

**BỘ CÔNG THƯƠNG  
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THỰC PHẨM TP.HCM  
KHOA CÔNG NGHỆ THÔNG TIN**

**ĐH\***

**Nguyễn Thị Bích Ngân – Dương Thị Mộng Thùy**



**GIÁO TRÌNH**

**NHẬP MÔN LẬP TRÌNH**

*(lưu hành nội bộ)*

**THÀNH PHỐ HỒ CHÍ MINH – NĂM 2019**

**BỘ CÔNG THƯƠNG**  
**TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THỰC PHẨM TP.HCM**  
**KHOA CÔNG NGHỆ THÔNG TIN**

**ĐOÀN**

**Nguyễn Thị Bích Ngân – Dương Thị Mộng Thùy**

**GIÁO TRÌNH**

**NHẬP MÔN LẬP TRÌNH**

*(Tài liệu dùng cho hệ đại học và cao đẳng)*

**THÀNH PHỐ HỒ CHÍ MINH – NĂM 2019**

## LỜI NÓI ĐẦU

Ngôn ngữ lập trình C/C++ là ngôn ngữ lập trình cơ bản trong hệ thống các ngôn ngữ lập trình. “*Nhập môn lập trình*” là môn học cung cấp các kiến thức cơ bản nền tảng về lập trình thông qua ngôn ngữ lập trình C/C++. Đây là môn học cơ sở cho sinh viên chuyên ngành công nghệ thông tin, và nó là kiến thức thiết yếu cho người lập trình Ở trường Đại học Công nghiệp Thực phẩm TP.HCM, kiến thức nhập môn lập trình cũng đang được giảng dạy cho sinh viên đại học, cao đẳng, và học sinh trung cấp chuyên ngành công nghệ thông tin. “*Nhập môn lập trình*” còn là một trong những môn thi trong các kỳ thi liên thông từ trung cấp lên cao đẳng hoặc từ cao đẳng lên đại học.

Trong giáo trình này, chúng tôi sẽ trình bày các khái niệm, qui định và những kỹ thuật lập trình căn bản thể hiện qua ngôn ngữ lập trình C như sau:

- Chương 1: Tổng quan
- Chương 2: Kiểu dữ liệu và phép toán
- Chương 3: Các lệnh điều khiển
- Chương 4: Hàm
- Chương 5: Mảng và con trỏ
- Chương 6: File dữ liệu
- Chương 7: Kiểu dữ liệu cấu trúc

Các vấn đề được trình bày chi tiết với những ví dụ rõ ràng, mỗi ví dụ chương trình đều có kết quả thực thi kèm theo để minh họa kiểm chứng. Cuối mỗi chương có phần bài tập được sắp xếp từ cơ bản đến nâng cao giúp sinh viên nắm vững và kiểm tra kiến thức bằng việc giải các bài tập. Chúng tôi mong rằng các sinh viên tự tìm hiểu trước

mỗi vấn đề, kết hợp với bài giảng trên lớp của giảng viên và làm bài tập để việc học môn này đạt hiệu quả.

Trong quá trình giảng dạy và biên soạn giáo trình này, chúng tôi đã nhận được nhiều đóng góp quý báu của các đồng nghiệp ở Bộ môn Công nghệ Phần mềm cũng như các đồng nghiệp trong và ngoài Khoa Công nghệ Thông tin. Chúng tôi xin cảm ơn và hy vọng rằng giáo trình này sẽ giúp cho việc giảng dạy và học môn “Nhập môn lập trình” của sinh viên trường chúng ta ngày càng tốt hơn. Chúng tôi hy vọng nhận được nhiều ý kiến đóng góp để giáo trình ngày càng hoàn thiện.

***Nhóm tác giả***

## MỤC LỤC

LỜI NÓI ĐẦU .....	1
CHƯƠNG 1. TỔNG QUAN .....	7
1.1 Giới thiệu về ngôn ngữ lập trình C.....	7
1.2 Đặc điểm của ngôn ngữ lập trình C.....	8
1.3 Cấu trúc chương trình C.....	9
1.3.1 Các chỉ thị tiền xử lý .....	9
1.3.2 Định nghĩa kiểu dữ liệu.....	10
1.3.3 Khai báo các biến ngoài.....	10
1.3.4 Khai báo các prototype của hàm tự tạo.....	10
1.3.5 Hàm main.....	10
1.3.6 Định nghĩa các hàm tự tạo .....	11
1.4 Thư viện hàm chuẩn C .....	16
1.5 Ưu và nhược điểm .....	17
1.5.1 Ưu điểm .....	17
1.5.2 Nhược điểm .....	18
BÀI TẬP CHƯƠNG 1.....	19
CHƯƠNG 2. KIỂU DỮ LIỆU VÀ PHÉP TOÁN .....	20
2.1 Danh hiệu.....	20
2.1.1 Kí hiệu.....	20
2.1.2 Tên.....	20
2.1.3 Từ khóa.....	20
2.1.4 Chú thích.....	21
2.2 Biến .....	23
2.3 Các kiểu dữ liệu chuẩn.....	23
2.3.1 Kiểu char.....	23
2.3.2 Kiểu int.....	25
2.3.3 Kiểu float và double.....	26
2.3.4 Các kiểu dữ liệu bổ sung.....	27
2.4 Hằng số .....	30
2.5 Biểu thức.....	31
2.6 Các phép toán .....	31
2.6.1 Toán tử số học .....	31

2.6.2	Toán tử quan hệ.....	33
2.6.3	Toán tử logic.....	34
2.6.4	Toán tử trên bit .....	34
2.6.5	Toán tử tăng giảm .....	35
2.6.6	Toán tử gán .....	36
2.6.7	Biểu thức phẩy.....	37
2.6.8	Biểu thức điều kiện .....	37
2.6.9	Độ ưu tiên của toán tử.....	38
BÀI TẬP CHƯƠNG 2.....		39
CHƯƠNG 3. CÁC LỆNH ĐIỀU KHIỂN .....		40
3.1	Câu lệnh .....	40
3.1.1	Lệnh đơn .....	40
3.1.2	Lệnh phức .....	40
3.2	Lệnh điều kiện .....	41
3.2.1	Lệnh if.....	41
3.2.2	Lệnh switch case .....	45
3.3	Lệnh lặp.....	50
3.3.1	Lệnh for.....	50
3.3.2	Lệnh while .....	53
3.3.3	Lệnh do...while .....	55
BÀI TẬP CHƯƠNG 3.....		58
CHƯƠNG 4. HÀM .....		62
4.1	Khái niệm hàm (Function).....	62
4.2	Định nghĩa hàm .....	63
4.3	Thực thi hàm .....	65
4.4	Truyền tham số .....	67
4.5	Kết quả trả về.....	69
4.6	Prototype của hàm.....	70
4.7	Các hàm chuẩn.....	71
4.8	Thư viện hàm .....	71
4.9	Sự đệ quy .....	71
BÀI TẬP CHƯƠNG 4.....		75
CHƯƠNG 5. MẢNG VÀ CON TRỎ .....		78
5.1	Mảng một chiều.....	78

5.1.1	Khái niệm và khai báo mảng một chiều .....	78
5.1.2	Gán giá trị vào các phần tử của mảng.....	79
5.1.3	Lấy giá trị các phần tử trong mảng .....	80
5.1.4	Các phần tử của mảng trong bộ nhớ .....	81
5.1.5	Khởi tạo mảng .....	81
5.2	Mảng hai chiều .....	83
5.2.1	Khái niệm.....	83
5.2.2	Chỉ số của mảng .....	83
5.2.3	Truy xuất phần tử mảng hai chiều.....	84
5.2.4	Khởi tạo mảng hai chiều .....	84
5.3	Con trỏ (Pointer).....	85
5.3.1	Khái niệm.....	85
5.3.2	Khai báo biến con trỏ .....	85
5.3.3	Toán tử địa chỉ (&) và toán tử nội dung (*) .....	86
5.3.4	Tính toán trên Pointer .....	88
5.3.5	Truyền tham số địa chỉ.....	90
5.4	Cấp phát và giải phóng vùng nhớ cho biến con trỏ.....	92
5.4.1	Cấp phát vùng nhớ cho biến con trỏ .....	92
5.4.2	Giải phóng vùng nhớ cho biến con trỏ.....	93
5.5	Sự liên hệ giữa cách sử dụng mảng và pointer .....	94
5.5.1	Khai thác một pointer theo cách của mảng.....	94
5.5.2	Khai thác một mảng bằng pointer .....	95
5.5.3	Những điểm khác nhau giữa mảng và con trỏ.....	96
5.5.4	Hàm có đối số là mảng.....	97
5.5.5	Hàm trả về pointer và mảng .....	98
5.5.6	Mảng các con trỏ hoặc con trỏ của con trỏ.....	100
5.5.7	Pointer chỉ đến pointer .....	103
5.6	Chuỗi ký tự .....	103
5.6.1	Chuỗi ký tự .....	103
5.6.2	Một số hàm thao tác trên chuỗi.....	104
BÀI TẬP CHƯƠNG 5.....		108
CHƯƠNG 6. FILE DỮ LIỆU .....		111
6.1	Giới thiệu về file.....	111
6.1.1	Giới thiệu .....	111

6.1.2	Khái niệm File .....	111
6.1.3	Cách thao tác với file .....	112
6.1.4	Tổ chức lưu trữ dữ liệu trên file .....	112
6.2	Định nghĩa biến file và các thao tác mở, đóng file .....	113
6.2.1	Định nghĩa biến file trong C .....	114
6.2.2	Hàm mở, đóng file chuẩn .....	114
6.2.3	Thao tác nhập, xuất với file .....	118
<b>BÀI TẬP CHƯƠNG 6.....</b>		<b>125</b>
<b>CHƯƠNG 7. KIỂU DỮ LIỆU CẤU TRÚC .....</b>		<b>127</b>
7.1	Kiểu dữ liệu cấu trúc (struct) .....	127
7.1.1	Giới thiệu .....	127
7.1.2	Định nghĩa .....	127
7.1.3	Khai báo .....	130
7.1.4	Cấu trúc lồng nhau .....	131
7.1.5	Khởi tạo cấu trúc .....	131
7.1.6	Truy xuất các thành phần của một biến cấu trúc .....	132
7.2	Mảng các struct .....	133
7.3	Con trỏ kiểu cấu trúc .....	133
7.4	Cấu trúc đệ quy .....	134
<b>BÀI TẬP CHƯƠNG 7.....</b>		<b>135</b>
<b>BÀI TẬP LUYỆN TẬP VÀ NÂNG CAO .....</b>		<b>138</b>
<b>PHỤ LỤC A. MỘT SỐ HÀM CHUẨN TRONG C/C++ .....</b>		<b>145</b>
<b>PHỤ LỤC B. HƯỚNG DẪN TÌM LỖI VÀ SỬA LỖI CHƯƠNG TRÌNH (DEBUG) .....</b>		<b>153</b>
1.	Chạy chương trình .....	153
2.	Breakpoints .....	153
3.	Watch Windows .....	154
4.	Data Tip .....	154
5.	Locals .....	155
6.	Autos .....	155
7.	Watch .....	155
8.	Debug chương trình .....	156
<b>TÀI LIỆU THAM KHẢO .....</b>		<b>158</b>



## **CHƯƠNG 1. TỔNG QUAN**

### **1.1 Giới thiệu về ngôn ngữ lập trình C**

Ngôn ngữ lập trình C là ngôn ngữ lập trình cấp cao, được sử dụng rất phổ biến để lập trình hệ thống cùng với Assembler và phát triển các ứng dụng.

Vào những năm cuối thập kỷ 60 đầu thập kỷ 70 của thế kỷ XX, Dennis Ritchie (làm việc tại phòng thí nghiệm Bell) đã phát triển ngôn ngữ lập trình C dựa trên ngôn ngữ BCPL (do Martin Richards đưa ra vào năm 1967) và ngôn ngữ B (do Ken Thompson phát triển từ ngôn ngữ BCPL vào năm 1970 khi viết hệ điều hành UNIX đầu tiên trên máy PDP-7) và được cài đặt lần đầu tiên trên hệ điều hành UNIX của máy DEC PDP-11.

Năm 1978, Dennis Ritchie và B.W Kernighan đã cho xuất bản quyển “Ngôn ngữ lập trình C” và được phổ biến rộng rãi đến nay.

Lúc ban đầu, ngôn ngữ lập trình C được thiết kế nhằm lập trình trong môi trường của hệ điều hành Unix nhằm mục đích hỗ trợ cho các câu lệnh lập trình phức tạp. Nhưng về sau, với những nhu cầu phát triển ngày một tăng của câu lệnh lập trình, ngôn ngữ lập trình C đã vượt qua khuôn khổ của phòng thí nghiệm Bell và nhanh chóng hội nhập vào thế giới lập trình, các công ty lập trình sử dụng ngôn ngữ lập trình C một cách rộng rãi. Sau đó, các công ty sản xuất phần mềm lần lượt đưa ra các phiên bản hỗ trợ cho việc lập trình bằng ngôn ngữ lập trình C và chuẩn ANSI C ra đời.

Ngôn ngữ lập trình C là một ngôn ngữ lập trình hệ thống rất mạnh và rất “mềm dẻo”, có một thư viện gồm rất nhiều các hàm (function) đã được tạo sẵn. Người lập trình có thể tận dụng các hàm này để giải quyết các bài toán mà không cần phải tạo mới. Hơn thế nữa, ngôn ngữ lập trình C hỗ trợ rất nhiều phép toán nên phù hợp cho việc giải quyết các bài toán kỹ thuật có nhiều công thức phức tạp.

Ngoài ra, C cũng cho phép người lập trình tự định nghĩa thêm các kiểu dữ liệu trừu tượng mới. Tuy nhiên, điều mà người mới vừa học lập trình C thường gặp “rắc rối” là “hơi khó hiểu” do sự “mềm dẻo” của ngôn ngữ lập trình C. Dù vậy, ngôn ngữ lập trình C được phổ biến khá rộng rãi và đã trở thành một công cụ lập trình khá mạnh, được sử dụng như là một ngôn ngữ lập trình chủ yếu trong việc xây dựng những phần mềm hiện nay.

## 1.2 Đặc điểm của ngôn ngữ lập trình C

- *Tính cô đọng (compact)*: C chỉ có 32 từ khóa chuẩn và 40 toán tử chuẩn, nhưng hầu hết đều được biểu diễn bằng những chuỗi ký tự ngắn gọn.
- *Tính cấu trúc (structured)*: C có một tập hợp những chỉ thị của lập trình như cấu trúc lựa chọn, lặp... Từ đó các chương trình viết bằng C được tổ chức rõ ràng, dễ hiểu.
- *Tính tương thích (compatible)*: C có bộ tiền xử lý và một thư viện chuẩn vô cùng phong phú nên khi chuyển từ máy tính này sang máy tính khác các chương trình viết bằng C vẫn hoàn toàn tương thích.
- *Tính linh động (flexible)*: C là một ngôn ngữ rất uyển chuyển về mặt cú pháp, chấp nhận nhiều cách thể hiện, có thể thu gọn kích thước của các mã lệnh làm chương trình chạy nhanh hơn.
- *Biên dịch (compile)*: C cho phép biên dịch nhiều tập tin chương trình riêng rẽ thành các tập tin đối tượng (object) và liên kết (link) các đối tượng đó lại với nhau thành một chương trình có thể thực thi được (executable) thống nhất.

Ngôn ngữ lập trình C cũng là một công cụ để truy nhập vào bộ nhớ máy tính, truy cập các chức năng bên trong DOS và BIOS, lập trình điều khiển cho các linh kiện điện tử khác.

### 1.3 Cấu trúc chương trình C

Một chương trình C bao gồm các phần như: Các chỉ thị tiền xử lý, định nghĩa kiểu dữ liệu mới, khai báo biến ngoài, các hàm tự tạo, hàm main.

Cú pháp:

**Các chỉ thị tiền xử lý**  
**Định nghĩa kiểu dữ liệu**  
**Khai báo các biến ngoài**  
**Khai báo các prototype của hàm tự tạo**  
**Hàm main**  
**Định nghĩa các hàm tự tạo**

#### 1.3.1 Các chỉ thị tiền xử lý

Bước tiền xử lý giúp diễn giải các mã lệnh rất đặc biệt gọi là các chỉ thị dẫn hướng của bộ tiền xử lý (destination directive of preprocessor). Các chỉ thị này được nhận biết bởi chúng bắt đầu bằng ký hiệu #.

Có hai chỉ thị quan trọng:

- Chỉ thị gộp vào của các tập tin nguồn khác: `#include`
- Chỉ thị định nghĩa các ký hiệu: `#define`

Chỉ thị `#include` được sử dụng để gộp nội dung của các tập tin cần có, đặc biệt là các hàm trong tập tin thư viện chuẩn.

Cú pháp:

**`#include <Tên tập tin thư viện>`**

**Ví dụ 1.1:**

`#include <stdio.h>`

Chỉ thị `#define` được sử dụng trong việc định nghĩa các ký hiệu

Cú pháp:

**`#define <Tên kí hiệu><giá trị tương ứng>`**

**Ví dụ 1.2:**

```
#define NB_COUPS_MAX 100
#define SIZE 25
```

**1.3.2 Định nghĩa kiểu dữ liệu**

Bước định nghĩa kiểu dữ liệu dùng để đặt tên lại cho một kiểu dữ liệu nào đó nhằm gợi nhớ hay đặt một kiểu dữ liệu riêng dựa trên các kiểu dữ liệu đã có. Đây là phần không bắt buộc định nghĩa trong chương trình.

Cú pháp:

**typedef <Tên kiểu cũ><Tên kiểu mới>****Ví dụ 1.3:**

```
typedef int SoNguyen; //Kiểu SoNguyen là kiểu int
```

**1.3.3 Khai báo các biến ngoài**

Bước khai báo biến ngoài dùng để khai báo các biến toàn cục được sử dụng trong cả chương trình. Đây là phần không bắt buộc khai báo trong chương trình.

**1.3.4 Khai báo các prototype của hàm tự tạo**

Khai báo các prototype là khai báo tên hàm, các tham số, kiểu kết quả trả về,... của hàm tự tạo sẽ cài đặt phía sau, phần này chỉ là các khai báo đầu hàm, không phải là phần định nghĩa hàm. Đây là phần không bắt buộc khai báo trong chương trình.

**Ví dụ 1.4:**

```
boolean isPrime(int a); //prototype của hàm isPrime
```

**1.3.5 Hàm main**

Khi chương trình thực thi thì hàm main được gọi trước tiên. Đây là phần bắt buộc khai báo trong chương trình.

Cú pháp:

```
<Kiểu dữ liệu trả về> main()
{
    //Các khai báo cục bộ trong hàm main ]
    //Các câu lệnh dùng để định nghĩa hàm main]
    [return <kết quả trả về>; ]
}
```

**Ví dụ 1.5:**

```
void main()
{
    printf("Hello");
    getch();
}
```

### 1.3.6 Định nghĩa các hàm tự tạo

Đây là phần không bắt buộc định nghĩa trong chương trình.

Cú pháp:

```
<Kiểu dữ liệu trả về> function( các tham số)
{
    //Các khai báo cục bộ trong hàm]
    //Các câu lệnh dùng để định nghĩa hàm]
    [return <kết quả trả về>;]
}
```

**Ví dụ 1.6:**

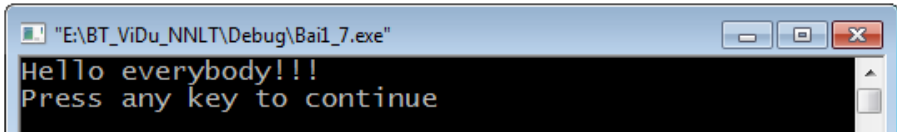
```
int tinh tong(int a, int b)
{
    int t = a+b;
    return t;
}
```

**Ví dụ 1.7:**

Chương trình sau sẽ hiển thị ra màn hình dòng chữ: Hello everybody!!!

```
#include "stdio.h"

void main( )
{
    printf("Hello everybody!!!") ;
}
```

**Kết quả thực thi chương trình**

Trong đó:

- **main:** hàm chính bắt buộc phải có trong ngôn ngữ lập trình C.
- **void:** hàm **main** không có giá trị trả về.
- **( ):** chương trình trên không có đối số nào, tức là không có giá trị truyền vào.
- Hai dấu “{“ và “}”: qui định thân chương trình, đóng vai trò báo hiệu điểm mở đầu và điểm kết thúc chương trình.
- **printf(“Hello everybody!!!”);** là lệnh hiển thị dòng chữ “Hello everybody!!!” ra màn hình.

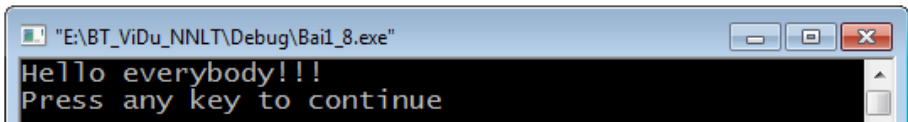
**Ví dụ 1.8:**

Chương trình hiển thị lên màn hình dòng chữ “Hello everybody!!!” có sử dụng hàm tự tạo.



```
#include "stdio.h"
void Hello();
void main()
{
    Hello();
}
void Hello()
{
    printf("Hello everybody!!!");
}
```

### Kết quả thực thi chương trình



Ở ví dụ 1.8, ta thấy cách gọi hàm trong ngôn ngữ lập trình C, hàm `main()` là hàm chính bắt buộc phải có trong mỗi chương trình. Hàm `Hello()` được hàm `main()` gọi đến để thực hiện. Cả ví dụ 1.7 và ví dụ 1.8 đều cùng thực hiện việc in ra câu: **Hello everybody!!!**. Nhưng ở đây cho thấy hai cách thể hiện của một câu lệnh trong ngôn ngữ lập trình C.

### 1.4 Thư viện hàm chuẩn C

Thư viện hàm chuẩn C là tập hợp các hàm đã được xây dựng trước. Mỗi thư viện hàm chứa các hàm theo một công dụng riêng. Tất cả trình biên dịch C đều chứa một thư viện *hàm chuẩn*. Một hàm được viết bởi lập trình viên có thể được đặt trong thư viện và được dùng khi cần thiết. Một số trình biên dịch cho phép thêm hàm vào thư viện chuẩn.



Một số thư viện chuẩn trong C:

- **stdio.h**: Tập tin định nghĩa các hàm vào/ra chuẩn (standard input/output). Gồm các hàm in dữ liệu (printf()), nhập giá trị cho biến (scanf()), nhận ký tự từ bàn phím (getc()), in ký tự ra màn hình (putc()), nhận một dãy ký tự từ bàn phím (gets()), in chuỗi ký tự ra màn hình (puts()), xóa vùng đệm bàn phím (fflush()), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw(),...
- **conio.h** : Tập tin định nghĩa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clrnl(),...
- **math.h**: Tập tin định nghĩa các hàm tính toán. Gồm các hàm abs(), sqrt(), log(), log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),...
- **alloc.h**: Tập tin định nghĩa các hàm liên quan đến việc quản lý bộ nhớ. Gồm các hàm calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(),...
- **io.h**: Tập tin định nghĩa các hàm vào ra cấp thấp. Gồm các hàm open(), \_open(), read(), \_read(), close(), \_close(), creat(), \_creat(), createnew(), eof(), filelength(), lock(),...
- **graphics.h**: Tập tin định nghĩa các hàm liên quan đến đồ họa. Gồm initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(),...

## 1.5 Ưu và nhược điểm

### 1.5.1 Ưu điểm

Ngôn ngữ lập trình C là một ngôn ngữ mạnh, mềm dẻo và có thể truy nhập vào hệ thống, nên thường được sử dụng để viết hệ điều hành, các trình điều khiển thiết bị, đồ họa, có thể xây dựng các phần mềm ngôn ngữ khác.

Ngôn ngữ lập trình C có cấu trúc module, từ đó ta có thể phân hoạch hay chia nhỏ chương trình để tăng tính hiệu quả, rõ ràng, dễ kiểm tra trong chương trình.

### **1.5.2 Nhược điểm**

Một số kí hiệu của ngôn ngữ lập trình C có nhiều ý nghĩa khác nhau. Ví dụ toán tử  $*$  là toán tử nhân, cũng là toán tử thay thế, hoặc dùng khai báo con trỏ. Việc sử dụng đúng ý nghĩa của các toán tử phụ thuộc vào ngữ cảnh sử dụng.

Vì ngôn ngữ lập trình C là một ngôn ngữ mềm dẻo, đó là do việc truy nhập tự do vào dữ liệu, trộn lẫn các dữ liệu,... Từ đó, dẫn đến sự lạm dụng và sự bất ổn của chương trình.

## **BÀI TẬP CHƯƠNG 1**

1. Viết chương trình xuất ra câu thông báo: “Chao ban den voi ngon ngu C”.
2. Viết chương trình xuất ra đoạn thông báo:  
    “Chao ban!  
    Day la chương trình C dau tien.  
    Vui long nhan phim Enter de ket thuc.”
3. Viết chương trình nhập vào 1 số nguyên, xuất ra màn hình số nguyên vừa nhập.
4. Viết chương trình nhập vào 2 số nguyên, tính và xuất kết quả tổng, hiệu, tích và thương của 2 số nguyên vừa nhập.
5. Viết chương trình xuất ra các số từ 1 đến 10.

## CHƯƠNG 2. KIỂU DỮ LIỆU VÀ PHÉP TOÁN

### 2.1 Danh hiệu

#### 2.1.1 Kí hiệu

Tập kí tự hợp lệ trong ngôn ngữ C bao gồm:

- 52 chữ cái : A,B,C, ... ,Z và a,b,c, ... ,z
- 10 chữ số : 0,1,2,3,4,5,6,7,8,9
- Các kí hiệu toán học: +, -, \*, /, =, <, >, (, )
- Kí tự gạch nối : \_ ( chú ý phân biệt với dấu trừ “ - ” )
- Các kí tự đặc biệt như : ., ; : [ ] { } ? ! \ & \ | # \$ " ' @ ^ ...
- Dấu cách (khoảng trắng) dùng để phân cách giữa các từ.

#### 2.1.2 Tên

Tên là một dãy các ký tự liên nhau bắt đầu bằng chữ cái hoặc ký tự gạch dưới theo sau là chữ cái, dấu gạch dưới, chữ số. Một tên không được chứa các kí tự đặc biệt như dấu cách, dấu chấm câu,...

#### Ví dụ 2.1:

Những tên hợp lệ: *x1, chieudai, hoc\_sinh, diem\_2, \_abc, \_x\_y\_2, \_123...*

Những tên không hợp lệ: *123, 1\_xyz, bien#, ma so sinh vien, ...*

#### 2.1.3 Từ khóa

Các từ sử dụng để dành riêng trong ngôn ngữ lập trình C gọi là từ khoá (keyword). Mỗi một từ khoá có một ý nghĩa riêng của nó. Các từ khóa không được sử dụng làm các biến, hằng, không được được định nghĩa lại các từ khoá. Bảng liệt kê các từ khoá:

auto	break	case	char	continue
default	do	double	else	extern
float	for	goto	if	int
long	register	return	short	sizeof
static	struct	switch	typedef	union
unsigned	void	while	_cs	_ds

<code>_es</code>	<code>_ss</code>	<code>_AH</code>	<code>_AL</code>	<code>_AX</code>
<code>_BH</code>	<code>_BL</code>	<code>_BX</code>	<code>_CH</code>	<code>_CL</code>
<code>_CX</code>	<code>_DH</code>	<code>_DL</code>	<code>_DX</code>	<code>_BP</code>
<code>_DI</code>	<code>_SI</code>	<code>_SP</code>		

### 2.1.4 Chú thích

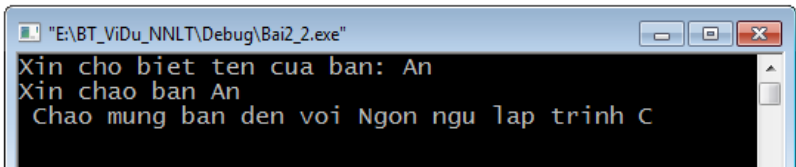
Chú thích là những dòng diễn tả ý nghĩa câu lệnh đang dùng, giải thích ý nghĩa của một hàm nào đó giúp người lập trình dễ nhớ, dễ hình dung được câu lệnh đang làm. Trình biên dịch không biên dịch các phần ghi chú trong chương trình.

Khi viết chương trình, thường xuyên ta cần phải có lời ghi chú về một đoạn chương trình nào đó để dễ nhớ và dễ điều chỉnh sau này. Phần nội dung ghi chú này khi biên dịch sẽ được bỏ qua. Trong ngôn ngữ lập trình C, nội dung chú thích phải được viết trong cặp dấu `/*` và `*/`.

#### Ví dụ 2.2:

```
#include <stdio.h>
#include<conio.h>
int main ()
{ /*khai bao bien ten kieu char
    co 50 ky tu */
    char ten[50];
/*Xuat chuoai ra man hinh*/
    printf("Xin cho biet ten cua ban: ");
/*Doc vao 1 chuoai la ten cua ban*/
    scanf("%s",ten);
    printf("Xin chao ban %s\n ",ten);
    printf("Chao mung ban den voi NNLT C");
/*Dung chuong trinh, cho go phim*/
    getch();
    return 0;
}
```

**Kết quả thực thi của chương trình:**



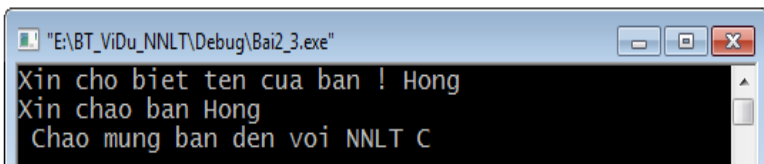
Ngoài ra, nếu chú thích chỉ nằm trên một dòng ta có thể sử dụng kí hiệu //.

### Ví dụ 2.3:

```
#include <stdio.h>
#include<conio.h>
int main ()
{
    char ten[50];//khai báo biến kiểu char 50 ký
tự
//Xuat chuoai ra man hinh
    printf("Xin cho biet ten cua ban !");
    scanf("%s",ten); //Doc vao 1 chuoai ten cua
ban.

    printf("Xin chao ban %s\n ",ten);
    printf("Chao mung ban den voi NNLT C");
//Dung chuong trinh, cho go phim
    getch();
    return 0;
}
```

**Kết quả thực thi chương trình:**



## 2.2 Biến

Biến là một khái niệm đại diện cho một giá trị dữ liệu cần lưu trữ tạm thời để tái sử dụng trong các câu lệnh phía sau trong khoảng thời gian chương trình thực thi. Sau khi kết thúc chương trình, biến này sẽ bị hủy. Giá trị này có thể bị thay đổi khi chương trình thực thi. Khi biến được tạo sẽ xuất hiện một vùng nhớ để lưu trữ giá trị của biến.

Một biến có một tên có ý nghĩa đại diện cho một vị trí vùng nhớ. Tên biến giúp chúng ta truy cập vào vùng nhớ mà không cần dùng địa chỉ của vùng nhớ đó.

Hệ điều hành đảm nhiệm việc cấp bộ nhớ còn trống cho những biến này mỗi khi người dùng cần sử dụng. Để tham chiếu đến một giá trị cụ thể trong bộ nhớ, chúng ta chỉ cần dùng tên của biến.

