



NHẬP MÔN LẬP TRÌNH NGÔN NGỮ LẬP TRÌNH C

BÀI GIẢNG LƯU HÀNH NỘI BỘ

KHOA CÔNG NGHỆ ĐIỆN TỬ
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP THÀNH PHỐ HỒ CHÍ MINH

BỘ MÔN ĐIỆN TỬ MÁY TÍNH

MỤC LỤC

| | |
|--|----|
| BÀI 1. GIỚI THIỆU VỀ LẬP TRÌNH VÀ CHƯƠNG TRÌNH (GV. Trần Hồng Vinh) | 1 |
| BÀI 2 CƠ BẢN VỀ GIẢI THUẬT (GV. Nguyễn Văn Duy) | 8 |
| BÀI 3: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C (GV. Nguyễn Thanh Đăng) | 21 |
| BÀI 4. CẤU TRÚC RỄ NHÁNH CÓ ĐIỀU KIỆN (GV. Đinh Quang Tuyền) | 33 |
| BÀI 5: CẤU TRÚC VÒNG LẶP (GV. Vũ Thị Hồng Nga) | 42 |
| BÀI 6. HÀM (GV. Lê Lý Quyên Quyên) | 50 |
| BÀI 7: CẤU TRÚC DỮ LIỆU KIỂU MẢNG (GV. Trần Hồng Vinh) | 57 |
| BÀI 8A: KIỂU DỮ LIỆU CẤU TRÚC (GV. Phan Tuấn Anh) | 81 |
| BÀI 8B. LẬP TRÌNH ỨNG DỤNG – ARDUINO (GV. Ong Mẫu Dũng) | 89 |

BÀI 1. GIỚI THIỆU VỀ LẬP TRÌNH VÀ CHƯƠNG TRÌNH

Mục tiêu

Sau khi hoàn tất bài này học viên sẽ hiểu các kiến thức kỹ năng cơ bản sau:

- Các khái niệm cơ bản về lập trình và chương trình.
- Các bước xây dựng chương trình.
- Công cụ viết chương trình

Mở đầu:

C là ngôn ngữ lập trình được thiết kế bởi Dennis Ritchie tại phòng thí nghiệm Bell Telephone năm 1972. Nó được viết với mục tiêu chính là xây dựng hệ điều hành UNIX. Vì thế ban đầu nó không hướng tới sự tiện dụng cho người lập trình. C được phát triển từ một ngôn ngữ lập trình có tên là B (B là ngôn ngữ lập trình được viết bởi Ken Thompson tại Bell Labs, và tên ngôn ngữ lấy theo tên của Bell Labs).

C là ngôn ngữ mạnh và mềm dẻo, linh hoạt, nó nhanh chóng trở thành ngôn ngữ phổ biến không chỉ trong phạm vi của Bell, C được các lập trình viên sử dụng viết nhiều loại ứng dụng ở các mức độ khác nhau. Cũng vì nó được dùng nhiều nơi nên xuất hiện những đặc điểm khác nhau, các phiên bản phát triển không thống nhất. Để giải quyết vấn đề này, năm 1983 Viện tiêu chuẩn Mỹ (ANSI) đã thành lập một chuẩn cho C và có tên ANSI C (ANSI standard C). Nói chung các chương trình dịch C ngày nay đều tuân theo chuẩn này ngoại trừ một số khác biệt nhỏ.

C được phổ biến bởi nó có các đặc điểm sau:

- C là ngôn ngữ mạnh và mềm dẻo. Có thể nói rằng sự hạn chế của C chỉ phụ thuộc vào người lập trình, tức là với C bạn có thể làm tất cả những điều theo ý tưởng của bạn. C được dùng cho những dự án từ nhỏ tới lớn như: Hệ điều hành, Đồ họa, Chương trình dịch,...
- C dễ chuyển đổi sang hệ thống khác (tính khả chuyển), tức là một chương trình C được viết trên hệ thống này có thể dễ dàng dịch lại chạy được trên hệ thống khác
- C là ngôn ngữ cô đọng, số lượng từ khóa không nhiều.
- C là ngôn ngữ lập trình cấu trúc. Mã lệnh của chương trình C được viết thành các hàm, các hàm này có thể sử dụng lại trong các ứng dụng khác.

Với các đặc điểm trên C là ngôn ngữ tốt cho việc học lập trình, hơn nữa sau này chúng ta còn có thể tiếp cận với lập trình hướng đối tượng, và một trong những ngôn ngữ lập trình chúng ta lựa chọn đầu tiên cho lập trình hướng đối tượng là Java, Python.

1. Tổng quan về lập trình:

a. Chương trình (Program):

- Chương trình là tập hợp các chỉ thị lệnh yêu cầu máy tính thực thi một tác vụ cụ thể.
- Một cách chi tiết, chương trình là một bộ các hướng dẫn cho máy tính biết phải làm gì.
- Nếu chúng ta thay đổi chương trình máy tính sẽ thực hiện một bộ các hành động hoặc nhiệm vụ khác.

b. Ngôn ngữ lập trình (Programming Language):

- Là một ngôn ngữ nhân tạo

- Gồm một tập các ký hiệu và cú pháp được chuẩn hóa để mô tả những xử lý mà người và máy đều có thể hiểu được



Ngôn ngữ máy

Ngôn ngữ lập trình

Ngôn ngữ tự nhiên.

