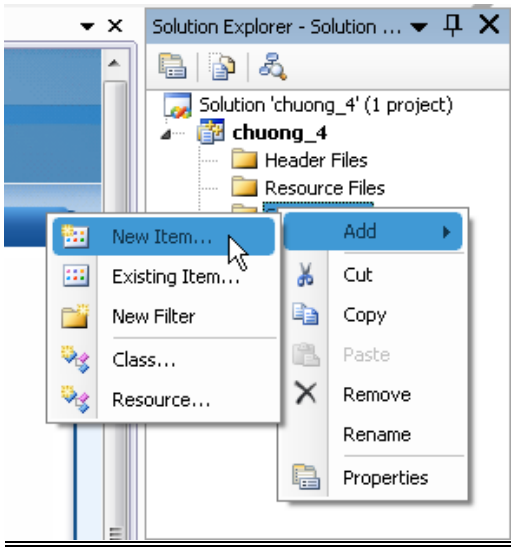


1.1.3.2. Tạo file mã nguồn (source code file)

B1: Nếu góc phải chưa có ô Solution Explorer thì click vào biểu tượng Solution Explorer trên góc phải.



B2:



1.1.4. Cách tạo file mã nguồn chương trình C/C++ bằng Dev-C++

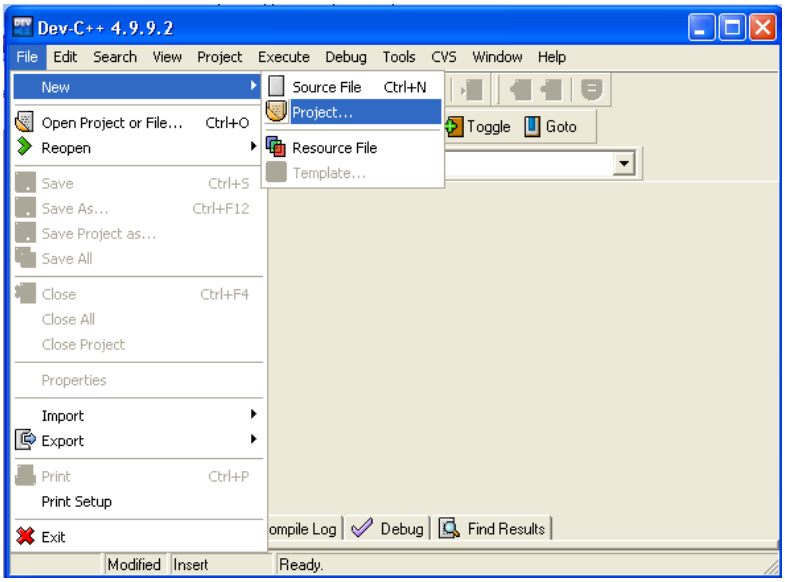
1.1.4.1. Khởi động Dev-C++

Start/Programs/Bloodshed Dev-C++/Dev-C++

Hoặc double clik vào biểu tượng Dev-C++ trên Desktop 

2. Tạo Project mới

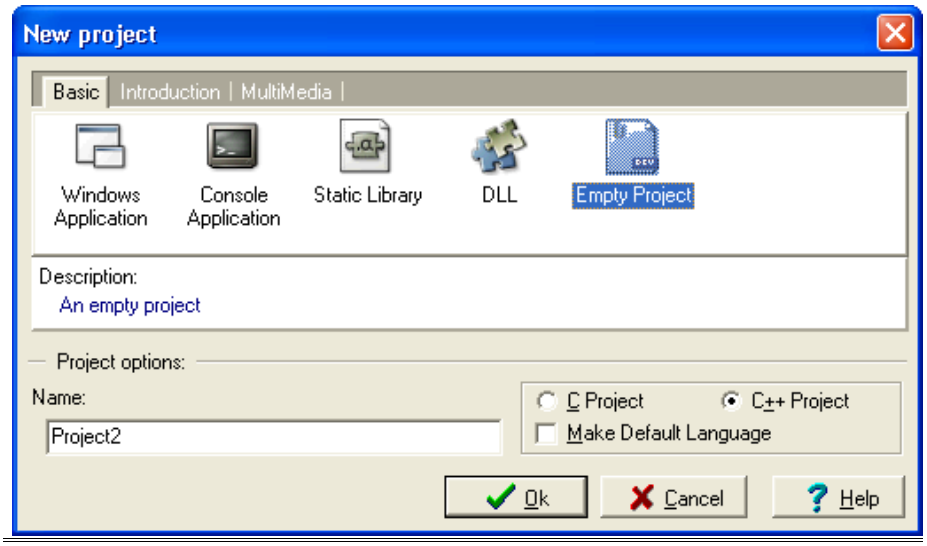
B1: Vào menu File/New/Project



B2:

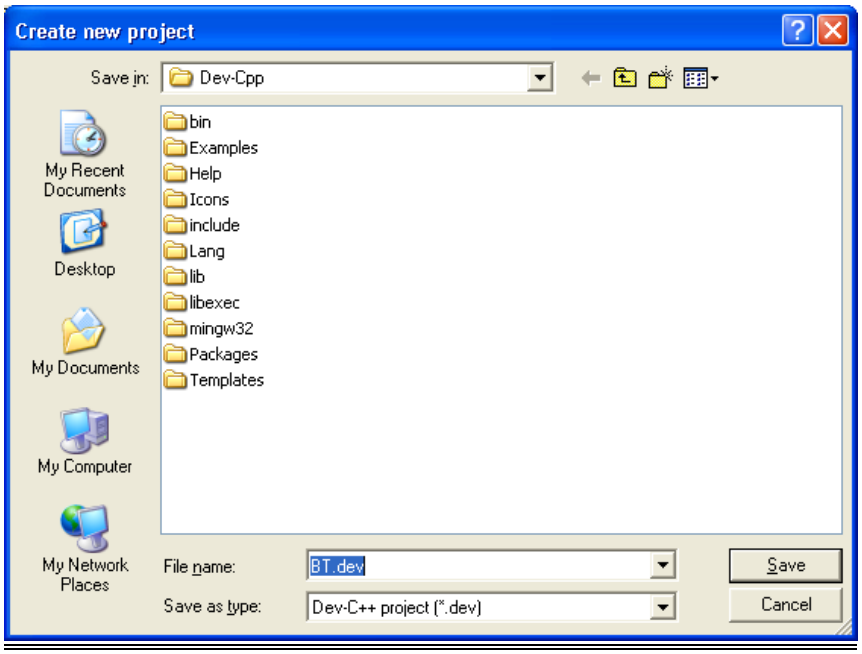
- Trong cửa sổ NewProject chọn Empty Project

- Trong ô Name: Gõ vào tên project
- Click chọn C Project hay C++ Project
- Click OK



B3:

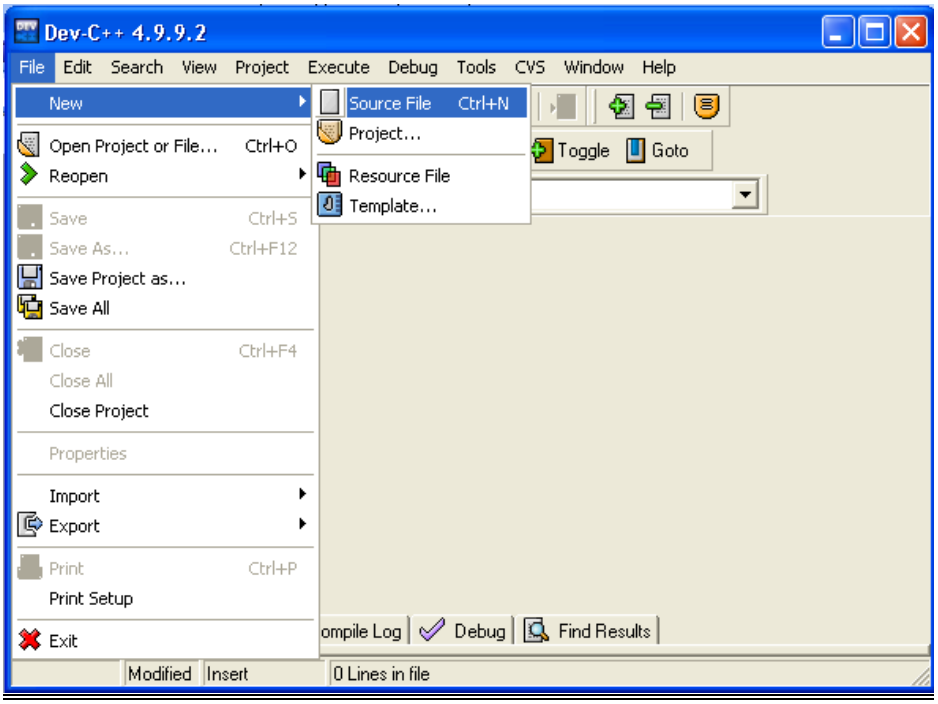
- Trong cửa sổ Create New Project chọn vị trí lưu Project



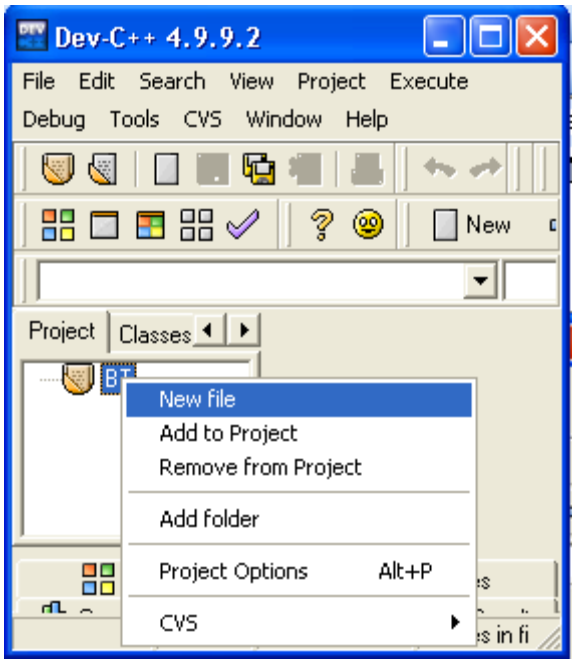
3. Tạo file mã nguồn (source code file)

- Cách 1: Vào menu File/New/Source File

Lưu ý: Ta có thể bỏ qua mục 2 và tạo luôn file mã nguồn vì Dev-C++ cho phép tạo luôn file mã nguồn mà không cần tạo project



❖ Cách 2: Click phải vào Project đã có và chọn New File



1.2. CÁC KIỂU DỮ LIỆU CƠ SỞ

1.2.1. Các kiểu dữ liệu cơ sở

Là tập hợp các giá trị mà một biến (variable), hằng (constant) hay hàm (function) thuộc kiểu đó có thể nhận được, tương ứng với chúng là các phép tóan có thể thực hiện. Tuỳ thuộc vào dữ liệu kiểu gì mà khi giá trị được ghi vào bộ nhớ để xử lý sẽ chiếm một kích thước nhất định (tính bằng byte). Sau đây là một số kiểu dữ liệu đơn giản trong C.

1.2.1.1. Kiểu ký tự (character)

Mỗi ký tự là một ký hiệu xác định được định nghĩa trong bảng mã chuẩn ASCII (xem phụ lục). Mỗi ký tự có độ dài 1 byte và tương ứng với một giá trị số là mã ASCII.

Mỗi ký tự được bao bằng dấu nháy đơn, ví dụ ‘a’, ‘A’,’1’,’2’,..và có định dạng là %c.

Ngoài ra, cũng có thể dùng kiểu ký tự để biểu diễn số nguyên:

Kiểu	Số byte	Minimum	Maximum	Định dạng
char	1	-128	127	%d, %i
unsigned char	1	0	255	%d, %i

Các hằng lưu trữ giá trị lớn nhất , nhỏ nhất của các kiểu dữ liệu (CHAR_MIN, CHAR_MAX,…) được định nghĩa trong thư viện <limits.h>.

1.2.1.2. Kiểu số nguyên (integer number)

Kiểu	Số byte	Minimum	Maximum	Định dạng
short	2	-32.768	32.767	%hi, %hd
unsigned short	2	0	65.535	%hu
int	4	-2.147.483.648	2.147.483.647	%i, %d
unsigned int	4	0	4.294.967.296	%u
long	4	-2.147.483.648	2.147.483.647	%li, %ld

unsigned long	4	0	4.294.967.245	%lu
long long	8	-9.223.372.036.854.775.808	9223372036854775807	%lld
unsigned long long	8	0	18446744073709551616	%llu

1.2.1.3. Kiểu số thực (real number)

Bảng dưới đây liệt kê các kiểu dữ liệu cơ sở là số thực.

Kiểu	Số byte	Minimum	Maximum	Định dạng
float	4	$-3.4x10^{38}$	$3.4x10^{38}$	%f, %e, %g
double	8	$-1.7x10^{308}$	$1.7x10^{308}$	%lf, %le, %lg
long double	10	$-3.4x10^{4932}$	$3.4x10^{4932}$	%Lf, %Le, %Lg

1.2.2. Khái niệm về hằng và biến

1.2.2.1. Hằng (Constant)

Trong C, một hằng có thể có một trong các kiểu dữ liệu ký tự, số nguyên, số thực như đã nêu trên. Giá trị của hằng không bị (và cũng không được) thay đổi trong phạm vi mà hằng được định nghĩa.

Hằng có thể được định nghĩa thông qua #define, phép gán trị (assignment), tham số của hàm bằng từ khóa const.

Với từ khóa const cho phép tạo một hằng với kiểu dữ liệu xác định. Do đó, hằng sẽ chỉ có thể là rvalue (right-side value), nghĩa là chỉ có thể đặt bên phải phép gán =.

Khai báo hằng:

```
#define <Tên hằng> <giá trị> //Không có ";" ở cuối câu lệnh
Hay
conts <kiểu dữ liệu> <Tên hằng> = < giá trị>;
```

Ví dụ 1.2:

```
#define MAX 100          // Hằng MAX có giá trị là 100
#define CZERO '\0'       // Hằng CZERO có giá trị là ký tự 0
#define PI 3.14159       // Hằng PI có giá trị là 3.14159
const float PI = 3.14159;// Hằng PI có kiểu float và được gán trị thực 3.14159
```

❖ Đối với hằng số nguyên, có 3 cách thể hiện theo hệ thống số:

- Thập phân (decimal): Viết theo dạng thông thường
- Bát phân (octal): Được bắt đầu bằng số 0
- Thập lục phân (hexadecimal): Được bắt đầu bằng 0x hay 0X

Chẳng hạn với giá trị nguyên 16 ta có thể viết: 16 hoặc 020 hoặc 0x10

Một số hằng điều khiển trong C/C++:

Hằng (kiểu char)	**Ý nghĩa**	**Ví dụ**
\a	Beep, phát tiếng beep	`printf("bell\a");`
\b	backspace, ký tự backspace	`printf("backspace\b");`
\f	formfeed, kéo giấy trên máy in	`printf(printer,"\f");`
\n	newline, về đầu dòng mới	`printf("new line\n");`
\r	carriage return, về đầu dòng hiện thời	`printf("carriage return\r");`
\t	horizontal tab, tab ngang (theo giá trị tab hiện thời)	`printf("horizontal tab\t");`
\0	null character, ký tự null	`str[0]='\0';`
\\	backslash, Dấu "\"	`printf("backslash\\");`
\'	single quote, nháy đơn	`printf("\'single quote\'");`

\"	double quote, nháy kép	printf("\"double quote\"");

2.2.2.2. Biến (Variable)

Biến là một tên gọi đại diện cho một địa chỉ xác định trong bộ nhớ. Thay vì ta truy cập vào địa chỉ của 1 ô nhớ, ta dùng tên gọi thay thế cho ô nhớ đó, điều này giúp cho việc viết chương trình đơn giản và hiệu quả hơn rất nhiều.

- Tên biến được đặt theo qui ước đã trình bày trong Bài 12
- Biến được khai báo phải thuộc một kiểu dữ liệu xác định (xem Bài 12)
- Phân biệt chữ hoa và chữ thường

Khai báo biến:

```
<Kiểu dữ liệu>  <Tên biến>;
<Kiểu dữ liệu>  <Danh sách tên biến>;
```

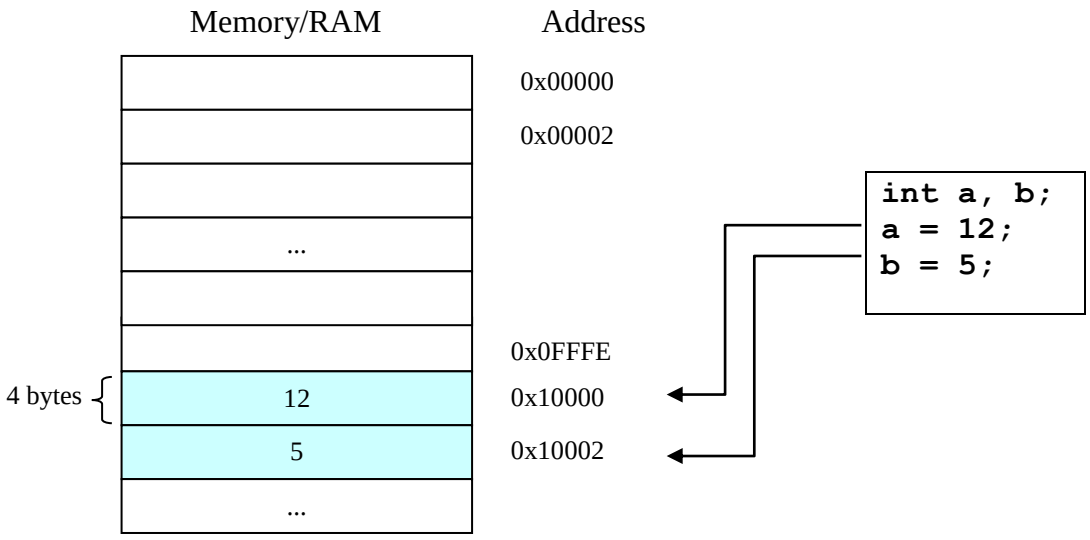
<Kiểu dữ liệu> : Kiểu dữ liệu (data type) là kiểu cơ sở hoặc kiểu dữ liệu mới do người dùng định nghĩa

<Danh sách tên biến>: Danh sách các tên biến, cách nhau bằng dấu phẩy

Ví dụ 1.3:

```
float x,y;
int n;
char ch;
int i=10;            // Biến int i được khai báo và gán giá trị ban đầu  bằng 10
char c='C';          // Biến char c được khai báo và gán giá trị ban đầu  bằng 'C'
```

Ví dụ 1.4: Hình ảnh biến được lưu trữ trong bộ nhớ



1.3. BIỂU THỨC – TÓAN TỬ – CÂU LỆNH

1.3.1. Biểu thức và tóan tử

1.3.1.1. Khái niệm về biểu thức

Biểu thức là một công thức tính toán bao gồm các phép toán, hằng, biến, hàm, và các dấu ngoặc.

Thứ tự ưu tiên: Khi tính giá trị của một biểu thức, ngôn ngữ C quy ước thứ tự ưu tiên của các phép toán từ cao xuống thấp như sau:

- Biểu thức trong ngoặc, phép gọi hàm,
- !
- / , *, \, %, &&
- + , -, ||
- <, >, ==, <= , >=, !=

Ví dụ 1.5:

Nếu a=3, b=5, c= 2 thì:

Biểu thức : a + b * c có giá trị là 13

Biểu thức: (b * b – 4 * a * c) / (2 * a) là 1/6

Biểu thức: (a % 2) != 0 là 1

1.3.1.2. Tóan tử

❖ **Toán tử số học**

Tóan tử	Ý nghĩa	Ví dụ
		int a = 5, b = 2;
		float x = 11;
		char c = 'm';

+	Cộng	a+b = 7 x+b = 13 c+1 = 110 char (c+1)='n'
-	Trừ	a-b = 3 x-b = 9 char (c - 2) = 'k'
*	Nhân	a*b = 10 x*b = 22 c*2 = 218 char (c*2)= '⌈'
/	Chia	a / b = 2 11/b = 5.5 c/2 = 54 char (c/2)='6'
%	Chia lấy dư	a%b = 1 c%b = 1 x%b → lỗi

❖ **Toán tử quan hệ (so sánh)**

Bao gồm các tóan tử so sánh quan hệ 2 đối tượng nào đó, kết quả của phép so sánh là một giá trị logic true (=1) hoặc false (=0)

Tóan tử	**Ý nghĩa**	**Ví dụ** a=4, b=1
<	So sánh nhỏ hơn	a<b có kết quả false
<=	So sánh nhỏ hơn hoặc bằng	a<=b có kết quả false
>	So sánh lớn hơn	a>b có kết quả true
>=	So sánh lớn hơn hoặc bằng	a>=b có kết quả true
==	So sánh bằng	a==b có kết quả false
!=	So sánh không bằng	a!=b có kết quả true

❖ **Toán tử logic:**

Gồm các tóan tử dùng trong các biểu thức logic, trong đó các tóan hạng là các biến, giá trị hay biểu thức logic. Kết quả thực hiện phép tóan logic là một giá trị logic true hoặc false.

Tóan tử	**Ý nghĩa**	**Ví dụ** a=true, b=false
!	NOT/Phủ định/Không	!a có kết quả false
&&	AND/Và	a&&b có kết quả false
||	OR/Hoặc	a||b có kết quả true

Bảng chân trị (truth table)

&&	False	True
False	False	False
True	False	True

||	False	True
False	False	True
True	True	True

❖ **Toán tử Bitwise**

Khi xử lý trên từng bit dữ liệu, thực tế là việc xử lý các thanh ghi (register) khi lập trình hardware, người lập trình đến các công cụ hay tóan tử cho phép thao tác trên từng bit của byte dữ liệu. Các tóan tử bitwise được liệt kê trong bảng dưới đây:

Tóan tử bitwise	**Ý nghĩa**	**Ví dụ** a=12
&	Bitwise AND	b=(a&5); cho kết quả b=4
|	Bitwise inclusive OR	b=(a|5); cho kết quả b=13
^	Bitwise excluse OR (XOR)	b=(a^5); cho kết quả b=9
~	Bù 1 (one's compliment)	b=~a; cho kết quả b=3
<<	Dịch sang trái n bit, tương ứng với phép nhân cho 2^n	b=(a<<2); sẽ tương ứng b = 12*2^2 = 12*4 = 48
>>	Dịch sang phải n bit, tương ứng với phép chia cho 2^n	b=(a>>2); sẽ tương ứng với b = 12/2^2 = 12/4 = 3

AND

&	0	1
0	0	0
1	0	1

OR

|	0	1
0	0	1
1	1	1

XOR

^	0	1
0	0	1
1	1	0

❖ **Toán tử tăng giảm**

Gồm các tóan tử thực hiện trên các biến hay biểu thức có giá trị nguyên như char, int, short, long. Các tóan tử này thực hiện việc tăng giảm giá trị của biến hay biểu thức 1 đơn vị.

Tóan tử	Ý nghĩa	Ví dụ a=5
++	Tăng 1 đơn vị	++a có kết quả 6
--	Giảm 1 đơn vị	--a có kết quả 4

Lưu ý: ++a và a++ đều tăng giá trị biến a lên 1 đơn vị, tuy nhiên có sự khác biệt. Chúng ta hãy xem 2 ví dụ đơn giản sau:

Version 1	Version 2
`int a=5;` `printf("a = %d", ++a);`	`int a=5;` `printf("a = %d", a++);`
Kết quả: a = 6	Kết quả: a = 5

Cũng tương tự như vậy đối với --a và a--

- **Toán tử điều kiện**

```
<Điều kiện>  ? <Biểu thức 1>: <Biểu thức 2>
```

<Điều kiện> : Điều kiện (condition) là biểu thức cho kết quả logic true hoặc false, trong đó sử dụng các tóan tử so sánh.

<Biểu thức 1> : Biểu thức (expression), biến, hằng, hàm bất kỳ tương ứng với *<Điều kiện>* có trị logic TRUE

<Biểu thức 2> : Biểu thức, biến, hằng, hàm bất kỳ tương ứng với *<Điều kiện>* có trị logic FALSE

Ví dụ 1.6: Xác định số lớn hơn trong 2 số a và b

```
int a = -1, b = 1;
max = (a>b) ? a : b;
```

Kết quả: max có trị là trị của biến b.

❖ **Các toán tử khác**

Ngòai các tóan tử đã trình bày ở các mục trên, C/C++ còn hỗ trợ nhiều tóan tử khác với mục đích tạo thuận lợi và linh hoạt cho việc viết chương trình.

Tóan tử	Ý nghĩa	Ví dụ
`()`	Nhóm biểu thức con	`(a*b+c)/(a*d)`
`[]`	Trích phần tử	`a[i]` `//a là một mảng,i là chỉ mục`
`.`	Thuộc tính của struct/class	`student.name` `student.age`
`->`	Con trỏ thuộc tính của struct/class	`Student->name` `student->age`
`&`	Địa chỉ của	`&a //địa chỉ của biến a`

`*`	Con trỏ/giá trị tại địa chỉ mà biến con trỏ lưu	`int *a; //a là biến con trỏ`
		`*a //giá trị tại địa chỉ a lưu`

`=`	Gán giá trị	`a=12; ghi giá trị hằng 12 vào biến a`

`+=`
`-=`
`*=`
`/=`
`%=`
	Các tóan tử số học ghép	`a+=2; //a=a+2;`
		`a-=3; //a=a-3;`
		`a*=5; //a=a*5;`
		`a/=2; //a=a/2;`
		`a%=2; //a=a%2;`

1.3.2. Lệnh và khối lệnh

– Trong C/C++ mỗi lệnh đơn (single statement) là một khai báo hay phát biểu được ghi trên 1 dòng đơn và kết thúc bằng dấu chấm phẩy (;).

- Trong khi đó, một khối lệnh (statement block/compound statement) là tập các lệnh đơn và được bao trong cặp ngoặc nhọn ({}) .

Ví dụ 1.6:

```
int count=5; // Lệnh đơn
printf("Cac so tu 5 den 0: \n"); // Lệnh đơn
while(count>=0)
{
   printf("%d, ", count); // Lệnh đơn
   count--; // Lệnh đơn
}
```

Khối lệnh

1.3.3. Chuyển đổi kiểu dữ liệu

Trong quá trình viết chương trình, ta phải xử lý trên nhiều kiểu dữ liệu (data type) khác nhau, và trong nhiều trường hợp việc chuyển đổi các giá trị từ kiểu dữ liệu này sang kiểu dữ liệu khác là rất cần thiết. C/C++ thực hiện phép chuyển kiểu dữ liệu thông qua tóan tử type cast bằng cách viết tên kiểu dữ liệu đích trong dấu ngoặc trước tên biến, hằng, hàm hay biểu thức nguồn.

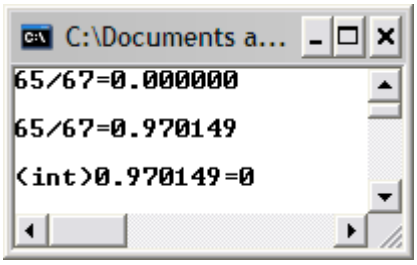
```
(<Kiểu dữ liệu đích>) <Biểu thức>
```

Ví dụ 1.7:

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

```
int main()
{
     int a=65, b=67, c;
     float x;
     x=a/b;
     printf("%d/%d=%f\n\n",a,b,x);
     x=(float)a/b; //Đổi a từ kiểu int sang kiểu float
     printf("%d/%d=%f\n\n",a,b,x);
     c=(int)x;     //c là phần nguyên của x
     printf("(int)%f=%d",x,c);
     getch();
     return 0 ;
}
```

Kết quả chạy chương trình:



1.2.4. CÁC HÀM CÓ SẴN TRONG C/C++

C/C++ tích hợp sẵn một số lượng lớn các hàm (predefined functions) để thực hiện các tác vụ khác nhau, các hàm được phân chia theo từng nhóm xử lý khác nhau và được định nghĩa và phân phối trong các tập tin header đi kèm với đuôi ".h". Muốn dùng hàm nào ta cần dùng dẫn hướng (directive) #include <header_file> ghi rõ tên tập tin header có định nghĩa hàm đó.

Các tập tin header thường dùng đến gồm stdio.h, conio.h, stdlib.h, string.h, math.h và time.h.

Để biết một hàm có sẵn được định nghĩa trong tập tin header nào hay được khai báo (prototye) ra sao, ta có thể dùng chức năng Help của chương trình dịch.

1.2.4.1. Tập tin header

Các tập tin header trong C/C++ là một thành phần rất quan trọng không thể tách rời từ giai đoạn thiết kế chương trình đến giai đoạn biên dịch và liên kết vđể tạo ra tập tin đối tượng (.obj) và tập tin nhị phân thi hành được (.exe, .dll). Các mô tả dữ liệu, các chỉ dẫn biên dịch, các khai báo prototype hàm thường được bố trí trong các tập tin header.

Khi muốn sử dụng các hằng, biến, hàm hay các mô tả khác trong một tập tin header, ta cần viết dẫn hướng đến tập tin header đó tại đầu chương trình trước cài đặt hàm main() và khai báo dữ liệu.

Đối với các tập tin header được phân phối theo phần mềm ngôn ngữ ta viết dẫn hướng như sau (lưu ý cuối dòng không có dấu chấm phẩy):

```
#include <header_file>
```

Đối với tập tin header do người dùng viết thường được đặt trong thư mục làm việc hiện thời, nên ta cần viết:

```
#include "header_file"
```

Nếu tập tin header được đặt tại một vị trí khác, ta cần xác định đường dẫn (path) đến tập tin header đó:

```
#include "path\header_file"
```

Ví dụ 1.8:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mymath.h"
```

1.2.4. 2. Khai báo hàm và cách gọi hàm

Hàm (function) là một thành phần cực kỳ quan trọng trong C/C++. Hàm thực chất là một chương trình con (routine) cho phép thực hiện một tác vụ cụ thể nào đó. Hàm có thể trả về giá trị thuộc kiểu xác định hoặc không trả về giá trị (đối với hàm kiểu void).

Một hàm được xác định thông qua các yếu tố sau:

- Tên hàm (function name)
- Danh sách tham số (parameters) và kiểu của tham số (data type)
- Tham số vào (input parameters) và tham số ra (output parameters)
- Kiểu dữ liệu trả về của hàm (return data type)

Để xác định các yếu tố trên đối vời một hàm cho sẵn của C/C++ ta dựa vào phần khai báo prototype của hàm .

Prototype của hàm có dạng sau:

```
<Kiểu DL trả về> <Tên hàm> ([<DS tham số & Kiểu dữ liệu>});
```

Ví dụ 1.9:

Hàm `sqrt ()` để lấy căn bậc 2 của một số có prototye:

```
float sqrt(float x);  //<math.h>
```

Hàm `pow()` để tính lũy thừa có prototye:

```
float pow(float base,float exponent); //<math.h>
```

Hàm `getch()` để nhận một ký tự được nhấn từ bàn phím có prototye:

```
int getch(void); //<conio.h>
```

Hàm `swap()` để hoán vị giá trị của 2 biến có prototye:

```
void swap(int &a, int &b); //<iostream>
```

Cách gọi hàm:

```
<Tên hàm> (Tên tham số>);
//Nếu là hàm trả về giá trị thì lời gọi hàm thường được để trong biểu thức hay câu lệnh xuất
```

Ví dụ 1.10:

Đoạn chương trình sau tính 2 nghiệm x1, x2 của phương trình : $3x^2+7x+4=0$ sau đó hoán vị 2 nghiệm này.

```
…
float a=3, b=7, c=4, delta, x1, x2;
delta=pow(b,2)-4*a*c; //delta=b*b-4*a*c;
x1=(-b- sqrt(delta))/2*a;
x1=(-b+ sqrt(delta))/2*a;
swap (x1,x2);
…
```

CHƯƠNG 2: CÁC HÀM NHẬP XUẤT

2.1. HÀM printf() & scanf()

2.1.1. Hàm `printf()`

2.1.1.1. Cú pháp

```
int printf("<Chuỗi định dạng>",<Danh sách tham số>);
//#include <stdio.h>
```

Hàm *printf()* xuất dữ liệu theo định dạng (format) trong *<chuỗi định dạng>* ra thiết bị xuất chuẩn (STDOUT), thông thường là màn hình.

Hàm *printf()* sẽ trả về số ký tự được xuất nếu thành công, ngược lại trả về một số âm.

<Chuỗi định dạng> : xác định các ký tự sẽ được in ra màn hình và định dạng của dữ liệu có dạng sau:

```
%[flag][width][.prec][hlL]control_character
```

<Danh sách tham số>: Được phân cách bằng dấu phẩy.

Bảng dưới đây liệt kê một số định dạng cho printf().

Định dạng (format)		Ý nghĩa
Mã định dạng (control character)	%c	Ký tự
	%d	Số nguyên có dấu (signed integer) dạng decimal
	%i	Số nguyên có dấu (signed integer) dạng decimal
	%e	Số dạng khoa học (scientific notation) với chữ thường "e" (exponent)
	%E	Số dạng khoa học (scientific notation) với chữ hoa "E" (exponent)
	%f	Số thực với dấu chấm động (floating point)
	%g	Dạng ngắn của %e hay %f
	%G	Dạng ngắn của %E hay %f
	%o	Số bát phân (octal)
	%s	Chuỗi ký tự (string)
	%u	Số nguyên không dấu (unsigned integer)
	%x	Số thập lục phân không dấu (unsigned hexadecimal) với chữ thường
	%X	Số thập lục phân không dấu (unsigned hexadecimal) với chữ hoa
	%p	Con trỏ
	%%	Dấu %
Flag	-	Canh trái (left justify)
	+	Dấu + hoặc – trước giá trị
	Space	Số dương với khoảng trắng trước giá trị
	0	Điền số 0 cho giá trị số
	#	Ký hiệu hệ thống đếm trước giá trị số: 0 với octal, 0x với hexadecimal, dấu chấm thập phân với số thực
Width		Kích thước tối thiểu của giá trị xuất
Prec		Số số lẻ sau dấu chấm thập phân
h		Thể hiện số short int
l		Thể hiện số long int (ld) hoặc double (lf)
L		Thể hiện số long double (Lf)

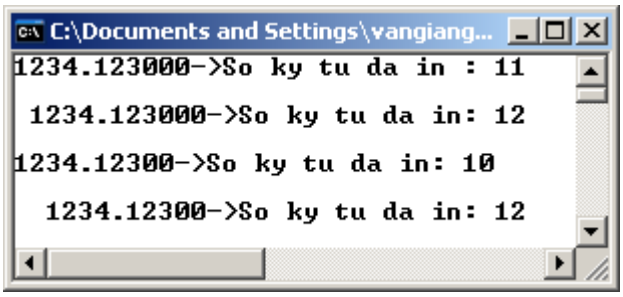
2.1.1.2. Ví dụ

Ví dụ 2.1:

Chương trình sau in ra màn hình một số kiểu định dạng của một giá trị có kiểu float

```
#include <stdio.h>
#include <conio.h>
int main()
{
    double x=1234.123;
    int t;
    t= printf("%lf",x);
    printf("->So ky tu da in : %d\n\n", t);

     t= printf("%12lf",x); //t= printf("%*lf", 12,x);
    printf("->So ky tu da in: %d\n\n", t);
    t= printf("%.5lf",x); //t= printf("%.*lf", 5,x);
    printf("->So ky tu da in: %d\n\n", t);

    t= printf("%12.5lf",x); //t= printf("%*.*lf", 12, 5,x);
    printf("->So ky tu da in: %d\n\n", t);

    getch();
    return 0;
}
```

Kết quả khi chạy chương trình:



2.1.2. Hàm `scanf()`

2.2.2.1. Cú pháp

```
int scanf(("<Chuỗi định dạng>",<Địa chỉ các tham số>);
//#include <stdio.h>
```

Hàm `scanf()` nhận dữ liệu từ thiết bị nhập chuẩn (STDIN), thông thường là bàn phím theo *<Chuỗi định dạng>* vào bộ đệm (buffer).

Hàm scanf() trả về số lượng tham số đã được nhận, trong trường hợp có lỗi nhập trước khi một tham số được nhận thành công thì trả về hằng EOF (= - 1)

<Chuỗi định dạng> : bao gồm ký tự điều khiển (control character), ký tự trắng (whitespace) và ký tự khác trắng (non-whitesapce). Các định dạng được ghi trong cặp ngoặc kép (" ").

Bảng dưới đây liệt kê một số ký tự điều khiển cho scanf().

Ký tự điều khiển	Ý nghĩa
%c	Ký tự đơn
%d	Số nguyên decimal (decimal integer)
%i	Số nguyên (integer)
%e, %f, %g	Số có dấu chấm động (floating point)
%o	Số bát phân (octal)

%s	Chuỗi ký tự (string)
%x	Số thập olục phân (hexadecimal)
%p	Con trỏ (pointer)
%n	Số lượng các ký tự
%u	Số nguyên không dấu (unsigned integer)
%[]	Tập các ký tự (set of characters)
%%	Dấu phần trăm (percent sign)

Ký tự khỏang trắng (whitespace) được định nghĩa gồm: khỏang trống (blank), tab ngang '\t' (horizontal tab), tab đứng '\v' (vertical tab), về đầu dòng '\r' (carriage return), dòng mới '\n' (newline) hay kéo trang '\f' (form feed). Các ký tự trắng sẽ được bỏ qua khi scanf() đọc dữ liệu từ thiết bị nhập.

Ký tự không trắng (non-whitespace) bao gồm các ký tự không thuộc tập các ký tự trắng đã nêu trên.

Bảng dưới đây liệt kê một số định dạng cho scanf().

Định dạng	Ý nghĩa
size	Kích thước tối đa của vùng nhập (input field)
H	Giá trị nhập được chứa theo dạng short int
L	Giá trị nhập được chứa theo dạng long int hoặc double
L	Giá trị nhập được chứa theo dạng long double
*	Vùng nhập bị bỏ qua và không được ghi

2.1.2.2. Ví dụ:

Ví dụ 2.2: Minh họa cách dùng hàm scanf() để cho phép nhập giá trị cho từng biến.

```
...
```

```
int n;
long m;
float x;
long double y;
printf(“Nhap mot so nguyen kieu int:”);
scanf("%d", &n);
printf(“Nhap mot so nguyen kieu long:”);
scanf("%ld", &m);
printf(“Nhap mot so thuc kieu float:”);
scanf("%f", &x);
printf(“Nhập mot so thuc kieu long double:”);
scanf("%Lf", &y);
```

Lưu ý 1:

Không nên viết : `scanf(" %d", &n);`

Vì người dùng sẽ phải nhấn phím spacebar để nhập khoảng trắng trước khi nhập số nguyên.

Ví dụ 2.3: Minh họa cách dùng hàm scanf() để cho phép nhập cùng lúc giá trị cho nhiều biến khác nhau và kiểm tra giá trị trả về của hàm scanf().

```
…
int n;
float x;
printf(“Nhap mot so nguyen va mot so thuc:”);
if (scanf("%d%f",&n,&x)==2)
    printf(“Nhap thanh cong !);
```

Lưu ý 2:

Hàm `scanf()` có khả năng kiểm tra sự hợp lệ của các ký tự thuộc một tập xác định được ghi trong cặp dấu ngoặc vuông [...]. Những ký tự ghi trong cặp dấu ngoặc này được xem là hợp lệ.

Ví dụ 2.4:

scanf("%[A-Za-z]", myString);

sẽ chỉ chấp nhận các chữ cái A-Z và a-z, bất kỳ 1 ký tự khác không thuộc tập này sẽ bị bỏ qua và không được ghi vào chuỗi myString. Nếu muốn cho phép có khỏang trống trong chuỗi nhập, ta ghi thêm khỏang trống vào tập hợp.

scanf("%[A-Za-z]", myString);

Ngòai ra nếu ký tự đầu tiên trong cặp ngoặc vuông là dấu '^' thì scanf() sẽ chỉ chấp nhận những ký tự không thuộc tập ký tự ghi trong cặp ngoặc này. Chẳng hạn

scanf("%[^\n]", myString);

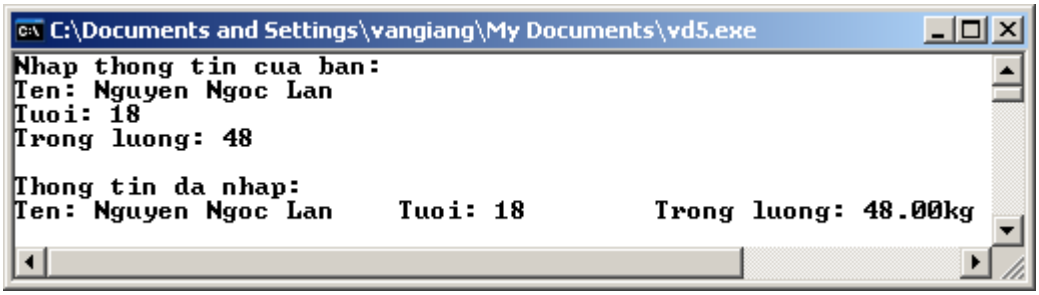
sẽ chấp nhận tất cả các ký tự, ngọai trừ ký tự newline '\n'.

Ví dụ 2.5: Chương trình sau cho phép nhập vào thông tin của người dùng gồm tên, tuổi và trọng lượng, sau đó in ra trên 1 dòng các thông tin đó.