Cú pháp:

<b>&lt;tên kiểu dữ liệu&gt;&lt;tên biến&gt; [=&lt;giá trị&gt;]</b>
--

## 2.3 Các kiểu dữ liệu chuẩn

### 2.3.1 Kiểu char

Trong ngôn ngữ lập trình C chúng ta có thể xử lý dữ liệu là các chữ viết (kí tự). Các kí tự này là các chữ viết thường dùng như các chữ cái A,B,C....Z, a,b,c,... z; các chữ số 0,1,2,...9; các dấu chấm câu ; , ! ...

Kiểu kí tự được biểu diễn trong ngôn ngữ lập trình C với từ khóa `char`. Kiểu `char` có chiều dài là 1 byte dùng để lưu giữ một kí tự và có miền giá trị (mã số `ascii`) trong khoảng -128...+127.

Cú pháp khai báo biến kiểu `char`:

<b><code>char &lt;tênbiến&gt;;</code></b>
---

**Ví dụ 2.4:**

**`char ch ; // khai báo ch là kiểu kí tự`**

Một hằng kí tự được biểu diễn bằng chữ viết nằm giữa hai dấu phẩy ‘ ’.

**Ví dụ 2.5:**

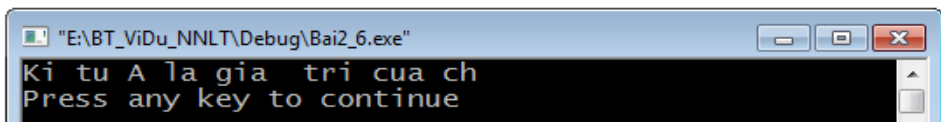
**'a', 'A', 'z', '\*', '!', '5'...**

Muốn truy xuất giá trị của một biến kí tự ta dùng kí hiệu đại diện %c. Khi đó, giá trị của biến được gọi sẽ hiển thị tại kí tự %c.

**Ví dụ 2.6:**

Chương trình xuất ra màn hình kí tự của ch, với biến ch được khởi tạo trước.

```
#include<stdio.h>
void main ()
{
    char ch ; /* khai báo biến ch có kiểu char */
    ch = 'A' ; /* gán giá trị 'A' cho biến ch */
    /* xuất chuỗi kèm theo giá trị biến ch*/
    printf("Kí tu %c la gia tri cua ch",ch) ;
}
```

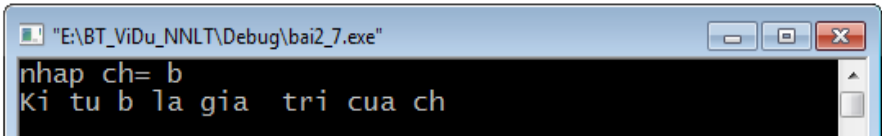
**Kết quả thực thi chương trình****Ví dụ 2.7:**

Chương trình nhận một kí tự từ bàn phím và sau đó hiển thị kí tự đó ra màn hình:

```
#include<stdio.h>
void main( )
{
    char ch;
    printf("nhap ch=");
    scanf("%c",&ch); //đọc kí tự ch từ bàn phím
    printf("Kí tu %c la gia tri cua ch",ch);
}
```



Kết quả thực thi chương trình



Bảng kí tự của bảng mã ASCII

Kí tự	Dãy mã	Giá trị trong bảng mã ASCII		Ý nghĩa
		Hexa-Decimal	Decimal	
BEL	\a	0x07	7	Tiếng chuông
BS	\b	0x08	8	Xóa trái (backspace)
HT	\t	0x09	9	Nhảy cách ngang (tab)
VT	\v	0x0B	11	Nhảy cách đứng
LF	\n	0x0A	10	Xuống dòng mới (newline)
FF	\f	0x0C	12	Xuống dòng dưới(form feed)
CR	\r	0x0D	13	Về đầu dòng(carriage return)
“	\”	0x22	34	Dấu “
‘	\’	0x27	39	Dấu ‘
?	\?	0x3F	63	Dấu ?
\	\\	0x5C	92	Dấu \
NULL	\0	0x00	00	Mã NULL

Bảng 2. 1. Một số kí tự đặc biệt của bảng mã ASCII

2.3.2 Kiểu int

Trong ngôn ngữ lập trình C, có nhiều loại kiểu số nguyên với các miền giới hạn khác nhau. Kiểu số nguyên cơ bản nhất được định nghĩa với từ khoá `int`. Tuy nhiên trên máy tính chỉ biểu diễn được một phần nhỏ của tập hợp các số nguyên. Mỗi biến kiểu `int` chiếm 2 bytes trong bộ nhớ, miền giá trị của kiểu `int` từ  $-2^{15}$  đến  $2^{15}-1$ .

Cú pháp khai báo biến kiểu số nguyên `int`:

**`int <tênbiến> ;`**

**Ví dụ 2.8:**

```
int N; // khai báo biến N là một số kiểu int
```

Khi truy xuất giá trị của một biến kiểu số nguyên ta dùng kí hiệu đại diện `%d`.

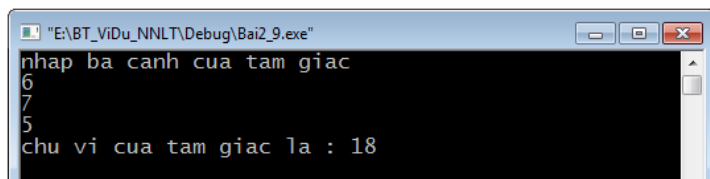
**Ví dụ 2.9:**

Chương trình nhận giá trị của ba cạnh tam giác, sau đó xuất ra chu vi của tam giác đó

```
#include<stdio.h>

void main()
{
    int a,b,c ; // ba cạnh của một tam giác
    int cv ; // chu vi của tam giác
    printf("nhap ba canh cua tam giac ");
    // Nhập vào ba cạnh của tam giác
    scanf("%d%d%d",&a,&b,&c);
    cv = a + b + c ; // tính chu vi của tam giác
    printf("chu vi cua tam giac la : %d", cv);
}
```

**Kết quả thực thi chương trình**



### 2.3.3 Kiểu float và double

Một số thực kiểu `float` được biểu diễn bằng 4 bytes, độ chính xác khoảng 6 chữ số, dãy giá trị trong khoảng  $1.2\text{E}-38 \div 3.4\text{E} + 38$ .

Một số thực kiểu `double` được biểu diễn bằng 8 bytes, độ chính xác khoảng 15 chữ số, dãy giá trị trong khoảng  $2.2\text{E}-308 \div 1.8\text{E} + 308$ .

Một số thực được khai báo như sau:

```
float    x; //khai báo số thực kiểu float
double  y;  //khai báo số thực kiểu double
```

### **Ví dụ 2.10 :**

Xuất ra một số thực có phần thập phân 6 chữ số như sau :

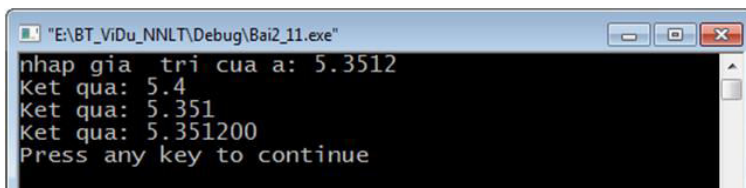
```
// khai báo và khởi tạo biến x = 123.4567
float x = 123.4567;
printf("giá trị của x là %f ", x);
```

Nếu cách hiển thị một số thực là `%.nf` khi đó giá trị số hiển thị n kí tự cho phần thập phân. Ví dụ như `%.5f` thì 5 kí tự cho phần thập phân của số hiển thị.

### **Ví dụ 2.11:**

```
#include<stdio.h>
void main()
{
    float a;
    printf("nhap gia tri cua a: ");
    scanf("%f", &a);
    printf("Ket qua: %.1f\n", a);
    printf("Ket qua: %.3f\n", a);
    printf("Ket qua: %.6f\n", a);
}
```

### **Kết quả thực thi của chương trình:**



## **2.3.4 Các kiểu dữ liệu bổ sung**

Các kiểu dữ liệu bổ sung bao gồm unsigned, signed, short, long

Các kiểu dữ liệu bổ sung kết hợp với các dữ liệu cơ sở làm thay đổi miền giá trị của chúng.

– **Kết hợp kiểu dữ liệu bổ sung và kiểu dữ liệu chuẩn**

Ta có thể tóm tắt các kiểu chuẩn và kiểu kết hợp qua bảng sau:

Kiểu	Chiều dài	Miền giá trị	Ý nghĩa
unsigned char	8 bits	0...255	Kiểu char không dấu
char	8 bits	-128...127	Kiểu char
enum	16 bits	-32768 ... 32767	Kiểu enum
unsigned int	16 bits	0 ... 65 535	Kiểu số nguyên không dấu
short int	16 bits	-32768 ... 32767	Kiểu short int
int	16 bits	-32768 ... 32767	Kiểu số nguyên (int)
unsigned long	32 bits	0 ... 4 294 483 647	Kiểu số nguyên (long) không dấu
long	32 bits	-2 147 483 648 ... 2 147 483 648	Kiểu số nguyên (long)
float	32 bits	$\pm 10^{-38} \dots 3.4 \cdot 10^{38}$	Kiểu số thực (float)
double	64 bits	$2.2 \cdot 10^{-308} \dots 1.8 \cdot 10^{308}$	Kiểu số thực có độ chính xác gấp đôi
long double	80 bits	$3.4 \cdot 10^{-4932} \dots 3.4 \cdot 10^{4932}$	Kiểu số thực có độ chính xác cao

Bảng 2. 2. Các kiểu dữ liệu chuẩn và kiểu dữ liệu kết hợp.

- **Mã quy cách định dạng:** sau đây là cách định dạng cho mỗi kiểu dữ liệu khác nhau đối với hàm printf()

Mã quy cách	Ý nghĩa
%c	In ra kí tự kiểu char, có thể dùng cho kiểu short và int
%d	Int ra kiểu số nguyên int, có thể dùng cho kiểu char
%u	In ra kiểu số nguyên không dấu, unsigned int, có thể dùng cho kiểu unsigned char, unsigned short
%ld	In ra kiểu long
%lu	In ra kiểu unsigned long
%x , %X	In ra kiểu số viết dưới dạng Hexa ứng với chữ thường và chữ hoa
%o	In ra kiểu số nguyên viết dưới dạng octal ( hệ đếm 8)
%f	In ra kiểu số thực dưới dạng bình thường với phần thập phân có 6 chữ số, dùng cho kiểu float, double
%e , %E	In ra kiểu số thực dưới dạng số mũ, với phần định trị có 6 chữ số thập phân, dùng cho kiểu float, double
%g , %G	In ra kiểu %f hoặc %e tùy thuộc kiểu dữ liệu nào ngắn hơn
%s	In ra chuỗi kí tự . Ta phải cung cấp địa chỉ của chuỗi kí tự

Bảng 2. 3. Mã quy cách định dạng cho các kiểu dữ liệu chuẩn.

- **Các mã quy cách dùng cho hàm scanf():**

Mã quy cách	Ý nghĩa
%c	Đọc một kí tự được khai báo là char
%d	Đọc một số nguyên kiểu int
%u	Đọc số nguyên unsigned int

<code>%hd</code>	Đọc số nguyên kiểu short int
<code>%hu</code>	Đọc số nguyên kiểu unsigned int
<code>%ld</code>	Đọc số nguyên kiểu long int
<code>%lu</code>	Đọc số nguyên kiểu unsigned long
<code>%f</code>	Đọc số thực float, có thể nhập theo kiểu viết thông thường hoặc viết theo dạng số mũ
<code>%e</code>	Giống <code>%f</code>
<code>%lf</code> hoặc <code>%lu</code>	Đọc số thực kiểu double
<code>%s</code>	Đọc xâu kí tự không chứa dấu cách, dùng với địa chỉ xâu
<code>%o</code>	Đọc vào số nguyên dưới cơ số 8 (octal)
<code>%x</code>	Đọc vào số nguyên dưới dạng hexa

Bảng 2. 4. Mã quy cách dùng cho hàm *scanf*.

## 2.4 Hằng số

Hằng số là một đại lượng có giá trị cố định trong chương trình.

Muốn sử dụng hằng, ta cũng phải khai báo trước với từ khóa `const`.

Cú pháp :

**`const <Tên kiểu dữ liệu><Tên hằng> = <giá trị hằng số>;`**

**Ví dụ 2.12:**

```
const int a= 32767 ;
const float a = 3.14;
const int a= 3, b = 4, C = 5;
```

Chúng ta có thể định nghĩa hằng số theo một kiểu khác với từ khóa `#define`

Cú pháp:

**`#define <Tên hằng số> <giá trị hằng số>`**

**Ví dụ 2.13:**

```
#define PI 3.14  
#define E 9.1083e-31
```

**Lưu ý:** khi định nghĩa hằng bằng `#define` thì không có dấu `=` và không có dấu chấm phẩy để kết thúc dòng định nghĩa, vì `#define` không phải là một lệnh.

**2.5 Biểu thức**

Biểu thức là một công thức tính toán để có một giá trị theo đúng quy tắc toán học nào đó. Một biểu thức (expression) bao gồm: toán tử (operator) và toán hạng (operand). Toán tử là phép toán, toán hạng có thể là hằng, là hàm, là biến. Các phần của biểu thức có thể phân thành các số hạng, thừa số, biểu thức đơn giản.

**Ví dụ 2.14:**

```
9 + 2 * PI * COS (x)
```

Trong ví dụ trên, các toán tử là các phép toán cộng (+), phép nhân (\*). Các toán hạng ở đây là các hằng số 9, 2, PI và hàm COS(x).

Các loại biểu thức:

- **Biểu thức số học:** là biểu thức tính ra kết quả là giá trị bằng số (số nguyên, số thực).
- **Biểu thức logic:** là biểu thức mà kết quả là đúng hoặc sai
- **Biểu thức quan hệ:** Một biểu thức chứa các toán tử quan hệ như `<`, `>`, `<=`, `>=`, `==`, `!=` được gọi là biểu thức Boolean đơn giản hay một biểu thức quan hệ. Các toán hạng trong biểu thức quan hệ có thể là các số nguyên, kí tự và chúng không nhất thiết phải tương thích với nhau về kiểu.

**2.6 Các phép toán****2.6.1 Toán tử số học**

Các toán tử số học thông thường là:

- Cộng : +
- Trừ: -

- Nhân: \*
- Chia: /
- Phép chia lấy phần dư của số nguyên : %

**Chú ý:**

- Phép toán % không dùng cho kiểu dữ liệu float hay double.
- Phép chia(/) thực hiện theo kiểu của các toán hạng dù là phép chia số nguyên hay số thực.

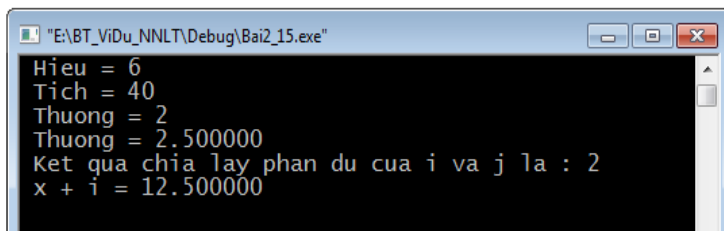
Với  $i, j$  là 2 biến kiểu số nguyên, có sự khác nhau giữa  $i/j$  và  $(float)i/j$ . Theo cách viết  $(float)i/j$  thì kết quả sẽ là một số thực, còn  $i/j$  thì kết quả là một số nguyên.

**Ví dụ 2.15:**

Chương trình sau minh họa cho các phép toán số học:

```
#include<stdio.h>
void main()
{
    int i = 10, j = 4, s, p, r;
    float x;
    s = i + j ;
    printf("\n Tong = %d",s);
    printf("\n Hieu = %d",i-j);
    p = i*j ;
    printf("\nTich = %d",p);
    printf("\nThuong = %d",i/j);
    x=(float)i/j;
    printf("\n Thuong = %f",x);
    r = i % j ;
    printf("\n Phan dư la : %d",r);
    printf("\n x + i = %f",x + i );
}
```



**Kết quả thực thi của chương trình trên:**

```
"E:\BT_ViDu_NNLT\Debug\Bai2_15.exe"
Hieu = 6
Tich = 40
Thuong = 2
Thuong = 2.500000
Ket qua chia lay phan du cua i va j la : 2
x + i = 12.500000
```

**2.6.2 Toán tử quan hệ**

Các toán tử quan hệ bao gồm:

`!=` : so sánh sự khác nhau

`==`: so sánh bằng nhau

`>=`: so sánh lớn hơn hoặc bằng

`<=`: so sánh nhỏ hơn hoặc bằng

`>`: so sánh lớn hơn

`<`: so sánh nhỏ hơn

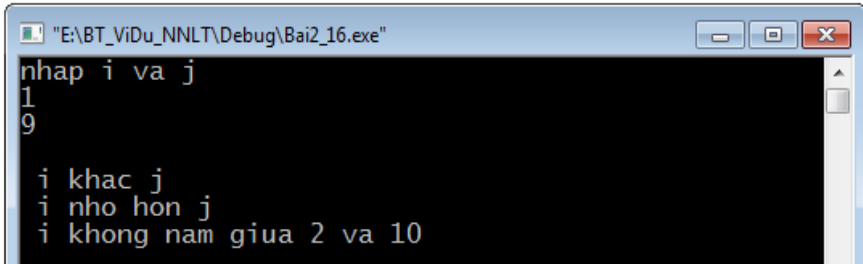
**Ví dụ 2.16:**

Chương trình sau sẽ minh họa cách sử dụng các toán tử quan hệ:

```
#include<stdio.h>

void main()
{
    int i, j;
    printf("nhap i va j");
    scanf("%d%d",&i,&j);
    if(i == j ) printf("i bang j");
    if(i != j ) printf("i khac j");
    if(i > j ) printf("i lon hon j");
    if(i < j ) printf("i nho hon j");
    if( i >= 2 && i<=10)
        printf("i nam giua 2 va 10");
    if(i < 2 || i > 10)
        printf("i khong nam giua 2 va 10");
}
```

### Kết quả thực thi chương trình



```
"E:\BT_ViDu_NNLT\Debug\Bai2_16.exe"
nhap i va j
1
9

i khác j
i nhỏ hơn j
i không nằm giữa 2 và 10
```

#### 2.6.3 Toán tử logic

Các phép toán logic gồm:

- && : phép AND logic
- || : phép OR logic
- ! : phép NOT logic

I. Trong đó:

- Phép && chỉ cho kết quả là đúng chỉ khi hai toán hạng đều đúng.
- Phép || chỉ cho kết quả là sai khi hai toán hạng đều sai.
- Phép ! phủ định lại toán hạng.

#### 2.6.4 Toán tử trên bit

Các toán tử trên bit cho phép xử lý các tín hiệu ở mức bit.

Các phép toán này không được dùng cho kiểu float và double.

Các toán tử này bao gồm:

- & : phép AND ở mức nhị phân.
- | : phép OR ở mức nhị phân.
- ^ : phép XOR ở mức nhị phân.
- << : Phép dịch trái(Shift left).
- >> : Phép dịch phải(Shift right).
- ~ : Phép đảo bit.

Cách thực hiện các phép toán trên bit:

$1 \& 1 = 1$	$1 \mid 1 = 1$	$1 \wedge 1 = 0$
$1 \& 0 = 0$	$1 \mid 0 = 1$	$1 \wedge 0 = 1$
$0 \& 1 = 0$	$0 \mid 1 = 1$	$0 \wedge 1 = 1$
$0 \& 0 = 0$	$0 \mid 0 = 0$	$0 \wedge 0 = 0$

Các phép dịch trái “<<”, dịch phải “>>” gây nên sự lệch trái hay lệch phải nội dung của một biến.

- $x \ll M$  nghĩa là dịch sang trái số nguyên  $x$  đi  $M$  bit, tương đương với  $x * 2^M$ .
- $x \gg M$  nghĩa là dịch sang phải số nguyên  $x$  đi  $M$  bit, tương đương với phép chia  $x / 2^M$  (chia lấy phần nguyên).

**Ví dụ 2.17:**

Ta có thể thay phép tính  $x * 80$  bằng cách viết:

$$x \ll 6 + x \ll 2 \text{ vì } 80 = 2^6 + 2^4$$

### 2.6.5 Toán tử tăng giảm

Trong ngôn ngữ lập trình C, phép tăng giảm 1 có thể viết gọn lại như sau:

- $i = i + 1$  có thể được viết thành :  $i++$  (tăng sau) hoặc  $++i$  (tăng trước).
- $i = i - 1$  có thể được viết thành :  $i--$  (giảm sau) hoặc  $--i$  (giảm trước).

Phép  $++i$  thì đầu tiên biến  $i$  được tăng 1, sau đó thực hiện phép gán. Còn phép  $i++$  thì phép gán được thực hiện trước, phép tăng 1 sẽ được thực hiện sau.

**Ví dụ 2.18:**

Với  $i = 3 ; j = 15;$

a/ $i = ++j ; i = j$	kết quả	$i = 16, j = 16$
b/ $i = j++$	kết quả	$i = 15, j = 16$
c/ $j = ++i + 5$	kết quả	$i = 4, j = 9$
d/ $j = i++ + 5$	kết quả	$j = 8, i = 4$

## 2.6.6 Toán tử gán

### - Phép gán đơn giản

Cú pháp:

**Tên\_một\_biến = biểu\_thức;**

**Ví dụ 2.19:**

```
i = 3 ; /* i được gán giá trị là 3 */
/* i cộng với 4 được 7, gán 7 vào i */
i = i + 4 ;
```

Điều này có nghĩa là giá trị của biểu thức bên phải dấu gán = sẽ được đặt vào ô nhớ của biến nằm bên trái dấu gán.

### - Phép gán kép

**Ví dụ 2.20:**

```
/* Gán giá trị 5 cho ba biến a, b, c */
a = b = c = 5 ;
/* Gán giá trị 5 cho biến c sau đó lấy giá trị c
cộng với giá trị b, và gán giá trị cho a */
a = b + ( c = 5 ) ;
```

### - Các phép gán mở rộng

Trong ngôn ngữ lập trình C, phép gán mở rộng được quy định như sau:

```
x += y ⇔ x = x + y
x -= y ⇔ x = x - y
x *= y ⇔ x = x * y
x /= y ⇔ x = x / y
x %= y ⇔ x = x % y
x >>= y ⇔ x = x >> y
x <<= y ⇔ x = x << y
x &= y ⇔ x = x & y
x |= y ⇔ x = x | y
x ^= y ⇔ x = x ^ y
```

### 2.6.7 Biểu thức phẩy

Mỗi câu lệnh trong ngôn ngữ lập trình C được kết thúc bằng dấu chấm phẩy, tuy nhiên trong một biểu thức của ngôn ngữ lập trình C có thể gồm nhiều câu lệnh được cách nhau bởi dấu phẩy.

#### Ví dụ 2.21:

```
x = a*b, q = x + y, k = q / z;
int n,m;
char ho[50], ten[50];
```

### 2.6.8 Biểu thức điều kiện

Cú pháp:

<Tên biến> = <Biểu thức điều kiện> ? <biểu thức 1> : <biểu thức 2>

Trong ngôn ngữ lập trình C, toán tử điều kiện (toán tử chấm hỏi “ ? ”) để so sánh giá trị đúng sai và cho phép có sự chọn lựa thích hợp.

#### Ví dụ 2.22:

```
m = a > b ? a : b /* m = max(a, b) */
```

Đầu tiên, biểu thức điều kiện  $a > b$  được kiểm tra. Nếu biểu thức này có giá trị đúng (true), giá trị của biến  $a$  sẽ được gán cho biến  $m$ , ngược lại, nếu biểu thức điều kiện  $a > b$  là sai (false) thì giá trị biến  $b$  sẽ được cho biến  $m$ .

#### Ví dụ 2.23:

```
printf("You have %d item%s.\n", n, n==1?"": "s");
```

Nếu  $n$  có giá trị là 1, thì câu “You have 1 item” sẽ hiển thị trên màn hình. Ngược lại, nếu  $n$  có giá trị lớn hơn 1, thì câu “You have 5 items” sẽ hiển thị.

Một cách tổng quát, toán tử điều kiện thực hiện các câu lệnh sau: đầu tiên tính biểu thức điều kiện (đứng trước dấu “?”). Nếu giá trị này bằng 1 thì máy sẽ dùng biểu thức thứ nhất (đứng trước dấu “:”), ngược lại nếu giá trị này bằng 0 thì máy sẽ dùng biểu thức thứ hai (đứng sau dấu “:”).

### 2.6.9 Độ ưu tiên của toán tử

Ta có thể minh họa độ ưu tiên của toán tử qua một bảng tổng kết sau, với độ ưu tiên được tính từ trên xuống dưới:

Toán tử	Độ ưu tiên
() [] ->	Ưu tiên từ trái sang phải
- ++ ! ~ sizeof()	Ưu tiên từ phải sang trái
* / %	Ưu tiên từ trái sang phải
+ -	Ưu tiên từ trái sang phải
<<>>	Ưu tiên từ trái sang phải
<< = >> =	Ưu tiên từ trái sang phải
== !=	Ưu tiên từ trái sang phải
&	Ưu tiên từ trái sang phải
^	Ưu tiên từ trái sang phải
	Ưu tiên từ trái sang phải
&&	Ưu tiên từ trái sang phải
	Ưu tiên từ trái sang phải
? :	Ưu tiên từ phải sang trái
= += -= *= /= %= ^=  = << = >>	Ưu tiên từ phải sang trái
,	Ưu tiên từ trái sang phải

Bảng 2. 5. Độ ưu tiên của phép toán.

## BÀI TẬP CHƯƠNG 2

1. Viết chương trình nhập vào 2 số thực. Tính và xuất kết quả tổng, hiệu, tích, thương của 2 số thực vừa nhập, kết quả lấy 2 số lẻ.
2. Viết chương trình đổi nhiệt độ từ đơn vị F (Ferarit) ra độ C (Celsius) theo công thức:  $C = 5/9 (F - 32)$
3. Viết chương trình tính giá trị  $F(x)$  và  $G(x)$ , trong đó  $x$  là số nguyên nhập từ phím:

$$F(x) = 5x^2 + 6x + 1 \qquad G(x) = 2x^4 - 5x^2 + 4x + 1$$

4. Viết chương trình tính giá trị của biểu thức, trong đó  $x$  là số nguyên nhập từ phím:

$$F(x) = \frac{1+x}{1-x} \qquad G(x) = \frac{1+5x-7x^2}{2+3x^3}$$

5. Viết chương trình tính giá trị của biểu thức, trong đó  $a, b, c$  là số nguyên nhập từ phím.

$$F(x) = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \qquad G(x) = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$$

6. Viết chương trình tính giá trị của biểu thức:

$$f(x) = \frac{3x^2 + 4x + 5}{2x + 1} \qquad g(x) = \frac{3x^5 + 2x + \sqrt{x+1}}{5x^2 - 3}$$

7. Viết chương trình nhập vào chiều dài, chiều rộng của 1 hình chữ nhật, hãy tính và xuất kết quả chu vi, diện tích của HCN trên.
8. Viết chương trình nhập vào bán kính của 1 hình tròn. Tính và xuất kết quả chu vi, diện tích của hình tròn trên.
9. Viết chương trình nhập vào chiều dài cạnh 1 hình vuông. Tính và xuất kết quả chu vi, diện tích, đường chéo của hình vuông trên.
10. Nhập vào 2 số nguyên  $a, b$ . Tìm số lớn nhất trong 2 số.
11. Nhập 3 số nguyên  $a, b, c$ . Xuất số lớn nhất và số nhỏ nhất của 3 số đó.

Gợi ý: Sinh viên dùng biểu thức điều kiện.

## CHƯƠNG 3. CÁC LỆNH ĐIỀU KHIỂN

### 3.1 Câu lệnh

#### 3.1.1 Lệnh đơn

Một câu lệnh đơn là một câu lệnh không chứa các câu lệnh khác bên trong nó và kết thúc bằng một dấu chấm phẩy (;)

##### **Ví dụ 3.1:**

```
int x=5; //một câu lệnh đơn
x++; //một câu lệnh đơn
printf("Giá trị x là: %d",x); //một câu lệnh đơn
```

#### 3.1.2 Lệnh phức

Một câu lệnh phức là một câu lệnh chứa câu lệnh khác bên trong nó hoặc một khối lệnh gồm nhiều câu lệnh như lệnh điều kiện (lệnh if), lệnh rẽ nhánh (lệnh switch), lệnh lặp (các vòng lặp for, while, do...while).

Một dãy các khai báo cùng với các câu lệnh nằm trong cặp dấu ngoặc móc { và } được gọi là một khối lệnh.

##### **Ví dụ 3.3:**

```
{
    char ten[30];
    printf("\n Nhập vào ten của bạn:");
    scanf("%s", ten);
    printf("\n Chao Ban %s",ten);
}
```

##### **Chú ý:**

- Nếu một biến được khai báo bên ngoài khối lệnh và không trùng tên với biến bên trong khối lệnh thì nó cũng được sử dụng bên trong khối lệnh.
- Một khối lệnh bên trong có thể sử dụng các biến bên ngoài, các lệnh bên ngoài không thể sử dụng các biến bên trong khối lệnh con.



**Ví dụ 3.4:**

```

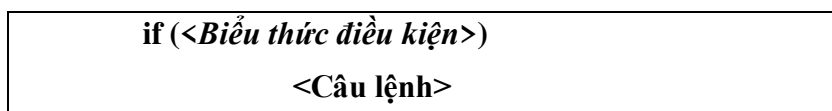
int a=0; /*biến a trong khối lệnh bên ngoài*/
for(int i=0; i<7; i++)
{
    int t = 5;
    /*biến a bên ngoài khối lệnh*/
    a= a+i;
}
printf("gia tri cua a la: %d",a);
printf("gia tri cua t la: %d",t);
/* báo lỗi! không sử dụng được */

```

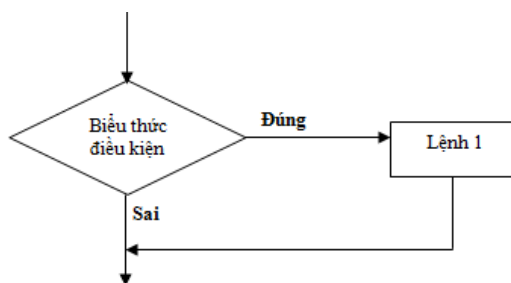
**3.2 Lệnh điều kiện****3.2.1 Lệnh if**

Câu lệnh if cho phép chúng ta thay đổi luồng thực thi của câu lệnh trong chương trình dựa vào điều kiện, một câu lệnh hoặc một khối các câu lệnh sẽ được quyết định thực thi hay không được thực thi.

- **Lệnh if** : Cú pháp:



Lưu đồ:



Hình 3-1. Lưu đồ thực hiện của câu lệnh if

Giải thích:

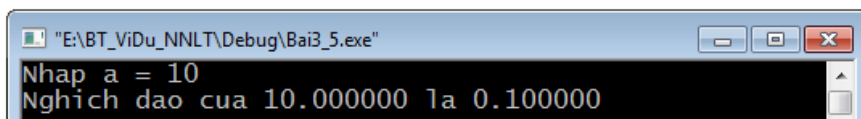
- + <Lệnh 1> có thể là một câu lệnh đơn, một khối lệnh hay một câu lệnh phức.
- + Kiểm tra *Biểu thức điều kiện* trước.
- + Nếu điều kiện đúng (bằng 1) thì thực hiện *Lệnh 1* theo sau biểu thức điều kiện.
- + Nếu điều kiện sai (bằng 0) thì bỏ qua *Lệnh 1* (những lệnh và khối lệnh sau đó vẫn được thực hiện bình thường vì nó không phụ thuộc vào điều kiện sau if).

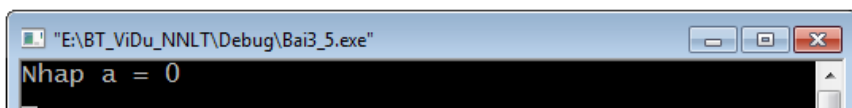
**Ví dụ 3.5:**

Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi  $a \neq 0$

```
#include <stdio.h>
#include <conio.h>
int main ()
{
    float a;
    printf("Nhập a = "); scanf("%f",&a);
    if (a !=0 )
        printf("Nghich dao cua %f la %f",a,1/a);
    getch();
    return 0;
}
```

Nếu nhập  $a \neq 0$ , thì câu lệnh `printf("Nghich dao cua %f la %f", a, 1/a)` được thực hiện. Ngược lại, nếu  $a=0$ , câu lệnh này không được thực hiện.

**Kết quả thực thi chương trình khi nhập a = 10****Kết quả thực thi chương trình khi a=0**



- **Lệnh if – else** : Cú pháp:

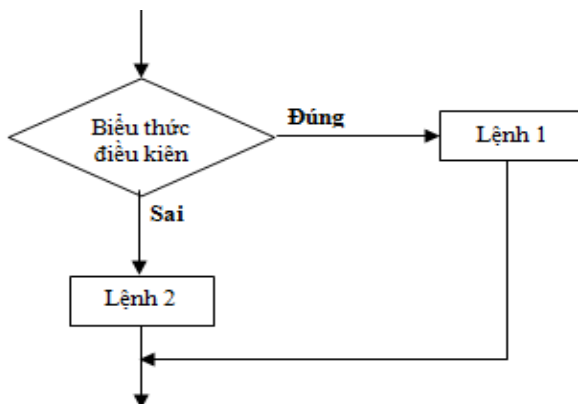
**if (<Biểu thức điều kiện>)**

**<Lệnh 1>**

**else**

**<Lệnh 2>**

Lưu đồ:



Hình 3-2. Lưu đồ thực hiện lệnh của câu lệnh if-else đơn giản

Giải thích:

- + *Lệnh 1*, *Lệnh 2* được thể hiện là một câu lệnh đơn, một khối lệnh hay một câu lệnh phức.
- + Đầu tiên *Biểu thức điều kiện* được kiểm tra trước.
- + Nếu điều kiện đúng thì thực hiện *Lệnh 1*.
- + Nếu điều kiện sai thì thực hiện *Lệnh 2*.
- + Các lệnh phía sau *Lệnh 2* không phụ thuộc vào điều kiện.

Lệnh if-else đơn giản làm giảm đi độ phức tạp của chương trình, làm cho chương trình dễ hiểu hơn.

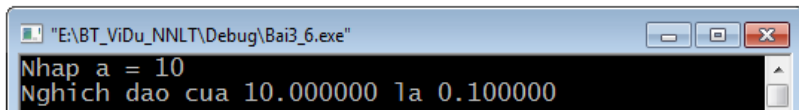
**Ví dụ 3.6:**

Yêu cầu người thực hiện chương trình nhập vào một số thực a. In ra màn hình kết quả nghịch đảo của a khi  $a \neq 0$ , khi  $a=0$  in ra thông báo “Không thể tìm được nghịch đảo của a”

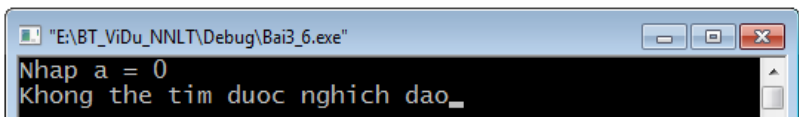
```
#include <stdio.h>
#include <conio.h>
int main ()
{
    float a;
    printf("Nhập a = "); scanf("%f",&a);
    if (a !=0 )
        printf("Nghịch đảo của %f là\n%f",a,1/a);
    else
        printf("Không thể tìm được nghịch\nđảo");
    getch();
    return 0;
}
```

Nếu nhập vào  $a \neq 0$  thì câu lệnh `printf("Nghịch đảo của %f là %f",a,1/a)` được thực hiện, ngược lại câu lệnh `printf("Không thể tìm được nghịch đảo của a")` được thực hiện.

### Kết quả thực thi của chương trình khi nhập $a \neq 0$



### Kết quả thực thi của chương trình khi nhập $a = 0$



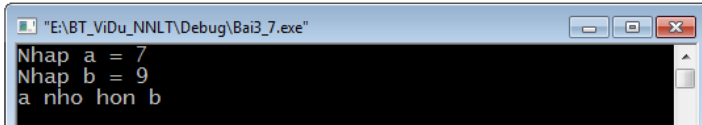
### Ví dụ 3.7:

Nhập vào 2 số a,b. So sánh số a với số b vừa nhập.

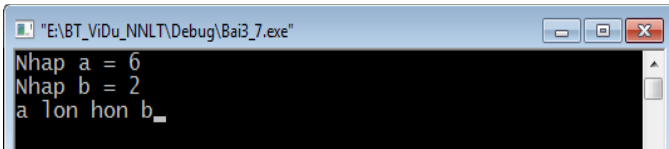
```
#include <stdio.h>
#include <conio.h>
int main ()
{
    int a, b;
    printf("Nhap a = "); scanf("%d", &a);
    printf("Nhap b = "); scanf("%d", &b);
    if (a > b )
        printf("a lon hon b");
    else
        if (a==b)
            printf("hai so bang nhau");
        else
            printf("a nho hon b");

    getch();
    return 0;
}
```

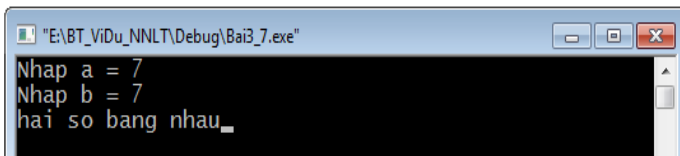
**Kết quả thực thi chương trình khi nhập a=7, b=9**



**Kết quả thực thi chương trình khi nhập a=6, b=2**



**Kết quả thực thi khi a=7 b=7**



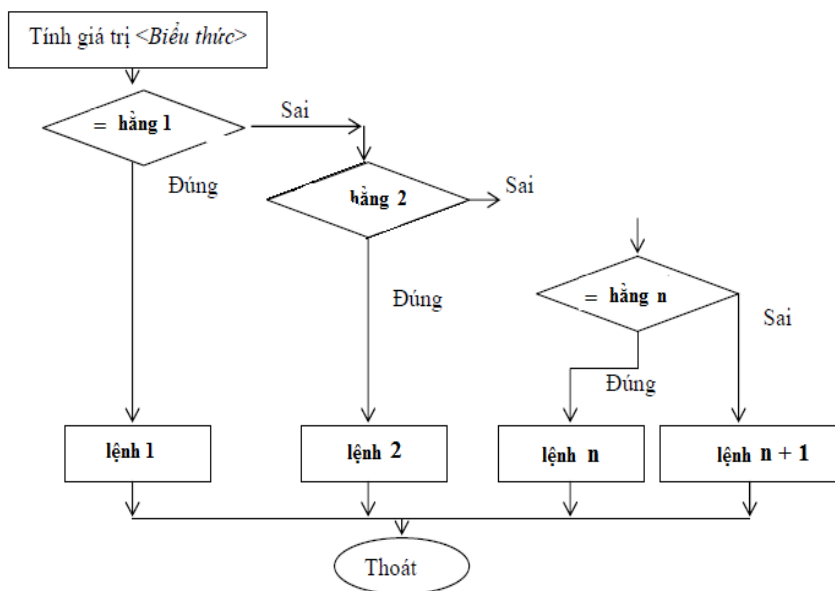
### 3.2.2 Lệnh switch case

Câu lệnh rẽ nhánh switch-case cho phép lựa chọn một trong các lựa chọn đã đưa ra. Nếu biểu thức có giá trị bằng với một giá trị sau từ khóa case nào đó, thì câu lệnh tương ứng sẽ được thực thi.

Cú pháp:

```
switch (<biểu thức>)
{
    case <hằng 1> :    đoạn lệnh 1 ; break;
    case <hằng 2> :    đoạn lệnh 2 ; break;
    .....
    case <hằng n> :    đoạn lệnh n ; break ;
    default :    lệnh n+1;
}
```

Lưu đồ :



Hình 3-3. Lưu đồ thực hiện của câu lệnh switch - case

Giải thích:

- Tính giá trị của biểu thức trước.
- Nếu giá trị của biểu thức bằng hằng 1 thì thực hiện lệnh 1 rồi thoát.
- Nếu giá trị của biểu thức khác hằng 1 thì so sánh với hằng 2, nếu bằng hằng 2 thì thực hiện lệnh 2 rồi thoát.
- Cứ như thế, so sánh tới hằng n.
- Nếu tất cả các phép so sánh trên đều sai thì thực hiện lệnh n+1 mặc định của trường hợp *default*.

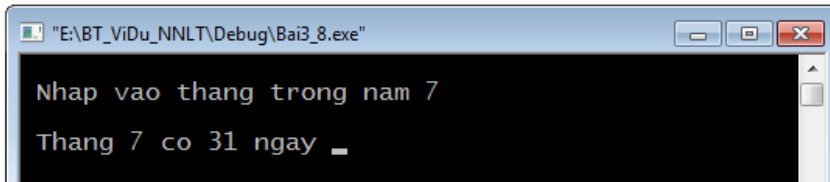
**Ví dụ 3.8:**

Nhập vào giá trị tháng của năm, xuất số ngày trong tháng.

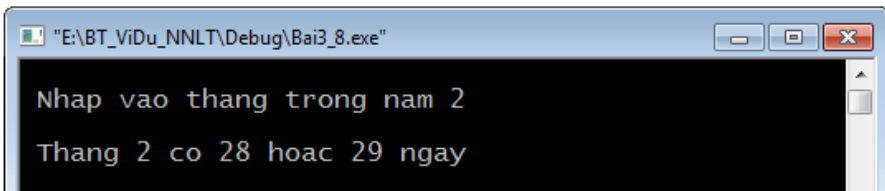
```
#include <stdio.h>
#include<conio.h>
void main()
{
    int thang;
    printf("\n Nhap vao thang trong nam ");
    scanf("%d",&thang);
    switch(thang)
    {
        case 1:
        case 3:
        case 5:
        case 7:
        case 8:
        case 10:
        case 12:
            printf("\n Thang %d co 31 ngay",thang);
            break;
        case 4:
```

```
case 6:
case 9:
case 11:
    printf("\n Tháng %d có 30 ngày
           ",thang);
    break;
case 2:
    printf ("\n Tháng 2 có 28 hoặc 29
           ngày");
    break;
default :
    printf("\n Không có tháng %d",
           thang);
    break;
}
}
```

### Kết quả thực thi khi nhập tháng = 7



### Kết quả thực thi khi nhập tháng = 2



### Ví dụ 3.9:

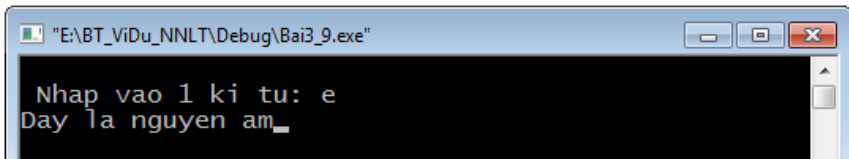


Nhập vào một kí tự, cho biết kí tự đó là nguyên âm hay phụ âm

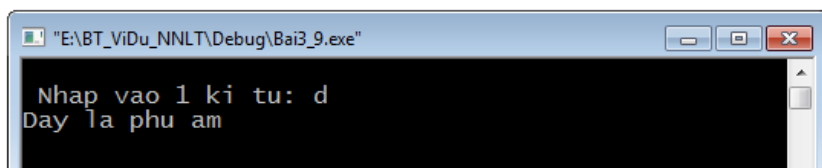
```
#include <stdio.h>
#include <conio.h>

void main ()
{
    char ch;
    printf("\n Nhập vào 1 kí tự: ");
    scanf("%c",&ch);
    switch(ch)
    {
        case 'a' :
        case 'o' :
        case 'e' :
        case 'u' :
        case 'y' :
        case 'i' : printf("Day la nguyen am") ;
                    break ;
        default : printf("Day la phu am");
    }
    getch();
}
```

**Kết quả thực thi chương trình khi nhập kí tự e:**



**Kết quả thực thi chương trình khi nhập kí tự d:**



Chú ý: Nếu câu lệnh case  $N_i$  không có câu lệnh break, thì máy sẽ tự động thực hiện câu lệnh của case  $N_{i+1}$

### 3.3 Lệnh lặp

Lệnh lặp là một câu lệnh, một đoạn lệnh trong chương trình được thực hiện lặp đi lặp lại nhiều lần cho đến khi một điều kiện xác định được thỏa mãn. Có thể nói một lệnh lặp cho phép lặp lại các câu lệnh nhiều lần.

#### 3.3.1 Lệnh for

Lệnh for thực thi việc lặp lại một câu lệnh, một khối lệnh nhiều lần với số lần lặp xác định trước.

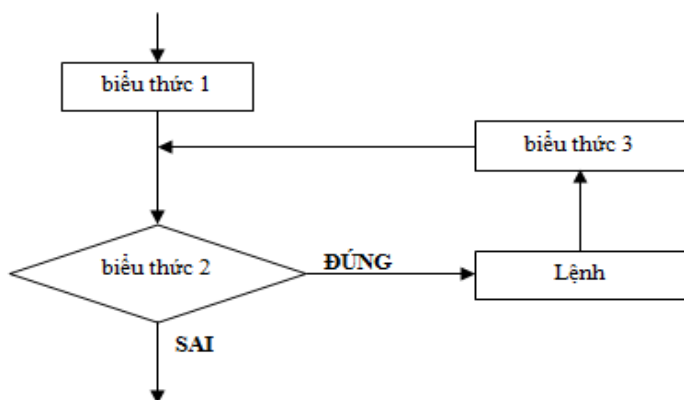
Cú pháp :

**for ( [ <biểu thức 1>; [ <biểu thức 2> ] ; [ <biểu thức 3> ] )**  
**<câu lệnh>;**

Quá trình thực hiện câu lệnh for:

- Bước 1: Xác định giá trị của biểu thức 1.
- Bước 2: Xác định giá trị của biểu thức 2.
- Bước 3: Nếu biểu thức 2 sai thì sẽ thoát vòng lặp for;  
Nếu biểu thức 2 đúng thì máy sẽ thực hiện câu lệnh .
- Bước 4: Tính giá trị của biểu thức 3 và quay lại Bước 2.

Lưu đồ:



Hình 3-4. Lưu đồ thực hiện của câu lệnh for

Ta có một số chú ý như sau:

- Biểu thức 1 là biểu thức gán trị khởi động cho biến lặp.
- Biểu thức 2 là biểu thức điều kiện. Nếu biểu thức 2 vắng mặt, điều kiện luôn đúng.
- Biểu thức 3 thông thường là biểu thức thay đổi điều kiện.
- Biểu thức 1, 3 có thể gồm nhiều biểu thức cách nhau bởi dấu phẩy.
- Biểu thức thứ 2 có thể bao gồm nhiều biểu thức, nhưng tính đúng sai của nó được xem là tính đúng sai của biểu thức cuối cùng.

### Ví dụ 3.10:

Viết chương trình nhập vào một số nguyên n, sau đó tính tổng của các số nguyên từ 1 đến n.

```

#include <stdio.h>
#include <conio.h>
int main ()
{
    int n,i,tong;
    printf("\n Nhập vào số nguyên dương n:");
    scanf("%d",&n);
    tong=0;

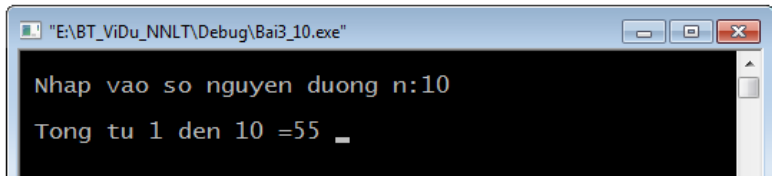
```

```

for (i=1; i<=n; i++)
    tong+=i;
printf("\n Tong tu 1 den %d =%d ",n,tong);
getch();
return 0;
}

```

### Kết quả thực thi chương trình:



Đối với câu lệnh for, ta có thể dùng câu lệnh break để thoát khỏi vòng lặp for tại một trường hợp nào đó.

### Ví dụ 3.11:

```

#include <stdio.h>
#include<conio.h>
int main()
{
    int i, j ;
    printf(" nhập hai số nguyên dương i và j :");
    scanf("%d%d",&i,&j);
    for( ; i > 0 &&j > 0; i- -,j- -)
    {
        if( j == 5 )
            break;
        printf(" i = %d, j = %d ", i, j);
    }
    getch();
    return 0;
}

```

Các vòng for có thể lồng với nhau để thực hiện một câu lệnh nào đó.

Xét ví dụ 3.12 là một minh họa cho việc dùng các câu lệnh lặp for lồng nhau nhằm thực hiện lặp lại 1 thao tác nhiều lần, và bản thân thao tác lặp cũng là một vòng lặp.

**Ví dụ 3.12 :**

Xuất bảng cửu chương từ 1 đến 9.

```
#include <stdio.h>
#include <conio.h>
int main()
{
    for( int i = 1 ; i <= 9 ; i++)
    {
        printf("\n bang cuu chuong thu %d ", i);
        for(int j = 1 ; j <= 9 ; j ++ )
            printf(" %d x %d = %d ", i, j, i * j);
    }
    getch();
    return 0;
}
```

### 3.3.2 Lệnh while

Lệnh while thực thi việc lặp lại một khối lệnh khi điều kiện kiểm tra là đúng. Điều kiện sẽ được kiểm tra trước khi vào thân vòng lặp do đó nếu có thay đổi giá trị kiểm tra ở trong thân vòng lặp thì khối lệnh vẫn được thực thi cho đến khi kết thúc khối lệnh. Nếu điều kiện kiểm tra là sai (FALSE) ngay từ đầu thì khối lệnh sẽ không được thực hiện dù chỉ là một lần.

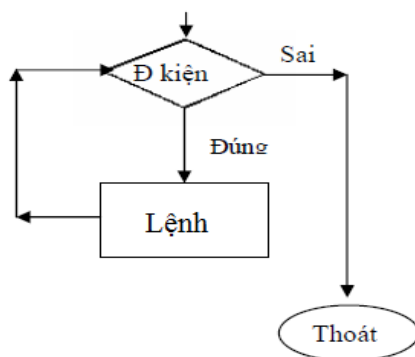
Cú pháp:

**while(<biểu thức điều kiện>)**  
**< Lệnh >**

Quá trình thực hiện của vòng lặp while:

- Bước 1: Tính giá trị của biểu thức điều kiện.
- Bước 2: Nếu biểu thức điều kiện là sai (FALSE), thì chương trình sẽ thoát ra khỏi vòng lặp. Nếu biểu thức điều kiện là đúng (TRUE) thì chương trình sẽ thực hiện câu lệnh và quay lại bước 1.

Lưu đồ:



Hình 3-5. Lưu đồ thực hiện của câu lệnh while

Chú ý: Trong biểu thức điều kiện của vòng while có thể gồm nhiều biểu thức cách nhau bởi dấu phẩy “,” nhưng tính đúng sai của nó là tính đúng sai của biểu thức sau cùng.

### Ví dụ 3.13:

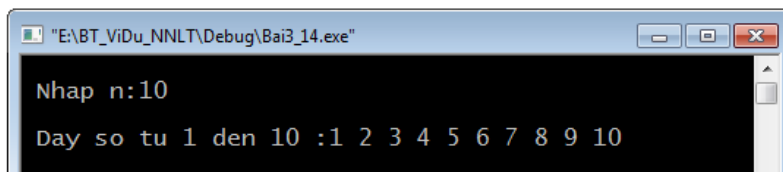
In các số nguyên từ 1 đến n, trong đó n nhập từ phím

```

#include <stdio.h>
#include<conio.h>
int main ()
{
    int i,n;
    printf("\n Nhập n:"); scanf("%d", &n);
    printf("\n Day so tu 1 den %d :",n);

    i=1;
    while (i<=n)
        printf("%d ",i++);
    getch();
    return 0;
}
  
```

### Kết quả thực thi chương trình khi nhập n=10



```
"E:\BT_ViDu_NNLT\Debug\Bai3_14.exe"
Nhap n:10
Day so tu 1 den 10 :1 2 3 4 5 6 7 8 9 10
```

Để tránh xảy ra trường hợp lặp vô hạn, cần chú ý:

- Giá trị của biến được sử dụng trong biểu thức phải được thiết lập trước khi vòng lặp **while** thực hiện. Đây gọi là bước khởi tạo giá trị. Lệnh này chỉ thực hiện một lần trước khi thực hiện vòng lặp.
- Thân vòng lặp phải làm thay đổi giá trị của biến trong biểu thức kiểm tra. Biến này được gọi là biến tăng (**incremented**) nếu giá trị trong thân vòng lặp tăng, và được gọi là biến giảm (**decremented**) nếu giá trị giảm.

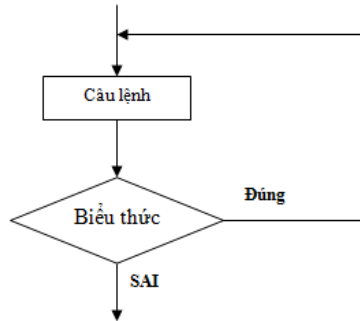
#### 3.3.3 Lệnh do...while

Lệnh do...while thực thi việc lặp lại một khối lệnh nhiều lần. Nó thực hiện khối lệnh ít nhất một lần. Sau đó sẽ kiểm tra điều kiện nếu điều kiện là đúng thì tiếp tục thực thi khối lệnh cần lặp. Nếu điều kiện là sai thì kết thúc vòng lặp.

Cú pháp:

```
do
< câu lệnh>
while(<biểu thức>)
```

Lưu đồ:



Hình 3-6. Lưu đồ thực hiện câu lệnh do-while

Quy trình thực hiện:

- Bước 1: Câu lệnh được thực hiện trước tiên.
- Bước 2: Tính giá trị của biểu thức, nếu biểu thức đúng thì quay lại bước 1, nếu giá trị biểu thức sai thì ngừng vòng lặp.

**Ví dụ 3.14:**

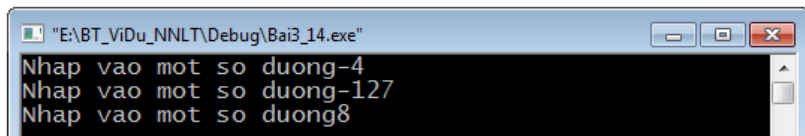
Viết chương trình bắt buộc nhập vào một số dương, nếu nhập số âm yêu cầu nhập lại.

```

#include <stdio.h>
#include<conio.h>
int main ()
{
    int value;
    do
    {
        printf( "Nhap vao mot so duong");
        scanf("%d",&value);
    } while(value <=0);
    getch();
    return 0;
}
  
```



### Kết quả thực thi chương trình



```
"E:\BT_ViDu_NNLT\Debug\Bai3_14.exe"  
Nhap vao mot so duong-4  
Nhap vao mot so duong-127  
Nhap vao mot so duong8
```

Người sử dụng được nhắc nhập vào số dương value, điều kiện đúng sau while được kiểm tra. Nếu một giá trị dương được nhập vào, khi đó điều kiện sai và vòng lặp kết thúc. Ngược lại, câu thông báo được hiển thị cho tới khi người sử dụng nhập vào một số dương.

### BÀI TẬP CHƯƠNG 3

1. Viết chương trình nhập vào số nguyên, kiểm tra số đã nhập là số âm hay số dương.
2. Viết chương trình nhập vào hai số  $a, b$ . Tìm số lớn nhất giữa hai số vừa nhập (so sánh 2 cách giải: dùng hàm if và biểu thức điều kiện).
3. Viết chương trình nhập vào một số nguyên, kiểm tra số vừa nhập là số chẵn hay số lẻ.
4. Viết chương trình nhập vào hai số nguyên  $a, b$ . So sánh hai giá trị vừa nhập vào và xuất ra kết quả so sánh:  $a$  lớn hơn  $b$ ,  $a$  bằng  $b$ ,  $a$  nhỏ hơn  $b$ .
5. Viết chương trình nhập vào hai số  $a, b$ . Kiểm tra  $a$  có là bội số của  $b$  không.
6. Viết chương trình nhập vào đơn giá một mặt hàng, và số lượng bán của mặt hàng. Tính tiền khách phải trả, với thông tin như sau:

Thành tiền: đơn giá \* số lượng.

Giảm giá : Nếu thành tiền  $> 100$ , thì giảm 3% thành tiền, ngược lại không giảm.

Tổng tiền phải trả: thành tiền – giảm giá.

7. Viết chương trình giải phương trình bậc 1 ( $ax + b = 0$ ).
8. Viết chương trình giải phương trình bậc 2. ( $ax^2 + bx + c = 0$ )
9. Viết chương trình nhập vào tháng trong năm. Xuất kết quả cho biết số ngày của tháng đó.

Biết rằng:

- tháng 1, 3, 5, 7, 8, 10, 12: có 31 ngày.
  - tháng 4, 6, 9, 11: có 30 ngày.
  - tháng 2: có 28 hoặc 29 ngày.
10. Nhập vào 1 số là năm. Cho biết năm đó nhuận hay không nhuận. Biết rằng năm  $N$  là nhuận khi  $N$  chia hết cho 400 hoặc  $N$  chia hết cho 4 nhưng không hết cho 100.

11. Viết chương trình cho phép người dùng chọn một trong những thác tác sau;
  - Nhập chiều dài, chiều rộng hình chữ nhật. Xuất ra chu vi, diện tích hình chữ nhật.
  - Nhập chiều dài cạnh hình vuông. Tính và xuất kết quả chu vi, diện tích hình vuông.
  - Nhập bán kính hình tròn. Tính và xuất kết quả chu vi, diện tích hình tròn.
12. Viết chương trình nhập vào hai số nguyên và lựa chọn phép toán (cộng, trừ, nhân, chia) để tính kết quả.
13. Viết chương trình in trên màn hình các số từ 1->100, các số ngăn cách nhau bởi 1 đoạn khoảng trắng.
14. Viết chương trình tính tổng:  $1 + 2 + 3 + 4 + 5 + \dots + 20$
15. Viết chương trình tính tích:  $1*2*3*4*5* \dots *n$ , trong đó  $n>0$  nhập từ phím.
16. Viết chương trình tính tổng:  $2 + 4 + 6 + 8 + \dots + 20$ , trong đó  $n>0$  nhập từ phím.
17. Viết chương trình tính tổng:  
 $1*2 + 2*3 + 3*4 + 4*5 + \dots + n(n+1)$ ,  
 trong đó  $n>0$  nhập từ phím.
18. Viết chương trình tính tổng:  

$$\frac{1}{1.2.3} + \frac{1}{2.3.4} + \frac{1}{3.4.5} + \dots + \frac{1}{n(n+1)(n+2)}$$
 trong đó  $n>0$  nhập từ phím.
19. Viết chương trình tính tổng:  
 $1^2 + 2^2 + 3^2 + \dots + n^2$ ,                      với  $n > 0$  nhập từ bàn phím.
20. Viết chương trình tính tổng:  
 $1 + (1+2) + (1+2+3) + \dots + (1+2+3+\dots+n)$ ,                      với  $n > 0$  nhập từ bàn phím.

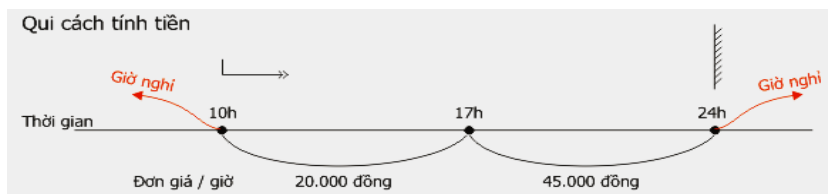
21. Viết chương trình in bảng cửu chương từ 1 -> 9 theo hàng ngang.
22. Viết chương trình in bảng cửu chương số n, n nhập từ bàn phím.
23. Viết chương trình vẽ hình chữ nhật có kích thước d x r, trong đó d là chiều dài, và r là chiều rộng được nhập từ phím.