```
#include <stdio.h>
#include <conio.h>
int main()
{
     char name[50];//Chuỗi chứa tên, tối đa 50 ký tự
     int age;       //Tuổi
     float weight; //Trọng lượng
     printf("Nhap thong tin cua ban:\n");
     printf("Ten: ");
     scanf("%[A-Za-z ]", name);
     //Chỉ nhận chữ cái,khoảng trắng
```

```
    printf("Tuoi: ");
    scanf("%d", &age);
    printf("Trong luong: ");
    scanf("%f", &weight);
    printf("\nThong tin da nhap:\n");
    printf("Ten: %s\tTuoi: %d\tTrong luong: %5.2fkg",
    name, age, weight);
    getch();
    return 0;
}
```

Kết quả:

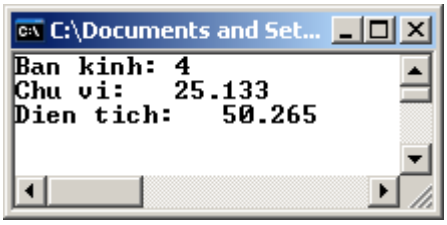


Ví dụ 2.6: Chương trình sau cho phép nhập vào bán kính hình tròn, tính và in ra màn hình chu vi và diện tích hình tròn đó.

```
#include <stdio.h>
#include <conio.h>
#define PI 3.14159
int main()
{
    float r, c, a;
    printf("Ban kinh: ");
```

```
      scanf("%f", &r);
      if(r>0)
      {
            c = 2 * PI * r;
            a = PI * r * r;
            printf("Chu vi: %8.3f\n", c);
            printf("Dien tich: %8.3f", a);
      }
      else
            printf("Gia tri nhap khong hop le");
      getch();
      return 0;
}
```

Kết quả:



2.2. HÀM getche() và getch()

2.2.1. Hàm `getche()`

2.2.1.1. Cú pháp

```
int getche();                          //#include <conio.h>
```

Dùng để đọc một ký tự từ bàn phím và trả về ký tự đó, đồng thời ký tự nhập sẽ xuất hiện trên màn hình tại vị trí con trỏ (cursor).

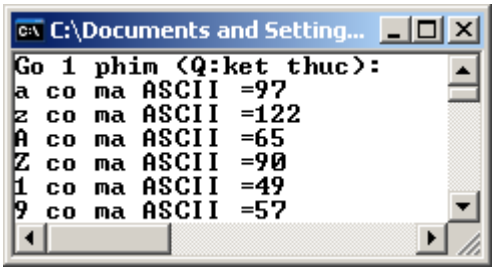
2.2.1.2. Ví dụ

Ví dụ 2.7:

Chương trình đơn giản dưới đây cho phép lặp đi lặp lại việc nhận 1 phím từ bàn phím và in ra màn hình ký tự tương ứng và mã ASCII của ký tự đó (được ghi trong ngoặc). Chương trình được kết thúc khi người dùng gõ phím 'q' hoặc 'Q'.

```
#include <stdio.h>
#include <conio.h>
int main()
{
     char key;
     printf("Go 1 phim (Q:ket thuc):\n");
     while(1)
     {
          key=getche();
          printf("/%d ", key);
          if(key=='q' || key=='Q')
               break;
     }
     return 0;
}
```

Kết quả chạy chương trình:



2.2.2. Hàm `getch()`

2.2.2.1. Cú pháp

```
int getch(void);                              //          #include
<conio.h>
```

Tương tự như hàm getche(), tuy nhiên hàm getch() không hiện ký tự đã nhập lên màn hình. Hàm này rất hữu dụng khi cần kiểm tra xem phím nào trên bàn phím được nhấn để quyết định thực hiện tác vụ tiếp theo.

Hàm `getch()` thường được để ở cuối chương trình để giúp dừng màn hình hiển thị kết quả chạy chương trình, chờ đến khi người dùng nhấn một phím mới kết thúc chương trình, khi đó màn hình hiển thị kết quả sẽ bị đóng.

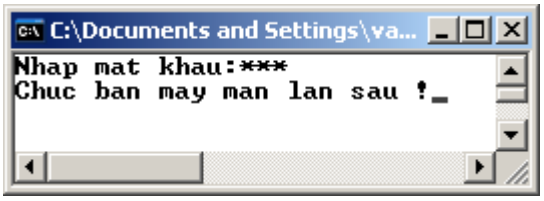
2.2.2.2. Ví dụ

Ví dụ 2.8: Chương trình kiểm tra mật khẩu đơn giản, cho phép người dùng nhập mật khẩu (không hiển thi ký tự nhập lên màn hình mà thay bằng dấu *) sau đó kiểm tra xem mật khẩu có đúng như yêu cầu không và xuất thông báo.

```
#include <stdio.h>
#include <conio.h>
int main()
{
     char a,b,c;
     printf("Nhap mat khau:");
     a=getch();printf("*");
     b=getch();printf("*");
     c=getch();printf("*");
     if (a=='L'&&b=='T'&&c=='T')
          printf("\nChuc mung ban !");
     else printf("\nChuc ban may man lan sau !");
     getch();
     return 0;
```

```
}
```

Kết quả khi chạy chương trình:



2.3. NHẬP XUẤT DỮ LIỆU DÙNG cin & cout

2.3.1. Nhập dữ liệu dùng cin

cin là một đối tượng thuộc dòng nhập (input stream object) thuộc lớp ***iostream*** (input/output stream) được định nghĩa trong C/C++.

cin cho phép lưu giá trị thuộc kiểu dữ liệu xác định từ dòng nhập ***istream*** (input stream) vào biến (variable)

2.3.1.1. Cú pháp:

```
cin >> <biến>;
```

2.3.1.2. Ví dụ :

So sánh cách nhập dữ liệu dùng **cin** và **scanf()**

Dùng cin	Dùng scanf()
int a; cin >>a;	int a; scanf("%d",&a);
float b; cin >>b;	float b; scanf("%f",&b);
float x,y; Cin >>x>>y;	float x,y; scanf("%f%f",&x,&y);

2.3.3. Xuất dữ liệu dung cout

cout là một đối tượng thuộc dòng xuất (output stream object) thuộc lớp ***iostream*** (input/output stream) được định nghĩa trong C/C++

cout cho phép ghi giá trị thuộc kiểu dữ liệu bất kỳ lên dòng xuất ***ostream*** (output stream) từ biểu thức (expression)

2.3.3.1. Cú pháp

```
cout << <biểu thức>;
```

Khác với ***scanf(), cout*** tự thực hiện việc định dạng dữ liệu tùy thuộc vào kiểu dữ liệu của đối tượng xuất.

Để tạo dòng mới ta có thể dùng ký tự **'\n'** như đối với hàm ***printf()*** hay dùng đối tượng **"endl"** của iostream

2.3.3.2. Ví dụ

So sánh cách xuất dữ liệu dùng **cout** và **printf()**

Dùng *cout*	Dùng *printf()*
cout<<"Nhap so n = ";	printf("Nhap so n = ");
cout<<"Ket qua = "<<s<<"\n";	printf("Ket qua = %f\n",s);
cout<<a<<"+"<<b<<"="<<a+b;	printf("%f+%f=%f",a,b,a+b);

Lớp iostream không hỗ trợ công cụ cho việc định dạng dữ liệu xuất. Để định dạng dữ liệu xuất ta phải dùng lớp ***iomanip*** (input/output manipulation) với các phương thức (method) sau:

Minh họa các phương thức định dạng dữ liệu xuất của lớp ***iomanip***

Phương thức	Ý nghĩa	Ví dụ
setbase(base)	Đặt chế độ xuất theo cơ số base	cout<<setbase(16)<<32; -> 20
set(int n)	Thiết lập độ rộng xuất n	cout<<setw(12)<<32; -> 20
setiosflags(flag)	Thiết lập cờ (flag) cho việc xuất số thực theo dạng khoa	cout<<setiosflags(ios_base::scientific)<<123.456;

	học (scientific)	->1.234560e+002
Resetiosflags(flag)	Xóa cờ đã thiết lập setiosflags(flag)	cout<<resetiosflags(ios_base::scientific);
setfill(c)	Điền ký tự c vào vị trí trống trước giá trị xuất	cout<<setw(12)<< setfill('#')<<32; ->##########20
setprecision(int n)	Thiết lập độ chính xác n cho giá trị số thực	cout<< setprecision(5)<<123.456 ->123.46

Ví dụ 2.6 được viết lại dùng cin và cout như sau:

```
#include <iostream>
#include <conio.h>
#define PI 3.14159
using namespace std;
int main()
{
     float r, c, a;
     cout<<"Ban kinh: ";
     cin>>r;
     if(r>0)
     {
          c = 2 * PI * r;
          a = PI * r * r;
          cout<<"Chu vi: "<<c;
          cout<<"Dien tich: "<<a;
```

```
    }
    else
        cout<<"Gia tri nhap khong hop le";
    getch();
    return 0;
}
```

CHƯƠNG 3: CẤU TRÚC ĐIỀU KHIỂN

3.1. CẤU TRÚC RẼ NHÁNH

3.1.1. Lệnh if…else

3.1.1.1. Cú pháp

```
if(<điều kiện>) <phát biểu>;
```

Nếu `<điều kiện>` có giá trị `true` thì thực hiện `<phát biểu>`

```
if(<điều kiện>)
    <phát biểu 1>;
else
    <phát biểu 2>;
```

Nếu `<điều kiện>` có giá trị `true` thì thực hiện `<phát biểu 1>`, ngược lại thực hiện `<phát biểu 2>`

`<điều kiện>`: Biểu thức điều kiện (condition) bất kỳ bao gồm các tóan tử so sánh, logic, bitwise với kết quả là một giá trị logic `true` (1) hoặc `false` (0). Biểu thức điều kiện phải được viết trong cặp ngoặc {…}

`<phát biểu >`: Phát biểu (statement) bất kỳ và có thể là phép gán, biểu thức, phép gọi hàm, và cũng có thể là một cấu trúc điều khiển bất kỳ. Kết thúc một phát biểu phải có

một dấu chấm phẩy “;”. Nếu <phát biểu > là khối lệnh thì phải được bao trong cặp ngoặc {…}.

3.1.1.2. Ví dụ

Ví dụ 3.1:

Chương trình sau cho phép nhập 2 số nguyên và in ra số lớn hơn.

```
#include <stdio.h>
#include <conio.h>
int main()
{
     int a,b;
     printf("Nhap 2 so nguyen bat ky: ");
     scanf("%d%d", &a, &b);
     if(a>b)
          printf("max(%d,%d)=%d\n", a, b, a);
     else
          printf("max(%d,%d)=%d\n", a, b, b);
     getch();
     return 0;
}
```

Kết quả chạy chương trình:

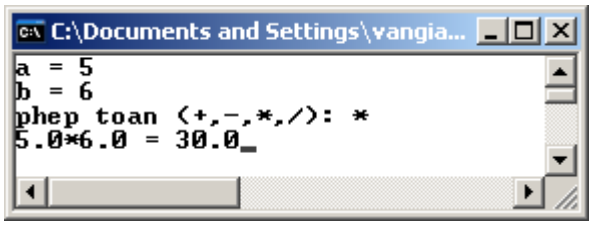
Ví dụ 3.2:

Chương trình sau cho phép người dùng nhập vào 2 số a, b và phép toán cần thực hiện (+, -, *, /). Sau đó in kết quả ra màn hình.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    float a,b;
    char pheptoan;
    printf("a = ");scanf("%f", &a);
    printf("b = ");scanf("%f", &b);
    printf("phep toan (+,-,*,/): ");
    pheptoan=getche();
    if (pheptoan=='+')
          printf("\n%.1f+%.1f = %.1f",a,b,a+b);
    else
        if (pheptoan=='-')
            printf("\n%.1f-%.1f = %.1f",a,b,a-b);
        else
            if (pheptoan=='*')
                printf("\n%.1f*%.1f = %.1f",a,b,a*b);
            else
                if (pheptoan=='/')
                    if (b==0)
                        printf("\nKhong chia duoc!");
                    else printf("\n%.1f/%.1f=%f",a,b,a/b);
                else printf("\nPhep toan khong hop le !");
```

```
    getch();

    return 0;

}
```

Kết quả khi chạy chương trình:



3.1.2. Lệnh switch

3.1.2.1. Cú pháp

```
switch(<biểu thức>)
{
    case <hằng 1>:
        <phát biểu 1>;
        [break;]
    case <hằng 2>:
        <phát biểu 2>;
        [break;]
    …
    [default:<phát biểu>;]
}
```

Ý nghĩa:

Nếu <biểu thức> = <hằng 1> : thực hiện <phát biểu 1>

Nếu <biểu thức> = <hằng 2> : thực hiện <phát biểu 2>

…

Trường hợp mặc định (default) ≠(<hằng 1>,<hằng 2>,…): thực hiện <phát biểu>

Trong đó:

<biểu thức>: Biểu thức (expression) cho kết quả là kiểu dữ liệu đếm được, nghĩa là các giá trị rời rạc kiểu char, int, long,…

<hằng 1>, <hằng 2>,…: Giá trị kết quả tương ứng của <biểu thức>

<phát biểu 1>, <phát biểu 2>, <phát biểu>: Phát biểu (statement) bất kỳ

Thông thường sau mỗi khối case cần có một phát biểu break nhằm chuyển điều khiển ra ngòai switch. Điều này sẽ làm cho các phát biểu ở các khối case sau đó không được thực hiện.

3.2.2.2. Ví dụ

Ví dụ 3.3: Chương trình ở ví dụ 2 được trình bày lại dùng cấu trúc switch.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    float a,b;
    char pheptoan;

    printf("a = ");scanf("%f", &a);
    printf("b = ");scanf("%f", &b);
    printf("phep toan (+,-,*,/): ");
    pheptoan=getche();
    switch (pheptoan)
    {
```

```
        case '+': printf("\n%.1f+%.1f = %.1f",a,b,a+b);break;
        case '-': printf("\n%.1f-%.1f = %.1f",a,b,a-b);break;
        case '*': printf("\n%.1f*%.1f = %.1f",a,b,a*b);break;
        case '/':
            if (b==0) printf("\nKhong chia duoc cho 0 !");
            else printf("\n%.1f/%.1f = %.1f",a,b,a/b);
            break;
        default: printf("\nPhep toan khong hop le !");
    }
    getch();
    return 0;}
```

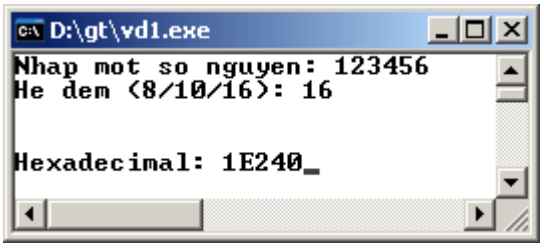
Ví dụ 3.4:

Chương trình sau cho phép người dung nhập vào một số nguyên và lựa chọn in giá trị đã nhập theo một trong các hệ đếm decimal, octal, hexadecimal

```
#include <stdio.h>
#include <conio.h>
int main()
{
   long num;
   int base;

   printf("Nhap mot so nguyen: ");
   scanf("%ld", &num);
   printf("He dem (8/10/16): ");
   scanf("%d", &base);
```

```
   printf("\n\n");
   switch(base)
   {
      case 8:
           printf("Octal: %lo", num);
           break;
      case 10:
           printf("Decimal: %ld", num);
           break;
      case 16:
           printf("Hexadecimal: %lX", num);
           break;
      default:printf("He dem %d khong xac dinh", base);
   }
   getch();
   return 0;
}
```

Kết quả chạy chương trình:



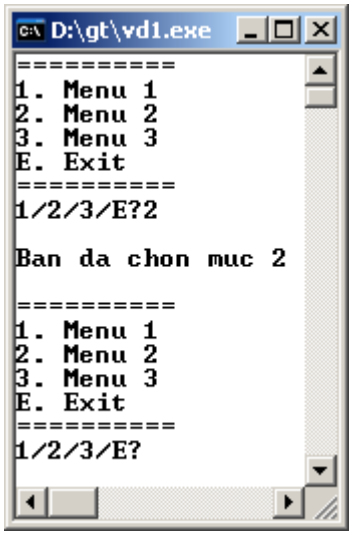
Ví dụ 3.5:

Chượng trình sau hiện lên màn hình 1 menu gồm 4 mục. Người dùng gõ 1 trong các phím 1, 2, 3 hoặc E để chọn mục tương ứng. Nếu phím gõ vào không thuộc 4 mục này sẽ bị báo lỗi. Chọn e hoặc E (exit) để kết thúc chương trình.

```
#include <stdio.h>
#include <conio.h>
int main()
{
   char key;
   do
   {
      printf("==========\n");
      printf("1. Menu 1\n");
      printf("2. Menu 2\n");
      printf("3. Menu 3\n");
      printf("E. Exit\n");
      printf("==========\n");
      printf("1/2/3/E?");
      key=getche();
      switch(key)
      {
         case '1':
         case '2':
         case '3':
            printf("\n\nBan da chon muc %c\n\n", key);
            break;
```

```
         case 'e':
         case 'E':
            quit=1;
            break;
         default:
            printf("\n\nKhong co muc %c\n\n", key);
      }
      if(quit)break;
   }while (key!='E'&&key!='e');
   return 0;
}
```

Kết quả:



3.2. VÒNG LẶP

3.2.1. Vòng lặp for

3.2.1.1. Cú pháp

```
for(<BT khởi tạo>;<BT điều kiện>; <BT lặp>)
   <phát biểu>;
```

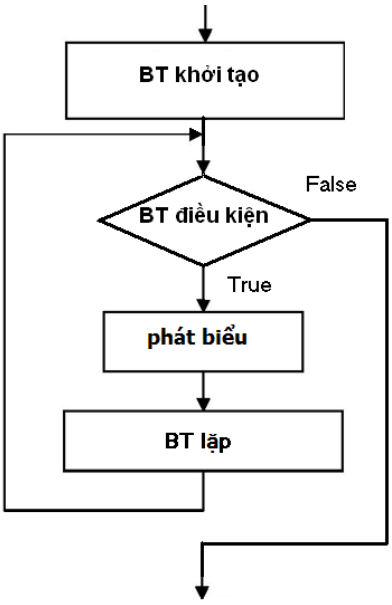
Nếu <*phát biểu*> là khối lệnh thì phải được bao trong cặp ngoặc {…}

<*BT khởi tạo*>: Biểu thức khởi tạo (init expression) được thực hiện trước khi lặp thường dùng để khởi tạo giá trị cho biến đếm số lần lặp.

<*BT điều kiện*>: Biểu thức điều kiện (cond expression) để mô tả điều kiện thực hiện của vòng lặp. Nếu <*BT điều kiện*> đúng (true) thì <*lệnh*> được thực hiện, ngược lại kết thúc lặp.

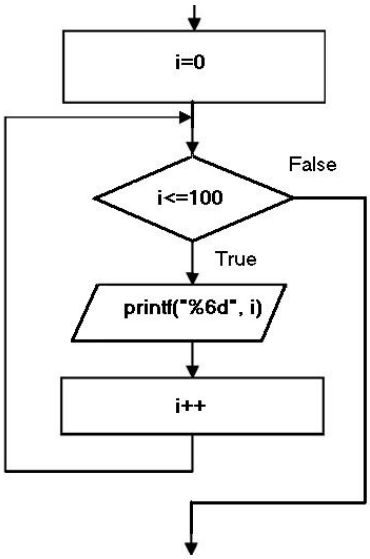
<*BT lặp*>: Biểu thức lặp (loop expression) được thực hiện sau khi thực hiện <lệnh>. <*BT lặp*> thường dùng để tăng giá trị cho biến đếm số lần lặp.

Ý nghĩa:



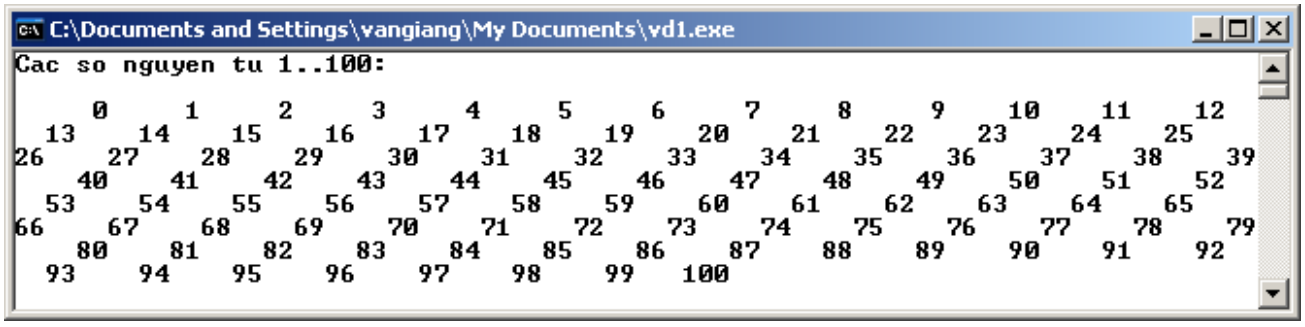
3.2.1.2. Ví dụ

Ví dụ 3.6: Chương trình sau cho phép in ra màn hình các số nguyên từ 0 đến 100.




```
#include <stdio.h>
#include <conio.h>
int main()
{
     int i;
     printf("Cac so nguyen tu 0..%d:\n\n", MAX);
     for(i=0; i<=100; i++)
          printf("%6d", i);
     getch();
     return 0;
}
```

Kết quả khi chạy chương trình:

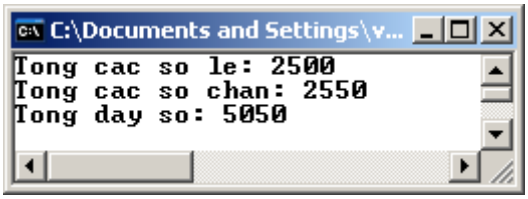


Ví dụ 3.7:

Tính tổng các số chẵn và tổng các số lẻ từ dãy số nguyên 1, 2,…,100

```
#include <stdio.h>
#include <conio.h>
#define MAX   100
int main()
{
     int i, odd=0, even=0;
     for(i=1; i<=MAX; i++)
          if(i%2!=0)          // i là số lẻ
               odd+=i;
          else                // i là số chẵn
               even+=i;
     printf("Tong cac so le: %d\n", odd);
     printf("Tong cac so chan: %d\n", even);
     printf("Tong day so: %d", odd+even);
     getch();
     return 0;
}
```

Kết quả khi chạy chương trình:



Lưu ý:

- Cấu trúc lặp `for` còn được gọi là cấu trúc lặp cố định (fixed loop) vì số lần lặp hòan t toàn được xác định. Người ta thường dùng cấu trúc lặp for khi biết trước số lần lặp.
- Với cách viết:

```
for( ; ; )
   <phát biểu>;
```

sẽ tạo ra việc lặp vô tận và chỉ có thể kết thúc bằng một trong các phát biểu `break`, `goto` hoặc `return`

3.2.2. Vòng lặp while

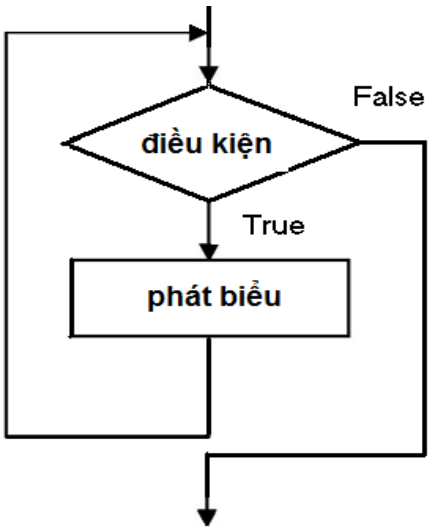
3.2.2.1. Cú pháp

```
while(<điều kiện>)
   <phát biểu>;
```

Nếu *<phát biểu>* là khối lệnh thì phải được bao trong cặp ngoặc {…}

<điều kiện> (condition): Biểu thức điều kiện bất kỳ bao gồm các tóan tử so sánh, logic, bitwise với kết quả là một giá trị logic `true` (1) hoặc `false` (0).

Ý nghĩa:



Lưu ý:

- Cấu trúc `while` còn gọi là cấu trúc lặp kiểm tra điều kiện trước (pre-test loop) vì điều kiện được kiểm tra trước khi thực hiện `<phát biểu>`
- `<phát biểu>` có thể không được thực hiện lần nào nếu lần kiểm tra *<điều kiện>* đầu tiên có trị `false`
- Người ta thường dùng cấu trúc lặp while khi chưa xác định được số lần lặp.
- Với phát biểu:

```
while(1);
```

sẽ tạo ra việc lặp vô tận, và chỉ có thể kết thúc bằng một trong các phát biểu `break`, `goto` hoặc `return`

3.2.2.2. Ví dụ

Ví dụ 3.8:

Viết lại chương trình in ra màn hình các số nguyên từ 0 đến 100, dùng cấu trúc `while` thay vì `for`

```
#include <stdio.h>
#include <conio.h>
#define MAX   100
int main()
{
  printf("Cac so nguyen tu 0..%d:\n\n", MAX);
  int i=0;
  while(i<=MAX)
    printf("%6d", i++);
  getch();
  return 0;
```

```
}
```

Ví dụ 3.9:

Viết lại chương trình tính tổng các số chẵn và tổng các số lẻ từ dãy số nguyên 1, 2,…,100, dùng cấu trúc `while` thay vì `for`

```
#include <stdio.h>
#include <conio.h>
#define MAX   100
int main()
{
     int i, sum, odd, even;
     i = 1;
     odd = even = sum = 0;
     while(i<=MAX)
     {
          sum+=i;
          if(i%2!=0)          // i là số lẻ
               odd+=i;
          else                // i là số chẵn
               even+=i;
          i++;
     }
     printf("Tong cac so le: %d\n", odd);
     printf("Tong cac so chan: %d\n", even);
     printf("Tong day so: %d", sum);
     getch();
     return 0;
}
```

Vòng lặp **for** và **while** có thể biến đổi tương đương như sau:

Vòng lặp for	Vòng lặp while
For (<BT khởi tạo> ; <BT điều kiện> ; <BT lặp>) **{** <Thân vòng lặp>; **}**	**<BT khởi tạo> ;** **while (<BT điều kiện>)** **{** <Thân vòng lặp>; **<BT lặp>;** **}**
for (; <BT điều kiện> ;) **{** <Thân vòng lặp>; **}**	**while (<BT điều kiện>)** **{** <Thân vòng lặp>; **}**

3.2.3. Vòng lặp do…while

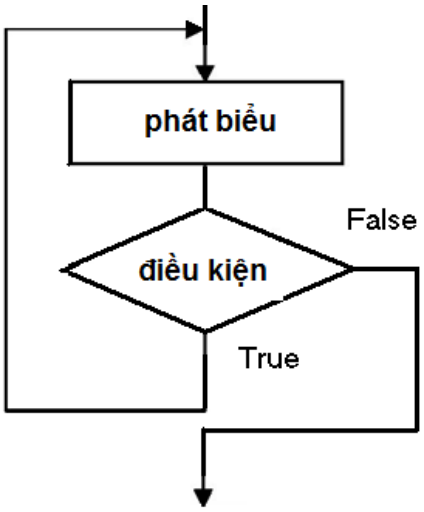
3.2.3.1. Cú pháp

```
do <phát biểu> while(<điều kiện>);
```

Nếu *<phát biểu>* là khối lệnh thì phải được bao trong cặp ngoặc {…}

<điều kiện>: Biểu thức điều kiện bất kỳ bao gồm các tóan tử so sánh, logic, bitwise với kết quả là một giá trị logic `true` (1) hoặc `false` (0). Biểu thức <điều kiện> phải được viết trong cặp ngoặc (…)

Ý nghĩa:



Lưu ý:

- Cấu trúc `do…while` còn gọi là cấu trúc lặp kiểm tra điều kiện sau (post-test loop) vì *<điều kiện>* được kiểm tra sau khi thực hiện *<phát biểu>*
- *<phát biểu>* được thực hiện tối thiểu 1 lần cho dù lần kiểm tra *<điều kiện>* đầu tiên có trị `true`
- Người ta thường dùng cấu trúc lặp `do…while` khi chưa xác định được số lần lặp hay cần kiểm tra dữ liệu nhập.
- Với phát biểu:

```
do ; while(1);
```

sẽ tạo ra việc lặp vô tận, và chỉ có thể kết thúc bằng một trong các phát biểu `break`, `goto` hoặc `return`

3.2.3.2. Ví dụ

Ví dụ 3.10:

Viết lại chương trình in ra màn hình các số nguyên từ 1 đến n, dùng cấu trúc `do…while`. Trong đó n là số nguyên dương được nhập vào.

```
#include <stdio.h>
#include <conio.h>
int main()
{
     int i=1,n;
     do
     {
          printf("Nhap so mot nguyen duong:");
          scanf("%d",&n);
     }while (n<=0);
```

```
    printf("Cac so nguyen tu 1..%d:\n\n", n);
    do printf("%6d", i++); while(i<=n);
    getch();
    return 0;
}
```

Ví dụ 3.11:

Viết lại chương trình tính tổng các số chẵn và tổng các số lẻ từ dãy số nguyên 1, 2,…,n dùng cấu trúc `do…while`. Trong đó n là số nguyên dương được nhập vào.

```
#include <stdio.h>
#include <conio.h>
int main ()
{
   int i, sum, odd, even, n;
   i = 1;
   odd = even = sum = 0;
     do
     {
          printf("Nhap so mot nguyen duong:");
          scanf("%d",&n);
   }while (n<=0);
   do{
      sum+=i;
      if(i%2!=0)    // i la so le
         odd+=i;
      else       // i la so chan
```

```
            even+=i;
        i++;
    }while(i<=n);
    printf("Tong cac so le: %d\n", odd);
    printf("Tong cac so chan: %d\n", even);
    printf("Tong day so: %d", sum);
    getch();
    return 0;
}
```

3.3. CÂU LỆNH RẼ NHÁNH (SV tự học)

3.3.1. Câu lệnh break

Dùng để kết thúc việc thực hiện các phát biểu từ sau vị trí ***break*** tới hết cấu trúc hiện thời như ***switch, for, while, do…while***. Điều khiển được chuyển tới phát biểu hay cấu trúc kế sau cấu trúc đã được kết thúc.

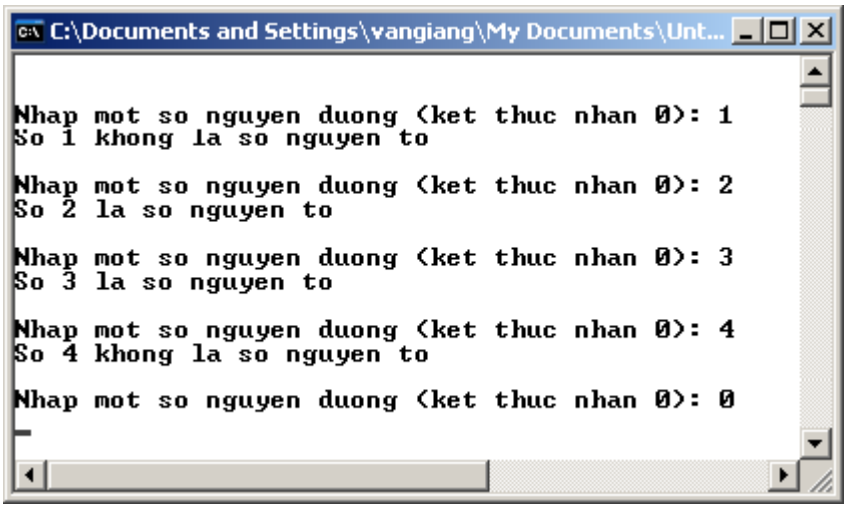
Trong những cấu trúc ***switch, for, while, do…while*** lồng vào nhau, lệnh ***break*** chỉ có tác dụng đối với cấu trúc bao nó gần nhất.

Ví dụ 3.12: Chương trình sau cho phép người dùng thực hiện nhiều lần việc nhập vào một số nguyên dương (kết thúc chương trình khi nhập số 0) và in ra thông báo số vừa nhập có là số nguyên tố không.

```
…
#include <math.h>
using namespace std;
int main()
{
    int m,i;
```

```
   while (1)
   {
      do
      {
         cout<<"\n\nNhap mot so nguyen duong (ket thuc
                                              nhan 0): ";
        cin>>m;
       }while (m<0);
       if (m==0) break; //thoát khỏi vòng lặp while (1)
       float can = sqrt((float) m);
      for(i=2; i<=can; i++)
          if(m%i==0) break; //thoát khỏi vòng lặp for
      if (m==1||i<=can)
          cout<<"So "<<m<<" khong la so nguyen to";
      else cout<<"So "<<m<<" la so nguyen to";
     }
   getch();
return 0;}
```

Kết quả chạy chương trình:



3.3.2. Câu lệnh continue

Giúp bỏ qua việc thực hiện các phát biểu sau ***continue*** trong các cấu trúc lặp ***for, while, do...while,*** và chuyển điều khiển về đầu cấu trúc lặp có chứa ***continue***.

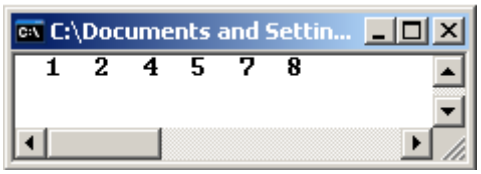
continue thường được sử dụng trong cấu trúc lặp nhằm mục đích chuyển điều khiển về đầu vòng lặp.

Trong những cấu trúc lặp ***for, while, do...while*** lồng vào nhau, lệnh continue chỉ có tác dụng đối với cấu trúc lặp gần nhất chứa nó.

Ví dụ 3.13: Chương trình in ra màn hình các số nằm trong đoạn từ 1 đến 10 không chia hết cho 3.