Ví dụ: nhập d=5, r = 3

```

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

24. Viết chương trình hiển thị tất cả các số lẻ nhỏ hơn n, trong đó n nhập từ phím.
25. Viết chương trình tính tổng các số chẵn nhỏ hơn n, trong đó n nhập từ bàn phím.
26. Viết chương trình in ra các số là bội số của 5 nhỏ hơn n, trong đó n nhập từ phím.
27. Viết chương trình nhập vào 2 số nguyên x, y. Tìm ước chung lớn nhất và bội chung nhỏ nhất của chúng.
28. Viết chương trình tính tiền karaoke theo cách sau:



29. Viết chương trình tính tiền điện sử dụng trong tháng:
  - Từ 1 – 100KW: 5\$
  - Từ 101 – 150KW: 7\$
  - Từ 151 – 200KW: 10\$
  - Từ 201 – 300KW: 15\$
  - Từ 300KW trở lên: 20\$
30. Nhập vào số nguyên n. Nếu  $n > 5$  thì tăng n lên 2 đơn vị và trả về giá trị n, ngược lại trả về giá trị 0

31. Viết chương trình đếm số lượng số chẵn trong  $[n,m]$ , trong đó  $n,m$  nhập từ phím.
32. Viết chương trình tính tổng các số tự nhiên nhỏ hơn  $n$  (sử dụng vòng lặp while).
33. Viết chương trình xuất ra các số có 3 chữ số sao cho các chữ số khác nhau đôi một.
34. Viết chương nhập số nguyên lớn  $N$  (khai báo: long  $N$ ) có  $k$  chữ số.
  - Tìm chữ số hàng đầu tiên của  $N$ .
  - Tính tổng các chữ số của  $N$ .
  - Tìm chữ số lớn nhất trong  $k$  chữ số đó.
  - Đếm số chữ số của  $N$ .
  - $N$  có phải là số có các chữ số khác nhau đôi một không?

Gợi ý: Áp dụng các phép toán  $/$  và  $\%$  cho 10 để lấy lần lượt từng chữ số của  $N$  theo chiều từ chữ số hàng đơn vị đến các chữ số hàng chục, hàng trăm,...

Ví dụ:  $N = 347285108$ ,  $x = N\%10=8$ ,  $y = N/10 = 34728510$ . Lặp lại 2 phép toán chia này ta lần lượt lấy được từng chữ số của  $N$  từ hàng đơn vị.

Khi đó  $N$  có:

- Chữ số hàng đầu tiên là 3.
  - Tổng các chữ số:  $3+4+7+2+8+5+1+0+8 = 38$
  - Chữ số lớn nhất là 8.
  - Số chữ số của  $N$ : 9
  - $N$  không phải số có các chữ số khác nhau đôi một vì có 2 chữ số 8.
35. Nhập vào ba cạnh  $a, b, c$  của tam giác. Xuất ra màn hình tam giác đó thuộc loại tam giác gì? (Thường, cân, vuông, đều hay vuông cân).

## CHƯƠNG 4. HÀM

### 4.1 Khái niệm hàm (Function)

Trong những chương trình lớn, có thể có những đoạn chương trình viết lặp đi lặp lại nhiều lần, để tránh rườm rà và mất thời gian khi viết chương trình; người ta thường phân chia chương trình thành nhiều module, mỗi module giải quyết một câu lệnh nào đó. Các module như vậy gọi là các chương trình con.

Một tiện lợi khác của việc sử dụng chương trình con là ta có thể dễ dàng kiểm tra xác định tính đúng đắn của nó trước khi ráp nối vào chương trình chính và do đó việc xác định sai sót để tiến hành hiệu đính trong chương trình chính sẽ thuận lợi hơn. Trong C, chương trình con được gọi là hàm. Hàm trong C có thể trả về kết quả thông qua tên hàm hay có thể không trả về kết quả.

Hàm `main()` là một hàm đặc biệt của ngôn ngữ lập trình C và là hàm đầu tiên được thực hiện trong chương trình. Khi thực hiện, hàm này sẽ gọi các hàm khác để thực hiện các chức năng riêng rẽ. Các hàm có thể ở trên các tập tin khác nhau và được biên dịch riêng lẻ, sau đó được ghép nối với nhau thành chương trình hoàn chỉnh.

Hàm có hai loại: hàm chuẩn và hàm tự định nghĩa. Trong chương này, ta chú trọng đến cách định nghĩa hàm và cách sử dụng các hàm đó.

Một hàm khi được định nghĩa thì có thể sử dụng bất cứ đâu trong chương trình. Trong C, một chương trình bắt đầu thực thi bằng hàm `main`.

#### ***Ví dụ 4.1:***

Hàm tìm số lớn giữa 2 số nguyên a, b.

```
int max(int a, int b)
{
    return (a>b) ? a:b;
}
```

**Ví dụ 4.2:**

Chương trình nhập vào hai số nguyên a,b, tìm số lớn nhất trong 2 số.

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b)
{
    return (a>b) ? a:b;
}
int main()
{
    int a, b;
    printf("\n Nhập vào 2 số a, b");
    scanf("%d%d", &a, &b);
    printf("\n Số lớn là %d", max(a, b));
    getch();
    return 0;
}
```

**4.2 Định nghĩa hàm**

Hàm người dùng là những hàm do người lập trình tự tạo ra nhằm đáp ứng nhu cầu xử lý của mình. Định nghĩa một hàm được hiểu là việc chỉ rõ các lệnh cần phải thực hiện mỗi khi hàm đó được gọi đến. Như vậy, có thể nói rằng tất cả các hàm được sử dụng trong một chương trình đều phải có một định nghĩa tương ứng cho nó.

Các hàm này nếu không có sẵn trong thư viện chúng ta phải định nghĩa trước. Nếu các hàm có sẵn sẽ được ngôn ngữ lập trình C tìm và lấy ra từ thư viện.

Cú pháp:

```
<Tên kiểu kết quả><Tên hàm> ([<kiểu t số><tham số>][...])
{
    [<Khai báo biến cục bộ và các câu lệnh thực hiện hàm>]
    [return <Biểu thức>;]
}
```

Trong đó:

- Tên kiểu kết quả: là kiểu dữ liệu của kết quả trả về, có thể là `int`, `byte`, `char`, `float`, `void`... Một hàm có thể có hoặc không có kết quả trả về. Trong trường hợp hàm không có kết quả trả về ta nên sử dụng kiểu kết quả là `void`.
- Tên\_hàm: phải đặt hợp lệ và không trùng với một biến nào hoặc một từ khoá nào của C.
- Kiểu tham số: là kiểu dữ liệu của tham số.
- Tham số: là tham số truyền dữ liệu vào cho hàm, một hàm có thể có hoặc không có tham số. Tham số này gọi là tham số hình thức, khi gọi hàm chúng ta phải truyền cho nó các tham số thực. Nếu có nhiều tham số, mỗi tham số phân cách nhau dấu phẩy (,).
- Bên trong thân hàm (phần giới hạn bởi cặp dấu {}): là các khai báo cùng các câu lệnh xử lý. Các khai báo bên trong hàm được gọi là các khai báo cục bộ trong hàm và các khai báo này chỉ tồn tại bên trong hàm mà thôi.

Khi định nghĩa hàm, ta thường sử dụng câu lệnh `return` để trả về kết quả thông qua tên hàm. Lệnh `return` dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.

Cú pháp:

```
return ; /*không trả về giá trị*/  
return <biểu thức>; /*Trả về giá trị của biểu thức*/  
return (<biểu thức>; /*Trả về giá trị của biểu thức*/
```

Nếu hàm có kết quả trả về, ta bắt buộc phải sử dụng câu lệnh `return` để trả về kết quả cho hàm.



### 4.3 Thực thi hàm

Cú pháp:

<b>&lt;Tên hàm&gt;([Danh sách các tham số])</b>
---

Một hàm khi định nghĩa thì chúng vẫn chưa được thực thi trừ khi có lời gọi đến hàm đó. Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi. Quá trình diễn ra như sau:

- Nếu hàm có tham số, trước tiên các tham số sẽ được gán giá trị thực tương ứng.
- Chương trình sẽ thực hiện tiếp các câu lệnh trong thân hàm bắt đầu từ lệnh đầu tiên đến câu lệnh cuối cùng.
- Khi gặp lệnh return hoặc dấu } cuối cùng trong thân hàm, chương trình sẽ thoát khỏi hàm để trở về chương trình gọi nó và thực hiện tiếp tục những câu lệnh của chương trình này.

#### **Ví dụ 4.3:**

Xuất ra màn hình một tam giác cân hợp bởi các kí tự '\*'

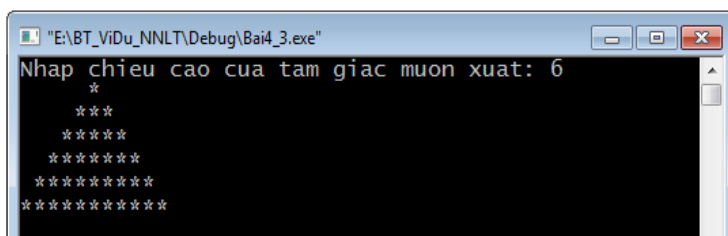
```
#include<stdio.h>
#include<conio.h>
void XuatC(char x, int n)
{
    int i;
    for(i = 1; i <= n; i++) putchar(x);
}
// Hàm chính
void main()
{
    int h, i;
    printf("Nhap chieu cao cua tam giac muon xuat: ");
    scanf("%d", &h);
    for(i = 1; i <= h; i++)
```

```

{    XuatC(' ', h-i); //gọi hàm thực thi
    XuatC('*',2*i-1); //gọi hàm thực thi
    XuatC('\n',1); //gọi hàm thực thi
}
getch();
}

```

### Kết quả thực thi chương trình



Ở ví dụ trên dòng *void XuatC (char x, int n)* sẽ cho ta biết rằng tên hàm là XuatC, các đối số là x, n lần lượt có kiểu là char và int. Từ khóa void cho ta biết rằng hàm này không trả về giá trị nào cả.

#### Ví dụ 4.4:

Chương trình tính lũy thừa của một số.

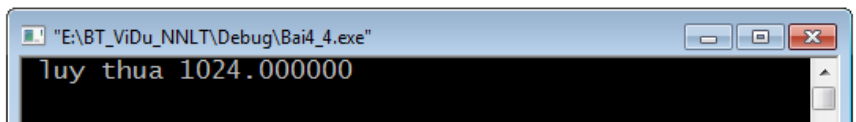
```

#include<stdio.h>
#include<conio.h>
float luythua(float x, int k)
{
    float r =1;
    while(k > 0)
    {
        r *= x;
        k--;
    }
    return r;
}

```

```
void main()  
{  
    int k;  
    float n, nk ;  
    n = 4; k = 5;  
    nk = luythua(n, k);  
    printf(" luy thua %f", nk);  
    getch();  
}
```

### Kết quả thực thi chương trình



Trong hàm lũy thừa có thể thấy rằng  $x$  và  $k$  là đối số, và khi đó trong hàm `main()`, đối số vào  $n$ ,  $k$  được truyền cho hàm `luythua(x, k)`.

Dòng lệnh `nk = luythua(n, k);` sẽ thực hiện việc lũy thừa và trả về kết quả này cho biến `nk`.

Trong ngôn ngữ lập trình C có ba loại biến :

- Biến toàn cục (global): Là biến khai báo bên ngoài mọi hàm.
- Biến cục bộ (local): Là biến khai báo bên trong một hàm. Biến này chỉ tồn tại khi hàm đang được thực thi. Vùng nhớ của biến cục bộ sẽ bị thu hồi khi hàm thực hiện xong.
- Biến tĩnh (static): Khai báo biến với chỉ thị `static` bên trong một hàm. Biến này vẫn tồn tại sau khi hàm thực thi xong.

### 4.4 Truyền tham số

Khi thực hiện một câu lệnh, chúng ta luôn luôn cần dữ liệu để làm tác động. Dữ liệu mà hàm sẽ tác động gọi là tham số (parameter) của hàm. Các tham số này ta xem như là dữ liệu đã có rồi.

Trong C ta dùng kỹ thuật truyền tham số bằng trị để không thay đổi giá trị và truyền tham số bằng địa chỉ hoặc tham chiếu để thay đổi giá trị của tham số. Mặc nhiên, việc truyền tham số cho hàm trong C là truyền theo giá trị nghĩa là các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các tham số hình thức.

**Ví dụ 4.5:**

```
void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

Ta thực hiện hàm main() như sau:

```
void main()
{
    int a,b;
    a = 10; b = 20;
    swap(a,b);
    printf("gia tri cua a=%d và b=%d",a,b);
}
```

Giá trị của a và b khi xuất ra màn hình là: a = 10, b = 20. Trong trường hợp này, giá trị 10 và 20 của a và b từ chương trình chính được truyền vào tham số hình thức x,y tương ứng trong hàm swap (x=10, y=20). Hàm swap thực hiện việc hoán đổi giá trị của x và y, kết quả là x và y hoán đổi giá trị cho nhau (x=20, y=10). Kết thúc hàm swap, quay về hàm main(), kết quả a và b không thay đổi (a=10, b=20).

Ta có thể viết hàm swap như sau:

**Ví dụ 4.6:**

```
void swap(int &x,int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
void main()
{
    int a,b;
    a = 10; b = 20;
    swap(a,b);
    printf("Gia tri cua a=%d và b=%d", a, b);
}
```

Giá trị của a và b khi xuất ra màn hình là: a = 20, b = 10. Trong trường hợp này, địa chỉ của biến a và b được truyền vào tham số hình thức x, y tương ứng trong hàm swap. Hàm swap thực hiện việc hoán đổi giá trị trong địa chỉ biến a và b, kết quả là a và b hoán đổi giá trị cho nhau. Kết thúc hàm swap, quay về hàm main(), kết quả a và b thay đổi giá trị (a=20, b=10).

#### 4.5 Kết quả trả về

Nếu hàm trả về giá trị, giá trị đó phải được trả về thông qua lệnh return.

**Ví dụ 4.8:**

Hàm kiểm tra kí tự có phải là một chữ số hay không.

```
int isdigit(char C)
{
    return ((C >= '0' && C <= '9')?1:0);
}
```

Nếu một hàm không trả về giá trị thì không cần dùng lệnh return.

**Ví dụ 4.9:**

```
void mean(int num1,int num2)
{
    printf(" trung bình cộng 2 so: %f,
(float) (num1 + num2)/2;

    /* vì 2 so num1 và num2 là số nguyên nên phép
/ cho ra kết quả số nguyên. Do đó nếu muốn kết quả
trung bình là số thực thì cần "ép kiểu" tổng trước
khi thực hiện chia.
}
```

**4.6 Prototype của hàm**

Cú pháp:

**<Tên Kiểu> <Tên Hàm> ([Danh sách các đối số]);**

Trong đó kiểu bắt buộc phải có là kiểu trả về của hàm. Kiểu này có thể là một trong những kiểu cơ bản hay kiểu void. Danh sách đối số là khai báo các đối số của hàm, các đối số của hàm được ngăn cách với nhau bằng dấu phẩy.

**Ví dụ 4.10:**

```
double atof(char s[]);
void func(int i,int j);
double luythua(double n, int so_mu);
```

Prototype của một hàm không những được dùng để khai báo kiểu của kết quả trả về của một hàm mà còn được dùng để kiểm tra các đối số và chuyển kiểu đối số. Việc đưa một prototype của một hàm giúp cho chương trình biên dịch có thể kiểm tra chặt chẽ hơn.

**Ví dụ 4.11:**

Cách sử dụng hàm lũy thừa trong chương trình chính

```
double luythua(double n,int pow);
```

và hàm `main()` được thực hiện như sau:

```
void main()  
{  
    int num;  
    num = 123;  
    num = luythua(num,2);  
    printf("%6d", num);  
}
```

`num` là một biến có kiểu `int` nhưng hàm lũy thừa vẫn thực hiện đúng nhờ sự chuyển kiểu bắt buộc của prototype.

#### 4.7 Các hàm chuẩn

Bên cạnh việc chúng ta có thể tự tạo hàm, tất cả các ngôn ngữ lập trình đều có một số hàm đã được định nghĩa trước. Trong nhiều trường hợp, chúng ta dựa vào các hàm chuẩn để tạo hàm riêng của mình.

##### *Ví dụ 4.12 :*

`avg()` – tính trung bình.

`sqrt()` – tính căn bậc hai.

#### 4.8 Thư viện hàm

Thư viện hàm là tập hợp các hàm đã được xây dựng trước. Mỗi thư viện hàm chứa các hàm theo một công dụng riêng. Ví dụ, thư viện hàm toán học chứa các hàm về phép tính toán học như căn bậc hai, trung bình, và lũy thừa,... Đó là `math.h`.

#### 4.9 Sự đệ quy

Đôi khi chúng ta rất khó định nghĩa đối tượng một cách tường minh, nhưng có thể dễ dàng định nghĩa đối tượng này qua chính nó. Kỹ thuật này gọi là đệ quy, đó cũng chính là việc định nghĩa nội dung thông qua chính bản thân của nó nhưng ở mức độ nhỏ hơn.

Ta xét một ví dụ minh họa phương pháp đệ quy qua một bài toán tính giai thừa. Ta biết rằng  $n!$  được định nghĩa như sau:

$$n! = \begin{cases} 1 & \text{với } n = 1 \\ n * (n - 1)! & \text{với } n > 1 \end{cases}$$

Áp dụng định nghĩa trên ta có thể tính  $5!$  như sau:

$$\begin{array}{c} 5! = 5 * 4! \\ \downarrow \\ 4! = 4 * 3! \\ \downarrow \\ 3! = 3 * 2! \\ \downarrow \\ 2! = 2 * 1! \\ \downarrow \\ 1 \end{array}$$

Suy ra  $5! = 120$

Một thuật toán được gọi là đệ quy nếu nó giải bài toán bằng cách rút gọn liên tiếp bài toán ban đầu tới bài toán giống như vậy nhưng có dữ liệu đầu vào nhỏ hơn.

Vì vậy, tư tưởng giải bài toán bằng đệ quy là đưa bài toán hiện tại về một bài toán cùng loại, cùng tính chất nhưng ở cấp độ thấp hơn chẳng hạn: độ lớn dữ liệu nhập nhỏ hơn, giá trị cần tính toán nhỏ hơn,... và quá trình này tiếp tục cho đến khi bài toán được đưa về một cấp độ mà tại đó có thể giải được. Từ kết quả ở cấp độ này, chúng ta sẽ đi ngược lại để giải bài toán ở cấp độ cao hơn cho tới khi giải được bài toán ban đầu.



Một thuật toán đệ quy gồm hai phần:

– **Phần cơ sở**

Là các trường hợp không cần thực hiện lại thuật toán cũng có nghĩa là khi làm đến đây sẽ không có việc gọi đệ quy nữa mà ở đây chỉ là một câu lệnh đơn giản dùng để kết thúc phần đệ quy. Nếu thuật toán đệ quy không có phần này thì sẽ dẫn đến việc lặp vô hạn và sẽ xuất hiện lỗi khi thi hành.

– **Phần đệ quy**

Là phần trong thuật toán có yêu cầu gọi đệ quy, tức là yêu cầu thực hiện lại thuật toán nhưng với cấp độ dữ liệu thấp hơn. Phần đệ quy này được dựa trên cơ sở công thức quy nạp của bài toán.

Bài toán đệ quy chỉ kết thúc khi và chỉ khi phần đệ quy gọi được hàm đệ quy ở mức cấp cơ sở. Điều này có nghĩa là trong hàm đệ quy cấp cơ sở chỉ thực hiện phần cơ sở của hàm, và không thực hiện phần đệ quy.

Chú ý: Bài toán đệ quy tốn rất nhiều vùng nhớ tạm để lưu trữ thông tin các cấp độ dữ liệu, nên khi sử dụng hàm đệ quy có thể tràn vùng nhớ tính toán.

**Ví dụ 4.13:**

Hàm đệ quy tính giai thừa của số n.

```
float tinhGT(int n)
{
    if (n==1)
        return 1;
    return n*tinhGT(n -1);
}
```

**Ví dụ 4.14:**

Hàm đệ quy tính  $S = 1/1 + 1/2 + 1/3 + \dots + 1/n$

```
float tinhS(int n)
{
    if (n==1) return 1;
    return tinhS(n-1) + 1/n;
}
```

## BÀI TẬP CHƯƠNG 4

Viết chương trình thực thi các bài tập sau, yêu cầu viết theo dạng chia chương trình thành các hàm:

1. Kiểm tra một số nguyên  $x$  là chẵn hay lẻ?
2. Kiểm tra một số nguyên  $x$  có nguyên tố không? Biết rằng số nguyên tố là số chỉ có 2 ước là 1 và chính nó.

Ví dụ: 2, 3, 5, 7, 11, 13, 17, 19, 23, ... là các số nguyên tố.

3. Kiểm tra một số nguyên  $x$  có là số chính phương không? Biết rằng số chính phương là bình phương của một số nguyên không âm.

Ví dụ: 1, ( $1=1*1$ ), 4 ( $4=2*2$ ), 9 ( $9=3*3$ ), 16 ( $16=4*4$ ), ... là các số chính phương.

4. Kiểm tra một số nguyên  $x$  có là số hoàn thiện (còn gọi là số hoàn chỉnh, số hoàn hảo) không? Biết rằng số hoàn thiện là số có tổng các ước của nó (trừ chính nó) bằng chính nó.

Ví dụ: 6 có các ước 1, 2, 3 và  $1+2+3=6$  nên 6 là số hoàn thiện.

5. Xuất  $n$  số nguyên tố đầu tiên với  $n>0$  nhập từ bàn phím.

Ví dụ:  $n=5$ . 5 số nguyên tố đầu tiên là:

2      3      5      7      11

6. Xuất  $n$  số chính phương đầu tiên với  $n>0$  nhập từ bàn phím.

Ví dụ:  $n=3$ . 3 số chính phương đầu tiên là: 1 4, 9

7. Đếm số ước của số nguyên  $N$ .

8. Nhập số  $N$  ( $0<n<1000$ , nhập sai thì yêu cầu nhập lại). Xuất ra cách đọc số  $N$ .

Ví dụ:  $N=256$  đọc: hai trăm nam mươi sáu.

9. Tìm số  $n$  lớn nhất sao cho  $1+2+3+...+n < M$  với  $M$  nhập từ bàn phím.

10. Nhập vào  $N$  là số ngày ( $N>0$ , nếu nhập sai thì thông báo và yêu cầu nhập lại). Cho biết  $N$  ngày đó có bao nhiêu tuần và ngày lẻ.

Ví dụ:  $N=37$ , có 5 tuần và 2 ngày lẻ.

11. Nhập vào số T giây. Đổi T về dạng h:m:s.

Ví dụ: T = 3850 giây  $\rightarrow$  1h:4m:10s

12. Nhập vào 3 phân số dạng a/b, c/d, e/f (với a,b,c,d,e,f là các số nguyên). Kiểm tra các mẫu số phải khác 0, nếu vi phạm yêu cầu nhập lại. Sau đó tính tổng, tích của 3 phân số với kết quả tối giản.
13. Tìm m số nguyên tố cùng nhau với số n, với n, m > 0 được nhập từ bàn phím. Biết rằng 2 số nguyên tố cùng nhau khi ước chung lớn nhất của chúng là 1.
14. Viết chương trình xuất ra các số có 3 chữ số sao cho các chữ số khác nhau đôi một.
15. Xuất các số có 5 chữ số sao cho có đúng 2 chữ số bằng nhau.
16. Xuất các số < 1000 sao cho tổng các chữ số bằng tích các chữ số.
17. Viết chương nhập số nguyên lớn N (khai báo: long N) có k chữ số.
- Tìm chữ số hàng đầu tiên của N.
  - Tính tổng các chữ số của N.
  - Tìm chữ số lớn nhất trong k chữ số đó.
  - Đếm số chữ số của N.
  - N có phải là số có các chữ số khác nhau đôi một không?

Gợi ý: Áp dụng các phép toán / và % cho 10 để lấy lần lượt từng chữ số của N theo chiều từ chữ số hàng đơn vị đến các chữ số hàng chục, hàng trăm,...

Ví dụ: N = 347285108, x = N%10=8, y = N/10 = 34728510. Lặp lại 2 phép toán chia này ta lần lượt lấy được từng chữ số của N từ hàng đơn vị.

Khi đó N có:

- Chữ số hàng đầu tiên là 3.
- Tổng các chữ số: 3+4+7+2+8+5+1+0+8 = 38
- Chữ số lớn nhất là 8.

- Số chữ số của N: 9
  - N không phải số có các chữ số khác nhau đôi một vì có 2 chữ số 8.
  - N có bao nhiêu chữ số chẵn và bao nhiêu chữ số
18. Viết chương trình thực hiện thực hiện các yêu cầu sau:
- Nhập vào 3 giá trị ngày, tháng, năm.
  - Kiểm tra ngày, tháng, năm có tạo 1 ngày hợp lệ theo lịch dương.
  - Nếu ngày hợp lệ thì tính ngày tiếp theo và ngày trước đó.
- Ví dụ: 2/4/2012 → hợp lệ → ngày tiếp theo: 3/4/2012, ngày trước đó: 1/4/2012
- 30/2/2011, 31/6/2010 → không hợp lệ
- 31/12/2011 → hợp lệ → ngày tiếp theo: 1/1/2012, ngày trước đó: 29/12/2011
19. Viết chương trình tính tiền lương ngày cho công nhân, nhập giờ vào ca, giờ ra ca của mỗi người.
- Biết :
- Tiền trả cho mỗi giờ trước 12 giờ là 6000đ và sau 12 giờ là 7500đ.
  - Giờ vào ca sớm nhất là 6 giờ sáng và giờ ra ca trễ nhất là 18 giờ (Giả sử
- giờ nhập vào nguyên).
20. Nhập N ( $N > 100.000$ ). Kiểm tra nếu nhập sai thì yêu cầu nhập lại. Có 3 loại tiền 5000, 10.000 và 20.000. Xuất các cách trả N tiền từ 3 loại tiền trên.
21. Nhập vào giờ, phút, giây. Kiểm tra giờ, phút giây có hợp lệ không? Nếu hợp lệ thì cho biết giờ sau đó 1 giây và trước đó 1 giây là bao nhiêu.
22. Viết chương trình nhập số nguyên dương n gồm 5 chữ số, kiểm tra xem các chữ số n có phải là số đối xứng hay không.
- Ví dụ: Đối xứng: 13531, Không đối xứng: 13921

## CHƯƠNG 5. MẢNG VÀ CON TRỎ

### 5.1 Mảng một chiều

#### 5.1.1 Khái niệm và khai báo mảng một chiều

Mảng một chiều là một nhóm các phần tử có cùng kích thước, cùng kiểu dữ liệu. Những phần tử này được lưu liên tiếp với nhau trong bộ nhớ. Số phần tử của mảng gọi là kích thước của mảng.

Cú pháp:

**<Tên kiểu dữ liệu> <Tên mảng> [ <số phần tử>];**

Trong đó:

- Tên kiểu dữ liệu: là kiểu dữ liệu mà mỗi phần tử mảng có dữ liệu thuộc vào.
- Tên mảng: là tên được đặt theo qui tắc đặt tên của danh hiệu trong ngôn ngữ lập trình C, còn mang ý nghĩa là tên biến mảng.
- Số phần tử: là một hằng số nguyên, cho biết số lượng phần tử tối đa trong mảng là bao nhiêu.

#### **Ví dụ 5.1:**

```
//Khai báo mảng một chiều tên a có 20 phần tử kiểu  
số nguyên int.
```

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

```
//Khai báo mảng một chiều tên b có 10 phần tử kiểu  
ký tự char.
```

```
char b[10];
```

Mỗi phần tử của mảng một chiều được truy nhập giá trị thông qua chỉ số (index) của nó. Chỉ số để xác định phần tử nằm ở vị trí nào trong mảng. Phần tử đầu tiên của mảng có chỉ số là 0, thành phần thứ hai có chỉ số là 1,... và tương tự tăng dần cho hết mảng.

**Ví dụ 5.2:**

```
int num[5];
```

Chỉ số và giá trị phần tử của mảng một chiều num được biểu diễn như sau:

Chỉ số mảng	0	1	2	3	4
Giá trị phần tử trong mảng	num[0]	num[1]	num[2]	num[3]	num[4]

Ở ví dụ 5.2, mảng có năm phần tử và chỉ số của mảng bắt đầu từ 0 cho nên chỉ số để truy xuất phần tử cuối cùng của mảng là 4. Như vậy, nếu một mảng có n phần tử thì chỉ số cuối cùng của mảng là n-1.

Chỉ số của mảng có thể là một giá trị cụ thể, giá trị của một biến hay giá trị được tính toán từ một biểu thức đại số.

**Ví dụ 5.3:**

```
int i = 3;
```

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

```
a[1] /* truy cập phần tử thứ 2 của mảng a, vì phần tử thứ 1 có chỉ số là 0 */
```

```
a[i] // truy cập phần tử thứ 4 của mảng a
```

```
a[i*2 - 1]// truy cập phần tử thứ 6
```

**5.1.2 Gán giá trị vào các phần tử của mảng**

Thông qua chỉ số phần tử của mảng, chúng ta cũng có thể thay đổi giá trị của các phần tử trong mảng.

**Ví dụ 5.4:**

Cho mảng num: `int num[5];`

Để gán 10 vào phần tử thứ 3 của mảng num, ta viết:

```
num[2] = 10;
```

Trong lập trình, câu lệnh này có nghĩa là giá trị biểu thức bên phải được gán vào biểu thức bên trái. Do đó, phần tử thứ 3 của mảng num sẽ chứa giá trị 10 sau khi thực hiện câu lệnh trên.

Chỉ số mảng	0	1	2	3	4
Giá trị phần tử trong mảng	num[0]	num[1]	10	num[3]	num[4]

Để chứa các kí tự chữ 'i', 'o', 'g', 'a', 'c' – chúng ta có thể khai báo mảng ký tự ch như sau:

```
char ch[5];
ch[0] = 'i';
ch[1] = 'o';
ch[2] = 'g';
ch[3] = 'a';
ch[4] = 'c';
```

Chỉ số mảng	0	1	2	3	4
Giá trị phần tử trong mảng	i	o	g	a	c

### 5.1.3 Lấy giá trị các phần tử trong mảng

- Khi muốn lấy giá trị một phần tử trong mảng tại vị trí chỉ số phần tử có cú pháp như sau:

**<tên mảng>[<chỉ số phần tử>]**

- Chẳng hạn muốn truy xuất phần tử thứ i trong mảng a, ta ghi là a[i].
- Khi khai báo một biến để nhận giá trị của mảng thì biến này phải có cùng kiểu dữ liệu với phần tử của mảng.

#### **Ví dụ 5.5:**

Giả sử có mảng một chiều num có năm phần tử là số nguyên như sau:

Chỉ số mảng	0	1	2	3	4
Giá trị phần tử trong mảng	20	30	10	4	6

Với đoạn lệnh sau:

```
int i;
i = num[4]; // i nhận giá trị 6
```

Lệnh trên sẽ lấy giá trị 6 lưu tại phần tử thứ 5 của mảng, và giá trị này sẽ được gán vào biến i.



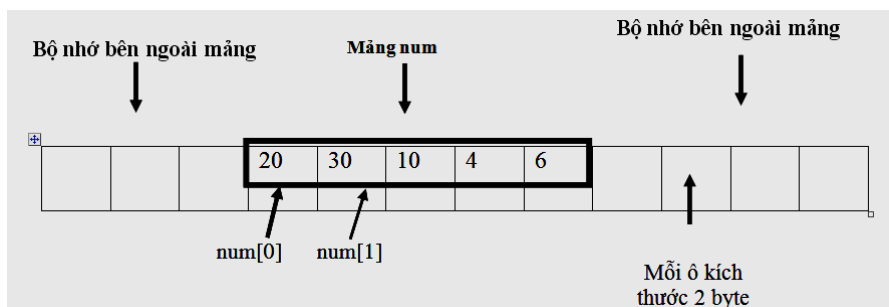
### 5.1.4 Các phần tử của mảng trong bộ nhớ

Bộ nhớ của máy tính được sắp xếp theo từng byte.

Khi khai báo `int diem;` thì `diem` là một biến có kiểu `int`. Kích thước lưu trữ của `int` trong ngôn ngữ lập trình C thường là 2 bytes. Vì thế, 2 bytes này được cố định trong bộ nhớ và được tham chiếu bằng tên `diem`.

Như vậy, khi ta khai báo `int num[5];` thì `num` là mảng một chiều có 5 phần tử số nguyên, mỗi số nguyên cần 2 bytes. Do đó, bộ nhớ cần cho mảng `num` là:  $5 \times 2 = 10$  bytes, và những phần tử của mảng này được chứa trong vùng nhớ liên tục.

Xét mảng `num` có năm phần tử ở ví dụ 5.5, ta có mô phỏng cách lưu trữ dữ liệu như hình 5.1:



Hình 5. 1. Mô phỏng cách lưu trữ dữ liệu của các phần tử mảng trong bộ nhớ.

### 5.1.5 Khởi tạo mảng

Trong ngôn ngữ lập trình C, có thể khởi tạo giá trị các phần tử mảng ngay khi mảng được khai báo. Việc khởi tạo cho mảng được thực hiện lúc khai báo mảng bằng một loạt giá trị hằng khởi động cho các phần tử của mảng. Các giá trị hằng được đặt giữa một cặp ngoặc nhọn `{ }`, các phần tử cách nhau bằng dấu phẩy `(,)`.

**Ví dụ 5.6:** Khởi tạo mảng một chiều a chứa số nguyên có 10 phần tử với các giá trị 5, 15, 20, 25, 30 như sau:

```
int a[10] = {5, 15, 20, 25, 30};
```

Trong việc khởi tạo mảng, kích thước của mảng không cần xác định, chương trình C sẽ đếm số phần tử được khởi động và lấy đó làm kích thước. Nếu có xác định kích thước, thì số giá trị được khởi tạo liệt kê phải không được lớn hơn kích thước đã khai báo

**Ví dụ 5.7:** Ta khai báo:

```
double b[ ] = {4.5, 7.5, 8.2, 4.32};
```

thì mảng b sẽ được hiểu là có 4 phần tử kiểu double.

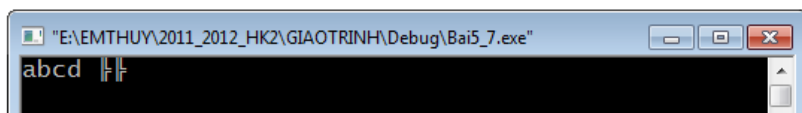
Nếu khai báo `int a[10] = {2, 5, 6, 1, 0, 6, 7};` thì một mảng gồm 10 phần tử được tạo ra. Trong đó có 7 phần tử đầu tiên được khởi tạo giá trị, các phần tử còn lại chưa được khởi tạo giá trị.

### Lưu ý:

Trình biên dịch C không báo lỗi khi có sự vượt quá giới hạn của mảng. Chẳng hạn đoạn chương trình sau đây vẫn chấp nhận trong khi có sự vượt qua giới hạn của mảng buf:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i;
    char buf[5] = {'a', 'b', 'c', 'd'};
    for(i = 0; i<= 10; i++)
        printf("%c", buf[i]);
    getch();
}
```

### Kết quả thực thi của chương trình:



## 5.2 Mảng hai chiều

### 5.2.1 Khái niệm

Mảng hai chiều  $m$  dòng  $n$  cột được xem như là một bảng hình chữ nhật chứa  $m*n$  phần tử cùng kiểu dữ liệu (còn gọi là ma trận  $m*n$ ). Nói cách khác, mảng hai chiều là mảng một chiều mà mỗi phần tử của nó là một mảng một chiều.

Cú pháp:

**<tên kiểu dữ liệu> <tên mảng> [<số dòng>] [<số cột>;**

Chẳng hạn khai báo mảng hai chiều  $a$  có 5 dòng 3 cột chứa các số nguyên, ta khai báo là: `int a[5][3];`

### 5.2.2 Chỉ số của mảng

Ta xét bảng điểm 3 môn học của các sinh viên như sau:

	Toán	Lý	Hóa
<b>Minh</b>	7	8	10
<b>Lan</b>	4	6	8
<b>Nhật</b>	9	10	10
<b>Ngọc</b>	3	5	7

Trong bảng thông tin trên, tiêu đề dòng chứa tên sinh viên và tiêu đề cột chứa môn học. Chúng ta lưu trữ điểm 3 môn học của mỗi sinh viên. Để đọc từng thông tin riêng biệt, cần xác định vị trí dòng, cột và đọc thông tin tại vị trí đó. Xét từ bảng điểm trên, tìm điểm môn toán của Lan như sau:

Tại dòng thứ hai của bảng chứa các điểm môn học của Lan. Cột thứ hai của dòng này chứa điểm môn Toán. Vì thế, điểm toán của Lan là 4. Do đó, ứng với cấu trúc loại này chúng ta có thể sử dụng một mảng hai chiều để lưu trữ.

Trong mảng 2 chiều, cách xác định chỉ số cũng như mảng một chiều. Đó là chỉ số dòng cột bắt đầu từ 0.

Chúng ta có thể khai báo mảng hai chiều điểm để lưu trữ bảng thông tin về điểm các sinh viên như sau: `int diem[4][3];`

	Cột 0	Cột 1	Cột 2
Dòng 0	diem[0][0]=7	diem[0][1]=8	diem[0][2]=10
Dòng 1	diem[1][0]=4	diem[1][1]=6	diem[1][2]=8
Dòng 2	diem[2][0]=9	diem[2][1]=10	diem[2][2]=10
Dòng 3	diem[3][0]=3	diem[3][1]=5	diem[3][2]=7

Trong mảng hai chiều diem trên, mảng có 4 dòng, 3 cột. Chỉ số dòng của mảng từ 0 đến 3, chỉ số cột của mảng từ 0 đến 2.

### 5.2.3 Truy xuất phần tử mảng hai chiều

Mỗi phần tử mảng hai chiều được truy xuất thông qua chỉ số dòng, chỉ số cột của mảng hai chiều.

Cú pháp:

**<tênmảng>[<chỉ số dòng phần tử>][<chỉ số cột phần tử>]**

Chẳng hạn muốn truy xuất một phần tử tại dòng thứ i, cột thứ j của mảng a, ta ghi là a[i][j]. Giá trị i, j được tính từ 0.

**Ví dụ 5.8:** Truy xuất phần tử dòng 2 cột 1 của mảng diem như sau: diem[1][0]

### 5.2.4 Khởi tạo mảng hai chiều

Khởi tạo giá trị cho mảng hai chiều là gán giá trị cho từng phần tử trong mảng hai chiều. Ngoài ra, có thể khởi tạo giá trị cụ thể cho các phần tử của mảng hai chiều trong khi khai báo mảng hai chiều.

**Ví dụ 5.9:**

```
int matrix[4][4] = {    {11, 12, 13, 14},
                        {21, 15, 41, 16},
                        {43, 58, 24, 91},
                        {32, 15, 25, 16}};

char table[7][1] = {"MON", "TUE", "WED", "THU", "FRI",
                   "SAT", "SUN"};
```

### 5.3 Con trỏ (Pointer)

Chúng ta đã biết các biến được chứa trong bộ nhớ. Mỗi vị trí các biến được chứa trong bộ nhớ thì được gán cho một con số duy nhất gọi là địa chỉ (address). Thông qua địa chỉ, chúng ta có thể biết được biến đó lưu trữ ở đâu trong bộ nhớ. Tương tự như vậy mỗi phần tử của mảng đều có một địa chỉ riêng. Con trỏ là một dạng biến để chứa loại địa chỉ này.

#### 5.3.1 Khái niệm

Con trỏ là một kiểu dữ liệu đặc biệt dùng để quản lý địa chỉ của các ô nhớ. Một con trỏ quản lý các địa chỉ mà dữ liệu tại các địa chỉ này có kiểu T thì con trỏ đó được gọi là con trỏ kiểu T. Con trỏ kiểu T chỉ được dùng để chứa địa chỉ của biến kiểu T. Nghĩa là con trỏ kiểu int chỉ được dùng để chứa địa chỉ biến kiểu int, con trỏ kiểu char chỉ được dùng để chứa địa chỉ biến kiểu char.

Con trỏ là một phần cơ bản quan trọng của ngôn ngữ lập trình C. Nó là cách duy nhất để thể hiện một số thao tác truy xuất và dữ liệu. Nó tạo ra mã code đơn giản, hiệu quả, là một công cụ thực thi mạnh mẽ.

#### 5.3.2 Khai báo biến con trỏ

Cú pháp:

<b>&lt;tên kiểu dữ liệu&gt; *&lt;tên biến con trỏ&gt;</b>
---

**Ví dụ 5.10:**

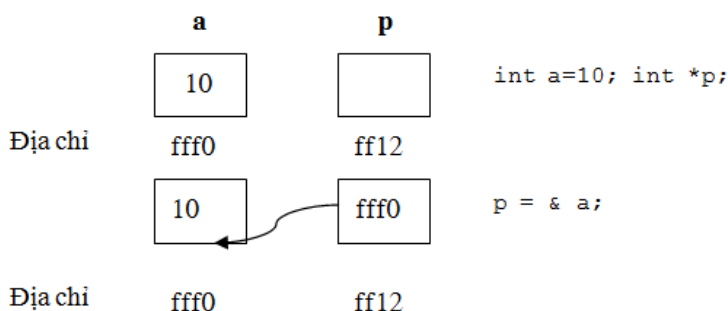
```
// x là biến kiểu int, còn px là con trỏ kiểu int.  
int x, *px;
```

px được khai báo là một con trỏ kiểu int, nó chứa địa chỉ của biến kiểu dữ liệu số nguyên. Dấu \* không phải là một phần của biến, int \* có nghĩa là con trỏ kiểu int.

Đặt tên biến con trỏ giống như tên của các biến khác. Để gán địa chỉ vào con trỏ chúng ta cần phải gán giá trị cho biến như sau:

**Ví dụ 5.11:**

```
int a = 10;
int *p;
p = &a; // giá trị p chứa địa chỉ của biến a
```



Hình 5. 2. Hình minh họa địa chỉ con trỏ p trong ví dụ 5.11

Ví dụ trên được hiểu như sau:

- a là biến kiểu int được khởi tạo bằng 10.
- p là biến con trỏ kiểu int, chứa địa chỉ của kiểu dữ liệu int, lúc này nó không chứa giá trị (hay chứa giá trị NULL).
- Câu lệnh `p = &a` có nghĩa là “gán địa chỉ của a vào p”. Biến con trỏ này bây giờ chứa địa chỉ của biến a.
- Giả sử địa chỉ của biến a và p trong bộ nhớ là fff0 và ff12. Câu lệnh `p = &a` để gán địa chỉ của a vào p. Dấu ‘&’ viết phía trước biến a được gọi là phép toán địa chỉ (address). Vì thế biến con trỏ này chứa giá trị fff0.

Mặc dù chúng ta khai báo biến con trỏ với dấu ‘\*’ ở phía trước, nhưng bộ nhớ chỉ gán cho p chứ không phải \*p.

### 5.3.3 Toán tử địa chỉ (&) và toán tử nội dung (\*)

#### – Toán tử địa chỉ (&)

Biến được khai báo là x thì **&x** là địa chỉ của x.

Kết quả của phép lấy địa chỉ (&) là một con trỏ, do đó có thể dùng để gán cho một biến pointer.

**Ví dụ 5.12:**

**a) `int *px, num;`**

// px là một pointer chỉ đến biến kiểu int là num.

px=&num;

//xuất địa chỉ của biến num dạng số hệ 16 (hệ hexa)

printf("%x", &num);

**b) `int *px, num;`**

px = &(num +4); // SAI vì ( num+4) không phải là một biến cụ thể

**Lưu ý:** Chúng ta thấy cú pháp lệnh nhập dữ liệu scanf (lệnh đã được học ở chương 2) trong ngôn ngữ lập trình C luôn có dấu & trước biến cần nhập. Điều này xác định cần đưa dữ liệu vào con trỏ chứa địa chỉ của biến tương ứng.

– **Toán tử nội dung (\*)**

Toán tử lấy nội dung của một địa chỉ được kí hiệu là dấu \* trước một pointer, dùng để lấy giá trị của biến mà con trỏ đang trỏ đến.

Xét lại ví dụ 5.12, ta có:

px là một pointer chỉ đến biến num như ví dụ 5.12 a, thì \* px là giá trị của biến num.

**Ví dụ 5.13:**

a) //num là biến được khai báo và gán giá trị là 10.

int num = 10 ;

int \*px; // px là một con trỏ chỉ đến kiểu int

px= &num ; //px là địa chỉ của biến num.

/\*giá trị của \*px (tức là num) cộng thêm 3, gán cho k. Sau đó \*px thực hiện lệnh tăng 1 đơn vị (++)\*/

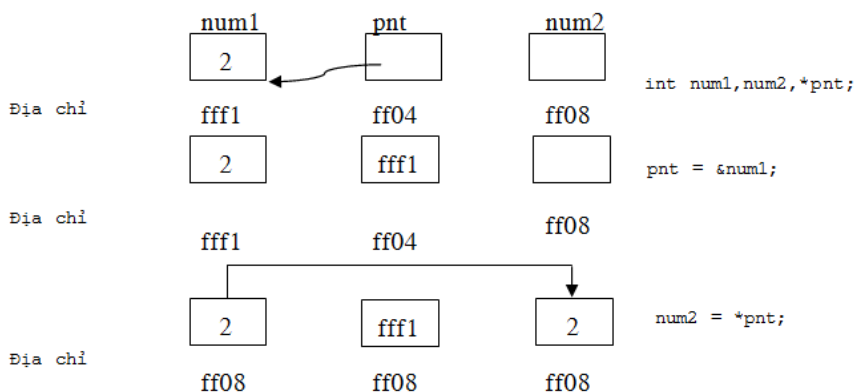
int k = (\* px)++ + 3 ;

// Sau câu lệnh trên num = 11, k = 13

```
b) int num1 = 2, num2, *pnt;
   pnt = &num1
   num2 = *pnt;
```

Trong ví dụ trên, biến `num1` được gán bằng 2. Dòng `pnt = &num1` nghĩa là biến con trỏ `pnt` chứa địa chỉ của biến `num1`. Phép gán `num2 = *pnt`, dấu `*` được đặt ở phía trước biến con trỏ, thì giá trị trả về của biến này là giá trị của biến được trỏ tới bởi con trỏ `pnt`. Do đó, `num2` có giá trị là 2.

Ta minh họa qua hình 5.3



Hình 5. 3. Mô phỏng quá trình thực thi của các con trỏ trong ví dụ 5.13

Lưu ý: “NULL” là hằng khi pointer mang ý nghĩa không chứa một địa chỉ nào cả. Ta gọi là pointer rỗng.

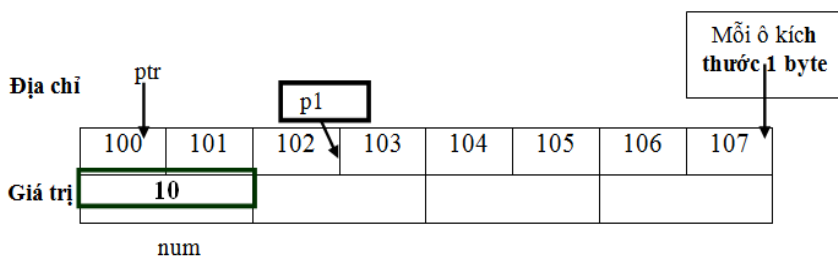
### 5.3.4 Tính toán trên Pointer

Một biến pointer có thể được cộng trừ với một số nguyên (int, long) để cho kết quả là một pointer chỉ đến một vùng nhớ khác.

**Ví dụ 5.14:**

```
int *ptr, *p1;
int num = 10;
ptr = &num;
p1 = ptr + 2;
```





Hình 5. 4. Mô phỏng quá trình thực thi của các con trỏ trong ví dụ 5.14

Việc cộng hoặc trừ pointer với một số nguyên n thì pointer sẽ chỉ đến một địa chỉ mới hay nói cách khác là chỉ đến một biến khác nằm cách biến đó n vị trí.

#### Ví dụ 5.15:

```
int v[10]; // mảng 10 phần tử lin tiếp.

int * p ; // Biến pointer chỉ đến một số int.

p = & v[0]; // p là địa chỉ phần tử đầu tiên của
mảng
for( i =0; i<10 ; i++)
{
    *p= i * i; // gán cho phần tử mà p đang chỉ
    đến
    p ++ ;// p được tăng lên để chỉ đến phần tử
    kế tiếp
}
```

#### Lưu ý:

- Cộng một pointer với một giá trị nguyên cho ta một pointer
- Phép cộng 2 pointer là không hợp lệ.
- Trừ hai pointer cũng là hợp lệ, kết quả cho ta một giá trị int biểu thị khoảng cách (số phần tử) giữa 2 pointer đó.

- Không thể nhân, chia, hoặc lấy dư của một pointer với bất kì một số nào.
- Đối với các phép toán khác, pointer được xử lý như một biến bình thường (gán, so sánh,...), các toán hạng phải cùng kiểu pointer và cùng kiểu đối tượng của chúng. Mỗi sự chuyển kiểu tự động luôn được cân nhắc và xác nhận từ trình biên dịch.

Địa chỉ của một biến được xem là một pointer hằng và ngôn ngữ lập trình C cho phép thực hiện các phép toán mà pointer chấp nhận trên nó, trừ phép gán lại và phép tăng giảm vì ta không thể gán lại một giá trị hằng bằng một giá trị khác được.

**Ví dụ 5.16:**

```
int p,*px,num;
px= &num ;//lấy địa chỉ của num gán vào biến:
ĐÚNG
px++; //tăng giảm trên một biến pointer: ĐÚNG
&p =px;//gán lại một địa chỉ hằng: SAI
&p++; //tăng giảm một địa chỉ hằng: SAI.
```

### 5.3.5 Truyền tham số địa chỉ

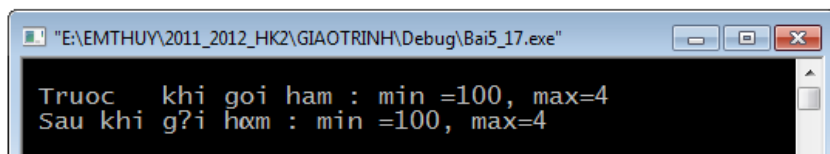
**Ví dụ 5.17:**

```
void swap( int x, int y)
{
    int tmp = x;
    x=y;
    y=tmp;
}

void main()
{
    int  min =100,max = 4;
    printf("\n Truoc    khi goi ham:");
```

```
printf(" min =%d, max=%d ",min,max);  
swap ( min,max);  
printf("\n Sau khi gọi hàm:");  
printf(" min =%d, max=%d ",min,max);  
}
```

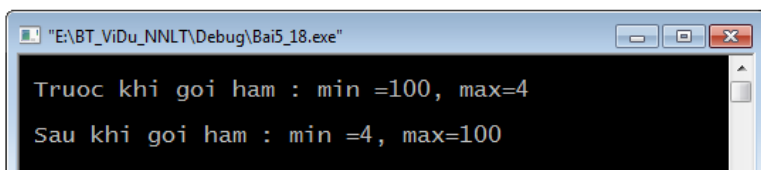
### Kết quả thực thi chương trình:



Sau khi gọi hàm ta không đạt được yêu cầu là hoán đổi giá trị min, max vì giá trị của biến không thay đổi khi ra khỏi hàm. Do đó ta phải dùng đến con trỏ.

### Ví dụ 5.18:

```
void swap ( int* px, int* py)  
{  
    int tmp = *px;  
    *px=*py;  
    *py=tmp;  
}  
  
void main()  
{  
    int min =100,max = 4;  
    printf("\n Truoc khi gọi ham:");  
    printf(" min=%d, max=%d", min,max);  
    swap ( &min, &max);  
    printf("\n Sau khi gọi ham:");  
    printf("min =%d, max=%d \n", min,max);  
}
```

**Kết quả thực thi chương trình:**

```
"E:\BT_ViDu_NNLT\Debug\Bai5_18.exe"
Truoc khi goi ham : min =100, max=4
Sau khi goi ham : min =4, max=100
```

Trong trường hợp này, do các pointer thực sự chỉ đến các biến min, max nên việc hoán đổi đối tượng các pointer này thực sự làm hoán đổi giá trị của 2 biến min, max ở hàm main(). Cách truyền tham số theo địa chỉ vào hàm khi ta muốn hàm đó có thể thay đổi giá trị của tham số mà chúng ta truyền vào.

**5.4 Cấp phát và giải phóng vùng nhớ cho biến con trỏ****5.4.1 Cấp phát vùng nhớ cho biến con trỏ**

Trước khi sử dụng biến con trỏ, ta nên cấp phát vùng nhớ cho biến con trỏ này quản lý địa chỉ. Việc cấp phát được thực hiện nhờ các hàm malloc(), calloc() trong thư viện alloc.h.

Cú pháp các hàm:

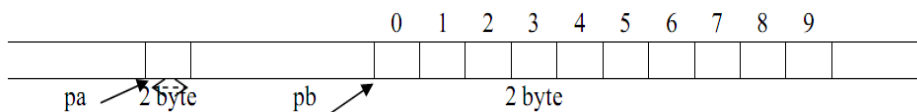
<b>void *malloc(size_t size)// Cấp phát vùng nhớ có kích thước size.</b>
--

<b>void *calloc(size_t nitems, size_t size)//Cấp phát vùng nhớ có kích thước là nitems*size.</b>
--

**Ví dụ 5.18:** Giả sử ta có khai báo:

```
int a, *pa, *pb;
pa = (int*)malloc(sizeof(int)); /* Cấp phát
vùng nhớ có kích thước bằng với kích thước của một
số nguyên */
pb= (int*)calloc(10, sizeof(int)); /* Cấp
phát vùng nhớ có thể chứa được 10 số nguyên*/
```

Hình ảnh minh họa trong bộ nhớ như sau:



Hình 5. 5. Sơ đồ lưu trữ con trỏ trong bộ nhớ

**Lưu ý:** Khi sử dụng hàm malloc() hay calloc(), phải ép kiểu vì nguyên mẫu các hàm này trả về con trỏ kiểu void.

### Cấp phát lại vùng nhớ cho biến con trỏ

Trong quá trình thao tác trên biến con trỏ, nếu cần cấp phát thêm vùng nhớ có kích thước lớn hơn vùng nhớ đã cấp phát, ta sử dụng hàm realloc().

Cú pháp:

**void \*realloc(void \*block, size\_t size)**

Ý nghĩa:

- Cấp phát lại một vùng nhớ cho con trỏ block quản lý, vùng nhớ này có kích thước mới là size; khi cấp phát lại thì nội dung của vùng nhớ trước đó vẫn tồn tại.
- Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới. Địa chỉ này có thể khác với địa chỉ được chỉ ra khi cấp phát ban đầu.

**Ví dụ 5.19:** Cấp phát lại vùng nhớ do con trỏ pa quản lý

```
int a, *pa;
/*Cấp phát vùng nhớ có kích thước 2 byte*/
pa=(int*)malloc(sizeof(int));
/* Cấp phát lại vùng nhớ có kích thước 6 byte*/
pa = realloc(pa, 6);
```

### 5.4.2 Giải phóng vùng nhớ cho biến con trỏ

Một vùng nhớ đã cấp phát cho biến con trỏ, khi không còn sử dụng nữa, ta sẽ thu hồi lại vùng nhớ này nhờ hàm free().

Cú pháp:

<b>void free(void *block)</b>
-------------------------------

**Ý nghĩa:** Giải phóng vùng nhớ được quản lý bởi con trỏ block.

**Ví dụ 5.20:** Xét lại ví dụ 5.19, sau khi thực hiện xong, ta giải phóng vùng nhớ cho 2 biến con trỏ pa và pb:

```
free(pa);
```

```
free(pb);
```

### 5.5 Sự liên hệ giữa cách sử dụng mảng và pointer

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác lập qua biến con trỏ.

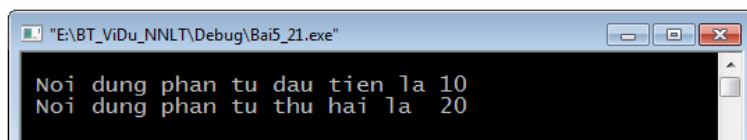
#### 5.5.1 Khai thác một pointer theo cách của mảng

Khi khai báo `int a[5];` ngoài cách tiếp cận truy xuất thông thường đã học, mảng còn được truy xuất, xử lý bằng pointer. Khi đó, sự truy xuất đến riêng tên `a` được hiểu là truy xuất đến địa chỉ của phần tử đầu tiên của mảng `a`.

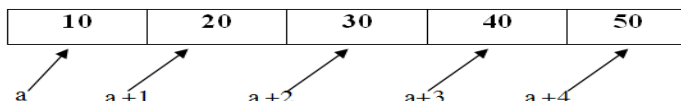
**Ví dụ 5.21:**

```
#include "stdio.h"
#include "conio.h"
void main()
{
    int a[10]={10,20,30,40,50};
    printf("Noi dung phan tu dau tien\n", *a);
    printf("Noi dung phan tu thu hai la %d", *(a+1));
}
```

## Kết quả thực thi của chương trình



Ta có thể thấy rõ hơn qua hình sau :



Hình 5. 6. Mô phỏng việc lưu trữ dữ liệu cho mảng *a* trong ví dụ 5.21

Với khai báo `int *pnum;` muốn *pnum* trỏ đến phần tử đầu tiên của mảng *a* thì viết là `pnum = a;` thay cho cách viết `pnum = &a[0];`

Vì bản thân tên mảng *a* lại có thể hiểu là một địa chỉ (hoặc một pointer) nên ta có sự tương đương ý nghĩa như sau:

<code>*a</code>	tương đương với	<code>a[0]</code>
<code>*(a+1)</code>	tương đương với	<code>a[1]</code>
...	...	...
<code>*(a+i)</code>	tương đương với	<code>a[i]</code>
<code>a</code>	tương đương với	<code>&amp;a[0]</code>
<code>a+1</code>	tương đương với	<code>&amp;a[1]</code>
...	...	-
<code>a+i</code>	tương đương với	<code>&amp;a[i]</code>

### 5.5.2 Khai thác một mảng bằng pointer

**Ví dụ 5.22:**

```
int num [10];
int *pnum;
```

*/\*phép gán kể từ lúc này pnum sẽ chỉ về phần tử thứ 1 của mảng num\*/*

```
pnum = &num[0];  
/* gán giá trị phần tử thứ 1 của mảng num  
(num[0]) cho x.*/  
x = *pnum;
```

Như vậy ta có thể hoàn toàn truy xuất đến mỗi phần tử của một mảng bằng cách sử dụng một pointer chỉ đến đầu mảng.

### 5.5.3 Những điểm khác nhau giữa mảng và con trỏ

- Khi khai báo và định nghĩa mảng, vị trí của mảng đó được cấp rõ ràng và đủ theo kích thước được khai báo. Còn pointer thì vị trí được cấp chỉ là một chỗ cho bản thân của pointer còn vị trí mà pointer chỉ đến thì không được chuẩn bị sẵn.
- Pointer thực sự là một biến, ta có thể tăng giảm và gán lại trên biến pointer đó các địa chỉ khác nhau. Còn tên mảng lại là một địa chỉ hằng chỉ đến vùng mảng cố định, ta có thể sử dụng chứ không thể tăng giảm hoặc gán lại nó.
- Ta có thể lấy địa chỉ của một pointer trỏ đến, địa chỉ của một phần tử mảng chứ không thể lấy địa chỉ của một mảng.
- Pointer có thể thay đổi địa chỉ trỏ tới còn tên mảng thì không thể.

#### Ví dụ 5.23:

```
float readings[20], totals[20];  
float *fptr;  
Các lệnh sau là hợp lệ  
// fptr chỉ tới phần tử đầu mảng readings  
fptr = readings  
// fptr chỉ tới phần tử đầu mảng totals  
fptr = totals;  
Các lệnh sau là bất hợp lệ  
readings = total;  
totals = fptr;
```



### 5.5.4 Hàm có đối số là mảng

Cách khai báo tham số trong hàm khi tham số hàm là mảng.

**Ví dụ 5.24:**

Xét ví dụ sau có hàm con là hàm đảo chuỗi.

```
char str[] = "ABCDEF";
void daoChuoi(char*s);
void main()
{
    printf("Trước khi đảo chuỗi : %s \n",str);
    daoChuoi(str);
    printf("Sau khi đảo chuỗi : %s \n",str);
}
void daoChuoi (char*s)
{
    ...
}
```

Ta có thể thấy rằng, trong hàm main() khi gọi hàm daoChuoi() ta chỉ truyền đối số là tên mảng *str* mà thôi, điều đó có ý nghĩa là ngôn ngữ lập trình C chỉ gửi địa chỉ của mảng, nhưng địa chỉ này chỉ là một bản sao và thực chất là một biến pointer chỉ đến phần tử đầu tiên của mảng được gọi. Do đó, có thể nói rằng hàm có đối số là mảng không nhận được một mảng, nó chỉ nhận một pointer chỉ đến phần tử đầu mảng đó. Điều này có nghĩa là khai báo đối số của một hàm là pointer hay là một mảng đều như nhau. Và nếu chúng ta khai báo là mảng một chiều, thì không cần xác định kích thước vì C không quan tâm đến điều đó. Ví dụ hàm daoChuoi() có thể khai báo lại như sau:

```
void daoChuoi(char s[])
{
    ...
}
```

Hơn nữa, vì đối số này thực sự là biến pointer, nên ta có thể xử lý giống như biến pointer. Ta xét một cách viết khác của hàm `strlen()` (hàm xác định chiều dài chuỗi) như sau:

```
int strlen(char *s) // hay có thể viết là char
s[]
{
    int i;
    for(i = 0; *s != '\0'; i++, s++);
    return i;
}
```

Cũng vì nguyên nhân trên, nội dung của các mảng khi truyền vào trong một hàm cũng có thể bị thay đổi. Muốn tránh được tình trạng đó, nên khai báo các mảng đối số của hàm đó là các pointer chỉ đến `const`. Khi đó nếu hàm này thay đổi giá trị nội dung của mảng truyền vào, chúng ta sẽ nhận được thông báo của chương trình biên dịch.

Chẳng hạn ta có thể khai báo: `int strlen(const char*s);`

Khi đó, mảng `s` gửi vào hàm `strlen` sẽ không bị thay đổi vì nếu có sự thay đổi chương trình biên dịch sẽ bắt lỗi.

### 5.5.5 Hàm trả về pointer và mảng

Hàm trả về một mảng hay hàm trả về một pointer, cách định nghĩa giống nhau

Cú pháp:

**<Tên kiểu dữ liệu> \* <Tên hàm>(<danh sách tham số>)**

Trong đó, tên kiểu xác định kiểu của biến mà pointer được trả về. Kiểu này có thể là một kiểu dữ liệu nào đó đã định nghĩa trước.

Điều quan trọng ở đây là mảng được trả về (hoặc biến mà pointer trả về này chỉ đến) phải có “thời gian tồn tại” cho đến lúc ra khỏi hàm này. Vì nếu đối tượng của một pointer không còn tồn tại thì việc trả về bản thân pointer không còn ý nghĩa gì cả.