```
#include <stdio.h>
#include <conio.h>
#define MAX 10
int main()
{
     for(int i=1; i<MAX; i++)
     {
          if(i%3==0) continue;
          printf("%3d", i);
```

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

Kết quả chạy chương trình: 

BÀI TẬP CHƯƠNG III

1. Viết chương trình hoán đổi giá trị của 2 số nguyên được nhập.

2. Viết chương trình nhập vào chiều dài và chiều rộng của một hình chữ nhật, tính và in ra màn hình:

   a. Chu vi hình chữ nhật

   b. Diện tích hình chữ nhật

3. Viết chương trình giải và biện luận phương trình bậc 2 với các hệ số a, b, c và nghiệm x là các số thực.

4. Viết chương trình nhập một ký tự, nếu là chữ hoa thì đổi sang chữ thường và ngược lại.

5. Viết chương trình nhập vào 3 số a, b, c và :

   a. Kiểm tra xem a, b, c có phải là độ dài 3 cạnh của một tam giác không.

   b. Kiểm tra tam giác trên (nếu có) là tam giác gì (vuông, cân đều, tam giác thường)

   c. Tính chu vi, diện tích hình tam giác.

6. Viết chương trình nhập vào một số nguyên là tổng số giây (second), đổi tổng số giây sang giờ, phút, giây và in ra màn hình theo dạng hh:mm:ss

7. Viết chương trình nhập vào 3 số nguyên. In ra cho biết số lớn thứ nhất, số lớn thứ hai, và số nhỏ nhất

8. Viết chương trình dùng các cấu trúc lặp `for`, `while`, `do…while` để in dãy các số nguyên lẻ nhỏ hơn 100 theo chiều ngược với mẫu sau:

99, 97, 95, …, 3, 1

9. Viết chương trình tính :

a. S=1+ 3+ 5 +…+n (n là số dương lẻ)

b. S= 2+4+6+…+2*n

c. S= 1/2+1/3+1/4+…+1/n

d. S=1/2 + 3/4 + 5/6 +…+n/(n+1)

e. n!=1*2*3*…*n

f. $P=x^n$ (thực hiện bằng 2 cách dùng và không dùng hàm pow())

g. P=1!+2!+3!+…+n!

9. Viết chương nhập một số nguyên không âm và ra số đảo ngược. Ví dụ, nếu nhập vào 483 thì in ra 384.

10. Viết chương trình tính tổng và tích của dãy số gồm n phần tử. Trong đó số n và các phần tử được nhập vào từ bàn phím khi thực hiện chương trình.

11. Viết chương trình kiểm tra và in ra các số nguyên tố nhỏ hơn 100

12.Viết chương trình cho phép người nhập vào n số thực bất kỳ với n là giá trị nhập. Sau đó thực hiện các công việc sau:

a. Đếm xem có bao nhiêu số thực dương

b. Tính tổng các số thực dương

c. Tính trung bình cộng của các số thực dương

d. Tìm cố lớn nhất và số nhỏ nhất trong tất cả các số được nhập

e. Tìm số lớn nhất và nhỏ nhất trong các số thực dương

13. Viết chương trình cho phép người nhập vào một dãy các số nguyên dương, kết thúc nhập khi người dùng nhập vào số 0. Sau đó thực hiện các công việc sau:

a. Đếm xem có bao nhiêu số chẵn

b. Tính tổng các số chẵn

c. Tính trung bình cộng của các số chẵn

d. Tìm số lớn nhất và số nhỏ nhất trong tất cả các số chẵn

e. Đếm xem có bao nhiêu số nguyên tố

14. Viết chương trình Trò chơi Oẳn tù tì

Hướng dẫn:

- Cho nhập tên 2 người chơi.

- Cho chương trình sinh 2 số ngẫu nhiên a và b khác nhau trong phạm vi từ 1 đến 3 ứng với 2 người chơi, sử dụng :

+ Hàm khởi động bộ sinh số ngẫu nhiên: srand() trong thư viện <cstdlib>.

+ Hàm sinh số ngẫu nhiên: rand() trong thư viện <cstdlib>.

+ Hàm lấy thời gian của hệ thống: time() trong thư viện <ctime>

- Chuyển đổi số nguyên n bất kỳ để giá trị nằm trong đoạn [1, m] (Dùng chép chia lấy dư : n%m +1

- In kết quả của từng người (1 : Kéo, 2: Búa, 3: Bao)

- So sánh 2 số:

Nếu a=0&&b=3 thì a thắng

Ngược lại:

Nếu a<b thì b thắng

Ngược lại a thắng

- Xuất thông báo người thắng cuộc.

Ví dụ kết quả chạy chương trình:

```
Ten nguoi choi 1: Lan
Ten nguoi choi 2: Dung
Lan chon KEO >< Dung chon BAO
NGUOI THANG CUOC: Lan
```

Phát triển trò chơi: Sau khi học xong phần vòng lặp cần cải tiến để cho phép chương trình chạy nhiều lần đến khi tìm được người thắng cuộc.

15. Viết Chương trình Trò chơi Bói vui

Cho người chơi nhập ngày tháng năm sinh và in ra dự đoán tương lai của người chơi.

Hướng dẫn:

- Cho người chơi nhập ngày (d), tháng (m), năm sinh (y) và biến đổi thành một số nguyên từ 1 đến n (n là số lượng lá số mà người lập trình muốn tạo).

Ví dụ: số sinh được = (d+m+y)%n+1.

- Ứng với mỗi số từ 1 đến n sẽ xuất ra lá số tương ứng (là những câu dự đoán vui ứng với từng con số).

Ví dụ kết quả chạy chương trình:

```
Ngay sinh cua ban (DD/MM/YY): 15/02/2000
LA SO CUA BAN:
Ban se lam viec cho cong ty da quoc gia va phai di lai rat nhieu !!!
```

16. Viết chương trình Trò chơi Chữ chạy

Cho người chơi nhập họ tên và cho phép họ người chơi chạy liên tục từ trên xuống dưới và từ trái qua phải và đổi màu chữ liên tục.

Hướng dẫn:

- Cho người chơi nhập họ tên.
- Khởi tạo biến n=1.
- Lặp liên tục các thao tác:
  + Xóa màn hình
  + Chèn n dòng trống
  + Chèn 1 khoảng tab.
  + Đổi màu ngẫu nhiên hoặc tuần tự từ 0 đến F
  + Xuất chuỗi họ tên.
  + Tăng n
- Chương trình sử dụng các hàm sau:
  + Hàm xóa màn hình: system("cls") trong thư viện <windows.h>
  + Hàm đổi màu chữ system("COLOR xy") trong thư viện <windows.h>
  + Hàm chờ Sleep() trong thư viện <stdlib.h>

CHƯƠNG 4: HÀM

4.1. TỔNG QUAN VỀ HÀM TRONG C/C++

4.1.1. Khái niệm hàm

4.1.1.1. Khái niệm

Hàm (function) là một chương trình con (subroutine) trong đó cho phép định nghĩa trước các phát biểu cần thực hiện nhằm giải quyết một nhiệm vụ xác định.

Trong một chương trình C, tối thiểu phải có một hàm, đó là hàm main(). Các hàm định nghĩa sẵn (pre-defined functions) của C/C++ và những hàm mới do người dùng định nghĩa sẽ được gọi tới trong hàm main(). Hàm main() được xem là điểm vào của một chương trình.

Thông thường thì một hàm sau khi được gọi thực hiện sẽ trả về một giá trị có kiểu xác định. Tuy nhiên điều này là không bắt buộc, nếu hàm không trả về giá trị thì kiểu của hàm được khai báo với từ khóa void (thực ra thì void cũng là một kiểu dữ liệu được định nghĩa trong C/C++).

4.1.1.2. Công dụng của hàm

Để giải quyết một một vấn đề nào đó, phương pháp đơn giản và hiệu quả là chia nhỏ vấn đề lớn thành các vấn đề nhỏ hơn còn gọi là các module. Mỗi module chỉ thực hiện việc giải quyết một vấn đề đơn giản theo một thuật toán xác định. Trong C/C++ mỗi module như vậy được hiện thực thành một hàm.

Như vậy, hàm là phương tiện cho phép chia một chương trình lớn thành các chương trình con nhỏ hơn. Điều này dẫn đến các lợi điểm sau:

- Việc lập trình trở nên đơn giản hơn, việc phân tích, sửa lỗi, nâng cấp chương trình cũng thuận tiện hơn.
- Một hàm có thể được gọi tới nhiều lần trong một chương trình và có thể được dùng trong các chương trình khác nhau, do đó có thể tiết kiệm được thời gian lập trình nhờ vào việc dùng lại các mã nguồn đã có này.
- Phần mềm thường được phát triển bởi một nhóm lập trình viên, việc chia nhỏ chương trình thành nhiều module giúp làm cho việc phân công công việc cho mỗi

thành viên sẽ thuận lợi hơn, mỗi người chỉ phải thực hiện một số module nhất định mà không cần phải biết hết toàn bộ chương trình.

4.1.2. Cấu trúc của hàm

4.1.2.1. Cấu trúc

Phần khai báo và mô tả hàm (prototype)

```
<Kiểu DL trả về>  <Tên hàm> ([<Danh sách tham số>]);
```

- ***<Kiểu DL trả về> (return type)***: Kiểu dữ liệu trả về của một hàm (ví dụ: **char, int, float,**…) hoặc là **void** nếu hàm không trả về giá trị . Trường hợp không ghi kiểu dữ liệu trả về thì sẽ được tự động xem là kiểu **int.**
- ***<Tên hàm> (function name)***: Tên của hàm, được đặt theo qui ước đặt tên của C/C++ . Lưu ý tên hàm không có dấu và khoảng trắng.
- ***<Danh sách tham số> (parameter list)*:** Danh sách tham số được truyền cho hàm, mỗi tham số phải có kiểu dữ liệu xác định, các tham số cách nhau bằng dấu phẩy. Nếu hàm không có tham số truyền thì sẽ được ghi là **void** hoặc để trống.

Phần khai báo thường được viết trước hàm **main()** sau phần các dẫn hướng biên dịch **#include** và khai báo hằng **#define** trong file **.cpp** hoặc viết trong file header **.h** nếu phần cài đặt hàm được viết trong file header **.h.** Sau khi đã khai báo ta có thể viết lời gọi sử dụng một hàm trước phần cài đặt của hàm đó.

Phần cài đặt hàm (Implementation)

Bao gồm các phát biểu cần thực hiện của hàm viết trong cặp ngoặc nhọn **{…}**. Phần cài đặt thường được viết ngay sau hàm **main().**

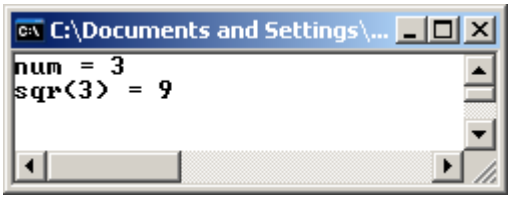
```
<Kiểu DL trả về> <Tên hàm>([<Danh sách tham số>])
{
      <Thân hàm>
}
```

Ví dụ 4.1:

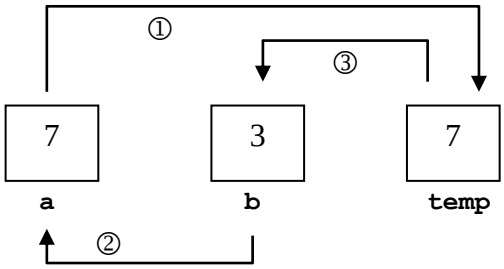
Hàm **sqr()** trong chương trình đơn giản sau sẽ trả về giá trị bình phương của một số nguyên **x** cho trước.

```
1   #include <stdio.h>
2   #include <conio.h>
3   // Prototype của hàm sqr()
4   long sqr(int x);
5   int main(void)
6   {
7         int num;
8         long square;
9         printf("num = ");
10        scanf("%d", &num);
11        square=sqr(num); // Gọi hàm sqr()
12        printf("sqr(%d) = %ld", num, square);
13        getch();
14        return 0;
15  }
16  // Implementation của hàm sqr()
17  long sqr(int x)
18  {
19        return (x * x);
20  }
```

Kết quả khi chạy chương trình:



Ví dụ 4.2: Chương trình sau sử dụng hàm **`Swap()`** để hoán vị giá trị của 2 biến **`a`** và **`b`** kiểu **`int`**. Chẳng hạn: **`a=7`** và **`b=3`** thì sau khi gọi hàm **`Swap(a,b)`** ta sẽ có **`a=3`** và **`b=7`**.



Hình 4.1: Thao tác để hóan vị giá trị của 2 biến **`a`** và **`b`**

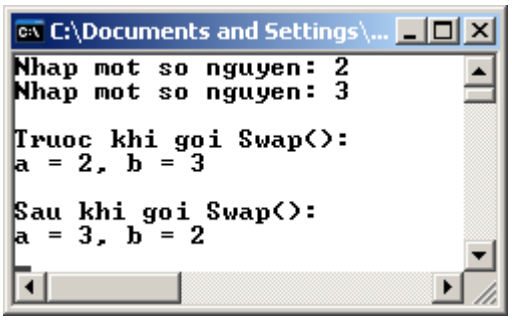
Ngoài ra trong chương trình còn có 2 hàm khác được cài đặt:

- **`GetNum()`** cho phép nhập dữ liệu từ bàn phím vào chương trình
- **`PrintOut()`** cho phép xuất dữ liệu ra màn hình

```
1   #include <stdio.h>
2   #include <conio.h>
3   // Prototype của các hàm
4   int GetNum(void);
5   void PrintOut(int a, int b);
6   void Swap(int &a, int &b);
7   int main(void)
8   {
9         int m, n;
10        m = GetNum();          // Gọi hàm GetNum() cho số thứ nhất
11        n = GetNum();          // Gọi hàm GetNum() cho số thứ hai
12        printf("\nTruoc khi goi Swap():\n");
13        PrintOut(m, n);        // Xuất giá trị ra màn hình
14        Swap(m, n);            // Gọi hàm Swap() để thực hiện hoán vị
15        printf("\nSau khi goi Swap():\n");
```

```
16        PrintOut(m,n);
17        getch();
18        return 0;
19  }
20  // Implementation của hàm Swap()
21  void Swap(int &a, int &b)
22  {
23        int temp = a;a = b; b = temp;
24  }
25  // Implementation của hàm GetNum()
26  int GetNum(void)
27  {
28        int x;
29        printf("Nhap mot so nguyen: ");
30        scanf("%d", &x);
31        return x;
32  }
33  // Implementation của hàm PrintOut()
34  void PrintOut(int a, int b)
35  {
36        printf("a = %d, b = %d\n", a, b);
37  }
```

Kết quả khi chạy chương trình:



4. 1.2.2. Cách trình bày phần khai báo và cài đặt hàm

Cách thông thường: Như trong các ví dụ trên.

- **Prototype** phải được viết trước khi gọi hàm để sử dụng.
- **Implementation** được viết sau khi gọi hàm thường là sau hàm main() hay cuối file chứa prototype.

Cách trình bày khác:

- Bỏ qua không viết phần **prototype** nhưng phải đảm bảo viết phần **implementation** của hàm trước khi sử dụng.

Ví dụ 4.1 được trình bày theo cách này như sau:

```
1   …
2   // Implementation của hàm sqr()
3   long sqr(int x)
4   {
5         return (x * x);
6   }
7   int main()
8   {
9         int num;
10        long square;
11        printf("num = ");
12        scanf("%d", &num);
13        square=sqr(num); // Gọi hàm sqr()
14        printf("sqr(%d) = %ld", num, square);
15        getch();
16        return 0;
17  }
```

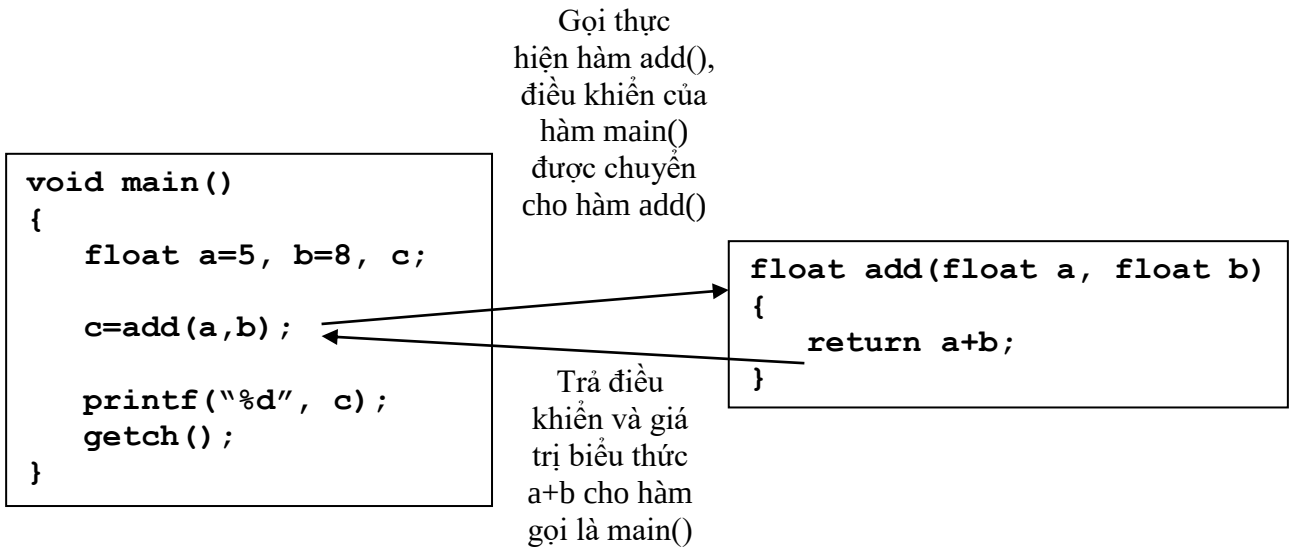
4.1.2.3. Lệnh *return* và *exit*

Lệnh *return*

```
return [<Biểu thức>];
```

- ***<Biểu thức> (expression)***: Biểu thức có giá trị kiểu tương ứng với kiểu trả về của hàm.
- Lệnh **return** trả điều khiển từ hàm được gọi cho hàm đã gọi nó tại vị trí của lệnh **return,** giá trị của ***<Biểu thức>*** nếu có được trả về cho hàm gọi. Những lệnh sau **return** sẽ bị bỏ qua khi **return** được thực hiện.
- **Đối với hàm có trả về giá trị:**

  Trong thân hàm luôn có câu lệnh : **return <biểu thức>;**



Hình 4.2: Minh họa điều khiển của chương trình khi gặp lệnh **return**

- **Đối với hàm không trả về giá trị:**

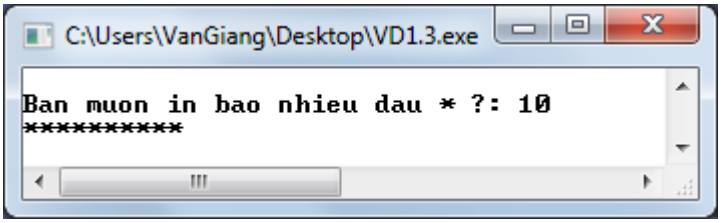
  Trong thân hàm có thể không có lệnh **return** hoặc dùng câu lệnh : **return ;**

Ví dụ 4. 3: Chương trình sau cho phép người dùng nhập vào một số nguyên n và in ra màn hình n dấu *.

```
…
1  void InSao(int n);
```

```
2   int main(void)
3   {
4         int  n;
5         printf("\nBan muon in bao nhieu dau * ?: ");
6         scanf("%d",&n);
7         InSao(n);
8         getch();
9         return 0;
10  }
11  void InSao(int n)
12  {
13        if (n<1) return ;
14        for (int i=1;i<=n;i++)
15           printf("*");
16  }
```

Kết quả khi chạy chương trình:



Hàm `exit()`

Hàm **`exit()`** dùng để kết thúc chương trình. Để có thể gọi hàm **`exit()`**, cần phải có dẫn hướng **`#include <stdlib.h>`** hoặc **`#include <process.h>`.**

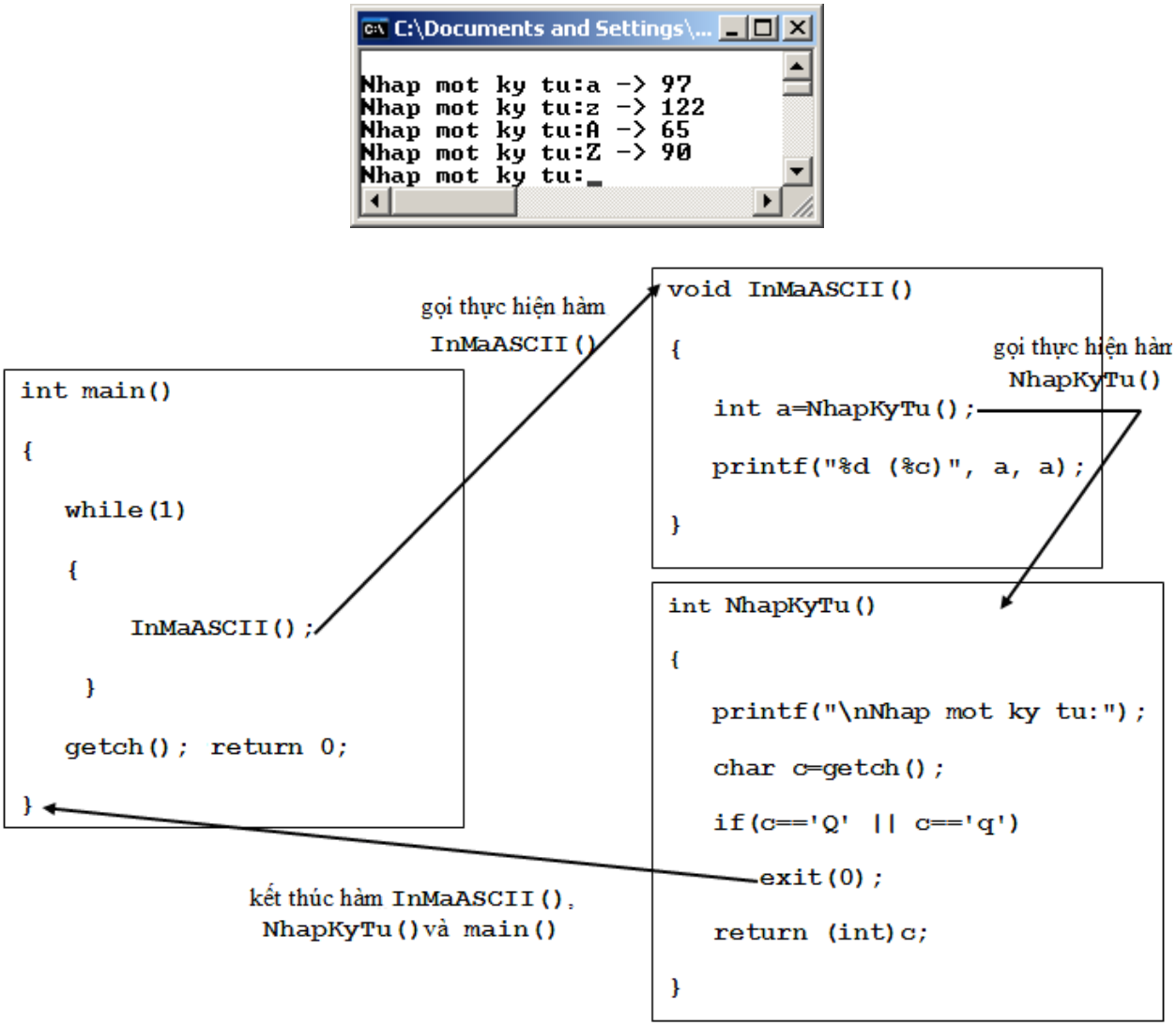
```
exit(<trạng thái>)
```

- ***<trạng thái> (status)***: Trạng thái kết thúc một quá trình, thường thì giá trị là 0 (hằng ***EXIT_SUCCESS***) nếu không có lỗi, ngược lại có giá trị khác 0 (hoặc hằng ***EXIT_FAILURE***).

Ví dụ 4. 4: Cho phép người dùng nhập các ký tự và in ra màn hình mã ASCII của ký tự này, nếu người dùng nhấn ‘q’ hay ‘Q’ chương trình sẽ kết thúc. Qua đó minh họa điều khiển của chương trình khi gặp hàm **`exit()`.**

```
    …
1   void InMaASCII();
2   int NhapKyTu();
3   int main()
4   {
5         while(1)
6         {
7              InMaASCII();
8         }
9         getch();
10        return 0;
11  }
12  void InMaASCII()
13  {
14        int a=NhapKyTu();
15        printf("%c -> %d ",a,a);
16  }
17  int NhapKyTu()
18  {
19        printf("\nNhap mot ky tu:");
20        char c=getch();
21        if(c=='Q' || c=='q')  exit(0);
22        return (int)c;
23  }
```

Kết quả khi chạy chương trình:



Hình 4.3: Minh họa điều khiển của chương trình khi gặp hàm **`exit()`**.

4.1.3. Sử dụng hàm

Prototype & implementation của một hàm có thể được viết trong một file header, khi đó để sử dụng hàm này cần phải có dẫn hướng **`#include`**.

Trường hợp hàm được cài đặt và biên dịch thành file nhị phân .dll (dynamic link library) thông thường, cần chép file .dll vào thư mục làm việc. Nếu hàm được cài đặt theo dạng COM/DCOM ((distributed) component object model) và được phân phối dưới dạng file .dll hay .ocx ta cần đăng ký nó (register) với hệ điều hành trước khi dùng.

Hàm có thể được gọi từ một khai báo hằng, biến, trong một biểu thức, vế phải của một phép gán, trong một hàm khác, và cũng có thể trong bản thân hàm đó trong trường hợp đệ quy (recursive).

Căn cứ vào kiểu trả về của hàm, ta có 2 trường hợp sau:

4.1.3.1. Hàm không trả về giá trị

Để gọi hàm chỉ cần ghi tên hàm & các tham số truyền (không khai báo kiểu dữ liệu của các tham số.

```
<Tên hàm> (<Tên các tham số>);
```

Bảng 4.1: Cách gọi hàm không trả về giá trị

Prototye của hàm	Cách gọi hàm	Ghi chú
`void Swap(int &a, int &b);`	`Swap(a, b);` `Swap(m, n);`	a, b, m, n phải được khai báo và gán/nhập giá trị trước khi dùng.
`void PrintOut(int a, int b);`	`PrintOut(a,b);`	

4.1.3.2. Hàm trả về giá trị

Cách 1: Xuất ngay giá trị trả về của hàm dùng **`printf()`** hay **`cout`**

Bảng 4.2: Cách gọi hàm trả về giá trị trong câu lệnh xuất

Prototye của hàm	Cách gọi hàm	Lưu ý
`long sqr(int x);`	`cout<< x<<"*"<<x<<"="<<sqr(x);` `printf("%d*%d=%ld",x,x,sqr(x));`	x, a, b phải được khai báo và gán/nhập giá trị trước khi dùng.
`float add(float a, float b);`	`cout<<"tổng là: "<<add(a,b);` `printf("tổng là: %f",add(a,b));`	

Cách 2: Gán giá trị trả về của hàm vào một biến và xử lý biến đó

Bảng 4.3: Cách gọi hàm trả về giá trị bằng cách gán vào biến

Prototye của hàm	Cách gọi hàm	Lưu ý

`long sqr(int x);`	`long t=sqr(x);`
	`cout<<x<<"*"<<x<<"="<<t;`
	`printf("%d*%d=%ld",x,x,t);`
		x, a, b phải được khai báo và gán/nhập giá trị trước khi dùng.

`float add(float a, float b);`	`float t= add(a,b);`
	`cout<<"tổng là: "<<t);`
	`printf("tổng là: %f",t);`	

4.1.4. Truyền tham số

Trong việc cài đặt hàm thì tham số (parameter) hay đối số (argument) truyền cho hàm là một vấn đề có tầm quan trọng rất lớn và nó ảnh hưởng đến cách thức cài đặt cũng như giải thuật sử dụng để cài đặt hàm.

4.1.4.1. Tham số thực (real parameters)

Là tham số được truyền thực sự cho hàm thông qua lời gọi hàm. Với tham số thực ta không chỉ quan tâm đến kiểu dữ liệu của tham số mà còn quan tâm đến tên của tham số tương ứng cần truyền. Tên của tham số thực do vậy có thể khác với tên của tham số hình thức, và trong nhiều trường hợp tham số thực không phải là tên mà là một hằng cụ thể.

4.1.4.2. Tham số hình thức (formal parameters)

Là tham số được khai báo trong prototype mô tả hàm. Đối với tham số hình thức điều ta cần quan tâm không phải là tên của tham số mà là kiểu của tham số cần truyền cho hàm và thứ tự các tham số ghi trong mô tả hàm. Như vậy tên của các tham số trong trường hợp này chỉ mang tính hình thức và chúng có thể khác tên với các tham số sẽ được truyền thực sự cho hàm thông qua lời gọi hàm.

Trong ví dụ 4.1 ở trên, ta có **`x`** là tham số hình thức và **`num`** là tham số thực của hàm **`sqr()`**.

```
1   long sqr(int x);// x: tham số được khai báo một cách hình thức
2   int main()
3   {
4         int num;
5         long square;
6         printf("num = ");scanf("%d", &num);
7         square=sqr(num); // num: tham số thực sự truyền cho hàm
8         printf("sqr(%d) = %ld", num, square);
9         getch();
10        return 0;
11  }
```

Tham số hình thức trị hay tham trị (value parameter)

Là tham số được truyền cho hàm bằng giá trị của tham số thực. Khi đó sự thay đổi giá trị của tham số trong hàm nếu có sẽ không làm thay đổi giá trị của tham số thực đó trong hàm gọi.

Trong ví dụ 1.1, với hàm : **`long sqr(int x);`**

thì **`x`** là tham trị, vì trong hàm **`main()`** giá trị của biến **`num`** là tham số thực truyền cho hàm **`sqr()`** qua phép gọi hàm **`square=sqr(num)`** không bị thay đổi sau khi kết thúc việc thực hiện hàm này.

Tham số hình thức biến hay tham biến (variable parameter):

Là tham số được truyền cho hàm bằng địa chỉ của tham số thực. Khi đó sự thay đổi giá trị tương ứng của tham số trong hàm (nếu có) sẽ làm thay đổi giá trị của tham số thực đó ở ngoài hàm.

Trong ví dụ 1.2, với hàm : **`void Swap(int &a, int &b);`**

thì **`a`** và **`b`** là các tham biến vì tham số được truyền cho hàm là **`&a`** (địa chỉ của a), **`&b`** (địa chỉ của b) nên sự thay đổi giá trị của **`a`**, **`b`** bên trong hàm **`Swap()`** sẽ được cập nhật ngay lên địa chỉ của **`a`**, **`b`** cũng chính là địa chỉ của tham số thực của **`m`** và **`n`** của hàm **`main()`** nên giá trị của **`m`**, **`n`** sẽ bị thay đổi theo.

Như vậy, ngoài giá trị trả về thông thường của hàm thông qua câu lệnh **`return`**, ta có thể "trả về" cho hàm gọi các giá trị khác bằng cách sử dụng tham biến.

Ví dụ 4.1 ở trên cũng có thể được trình bày lại như sau:

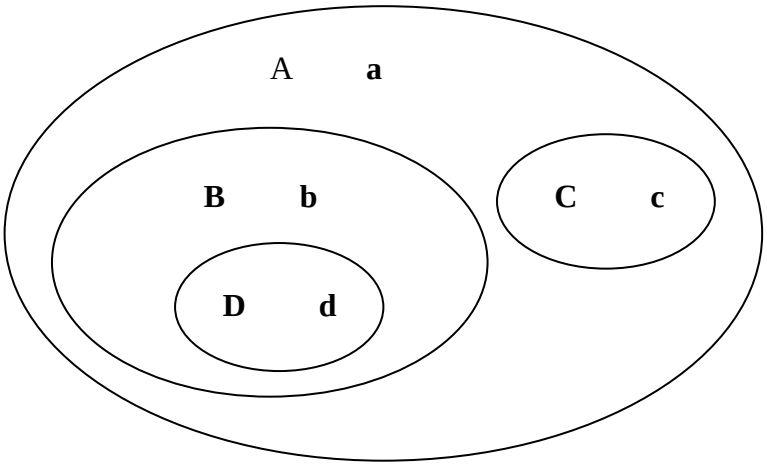
```
    …
1   void sqr(int x, long &result)
2   {
3       result =x * x;
4   }
5   int main()
6   {
7       int num;long square;
8       printf("num = ");
9       scanf("%d", &num);
10      sqr(num,square);
11      printf("sqr(%d) = %ld", num, square);
12      getch();
13      return 0;
14  }
```

4.2. VẤN ĐỀ TẦM VỰC

4.2.1. Khái niệm tầm vực

Tầm vực (scope) của một đối tượng (object) là phạm vi/miền (domain) mà đối tượng đó được biết đến và có thể sử dụng được trong phạm vi/miền đó.

Cụ thể hơn, đối tượng ở đây là một hàm (function), một biến (variable), một hằng (constant), một lớp (class). Phạm vi/miền ở đây là một cấu trúc điều khiển (control block), một hàm, một file , một chương trình, hay một hệ chương trình.



Hình vẽ trên là một ví dụ minh họa về tầm vực. Trong đó miền A chứa đối tượng a, miền B chứa đối tượng b, miền C chứa đối tượng c, và miền D chứa đối tượng d.

- Tầm vực của đối tượng a là miền A, B, C và D.
- Tầm vực của đối tượng b là miền B và D.
- Tầm vực của đối tượng c là miền C.
- Tầm vực của đối tượng d là miền D.

Khi đề cập đến tầm vực, ta quan tâm đến tầm vực của hàm và tầm vực của biến.

4.2.2. Qui tắc cơ bản về tầm vực

4.2.2.1. Tầm vực của hàm

Khi một hàm được mô tả trong trong một file (mã nguồn **`.cpp`** hoặc header **`.h`**) thì nó có phạm vi sử dụng chỉ trong file đó. Tầm vực của hàm này là chính nó và tất cả các hàm nằm sau phần khai báo hay cài đặt của hàm này.

Ví dụ 4.5: Chương trình sau cho phép nhập vào bán kính **`r`** của một hình tròn và in ra màn hình chu vi **`C`**, diện tích **`S`** tương ứng. Trong đó:

- Hàm **`NhapSo()`** có tầm vực là hàm **`ChuVi(), DienTich()`** và hàm **`main()`**
- Hàm **`ChuVi()`** có tầm vực là hàm **`DienTich()`** và hàm **`main()`**
- Hàm **`DienTich()`** có tầm vực là hàm **`main()`**

```
    …
1   float NhapSo();
2   float ChuVi(float r);
3   float DienTich(float r);
4   void main(void)
5   {
6       float r, C, S;
7       r = NhapSo();
8       C = ChuVi(r);
9       S = DienTich(r);
10      printf("\n\n");
11      printf("C = %-7.2f\n", C);
12      printf("S = %-7.2f\n", S);
13      getch();return 0;
14  }
15  float NhapSo()
16  {
17      float r;
18      do
19      {
20          printf("Ban kinh:\n");
21          scanf("%f", &r);
22      }while (r<0);
23      return r;
24  }
```

```
25  float ChuVi(float r)
26  {
27      if(r>0) return 2*PI*r;
28      else return -1;
29  }
30  float DienTich(float r)
31  {
32      if(r>0) return
33          PI*r*r;
34      else return -1;
35  }
```

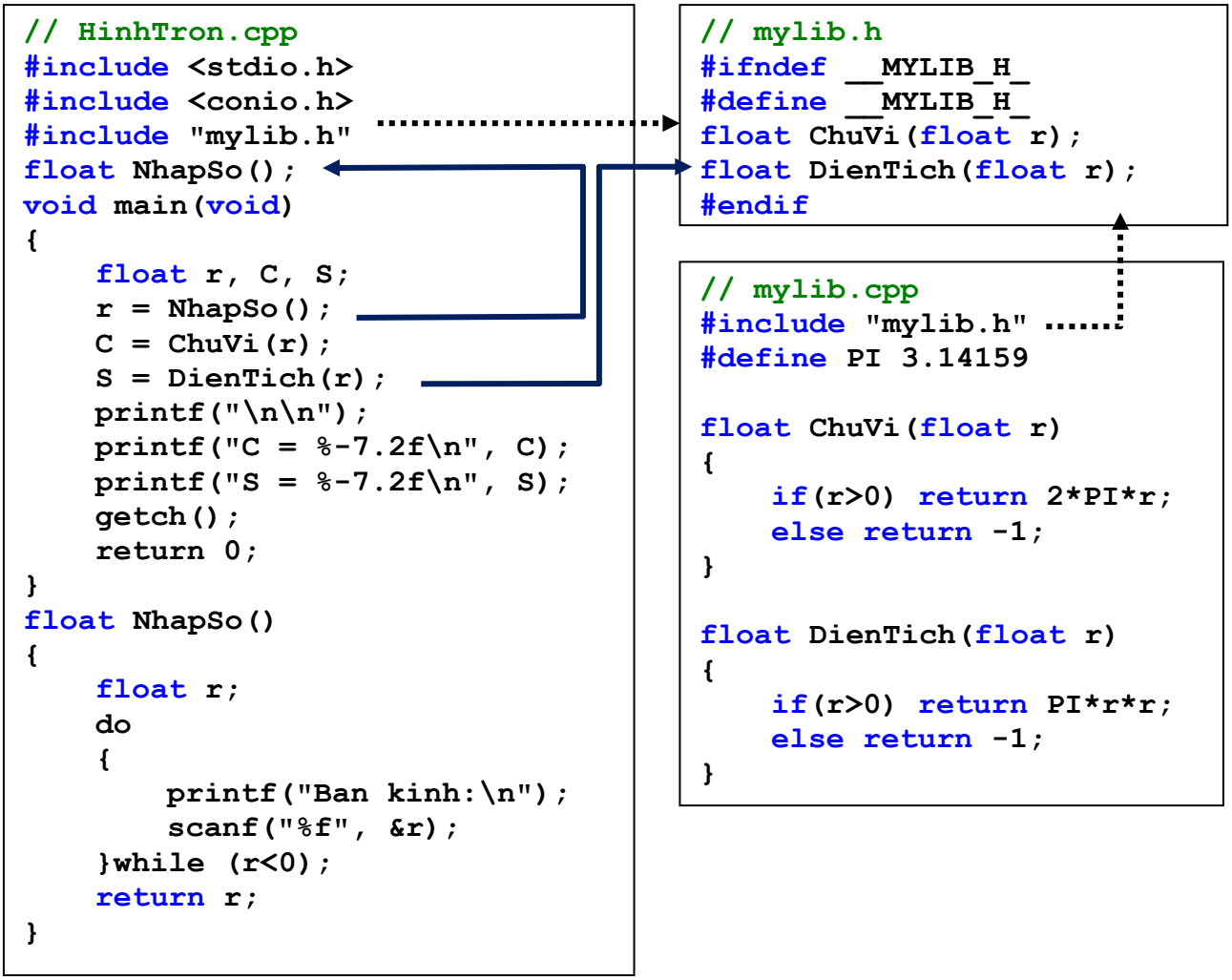
Trong trường hợp có dẫn hướng **#include** trong một file thì tầm vực của hàm trong file ghi trong dẫn hướng **#include** được mở rộng đến file có chứa dẫn hướng **#include**.

Ví dụ 4.6: Chương trình trong ví dụ 4.5 có thể trình bày lại bằng cách phân bố trên 3 file: **mylib.h, mylib.cpp, HinhTron.cpp**. Trong đó **HinhTron.cpp** là file mã nguồn chính có chứa hàm **main().**

Các hàm trong file **mylib.cpp** là cài đặt của các hàm có prototype mô tả trong **mylib.h**.

Bằng dẫn hướng **#include "mylib.h"** trong file **HinhTron.cpp** làm cho các hàm trong **mylib.h** có thể gọi được trong file **HinhTron.cpp**.

Hàm **NhapSo()** trong file **HinhTron.cpp** gọi được trong hàm **main()** và các hàm khác nếu có trong file này nhưng không thể gọi được từ các file **mylib.h** và **mylib.cpp**.



```
// HinhTron.cpp
#include <stdio.h>
#include <conio.h>
#include "mylib.h"
float NhapSo();
void main(void)
{
    float r, C, S;
    r = NhapSo();
    C = ChuVi(r);
    S = DienTich(r);
    printf("\n\n");
    printf("C = %-7.2f\n", C);
    printf("S = %-7.2f\n", S);
    getch();
    return 0;
}
float NhapSo()
{
    float r;
    do
    {
        printf("Ban kinh:\n");
        scanf("%f", &r);
    }while (r<0);
    return r;
}
```

```
// mylib.h
#ifndef __MYLIB_H_
#define __MYLIB_H_
float ChuVi(float r);
float DienTich(float r);
#endif
```

```
// mylib.cpp
#include "mylib.h"
#define PI 3.14159

float ChuVi(float r)
{
    if(r>0) return 2*PI*r;
    else return -1;
}

float DienTich(float r)
{
    if(r>0) return PI*r*r;
    else return -1;
}
```

4.2.2.2. Tầm vực của biến:

Biến cục bộ (local variable): Biến được khai báo trong một cấu trúc điều khiển hay hàm. Tầm vực của biến cục bộ là phạm vi của cấu trúc điều khiển hay hàm đó. Việc tham chiếu đến một biến ngoài tầm vực của nó sẽ gây ra lỗi biên dịch.

Biến toàn cục (global variable): Biến được khai báo bên ngoài tất cả các hàm. Tầm vực của biến toàn cục là toàn bộ file chứa khai báo đó.

Khi trong một file có chứa dẫn hướng **`#include`** thì biến toàn cục được khai báo trong file ghi trong **`#include`** có thể dùng trong file có chứa **`#include`.**

Sự thay đổi giá trị của biến toàn cục trong một hàm sẽ làm thay đổi giá trị của biến tòan cục bên ngoài hàm.

Các ví dụ sau đây sẽ minh họa tầm vực của 2 loại biến cục bộ và toàn cục đã nêu trên.

Ví dụ 4.7:

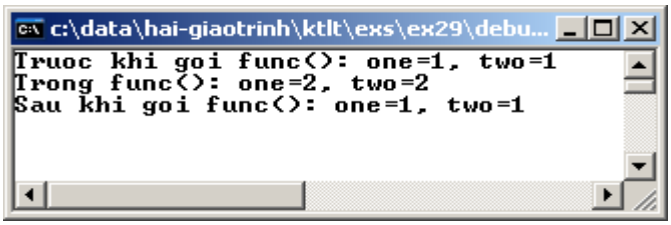
```
1   #include <stdio.h>
2   #include <conio.h>
3   int one=0;        // one là biến toàn cục
4   void func()
5   {
6       int two=2;    // two là biến cục bộ
7       one = two;    // one là biến toàn cục
8       printf("Trong func(): one=%d, two=%d\n",one,two);
9   }
10  int main()
11  {
12      int one, two;
13      one = 1;      // one là biến cục bộ
14      two = 1;      // two là biến cục bộ
15      printf("Truoc khi goi func(): one=%d, two=%d\n",one,two);
16      func();
17      printf("Sau khi goi func(): one=%d, two=%d\n",one,two);
18      getch();
19      return 0;
20  }
```

Trong chương trình này, biến toàn cục **one** được khai báo bên ngoài các hàm nên nó có thể sử dụng trong tất cả các hàm.

Biến cục bộ **one** được khai báo cục bộ bên trong hàm **main()**, nên biến tòan cục có cùng tên **one** sẽ bị "**che**", nghĩa là việc thay đổi giá trị của **one** trong **main()** sẽ không làm thay đổi giá trị của **one** ở ngoài **main()**.

Tuy có sự thay đổi giá trị của biến **two** trong hàm **func()**, nhưng sẽ không làm thay đổi giá trị của biến **two** trong **main()** vì chúng không có liên hệ gì với nhau vì được khai báo cục bộ trong 2 hàm khác nhau.

Kết quả chạy chương trình sẽ như sau:



Ví dụ 4.8: Chương trình sau cho phép nhập tổng số giây (second) và đổi sang giờ (hour), phút (minute), giây (second). Kết quả xuất có dạng hh:mm:ss

```
1   #include <stdio.h>
2   #include <conio.h>
3   int hour, minute, second; // Các biến toàn cục
4   int GetSecond();          // Hàm nhập tổng số giây
5   void Convert(int secs);   // Hàm đổi sang giờ, phút, giây
6   void ShowTime();          // Hàm hiển thị thời gian dạng hh:mm:ss
7   int main()
8   {
9       int totalsec;         // Biến cục bộ của hàm main()
10      totalsec = GetSecond();
11      Convert(totalsec);
12      ShowTime();
13      getch();
14      return 0;
15  }
16  int GetSecond()
17  {
18      int totalsec;         // Biến cục bộ của hàm GetSecond ()
```

```
19      printf("Total seconds: ");
20      scanf("%d", &totalsec);
21      return totalsec;
22  }
23  void Convert(int secs)
24  {
25      int temp;              // Biến cục bộ của hàm Convert()
26      second = secs%60;      // Giá trị biến toàn cục bị thay đổi
27      temp = secs/60;
28      minute = temp%60;
29      hour = temp/60;
30  }
31  void ShowTime()
32  {
33      printf("Time: %d:%d:%d", hour, minute, second);
34  }
```

Kết quả chạy chương trình:

4.3. ĐỆ QUY

4.3.1. Khái niệm

4.3.1.1. Khái niệm về đệ quy

Trong lập trình, việc gọi một hàm từ một hàm khác trong tầm vực cho phép như các ví dụ ở các bài trước là một xử lý rất bình thường. Tuy nhiên, trong thực tế có lúc trong một hàm sẽ có lời gọi hàm tới chính hàm đó, việc gọi đến chính hàm đó trong thân hàm được gọi là sự đệ quy (recursion).

Trong thực tế có nhiều bài tóan mà việc dùng giải thuật với cấu trúc lặp thông thường sẽ làm cho việc thiết kế giải thuật trở nên khó khăn, thậm chí khó có thể sử dụng cấu trúc lặp. Việc chuyển hướng từ giải thuật lặp sang giải thuật đệ quy nhiều khi sẽ làm cho việc thiết kế chương trình trở nên đơn giản hơn, gần với cách suy nghĩ và giải quyết tự nhiên.

Tuy vậy việc sử dụng đệ quy đôi khi cũng gây ra những điểm bất lợi như tiêu tốn nhiều tài nguyên máy (bộ nhớ trong/ngoài), làm cho chương trình chậm đi do việc sử dụng nhiều các tác vụ trên stack như push (thêm phần tử vào stack)và pop (lấy phần tử ra khỏi stack) để lưu giá trị và trạng thái hiện thời.

4.3.1.2. Cấu trúc của hàm đệ quy

Hàm đệ quy phải bao gồm 2 thành phần: thành phần dừng và thành phần đệ quy.

- Thành phần dừng xác định điểm dừng của sự đệ quy, hay nói cách khác, thành phần dừng là điều kiện để kết thúc giải thuật và trong đó không bao gồm lời gọi hàm tới chính nó.
- Thành phần đệ quy là thành phần mà trong đó một hàm sẽ được gọi bởi chính nó để có thể tìm ra giá trị cho các bước đệ quy sau.

4.3.2. Các ví dụ

4.3.2.1. Ví dụ 4.9: Bài tóan tính n!

Dùng giải thuật lặp

Chẳng hạn với số giai thừa ta có định nghĩa tóan học như sau:

$$n! = n \times (n-1) \times (n-2) \times ... \times 3 \times 2 \times 1 \quad \textbf{, với } n \geq 1 \textbf{ và } 0! = 1 \qquad \textbf{(1)}$$

Giả sử với n=5, theo định nghĩa ta có:

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

Theo công thức (1), n! sẽ được tính bằng cách lấy 1 nhân liên tiếp với số đếm (tăng từ 2 đến n) vì thế ta sẽ cài đặt hàm `Factorial()` dùng vòng lặp for như sau:

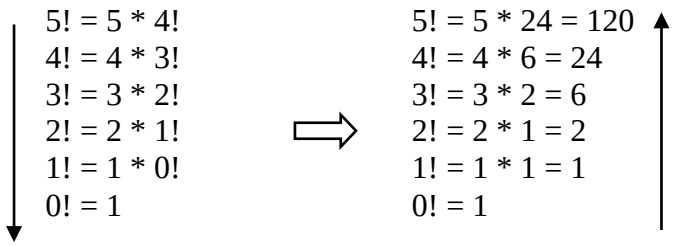
```
1   long Factorial(int n)
2   {
3       int i;
4       long product=1;
5       for(i=2; i<=n; i++)
6           product*=i;
7       return product;
8   }
```

Dùng giải thuật đệ quy

Từ công thức (1), ta thấy có thể biểu diễn n! theo dạng đệ quy như sau:

$n! = n \times (n-1)!$, **với** $n \geq 1$ **và** $0! = 1$ (2)

Với n=5, ta có:



Từ công thức (2) ta có hàm **`Factorial`** () được viết theo kiểu đệ quy như sau:

```
1   long Factorial(int n)
2   {
3       if(n==0) return 1;          // Thành phần dừng
4       return n*Factorial(n-1);    // Thành phần đệ qui
5   }
```

Ví dụ để tính **`Factorial(5)`**, quá trình gọi đệ quy sẽ được thực hiện như sau :

```
Factorial(5)    = 5*Factorial(4)
                = 5*4*Factorial(3)
                = 5*4*3*Factorial(2)
                = 5*4*3*2*Factorial(1)
                = 5*4*3*2*1*Factorial(0)
                = 5*4*3*2*1*1
                = 5
```

4.3.2.2. Ví dụ 4.10: Bài toán tính số hạng thứ n của dãy Fibonacci

Các số Fibonacci được định nghĩa như sau:

$$F1 = F2 = 1 \qquad (3)$$

$$Fn = Fn\text{-}1 + Fn\text{-}2, \text{ với } n>2$$

Chẳng hạn với n=6, ta có:

$F_6 = F_5 + F_4$ ⇒ $F_6 = 5 + 3 = 8$
$F_5 = F_4 + F_3$ ⇒ $F_5 = 3 + 2 = 5$
$F_4 = F_3 + F_2$ ⇒ $F_4 = 2 + 1 = 3$
$F_3 = F_2 + F_1$ ⇒ $F_3 = 1 + 1 = 2$
$F_2 = F_1 = 1$ ⇒ $F_2 = F_1 = 1$

Dùng giải thuật lặp

Theo công thức (3), để tính F_n thì ta phải tính liên tiếp F_3, F_4,…,F_{n-1}. Trong đó $F_i = F_{i-1} + F_{i-2}$ với i=3,4,…,n.

Gọi:

i là biến dùng để đếm số bước lặp (i=3,4,…,n)

fi tương ứng với giá trị F_i cần tìm.

a và **b** tương ứng với F_{i-2} và F_{i-1} là các giá trị liên tiếp trước F_i. Ban đầu **a=b=1.**

Tại mỗi bước lặp ta tính **fi = a + b** rồi cập nhật lại giá trị **a,b** là các giá trị F_{i-1} và F_i vừa tính được để dùng cho lần lặp sau.

Ta có hàm **Fib()** như sau:

```
1   long Fib(int n)
2   {
3       long fi=1, a=1, b=1;
4       for(int i=3; i<=n; i++)
5       {
6           fi = a + b;
7           a = b;
8           b = fi;
10      }
11      return fi;
12  }
```

Dùng giải thuật đệ quy:

Từ công thức (3) theo kiểu đệ quy ta dễ dàng xây dựng hàm **`Fib()`** dùng đệ quy như sau:

```
1   long Fib(int n)
2   {
3       if(n==1 || n==2)return 1; // Thành phần dừng
4       return Fib(n-1)+Fib(n-2); // Thành phần đệ qui
5   }
```

4.3.3. Các loại đệ quy

4.3.3.1. Đệ quy tuyến tính:

Một hàm được gọi là đệ quy tuyến tính hay đệ quy đuôi nếu trong thân hàm này có duy nhất một lời gọi hàm đến chính nó.

Ví dụ hàm **`Factorial()`** trong Ví dụ 4.9 là hàm đệ quy tuyến tính.

```
1  long Factorial(int n)
2  {
3      if(n==0) return 1;          // Thành phần dừng
4      return n*Factorial(n-1);    // Thành phần đệ qui
5  }
```

4.3.2.2. Đệ quy nhị phân

Một hàm được gọi là đệ quy nhị phân nếu trong thân hàm này có hai lời gọi hàm đến chính nó.

Ví dụ hàm **`Fib()`** trong ví dụ 4.10 là hàm đệ quy nhị phân.

```
1  long Fib(int n)
2  {
3      if(n==1 || n==2)return 1; // Thành phần dừng
4      return Fib(n-1)+Fib(n-2); // Thành phần đệ qui
5  }
```

4.3.2.3. Đệ quy phi tuyến (đệ quy phức)

Một hàm được gọi là đệ quy phi tuyến nếu trong thân hàm này có n lời gọi hàm đến chính nó (thông thường lời gọi hàm đến chính nó được đặt trong thân vòng lặp).

Ví dụ 4.11:

Viết hàm tính số hạng thứ n của dãy sau:

$$x_0 = 1,\ x_n = x_0 + x_1 + x_2 + \ldots + x_{n-1}$$

```
1  long X(int n){
2      if (n==0) return 1;
3      long s=0;
4      for (int i=0;i<=n-1;i++)
5          s=s+X(i);
6      return s;}
```

4.3.2.4. Đệ quy hỗ tương

Hai hàm được gọi là đệ quy hỗ tương nếu trong thân hàm này có lời gọi hàm đến hàm kia và ngược lại.

Ví dụ 4.12:

Viết các hàm tính số hạng thứ n của hai dãy (x) và (y) như sau:

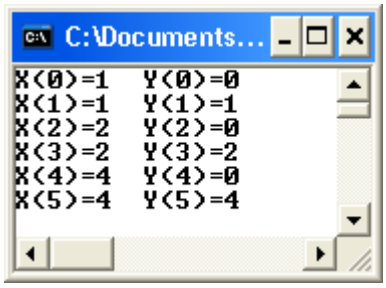
$x_0 = 1, y_0 = 1$

$x_n = x_{n-1} + y_{n-1}$

$y_n = x_{n-1} - y_{n-1}$

```
    …
1   long Y(int n);
2   long X(int n)
3   {
4       if (n==0) return 1;
5       return X(n-1)+Y(n-1);
6   }
7   long Y(int n)
8   {
9       if (n==0) return 0;
10      return X(n-1)-Y(n-1);
11  }
12  int main()
13  {
14      for (int i=0;i<=5;i++)
15          printf("X(%d)=%ld\tY(%d)=%ld\n",i,X(i),i,Y(i));
16      getch();
17      return 0;
18  }
```

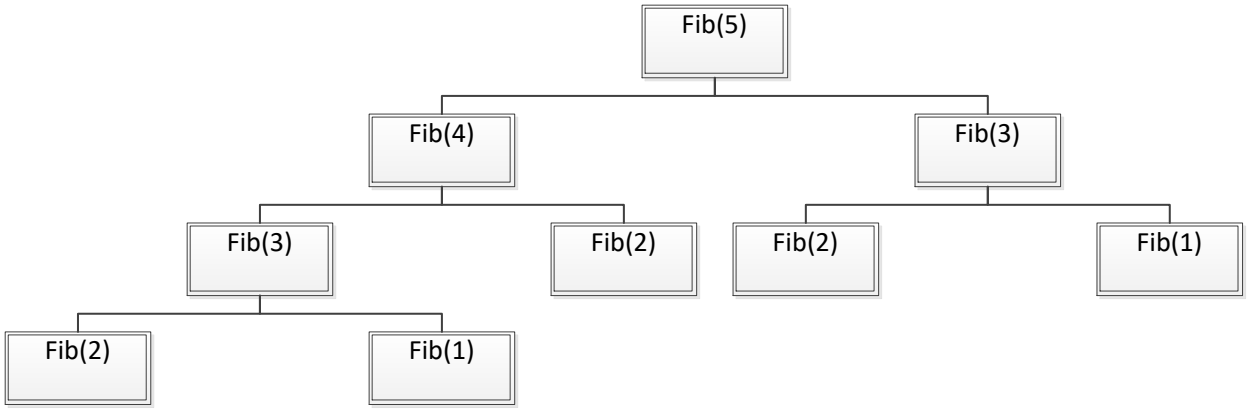
Kết quả khi chạy chương trình:



4.3.4. Khử đệ quy

Như đã trình bày trong phần 4.3.1.1, việc sử dụng đệ quy để cài đặt các hàm thường dễ dàng hơn cho người lập trình nhưng thường làm chương trình chạy chậm hơn.

Ví dụ trong Ví dụ 4.10, để tính hàm Fib (5) dùng đệ quy thì chương trình sẽ phải gọi hàm đệ quy 9 lần trong đó Fib(3) gọi 2 lần, Fib(2) gọi 3 lần, Fib(1) gọi 2 lần. Trong khi đó nếu dùng vòng lặp như phần 3.2.2.a thì chỉ cần lặp 3 lần.



Vì thế trong lập trình người ta luôn hướng tới việc sử dụng các hàm không đệ quy. Việc chuyển một hàm đệ quy thành không đệ quy được gọi là khử đệ quy.

Ví dụ 4.13 : Hàm X() trong ví dụ 4.11 có thể khử đệ quy như sau:

```
1   long X(int n) {
2       long s=1;
3       for (int i=2;i<=n-1;i++)
4           s=s*2;
5       return s;}
```

Giá trị của hàm X() ứng với các giá trị n như sau:

n	0	1	2	3	4	5	…
X(n)	1	1	2	4	8	16	…

Ví dụ 4.14 : Hàm X(), Y() trong ví dụ 4.12 có thể khử đệ quy như sau:

```
1   Long X(int n)
2   {
3       long s=1;
4       for (int i=1;i<=n;i=i+2)
5           s=s*2;
6       return s;
7   }
8   long Y(int n)
9   {
10      if (i%2!=0)
11          return 0;
12      long s=1;
13      for (int i=1;i<=n/2;i++)
14          s=s*2;
15      return s;
16  }
```

Giá trị của hàm X(), Y() ứng với các giá trị n như sau:

N	0	1	2	3	4	5	…
X(n)	1	2	2	4	4	8	…
Y(n)	1	0	2	0	4	0	

BÀI TẬP CHƯƠNG 4

1. Gọi a, b, c là độ dài các cạnh của một tam giác. Diện tích S của tam giác được tính theo công thức:

$S = \sqrt{p(p-a)(p-b)(p-c)}$ trong đó: $p = \frac{a+b+c}{2}$

Hãy viết hàm **`Triangle()`** và chương trình để tính diện tích tam giác.

2. Viết chương trình xuất ra màn hình một thực đơn cho phép chọn 1 trong 2 tác vụ:

a. Nhập tổng số thời gian tính bằng giây, đổi sang dạng: **`hour:minute:second`**

b. Nhập vào thời gian dạng hour:minute:second, đổi sang thời gian bằng giây

Trong chương trình, cần thiết kế các hàm sau:

```
int Menu(); //Xuất thực đơn và chọn mục trong thực đơn
void Sec2Time(long sec);  //Chuyển số giây sang dạng thời gian
long Time2Sec(int hour, int minute, int second);
//Chuyển thời gian sang số giây
```

3. Hãy viết một chương trình có gọi sử dụng các hàm sau:

a. Hàm tính n!

```
long Giaithua(int n);
```

b. Hàm hoán vị 2 số

```
void Swap(float &a,float &b);
```

c. Hàm in n dấu *

```
void InSao(int n);
```

d. Hàm kiểm tra số nguyên tố.

```
int KTNTo(int n);//trả về 1 nếu n nguyên tố, ngược lại trả về 0
```

e. Hàm đếm số chữ số của số n.

```
int SoChuSo(int n);//ví dụ số 231 có 3 chữ số
```

f. Hàm tìm ước số chung lớn nhất của 2 số.

```
int UCLN(int a, int b);
```

4. Cho biết kết quả khi chạy các chương trình sau:

a)

```
int main(void)
{
   int x=0;
   change(x);
   printf("%d", x);
   getch() ;
   return 0 ;
}
```

```
void change(int y)
{
   y=1;
}
```

b)

```
int x;
void main(void)
{
   int x=0, y=0;
   printf("%d  %d\n", x, y);
   change(x);
   printf("%d  %d\n", x, y);
   getch() ;
   return 0 ;
}
```

```
void change(int y)
{
   x = 1;
   y = x+1;
   printf("%d %d\n", x, y);
   x = y;
}
```

5. Tạo file **"Mylib.h"** chứa các hàm trong bài tập 1.3 và sử dụng thư viện này viết các chương trình sau:

g. a. In ra hình chữ nhật gồm a dòng b cột.

h. Cho người dùng nhập 2 số nguyên dương n và k (có kiểm tra điều kiện nhập). Nếu n < k thì hoán vị 2 giá trị này. Sau đó tính tổ hợp chập k của n phần tử

Biết: $C_n^k = \frac{n!}{k!(n-k)!}$ hay $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, với 0≤ k≤ n

c. Cho người dùng nhập vào n số nguyên dương và in ra số chữ số của từng số nguyên đồng thời đếm xem có bao nhiêu số nguyên tố trong dãy số vừa nhập.

Ví dụ:

```
n = 5
Số thứ 1: 3
Có 1 chữ số
Số thứ 2: 11
Có 2 chữ số
Số thứ 3: 122
Có 3 chữ số
Số thứ 4: 0
Có 1 chữ số
Số thứ 5: 12
Có 2 chữ số
Có tất cả 2 số nguyên tố.
```

d. Cho người dùng nhập vào n số nguyên dương và in ra max , min của dãy số này sau đó in ra ước số chung lớn nhất của max và min.

6. Viết các hàm sau dùng giải thuật đệ quy :

a. Tính $S_n = n + S_{n-1}$

b. Hàm in n dấu *

c. Hàm đếm số chữ số của số n.

d. Hàm tìm ước số chung lớn nhất của 2 số.

7. Viết các hàm sau bằng 2 cách đệ quy và không đệ quy :

a. Hàm in ra màn hình các giá trị giảm dần từ n đến 1.

b. Hàm tính tổ hợp chập m của n phần tử

Biết: $C_n^k = \frac{n!}{k!(n-k)!}$ hay $C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$, với 0≤ k≤ n

8. Viết các hàm sau bằng cách đệ quy :

a. Đổi một số thập phân sang số nhị phân

b. Đổi một số thập phân sang số bát phân

c. Đổi một số thập phân sang số thập lục phân

9. Hãy viết chương trình đếm số lần gọi đệ quy của các hàm trong bài 1.7, 1.8

10. Giả sử lãi suất tiết kiệm kỳ hạn 1 tháng là 5 % tháng (không phân biệt số ngày của tháng, không cho phép rút tiền trước hạn). Sau mỗi tháng nếu người gửi không rút lãi thì lãi sẽ được nhập vốn để tính cho tháng sau. Hãy viết chương trình cho phép người dùng nhập vào số tiền gửi, số tháng gửi và in ra số tiền nhận được gồm cả vốn và lãi. Yêu cầu viết hàm tính bằng 2 cách đệ quy và không đệ quy.

Ví dụ:

```
Số tiền gửi (đơn vị tính 1.000 VND)= 100000
Thời gian gửi: 3 tháng
Số tiền nhận được (đơn vị tính 1.000 VND): 101255.2
```

CHƯƠNG 5: MẢNG VÀ CHUỖI

5.1. MẢNG

5.1.1. Khái niệm

Mảng (array) là một tập hợp (set) hay một danh sách (list) các mục dữ liệu (data items) còn gọi là phần tử mảng (array element) với cùng một kiểu dữ liệu.

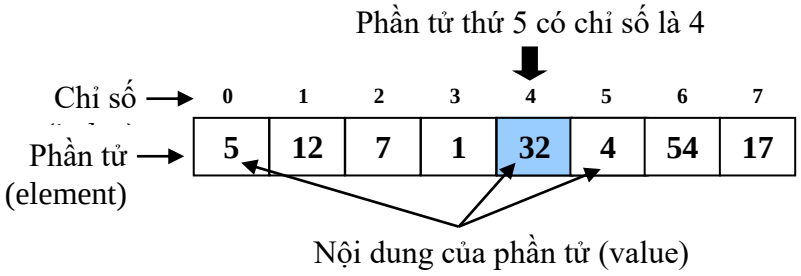
Chẳng hạn một mảng 10 số nguyên kiểu int là một tập gồm 10 phần tử, mỗi phần tử thuộc kiểu int. Ta không thể trộn lẫn các phần tử của một mảng với nhiều kiểu dữ liệu khác nhau, chẳng hạn ta không thể gán phần tử thứ nhất là int, phần tử thứ hai là float, phần tử thứ ba lại là int,...

Trong C/C++, một mảng khi được khai báo sẽ được cấp một khối bộ nhớ liên tục. Ngòai ra kích thước của mảng một khi đã được khai báo sẽ là cố định và không thể thay đổi được. (Trong một số ngôn ngữ lập trình khác, chẳng hạn Basic, cho phép xác định lại kích thước mới cho mảng thông qua chỉ dẫn redim)

Tùy theo yêu cầu dùng mảng trong bài tóan cụ thể, mảng có thể được khai báo là mảng 1 chiều (1 dimension) gồm 1 chỉ số, 2 chiều (2 dimensions) với một cặp 2 chỉ số, hoặc nhiều hơn 2 chiều.

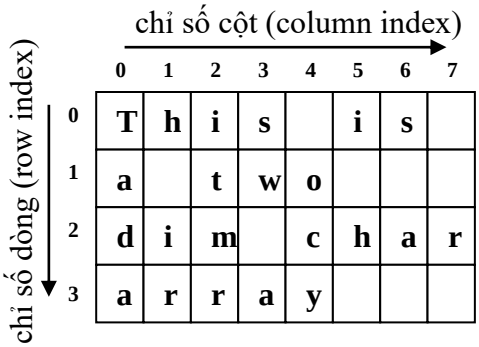
Ví dụ 5.1:

– Mảng các số nguyên với 8 phần tử được đánh chỉ số từ 0 đến 7:



Hình 5.1: Minh họa mảng 1 chiều chứa các số nguyên

– Mảng các ký tự `char` gồm 4 dòng (0..3) và 8 cột (0..7):



Hình 2.2: Minh họa mảng 2 chiều chứa các ký tự

Do vậy, chẳng hạn để chứa một dãy gồm 100 số nguyên, thay vì phải khai báo 100 biến có cùng kiểu int, thì ta chỉ cần khai báo một mảng kiểu int có kích thước là 100 phần tử. Để tham khảo đến 1 phần tử bất kỳ trong mảng, ta cần xác định chỉ số tương ứng thuộc phạm vi 0..99.

5.1.2. Mảng một chiều

Mảng 1 chiều là một danh sách các phần tử sắp liên tục sao cho mỗi phần tử ứng với 1 vị trí trong mảng gọi là chỉ số hay chỉ mục (index). Chỉ số của mảng luôn được đánh số từ 0.

Một danh sách gồm 50 sinh viên được đánh chỉ số từ 0 đến 49. Danh sách các tháng trong năm được đánh chỉ số từ 0 đến 11,...

5.1.2.1. Khai báo mảng 1 chiều

```
<Kiểu dữ liệu> <Tên mảng>[<Kích thước>];
```

Trong đó:

<Kiểu dữ liệu> (type): Kiểu dữ liệu của các phần tử trong mảng, có thể là kiểu cơ sở như char, int, float,... cũng có thể là kiểu struct, union hoặc kiểu mới do người dùng định nghĩa.

<Tên mảng> (name): Tên biến mảng theo qui ước đặt tên của C/C++

<Kích thước> (size): Kích thước tối đa của mảng, là một số nguyên dương.

Ví dụ 5.2:

`int a[100];` // Mảng tối đa 100 phần tử là số nguyên

`float b[50];` // Mảng tối đa 50 phần tử là số thực

5.1.2.2. Khai báo và khởi tạo giá trị ban đầu cho mảng 1 chiều

Khi đó ta không cần khai báo số phần tử tối đa của mảng.

Ví dụ 5.3:

`int a[]={1,5,7};` // Mảng ban đầu có 3 phần tử là 1,5,7

```
char name[] = {'F','e','r','n','a','n','d','o'};
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
name	F	e	r	n	a	n	d	o	\0						

5.1.2.3. Truy xuất phần tử của mảng

Cách tham khảo một phần tử mảng:

<Tên mảng>[<Chỉ số>]

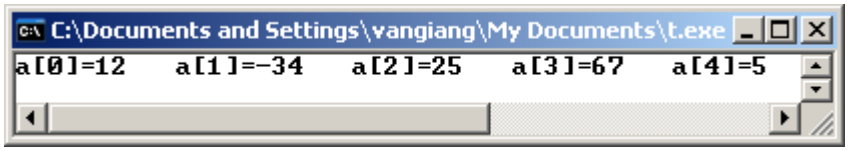
Ví dụ 5.4: Với khai báo : **`int a[100];`**

- Phần tử đầu tiên của mảng là a[0], phần tử tiếp theo là a[1], …phần tử có chỉ số thứ i là a[i]
- Gán giá trị 20 cho phần tử `a[2]`: **`a[2]=20;`**
- In phần tử thứ i: **`printf("%d",a[i]);`**
- Nhập phần tử thứ i: **`scanf("%d",&a[i]);`**

Ví dụ 5.5: Chương trình sau sẽ khai báo 2 biến mảng lần lượt là a và b có kiểu int, gán giá trị của mảng b cho mảng a và xuất mảng a ra màn hình.

```
1   …
2   int main()
3   {
4       int a[5],b[5]={12,-34,25,67,5};
5       // Gán giá trị của mảng a cho mảng b
6       for(int i=0; i<5; i++)
7           a[i]=b[i];
8       // Xuất mảng a
9       for(int i=0; i<5; i++)
10          printf("a[%d]=%d, ", i, a[i]);
11      getch();
12      return 0;
13  }
```

Kết quả chạy chương trình:



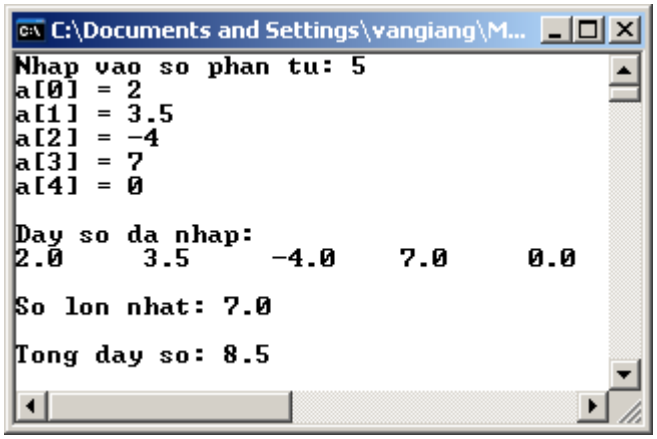
Ví dụ 5.6:

Chương trình sau cho phép nhập mảng gồm n số thực (số n do người dùng nhập vào) sau đó xuất mảng đã nhập, số lớn nhất và tổng các phần tử của mảng.

```
    …
1   #define N 100
2   void Input(float a[], int &n)  // Nhập mảng
3   {
4       printf("Nhap vao so phan tu: "); scanf("%d",&n);
5       for(int i=0; i<n; i++)
6       {
7           printf("a[%d] = ", i);  scanf("%f", &a[i]);
8       }
9   }
10  void Output(float a[], int n) // Xuất mảng
11  {
12      printf("\nDay so da nhap:\n");
13      for(int i=0; i<n; i++)
4           printf("%.1f\t", a[i]);
15      printf("\n");
16  }
17  float MaxArray(float a[], int n) // Tìm số lớn nhất
18  {
19      float max= a[0];
20      for(int i=1; i<n; i++)
21          if(max<a[i]) max = a[i];
22      return max;
23  }
```

```
24  float SumArray(float a[], int n) // Tính tổng các phần tử
25  {
26      float sum=0;
27      for(int i=0; i<n; i++)
28          sum+=a[i];
29      return sum;
30  }
31  int main(){
32      float a[N], sum;
33      int n;
34      Input(a,n);
35      Output(a,n);
36      printf("\nSo lon nhat: %.1f\n", MaxArray(a,n));
37      printf("\nTong day so: %.1f", SumArray(a,n));
38      getch();
39      return 0;
40  }
```

Kết quả chạy chương trình:



Ví dụ 5.7:

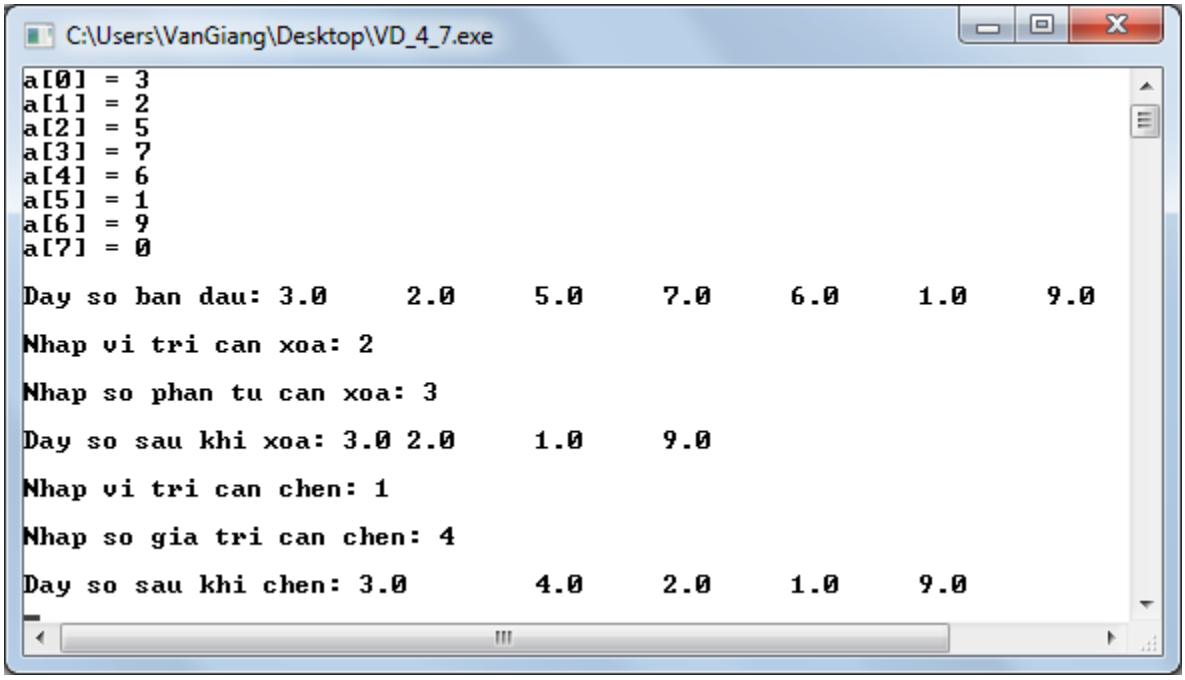
Chương trình sau cho phép nhập mảng gồm các số thực dương (kết thúc nhập khi người dùng nhấn số 0). Sau đó cho phép xóa d phần tử của mảng tại vị trí k cho trước và chèn một số x vào mảng tại vị trí k.

```
    …
1   #define N 100
2   void Input(float a[], int &n)  // Nhập mảng
3   {
4       n=0;
5       while (1)
6       {
7           do
8           {
9               printf("a[%d] = ", n);    scanf("%f", &a[n]);
10          }while (a[n]<0);
11          if (a[n]==0) break;
12          else n++;
13      }
15  }
16  void Output(float a[], int n) // Xuất mảng
17  {
18      for(int i=0; i<n; i++)
19          printf("%.1f\t", a[i]);
20      printf("\n");
21  }
```

```
    //Xóa d phần tử liên tiếp của mảng ở vị trí k
22  void DeleteArray(float a[], int &n, int k, int d)
23  {
24      for(int i=k; i<n-d; i++)
25              a[i]=a[i+d];
26      n=n-d;
27  }
28  //Chèn phần tử x vào mảng ở vị trí k
29  void InsertArray(float a[], int &n, int k, int x)
30  {
31      for(int i=n; i>k; i--)
32          a[i]=a[i-1];
33      a[k]=x;
34      n++;
35  }
36  int main()
37  {
38      float a[N], x;
39      int n,d,k;
40      Input(a,n);
41      printf("\nDay so ban dau: ");
42      Output(a,n);

43      printf("\nNhap vi tri can xoa: "); scanf("%d",&k);
44      printf("\nNhap so phan tu can xoa: "); scanf("%d",&d);
45      DeleteArray(a,n,k,d);
```

```
46      printf("\nDay so sau khi xoa: ");
47      Output(a,n);


48      printf("\nNhap vi tri can chen: "); scanf("%d",&k);
49      printf("\nNhap so gia tri can chen: "); scanf("%f",&x);
50      InsertArray(a,n,k,x);
51      printf("\nDay so sau khi chen: ");
52      Output(a,n);


53      getch();
54      return 0;
    }
```

Kết quả chạy chương trình:



5.1.3. Mảng nhiều chiều

Mảng nhiều chiều (multi-dimension array) là một mảng với số chiều lớn hơn 1. Đối với mảng n-chiều sẽ có bộ n chỉ mục để tham khảo đến một phần tử của mảng. Thông thường thì mảng 2 chiều sẽ được dùng để lưu và xử lý dữ liệu dạng bảng hoặc xử lý ma trận.

Cụ thể với mảng 2 chiều ta cần bộ 2 chỉ số (row, column). Nói cách khác mảng 2 chiều là một bảng gồm có n dòng và m cột. Mỗi dòng trong mảng 2 chiều là một mảng 1 chiều.

Trong giáo trình này ta chỉ xét mảng tối đa 2 chiều.

5.1.3.1. Khai báo mảng 2 chiều

```
<Kiểu dữ liệu> <Tên mảng>[<Số dòng tối đa>][<Số cột tối đa>];
```

Ví dụ 5.7:

Khai báo mảng 2 chiều chứa số nguyên tối đa 10 dòng, 20 cột:

```
int a[10][20];
```

Khai báo một biến mảng 2 chiều để lưu một danh sách gồm 20 sinh viên, mỗi dòng chứa họ tên một sinh viên với chiều dài tối đa 30 ký tự:

```
char hoten[20][30];
```

Hoten	0	1	2			3	...	29
0	L	e		V	a	n	S	...	
1	D	i	n	h		B	a	o	...	
	...									
19	P	h	a	m		T	r	u	...	

5.1.3.2. Khai báo và khởi tạo giá trị cho mảng 2 chiều

Ví dụ 5.8:

Khai báo và khởi động giá trị cho biến mảng 2 chiều có tên a dùng để chứa một bảng các số nguyên gồm 3 dòng và 5 cột, hay nói cách khác a là một ma trận có kích thước 3x5:

```
int a[3][5] = {{1,0,1,0,0}, {0,0,1,1,0}, {1,1,0,0,1}};
```

a	0	1	2	3	4
0	1	0	1	0	0
1	0	0	1	1	0
2	1	1	0	0	1

5.1.3.3. Tham khảo phần tử của mảng 2 chiều:

```
<Tên mảng>[<Chỉ số dòng][<Chỉ số cột>];
```

Ví dụ 5.9:

Với mảng a như hình bên

- Ta có : **a[1][3]=0**
-
- Gán giá trị **2** cho phần tử a[2][3]: **a[2][3]=2;**
- In phần tử dòng **i**, cột **j**: **printf("%d",a[i][j]);**
- phần tử dòng **i**, cột **j**: **scanf("%d",&a[i][j]);**

	0	1	2	3	4
0	1	3	5	7	7
1	2	3	0	3	1
2	1	9	1	2	5
3	5	9	8	4	7

Ví dụ 5.10:

Chương trình sau cho phép nhập vào mảng 2 chiều gồm n dòng và m cột chứa các số thực. Sau đó thực hiện các thao tác:

- Xuất ma trận.

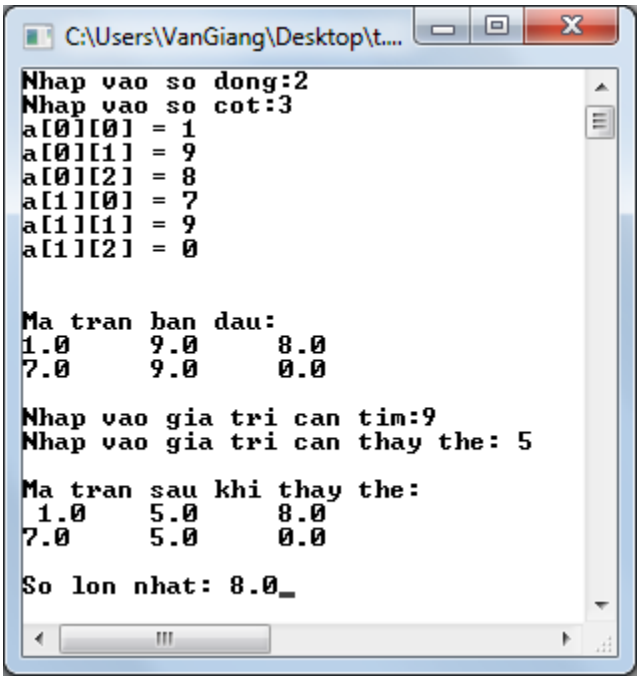
- Thay thế tất cả các giá trị x có trong ma trận bằng giá trị y.
- Tìm phần tử lớn nhất của ma trận.

```
    …
1   #define ROW 10
2   #define COL 10
3   void GetMatrix(float a[ROW][COL],int &m, int &n) //Nhập ma trận
4   {
5       int i, j;
6       printf("Nhap vao so dong:");scanf("%d",&m);
7       printf("Nhap vao so cot:");scanf("%d",&n);
8       for(i=0; i<m; i++)
9           for(j=0; j<n; j++)
10          {
11              printf("a[%d][%d] = ", i, j);scanf("%f", &a[i][j]);
12          }
13      printf("\n\n");
14  }
15  void PrintMatrix(float a[ROW][COL], int m, int n) //Xuất ma trận
16  {
17      for(int i=0; i<m; i++)
18      {
19          for(int j=0; j<n; j++)
20              printf("%.1f\t", a[i][j]);
21          printf("\n");
22      }
23  }
```

```
    //Thay thế phần tử của ma trận
24  void ReplaceMatrix(float a[ROW][COL], int m, int n, float x,
                                                          float y)
25  {
26      for(int i=0; i<m; i++)
27          for(int j=0; j<n; j++)
28              if (a[i][j]==x) a[i][j]=y;
29  }
    //Tìm phần tử lớn nhất trong ma trận
30  float MaxMatrix(float a[ROW][COL], int m, int n)
31  {
32      float max=a[0][0];
33      for(int i=0; i<m; i++)
34          for(int j=0; j<n; j++)
35              if (a[i][j]>max) max=a[i][j];
36      return max;
37  }
38  int main()
39  {
40      float a[ROW][COL], x, y;
41      int m, n;
42      GetMatrix(a,m,n);
43      printf("Ma tran ban dau:\n");
44      PrintMatrix(a,m,n);
45      printf("Nhap vao gia tri can tim:");scanf("%f",&x);
46      printf("Nhap vao gia tri can thay the: ");scanf("%f",&y);
```

```
47      ReplaceMatrix(a,m,n,x,y);
48      printf("Ma tran sau khi thay:\n ");
49      PrintMatrix(a,m,n);
50      printf("So lon nhat: %.1f", MaxMatrix(a,m,n));
51      getch();
52      return 0;
53  }
```

Kết quả chạy chương trình:



Ví dụ 5.11:

Chương trình sau cho phép nhập vào 2 ma trận **A, B** chứa các số nguyên và tính ma trận tổng **C** của chúng.