**Ví dụ 5.25:** Viết một hàm nhận một mảng *so* từ bàn phím rồi trả về mảng đó :

```
int*input()
{
    int daySo[10];
    printf("hay nhap vao 10 so : \n");
    for(i = 0; i < 10;i++)
    {
        printf("So thu %d",i);
        scanf("%d",& daySo [i]);
    }
}
```

Trong hàm trên, sẽ không sử dụng được vì có thể địa chỉ của mảng *daySo* vẫn được trả về nhưng đối tượng của địa chỉ đó thì không còn ý nghĩa do “thời gian tồn tại” của mảng *daySo* này chỉ là ở trong hàm *input()*. Vì vậy, muốn sử dụng hàm này thì mảng *daySo* phải là biến ngoài hàm *input*, mảng static (mảng tĩnh), hoặc là mảng của hàm khác gửi đến cho hàm *input()*. Chẳng hạn, có thể hiệu chỉnh lại hàm *input()* như sau:

```
int * input(int* daySo) //hay int daySo[]
{
    printf("hay nhap vao 10 so : \n");
    for(i = 0; i < 10;i++)
    {
        printf("So thu %d",i);
        scanf("%d",& daySo[i]);
    }
    return (daySo);
}
```

### 5.5.6 Mảng các con trỏ hoặc con trỏ của con trỏ

Cú pháp:

**<tên kiểu dữ liệu>\* <tên mảng> [ kích thước ];**

**Ví dụ 5.26:**

//khai báo mảng a gồm 10 pointer chỉ đến kiểu char

```
char *a[10];
```

**Ví dụ 5.27:** Xét đoạn code sau:

```
// Hàm sắp xếp các phần tử
void Sort(int *a[], int n)
{
    int i, j, *tmp;
    for(i = 0; i < n -1; i++)
        for(j = i +1; j < n; j++)
            if(*a[i] > *a[j])
            {
                tmp = a[i];
                a[i] = a[j];
                a[j] = tmp;
            }
}

void main()
{
    int d = 10, e = 3, f = 7;
    int a = 12, b = 2, c = 6;
    int * ma[6];
    ma[0] = &a;          ma[3] = &d;
    ma[1] = &b;          ma[4] = &e;
    ma[2] = &c;          ma[5] = &f;
```

```
Sort(ma, 6); // sắp xếp lại thứ tự mảng
for(i = 0; i < 6; i++)
    printf("%d\n", *ma[i]);
}
```

Hàm `Sort()` sắp xếp lại các địa chỉ trong mảng `a` sao cho các địa chỉ trỏ đến giá trị nhỏ sẽ được sắp trước các địa chỉ trỏ đến giá trị lớn sẽ được sắp sau.

Nếu các phần tử trong một mảng các pointer lại được gán địa chỉ của mảng khác thì ta sẽ được một mảng của các mảng nhưng không giống như mảng 2 chiều. Vì trong mảng 2 chiều các phần tử nằm liên tục nhau trong bộ nhớ, nhưng với mảng con trỏ thì các mảng con nằm ở vị trí bất kì. Ta chỉ cần lưu trữ địa chỉ của chúng nên việc sắp xếp mảng là sắp xếp các địa chỉ của chúng trong mảng các pointer của chúng.

Như vậy, qua ví dụ trên ta thấy rằng việc sử dụng mảng các pointer có các ý niệm gần giống như việc sử dụng mảng hai chiều.

**Ví dụ 5.27:** Nếu chúng ta khai báo:

```
int m[10][9]; int *n[10];
```

thì cách viết để truy xuất của các mảng này có thể tương tự nhau, chẳng hạn:

`m[6][5]` và `n[6][5]` đều cho ta một kết quả là một số `int`.

Tổng kết, mảng 2 chiều và mảng các pointer có sự khác nhau cơ bản sau:

- Mảng 2 chiều thực sự là một mảng có khai báo, do đó có chỗ đầy đủ cho tất cả các phần tử của nó.
- Mảng các pointer chỉ mới có chỗ cho các pointer mà thôi. Vì vậy, ta cần phải xin cấp phát các vùng nhớ để các pointer này trỏ đến.
- Như vậy mảng các pointer có thể được xem là tốn chỗ hơn là mảng 2 chiều, vì vừa phải lưu trữ các pointer và vừa phải có chỗ cho mỗi phần tử sử dụng.

– Mảng pointer có các ưu điểm:

- + Việc truy xuất đến các phần tử là truy xuất gián tiếp qua các pointer. Vị trí của các mảng con này có thể bất kì, và chúng có thể là những mảng đã có bằng cách xin cấp động (malloc) hay bằng khai báo mảng bình thường.
- + Các mảng con của nó được chỉ đến bởi các pointer, có thể có độ dài tùy ý, hay có thể không có.
- + Đối với các mảng pointer, ta có thể hoán chuyển thứ tự của các mảng con được chỉ đến bởi các pointer này, bằng cách chỉ hoán chuyển bản thân các pointer trong mảng pointer là đủ.

**Ví dụ 5.28:**

Viết hàm nhập vào n là số tháng trong năm, sau khi thực thi hàm trả về chuỗi tên tháng tương ứng. Ta dùng mảng các con trỏ ký tự để lưu trữ giá trị chuỗi tên tháng như sau:

```
//n là số của tháng trong năm
char* chuyenTenThang (int n)
{
    static char* tenThang[12]={ "January",
    "February", "March", "April", "May", "June",
    "July", "August", "September", "October",
    "November", "December"
    };
    if(n< 1 || n> 12)        return NULL;
    return (tenThang[n -1]);
}
```

Trong hàm trên, chúng ta đã khởi tạo cho mảng các pointer trỏ đến kiểu char là tenThang được khai báo static bằng các chuỗi hằng. Giá trị trả về của hàm là pointer chỉ đến một chuỗi ứng với giá trị n tương ứng hoặc trả về một pointer NULL nếu tháng không đúng.

### 5.5.7 Pointer chỉ đến pointer

Có thể khai báo một pointer chỉ đến một pointer trở đến một biến có kiểu dữ liệu là kiểu pointer như sau: kiểu `**tenpointer`;

**Ví dụ 5.29 :**

```
int **pp;
```

Với cách khai báo này, có thể biến pointer của pointer là pp, và được ghi nhận rằng biến của biến pp này chỉ đến là một pointer chỉ đến một biến int. Chúng ta một lần nữa phải cần nhớ rằng thực sự biến pp cho đến lúc này vẫn chưa có một đối tượng để trỏ đến, và chúng ta phải tự mình gán địa chỉ của một pointer nào đó cho nó.

**Ví dụ 5.30:**

```
char *monthname[20] = {"January", "February",  
    "March", "April", "May", "June", "July",  
    "August", "September", "October", "November",  
    "December"    };
```

```
char**pp ;
```

```
pp = monthname; //tên mảng là địa chỉ của phần tử đầu tiên.
```

Lúc đó, \*pp sẽ chỉ đến pointer đầu tiên của mảng, pointer này lại chỉ đến chuỗi "January".

Nếu tăng pp lên pp++ thì sẽ chỉ đến pointer kế tiếp của mảng, pointer này chỉ đến chuỗi "February". ...

## 5.6 Chuỗi ký tự

### 5.6.1 Chuỗi ký tự

Trong ngôn ngữ lập trình C không có kiểu dữ liệu chuỗi mà chuỗi trong C là một dãy các ký tự kiểu char. Một chuỗi trong C được đánh dấu kết thúc là '\0' (còn gọi là NULL trong bảng mã ASCII) và có độ dài tùy ý, điều này cũng có nghĩa chuỗi ký tự trong C là một mảng các ký tự char.

Chúng ta có thể gán một chuỗi cho một biến pointer chỉ đến char

**Ví dụ 5.30:**

```
char str[20]= " \nHappy New Year"
```

Không thể cộng, trừ, nhân, chia 2 chuỗi kí tự lại bằng phép toán đơn thuần. Tất cả những điều đó phải được làm bằng các hàm riêng lẻ. Ta có thể gán một chuỗi này bằng một chuỗi khác (strcpy), so sánh 2 chuỗi kí tự với nhau theo thứ tự từ điển (strcmp), cộng 2 chuỗi với nhau (strcat),...

Mọi hằng chuỗi đều được ngôn ngữ lập trình C lưu trữ như là một mảng các **char** và kết thúc bằng kí tự '\0'. Hơn nữa, một chuỗi trong chương trình chúng ta chỉ nhận được địa chỉ và chỉ đến đầu mảng lưu trữ. Việc truy xuất đến một hằng chuỗi đều được thực hiện qua một pointer chỉ đến mảng đó.

**Ví dụ 5.31:**

```
printf("Happy new year\n");
```

Trong ví dụ trên, hàm printf() thực sự cũng chỉ ghi nhận được pointer chỉ đến mảng kí tự này đang lưu trữ đến một chỗ nào đó mà thôi. Vì vậy, chúng ta có thể gán một hằng chuỗi cho một biến pointer chỉ đến char.

**Ví dụ 5.32:**

```
char*str;  
str = "Happy new year \n";
```

Lúc này, ta đã đưa pointer str giữ địa chỉ của chuỗi kí tự này. Ta có thể quy định rằng một mảng kí tự tận cùng bằng kí tự '\0' được gọi là một chuỗi.

**5.6.2 Một số hàm thao tác trên chuỗi****a. Hàm nhập xuất một chuỗi**

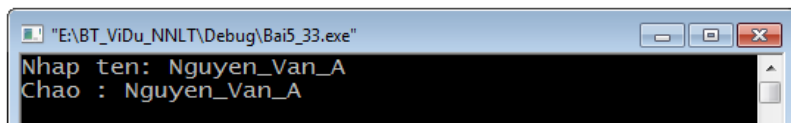
Ta có thể dùng hàm scanf() với định dạng %s để nhập chuỗi.



**Ví dụ 5.33:**

```
#include<stdio.h>

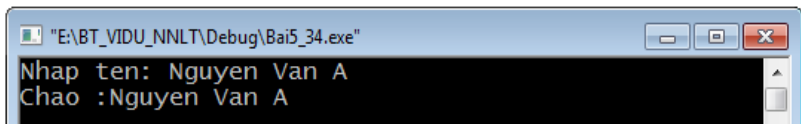
void main()
{
    char ten[50];
    printf("Nhap ten: ");
    /*Không có chỉ thị & vì ten
    chuỗi đã là một địa chỉ*/
    scanf("%s",ten);
    printf("Chao : %s\n",ten);
    getch();
}
```

**Kết quả thực hiện chương trình:**

**Lưu ý:** Nếu dùng hàm *scanf()* để nhập dữ liệu và kết thúc việc nhập dữ liệu bằng phím Enter, thì lúc này phím Enter sẽ cho hai ký tự có mã ASCII là 13 và 10 trong vùng đệm. Như vậy nếu dùng hàm *scanf()* thì ký tự có mã ASCII 10 vẫn còn nằm trong vùng đệm. Nếu ta dùng hàm *gets(chuỗi s)*, ký tự có mã ASCII là 10 được chuyển ngay vào chuỗi s. Tức là hàm *gets* sẽ lấy tất cả các ký tự trong buffer (vùng đệm) của màn hình vào chuỗi cho nên đôi khi chúng ta sẽ nhận được chuỗi không mong muốn do *gets* nhận những ký tự dư của các hàm nhập khác. Để tránh điều này ta dùng hàm *int fflush(void)* để xóa mọi buffer (vùng đệm) hoặc hàm *fflush(stdin)* để xóa vùng đệm bàn phím trước hàm nhập chuỗi *gets(chuỗi s)*.

**Ví dụ 5.34:**

```
void main()
{
    char ten[30];
    printf("Nhap ten: ");
    fflush(); //hoặc dùng hàm fflush(stdin);
    gets(ten);
    printf("Chao :");
    puts(ten);
}
```

**Kết quả thực thi của chương trình:**

Nhập chuỗi kết thúc bằng phím Enter : **char\*gets(char\*s);**

Xuất một chuỗi có xuống dòng sau khi xuất:

```
int put(const char*s);
int printf("%s\n",s);
```

Xuất một chuỗi không xuống dòng sau khi xuất:

```
int printf("%s",s);
int printf(s);
```

**b. Một số hàm xử lý cơ bản của mảng chuỗi**

- *Gán một chuỗi này bằng một chuỗi khác*
  - + Thực chất của việc “gán” này là việc gán từng từng phần tử của chuỗi này vào chuỗi kia. Để làm việc này ta dùng hàm strcpy().
  - + Hàm strcpy() sẽ gán chuỗi source vào chuỗi dest từng phần tử cho đến khi copy kí tự kết thúc chuỗi ‘\0’.

```
char*strcpy(char*dest, const char*source);
```

- *So sánh hai chuỗi kí tự với nhau theo thứ tự từ điển*

- + Việc so sánh được thực hiện bằng cách so sánh từng cặp phần tử của hai chuỗi với nhau. Nếu chúng hoàn toàn giống nhau cho đến kí tự kết thúc thì xem như hai chuỗi là bằng nhau.

```
int strcmp(const char*s1, const char*s2);  
//phân biệt chữ in và chữ thường
```

```
int strcasecmp(const char*s1, const char*s2); //  
không phân biệt chữ in và chữ thường
```

```
int stricmp(const char*s1, const char*s2); //  
không phân biệt chữ in và chữ thường
```

- + Nếu chuỗi s1 nhỏ hơn chuỗi s2 thì sẽ trả về một số m.
- + Nếu chuỗi s1 lớn hơn chuỗi s2 thì sẽ trả về một số dương.
- + Nếu hai chuỗi bằng nhau sẽ trả về số không ( 0 ).

- *Cộng hai chuỗi với nhau*

Hàm chép chuỗi source vào cuối chuỗi dest.

```
char* strcat(char*dest, char*source);
```

- *Tìm một kí tự nào đó trong một chuỗi cho trước*

```
char* strchr(char*s, char ch);
```

- *Tìm độ dài của chuỗi*

```
int strlen(const char*s);
```

## BÀI TẬP CHƯƠNG 5

1. Khai báo mảng số nguyên chứa 100 phần tử.
2. Biểu thức `array[2][4]` có nghĩa là gì?
3. Định nghĩa cấu trúc để tính diện tích hình vuông.
4. Xét ví dụ qua phát biểu sau đây:

```
void main()  
{  
    int a=2, b=3, *pnt  
    pnt=&a  
    b=*pnt  
}
```

Tính giá trị của b.

5. Xét mảng 2 chiều sau:

10	3	2
40	56	1
30	45	89
28	67	100

Cho biết chỉ số của các phần tử của mảng trong các trường hợp giá trị các phần tử như sau: 10, 56, 89, 28, 100.

Viết chương trình (mã giả) để đảo ngược vị trí các phần tử trong mảng một chiều.

6. Viết chương trình để nhập ma trận  $2 \times 3$ , và trừ tất cả các phần tử 2 đơn vị.
7. Viết hàm nhập một mảng a gồm n số nguyên.
8. Viết hàm xuất mảng một chiều gồm n phần tử.
9. Nhập mảng a gồm n phần tử sao cho các số chẵn và lẻ xen kẽ nhau.
10. Viết hàm tìm phần tử lớn nhất trong mảng số nguyên n phần tử.
11. Viết hàm tìm phần tử nhỏ nhất trong mảng số nguyên n phần tử.

12. Viết hàm tính tổng các phần tử chẵn (hoặc lẻ) của mảng số nguyên  $n$  phần tử.
13. Viết hàm xuất các phần tử ở vị trí chẵn (hoặc lẻ) của mảng số nguyên  $n$  phần tử.
14. Tìm số âm lớn nhất/ số dương nhỏ nhất trên mảng số nguyên  $n$  phần tử.
15. Nhập mảng  $a$  gồm  $n$  phần tử sao cho mọi phần tử lặp lại không quá 2 lần.
16. Nhập mảng  $a$  gồm  $n$  phần tử số nguyên sao cho các số dương có thứ tự tăng.
17. Xây dựng hàm nhập mảng  $a$  gồm  $n$  phần tử thỏa:
  - Không chứa số âm.
  - Có nhiều nhất là 3 phần tử có giá trị 0.
  - Khoảng cách giữa 2 phần tử bất kỳ không quá 4.
18. Xây dựng hàm nhập mảng  $a$  gồm  $n$  phần tử số nguyên phân biệt thỏa:
  - Không chứa số nguyên tố lớn hơn 200.
  - Các số không nguyên tố có thứ tự giảm.
19. Viết hàm tính tổng các phần tử của mảng gồm  $n$  phần tử.
20. Tính tổng các phần tử của mảng  $a$  số nguyên có  $n$  phần tử.
21. Tính giá trị trung bình của các phần tử trong mảng số nguyên  $n$  phần tử.
22. Tìm kiếm vị trí đầu tiên của  $x$  trong mảng  $a$  có  $n$  phần tử.
23. Xóa phần tử thứ  $i$  trong mảng  $a$  có  $n$  phần tử.
24. Chèn một phần tử  $x$  vào vị trí thứ  $i$  của mảng  $a$ .
25. Viết chương trình nhập một mảng số nguyên. Tính tổng các vị trí chẵn và tổng các vị trí lẻ.
26. Đếm các số không âm trong mảng  $a$  có  $n$  phần tử.

27. Đếm các số nguyên tố trong mảng a.
28. Đếm số lần xuất hiện của phần tử x trong mảng a.
29. Tìm số nguyên tố lớn nhất trong mảng a.
30. Tìm số có bình phương nhỏ nhất trong mảng a.
31. Tìm số có số lần xuất hiện nhiều nhất trong mảng a.
32. Tạo mảng b chứa tất cả các số dương của mảng a.
33. Tạo mảng b chứa tất cả các phần tử của mảng a sao cho mỗi phần tử chỉ xuất hiện trong b đúng một lần.
34. Xóa tất cả các số nguyên tố trong mảng a.
35. Sắp xếp mảng a sao cho:
  - Các số chẵn ở đầu mảng và có thứ tự tăng.
  - Các số lẻ ở cuối mảng và có thứ tự giảm.
36. Sắp xếp mảng a sao cho:
  - Số dương ở đầu mảng và có thứ tự tăng
  - Số âm ở giữa mảng và có thứ tự giảm
  - Số 0 ở cuối
37. Sắp xếp mảng sao cho các phần tử chẵn tăng, các phần tử còn lại cố định. Sắp thứ tự tăng theo hai tiêu chuẩn: Số lần xuất hiện và giá trị xuất hiện.
38. Viết chương trình nhập một ma trận nguyên tối đa 20 dòng 20 cột và xuất ra ma trận này.
39. Tìm phần tử lớn nhất của ma trận.
40. Viết chương trình tính tổng dòng thứ i và cột thứ j của ma trận số nguyên  $m \times n$ .

## **CHƯƠNG 6. FILE DỮ LIỆU**

### **6.1 Giới thiệu về file**

#### **6.1.1 Giới thiệu**

Như chúng ta đã biết, máy tính là công cụ giúp con người lưu trữ và xử lý thông tin. Hầu hết các chương trình đều cần phải lưu trữ dữ liệu sau khi xử lý. Vì vậy C cung cấp cho chúng ta các kỹ thuật xử lý lưu trữ trên file.

#### **6.1.2 Khái niệm File**

- Là một đơn vị lưu trữ logic.
- Được biểu thị bằng một tên.
- Bao gồm một tập hợp dữ liệu do người tạo xác định.
- Được lưu trữ trên thiết bị lưu trữ phụ bằng cách ánh xạ lên đơn vị vật lý của thiết bị.
- Được C hỗ trợ các thao tác truy xuất.
  - + Tạo mới.
  - + Đọc, ghi phân tử.
  - + Xóa.
  - + Đổi tên.

Mỗi khi hệ điều hành mở một file, hệ điều hành thao tác với đĩa, truy xuất thông tin cơ bản của file, rồi trả về địa chỉ vùng lưu trữ gọi là handle của file – file ID – để nhận dạng duy nhất cho file này. Chương trình của chúng ta mỗi khi thao tác phải thông qua biến pointer đó. Để lấy pointer của file, chúng ta phải khai báo biến file, ngôn ngữ lập trình C dùng khai báo biến FILE \* f, để lấy handle bằng lệnh mở file.

### 6.1.3 Cách thao tác với file

Thao tác chuẩn: người lập trình không cần biết quá trình thực hiện việc thao tác với file như thế nào. Đó là việc của hệ thống.

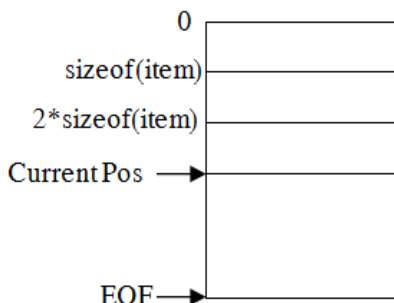
Thao tác mức hệ thống (thao tác thủ công): thao tác file thông qua bộ đệm (buffer - một vùng nhớ). Người lập trình phải tự quản lý các bộ đệm đọc ghi file. Thao tác file này gần giống với cách thao tác file của hệ điều hành MS – DOS. Thông thường chỉ có những người lập trình hệ thống mới sử dụng thao tác file mức hệ thống.

Thao tác với file phải thao tác với phần cứng. Do đó, việc thao tác với file có thể thành công hoặc thất bại.

### 6.1.4 Tổ chức lưu trữ dữ liệu trên file

#### Cách nhìn logic

Các phần tử được lưu trữ liên tục (danh sách đặc):



Hình 7. 1. Cách nhìn logic việc lưu trữ dữ liệu trên file

Địa chỉ tuyến tính bắt đầu từ 0 (Zero – base address).

Truy xuất từng n phần tử, từng khối dữ liệu.

#### Cách nhìn vật lý

- Dữ liệu được phân bố trên từng sector (đơn vị lưu trữ vật lý).
- Bao gồm một tập sector xác định.
- Các sector có thể liên tục hay rời nhau.
- Truy xuất từng sector.



Ngôn ngữ lập trình C xem file như là một dòng (stream) các byte, với các thiết bị xuất nhập theo từng byte cũng được xem là file, C định nghĩa sẵn các tên cho các thiết bị này và các file này đã được mặc định mở sẵn cho ta truy xuất ngay khi mở máy tính.

Handle	Tên	Thiết bị
0	<b>stdin</b>	Standard input – Thiết bị nhập chuẩn – Bên phím
1	<b>stdout</b>	Standard output – Thiết bị xuất chuẩn – Màn hình
2	<b>stderr</b>	Standard error – Thiết bị xuất lỗi chuẩn – Màn
3	<b>stdaux</b>	hình
4	<b>stdprn</b>	Standard auxiliary – Thiết bị ngoại vi chuẩn – Cổng nối tiếp Standard printer – Thiết bị in chuẩn – Máy in song song

*Bảng 6.1. Tên và ý nghĩa tương ứng của các lệnh trong thư viện chuẩn*

Khi thao tác với file, ở mỗi thời điểm chỉ truy xuất được một phần tử lưu trữ trong file. Vị trí hiện hành đang thao tác (file position) gọi là chỉ số trong file hay con trỏ file – chính là số thứ tự của phần tử truy xuất hiện hành. Chỉ số thứ tự này bắt đầu từ 0.

## 6.2 Định nghĩa biến file và các thao tác mở, đóng file

Như chúng ta đã biết, dữ liệu được mã hóa thành dạng nhị phân. Như vậy, với một ký tự lưu trữ ta xem nó như một byte hay là một tập các giá trị số nhị phân.

Khi mở một file, vấn đề quan trọng là chúng ta phải chỉ định cách nhìn của chúng ta về các byte lưu trữ dữ liệu. Nếu chúng ta xem mỗi byte là mã ASCII của ký tự, ta chỉ định mở file dạng văn bản. Nếu chúng ta xem mỗi byte là một số nhị phân, việc xử lý byte này 1 số hay chữ sẽ do chương trình giải quyết, thì ta chỉ định mở file dạng nhị phân.

### 6.2.1 Định nghĩa biến file trong C

Thao tác file chuẩn **FILE \* f;**

Thao tác mức hệ thống **int f;**

Dữ liệu trên file là một dãy các byte(8 bit) có giá trị từ 0 đến 255. Số byte của dãy là kích thước thật của file(size on disk).

#### Có hai loại file

##### File văn bản thô (text)

- Dữ liệu là một chuỗi kí tự liên tục.
- Phân biệt các kí tự điều khiển.
- Xử lý đặc biệt với các kí tự điều khiển.

##### File nhị phân (binary)

- Dữ liệu được xem như là một dãy byte liên tục.
- Không phân biệt các kí tự điều khiển.
- Chuyển đổi dữ liệu tùy thuộc vào biến lưu trữ khi đọc, ghi.

### 6.2.2 Hàm mở, đóng file chuẩn

#### Mở file

Cú pháp:

**FILE \* fopen(const char \* filename, const char \* mode);**

Hàm trên sẽ mở file có tên là filename, dạng mở mode. Nếu mở thành công thì trả về một pointer có trị khác NULL, không mở được trả về trị NULL. Tham số mode là một chuỗi kí tự chỉ định dạng mở.

Các mode mở file thông dụng:

- “r”: Mở file để đọc (read).
- “w”: Tạo file mới để ghi(write). Nếu file này đã tồn tại trên đĩa thì bị ghi đè.

- “a”: Mở để ghi vào cuối file nếu file này đã tồn tại, nếu file này chưa có thì sẽ được tạo mới để ghi (append).
- “r+”: Mở file để đọc để cập nhật (cả đọc lẫn ghi).
- “w+”: Mở file mới để được cập nhật (cả đọc lẫn ghi). Nếu file này đã có sẽ bị ghi đè.
- “a+”: Mở để đọc và cập nhật (ghi) vào cuối file. Sẽ tạo mới nếu file này chưa có.
- Ghi chú:
- Thêm kí tự “t” để mô tả mở file dạng text mode (Thí dụ: “rt”, “w+t”,...).
- Thêm kí tự “b” để mô tả mở file dạng nhị phân (Thí dụ: “wb”, “a+b”,...).

### **Đóng file**

Cú pháp:

<b>int fclose(FILE * f);</b>
------------------------------

Nếu đóng file thành công trả giá trị 0, nếu đóng thất bại trả về giá trị EOF (giá trị -1)

### **Kết thúc file**

Sau khi tạo xong một file văn bản, đóng file này, byte mang giá trị 1Ah (26 của hệ 10 – tương đương với khi gõ tổ hợp phím Ctrl + Z) sẽ tự động chèn vào cuối file để ấn định hết file.

Nói chung, file được quản lý bằng kích thước của file (số bytes). Khi đã đọc hết số byte có trong file, thì dấu kí hiệu EOF (end of file) được DOS thông báo cho chương trình. Dấu hiệu EOF là tên hằng mà C khai báo sẵn trong thư viện STDIO.H và nó mang giá trị -1.

Như vậy, nếu một file dữ liệu (có cả số) được mở dạng văn bản, nếu trong giữa file mà có giá trị 1Ah thì quá trình đọc sẽ bị ngưng nửa

chừng (hàm đọc file sẽ trả về giá trị -1 cho chương trình báo đã kết thúc file).

Chỉ định file ở chế độ	Dữ liệu trong chương trình C	Được lưu trữ trên file
Văn bản	'\n' EOF	CR (13), LF (10) 1Ah
Nhị phân	'\n' '\r'\n' 1Ah (số)	LF CR LF 1Ah

*Bảng 6.2. Sự tương ứng giữa dữ liệu trong chương trình C và dữ liệu trên file*

Hàm `int eof(int handle);` trong `IO.H`

Trả về        -1        :        Đã hết file  
                 0        :        Chưa hết file  
Số nguyên khác :        Mô tả lỗi

### **Sự khác biệt giữa mở file dạng text và dạng nhị phân:**

Kiểu file văn bản thường hay được dùng trong hệ điều hành UNIX, file thường hay gặp ở DOS. Do vậy, dòng Borland duy trì cả hai dạng để có sự tương hợp với cả hai hệ điều hành.

- Dạng nhị phân: Lưu trữ giống như lưu trữ trong bộ nhớ trong (RAM, lưu trữ nhị phân, dữ liệu được xem như các số nhị phân).
- Dạng văn bản: Lưu trữ dạng nhị phân là m ASCII của kí tự số (phân loại này nảy sinh do cách lưu trữ số).

### **Ví dụ 6.1:**

Ghi một vi phần tử lên file TEXT và file BIN

```
FILE *fTXT, *fBIN;
fTXT = fopen("D:\\z\\TEST.TEXT", "wt");
fBIN = fopen("D:\\z\\TEST.BIN", "wt");
//Lần lượt ghi 'A', 26, 10, 'B' lên hai file
fput('A', fTXT);
```

```
fput(26, fTXT);  
fput(10, fTXT);  
fput('B', fTXT);  
fput('A', fBIN);  
fput(26, fBIN);  
fput(10, fBIN);  
fput('B', fBIN);  
fclose(fTXT);  
fclose(fBIN);
```

**Kết quả:**

File TEST.TEXT chứa chuỗi : 65 26 13 10 66 (5 bytes)

File TEST.BIN chứa chuỗi : 65 26 10 66 (5 bytes)

**Đọc dữ liệu trên file TEST.TEXT**

```
fTXT = fopen("D:\\z\\TEST.TEXT", "rt");  
while(!feof(fTXT))  
{  
    char c = fgetc(fTXT);  
    printf("%c\t", c);  
}
```

Kết quả chỉ xuất ra được kí tự A vì khi gặp kí tự 26 máy sẽ hiểu là kết thúc file.

**Đọc dữ liệu trên file TEST.BIN**

```
fBIN = fopen("D:\\z\\TEST.BIN", "rb");  
while(!feof(fBIN))  
{  
    char c = fgetc(fBIN);  
    printf("%c\t", c);  
}
```

Kết quả in ra đầy đủ bởi không quan tâm đến các giá trị

6.2.3 Thao tác nhập, xuất với file

Thao tác xuất nhập chuẩn trong stdio.h

Ta gọi:

```
FILE * f; // handle của file
char c; //kí tự cần đọc/ghi
char * s; // chuỗi kí tự cần đọc/ghi
int n; //số kí tự cần đọc hay là số record cần thao tác
struct { } Rec; //biến cấu trúc
```

Đơn vị của file	Hàm nhập(đọc từ file ra biến)	Hàm xuất(ghi từ biến lên file )
Ký tự	C = fgetc(f); Đọc kí tự hiện hành trong file f ra c.	int fputc(c,f); Ghi c vào vị trí hiện hành trong f.
Chuỗi kí tự	char * fgets(char*s,int n,f) Đọc n byte trong f vào s, không đọc được trả về NULL	fputs(char*s,f) Thành công : trả về kí tự cuối cùng đã ghi. Có lỗi trả về EOF
Chuỗi định dạng	fscanf(f,"chuỗi định dạng", &biến1,&biến,&biến,... )	fprintf(f,"chuỗi định dạng", biến1, biến, biến,... )
Struct	size_t fread(&Rec,sizeof(struct), n, f); Đọc n record từ f đưa vào biến Rec Đọc được : Trả về số phần tử đọc được Không đọc được : trả về 0	int fwrite(&Rec,sizeof(struct), n,f); Ghi n record từ Rec lên file Ghi được trả về số phần tử đã được ghi

Bảng 6.3. Hàm nhập/xuất cho từng loại chuỗi

### Thao tác xuất nhập mức hệ thống trong IO.H

`int read(int f, void*buf, unsigned len);` : Đọc từng byte từ file f đưa vào buffer buf.

Kết quả trả về:

- Nếu thao tác xuất thành công, hàm sẽ trả về số nguyên dương cho biết số byte đã đọc được.
- Khi thao tác đọc thất bại, hàm trả về -1.

### Thao tác ghi

`int write(int f, void * buf, unsigned len);` : Đọc từng byte trong buf và ghi vào file f.

Kết quả trả về:

- Nếu thao tác ghi thành công, hàm trả về số nguyên cho biết số byte đã ghi được.
- Nếu thao tác ghi thất bại, hàm trả về -1

### Truy xuất lỗi xuất nhập file

Có thể có nhiều trường hợp có lỗi khi mở file như hết khoảng trống trên đĩa, đĩa mềm dán chống ghi, đĩa hỏng, ổ đĩa hư, ... Khi mở file có lỗi, hàm `fopen()` sẽ trả về giá trị NULL.

**Ví dụ 6.2:** Xuất nội dung file “textfile.txt” ra màn hình. Mở file có kiểm tra, nếu không mở được, thì xuất thông báo, nếu mở được đọc từng kí tự vào biến `ch` xuất ra màn hình và đóng file lại.