Biết phần tử dòng i cột j của ma trận **C** được tính theo công thức:

c[i][j]=a[i][j]+b[i][j] (với a[i][j], b[i][j] là các phần tử dòng i, cột j của ma trận **A** và **B**)

Chẳng hạn cho 2 ma trận **A** và **B**:

$$A=\begin{bmatrix}1 & 0 & 2\\ 1 & 3 & 3\end{bmatrix} \qquad B=\begin{bmatrix}2 & 1 & 1\\ 0 & 0 & 1\end{bmatrix}$$

Thì ma trận tổng **C= A+B** sẽ là $C=\begin{bmatrix}3 & 1 & 3\\ 1 & 3 & 4\end{bmatrix}$

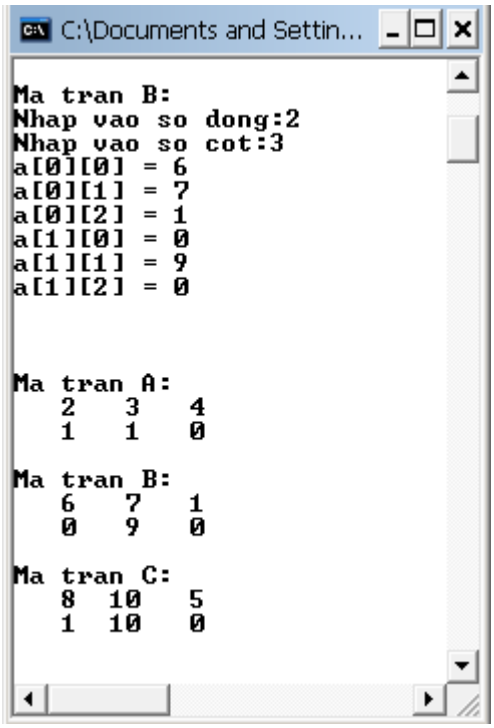
Trong các hàm **GetMatrix()** và **PrintMatrix()** ngòai tham số là mảng 2 chiều, còn có tham số **name** là mảng một chiều các ký tự. Mảng **name** được dùng để nhận vào tên của ma trận tương ứng được nhập hay xuất được gọi trong hàm **main()**.

```
    …
1   #define ROW 10
2   #define COL 10
3   void GetMatrix(char name[], int a[ROW][COL],int &m, int &n)
4   {
5       int i, j;
6       printf("\nMa tran %s:\n", name);
7       printf("Nhap vao so dong:");scanf("%d",&m);
8       printf("Nhap vao so cot:");scanf("%d",&n);
9       for(i=0; i<m; i++)
10          for(j=0; j<n; j++)
11          {
12              printf("a[%d][%d] = ", i, j);scanf("%d", &a[i][j]);
13          }
14      printf("\n\n");
15  }
```

```
16  void AddMatrix(int c[ROW][COL], int a[ROW][COL], int b[ROW][COL],
                                                      int m, int n)
17  {
18      int i, j;
19      for(i=0; i<m; i++)
20          for(j=0; j<n; j++)
21              c[i][j] = a[i][j] + b[i][j];
22  }
23  void PrintMatrix(char name[], int a[ROW][COL], int m, int n)
24  {
25      printf("\nMa tran %s:\n", name);
26      for(int i=0; i<m; i++)
27      {
28          for(int j=0; j<n; j++)
29              printf("%4d", a[i][j]);
30          printf("\n");
31      }
32  }
33  int main()
34  {
35      int a[ROW][COL], b[ROW][COL], c[ROW][COL],ma,na,mb,nb;
36      GetMatrix("A", a,ma,na);
37      GetMatrix("B", b,mb,nb);
38      PrintMatrix("A", a,ma,na);
39      PrintMatrix("B", b,mb,nb);
```

```
40      if (ma==mb&&na==nb)
41      {
42           AddMatrix(c, a, b,m,n);
43           PrintMatrix("C", c,m,n);
44      }
45      else printf("Không tính được tổng 2 ma trận A&B");
46      getch();
47      return 0;
48  }
```

Kết quả chạy chương trình:



5.2. CHUỖI

5.2.1. Khái niệm

Chuỗi (string) thực chất là một mảng các ký tự (array of characters).

Nếu quan niệm là mảng các ký tự ta có thể không quan tâm đến ký tự kết thúc chuỗi '\0' có mã ASCII là 0. Khi đó việc truy cập đến từng phần tử của mảng ký tự tương tự như việc truy cập một mảng số nguyên thông thường bằng việc dùng các chỉ số. Tuy nhiên khi dùng các hàm xử lý chuỗi sẽ gây ra hiện tượng chuỗi được xử lý sẽ bao gồm "rác" (garbage), tức là các ký tự bất kỳ không biết trước và không mong muốn trong chuỗi.

Nếu quan niệm là chuỗi (string), ta cần quan tâm đến ký tự kết thúc chuỗi '\0'. Khi đó chuỗi có độ dài thực (không gồm "rác") được xác định căn cứ vào ký tự phân cách (delimeter character) là ký tự zero '\0'.

Ví dụ 5.12:

Với chuỗi **word** có độ dài không xác định trước được khai báo và khởi động giá trị:

```
char word[]=
{'H','e','l','l','o',',',' ','W','o','r','l','d','!'};
```

thì hình ảnh của chuỗi **word** sẽ như hình dưới đây. Sau ký tự `'!'` là các ký tự không xác định hay còn gọi là "rác" :

	0	1	2	3	4	5	6	7	8	9	10	11		
word	**H**	**e**	**l**	**l**	**o**	**,**		**W**	**o**	**r**	**l**	**d**	**!**	...

Để in chuỗi như thế ta phải biết số ký tự của chuỗi và in từng ký tự như thao tác trên mảng. Còn với câu lệnh : **`printf("%s", word);`**

thì chuỗi sẽ được xuất trên màn hình như sau:

5.2.2. Khai báo chuỗi

Chuỗi được khai báo tương tự như khai báo mảng thông thường. Trong trường hợp khởi tạo giá trị ban đầu cần quan tâm đến ký tự kết thúc chuỗi.

Ví dụ 5.13:

Khai báo chuỗi **word** có tối đa 50 ký tự : **char word[50];**

Nếu cần khởi tạo giá trị ban đầu có thể dùng một trong các cách sau:

```
char word[50]=
{'H','e','l','l','o',',',' ','W','o','r','l','d','!','\0'};
char word[]=
{'H','e','l','l','o',',',' ','W','o','r','l','d','!','\0'};
char word[50]="Hello, Word!";
char word[]="Hello, Word!";
```

5.2.3. Các hàm xử lý chuỗi

5.2.3.1. Hàm nhập chuỗi

Hàm **gets()** có prototype định nghĩa trong tập tin header **<stdio.h>**

```
gets(<biến chuỗi>);
```

Ví dụ 5.14:

```
char name[50];
printf("Nhap chuoi ho ten: "); gets(name);
```

Lưu ý:

- Nếu trước khi dùng hàm **gets()** có sử dụng hàm **scanf()** hay **cin** thì phải làm sạch bộ đệm bàn phím bằng hàm: **fflush(stdin)**

Ví dụ 5.15:

```
char name[50];
int ID;
printf("Nhap ma so:"); scanf("%d",&ID);
printf("Nhap chuoi ho ten: "); fflush(stdin); gets(name);
```

- Ta cũng có thể dùng hàm **scanf()** hay **cin** để nhập chuỗi:

Ví dụ 5.16:

```
char name[50];
printf("Nhap chuoi: ");
scanf("%[A-Aa-z ]",name);//cin.getline(name,50);
//nếu dùng scanf("%s",name); hay cin>>name; sẽ không nhận được //khoảng trắng
```

5.2.3.2. Hàm xuất chuỗi

Hàm **puts()** có prototype định nghĩa trong tập tin header **<stdio.h>**

```
puts(<chuỗi>);          //Sau khi xuất xong tự động xuống dòng
```

Ví dụ 5.17:

```
char name[50]= "Ly Tu Trong";
printf("Chuoi da khoi tao: "); puts(name);
```

Lưu ý:

Ta cũng có thể dùng hàm **printf()** hay **cout** nhưng sau khi xuất không tự động xuống dòng.

Ví dụ 5.18:

```
char name[50]= "Ly Tu Trong";
```

```
printf("Chuoi da khoi tao:%s",name);
//cout<<"Chuoi da khoi tao :"<<name;
```

5.2.3.3. Hàm tính chiều dài chuỗi

Các hàm xử lý trên chuỗi từ phần này trở đi đều có prototype định nghĩa trong tập tin <string.h>, ngoại trừ các hàm trong mục 11,12

```
int strlen(<chuỗi>);  //Trả về chiều dài chuỗi
```

Ví dụ 5.19:

```
char name[50]="Ly Tu Trong";
int len = strlen(name);  //→  len = 11
```

5.2.3.4. Hàm sao chép chuỗi

```
strcpy(<chuỗi đích>,<chuỗi nguồn>);
// Chép <chuỗi nguồn> vào <chuỗi đích>
```

Ví dụ 5.20:

Chép chuỗi "Hello, World!" trong biến s1 vào biến s2

```
char s1[]="Hello, World!", s2[30];
strcpy(s2, s1);
printf("%s", s2);   //→  s2 ="Hello, World!"
```

5.2.3.5. Hàm nối chuỗi

```
strcat(<chuỗi đích>,<chuỗi nguồn>);
// Nối <chuỗi nguồn> vào cuối <chuỗi đích>
```

Ví dụ 5.21: Ghép 2 chuỗi trong các biến s1 và s2 vào biến s3 theo thứ tự đó

```
char s1[]="Hello", s2[]=", World!", s3[30]={\0};
strcat(s3, s1);
```

```
strcat(s3, s2);
printf("%s", s3);   //→  s3 = "Hello, World! "
```

5.2.3.6. Hàm so sánh chuỗi

```
int strcmp(s1,s2);  //   Trả về 0 nếu s1 = s2,
                    //   Trả về số dương nếu s1 > s2,
                    //   Trả về số âm nếu s1 < s2
```

Hàm ***strcmp()*** sẽ so sánh theo mã ASCII từng ký tự trong chuỗi cho đến khi hết chuỗi hoặc gặp ký tự khác nhau

Ví dụ 5.22:

```
char s1[]="Lan", s2[]="lan";
kq=strcmp(s1,s2) // → kq= -1
char s1[]="Nguyen Tuan", s2[]="Nguyen Thi Lan";
kq=strcmp(s1,s2) // → kq= 1
```

5.2.3.7. Hàm đổi chuỗi sang chữ hoa

```
strupr(<chuỗi>);
// Đổi các ký tự trong  <chuỗi> sang dạng viết hoa.
```

Ví dụ 5.23:

```
char name[]="Ly TU Trong";
strupr(name);
printf("%s",name); //→  name =  "LY TU TRONG"
```

5.2.3.8. Hàm đổi chuỗi sang chữ thường

```
strlwr(<chuỗi>);

//Đổi các ký tự trong <chuỗi>  sang dạng viết thường.
```

Ví dụ 5.24:

```
char name[]="Ly TU Trong";

strlwr(name);

printf("%s",name); //→  name =  "ly tu trong"
```

BÀI TẬP CHƯƠNG 5

1. Viết chương trình thực hiện các chức năng sau (mỗi chức năng viết một hàm riêng):

a. Nhập mảng các số thực, kết thúc nhập khi người dùng nhập 9999.

b. Xuất mảng các số thực.

c. Tính trung bình cộng các số thực có trong mảng.

d. In ra 2 dòng, một dòng gồm các số thực dương, một dòng gồm các số thực âm.

e. Đếm số lần xuất hiện của số x.

Ví dụ với dãy số: 1.0, -3.5, 6.7, 1.0, 8.7, 12.32, 0.58, 6.7, 12.5, -1.0, 6.7, 4.15

thì số lần xuất hiện của x=6.7 là 3.

f. Tìm giá trị min của mảng

g. Xóa tất cả các phần tử có giá trị bằng x trong mảng.

h. Đảo ngược các phần tử của mảng.

k. Kiểm tra mảng có là mảng tăng dần không

Ví dụ:

Mảng gồm các số : 1, 2, 4, 6, 7 là mảng tăng dần

Mảng gồm các số: 1, 2, 2, 6, 7 không là mảng tăng dần

Mảng gồm các số: 1, 2, 6, 4, 7 không là mảng tăng dần

l. Chèn x vào mảng tăng dần để mảng vẫn là mảng tăng dần.

m. Kiểm tra mảng có là mảng đối xứng không

Ví dụ:

Mảng gồm các số : 1, 2, 4, 2, 1 là mảng đối xứng

Mảng gồm các số : 1, 2, 2, 1 là mảng đối xứng

2. Viết chương trình thực hiện các chức năng sau (mỗi chức năng viết một hàm riêng):

a. Nhập mảng gồm n>0 các số nguyên dương (có kiểm tra dữ liệu nhập).

b. Xuất mảng gồm n>0 các số nguyên dương.

c. In ra các số nguyên tố có trong mảng.

d. Tìm giá trị max của các số chẵn trong mảng.

e. Tìm ước số chung lớn nhất của từng cặp số trong mảng.

Ví dụ với mảng gồm 4 số: 2, 6, 4, 9 thì in ra:

USCLN(2,6)=2

USCLN(2,4)=2

USCLN(2,9)=1

USCLN(6,4)=2

USCLN(6,9)=3

USCLN(4,9)=1

f. In ra biểu đồ nằm ngang của các số trong mảng.

Ví dụ với mảng gồm 4 số: 2, 6, 4, 9 thì in ra:

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

g. In ra biểu đồ đứng của các số trong mảng.

Ví dụ với mảng gồm 4 số: 2, 6, 4, 9 thì in ra:

```
         *
```

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

h. Sao chép các số chẵn trong mảng vào một mảng khác.

3. Viết chương trình thực hiện các chức năng sau (mỗi chức năng viết một hàm riêng):

a. Nhập một ma trận số thực vuông cấp n.

b. In ma trận đã nhập.

c. In đường chéo chính của ma trận (phần tử có chỉ số dòng bằng chỉ số cột).

d. Kiểm tra ma trận có đối xứng qua đường chéo chính không ?

e. Tính ma trận nghịch đảo.

4. Viết chương trình thực hiện các chức năng sau (mỗi chức năng viết một hàm riêng):

a. Nhập một ma trận số nguyên cấp m x n

b. In ma trận đã nhập

c. Xác định giá trị nhỏ nhất, lớn nhất, tổng các phần tử của ma trận

d. Xác định giá trị nhỏ nhất của từng dòng ma trận

e. Tính tổng từng cột của ma trận

f. Tính tích của 2 ma trận.

Biết tích của 2 ma trận $A_{m\times r}$ và $B_{r\times n}$ là ma trận $C_{m\times n}$ được định nghĩa như sau:

$$C_{m\times n} = A_{m\times r} \times B_{r\times n} \quad \text{với} \quad c_{i,j} = \sum_{k=1}^{r} a_{i,k} b_{k,j}$$

$c_{i,j}$ Phần tử ở dòng i cột j của ma trận tích $C_{m\times n}$

$a_{i,k}$ Phần tử ở dòng i cột k của ma trận $A_{m\times r}$

$b_{k,j}$ Phần tử ở dòng k cột j của ma trận $B_{r\times n}$

Ví dụ:

$$A_{2\times 3} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad B_{3\times 3} = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \rightarrow \quad C_{2\times 3} = \begin{bmatrix} 30 & 36 & 42 \\ 66 & 81 & 96 \end{bmatrix}$$

5. Viết chương trình thực hiện các chức năng sau (mỗi chức năng viết một hàm riêng):

a. Nhập một ma trận số thực vuông cấp nxm.

b. In ma trận đã nhập.

c. Hoán vị 2 dòng k và h của ma trận.

d. Thay cột h bằng tổng của cột h và cột k.

e. Xóa một dòng k bất kỳ của ma trận.

f. Xóa h dòng liên tiếp từ dòng thứ k

g. Chèn thêm một dòng vào ma trận.

6. Viết chương trình cho phép người dùng nhập vào một số nhị phân, bát phân hay thập lục phân và đổi sang hệ thập phân.

7. Viết các hàm chuẩn hóa chuỗi sau:

a. Xóa khoảng trắng đầu chuỗi.

b. Xóa khoảng trắng cuối chuỗi.

c. Xóa khoảng trắng dư giữa các từ trong chuỗi (chỉ để 1 khoảng trắng giữa các từ).

d. Đổi chuỗi thành chữ thường, riêng các chữ cái đầu mỗi từ thành chữ hoa.

8. Viết các hàm thao tác trên chuỗi họ tên:

a. Đảo chuỗi họ tên.

Ví dụ :

Tên ban đầu:" Phan Minh Anh"

Tên sau khi đảo :" Anh Phan Minh"

b. Tạo địa chỉ email theo chuỗi họ tên bằng 3 kiểu.

Ví dụ :

Tên :" Phan Minh Anh"

Địa chỉ email :phanminhanh@yahoo.com hay phan_minh_anh@yahoo.com hay

anhpm@yahoo.com

9. Viết các hàm thao tác trên chuỗi :

a. Đếm số từ trong chuỗi.

b. Tách từng từ của chuỗi.

c. Tìm số lần xuất hiện của từng chữ cái có trong chuỗi, hiển thị kết quả của các chữ cái theo thứ tự từ điển.

10. Viết chương trình cho phép người dùng nhập vào 2 số nguyên dương a và b (tối đa 100 chữ số) sau đó thực hiện các thao tác sau:

a. In số đảo ngược của số a

b. Hiển thị 2 số a, b theo thứ tự tăng dần.

c. Cộng và trừ 2 số a, b.

d. Tạo số c bằng cách ghép 2 số a, b theo thứ tự tăng dần

11. Chương trình Trò chơi Sổ số Vietlott

Viết chương trình cho người chơi đoán 6 số trong phạm vi từ 1 đến 45. Sau đó chương trình sẽ sinh 6 số ngẫu nhiên trong phạm vi 1 đến 45.

Nếu người chơi đoán đúng cả 6 số: Xuất thông báo "Bạn đã trúng 12 tỷ đồng"

Nếu người chơi đoán đúng 5 số: Xuất thông báo "Bạn đã trúng 10 triệu đồng"

Nếu người chơi đoán đúng 4 số: Xuất thông báo "Bạn đã trúng 300 ngàn đồng"

Nếu người chơi đoán đúng 3 số: Xuất thông báo "Bạn đã trúng 30 ngàn đồng"

Nếu người chơi đoán đúng ít hơn 3 số: Xuất thông báo số con số đoán đúng và "Chúc bạn may mắn lần sau".

Ví dụ kết quả chạy chương trình:

```
Moi ban chon 6 so tu 1 den 45
So thu 1: 2
So thu 2: 7
So thu 3: 15
So thu 4: 21
So thu 5: 30
So thu 6: 42
Ket qua so xo: 4       9    11   15   21   29  45
Ban doan trung 1 so . Chuc ban may man lan sau!
```

Phát triển trò chơi:

- Khi người chơi đoán số, kiểm tra nếu số trùng với số đã chọn thì yêu cầu chọn số khác

- Khi hệ thống sinh số ngẫu nhiên nếu số trùng thì sinh số khác

- Xuất dãy số đoán của người dùng và dãy số trúng giải theo thứ tự tăng dần (không cần dùng hàm sắp xếp mảng)

12. Chương trình Kiểm tra password của người dùng

Viết đoạn lệnh để yêu cầu người dùng nhập password của chương trình (ví dụ chương trình xổ số Vietlott). Nếu người dùng nhập đúng thì cho phép sử dụng chương trình, ngược lại xuất thông báo và cho phép nhập lại. Nếu sau 3 lần nhập vẫn bị sai thì thoát khỏi chương trình.

Ví dụ kết quả chạy chương trình:

```
Moi nhap password:  LTT2019
Password sai !
Moi nhap password:  LTT@2019@
Password đúng.
Chao mung ban den voi chuong trinh tro choi !
…
```

Phát triển chương trình:

- Che password người dùng nhập bằng dấu *

```
Moi nhap password:  ******

Password sai !

Moi nhap password:  *******

Password sai !

Moi nhap password:  ********

Password sai !

Ban khong duoc phep su dung chuong trinh !
```

13. Chương trình Trò chơi Chiếc nón kỳ diệu

Viết chương trình trò chơi chiếu nón kỳ điệu. Chương trình xuất lời gợi ý của ô chữ và cho người sử dụng đoán từng chữ cái trong ô chữ. Nếu người dùng đoán đúng thì chữ cái này sẽ xuất hiện trong ô chữ. Chương trình dừng khi tất cả các chữ cái đều được đoán hoặc người dùng hết lượt chơi (ví dụ 9 lần đoán)

Ví dụ kết quả chạy chương trình:

```
Day là ten mot Truong cao dang chat luong cao, gom 9 ky tu.
*********
Moi ban doan chu: C
Khong co chu C
*********
Moi ban doan chu: T
Co 2 chu T
**T*T****
Moi ban doan chu: N
Co 1 chu N
**T*T**N*
...
```

Phát triển trò chơi:

- Tạo nhiều ô chữ. Mỗi lần chơi chương trình sẽ chọn ô chữ ngẫu nhiên.

- Mỗi lần người dùng đóan đúng, người dùng sẽ được cộng một số điểm ngẫu nhiên. Nếu người dùng đoán sai sẽ bị trừ một số điểm ngẫu nhiên.

- Mở rộng cho 3 người chơi để tìm người thắng cuộc.

Day là ten mot Truong cao dang chat luong cao, gom 9 ky tu.

****

Moi nguoi choi thu nhat doan chu: T

Co 2 chu T. Ban duoc cong 7 diem.

T*T**

Moi nguoi choi thu 2 doan chu: A

Khong co chu A. Ban bi tru 5 điem.

T*TN*

...

CHƯƠNG 6: CON TRỎ

6.1. KHÁI QUÁT VỀ CON TRỎ

6.1.1. Khái niệm

Mỗi dữ liệu đều có một địa chỉ tương ứng dùng để chỉ vị trí của dữ liệu đó trên bộ nhớ. Con trỏ (pointer) là biến mà nội dung của biến đó là một địa chỉ bộ nhớ (memory address) dùng để chỉ đến một dữ liệu nào đó. Dùng con trỏ ta có thể truy cập biến thông qua địa chỉ của biến.

6.1.2. Lý do dùng con trỏ

Con trỏ là một khái niệm rất quan trọng trong ngôn ngữ C/C++. Nếu không có nó thì nhiều vấn đề trong lập trình sẽ rất khó có thể giải quyết:

- Để thay đổi giá trị một biến là tham số của một hàm, ta cần dùng một tham biến cho hàm, thực chất tham số này là một con trỏ.

- Việc dùng con trỏ cho mảng (array), chuỗi (string) và cấu trúc (structure) sẽ làm cho việc xử lý chúng trỏ nên dễ dàng và linh hoạt hơn.

- Đối với các cấu trúc dữ liệu có độ phức tạp cao như danh sách (linked list) và cây (tree), nếu không có con trỏ sẽ không giải quyết được.

- Việc dùng con trỏ trong chương trình sẽ làm cho việc sử dụng bộ nhớ tối ưu hơn nhiều so với việc không dùng con trỏ. Nếu dùng biến thông thường, thì chúng sẽ được cấp phát các khối nhớ một cách tĩnh (static) hay cố định (fixed) trong vùng nhớ qui ước gọi là stack. Còn khi dùng biến con trỏ thì các biến này sẽ được cấp phát các khối nhớ một cách động (dynamic) trong vùng nhớ qui ước gọi là heap.

Ví dụ để lưu một danh sách các nhân viên của một công ty với cấu trúc mảng chẳng hạn, ta cần phải xác định số phần tử tối đa của mảng. Mà ta biết là số lượng nhân viên là không cố định, do đó cần dự trù một số tối đa các phần tử phải đủ lớn. Điều này gây ra sự lãng phí bộ nhớ lớn nếu số nhân viên ít hơn nhiều so với số

phần tử mảng. Vấn đề này sẽ được giải quyết một cách dễ dàng bằng cách dùng con trỏ. Cần bao nhiêu phần tử cho danh sách, ta chỉ cần xin cấp phát cho đủ trong quá trình chạy chương trình.

Tuy nhiên, con trỏ có một điểm bất lợi và mang tiếng xấu, đó là việc hiểu và sử dụng nó tương đối khó với một số người. Trong một số trường hợp việc dùng con trỏ sẽ làm cho chương trình trở nên không còn dễ dàng để hiểu được giải thuật và làm cho chương trình giảm đi sự trong sáng. Để vượt qua sự bất lợi này, chỉ có cách là phải hiểu thấu đáo về nó!.

Trong những trường hợp sau, ta cần sử dụng con trỏ:

- Khi sử dụng các cấu trúc dữ liệu mà trong đó ta không biết trước được lượng dữ liệu cần dùng đến
- Độ lớn của dữ liệu sẽ thay đổi trong qúa trình thực hiện chương trình
- Chủ động xin cấp phát hay hủy vùng nhớ cho các biến
- Tham số truyền cho hàm là tham số biến
- Khi thao tác trên mảng và chuỗi bằng các tính tóan địa chỉ nhằm truy cập đến 1 phần tử của mảng/chuỗi

6.1.3. Khai báo biến con trỏ

```
<Kiểu dữ liệu>* <Tên biến con trỏ>;
<Kiểu dữ liệu> *<Tên biến con trỏ>;
```

Ví dụ 6.1:

```
int *x,*y;//x,y là con trỏ lưu địa chỉ của một số nguyên
float *z; //z là con trỏ lưu địa chỉ của một số thực
char *s; //s là con trỏ lưu địa chỉ của một số ký tự
```

6.1.4. Sử dụng biến con trỏ trong các biểu thức

Để sử dụng được biến con trỏ ta cần phân biệt 2 ký hiệu cũng chính là 2 tóan tử sau:

*****	Tóan tử gián tiếp *(indirection operator)* dùng để tham khảo đến giá trị của ô nhớ mà biến con trỏ trỏ vào
&	Tóan tử địa chỉ *(address operator)* dùng để tham khảo đến địa chỉ của biến

Ví dụ 6.2: Để gán giá trị **1234** cho vùng nhớ mà biến con trỏ **p** trỏ tới, ta viết: ***p = 1234;**

Để gán địa chỉ biến **i** cho biến con trỏ **p**, ta viết: **p = &i;**

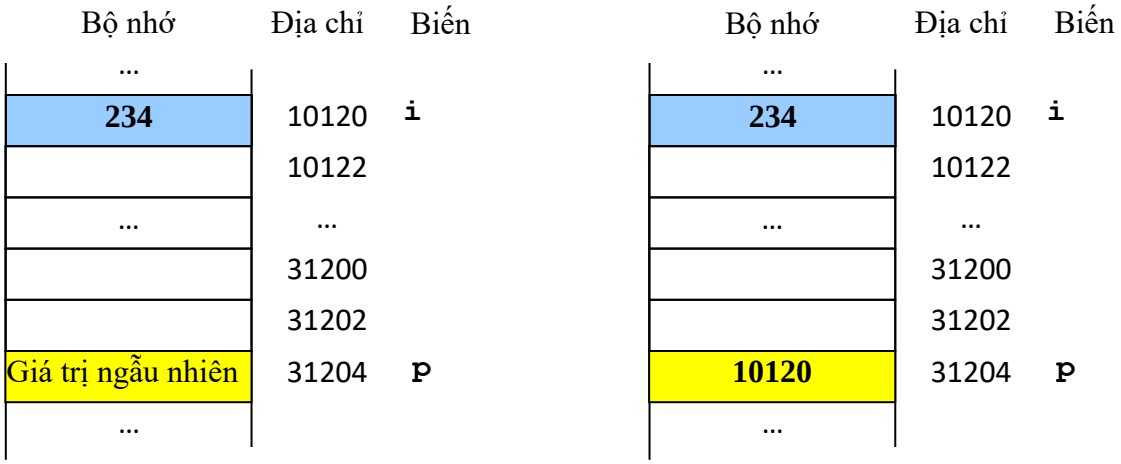
Phép gán : **p = i;** sẽ gây ra lỗi biên dịch: **cannot convert from 'int' to 'int *'**, vì nội dung của **p** là một địa chỉ, còn nội dung của **i** là một giá trị nguyên: 2 kiểu dữ liệu khác nhau.

Ví dụ 6.3: Xét đoạn chương trình sau

Đoạn chương trình	Giải thích
… `int i=234;`	Khai báo và cấp phát một vùng nhớ cho biến **i** (2 byte hoặc 4 byte tùy thuộc vào chương trình dịch và hệ điều hành) bắt đầu tại địa chỉ nào đó (giả sử là **10120)** và gán giá trị tại địa chỉ này là **234**
`int *p;`	Khai báo **p** là biến lưu một địa chỉ của một số nguyên nào đó, hiện địa chỉ này chưa xác định
`P = &i;`	**p** sẽ lưu địa chỉ của biến **i** là **10120**. Ta nói "**p** trỏ vào **i**"
`printf("%d", *p)`	In giá trị tại địa chỉ được lưu trong biến p tức là in giá trị của biến **i**, hay nói cách khác là giá trị mà biến **p** trỏ vào.

Bảng 6. 1: So sánh toán tử trực tiếp và toán tử địa chỉ dùng trong ví dụ 6.1

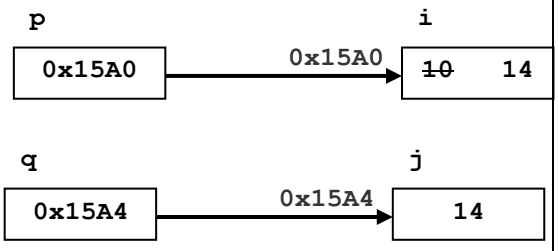
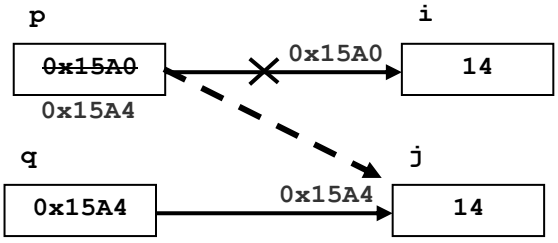
`int i=234;`	`int *p=&I;`
`i=234`	`*p=234`
`&i=10120`	`p=10120`



Ví dụ 6.4: Xét đoạn chương trình sau

```
int i=10, j=14;int *p=&i;

int *q=&j;
```

Bảng 6. 2: Minh họa sự khác nhau giữa gán giá trị và gán địa chỉ

Gán giá trị	Gán địa chỉ
`*p=*q;`	`p=q;`
	

6.1.5. Cấp phát và giải phóng vùng nhớ

Trước khi sử dụng biến con trỏ, nếu không muốn gán địa chỉ đã có sẵn cho biến con trỏ (như trong các ví dụ 3.1, 3.2 với câu lệnh **`p=&i;`**) ta phải xin hệ điều hành cấp phát một lượng bộ nhớ tương ứng với kiểu dữ liệu đã khai báo cho biến con trỏ.

Để biết kích thước tương ứng đối với từng kiểu dữ liệu ta dùng hàm **`sizeof()`**

`sizeof(<Kiểu dữ liệu>);`

(Trả về kiểu ***`size_t`*** là một số nguyên không dấu)

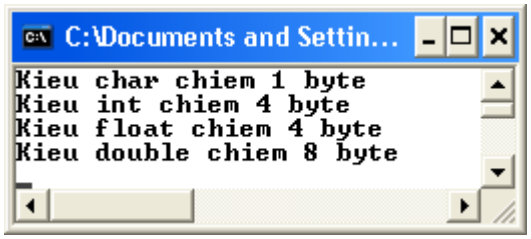
`sizeof(<Biểu thức>);`

(Cho biết số **`byte`** mà ***`<kiểu dữ liệu>`*** hay ***`<biểu thức>`*** chiếm trong bộ nhớ)

Ví dụ 6.5: Chương trình sau minh họa cách dùng hàm ***`sizeof()`***

```
1   #include <stdio.h>
2   #include <conio.h>
3   int main()
4   {
5       printf("Kieu char chiem %d byte\n", sizeof(char));
6       printf("Kieu int chiem %d byte\n", sizeof(int));
7       printf("Kieu float chiem %d byte\n", sizeof(float));
8       printf("Kieu double chiem %d byte\n", sizeof(double));
9       getch();
10      return 0;
11  }
```

Kết quả chạy chương trình:



6.1.5.1. Cấp phát vùng nhớ dung toán tử new

Cú pháp:

```
<Biến con trỏ> = new <Kiểu dữ liệu>;
```

6.1.5.2. Giải phóng vùng nhớ dung toán tử delete

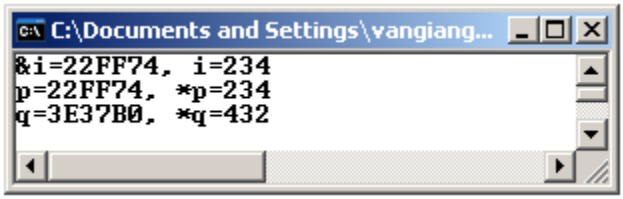
Cú pháp:

```
delete <biến con trỏ>;
```

Ví dụ 6.6: Xét chương trình in ra địa chỉ (theo dạng số hexadecimal) và nội dung của các biến như sau:

```
    …
1   int main(){
2       int i=234;
3       int *p, *q;
4       p = &i;
5       q = new int;
6       *q = 432;
7       printf("&i=%X, i=%d\n", &i, i);
8       printf("p=%X, *p=%d\n", p, *p);
9       printf("q=%X, *q=%d\n", q, *q);
10      delete q;
11      getch();
12      return 0;
13  }
```

Kết quả chạy chương trình (các giá trị địa chỉ mỗi lần chạy chương trình sẽ có thể là không giống nhau):



Trong chương trình trên:

-Vì **p** chỉ là con trỏ trỏ vào vùng nhớ của **i** nên không cần xin cấp phát vùng nhớ, do vậy cũng không phải giải phóng vùng nhớ.

- Trong khi đó **q** là con trỏ trỏ vào vùng nhớ của riêng nó nên cần xin cấp phát vùng nhớ bằng **new**, sau khi dùng xong cần giải phóng vùng nhớ đã chiếm dụng bằng **delete**.

6.2. ỨNG DỤNG CỦA CON TRỎ

6.2.1. Hàm có tham số con trỏ

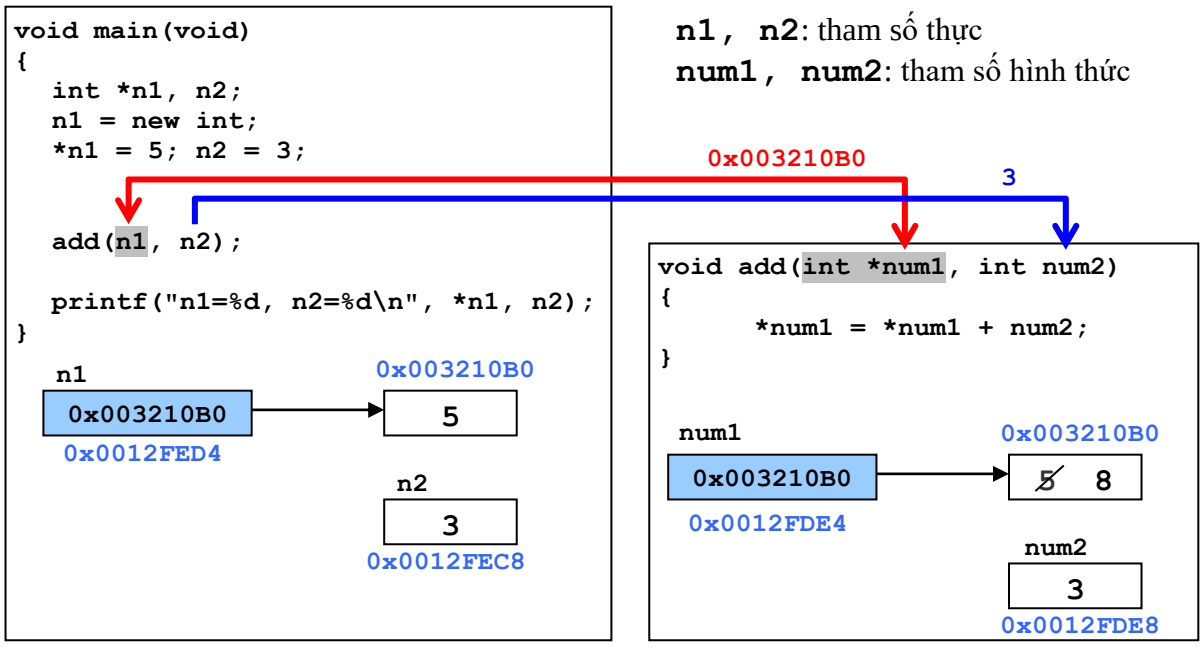
Như đã đề cập đến ở chương 1, có 2 loại tham số hình thức là tham trị và tham biến. Trong trường hợp dùng tham biến thì tham số truyền cho hàm không phải là giá trị mà là địa chỉ của biến nên là một con trỏ . Khi đó mọi sự thay đổi giá trị của tham số hình thức bên trong thân hàm sẽ làm thay đổi giá trị của tham số thực bên ngòai hàm.

Ví dụ 6.7: Trong chương trình sau, hàm **`add()`** có 2 tham số, tham số thứ nhất **`num1`** là tham biến (truyền địa chỉ), và tham số thứ hai **`num2`** là tham trị (truyền giá trị).

```
    …
1   void add(int &num1, int num2){
2        num1 = num1 + num2;
3   }
4   int main(void){
5        int n1, n2;
6        n1 = 5;
7        n2 = 3;
8        add(n1, n2);
9        printf("n1=%d, n2=%d\n", n1, n2);
10       getch() ;
11       return 0 ;}
```

Kết quả chạy chương trình:

Nếu dùng con trỏ thì ví dụ trên sẽ được trình bày như sau :



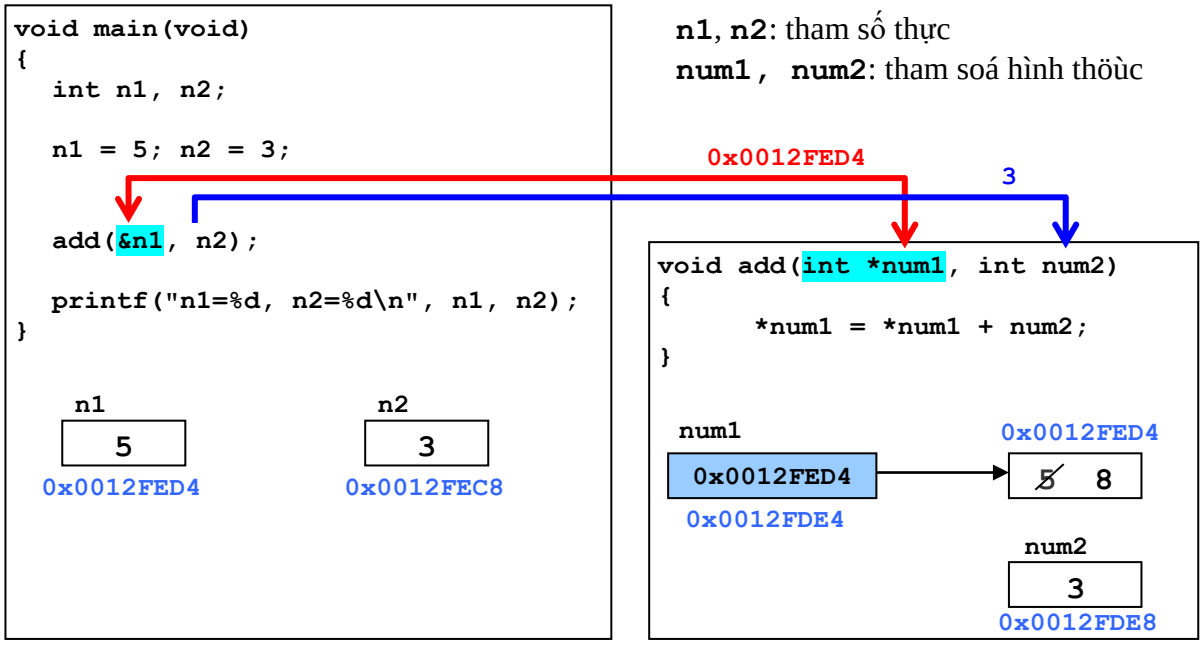
Hình 6.2: Minh họa hình ảnh việc truyền tham số dùng con trỏ

Bảng 6. 3: Giải thích ví dụ 7.1

	Chương trình	Giải thích
1	`int main()`	
2	`{`	**n1** được khai báo là một con trỏ trỏ đến một giá trị kiểu **int**, **n2** là biến thông thường kiểu **int**
3	`int *n1, n2;`	
4	`n1 = new int;`	Sau khi xin cấp phát bộ nhớ bằng tóan tử **new**, **n1** trỏ vào địa chỉ ô nhớ đã được cấp phát là **0x003210B0** (nghĩa là **n1** chứa địa chỉ này).
5	`*n1 = 5;`	Ô nhớ có địa chỉ **0x003210B0** sẽ có giá trị là **5**
6	`n2 = 3;`	

7	add(n1, n2);	Tham số hình thức **num1** sẽ nhận được địa chỉ **0x00B3210B0** Tham số hình thức **num2** sẽ nhận được giá trị **3**
8	printf("n1=%d, n2=%d\n", *n1, n2);	
9	return 0;	
10	}	
11	void add(int *num1, int num2)	Giá trị tại địa chỉ mà con trỏ **num1** trỏ vào (**0x003210B0**) bị thay đổi thành **8** (**5+3**). Do **n1** cùng trỏ vào ô nhớ này nên giá trị của **n1** cũng bị thay đổi theo
12	{	
13	*num1 = *num1 + num2;	
14	}	

Ngòai ra, chương trình trên ta cũng có thể không dùng con trỏ **n1** mà dùng biến thông thường. Để gọi hàm **add()** với tham biến ta truyền địa chỉ của biến **n1** cho **num1**. Lúc đó việc thao tác dữ liệu của **num1** thực chất là **n1**, vì **num1** là con trỏ trỏ vào **n1.**



Hình 6.3: Minh họa hình ảnh việc truyền tham số là địa chỉ biến kiểu nguyên

6.2.2. Con trỏ và mảng

Mảng có thể được cấp phát với số phần tử của mảng là cố định như trong chương 2 đã đề cập đến.

Chẳng hạn để lưu một dãy tối đa 100 số nguyên ta khai báo:

```
int a[100];
```

Tuy nhiên trong nhiều trường hợp ta không biết trước số phần tử tối đa mà người dùng sẽ sử dụng là bao nhiêu nên nếu khai báo số phần tử tối đa quá lớn sẽ gây lãng phí bộ nhớ, ngược lại thì số phần tử có khi sẽ vượt quá con số được khai báo.

Trong trường hợp này nếu dùng con trỏ thì ta có thể xin cấp phát lượng bộ nhớ vừa đủ theo yêu cầu của người dùng và chủ động giải phóng vùng nhớ khi không dùng đến mảng nữa.

Để thực hiện điều này ta sẽ cho phép người dùng nhập vào số phần tử n trước khi xin cấp phát n phần tử nhớ.

Ngoài ra khi tham khảo đến các phần tử của mảng, ta cũng có thể sử dụng ký hiệu kiểu con trỏ như sau:

Bảng 6. 4: So sánh ký hiệu thông thường và ký hiệu dùng con trỏ

Thao tác	Ký hiệu thông thường	Sử dụng con trỏ
Tham khảo địa chỉ phần tử	`&a[i]`	`a+i`
Tham khảo nội dung phần tử	`a[i]`	`*(a+i)`

Ví dụ 6.8: Hàm `Input()` trong ví dụ 5.7 được viết lại theo cách dùng con trỏ như sau:

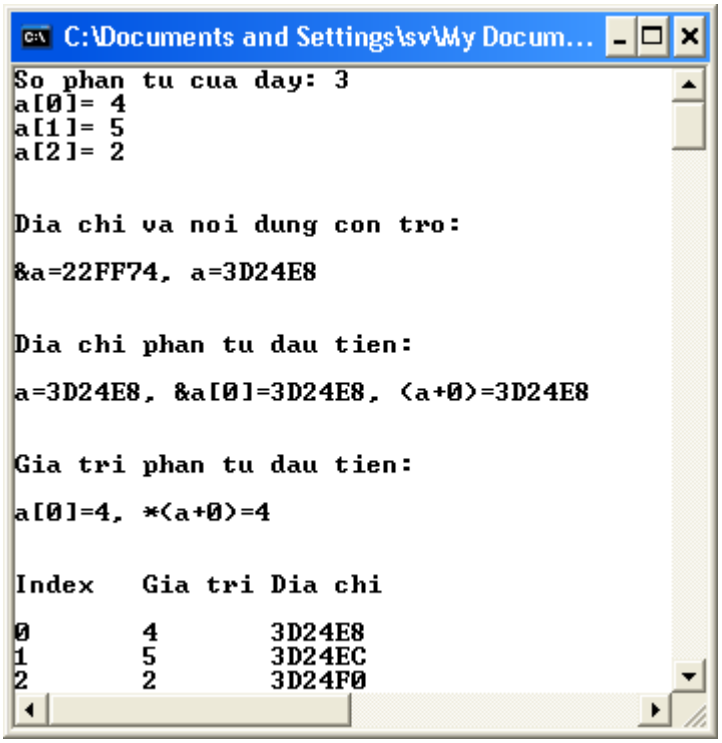
```
1  void Input(float *&a, int &n)  {
2      printf(“Nhap vao so phan tu: ”);
3      scanf(“%d”,&n);
4      a = new float[n];
5      for(int i=0; i<n; i++){
6          printf(“a[%d] =”, i); scanf(“%f”, (a+i));
7      }
8  }
```

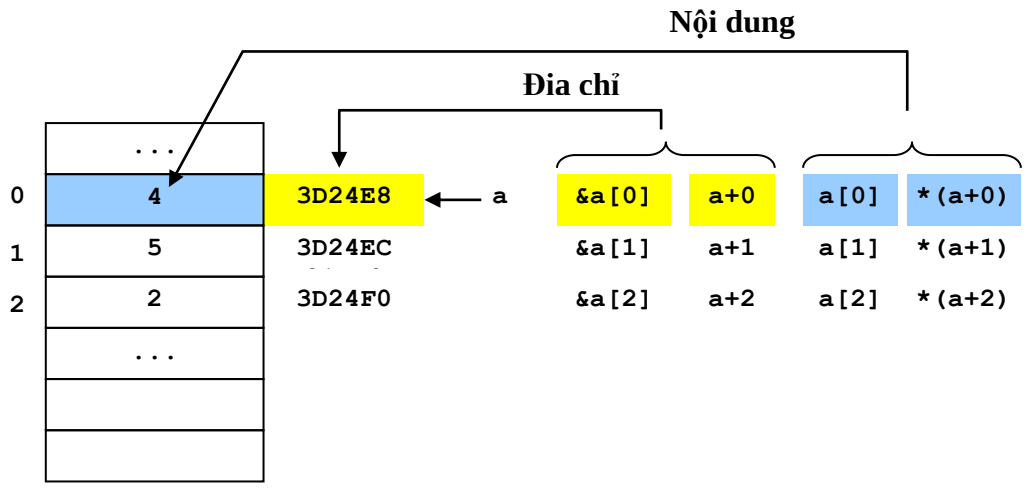
Ví dụ 6.9: Chương trình sau minh họa sự khác nhau giữa địa chỉ của biến con trỏ, địa chỉ mà biến con trỏ trỏ tới với giá trị dữ liệu tại địa chỉ mà biến con trỏ trỏ tới.

```
   …
1  int main(){
2      int *a, i, n;
3      do{
4          printf("So phan tu cua day: ");scanf("%d", &n);
5      }while (n<=0);
6      a = new int[n];
7      for(i=0; i<n; i++){
8          printf("a[%d]= ", i);
9          scanf("%d",a+i);
10     }
```

```
11      printf("\n\nDia chi va noi dung con tro:\n\n");
12      printf("&a=%X, a=%X\n", &a, a);
13      printf("\n\nDia chi phan tu dau tien:\n\n");
14      printf("a=%X, &a[0]=%X, (a+0)=%X\n", a, &a[0], (a+0));
15      printf("\n\nGia tri phan tu dau tien:\n\n");
16      printf("a[0]=%d, *(a+0)=%d\n", a[0], *(a+0));
17      printf("\n\nIndex\tGia tri\tDia chi\n\n");
18      for(i=0; i<n; i++)
19          printf("%d\t%d\t%X\n", i, a[i], a+i);
20      delete []a;
21      getch(); return 0;
22  }
```

Kết quả chạy chương trình:





Hình 6.4: Minh họa hình ảnh hình ảnh bộ nhớ của mảng a trong ví dụ 6.8

6.2.3. Con trỏ va chuỗi

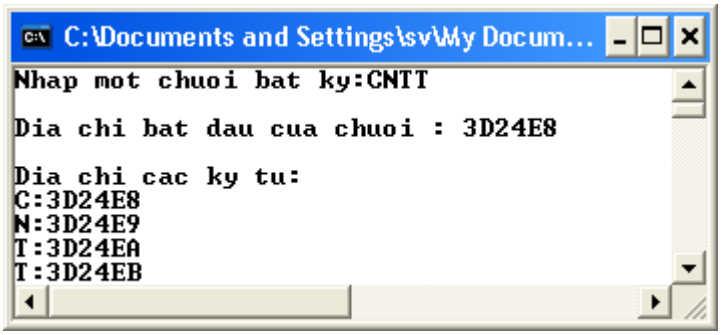
Tương tự như đối với mảng số, con trỏ cũng có thể dùng để trỏ đến một mảng các ký tự hay chuỗi. Việc tính toán các địa chỉ cho chuỗi cũng tương tự như mảng thông thường. Ngoại trừ việc nhập/xuất chuỗi, cũng như một số phép tóan trên chuỗi ta dùng các hàm chuyên cho việc xử lý chuỗi như đã trình bày trong chương 5. Các ví dụ sau đây sẽ minh họa việc dùng con trỏ trỏ vào chuỗi.

Ví dụ 6.10

Trong ví dụ này ta dùng con trỏ để trỏ vào một chuỗi ký tự thay cho việc dùng mảng ký tự như đã trình bày ở chương 2.

```
    …
1   int main()
2   {
3       char *s = new char[255];
4       printf("Nhap mot chuoi bat ky:\n");
5       gets(s);
6       int len=strlen(s);
7       printf("\nDia chi bat dau cua chuoi : %X\n", s);
8       printf("\nDia chi cac ky tu:\n");
9       for(int i=0; i<len; i++)
10          printf("%c:%X\n", *(s+i), s+i);
11      getch();
12      return 0;
13  }
```

Kết quả chạy chương trình:

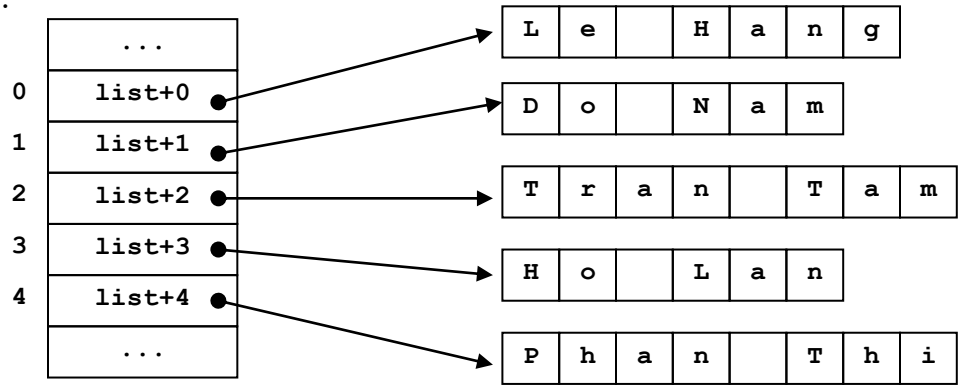


Ví dụ 6.11

Trong trường hợp để lưu một danh sách, chẳng hạn danh sách tên sinh viên, ta có thể dùng một mảng 2 chiều: mỗi dòng sẽ xác định một chuỗi ký tự, là tên sinh viên tương ứng.

Trong trường hợp này ta cũng có thể dùng con trỏ 2 cấp, tức là con trỏ trỏ vào

con trỏ.



Tương tự để khai báo một con trỏ 2 cấp hay còn gọi là con trỏ kép dùng để trỏ đến chuỗi ký tự, ta viết ****** trước tên biến.

```
char **list;
```

Ta có thể xin cấp phát bộ nhớ cho con trỏ **list** như sau với n là số dòng:

```
list = new char*[n];
```

Việc tiếp theo là xin cấp phát cho mỗi chuỗi tên. Vì mỗi địa chỉ của list gồm **(list+0), (list+1), (list+2),**…trỏ vào một chuỗi, do đó để xin cấp phát vùng nhớ cho **n** chuỗi ta dùng cấu trúc lặp với **n** lần lặp:

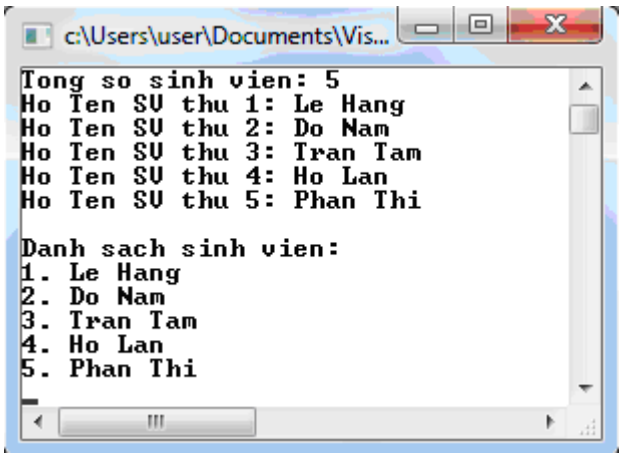
```
for(int i=0; i<n; i++)
     list[i] = new char[m];
```

với m là dộ dài của chuỗi muốn xin cấp phát.

Sau đây là toàn bộ chương trình:

```
    …
1   int main(void){
2       char **list; int n;
3       printf("Tong so sinh vien: "); scanf("%d", &n);
4       if(n>0)  list=new *char[n];
5       else return 0;
6       for(int i=0; i<n; i++){
7           list[i]=new char[50];
8           printf("Ho Ten SV #%d: ", i+1);
9           fflush(stdin);
10          gets(list[i]);
11      }
12      printf("\nDanh sach sinh vien:\n");
13      for(int i=0; i<n; i++)
14          printf("%d. %s\n",i+1, *(list+i));
15      getch(); return 0;}
```

Kết quả chạy chương trình:



BÀI TẬP CHƯƠNG 6

1. Viết chương trình dùng con trỏ thực hiện các chức năng sau (mỗi chức năng viết một hàm riêng):

a. Dùng phương pháp cấp phát bộ nhớ động để nhập một dãy gồm n số nguyên. In ra màn hình:

b. Tòan bộ dãy số theo thứ tự đã nhập

c. Tòan bộ dãy số theo thứ tự ngược với thứ tự đã nhập

d. Dãy con gồm các số tại các vị trí chỉ mục chẵn

e. Dãy con gồm các số chẵn

f. Đếm xem có bao nhiêu số chẵn

g. Tính tổng các số chẵn của dãy số

h. Tìm số lớn nhất trong dãy số

i. Tìm số âm lớn nhất trong dãy số

j. Xóa tất cả các số chẵn

k. Chèn một số x vào vị trí k trong dãy số

2. Viết chương trình dùng con trỏ nhập vào một ma trận vuông cấp n>2. Thực hiện:

- In ma trận ra màn hình
- Đếm số các phần tử có giá trị lẻ và tính tổng của chúng
- Tìm phần tử lớn nhất của từng dòng và nhỏ nhất của từng cột

3. Viết chương trình dùng con trỏ nhập vào một danh sách nhiều họ tên sinh viên. Nhập vào một họ tên, tìm và cho biết có sinh viên nào trong danh sách trùng với họ tên đã nhập hay không. Yêu cầu dùng các hàm riêng cho từng chức năng.

4. Viết chương trình dùng con trỏ nhập vào một danh sách có nhiều tên sinh viên. Thay vì in các ký tự trong các chuỗi, in ra màn hình mã ASCII tương ứng với từng ký tự.

Ví dụ:

Nhập: "**Son**", "**Nam**", "**Tuan**"

Xuất: "**83 111 110**", "**78 97 109**", "**84 117 97 110**"

5. Viết chương trình dùng con trỏ cho phép người dùng nhập vào danh sách các sinh viên gồm họ tên, năm sinh, điểm. sau đó thực hiện các thao tác:

a. In ra toàn bộ danh sách đã nhập theo dạng bảng như sau:

STT	Ho va ten	Nam sinh	Diem
1	Tran Van Tuan	1992	7
2	Nguyễn Thị Lan	1993	8
…			

d. Đếm xem có bao nhiêu sinh viên có điểm nhỏ hơn 5

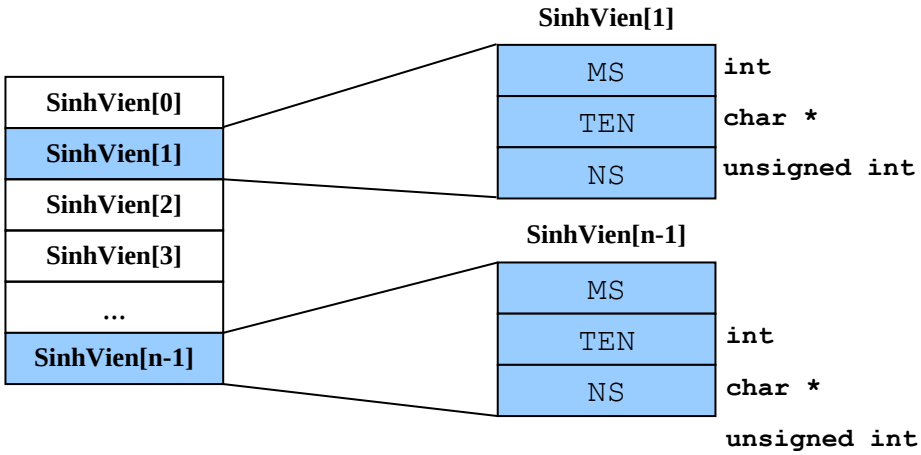
CHƯƠNG 7: KIỂU DỮ LIỆU DO NGƯỜI DÙNG ĐỊNH NGHĨA

7.1. KIỂU CẤU TRÚC (STRUCT)

Struct (structure) là một dạng thức mà C/C++ cung cấp cho lập trình viên khả năng xây dựng một kiểu dữ liệu mới trên cơ sở các kiểu đã được định nghĩa và kết hợp chúng lại với nhau.

Chẳng hạn để lưu thông tin của một sinh viên gồm mã số, họ tên, năm sinh, thường thì ta cần 3 biến có kiểu khác nhau. Nếu muốn lưu một danh sách sinh viên với các thông tin như trên, ta có thể xây dựng 3 mảng 1 chiều. Tuy nhiên, lúc đó sẽ không có sự đồng bộ khi tham khảo đến 1 sinh viên trong danh sách, việc xử lý cũng sẽ trở nên phức tạp hơn.

Với cách tiếp cận khác, dùng **`struct`**, sẽ làm cho bài tóa trên trở nên đơn giản. Lúc đó, mỗi một sinh viên là một cấu trúc với 3 thông tin ràng buộc nhau: **`MS`** – Mã số, **`TEN`** – Họ tên, **`NS`** – Năm sinh. Danh sách sinh viên sẽ là một mảng các cấu trúc đã nêu, được mô tả như hình vẽ. Trong đó, các thành phần (components) hay thành viên (members) của một **`struct`** có kiểu dữ liệu cơ sở, chẳng hạn **`ID`** có kiểu **`int`**, **`TEN`** có kiểu chuỗi **`char*`**, và **`NS`** có kiểu **`unsigned int`**. Trong những cấu trúc phức tạp khác thì kiểu dữ liệu của thành phần cũng có thể lại là một **`struct`** hay kiểu dữ liệu khác do người dùng định nghĩa.



Hình 7.1: Minh họa một mảng các struct

7.1.1. Khai báo cấu trúc dữ liệu

7.1.1.1. Khai báo struct

Cũng như các biến thông thường, để có thể sử dụng một kiểu dữ liệu mới dùng struct, ta cần khai báo và mô tả một struct theo cú pháp sau:

```
struct [<Tên kiểu>]
{
   <Mô tả các thành phần>;
} [<Danh sách biến>];
```

<Tên kiểu>	Tên của kiểu dữ liệu mới, tên này không được trùng với kiểu dữ liệu đã có
<Danh sách biến>	Danh sách các tên biến có kiểu struct đã mô tả, phân cách bằng dấu ",".
<Mô tả các thành phần>	Các mô tả cho từng thành phần trong struct

Sau khi một **struct** đã được mô tả như cú pháp ở trên, ta có thể khai báo các biến có kiểu dữ liệu mới này theo cách khai báo biến theo cú pháp

```
struct <Tên kiểu> <Danh sách biến>;
```

Ví dụ 7.1:

Mô tả struct có tên SV gồm mã số, họ tên, năm sinh và khai báo 2 biến x và y có kiểu dữ liệu mới này:

```
1  struct SV
2  {
3      int ms;
4      char ten[50];
5      int ns;
6  } x, y;
7  struct SV t;
8  struct SV *p;
9  struct SV SinhVien[100];
```

Ví dụ 7.2: Mô tả struct có tên date để chứa ngày tháng năm

```
1  struct date
2  {
3      int day;
4      int month;
5      int year;
6  };
7  struct date today;
8  struct date birthday[100];
```

7.1.1.2. Khai báo và khởi tạo giá trị cho biến struct

struct <Tên kiểu> <Tên biến> = {<Giá trị của các thành phần>};

Ví dụ 7.3:

```
1   struct SV x={1, "Phan Khang", 1992};
2   struct SV SinhVien[]={{1,"Phan Khang",1992},
                                          {2,"Phan Thi An",1991}};
3   struct date today={1, 4, 2011};
```

7.1.1.3. Định nghĩa kiểu dữ liệu mới bằng typedef

Ngoài cách khai báo cấu trúc dữ liệu như trên, ta cũng có thể định nghĩa **struct** dùng từ khóa **typedef**.

Khi dùng **typedef** ngoài việc định nghĩa một kiểu dữ liệu mới, việc khai báo các biến sẽ không phải viết từ khóa **struct** nữa nên sẽ tự nhiên hơn, như các kiểu dữ liệu cơ sở khác.

```
typedef struct <Tên kiểu>
{
   <Mô tả các thành phần>;
};
```

Ví dụ 7.4:

```
1   typedef struct SV
2   {
3       int ms;
4       char ten[50];
5       int ns;
6   };
    //Khai báo biến kiểu SV:
7   SV t;
8   SV *p;
9   SV a[100];
```

7.1.2. Tham khảo các thành phần của struct

Khi các biến **struct** đã được khai báo, ta có thể truy cập đến từng thành phần/thành viên (component/member) của **struct** bằng toán tử thành phần (member operator) với dấu chấm "**.**" đối với biến thường hoặc dùng dấu mũi tên "→" khi dùng biến con trỏ.

```
<Tên biến>.<Thành phần>;        //Biến thường
<Tên biến> -> <Thành phần>;    // Biến con trỏ
```

Ví dụ 7.5:

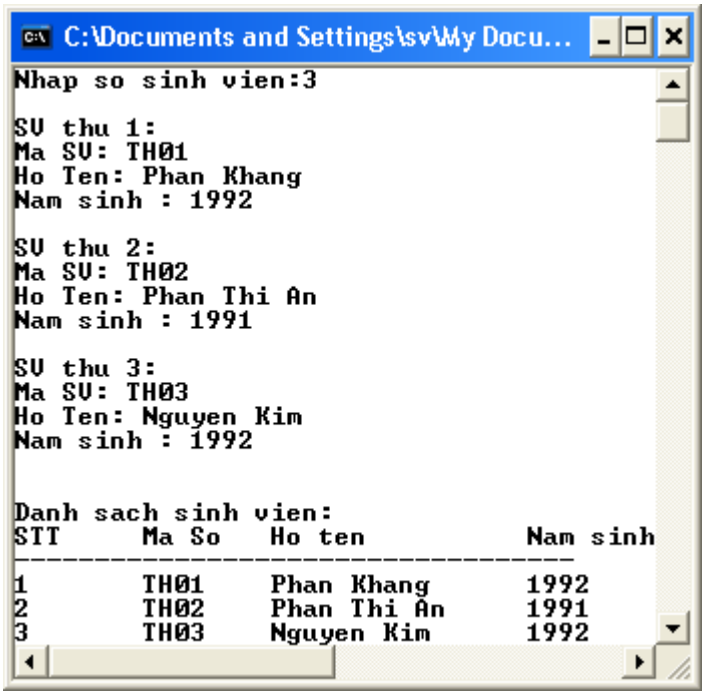
```
    ...
1   SV x,*p,a[100];
2   x.ms=1;
3   strcpy(x.ten , "Phan Khang");
4   std1.ns=1992;
5   p=(SV*) malloc(sizeof(SV));
6   p->ms=2;
7   strcpy(p->ten , "Phan Thi An");
8   p->ns=1991;
9   a[0].ms=3;
10  strcpy(a[0].ten , "Nguyen Kim");
11  a[0].ns=1992;
    ...
```

Ví dụ 7.6: Chương trình sau cho phép nhập vào một danh sách sinh viên gồm các thông tin: mã sinh viên, họ tên , năm sinh. Sau đó in danh sách sinh viên theo dạng bảng.

```
    …
1   #define N 100
2   typedef struct SV
3   {
4       char ms[10];
5       char ten[50];
6       int ns;
7   };
8   void Input(SV a[],int &n)
9   {
10      printf("Nhap so sinh vien:"); scanf("%d",&n);
11      for(int i=0; i<n; i++)
12      {
13          printf("\nSV thu %d:\n", i+1);
14          printf("Ma SV: ");
15          fflush(stdin); gets(a[i].ms);
16          printf("Ho Ten: ");
17          fflush(stdin); gets(a[i].ten);
18          printf("Nam sinh : "); scanf("%d", &a[i].ns);
19      }
20  }
21  void Output(SV a[],int n)
22  {
23      printf("\n\nDanh sach sinh vien:\n");
24      printf("STT\tMa So\tHo ten\t\tNam sinh\n");
25      printf("---------------------------------\n");
```

```
26      for(int i=0; i<n; i++)
27          printf("%d\t%s\t%s\t%d\n", i+1, a[i].ms, a[i].ten,
28          a[i].ns);
29  }
30  int main()
31  {
32      SV a[N];int n;
33      Input(a,n);
34      Output(a,n);
35      getch();
36      return 0;
37  }
```

Kết quả chạy chương trình:



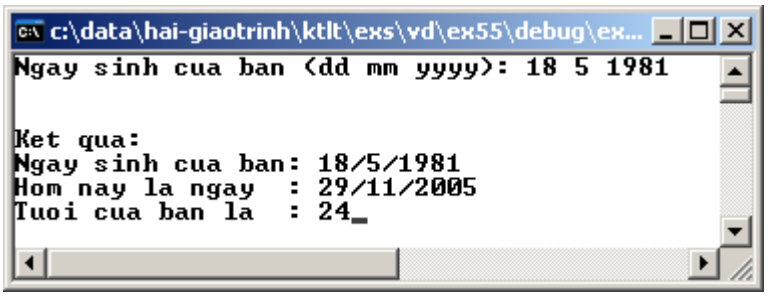
Ví dụ 7.7: Viết chương trình dùng struct mô tả một cấu trúc chứa ngày, tháng, năm để nhập vào ngày sinh của một người, tính và in ra tuổi của người đó so với

ngày hiện thời.

```
    …
1   #include <time.h>
2   typedef struct Date{
3       int day;
4       int month;
5       int year;
6   };
7   int main(){
8       struct tm *now;
9       time_t timer;
10      Date dob, today; // dob: date of birth
11      int age;
12      timer = time(NULL);
13      now = localtime(&timer);
14      today.day = now->tm_mday;
15      today.month = now->tm_mon+1;
16      today.year = now->tm_year+1900;
17      printf("Ngay sinh cua ban (dd mm yyyy): ");
18      scanf("%d%d%d", &dob.day, &dob.month, &dob.year);
19      if(today.month>dob.month ||
20          (today.month==dob.month && today.day>=dob.day))
21          age = today.year - dob.year;
22      else age = today.year - dob.year - 1;
23      printf("\n\nKet qua:\n");
```

```
24      printf("Ngay sinh cua ban: %d/%d/%d\n", dob.day, dob.month,
                                                          dob.year);
25      printf("Hom nay la ngay  : %d/%d/%d\n", today.day,
                                          today.month, today.year);
26      printf("Tuoi cua ban la  : %d", age);
27      getch();
28      return 0;}
```

Kết quả chạy chương trình:



Trong chương trình ta sử dụng cấu trúc **`struct tm`** được xây dựng sẵn trong file header **`time.h`** như sau:

```
struct tm{
    tm_sec  Seconds after minute (0 - 59).
    tm_min  Minutes after hour (0 - 59).
    tm_hour Hours after midnight (0 - 23).
    tm_mday Day of month (1 - 31).
    tm_mon  Month (0 - 11; January = 0).
    tm_year     Year (current year minus 1900).
    tm_wday     Day of week (0 - 6; Sunday = 0).
    tm_yday     Day of year (0 - 365; January 1 = 0).
};
```

Để lấy ngày hiện thời, hàm **`localtime(&timer)`** được gọi, trong đó tham số **`timer`** là con trỏ trỏ đến kiểu **`time_t`** đã được định nghĩa sẵn. Hàm **`localtime()`** trả về một con trỏ trỏ vào một **`struct tm`**.

BÀI TẬP CHƯƠNG 7

1. Cho danh sách lưu thông tin của các mặt hàng, thông tin gồm:

- Mã mặt hàng (chuỗi 10 ký tự)
- Tên mặt hàng (chuỗi 50 ký tự)
- Số lượng mặt hàng (số nguyên)
- Đơn giá mặt hàng (số thực, đơn vị 1000 VNĐ)

Viết chương trình thực hiện các chức năng sau, mỗi chức năng viết một hàm riêng

a. Nhập danh sách các mặt hàng, lưu ý khi nhập cần kiểm tra trùng mã mặt hàng.

b. Xuất danh sách các mặt hàng theo dạng bảng, và tính thành tiền của từng mặt hàng.

STT	Mã hàng	Tên hàng	Số lượng	Đơn giá	Thành tiền
1	TV	Tivi	100	2000	200000
2.	TL	Tu lanh	200	3000	600000
…					

c. Tìm một mặt hàng theo mã mặt hàng.

d. Xuất danh sách các mặt hàng có số lượng lớn hơn hay bằng 100.

e. Đếm xem có bao nhiêu mặt hàng có số lượng lớn hơn hay bằng 100.

f. Tính giá trung bình của các mặt có giá >1.000.000 VNĐ

g. Xuất các mặt hàng có giá lớn nhất.

h. Xuất các mặt hàng có số lượng nhỏ nhất.

l. Cập nhật giảm giá các mặt hàng 10%.

m. Xóa các mặt hàng có số lượng = 0.

2. Cho danh sách lưu thông tin của các sinh viên, thông tin gồm:

- Mã sinh viên (chuỗi 10 ký tự)
- Tên sinh viên (chuỗi 50 ký tự)
- Năm sinh (số nguyên)
- Điểm (số thực)

Viết chương trình thực hiện các chức năng sau, mỗi chức năng viết một hàm riêng

a. Nhập danh sách các sinh viên và lưu vào file.

b. Đọc dữ liệu từ file và lưu vào mảng.

c. Xuất danh sách các sinh viên (từ mảng) theo dạng bảng.

d. Tìm một sinh viên theo mã sinh viên.

e. Xuất danh sách các sinh viên có điểm <5.

f. Đếm xem có bao nhiêu sinh viên có điểm <5.

g. Tính điểm trung bình của các sinh viên có năm sinh 1992.

k. Xuất các sinh viên có điểm lớn nhất.

l. Xóa các sinh viên có điểm = 0.

PHỤ LỤC A

Bảng mã ASCII chuẩn (ASCII Standard Character Set)

Char	Ctrl	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex
NUL	^@	0	00	space	32	20	@	64	40	`	96	60
SOH	^A	1	01	!	33	21	A	65	41	a	97	61
STX	^B	2	02	"	34	22	B	66	42	b	98	62
ETX	^C	3	03	#	35	23	C	67	43	c	99	63
EOT	^D	4	04	$	36	24	D	68	44	d	100	64
ENQ	^E	5	05	%	37	25	E	69	45	e	101	65
ACK	^F	6	06	&	38	26	F	70	46	f	102	66
BEL	^G	7	07	'	39	27	G	71	47	g	103	67
BS	^H	8	08	(	40	28	H	72	48	h	104	68
HT	^I	9	09	)	41	29	I	73	49	i	105	69
LF	^J	10	0A	*	42	2A	J	74	4A	j	106	6A
VT	^K	11	0B	+	43	2B	K	75	4B	k	107	6B
FF	^L	12	0C	,	44	2C	L	76	4C	l	108	6C
CR	^M	13	0D	-	45	2D	M	77	4D	m	109	6D
SO	^N	14	0E	.	46	2E	N	78	4E	n	110	6E
SI	^O	15	0F	/	47	2F	O	79	4F	o	111	6F
DLE	^P	16	10	0	48	30	P	80	50	p	112	70
DC1	^Q	17	11	1	49	31	Q	81	51	q	113	71
DC2	^R	18	12	2	50	32	R	82	52	r	114	72
DC3	^S	19	13	3	51	33	S	83	53	s	115	73
DC4	^T	20	14	4	52	34	T	84	54	t	116	74
NAK	^U	21	15	5	53	35	U	85	55	u	117	75
SYN	^V	22	16	6	54	36	V	86	56	v	118	76
ETB	^W	23	17	7	55	37	W	87	57	w	119	77
CAN	^X	24	18	8	56	38	X	88	58	x	120	78
EM	^Y	25	19	9	57	39	Y	89	59	y	121	79
SUB	^Z	26	1A	:	58	3A	Z	90	5A	z	122	7A
ESC	^[	27	1B	;	59	3B	[	91	5B	{	123	7B
FS	^\	28	1C	<	60	3C	\	92	5C	|	124	7C
GS	^]	29	1D	=	61	3D	]	93	5D	}	125	7D
RS	^^	30	1E	>	62	3E	^	94	5E	~	126	7E
US	^_	31	1F	?	63	3F	_	95	5F	delete	127	7F

Bảng mã ASCII mở rộng (ASCII Extended Character Set)

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	│	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	⊛	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

PHỤ LỤC B

HƯỚNG DẪN SỬA LỖI CỦA CHƯƠNG TRÌNH

I. TỔNG QUAN VỀ CÁC LOẠI LỖI

Có 3 loại lỗi đối với một chương trình:

1. Lỗi cú pháp (Syntax Errors):

- Mỗi ngôn ngữ lập trình đều có một bộ quy tắc về ngữ pháp riêng mà người lập trình phải tuân thủ. Lỗi cú pháp xảy ra khi mã nguồn (source code) không tuân thủ theo quy tắc này.

- Trình biên dịch sẽ phát hiện ra lỗi này tại thời điểm biên dịch chương trình và hiển thi các lỗi này trong danh sách lỗi (Error list)

2. Lỗi thời gian chạy (Runtime Errors):

- Các lỗi xảy ra trong quá trình thực thi chương trình được gọi là lỗi thời gian chạy.

- Khi một lỗi thời gian chạy xảy ra, máy tính dừng việc thực hiện chương trình và hiển thị một câu thông báo lỗi.

3. Lỗi logic (Logical Errors):

- Các lỗi trong logic của chương trình được gọi là lỗi logic.

- Trình biên dịch không thể phát hiện các lỗi logic. Một chương trình có các lỗi logic được biên dịch (dịch) và chạy thành công, nhưng nó không cho kết quả chính xác.

- Khi chương trình cho ra kết quả sai hoặc không cho ra kết quả, sau khi xem lại code mà vẫn không tìm được nguyên nhân, chúng ta nên dùng công cụ Debug để kiểm tra cách thức thi hành của chương trình

hoặc xem kết quả chạy chương trình tại những thời điểm (dòng lệnh) cần thiết.

4. Các bước thông thưởng để sửa lỗi một chương trình C/C++:

II. CÁCH SỬA LỖI CÚ PHÁP

1. Cách xem lỗi cú pháp

- Để biên dịch chương trình tìm lỗi cú pháp: Chọn view/compile (Ctrl+F7) hay thực thi luôn chương trình Start Debugging (F5), khi đó nếu chương trình có lỗi cú pháp thì danh sách lỗi sẽ được thông báo.

- Để chọn chế độ hiển thị danh sách lỗi: Chọn View/Error list

2. Cách sửa lỗi cú pháp

- Nhấp đúp vào mỗi dòng thông báo lỗi, chương trình dịch sẽ chỉ ra dòng bị lỗi tương ứng trong chương trình. Ta cần đọc hiểu câu thông báo lỗi để sửa lỗi nhanh mà không phải "đoán mò" rất mất thời gian.

- Lưu ý:

+ Cần chú ý là warning (cảnh báo) không phải là lỗi.

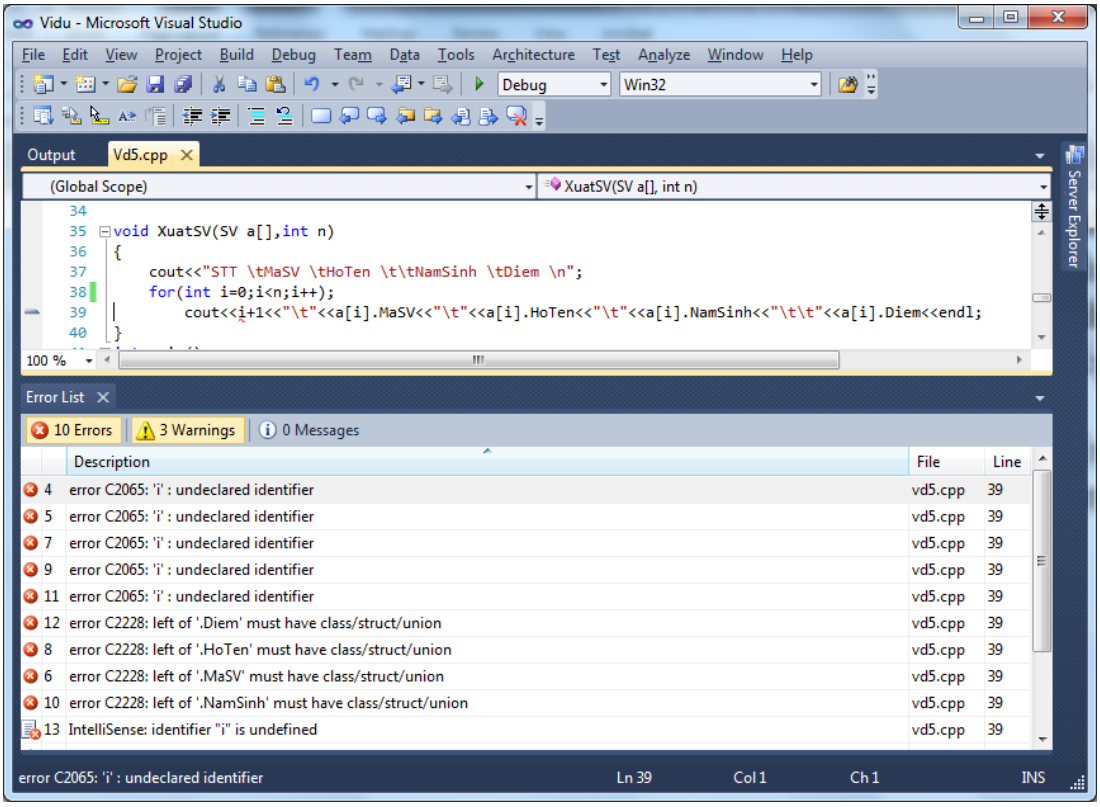
+ Trong danh sách lỗi, chỉ có lỗi đầu tiên (lỗi được đánh số nhỏ nhất và không phải lúc nào cũng được đánh số từ 1) là được thông báo chính xác, các lỗi tiếp theo sau có thể là lỗi phát sinh do lỗi đầu tiên. Vì vậy để tránh mất thời gian ta ***chỉ sửa lỗi đầu tiên*** sau đó dịch lại chương trình để tìm lỗi tiếp theo.

Trong hình ở trang sau, chương trình có 10 lỗi cú pháp và lỗi đầu tiên

được đánh số 4 (vì trước đó có 3 cảnh báo). Sau khi ta sửa được lỗi đầu tiên, các lỗi khác sẽ "biến mất" sau khi biên dịch lại chương trình.

+ Khi chương trình dịch chạy đến dòng trong thông báo lỗi thì phát hiện ra lỗi nên có khả năng lỗi không nằm trên dòng này mà nằm trên dòng trước đó.

Trong hình ở trang sau, trình biên dịch phát hiện lỗi ở dòng 39 nhưng lỗi lại nằm ở dòng 38 (do dư dấu ';' ở cuối dòng 38)



3. Các câu thông báo lỗi cú pháp thường gặp

Do tâm lý ngại đọc tiếng Anh nên khi gặp thông báo lỗi nhiều bạn SV thường không đọc mà chỉ "sửa mò" nên mất rất nhiều thời gian. Tuy nhiên, trên thực tế số câu thông báo lỗi thường gặp không nhiều nên các bạn cần chịu khó đọc hiểu và tra từ nếu cần. Sau một hai lần thì các bản sẽ nhớ và

hiểu nên sẽ sửa lỗi rất nhanh.

Chắc chắn rằng nếu bạn không thể đọc hiểu các câu thông báo lỗi thì không thể lập trình được.