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

```
void main()
{
    FILE*fptr;
    int  ch;
```

```
if(fp_ptr = fopen("textfile.txt", "r") == NULL)
{
    printf("Không thể mở file\n");
    getch();
    exit(0);
}

while(ch = fgetc(fp_ptr) != EOF)
printf("%c", ch);

fclose(fp_ptr); // Đóng file
}
```

### **Để xác định lỗi file hay không ta dùng các hàm:**

`int ferror(FILE *fp_ptr);` - Hàm này trả về giá trị 0 nếu không có lỗi, trả giá trị khác 0 nếu có lỗi.

`void perror(const char *string);` - Hàm xuất thông báo lỗi muốn thông báo lên màn hình.

### **Quy trình xử lý File**

#### **Bước 1: Mở file.**

- Xác định chế độ mở chính xác(text/binary).
- Kiểm tra lỗi.

#### **Bước 2: Truy xuất xử lý.**

- Áp dụng hợp lý cách truy xuất tùy theo chế độ mở.
- Quản lý con trỏ chỉ vị trí.
- Kiểm tra lỗi.

#### **Bước 3: Đóng file nhằm đảm bảo tính toàn vẹn dữ liệu.**

### **Ví dụ về thao tác mở file như sau:**

```
//Mở chế độ text
FILE *fTXT;

fTXT = fopen("D:\\Z\\TEST.TXT", "wt");

/*Mở để ghi/tạo mới */
```



```

fTXT = fopen("D:\\Z\\TEST.TXT","rt"); // Mở để
đọc
fTXT = fopen("D:\\Z\\TEST.TXT","r+t");
        /*Mở để đọc/ghi*/

//Mở chế độ binary
FILE *fBIN;
fBIN = fopen("D:\\Z\\TEST.BIN","wb");
        /*Mở để ghi/tạo mới */
fBIN = fopen("D:\\Z\\TEST.BIN","rb");
        /*Mở để đọc*/
fBIN = fopen("D:\\Z\\TEST.BIN","r+b");
        /* Mở để đọc/ghi */
fBIN = fopen("D:\\Z\\TEST.BIN","ab");
        /* Mở để ghi thêm(append) vào cuối */

//Xử lý lỗi
if(fTXT == NULL)
{
    printf("Loi mo file = %d. Noi dung =
%s",errno,strerror(errno));
    return 0;
}

```

**Lưu ý:** Thao tác đọc, ghi không thể thực hiện liên nhau, cần phải có một thao tác fseek hay rewind ở giữa.

Xét ví dụ sau:

```

FILE*fBIN;
fBIN = fopen("D:\\z\\TEST.BIN"."r+b");// Mở để
update
int x;
while(fread(&x,sizeof(x),1,fBIN))

```

```
if(x == 0)
{
    x++;
    fseek(fBIN, -sizeof(x), SEEK_CUR);
    fwrite(&x, sizeof(x), 1, fBIN);
}
fclose(fBIN);
```

**Kết quả:** Chỉ update được 1 phần tử

**Ví dụ về truy xuất dữ liệu ở chế độ text như sau:**

```
n = fscanf(fTXT, "%d,%f,%s", &Item, &fItem, szItem);
/* trả về số phần tử đọc được */
if(n < 3 || n == EOF)
{
    printf("Không đọc được!! Mã lỗi=%d", errno);
    ... // Các xử lý đặc biệt
}
n = fprintf(fTXT, "%d,%f,%s", &Item, &fItem, szItem);
/*trả về số byte ghi được; */
if(n < sizeof(Item)+sizeof(fItem)+strlen(szItem))
{
    printf("Ghi không được. Mã lỗi = %d", errno);
    ... // Các xử lý đặc biệt
}
```

**Ví dụ về truy xuất dữ liệu ở chế độ binary như sau:**

```
n = fread(&Item, sizeof(Item), nItemRead,
fBIN);
/*trả về số phần tử đọc được*/
if(n < nItemRead)
{
    printf("Không đọc được. Mã lỗi = %d", errno);
    ... // Các xử lý đặc biệt
```

```

    }
    n = fwrite(&Item, sizeof(Item), nItemWrite,
fBIN);
    /*trả về số byte đọc được*/
    if(n < nItemWrite)
    {
        printf("Ghi không thành công. M lỗi =
%d",errno);
        ... // Các xử lý đặc biệt
    }

```

Ví dụ sau sẽ thực hiện việc cập nhật SAI một phần tử khi quản lý bằng con trỏ:

```

fread(&Item,sizeof(Item),1,f);
Item++; // cập nhật phần tử Item
fwrite(&Item,sizeof(Item),1,f);

```

Ví dụ về đóng file sau khi xử lý

- **Đóng từng file:**

```

fclose(fTXT);
fclose(fBIN);

```

- **Đóng tất cả các file đang mở:**

```

fcloseall();

```

- **Ví dụ sau về việc không bảo toàn dữ liệu khi không đóng file:**

```

char szFileName[] = "D:\\z\\TEST.BIN";
//Ghi dữ liệu lên file
FILE*f = fopen(szFileName,"wb");
for( i = 0; i < n; i++)
    fwrite(&i,sizeof(i),1,f);
//Đọc dữ liệu từ file
i = 0;
f = fopen(szFileName,"rb");

```

```
while(fread(&i,sizeof(i),1,f))  
    i++;  
printf("%d",i);  
fclose(f);
```

Kết quả: (tùy thuộc vào hệ thống)

```
n = 100    → i = 0  
n = 500    → i = 256  
n = 513    → i = 512  
n = 1000   → i = 768
```

## BÀI TẬP CHƯƠNG 6

1. Tạo file input.txt chứa văn bản (tùy ý). Viết chương trình:
  - a. Đọc dữ liệu từ file, xuất ra màn hình kèm độ dài của dữ liệu (bao nhiêu ký tự).
  - b. Ghi nội dung vừa đọc vào file output.txt.
2. Viết chương trình đọc một file text và xóa các dòng trống nếu có trong file.
3. Viết chương trình cắt bỏ các dòng thừa, cắt bỏ các khoảng trống thừa, đổi các ký tự đầu mỗi từ ra chữ hoa của một file text.
4. Tạo file IP.txt chứa dãy số nguyên ngẫu nhiên gồm: số phần tử và các giá trị phần tử trong dãy. Viết chương trình đọc dãy số đó vào mảng 1 chiều a. Thực hiện các xử lý và ghi kết quả vào file OP.txt chứa:
  - a. Thứ tự và giá trị từng phần tử.
  - b. Giá trị max/min của dãy số.
  - c. Giá trị tổng/tích của dãy số
  - d. Kết quả dãy số sau khi xóa các số chẵn
  - e. Kết quả dãy số sau khi sắp xếp tăng dần/giảm dần.
5. Tạo file IP.txt chứa thông tin dãy số nguyên ngẫu nhiên trong mảng 2 chiều gồm: số dòng, số cột và các giá trị phần tử trong mảng 2 chiều. Viết chương trình đọc dãy số đó vào mảng 2 chiều a. Thực hiện các xử lý và ghi kết quả vào file OP.txt chứa:
  - a. Mảng 2 chiều dạng bảng.
  - b. Giá trị max/min của mảng.
  - c. Giá trị tổng/tích của mảng.
  - d. Tổng từng dòng
  - e. Tích từng cột
  - f. Dãy các phần tử trên đường chéo chính/đường chéo phụ.
  - g. Các phần tử thuộc từng biên: top, bottom, left, right.
  - h. Kết quả mảng sau khi sắp xếp tăng dần theo thứ tự từ trái qua phải, từ trên xuống dưới.

6. Tạo một file văn bản IP.txt, đọc dữ liệu từ file và thực hiện các yêu cầu sau:
  - a. Đếm số lần xuất hiện của một kí tự chữ cái trong một tập tin văn bản, ký tự nhập từ bàn phím.
  - b. Đếm số từ có trong văn bản.
  - c. Văn bản có bao nhiêu đoạn văn.
7. Tạo 2 file văn bản, viết chương trình đọc và nối hai tập tin văn bản với nhau thành một tập tin văn bản mới.
8. Tạo file văn bản IP.txt chứa số lượng và thông tin họ tên của danh sách sinh viên, mỗi tên nằm trên một dòng. Viết chương trình đọc danh sách tên từ file vào và thực hiện các yêu cầu sau:
  - a. Tìm trong danh sách có tên sinh viên “Le Van Anh” không?
  - b. Có bao nhiêu sinh viên họ “Nguyen”
  - c. Giả sử tên nữ luôn có tên lót là Thi và nam luôn có tên lót là Van. Hỏi có bao nhiêu sinh viên nam và nữ trong danh sách.
  - d. Xuất các sinh viên tên Lan
  - e. Ghi danh sách sinh viên nam và nữ vào 2 file riêng biệt.

## CHƯƠNG 7. KỂ DỮ LIỆU CẤU TRÚC

### 7.1 Kiểu dữ liệu cấu trúc (struct)

#### 7.1.1 Giới thiệu

Đối với một vấn đề cần mô tả, tùy góc nhìn, tùy vào mức độ quan tâm của mỗi người mà có các mô tả khác nhau.

**Ví dụ 7.1:** Mô tả một nhân viên trong cơ quan:

- **Đối với người kế toán:** một nhân viên được mô tả bằng: mã số, họ, tên lót, tên, ở phòng ban nào, chức vụ gì...
- **Đối với người làm công tác tổ chức:** một nhân viên được mô tả bằng: mã số, họ, tên lót, tên, ngày tháng năm sinh, nơi sinh, địa chỉ, ở phòng ban nào, chức vụ, giới tính, đã có gia đình chưa, lương, ...

**Ví dụ 7.2:** Mô tả một sinh viên.

- **Đối với phòng giáo vụ:** một sinh viên được mô tả bằng: mã số, họ, tên lót, tên, năm sinh, địa chỉ, cha, mẹ, năm nhập học, điểm trúng tuyển, các điểm cho từng môn học...
- **Đối với một giáo viên đang dạy một môn học:** một sinh viên được mô tả bằng: mã số, họ, tên lót, tên, điểm môn học mình dạy...

Cấu trúc (struct) là kiểu dữ liệu phức hợp được xây dựng từ những kiểu dữ liệu khác. Các kiểu dữ liệu bên trong một cấu trúc có thể là một kiểu cấu trúc khác.

#### 7.1.2 Định nghĩa

##### – Kiểu 1

```
struct <tên cấu trúc>
{
    <tên kiểu dữ liệu><tên thành phần 1>;
    <tên kiểu dữ liệu><tên thành phần 2>;
    ...
};
```

**Ví dụ 7.3:** Một nhân viên được mô tả bằng 3 thành phần thông tin, còn gọi là trường (field): mã số (int), họ tên (tối đa 29 ký tự), lương (float) có thể khai báo như sau:

```
struct Tenhanvien// struct khai báo một cấu trúc
{
    int maso;
    char hoten[30]; //chứa 1 ký tự cuối chứa '\0'
    float luong;
}; //để kết thúc một khai báo cấu trúc.
```

**Ví dụ 7.4:** Một ngày được mô tả bằng 4 thành phần thông tin: ngày (day), tháng (month), năm (year) và thứ của ngày trong tuần (chuỗi ngày).

```
struct date // mô tả ngày tháng năm
{
    int day ;
    int month;
    int year;
    char weekdays [4]; //Mon, Tue, Wed
};
```

### **Ví dụ 7.5**

Màn hình máy tính sử dụng hệ tọa độ nguyên, một ký tự xuất hiện trên màn hình có thể được mô tả như sau:

```
struct SCR_CHAR // screen character
{
    char c; //ký tự gì
    short x,y; // được xuất ở điểm nào trên màn
    hình
};
```



– **Kiểu 2**

```
typedef struct <tên cấu trúc>  
{  
<tên kiểu dữ liệu><tên thành phần 1>;  
<tên kiểu dữ liệu><tên thành phần 2>;  
...  
}<tên cấu trúc mới>;
```

**Ví dụ 7.6:** Kiểu nhân viên có thể được khai báo như sau:

```
typedef struct Tennhanvien  
{  
    int maso;  
    char ten[30];  
    float luong;  
} TENNV;
```

Sau câu lệnh trên chúng ta có kiểu dữ liệu Nhân viên có 2 tên gọi là Tennhanvien và TENNV.

Nếu chỉ khai báo struct, khi cần dùng kiểu dữ liệu đó, chúng ta phải dùng tên kèm theo struct phía trước tên. Dùng typedef để định nghĩa kiểu struct thì khi cần kiểu dữ liệu mới này không cần phải đánh lại chữ struct ở đầu, mà có thể dùng tên gọi mới sau khai báo struct.

Chẳng hạn để khai báo một biến thuộc kiểu Nhân viên, ta có 2 cách ghi:

```
struct Tennhanvien a;  
hoặc  
TENNV a;
```

### 7.1.3 Khai báo

Khai báo một cấu trúc như trên chỉ là khai báo thêm một kiểu dữ liệu mới cho chương trình thay vì dùng các kiểu dữ liệu có sẵn như **int**, **float**... Để có được một biến cấu trúc ta phải khai báo biến với kiểu là cấu trúc đó.

Cú pháp:

<b>&lt;Tên cấu trúc&gt;&lt;tênbiến&gt; ;</b>
--

#### ***Ví dụ 7.7:***

```
TENNV x; // Khai báo biến x kiểu Nhân viên;
```

```
TENNV dsNhanvien[100]; //khai báo mảng 1 chiều dsNhanvien chứa 100 nhân viên.
```

Nếu định nghĩa cấu trúc kiểu 1 ta có thể khai báo biến cấu trúc ngay khi định nghĩa cấu trúc (không áp dụng được nếu định nghĩa cấu trúc kiểu 2).

#### ***Ví dụ 7.8:***

Khai báo biến cấu trúc Ngay kiểu date kết hợp khi khai báo cấu trúc:

```
struct date
{
    int day ;
    int month;
    int year;
    char weekdays [4];
} Ngay;
```

Khai báo biến Ngay1 kiểu cấu trúc date

```
struct date Ngay1;
```

**Ví dụ 7.9:**

```
typedef struct Tenhanvien
{
    int maso;
    char ten[30];
    float luong;
}NV;
NV nv1, nv2;
//khai báo hai biến cấu trúc nv1, nv2 kiểu
Tenhanvien
```

**7.1.4 Cấu trúc lồng nhau**

Cấu trúc lồng nhau là một cấu trúc có thành phần lại là một cấu trúc.

**Ví dụ 7.10:**

```
typedef struct DATE
{
    int d, m, y; // date, month, year
}Date;

struct Tenhanvien
{
    int maso;
    char ten[30];
    Date ngaysinh;
    float luong;
}nv1, nv2; // định nghĩa 2 biến nv1 và nv2.
```

**7.1.5 Khởi tạo cấu trúc**

Việc khởi tạo biến cấu trúc tương tự như khởi động một mảng. Ta khởi tạo 2 biến cấu trúc nv1 và nv2 như sau:

```
struct Tenhanvien nv1={101, "TRAN HUNG DUNG",
1250000};
```

hay

```
Tennhanvien nv2={106, "NGUYEN HOANG ANH THU",
16750000};
```

Khởi tạo biến cấu trúc là ấn định giá trị hằng cho lần lượt các field theo thứ tự đã khai báo, cách nhau bằng dấu phẩy, bao tất cả lại trong cặp {} kết thúc bằng dấu ";"

Khởi tạo một biến cấu trúc Ngay có kiểu date:

```
date Ngay = { 2, 9, 1989, "Sat" };
```

### 7.1.6 Truy xuất các thành phần của một biến cấu trúc

Để truy xuất một thành phần của biến cấu trúc, ngôn ngữ C có các phép toán lấy thành phần như sau:

- Nếu biến cấu trúc là biến thông thường (không phải là con trỏ) thì truy xuất thành cấu trúc bằng dấu chấm "."

**<Tên biến cấu trúc> . <Tên thành phần>**

- Nếu biến cấu trúc là biến con trỏ thì truy xuất thành phần bằng dấu mũi tên "->" (dấu trừ và dấu lớn).

**<Tên biến con trỏ cấu trúc> -> <Tên thành phần>**

#### Ví dụ 7.11:

```
1) Date Ngay;
```

```
Ngay.day = 2; // gán giá trị 2 cho thành phần
day của biến cấu trúc Ngay
```

```
2) struct Tennhanvien
```

```
{
```

```
int maso;
```

```
char ten[30];
```

```
float luong;
```

```
} nv1, nv2;
```

```
nv1.maso; // để truy xuất maso của nhân viên
```

```
nv1.ten; // để truy xuất ten của nhân viên
```

```
nv1.luong;          // để truy xuất lương của nhân
viên
```

```
3)   Date *d;
```

```
d->day = 2; // gán giá trị 2 cho thành phần
day của biến con trỏ cấu trúc d
```

## 7.2 Mảng các struct

Khai báo mảng các cấu trúc hoàn toàn tương tự như khai báo mảng khác, chỉ có điều ở phần kiểu sẽ là tên một cấu trúc.

Cú pháp:

```
struct <tên cấu trúc> <tên mảng> [ <kích thước>];
```

**Ví dụ 7.12:**

```
struct date d[10];
```

Khi đó việc truy xuất đến một phần tử của mảng như `d[2]` ta sẽ thu được một biến có cấu trúc `date` và có thể lại tiếp tục truy xuất đến các thành phần của nó.

**Ví dụ 7.13:**

```
d[2].month = 10;
```

```
strcpy ( d[2]. weekday, "SUN");
```

## 7.3 Con trỏ kiểu cấu trúc

Ta có thể khai báo một biến pointer chỉ đến một cấu trúc để có thể lưu giữ lại địa chỉ của một biến cấu trúc nào đó cần thiết.

Cú pháp:

```
struct <tên cấu trúc> * <tên pointer>;
```

**Ví dụ 7.14:**

```
struct date *p ;
```

Việc sử dụng pointer chỉ đến cấu trúc thường được dùng để gửi cấu trúc đến cho một hàm.

Việc truy xuất đến thành phần của một cấu trúc thông qua một pointer được thực hiện bằng một toán tử: `->` là phép toán lấy thành phần nội dung của pointer (đã được đề cập ở phần 6.1.6).

**Ví dụ 7.15:**

```
printf("\n Ngày lưu trữ là : %d ", p -> day);  
printf("\n Tháng lưu trữ là : %d ", p ->  
weekday);
```

**7.4 Cấu trúc đệ quy**

Người ta thường dùng cấu trúc đệ quy để chỉ các cấu trúc mà thành phần của nó lại có các pointer chỉ đến một biến cấu trúc cùng kiểu.

**Ví dụ 7.16:**

```
struct node  
{  
    int num ;  
    struct node *pNext ;  
};
```

Hoặc có thể có một cấu trúc như sau:

```
struct pNode  
{  
    int key ;  
    struct pNode * left ;  
    struct pNode * right ;  
};
```

Đoạn lệnh trên được xem là một cấu trúc đệ quy .

## BÀI TẬP CHƯƠNG 7

1. Một cuốn sách gồm các thông tin sau:

*Mã sách là chuỗi có 10 ký tự*

*Tên sách là chuỗi tối đa 30 ký tự*

*Giá bán là một số thực*

*Số lượng là một số nguyên*

Hãy viết chương trình thực hiện các yêu cầu sau :

- a. Xây dựng cấu trúc SACH mô tả cuốn sách
  - b. Cho một mảng có n cuốn sách. Tạo file IP.txt chứa thông tin n cuốn sách. Đọc dữ liệu từ file IP.txt
  - c. Xuất thông tin n cuốn sách.
  - d. Viết hàm sắp xếp mảng tăng dần theo tên sách.
  - e. Viết hàm xuất các sách có số lượng >10.
  - f. Xuất các sách có mã số bắt đầu là « SA »
  - g. Xuất các sách có tên chứa chuỗi « Lap trình »
  - h. Xóa các sách có số lượng 0.
  - i. Sắp xếp danh sách giảm dần theo số lượng.
2. Một điện thoại gồm các thông tin sau:
- Mã điện thoại là chuỗi có 10 ký tự*
- Tên điện thoại là chuỗi tối đa 20 ký tự*
- Nhà sản xuất là chuỗi tối đa 20 ký tự*
- Giá bán là một số thực*
- Số lượng là một số nguyên*

Hãy viết chương trình thực hiện các yêu cầu sau, kết quả mỗi yêu cầu cần xuất ra màn hình và ghi ra file:

- a. Xây dựng cấu trúc DIENTHOAI mô tả điện thoại
  - b. Cho một mảng có n điện thoại. Tạo file IP.txt chứa thông tin n điện thoại. Đọc dữ liệu từ file IP.txt
  - c. Xuất thông tin n điện thoại.
  - d. Viết hàm sắp xếp mảng giảm dần theo giá bán.
  - e. Viết hàm xuất các điện thoại có số lượng <10.
  - f. Xuất các điện thoại có mã số bắt đầu là « IP »
  - g. Xuất các điện thoại thuộc hãng « SAMSUNG »
  - h. Đếm số lượng sản phẩm của từng hãng.
3. Một phân số gồm 2 thông tin tử số và mẫu số (khác 0) đều là số nguyên. Hãy viết chương trình thực hiện các yêu cầu sau, kết quả mỗi yêu cầu cần xuất ra màn hình và ghi ra file :
  - a. Xây dựng cấu trúc PHANSO mô tả phân số
  - b. Cho một mảng có n phân số. Tạo file IP.txt chứa thông tin n phân số. Đọc dữ liệu từ file IP.txt. Nếu phân số nào có mẫu = 0 thì loại bỏ khỏi danh sách.
  - c. Xuất thông tin n phân số và số lượng phần tử.
  - d. Xuất dãy phân số sau khi rút gọn
  - e. Viết hàm sắp xếp dãy phân số giảm dần.
  - f. Tính tổng các phân số.
  - g. Tìm phân số max/min
  - h. Đếm số phân số >1
  - i. Xuất các phân số có giá trị nguyên (nghĩa là tử chia hết cho mẫu).
  - j. Qui đồng các phân số.
4. Viết chương trình quản lý nhân sự cho một công ty, mỗi nhân viên trong công ty gồm các thông tin sau: mã số (không có hai



người trùng mã số), họ, tên, ngày sinh, nơi sinh, địa chỉ, ngày công tác, lương. Viết chương trình quản lý nhân viên với các thao tác sau:

- Thêm vào một nhân viên.
- Xem danh sách nhân viên.
- Tìm nhân viên theo mã số.
- Tìm một nhân viên theo tên.
- In ra bảng lương của các nhân viên trong công ty theo thứ tự giảm dần.
- Xóa một nhân viên.

## BÀI TẬP LUYỆN TẬP VÀ NÂNG CAO

Viết chương trình dạng hàm thực hiện các yêu cầu sau:

1. Tính  $S(n) = 1 + 2^2 + 3^3 + \dots + n^n$ , với  $n \geq 0$

2. Tính  $S(n) = \frac{1}{2} + \frac{2}{3} + \frac{3}{4} + \dots + \frac{n}{n+1}$ , với  $n > 0$

3. Tính  $S(n) = 1 + \frac{1+2}{2!} + \frac{1+2+3}{3!} + \dots + \frac{1+2+3+\dots+n}{n!}$ , với  $n > 0$

4. Tính  $S(n) = \sqrt{n + \sqrt{(n-1) + \sqrt{(n-2) + \dots + \sqrt{1}}}}$ , với  $n > 0$

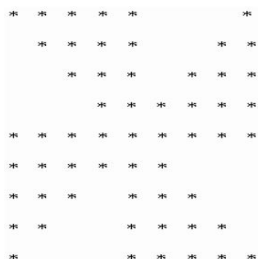
5. Tính  $S(n) = \sqrt{1 + \sqrt{2 + \sqrt{3 + \dots + \sqrt{n}}}}$ , với  $n > 0$

Gợi ý: dùng vòng lặp xác định để tính giá trị từng số hạng rồi cộng vào tổng S. Tại mỗi giá trị biến chạy i của vòng lặp, xác định công thức tính tổng quát cho số hạng theo i.

6. Giải và biện luận phương trình bậc 2:  $ax^2 + bx + c = 0$

7. Giải và biện luận phương trình bậc 4:  $ax^4 + bx^2 + c = 0$

8. Vẽ hình cánh quạt sau:



Gợi ý: Sử dụng các hàm `cprintf()`, `textcolor()`, `delay()`, `kbhit()`, ... thay đổi màu để tạo cảm giác cho cánh quạt xoay cho đến khi nhấn một phím bất kỳ.

9. Nhập vào một ma trận các số nguyên và cho biết ma trận này có đối xứng qua đường chéo chính hay không (các phần tử đối xứng qua đường chéo chính phải bằng nhau)?

Gợi ý:  $a[i][j]$  đối xứng với  $a[j][i]$

10. Viết chương trình nhập vào một ma trận các số nguyên. Sắp xếp ma trận tăng dần theo chiều từ trái sang phải, từ trên xuống dưới.

Gợi ý: Không cần chuyển ma trận về mảng một chiều, xác định công thức liên hệ giữa phần tử  $a[i][j]$  và số thứ tự của phần tử trong ma trận (số thứ tự từ 0,1, 2 ... tính từ trái qua phải và trên xuống dưới trên ma trận).

Ví dụ: xét ma trận  $3 \times 4$ , các giá trị trong ô là số thứ tự phần tử

1	2	3	4
5	6	7	8
9	10	11	12

Gọi  $k$  là số thứ tự của  $a[i][j]$ , cần xác định mối liên hệ giữa  $k$  và  $i, j$  trong ma trận  $m \times n$

11. Tìm giá trị lớn nhất trên từng dòng của ma trận.
12. Tìm giá trị lớn nhất trên từng cột của ma trận.
13. Tìm phần tử trong ma trận gần với  $x$  nhất.
14. Viết chương trình nhập một ma trận vuông cấp  $n$ . Tính tổng tam giác trên, tổng tam giác dưới (kể cả đường chéo).
15. Viết chương trình tìm dòng có tổng lớn nhất trong các dòng và cột có tổng lớn nhất trong các cột của ma trận.
16. Viết chương trình nhập hai ma trận  $A_m \times k$ ,  $B_k \times n$  và tính tích hai ma trận  $A * B$ .

17. Tạo ma trận  $b$  có cùng kích thước với ma trận  $a$  sao cho  $b[i][j] =$  tổng các phần tử lớn nhất dòng  $i$  và nhỏ nhất cột  $j$ .
18. Tạo mảng  $Max$  chứa các giá trị lớn nhất trên từng dòng.
19. Tạo mảng  $Min$  chứa các giá trị nhỏ nhất trên từng cột.
20. Sắp xếp ma trận  $a$  tăng theo cột/ tăng theo dòng.
21. Tìm phần tử lớn nhất dòng  $i$  và phần tử nhỏ nhất cột  $j$ .
22. Cho 2 mảng số nguyên  $a$  và  $b$  kích thước lần lượt là  $n$  và  $m$ . Viết chương trình nối 2 mảng trên thành mảng  $c$  theo nguyên tắc chẵn ở đầu mảng và lẻ ở cuối mảng.

Ví dụ: Mảng  $a$ : 3 2 7 5 9 Mảng  $b$ : 1 8 10 4 12 6

Mảng  $c$ : 6 12 4 10 2 8 3 1 7 5 9

23. Viết hàm tính tổng của từng dãy con giảm có trong mảng.
24. Cho mảng các số nguyên  $a$  gồm  $n$  phần tử ( $30000 \leq n$ ) và số dương  $k$  ( $nk \leq$ ). Hãy chỉ ra số hạng lớn thứ  $k$  của mảng.

Ví dụ: Mảng  $a$ : 6 3 1 10 11 18

$k = 2$  Kết quả: 10

25. Viết hàm liệt kê các bộ 4 số  $a, b, c, d$  trong mảng các số nguyên (có ít nhất 4 phần tử và đôi một khác nhau) sao cho  $a + b = c + d$ .
26. Viết hàm tìm và xóa tất cả các phần tử trùng với  $x$  trong mảng một chiều các số nguyên, nếu không tồn tại phần tử  $x$  trong mảng thì trả về -1.
27. Viết hàm xóa tất cả những phần tử trùng nhau trong dãy chỉ giữ lại một phần tử trong đó.

Ví dụ: 1 6 2 3 2 4 2 6 5 1 1 6 2 3 4 5

28. Viết hàm xóa các khoảng trắng ở đầu chuỗi, ở cuối chuỗi.
29. Không dùng hàm `strlwr()` hãy viết chương trình đổi chuỗi ra thành chuỗi thường.
30. Không dùng hàm `strupr()` hãy viết chương trình đổi chuỗi ra thành chuỗi hoa.
31. Viết hàm đếm số từ trong một chuỗi.
32. Viết chương trình chèn chuỗi  $S2$  vào vị trí thứ  $i$  của chuỗi  $S1$ .

33. Viết hàm tìm số lần xuất hiện của chuỗi S2 trong chuỗi S1.
  34. Viết hàm kiểm tra một chuỗi S có đối xứng hay không
  35. Viết chương trình nhập số nguyên dương n gồm k chữ số ( $k \leq 50$ ), kiểm tra xem các chữ số của n có được sắp thứ tự không.  
Ví dụ: Nhập n=1569 hoặc n=8521      Kết quả: Có thứ tự.
  36. Viết chương trình in ra màn hình ngày/tháng/năm của ngày hiện tại, cho phép sử dụng các phím mũi tên lên, xuống để tăng hoặc giảm một ngày.
  37. Viết chương trình in ra màn hình giờ:phút:giây hiện tại, cho phép sử dụng các phím mũi tên lên, xuống để tăng hoặc giảm một giây.
  38. Viết chương trình tạo một mảng các số phức. Hãy viết hàm tính tổng, tích các số phức có trong mảng.
  39. Viết chương trình tạo một mảng các phân số. Hãy viết hàm thực hiện các công việc sau :
    - Tính tổng tất cả các phân số (kết quả dưới dạng phân số tối giản)
    - Tìm phân số lớn nhất, phân số nhỏ nhất.
    - Sắp xếp mảng tăng dần.
  40. Tổ chức dữ liệu để quản lí sinh viên bằng cấu trúc mẫu tin trong một mảng N phần tử, mỗi phần tử có cấu trúc như sau:
    - Mã sinh viên.
    - Tên.
    - Năm sinh.
    - Điểm toán, lý, hoá, điểm trung bình.
- Viết chương trình thực hiện những công việc sau:
- Nhập danh sách các sinh viên cho một lớp học.
  - Xuất danh sách sinh viên ra màn hình.
  - Tìm sinh viên có điểm trung bình cao nhất.
  - Sắp xếp danh sách lớp theo thứ tự tăng dần của điểm trung bình.
  - Sắp xếp danh sách lớp theo thứ tự giảm dần của điểm toán.

- Tìm kiếm và in ra các sinh viên có điểm trung bình lớn hơn 5 và không có môn nào dưới 3.
- Tìm sinh viên có tuổi lớn nhất.
- Nhập vào tên của một sinh viên. Tìm và in ra các thông tin liên quan đến sinh viên đó (nếu có).

41. Tổ chức dữ liệu quản lý danh mục các bộ phim VIDEO, các thông tin liên quan đến bộ phim này như sau:

- Tên phim (tựa phim).
- Thể loại (3 loại : hình sự, tình cảm, hài).
- Tên đạo diễn.
- Tên diễn viên nam chính.
- Tên diễn viên nữ chính.
- Năm sản xuất.
- Hãng sản xuất

Viết chương trình thực hiện những công việc sau:

- Nhập vào bộ phim mới cùng với các thông tin liên quan đến bộ phim này.
- Nhập một thể loại: In ra danh sách các bộ phim thuộc thể loại này.
- Nhập một tên nam diễn viên. In ra các bộ phim có diễn viên này đóng.
- Nhập tên đạo diễn. In ra danh sách các bộ phim do đạo diễn này dàn dựng.

42. Viết chương trình tạo một file chứa 10000 số nguyên ngẫu nhiên đôi một khác nhau trong phạm vi từ 1 đến 32767 và đặt tên là “SONGUYEN.INP”.

43. Viết chương trình tạo một file chứa các số nguyên có tên SONGUYEN.INP. Sau đó đọc file SONGUYEN.INP và ghi các số chẵn vào file SOCHAN.OUT và những số lẻ vào file SOLE.OUT.

44. Viết chương trình ghi vào tập tin SOCHAN.DAT các số nguyên chẵn từ 0 đến 100.
45. Cho mảng các số nguyên, tính tổng các phần tử của mảng. Dữ liệu vào : tập tin văn bản ARRAY.INP gồm hai dòng
- Dòng 1 chứa số nguyên  $n$  ( $n \leq 10$ )
  - Dòng 2 chứa  $n$  số nguyên

Kết quả : Đưa ra tập tin văn bản ARRAY.OUT gồm một dòng ghi tổng các phần tử trong mảng.

46. Tạo file văn bản có tên là “INPUT.TXT” có cấu trúc như sau:
- Dòng đầu tiên ghi  $N$  ( $N$  là số nguyên dương nhập từ bàn phím).
  - Trong các dòng tiếp theo ghi  $N$  số nguyên ngẫu nhiên trong phạm vi từ 0 đến 100, mỗi dòng 10 số (các số cách nhau ít nhất một khoảng trống).

Hãy đọc dữ liệu của file “INPUT.TXT” và lưu vào mảng một chiều  $A$ .

Thực hiện các công việc sau :

- Tìm giá trị lớn nhất của mảng  $A$ .
- Đếm số lượng số chẵn, số lượng số lẻ của mảng  $A$ .
- Hãy sắp xếp các phần tử theo thứ tự tăng dần.

Hãy ghi các kết quả vào file văn bản có tên OUTPUT.TXT theo mẫu sau:

INPUT.TXT	OUTPUT.TXT
18	Cau a: 99
87 39 78 19 89 4 40 98 29	Cau b: 9 9
65	Cau c:
20 43 1 99 38 34 58 4	1 4 4 19 20 29 34 38 39 40
	43 58 65 78 87 89 98 99

47. Tạo một file text có tên là “INPUT.TXT” có cấu trúc như sau :
- Dòng đầu tiên ghi hai số  $M$  và  $N$  ( $M, N$  là hai số nguyên dương nhập từ bàn phím).

- Trong M dòng tiếp theo mỗi dòng ghi N số nguyên ngẫu nhiên trong phạm vi từ 0 đến 100 (các số này cách nhau ít nhất một khoảng trắng).

Hãy đọc dữ liệu từ file trên và lưu vào mảng hai chiều. Rồi thực hiện các công việc sau:

- Tìm giá trị lớn nhất của ma trận.
- Đếm số lượng số chẵn , lẻ, nguyên tố có trong ma trận.
- Hãy tính tổng các phần tử trên mỗi dòng của ma trận.

Hãy ghi kết quả này vào filetext có tên là “OUTPUT.TXT”

INPUT.TXT						OUTPUT.TXT					
6	6					Cau a:	49				
41	17	33	23	12	1	Cau b:	17	19			
44	24	23	49	5	24	Cau c:	127	169	147	214	132
33	20	17	25	33	19		146				
0	48	45	48	41	32						
10	24	36	19	19	24						
30	4	23	26	27	36						



## PHỤ LỤC A. MỘT SỐ HÀM CHUẨN TRONG C/C++

### 1. Thư viện `stdio.h`

Hàm	Ý nghĩa hàm
<code>printf</code>	Xuất dữ liệu theo định dạng
<code>scanf</code>	Nhập dữ liệu theo định dạng
<code>putchar</code>	Xuất ký tự
<code>getchar</code>	Nhập ký tự
<code>puts</code>	Xuất chuỗi
<code>gets</code>	Nhập chuỗi
<code>fflush(stdin)</code>	xoá vùng đệm bàn phím

### 2. Thư viện `conio.h`

Hàm	Ý nghĩa hàm
<code>clrscr</code>	Xóa màn hình
<code>gotoxy</code>	Đưa điểm nháy đến toạ độ x,y trên màn hình
<code>textcolor</code>	đặt màu chữ mới
<code>textbackground</code>	Đặt màu nền mới
<code>getch</code>	Nhập ký tự( không hiện ra màn hình)
<code>getche</code>	Nhập ký tự(có hiện ra màn hình)

### 3. Thư viện `stdlib.h`

Hàm	Ý nghĩa hàm
<code>randomize()</code>	Khởi động cơ chế tạo số ngẫu nhiên
<code>random(n)</code>	Trả về số nguyên trong khoảng 0 đến n-1
<code>rand()</code>	Cho một giá trị ngẫu nhiên trong khoảng 1 đến
<code>flushall()</code>	Xoá vùng đệm bàn phím, lệnh này thường được sử

#### 4. Thư viện ctype.h : Thư viện xử lý ký tự

Các hàm sau sẽ thực hiện việc kiểm tra 1 thao tác trên ký tự, nếu kết quả kiểm tra là sai thì trả về là 0, ngược lại là giá trị khác 0.

Cú pháp hàm	Ý nghĩa hàm
int isalpha(int c)	Kiểm tra c có phải là ký tự chữ. (A ... Z, a ... z), với c là giá trị mã ASCII của ký tự.
int isascii(int c)	Kiểm tra c có phải là ký tự ASCII. (giá trị c thuộc khoảng 0 ... 127).
int isctrl(int c)	Kiểm tra c có phải là ký tự điều khiển.
int isdigit(int c)	Kiểm tra c có phải là ký tự số 0 ... 9.
int isgraph(int c)	Kiểm tra c có phải là ký tự in được, trừ khoảng trống.
int islower(int c)	Kiểm tra c có phải là ký tự thường.
int isprintf(int c)	Kiểm tra c có phải là ký tự in được.
int ispunct(int c)	Kiểm tra c có phải là ký tự điều khiển hay khoảng trống.
int isspace(int c)	Kiểm tra c có phải là ký tự khoảng trống.
int isupper(int c)	Kiểm tra c có phải là ký tự in hoa? (A ... Z).

#### 5. Thư viện math.h: Các hàm toán học

Cú pháp hàm	Ý nghĩa hàm
double acos(double x)	Tính arc cosine(x).
double asin(double x)	Tính arc sine(x).
double atan(double x)	Tính arc tangent(x).
double atan2(double x, double y)	Tính arc tangent(x/y).
double ceil(double x)	Trả về số nguyên(có kiểu double) nhỏ nhất và không nhỏ hơn x.
double cos(double x)	Tính cosine (x), x: radian.
double cosh(double x)	Tính hyperbolic cosine (x).
double exp(double x)	Tính $e^x$ .
double fabs(double x)	Tính $ x $ .
double floor(double x)	Trả về số nguyên lớn nhất và không lớn hơn x.
double fmod(double x, double y)	Trả về số dư(kiểu double) của phép chia nguyên x/y.

double frexp(double x, int* exponent)	Chia x làm thành phần định trị (mantisa) và lũy thừa (exponent) của 2 ( $x = a * 2^{\text{exponent}}$ ) trả về giá trị a.
double ldexp(double x, int exp)	Ngược lại với frexp, trả về $x * 2^{\text{exp}}$
double log(double x)	Trả về giá trị log Neper của x.
double log10(double x)	Trả về giá trị log 10 của x.
double modf(double x, double* intptr)	Chia x thành phần lẻ (fractional – kết quả của hàm) và phần nguyên
double pow(double x, double y)	Tính $x^y$
double sin(double x)	Tính $\sin(x)$ , x: radian
double sinh(double x)	Tính hyperbolic sine của x.
double sqrt(double x)	Tính căn bậc 2 của x.
double tan(double x)	Tính tangent của x.
double tanh(double x)	Tính hyperbolic tangent của x.

## 6. Các hàm xuất nhập chuẩn:

Các hàm xuất nhập dữ liệu thông thường

Cú pháp hàm	Ý nghĩa hàm
int getchar(void)	Nhập một kí tự từ bên phím (stdin).
int putchar(int c)	Xuất kí tự ra màn hình (stdout).
char* gets(char*s)	Nhập một chuỗi kí tự từ bên phím
int puts(const char*s)	Xuất một chuỗi kí tự ra màn hình (có xuống dòng)
int printf (const char* format, [argument,...])	Xuất dữ liệu có định dạng ra màn hình.
int scanf (const char* format, [address, ...])	Nhập dữ liệu có định dạng ra màn hình.
int sprintf (char*buffer, const char*format[argument, ...])	Xuất dữ liệu có định dạng sang 1 chuỗi.
int sscanf(const char*buffer, const char*format[argument, ...])	Đọc một chuỗi.

d)	
int vprintf (const char* format, va_list arlist)	Xuất dữ liệu định dạng dùng danh sách đối số.
int vscanf (const char* format, va_list arlist)	Nhập dữ liệu định dạng dùng danh sách đối số.
int vsprintf(char*buffer,const char*format, va_list arlist)	Xuất dữ liệu định dạng ra chuỗi dùng danh sách đối số.
int vsscanf (const char* buffer,const char*format, va_list arlist)	Nhập dữ liệu định dạng vào chuỗi dùng danh sách đối số.

## 7. Xuất nhập file

Cú pháp hàm	Ý nghĩa hàm
void clearer (FILE*stream)	Xóa thông báo lỗi.
int fclose(FILE*stream)	Đóng file.
int fcloseall(void)	Đóng tất cả các file đang mở.
FILE* fdopen(int handle, char*type)	Gán 1 dòng(stream) cho file handle.
FILE* fopen(const char*filename, char*type)	Mở một file.
FILE* freopen(const char*filename,const char * mode,FILE*stream)	Mở một file mới và gán cho 1 file handle đã mở.
int feof(FILE*stream)	Kiểm tra hết file(Macro).
int ferror(FILE*stream)	Kiểm tra có lỗi hay không.
int fflush(FILE*stream)	Ghi buffer của dòng(stream) lên file.
int fgetc(FILE*stream)	Lấy kí tự từ file.
int fputc(int c, FILE*stream)	Ghi kí tự c lên file.
int fgetchar(void)	Lấy kí tự từ thiết bị nhập chuẩn(stdin).
int fputchar(int c)	Xuất lí tự ra thiết bị xuất chuẩn(stdout).
int getpos(FILE*stream, fpos_t*pos)	Lấy vị trí hiện hành.
int fsetpos(FILE*stream, const	Ấn định vị trí file hiện hành

fpos_t*pos)	
char* fgets(char*s, int n, FILE*stream)	Lấy một chuỗi từ file.
int fputs(const char*s, FILE*stream)	Ghi một chuỗi lên file.
int fileno(FILE*stream)	Lấy file handle.
int fflushall(void)	Xô các buffer của dòng nhập.
int fprintf(FILE*stream, const char*format, [argument,... ])	Ghi kết xuất có định dạng lên file.
int fscanf(FILE*stream, const char* format, [address,... ])	Đọc dữ liệu có định dạng từ file.
size_t fread(void *ptr, size_t size, size n, FILE*stream)	Đọc dữ liệu từ file.
size_t fwrite(const void *ptr, size_t size, size n, FILE*stream)	Ghi dữ liệu lên file.
int fseek(FILE*stream, long offset, int whence)	Nhảy đến vị trí offset trong file kể từ vị trí whence.
long ftell(FILE*stream)	Lấy vị trí file hiện hình.
int getw(FILE*stream)	Đọc một số nguyên từ file.
int putw(int w, FILE*stream)	Xuất một số nguyên từ file.
void perror(const char*s)	Xuất một thông báo lỗi hệ thống.
int remove(const char*filename)	Macro xóa một file.
int rename(const char*oldname, const char*newname)	Đổi tên một file.
void rewind(FILE*stream)	Đưa con trỏ về đầu file.
int rmtmp(void)	Xoá các file tạm đã mở.

## 8. Gán buffer cho một file

Cú pháp hàm	Ý nghĩa hàm
FILE*tmpfile(void)	Mở một file tạm nhị phân
char* tempnam(char*dir, char *prefix)	Tạo một file có tên duy nhất trong thư mục
int unget(int c, FILE*stream)	Trả kí tự về cho file

int unlink(const char*filename)	Xóa một file
char*_strerror(const char*s);	Tạo thông báo lỗi từ chuỗi
char*strerror(int errnum);	Tạo thông báo lỗi từ số.

## 9. Đổi chuỗi thành số

Cú pháp hàm	Ý nghĩa hàm
double atof(const char*s)	Chuyển chuỗi s dạng số thành số thực dạng double
long double atold(const char*s)	Chuyển chuỗi s dạng số thành số thực dạng long double
int atoi(const char*s)	Chuyển chuỗi s dạng số thành số nguyên dạng int

## 10. Xử lý số

Cú pháp hàm	Ý nghĩa hàm
int abs(int x)	Lấy trị tuyệt đối của số nguyên int.
double fabs(double x)	Lấy trị tuyệt đối số thực.
long labs(long int x);	Lấy trị tuyệt đối của số nguyên long int.
void randomize(void);	Khởi động cơ chế lấy số ngẫu nhiên.
int rand(void)	Lấy số ngẫu nhiên từ 0 đến RAND_MAX.
int random(int num)	Lấy số ngẫu nhiên từ 0 đến num – 1.

## 11. Cấp phát và thu hồi bộ nhớ:

Cú pháp hàm	Ý nghĩa hàm
void * malloc (size_t size).	Hàm xin cấp phát một vùng nhớ có size bytes.
void far*farmalloc(unsigned long nbytes)	Hàm xin cấp phát một vùng nhớ có size nbytes byte
void* calloc (size_t nitems, size_t size)	Hàm xin cấp phát một vùng nhớ cho nitems phần tử, mỗi phần tử có size byte.
void far*farcalloc(unsigned long nitems, unsigned long size);	Hàm xin cấp phát một vùng nhớ cho nitems phần tử, mỗi phần tử có size byte.

<code>void * realloc(void * oldblock, size_t size).</code>	Hàm xin cấp phát lại vùng nhớ lúc trước đã cấp phát rồi ở địa chỉ oldblock với kích thước mới là size, có copy nội dung cũ sang vùng nhớ mới.
<code>void far* farrealloc(void far* oldblock, unsigned long nbytes).</code>	Hàm xin cấp phát lại vùng nhớ lúc trước đã cấp phát rồi ở địa chỉ oldblock với kích thước mới là nbytes, có copy nội dung cũ sang vùng nhớ mới
<code>void * free(void*block).</code>	Hàm trả về bộ nhớ cho hệ thống.
<code>void far* farfree(void far*block).</code>	Hàm trả về bộ nhớ cho hệ thống.

## 12. Xử lý string

Cú pháp hàm	Ý nghĩa hàm
<code>void *memcpy (void *dest, const void *src, size_t n);</code>	Copy một khối bộ nhớ n bytes từ src sang dest.
<code>void *memchr (const void *s, int c, size_t n)</code>	Tìm kiếm kí tự c trong khối bộ nhớ s(n byte).
<code>int memcmp (const void *s1, const void *s2, size_t n);</code>	So sánh n byte đầu tiên giữa hai chuỗi s1 và s2, có phân biệt chữ hoa, chữ thường.
<code>int memicmp(const void *s1, const void *s2, size_t n)</code>	So sánh n byte đầu tiên giữa hai chuỗi s1 và s2, không phân biệt chữ hoa, chữ thường.
<code>void *memset (void *s, int c, size_t n)</code>	Cho n byte đầu tiên của s đều là kí tự c.
<code>char *strcat(char *dest, const char *src);</code>	Nối chuỗi src với chuỗi dest.
<code>char *strchr(const char *s, int c);</code>	Tìm địa chỉ vị trí xuất hiện đầu tiên của kí tự c trong chuỗi s.
<code>int strcmp(const char *s1, const char*s2)</code>	So sánh hai chuỗi, hàm trả về >1 nếu chuỗi s1>s2, trả về 0 nếu 2 chuỗi bằng nhau, và trả về -1 nếu s1<s2. Hàm có phân biệt chữ hoa, chữ thường.

<code>int strcmpi(const char *s1, const char *s2)</code>	So sánh hai chuỗi, hàm trả về >1 nếu chuỗi s1>s2, trả về 0 nếu 2 chuỗi bằng nhau, và trả về -1 nếu s1<s2. Hàm không phân biệt chữ hoa, chữ thường.
<code>char *strcpy(char *dest, const char *src);</code>	Copy chuỗi src sang chuỗi dest
<code>size_t strcspn(const char *s1, const char *s2)</code>	Tìm đoạn trong chuỗi s1 không chứa các kí tự có trong s2
<code>size_t strspn(const char *s1, const char *s2)</code>	Tìm đoạn trong chuỗi s1 có chứa các kí tự có trong s2.
<code>char *strdup(const char *s)</code>	Copy một chuỗi sang vị trí khác.
<code>size_t strlen(const char *s)</code>	Tìm độ dài của chuỗi.
<code>char *strlwr(char *s)</code>	Đổi một chuỗi thành chuỗi chữ thường.
<code>char *strupr(char *s)</code>	Đổi một chuỗi thành chuỗi chữ hoa.
<code>char *strncat(char *dest, const char *src, size_t maxlen);</code>	Nối một đoạn của chuỗi src vào chuỗi dest.
<code>int strncmp(const char *s1, const char*s2);</code>	So sánh hai chuỗi.
<code>char *strncpy(char *dest, const char *src, size_t maxlen)</code>	Copy tối đa maxlen kí tự từ chuỗi src sang chuỗi dest.
<code>char *strnset(char *s, int ch, size_t n)</code>	Cho n byte của chuỗi s lên kí tự ch.
<code>char *strpbrk(const char *s1, const char *s2)</code>	Tìm xuất hiện đầu tiên của kí tự bất kỳ của chuỗi s2 trong chuỗi s1.
<code>char *strchr(const char *s, int c)</code>	Tìm xuất hiện cuối cùng của kí tự c trong chuỗi s.
<code>char *strrev(char *s)</code>	Đảo chuỗi.
<code>char *strset(char *s, int ch)</code>	Thiết lập tất cả chuỗi s đều mang kí tự ch.
<code>char *strstr(const char *s1, const char *s2)</code>	Tìm xuất hiện đầu tiên của chuỗi s2 trong s1.
<code>char *strtok(char *s1, const char *s2)</code>	Tìm từ đầu tiên trong s1 không có mặt trong s2.



## **PHỤ LỤC B. HƯỚNG DẪN TÌM LỖI VÀ SỬA LỖI CHƯƠNG TRÌNH (DEBUG)**

### **I. Khái niệm Debug**

Debug là quá trình tìm kiếm và sửa lỗi của chương trình. Quá trình debug thường mất nhiều thời gian hơn việc viết chương trình, vì những lỗi liên quan đến logic thường rất khó phát hiện. Visual Studio cung cấp cho chúng ta khá nhiều công cụ trực quan để đơn giản hoá quá trình debug, giúp ta tiến hành tìm và sửa lỗi chương trình một cách dễ dàng hơn.

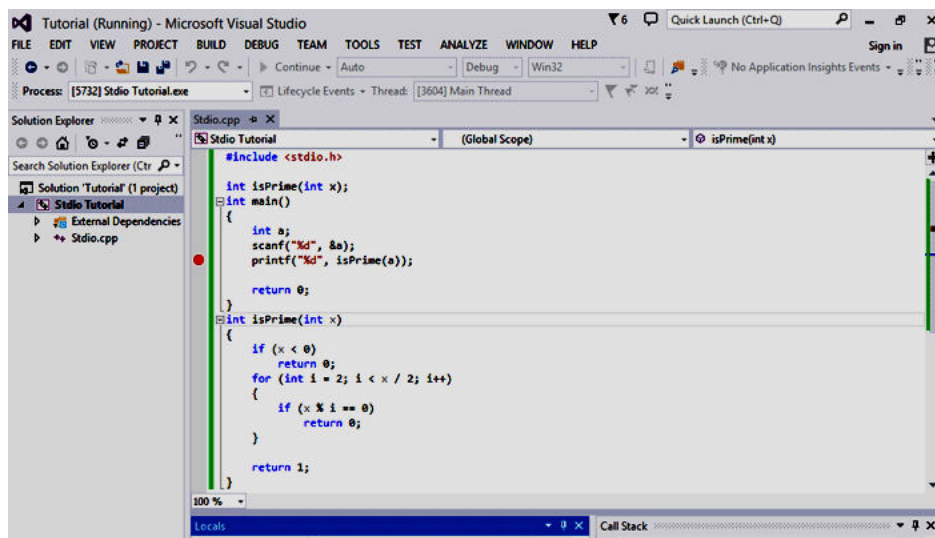
### **II. Một số khái niệm liên quan**

#### **1. Chạy chương trình**

Một chương trình sẽ lần lượt duyệt qua các dòng code có trong hàm main. Trong đó, có những đoạn “thẳng”, có những đoạn “ rẽ nhánh” (if/switch), có những đoạn “vòng” (for/while/...), có những đoạn đường nhỏ hơn (các function). Để sửa lỗi chương trình, cần tìm ra được những “đoạn rối” của nó, tức là xác định xem đoạn code ở vị trí nào có khả năng phát sinh lỗi.

#### **2. Breakpoints**

Breakpoints là vị trí mà chương trình sẽ dừng lại để lập trình viên xem xét sự thay đổi của các biến qua từng dòng lệnh, từ đó phát hiện ra vị trí dòng code bị lỗi. Breakpoint được kí hiệu bằng chấm tròn màu đỏ ở đầu dòng code. Để tạo ra một breakpoint, cách đơn giản nhất là click chuột vào đầu dòng code (như trong hình). Để huỷ breakpoint, chỉ cần click chuột vào breakpoint đó một lần nữa. Ngoài ra các bạn cũng có thể tạo/huỷ breakpoint bằng phím F9.

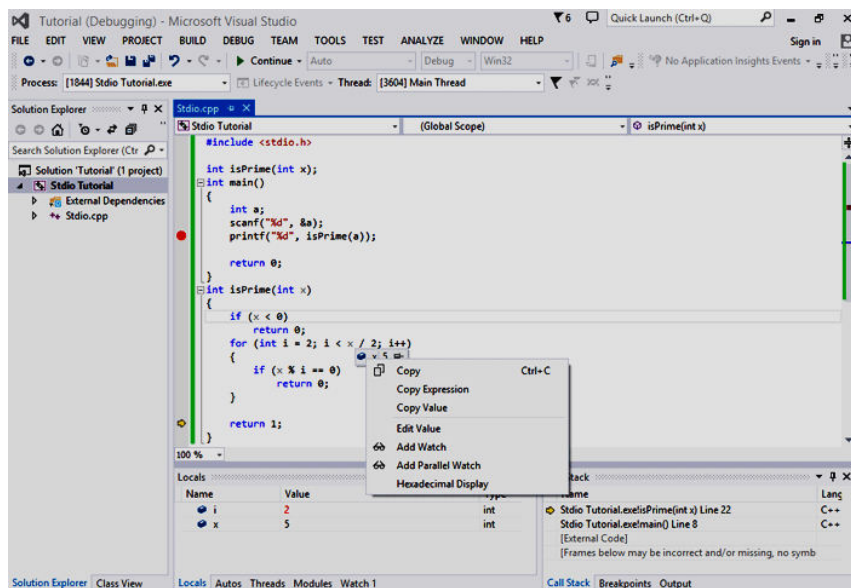


### 3. Watch Windows

Trong quá trình Debug, sau khi chương trình dừng lại ở một breakpoint nào đó, điều cần thiết của chúng ta là kiểm tra sự thay đổi giá trị của các biến, hàm,... qua từng dòng code để nhìn ra sai sót trong thuật toán. Watch windows là tập hợp các công cụ giúp giúp lập trình viên quan sát được giá trị hiện tại của biến. Các cửa sổ này có thể được tìm thấy trong menu Debug → Windows (chỉ xuất hiện khi đang trong quá trình debug).

### 4. Data Tip

Khi di chuyển con trỏ chuột đến tên biến ở bất kỳ vị trí nào trong phạm vi cặp dấu `{ }` (scope) hiện tại, giá trị của biến sẽ được hiển thị trên màn hình. Khi đó, các bạn có thể “ghim” biến đó lên màn hình để tiện quan sát, hoặc add vào cửa sổ Watch, copy giá trị, thay đổi giá trị của biến, ...



## 5. Locals

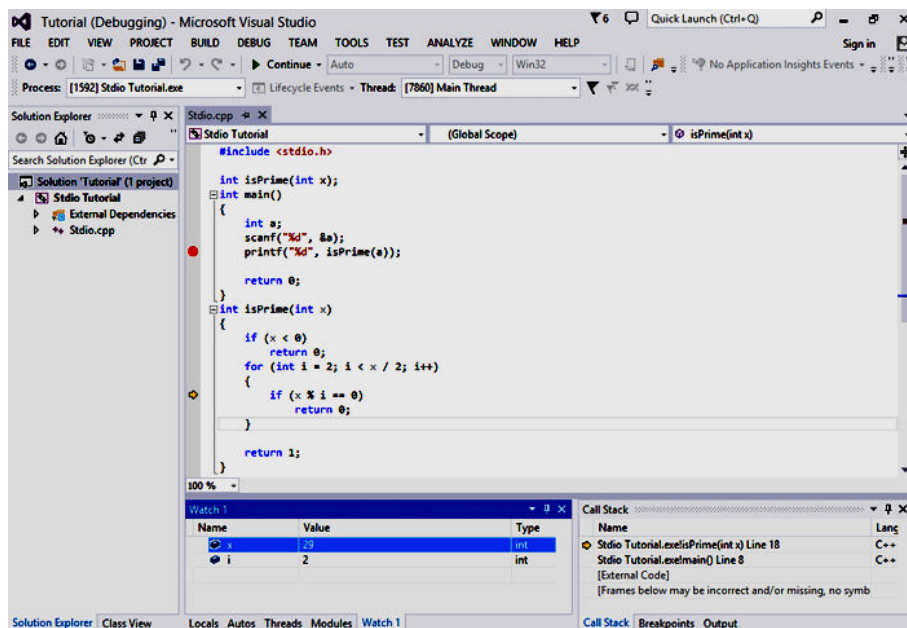
Cửa sổ Locals sẽ hiển thị tất cả các biến có liên quan đến dòng code hiện tại một cách tự động. Các biến hiển thị ở đây sẽ được thay đổi qua từng dòng code. Ngoài ra, màu sắc của các biến giúp ta phân biệt được những biến nào vừa thay đổi giá trị.

## 6. Autos

Tương tự như Locals, cửa sổ Autos hiển thị các biến vừa được sử dụng trong các dòng code trước. Visual Studio sẽ tự động nhận diện biến nào không còn cần thiết và loại bỏ ra khỏi cửa sổ Autos.

## 7. Watch



Visual Studio không thể nhận diện được tất cả những gì lập trình viên cần. Trong một số trường hợp, chúng ta cần theo dõi cụ thể một giá trị nào đó, chẳng hạn như phần tử thứ 10 trong một mảng số nguyên hay ký tự thứ 5 của chuỗi “Stdio Tutorial”, ... Chúng ta sẽ cần sử dụng đến cửa sổ Watch. Nó cho phép lập trình viên nhập vào tên biến, hàm, ... cụ thể trong scope hiện tại. Giá trị của các biến, hàm sẽ được hiển thị bên cạnh tên biến.




## 8. Debug chương trình

Trước khi debug, các bạn cần tạo ra các breakpoints cần thiết để tìm và sửa lỗi chương trình. Để bắt đầu tiến hành debug một chương trình, các bạn vào menu Debug → Start Debugging hoặc nhấn phím F5 trên bàn phím. Visual Studio sẽ tiến hành Build chương trình. Sau khi Build xong và không có lỗi biên dịch (compile error), cửa sổ Debug sẽ xuất hiện. Nếu không có các lệnh dừng màn hình chờ nhập dữ liệu (như scanf, gets, ...), chương trình sẽ dừng lại tại breakpoint đầu tiên.

Một thanh công cụ Debug sẽ xuất hiện. Trong đó, có các nút công cụ quan trọng như sau:

-  **Step Over:** Chạy step by step, lướt qua hàm (chỉ nhận giá trị return của hàm).
-  **Step Into:** Chạy step by step, đi vào nội dung của các hàm con.

-  **Step Out:** “Nhảy” đến breakpoint kế tiếp. Nếu không còn breakpoint nào thì sẽ kết thúc debug. Ngoài ra nó còn có chức năng chạy lướt qua hàm con hiện tại.
- **Một số phím tắt hữu ích**
- **F5:** Bắt đầu quá trình Debug.
- **Shift + F5:** Thoát Debug.
- **Ctrl + F5:** Chạy chương trình không dùng công cụ Debug
- **F9:** Tạo/hủy một breakpoint.
- **F10:** Step Over.
- **F11:** Step Into.
- **Shift + F11:** Step Out.

**TÀI LIỆU THAM KHẢO**

- [1] Khoa CNTT, *Ngôn ngữ lập trình*, Trường ĐH CNTT TPHCM, 2003
- [2] Nguyễn Thanh Thủy (chủ biên), *Nhập môn lập trình ngôn ngữ C*, NXB Khoa học và Kỹ thuật, Hà Nội, 2005.
- [3] Nguyễn Đình Tê, Hoàng Đức Hải, *Giáo trình Lý thuyết và bài tập ngôn ngữ C Tập 1*, NXB Giáo dục, Hà Nội, 1999.
- [4]. Nguyễn Đình Tê, Hoàng Đức Hải, *Giáo trình Lý thuyết và bài tập ngôn ngữ , Tập 2*, NXB Giáo dục, Hà Nội, 1999.
- [5] Nguyễn Thanh Thủy và Nguyễn Quang Huy, *Bài tập lập trình ngôn ngữ C*, NXB Khoa học và Kỹ thuật, Hà Nội, 1999.
- [6] Brian.W.Kernighan, Dennis M.RitChie, *C Programming Language*, 2<sup>nd</sup> Edition, Prentice Hall Software Series.
- [7] [http://publications.gbdirect.co.uk/c\\_book/](http://publications.gbdirect.co.uk/c_book/) : *The C Book*, Mike Banahan, Declan Brady and Mark Doran (the online version).
- [8] Địa chỉ một số website tham khảo của ngôn ngữ lập trình C/C++:  
<https://en.cppreference.com>  
<http://www.cplusplus.com/>  
<https://www.cprogramming.com/reference/>  
<https://docs.microsoft.com/en-us/cpp/c-language/>  
<https://www.learncpp.com>