Các bạn cũng có thể tra cứu nhanh các câu thông báo lỗi được trình bày ở trang sau. Tuy nhiên các bạn vẫn nên ghi nhớ ý nghĩa của các từ tiếng Anh trong các câu thông báo lỗi để sau này có thể đọc được nhiều câu thông báo lỗi khác nữa.

STT	Câu thông báo lỗi	Ý nghĩa
1	**C**annot convert parameter 1 from ‘float’ to ‘int&’	*Không thể chuyển đổi tham số thứ nhất từ kiểu ‘float’ sang kiểu ‘&int’*
2	**C**annot open include file : ‘Mylib.h’: No such file or directory	*Không thể mở file bao gồm: ‘Mylib.h’: Không có file hay thư mục nào như thế*
3	**E**nd of file found before the left brace '{'	*Kết thúc file nằm trước dấu ngoặc móc trái ‘{‘ (do thiếu ngoặc ‘}’ kết thúc hàm main())*
4	**E**ntry point must be defined	*Điểm vào của chương trình phải được định nghĩa (chưa có hàm main())*
5	**E**xpected constant expression	*Chờ đợi biểu thức hằng*
6	**F**unction does not take 1 arguments	*Hàm không phải có 1 đối số*

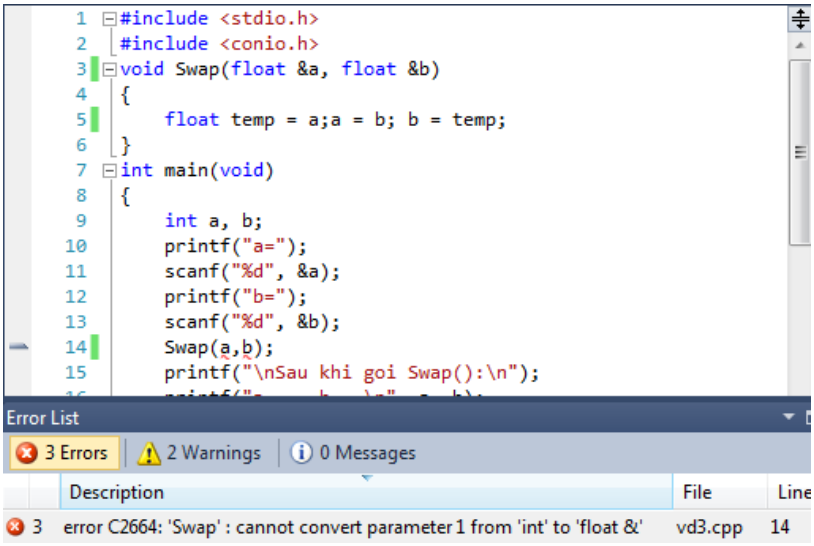
7	**F**unction …. already has a body	*Hàm … đã có rồi*
8	'x' followed by 'void' is illegal (did you forget a ';'?)	*'x' được theo sau bởi void là không hợp lệ (có phải bạn quên dấu ':' không ?)*
9	'x' **i**s not a member of struct	*'x' không phải là thành viên của struct*
10	**I**dentifier not found	*Định danh không tìm thấy*
11	**I**llegal else without matching if	*else không ứng với if thì không hợp lệ*
12	**m**ain already defined in …	*Hàm main() đã được định nghĩa trong …*
13	**M**issing ': ' before 'if'	*Thiếu dấu ';' trước 'if'*
14	**N**ewline in constant	*Dòng mới trong chuỗi hằng (do thiếu dấu " để đóng chuỗi)*
15	'Tong': **m**ust be return a value	*Hàm 'Tong' phải trả về một giá trị*
16	**L**eft of '.HoTen' must have class/struct/union	*Bên trái của '.HoTen' phải là class/struct/union*
17	**L**ocal function definitions are illegal	*Định nghĩa hàm cục bộ (nằm trong hàm khác) là không hợp lệ*
18	'int i' : **r**edefinition	*Định nghĩa lại biến i*
19	**U**ndeclared identifier	*Định danh chưa khai báo*

20	'void' **f**unction returning a value	*Hàm 'void ' trả về giá trị (là không hợp lệ)*

4. Một số minh họa về cách sửa lỗi cú pháp

Các ví dụ trong phần này được trình bày theo thứ tự các câu thông báo lỗi ở mục 3

Ví dụ 1:



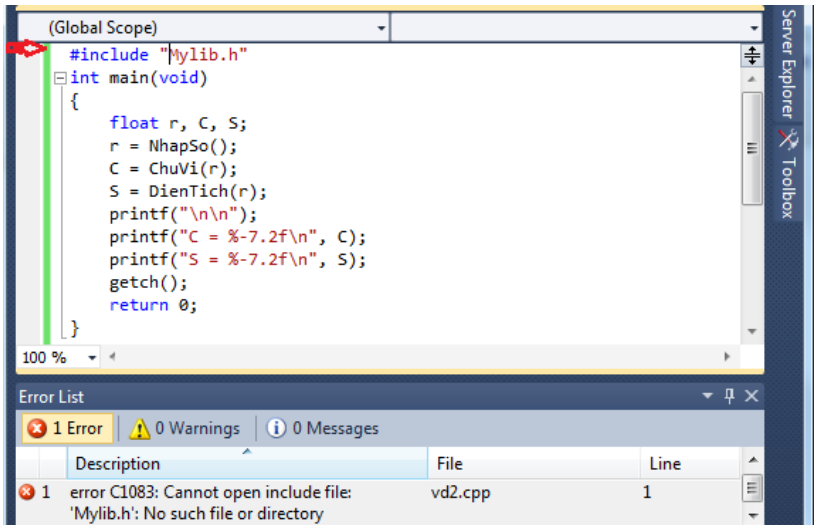
Lỗi sai ở dòng 14: Hàm 'Swap' không thể chuyển đổi tham số thứ 1 từ kiểu'int' sang kiểu '&float'

Cách sửa lỗi:

Cách 1: Phải sửa lại phần khai báo biến a,b ở dòng 9 có kiểu là 'float' để phù hợp với kiểu của tham số được truyền cho hàm Swap như khai báo ở dòng 3.

Cách 2: Phải sửa lại phần khai báo tham số được truyền cho hàm Swap ở dòng 3 là '&int' . Khi đó biến temp cũng phải được khai báo kiểu int.

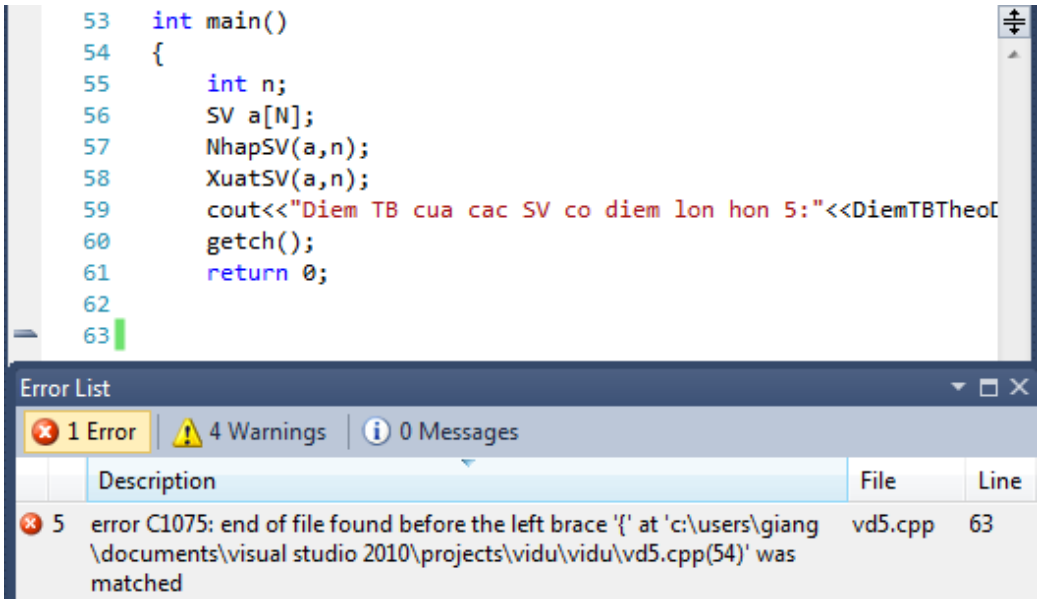
Ví dụ 2:



Lỗi sai : Không thể mở file bao gồm: ‘Mylib.h’: Không có file hay thư mục nào như thế.

Cách sửa lỗi: Cần kiểm tra lại tên file ‘Mylib.h’ có đúng không. Nếu file này không nằm cùng thư mục với file chương trình thì phải thêm đường dẫn.

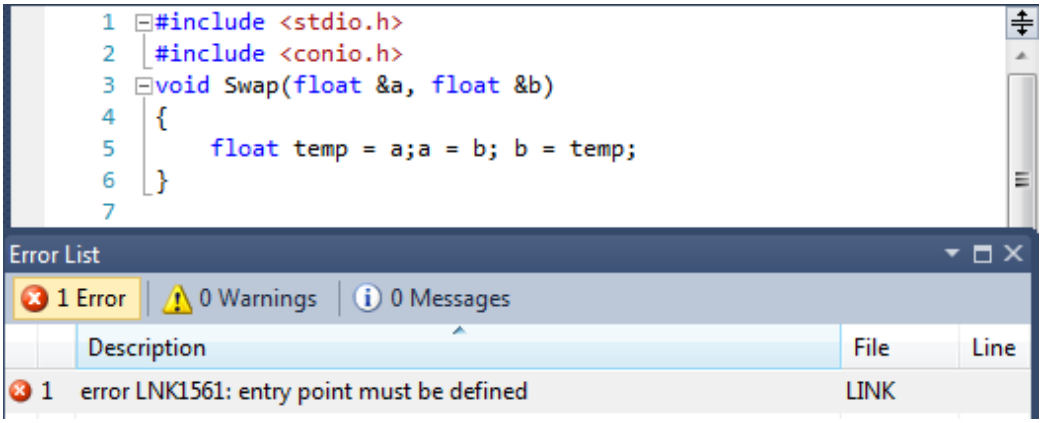
Ví dụ 3:



Lỗi sai : Kết thúc file nằm trước dấu ngoặc móc trái ‘{‘

Cách sửa lỗi: Thêm ngoặc ‘}’ kết thúc hàm main()

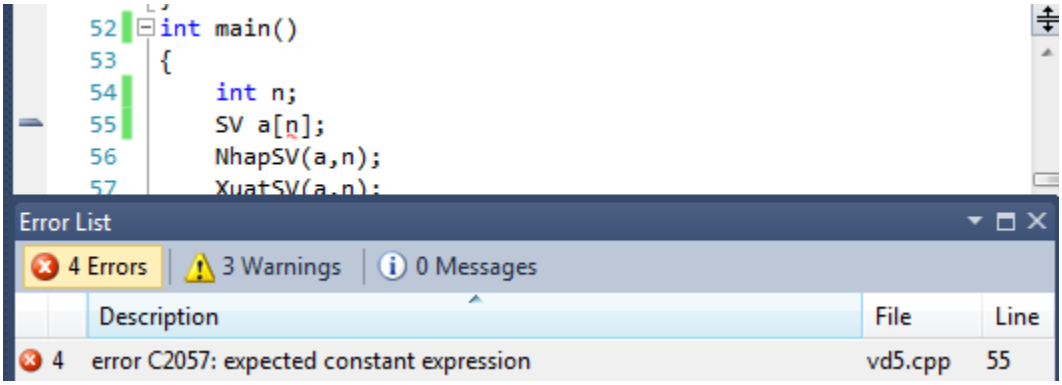
Ví dụ 4:



Lỗi sai : Điểm vào của chương trình phải được định nghĩa (chưa có hàm main())

Cách sửa lỗi: Cần bổ sung thêm hàm main()

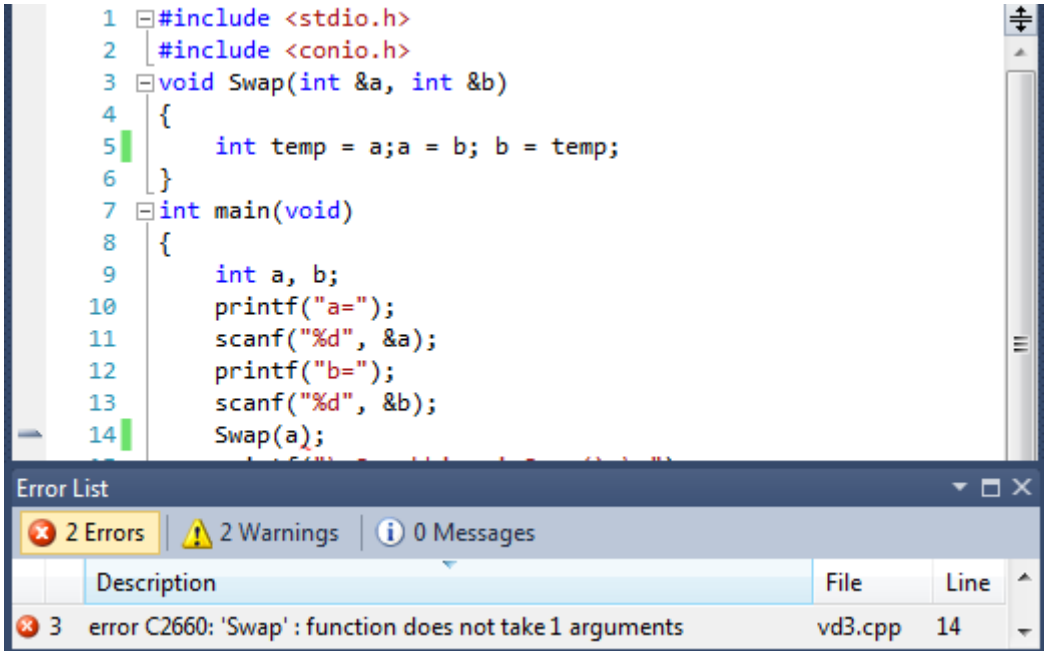
Ví dụ 5:



Lỗi sai ở dòng 55: Chờ đợi biểu thức hằng (do ngôn ngữ C không cho phép khai báo kích thước mảng là biến)

Cách sửa lỗi: Thay kích thước mảng bằng hằng số, ví dụ : **SV a[100];**

Ví dụ 6:



Lỗi sai ở dòng 14: Hàm ‘Swap’ không phải có 1 tham số

Cách sửa lỗi: Bổ sung thêm tham số cho câu lệnh gọi hàm Swap() ở dòng 14**: Swap(a,b);**

Ví dụ 7:

```
35  void XuatSV(SV a[],int n)
36  {
37      cout<<"STT \tMaSV \tHoTen \t\tNamSinh \tDiem \n";
38      for(int i=0;i<n;i++)
39          cout<<i+1<<"\t"<<a[i].MaSV<<"\t"<<a[i].HoTen<<"\t"<<a[
40  }
41  float DiemTBTheoDK(SV a[],int n)
42  {
43      float tong=0;int dem=0;
44      for(int i=0;i<n;i++)
45          if (a[i].Diem>5)
46          {
47              tong=tong+a[i].Diem;
48              dem++;
49          }
50      return tong/dem;
51  }
52  void XuatSV(SV a[],int n)
53  {
54      cout<<"STT \tMaSV \tHoTen \t\tNamSinh \tDiem \n";
55      for(int i=0;i<n;i++)
56          cout<<i+1<<"\t"<<a[i].MaSV<<"\t"<<a[i].HoTen<<"\t"<<a[
57  }
```

Error List

3 Errors | 3 Warnings | 0 Messages

	Description	File	Line
4	error C2084: function 'void XuatSV(SV [],int)' already has a body	vd5.cpp	53

Lỗi sai ở dòng 53: Hàm void XuatSV() đã có rồi

Cách sửa lỗi: Xóa hàm void XuatSV() ở dòng 53

Ví dụ 8:

```
4   #define N 100
5   using namespace std;
6   typedef struct SV
7       {
8           int NamSinh;
9           float Diem;
10          char MaSV[10],HoTen[50];
11      }
12  void NhapSV(SV a[],int &n)
```

100 %

Output | Error List

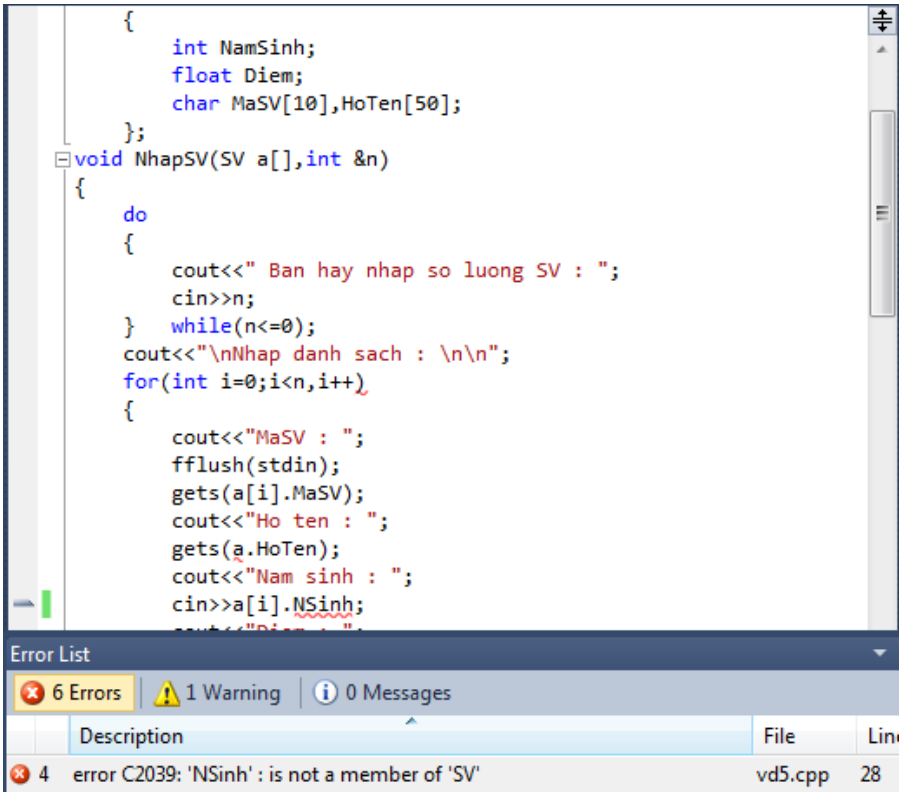
7 Errors | 0 Warnings | 0 Messages

	Description	File	Line
1	error C2628: 'SV' followed by 'void' is illegal (did you forget a ';'?)	vd5.cpp	12

Lỗi sai ở dòng 12: ‘SV’ được theo sau bởi ‘void’ thì không hợp lệ (do khai báo kiểu ‘SV’ chưa xong)

Cách sửa lỗi: Bổ sung thêm dấu ‘;’ kết thúc câu lệnh khai báo kiểu SV ở cuối dòng 11.

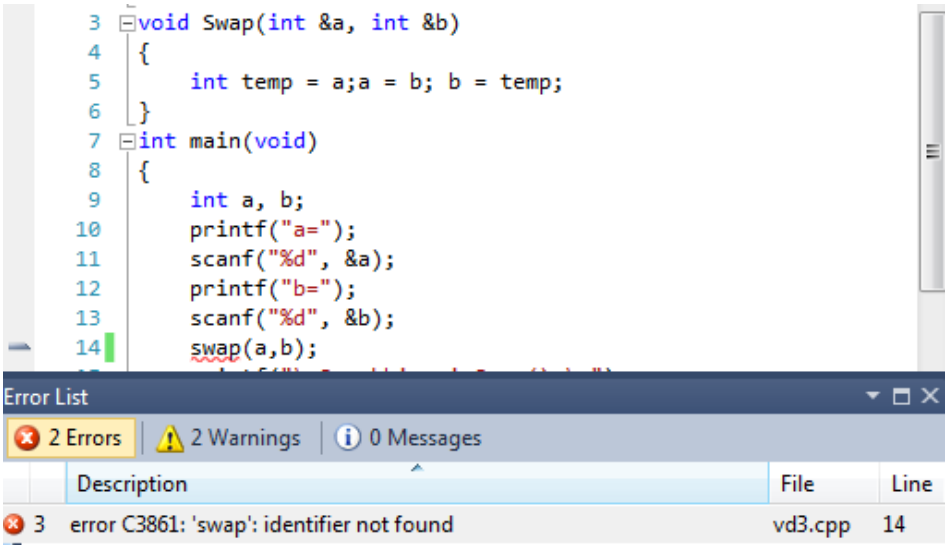
Ví dụ 9:



Lỗi sai ở dòng 29: ‘NSinh’ không phải là thành phần của struct

Cách sửa lỗi: Thay ‘NSinh’ bằng ‘NamSinh’ như phần khai báo của struct SV

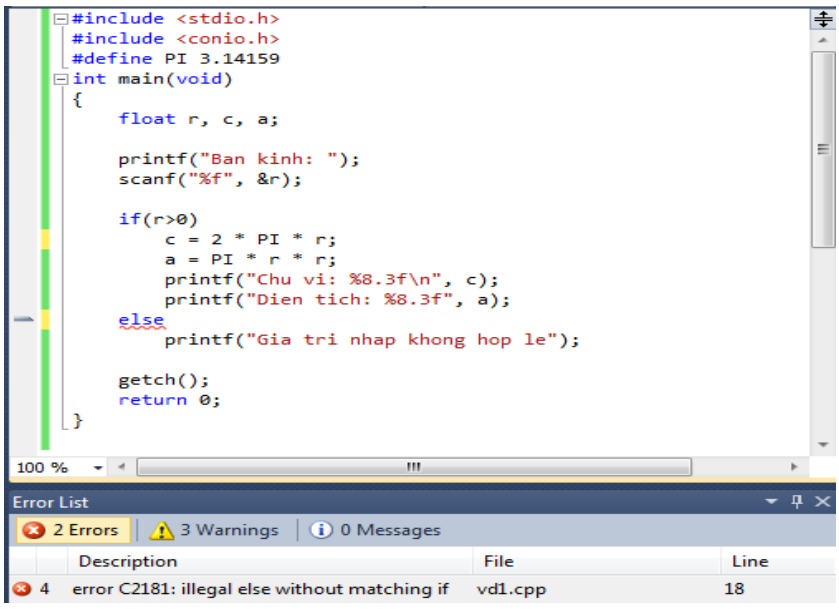
Ví dụ 10:



Lỗi sai ở dòng 14: 'SWAP' là định danh không tìm thấy

Cách sửa lỗi: Thay 'SWAP' bằng 'Swap' giống như tên hàm đã cài đặt ở trên

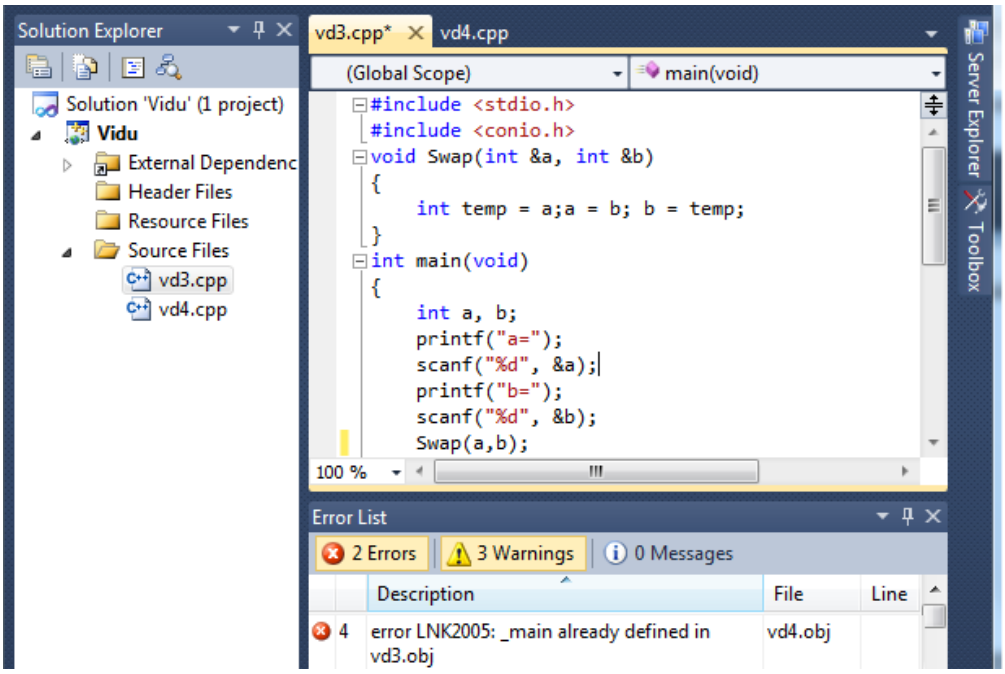
Ví dụ 11:



Lỗi sai ở dòng 18: else không ứng với if thì không hợp lệ

Cách sửa lỗi: Thêm dấu {} trong câu lệnh if phía trên else (vì if có 4 câu lệnh con nên cần có ngoặc)

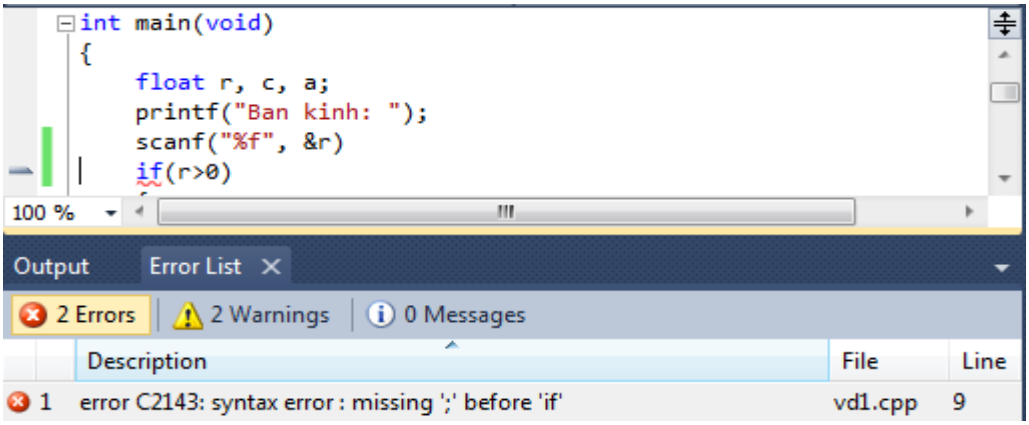
Ví dụ 12:



Lỗi sai : Hàm main() đã được định nghĩa

Cách sửa lỗi: Cách sửa lỗi: Dời chỗ (remove) file chương trình vd4.cpp (Nhấp phải lên file vd4.cpp và chọn remove). Lưu ý, tại mỗi thời điểm chỉ cho phép thực thi một hàm main() nên không được để nhiều hơn một file chương trình (.cpp) trong mỗi project khi thực thi chương trình.

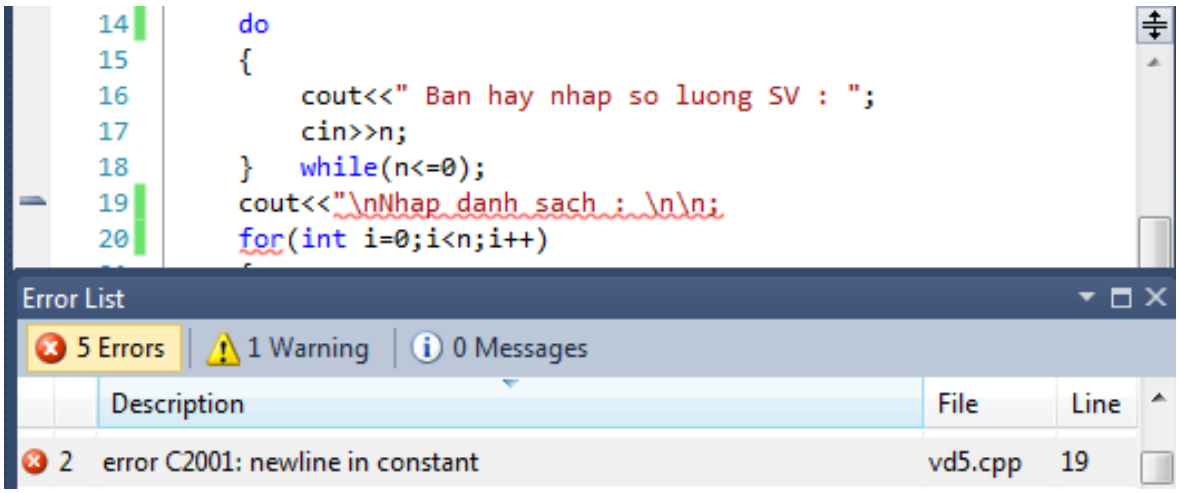
Ví dụ 13:



Lỗi sai ở dòng 9: Thiếu dấu ‘;’ trước ‘if’

Cách sửa lỗi: Thêm dấu ‘;’ ở cuối dòng 9

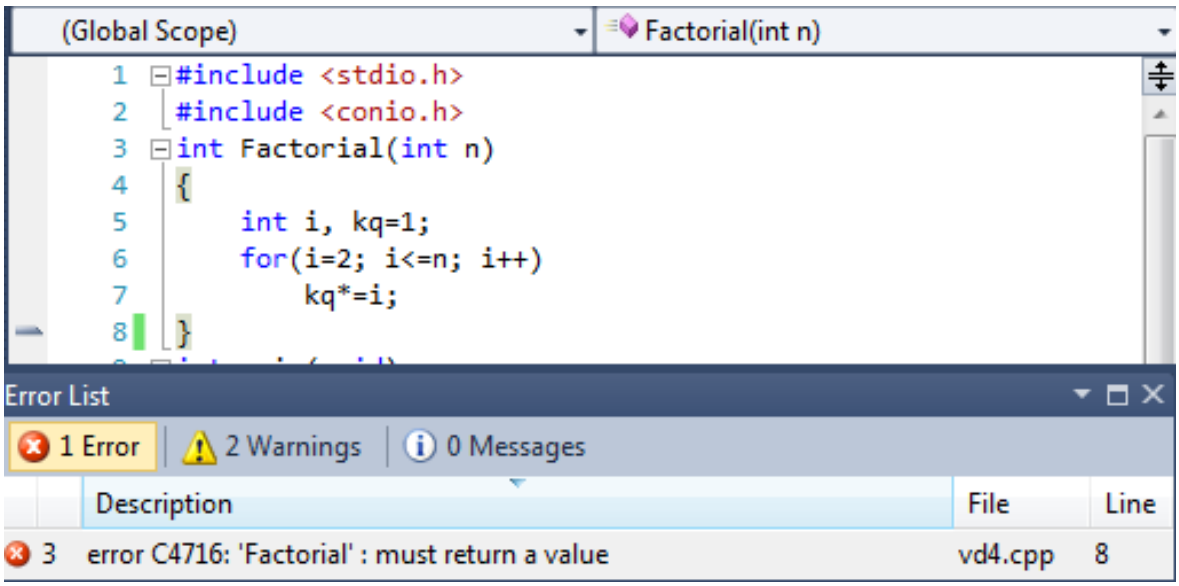
Ví dụ 14:



Lỗi sai ở dòng 19: Dòng mới trong (chuỗi) hằng

Cách sửa lỗi: Thêm dấu “ để đóng chuỗi ở cuối dòng 19

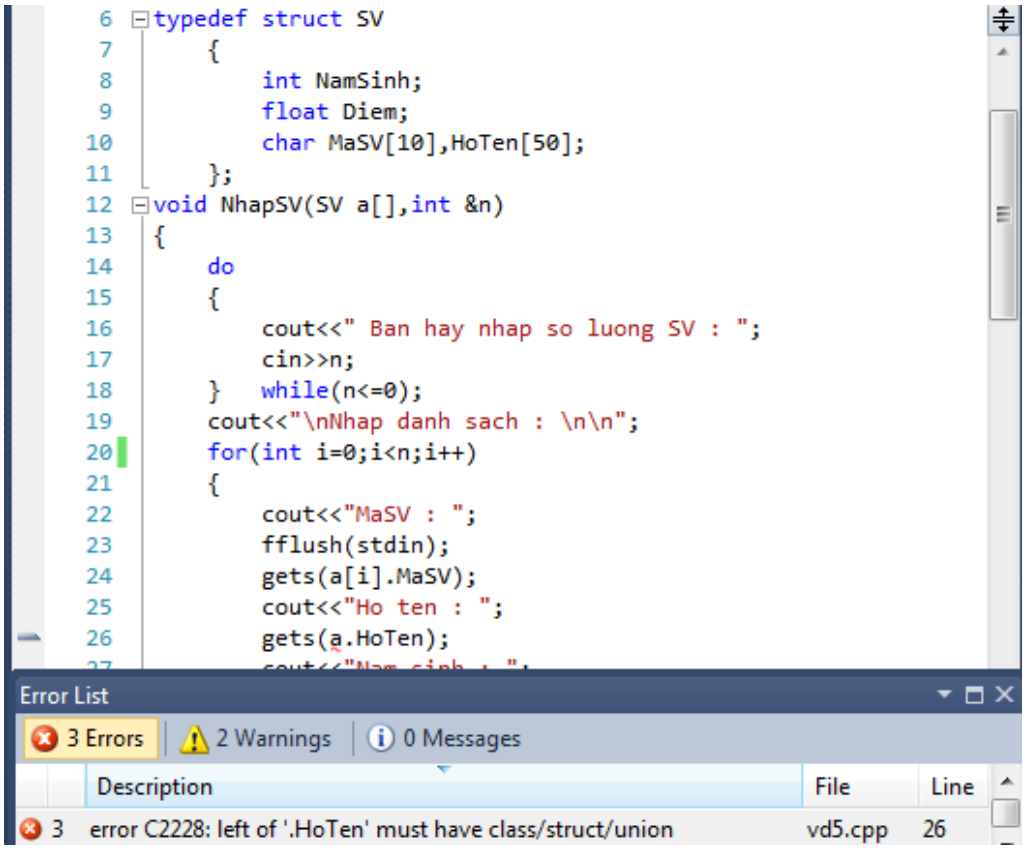
Ví dụ 15:



Lỗi sai ở dòng 8: Hàm 'Factorial' phải trả về giá trị (do dòng 3 đã khai báo hàm này trả về kiểu int)

Cách sửa lỗi: Bổ sung câu lệnh: return kq; ở cuối hàm.

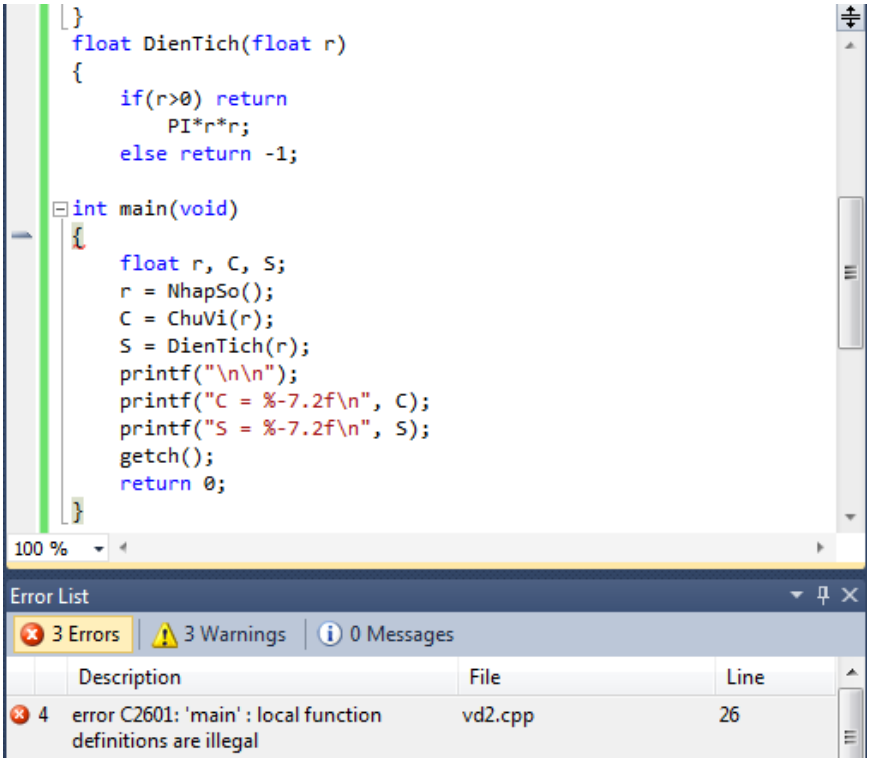
Ví dụ 16:



Lỗi sai ở dòng 26: Bên trái của '.Hoten' phải là class/struct/union (trong trường hợp này là struct vì HoTen là 1 thành phần của struct SV)

Cách sửa lỗi: Sửa lại là: gets(a[i].Hoten); (a là biến lưu địa chỉ của mảng a chứa danh sách sinh viên chứ không phải là biến lưu thông tin của một sinh viên)

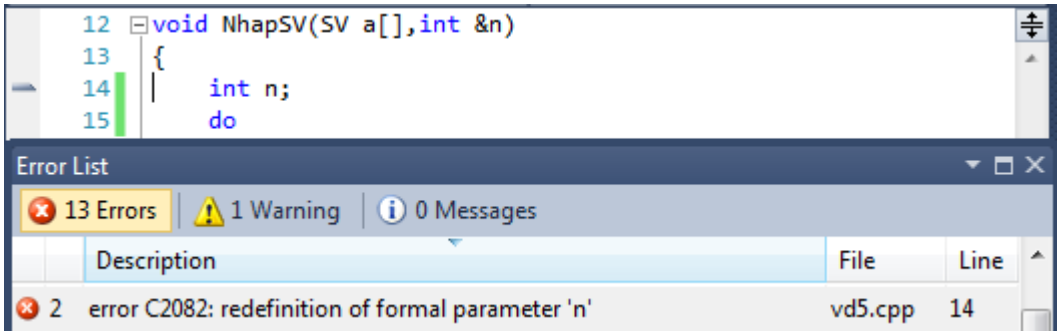
Ví dụ 17:



Lỗi sai ở dòng 26: ‘main’ : định nghĩa hàm cục bộ (nằm trong hàm khác) là không hợp lệ.

Cách sửa lỗi: Do hàm main() đang nằm trong hàm DienTich() nên cần đóng ngoặc kết thúc hàm DienTich()

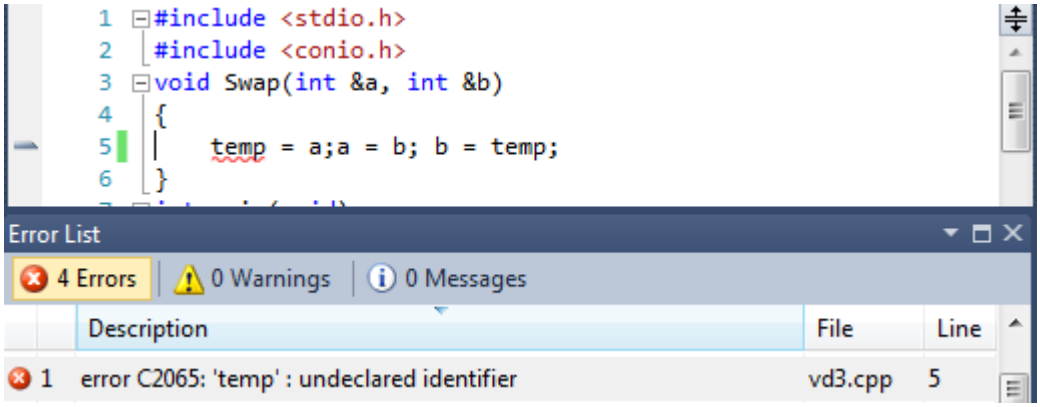
Ví dụ 18:



Lỗi sai ở dòng 14: Định nghĩa lại tham số hình thức ‘n’

Cách sửa lỗi: Bỏ khai báo lại biến n ở dòng 14.

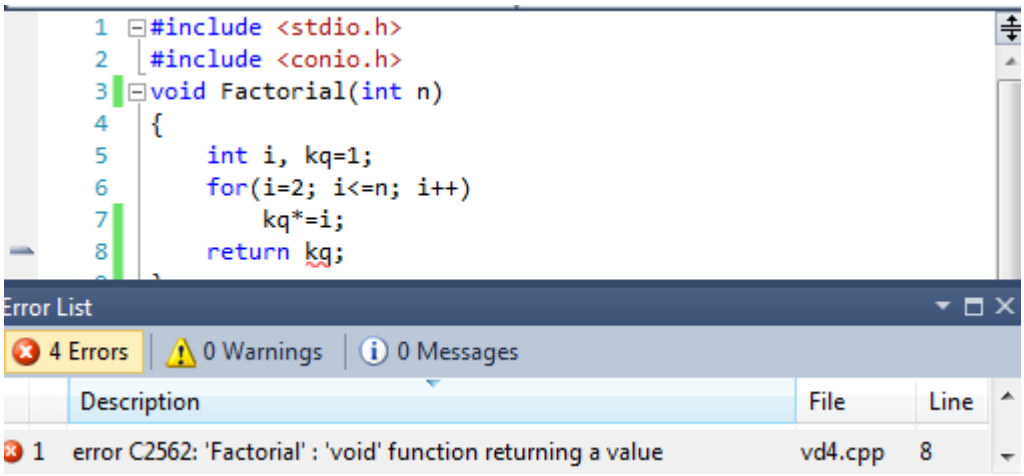
Ví dụ 19:



Lỗi sai ở dòng 5: ‘temp’ chưa khai báo

Cách sửa lỗi: Khai báo biến ‘temp’

Ví dụ 20:



Lỗi sai ở dòng 8: Hàm ‘Factorial’ là hàm void mà trả về giá trị

Cách sửa lỗi: Phải sửa lại kiểu trả về của hàm là int (trong phần khai báo ở dòng 3)

III. CÁCH SỬA LỖI THỜI GIAN CHẠY

Các lỗi thời gian chạy có thể xảy ra vì các lý do sau đây:

+ Khi chương trình cố gắng thực hiện một hoạt động bất hợp pháp như phân chia một số cho số không.

+ Việc nhập dữ liệu không thành công.

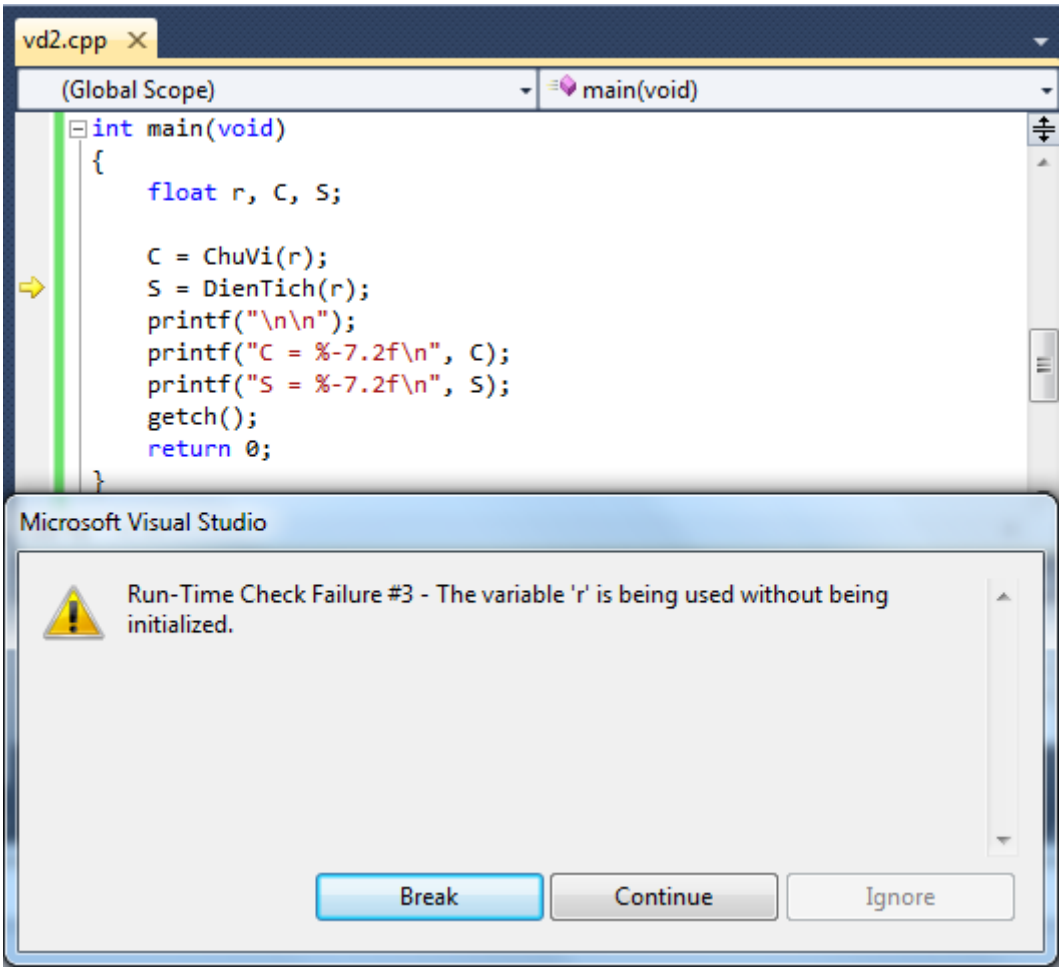
+ Xảy ra vấn đề phần cứng như lỗi đĩa cứng hoặc ổ đĩa cứng hoặc lỗi máy in,…

1. Cách sửa lỗi thời gian chạy

- Nhấp chọn nút Break, chương trình dịch sẽ chỉ ra dòng bị lỗi tương ứng trong chương trình. Ta cần đọc hiểu câu thông báo lỗi để sửa lỗi nhanh.

2. Các ví dụ minh họa về lỗi thời gian chạy

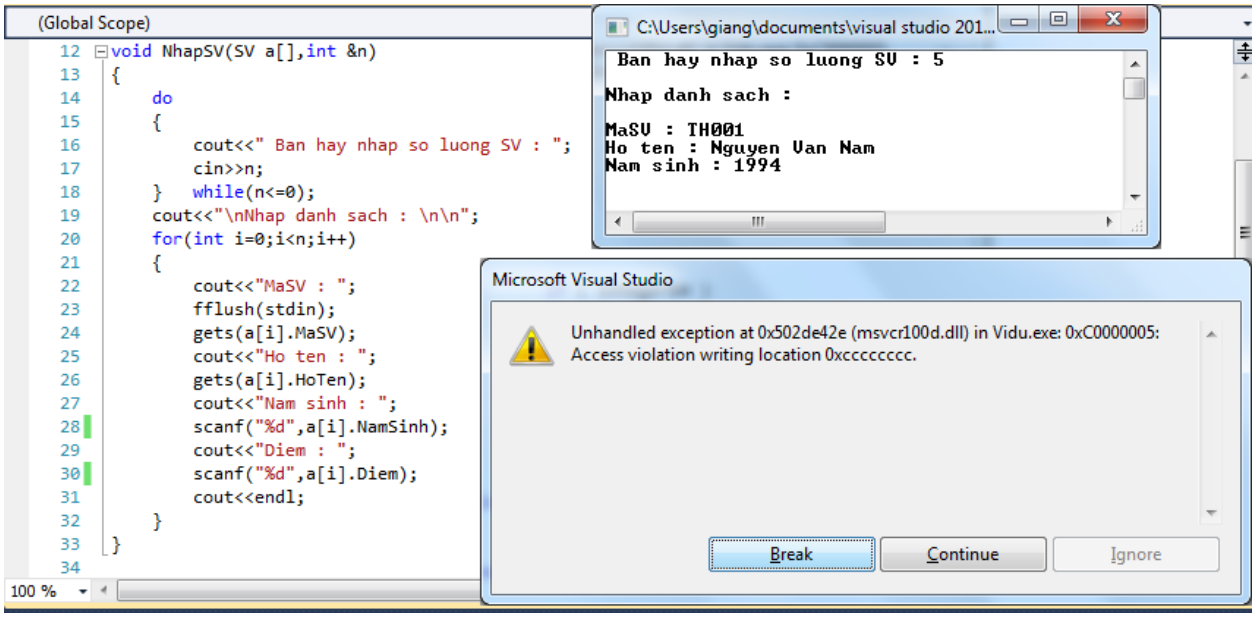
Ví dụ 1:



Lỗi sai: Biến ‘r’ đang được sử dụng mà chưa khởi tạo (giá trị ban đầu)

Cách sửa lỗi: Cần bổ sung câu lệnh nhập giá trị cho biến r trước khi gọi hàm ChuVi() và DienTich()

Ví dụ 2:



Lỗi sai: Lỗi ngoài tầm kiểm soát, ghi vào địa chỉ không được phép.

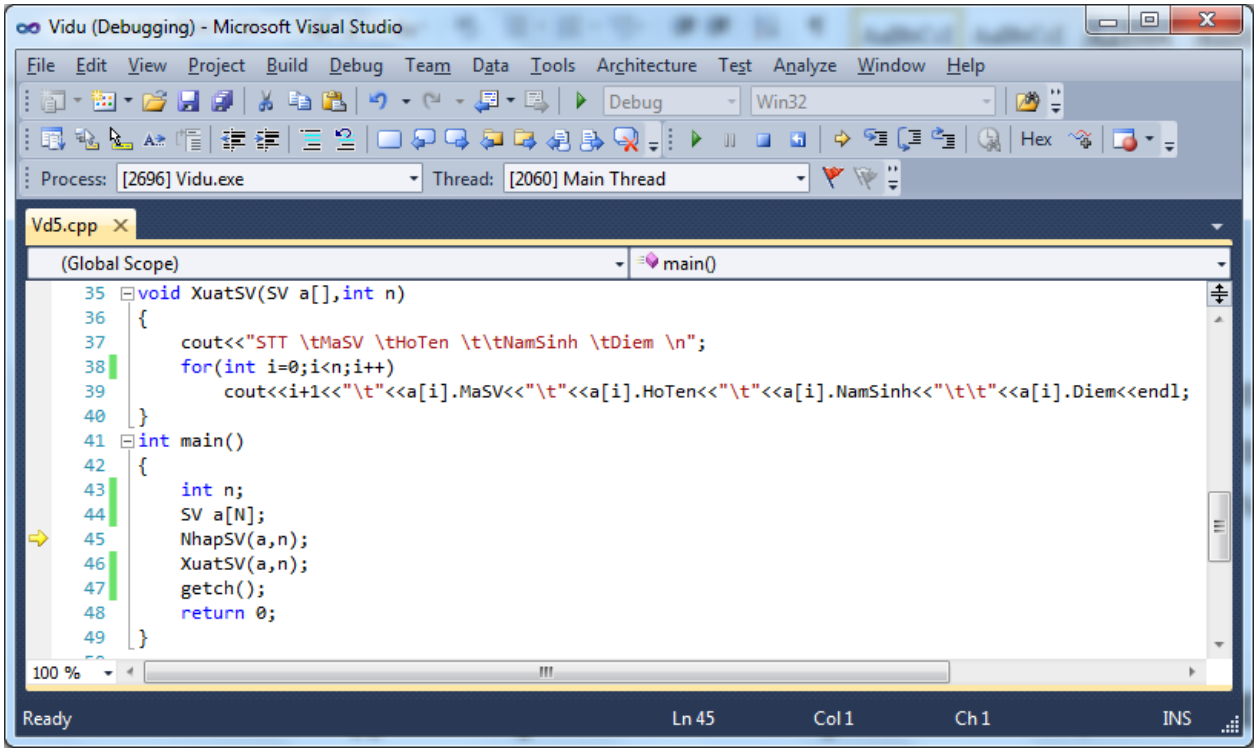
Cách sửa lỗi: Khi chương trình chạy xong câu lệnh nhập năm sinh của SV ở dòng 28 thì bị lỗi nên dễ thấy câu lệnh này bị sai cú pháp của hàm scanf(). Cần bổ sung toán tử & trước biến cần nhập: scanf("%d", &a[i].NamSinh);

IV. CÁCH SỬA LỖI LOGIC

Phần này sẽ trình bày cách sửa lỗi logic bằng công cụ Debug.

1. Chạy từng lệnh

Khi bắt đầu chạy debug từng lệnh (bằng cách nhấn phím F10 hay F11) thì điều khiển của chương trình (hiển thị bằng mũi tên màu vàng ở cột mốc trái của vùng soạn thảo) sẽ chạy từng dòng lệnh và luôn luôn bắt đầu từ hàm main() như hình sau:



a. Chạy lướt qua hàm: nhấn phím **F10** hoặc chọn **Debug → Step Over**

Nếu dòng hiện tại có lời gọi đến một hàm nào nào đó trong chương trình thì điều khiển của chương trình sẽ lướt qua hàm này chứ không đi vào để chạy từng lệnh trong hàm này.

b. Đi vào hàm: nhấn phím **F11** hoặc chọn **Debug → Step Into**

Nếu dòng hiện tại có lời gọi đến một hàm nào đó thì điều khiển sẽ đi vào để chạy từng lệnh trong hàm này.

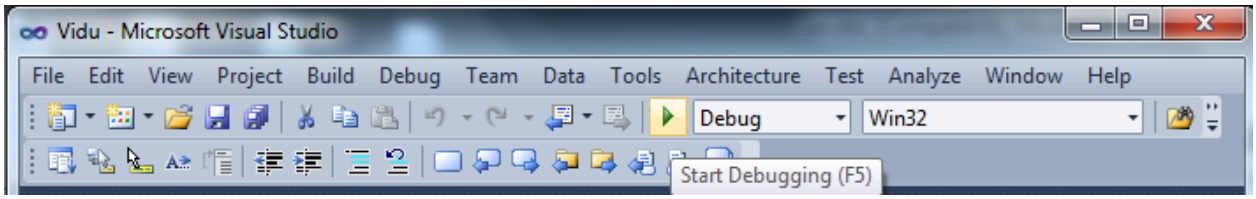
Hiển nhiên, nếu dòng hiện tại không chứa lời gọi đến một hàm nào khác thì nhấn phím F10 hay F11 là như nhau.

c. Chạy nhanh ra khỏi hàm (mà điều khiển của chương trình đang dừng): nhấn tổ hợp phím **Shift+F11** hoặc chọn **Debug → Step Out.**

2. Dừng chương trình tại các điểm dừng

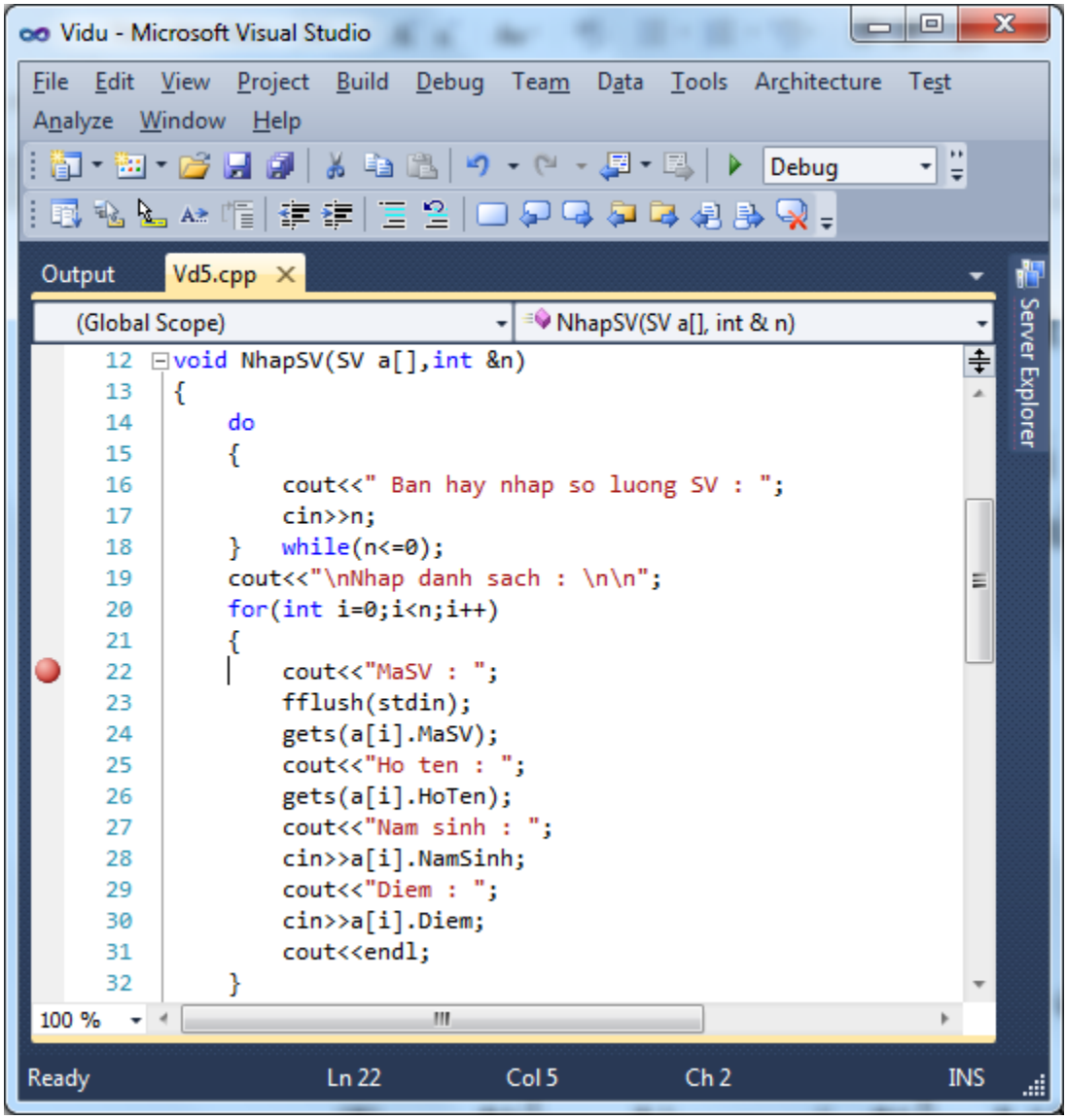
Nếu bạn không muốn mất thời gian chạy từng dòng lệnh của toàn bộ chương trình mà chỉ cần kiểm tra chương trình ở một vài lệnh hay thời điểm cần thiết nào đó thì ta sẽ tạo các bạn có thể tạo các điểm dừng (break point) để kiểm tra giá trị các biến hay cách thức chương trình chạy tại các điểm này.

Sau khi tạo điểm dừng bạn nhấn F5 hay phím Start Debugging, điều khiển của chương trình sẽ dừng tại breakpoint mà bạn đã tạo



a. Cách tạo một breakpoint:

Nhấp chuột vào cột mốc trái tại hàng tương ứng, khi đó, tại hàng tương ứng trong cột mốc trái sẽ có một nút tròn màu đỏ như trong hình sau:



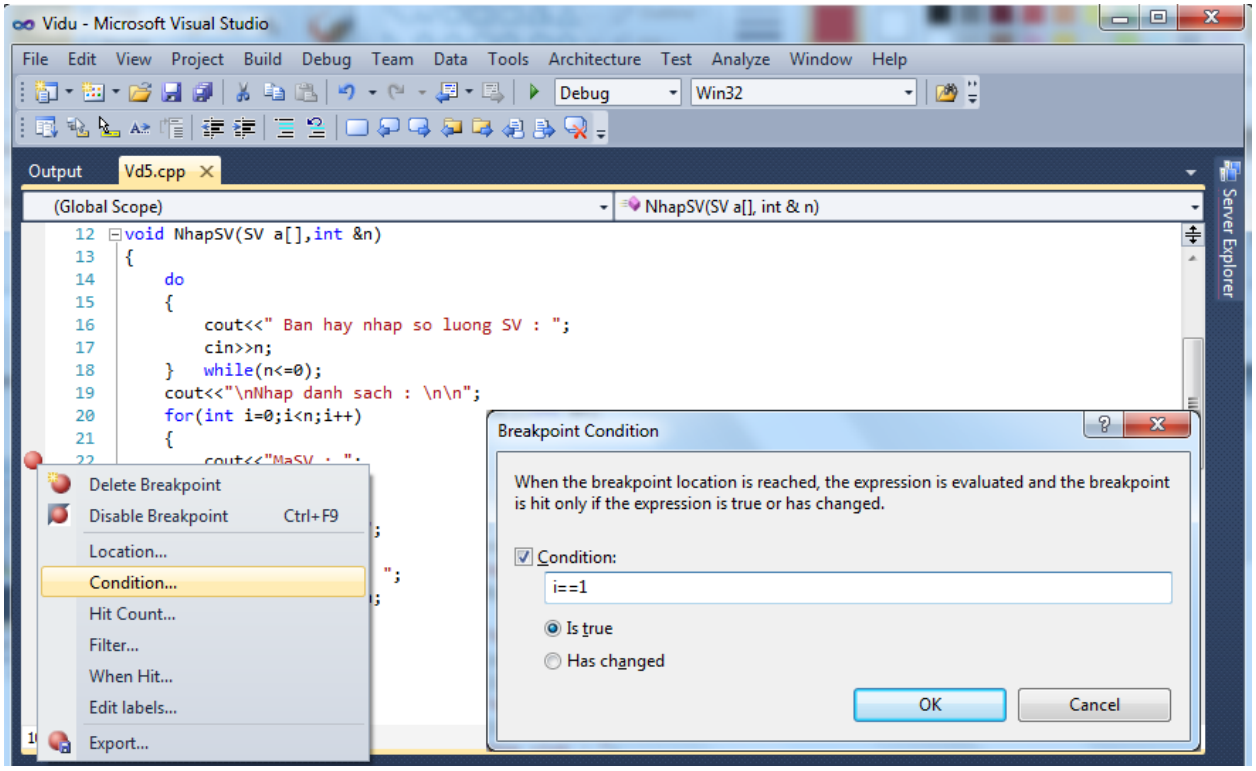
b. Hủy một breakpoint: Nhấp chuột vào breakpoint muốn hủy

c. Tạo điểm dừng có điều kiện (conditional breakpoint):

Nếu ta chỉ muốn chương trình sẽ dừng tại lần lặp hay lần gọi đệ quy cụ thể nào đó: Tạo một breakpoint tại hàng muốn dừng sau đó nhấp phải chuột phải tại breakpoint này và chọn **Condition** . Sau đó nhập điều kiện vào vùng condition.

Trong hình sau tôi tạo một breakpoint trong vòng lặp tại lần lặp thỏa

điều kiện là i==1.



d. Dừng chương trình. Nếu không muốn chạy Debug tiếp nữa: Nhấn tổ hợp phím **Shift+F5** hoặc chọn **Debug → Stop Debugging** để dừng chương trình.

3. Xem giá trị các biến khi debug

Việc xem giá trị các biến tại một thời điểm nào đó khi thực thi chương trình sẽ giúp ta kiểm tra xem giá trị các biến này có đúng không và cách thức chương trình thực hiện các lệnh mà ta đã viết để tìm ra nguyên nhân làm chương trình cho kết quả sai.

Ta có thể xem giá trị các biến bằng một trong 3 cửa sổ:

- **Autos**: Hiển thị giá trị tất cả các biến có trong dòng lệnh hiện tại và dòng lệnh trước đó.

- Locals: Hiển thị giá trị tất cả các biến cục bộ trong hàm hiện tại

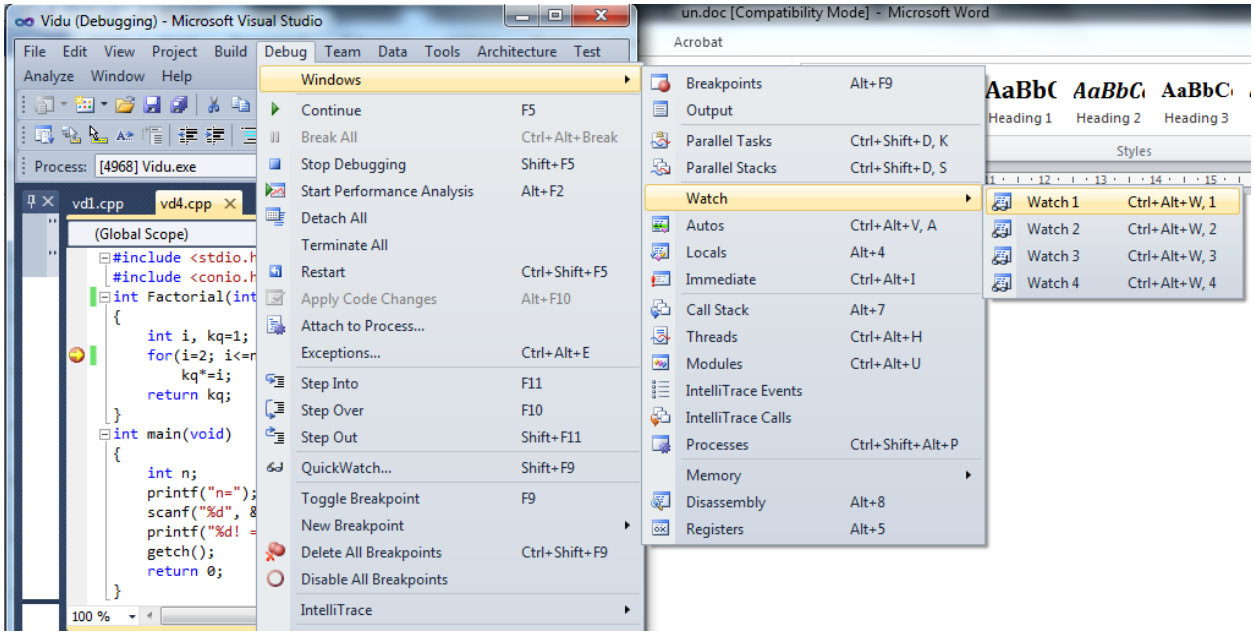
-Watch: Cho phép xem giá trị các biến hay biểu thức bất kỳ mà bạn muốn

Trong trường hợp các cũa sổ này không có sẵn chúng ta có thể mở bằng cách:

a. Cách mở cửa sổ Autos: nhấn tổ hợp phím **Ctrl+Alt+V,A** hoặc chọn **Debug → Windows → Autos**

b. Cách mở cửa sổ Local: nhấn tổ hợp phím **Alt+4** hoặc chọn **Debug → Windows → Locals**

c. Cách mở cửa sổ watch: chọn **Debug → Windows → Watch → Watch1**



4. Ví dụ tìm lỗi logic bằng công cụ Debug

Ví dụ: Cho một mảng a chứa n số thực, ta cần viết hàm tìm giá trị nhỏ nhất trong các số dương của mảng.

Nếu chỉ cần tìm giá trị nhỏ nhất (Min) của một dãy số, ta có 2 bước:

Bước 1: Gián giá trị Min ban đầu (thường gán là giá trị phần tử đầu tiên của mảng tức là a[0])

Bước 2: So sánh giá trị Min với các giá trị còn lại của mảng (a[i], i=1,…,n-1). Nếu thấy giá trị a[i] nào nhỏ hơn Min thì đổi lại giá trị Min bằng a[i] đó.

Hàm cài đặt như sau:

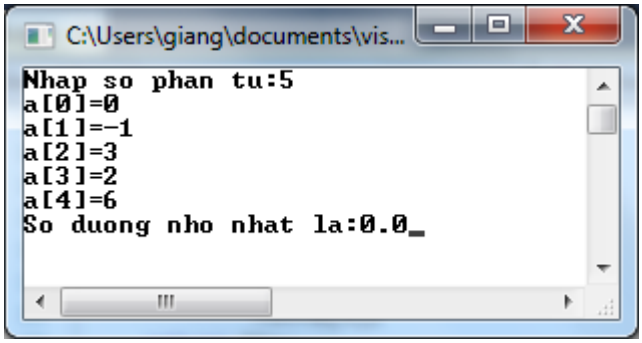
```
float MinMang(float a[],int n)
{
    float Min=a[0];
    for(int i=0;i<n;i++)
        if(a[i]<Min)
            Min=a[i];
    return Min;
}
```

Vậy để tìm Min của các số dương trong mảng, **thông thường các SV sẽ viết hàm như sau:**

```
float MinSoDuong(float a[],int n)
{
    float Min=a[0];
    for(int i=0;i<n;i++)
        if(a[i]>0&&a[i]<Min)
            Min=a[i];
```

```
    return Min;
}
```

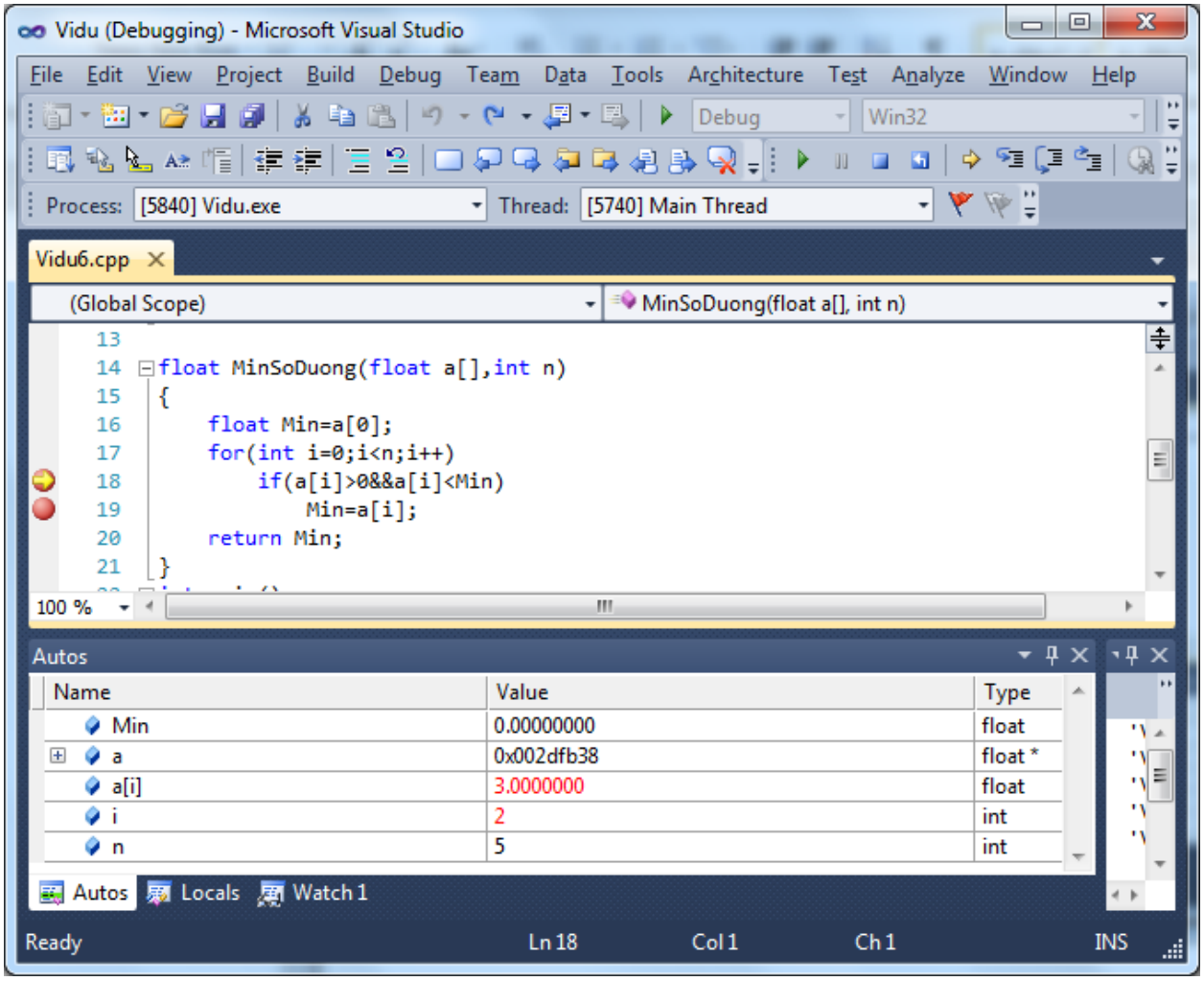
Tuy nhiên khi chạy thử kết quả sẽ bị sai (kết quả đúng là số 2):



Dĩ nhiên chương trình dịch sẽ không thông báo lỗi này vì là lỗi sai về logic của giải thuật. Vì vậy để dò tìm chúng ta có thể tự kiểm tra lại hàm MinSoDuong () bằng cách tự nhẩm hay viết ra nháp kết chạy tửng câu lệnh trên dãy số cho trước. Tuy nhiên nếu thấy khó khăn, ta có thể dùng công cụ Debug để tìm nhanh ra nguyên nhân mà chương trình cho kết quả sai.

Quá trình Debug gồm các bước:

Bước 1: Xác định câu lệnh cần kiểm tra. Trong ví dụ này là câu lệnh ở dòng 18, 19 vì đây là câu lệnh mấu chốt ảnh hưởng đến giá trị của biến Min. sau đó chạy từng lệnh (F10/F11) đến lệnh cần kiểm tra hoặc chọn luôn điểm dừng (breakpoint) tại dòng 18 và 19.



Bước 2: Nhấn F5 hay nhấp chọn Start Debugging để thực thi chương trình. Điều khiển chương trình sẽ dừng ở dòng lệnh có Breakpoint là dòng 18.

Bước 3: Chọn cửa sổ cần xem giá trị các biến. Kiểm tra sự thay đổi giá trị các biến. Trong ví dụ này tôi chọn cửa sổ Autos và quan sát giá trị biến Min thay đổi theo a[i] như thế nào.

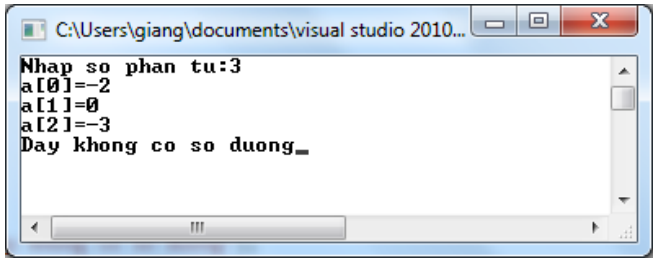
Bước 4: Nhấn F5 hay nhấp chọn Start Debugging nhiều lần, mỗi lần như thế điều khiển của chương trình sẽ nhảy đến Breakpoint kế tiếp.

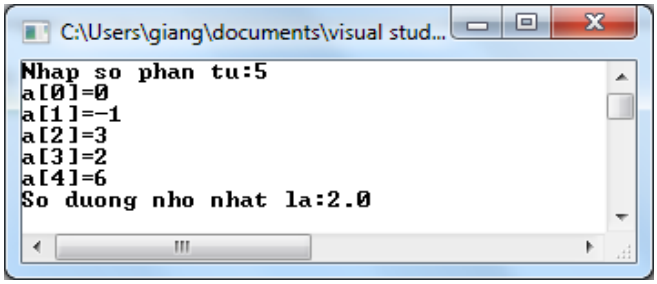
Trong trường hợp này ta nhận thấy ở tất cả các lần lặp, điều khiển của chương trình không bao giờ nhảy xuống dòng 19 tức là chương trình không thực hiện câu lệnh gán: a[i]=Min. Vì thế Min luôn bằng 0. Lý do duy nhất là điều kiện a[i]<Min không được thỏa (vì Min ban đầu được gán là 0 nên luôn nhỏ hơn các số a[i] dương).

Từ sự phân tích ở trên ta kết luận việc gán Min=a[0] (=0) như dòng 16 là sai. Ta cần phải gán Min là giá trị dương đầu tiên của dãy (trong ví dụ này là số 3).

Vì thế chương trình được sửa lại như sau:

```
#include<stdio.h>
#include<conio.h>
void NhapMang(float a[],int &n)
{
     printf("Nhap so phan tu:");
     scanf("%d",&n);
     for (int i=0;i<n;i++)
     {
         printf("a[%d]=",i);
         scanf("%f",&a[i]);
     }
 }
float MinSoDuong(float a[],int n)
{
     float Min=-1; int k;
     //Gán Min là số dương đầu tiên
     for(k=0;k<n;k++)
         if(a[k]>0)
          {
                Min=a[k];
                break;
           }
      //So sánh Min với các số dương còn lại và gán Min là
      //số nhỏ hơn
     for(int i=k+1;i<n;i++)
         if(a[i]>0&&a[i]<Min)
             Min=a[i];

     return Min;
     //Nếu dãy không có số dương hàm sẽ trả về giá trị Min
     //gán ban đầu là -1
}
int main()
{
     float a[100],t;
     int n;
     NhapMang(a,n);
     t=MinSoDuong(a,n);
     if (t==-1) printf("Day khong co so duong");
     else printf("So duong nho nhat la:%.1f",t);
     getch();
     return 0;
```





V. MỘT SỐ TỪ/CỤM TỪ TIẾNG ANH THƯỜNG GẶP TRONG CÁC CÂU THÔNG BÁO LỖI

STT	Từ/ cụm từ	Ý nghĩa	Ví dụ
1	access	*truy cập*	Access denied: *Truy cập bị từ chối*
2	already	*đã …rồi*	**m**ain already defined: *Hàm main đã định nghĩa rồi*
3	argument	*đối số*	**F**unction does not take 1 arguments: *Hàm không phải có 1 đối số*
4	constant	*hằng số*	Constant expression: *Biểu thức hằng*
5	convert	*biến đổi, chuyển đổi*	**C**annot convert parameter 1 from ‘float’ to ‘int&’: *Không thể chuyển đổi tham số thứ nhất từ kiểu ‘float’ sang kiểu ‘&int’*
6	define (v) definition (n)	*định nghĩa*	**E**ntry point must be defined: *Hàm main() đã được định nghĩa*

7	entry point	*điểm vào của chương trình (hàm main())*	
8	exception	*ngoại lệ*	Unhandled exception: *Ngoại lệ không điều khiển được*
9	expected	*chờ đợi, mong đợi*	**E**xpected constant expression: *Chờ đợi biểu thức hằng*
10	expression	*biểu thức*	Constant expression: *Biểu thức hằng*
11	find/found	*tìm thấy*	**I**dentifier not found: *Định danh không tìm thấy*
12	forget	*quên*	Did you forget a ';'?: *Bạn quên dấu';'?*
13	follow	*theo sau*	'x' **f**ollowed by 'void' is illegal
14	function	*hàm*	'void' **f**unction : *hàm 'void'*
15	identifier	*định danh (biến, hàm,…)*	**I**dentifier not found: *Định danh không tìm thấy*
16	illegal	*không hợp lệ*	**I**llegal else without matching if: *else không ứng với if thì không hợp lệ*
17	initialize	*khởi tạo*	The variable 'r' being used without being initialized: *Biến 'r' đang được sử dụng mà chưa khởi tạo (giá trị)*
18	local	*cục bộ*	**L**ocal function definitions are illegal: *Định nghĩa hàm cục bộ là không hợp lệ*
19	matching	*tương ứng*	**I**llegal else without matching if: *else không ứng với if thì không hợp lệ*

20	member	*thành viên, thành phần (của struct/class/union)*	'x' **i**s not a member of struct : *'x' không phải là thành phần của struct*
22	missing	*thiếu*	**M**issing ': ' before 'if': *Thiếu dấu ';' trước 'if'*
23	parameter	*tham số*	**C**annot convert parameter 1 from 'float' to 'int&': *Không thể chuyển đổi tham số thứ nhất từ kiểu 'float' sang kiểu '&int'*
24	**r**edefinition	*định nghĩa lại*	'int i' : **r**edefinition: *Định nghĩa lại biến i*
25	return	*trả về*	**…m**ust be return a value:…*phải trả về giá trị*
26	undeclared	*chưa khai báo*	**U**ndeclared identifier*: Định danh chưa khai báo*
27	unnhandled	*Không điều khiển/kiểm soát được*	Unhandled exception: *Ngoại lệ không điều khiển được (ngoài tầm kiểm soát)*
28	value	*giá trị*	**…m**ust be return a value:…*phải trả về giá trị*
29	variable	*biến*	The variable 'r' being used without being initialized: *Biến 'r' đang được sử dụng mà chưa khởi tạo (giá trị)*
30	violation	*sự vi phạm*	Access violation at address...:*Vi phạm truy cập tại địa chỉ…*