Lập trình (Programming): Là quá trình xây dựng các chương trình nguồn được viết bằng một hoặc nhiều ngôn ngữ lập trình

Tại sao nên học lập trình? Vì lập trình là một phần cơ bản của rất nhiều ngành kỹ thuật: điện tử-máy tính, điện tử-viễn thông, IoT và trí tuệ nhân tạo, v.v. Có sự hiểu biết về lập trình sẽ là nền tảng vững chắc giúp chúng ta tiếp tục tìm hiểu các học phần chuyên ngành. Lập trình có thể tạo sản phẩm phần mềm phục vụ công việc, và còn rất nhiều lợi ích khác.

2. Các bước xây dựng chương trình:



Hình 1.1. Các bước xây dựng chương trình.

3. Cài đặt công cụ

Để có thể lập trình C, chúng ta cần cài bộ dịch gcc hoặc g++. Tuy nhiên để dễ dàng thì hiện tại có nhiều phần mềm tích hợp (hay gọi là môi trường phát triển – IDE) để chúng ta có thể dễ dàng học tập và sử dụng.

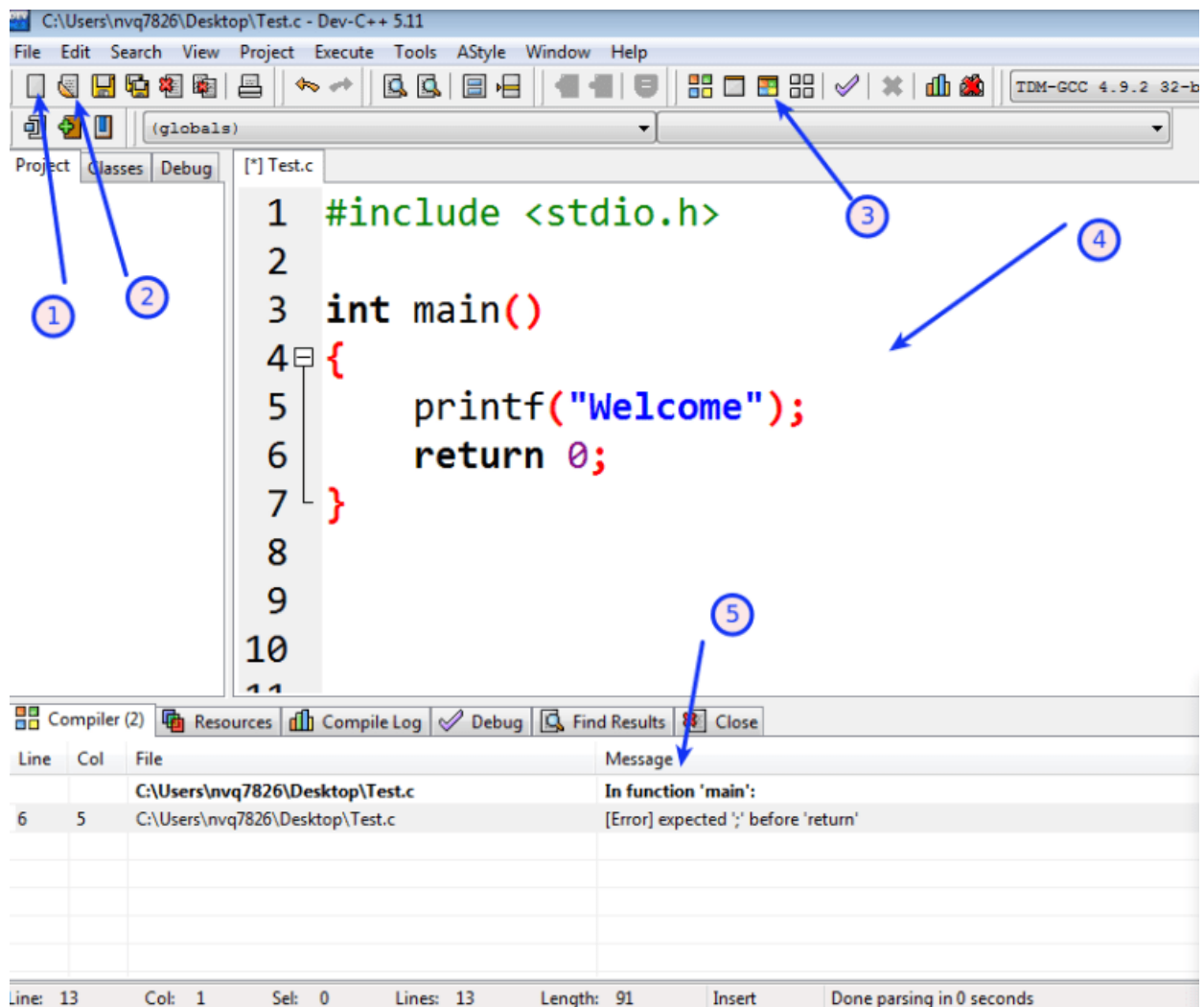
- Đối với Windows: có thể download Dev-C++ hoặc Code::Blocks về cài đặt và chúng ta có thể dùng ngay để viết các chương trình đơn giản.
- Đối với Linux (Ubuntu, ...): có thể cài bằng cách mở terminal lên và chạy lệnh cài đặt sau:
`sudo apt-get install build-essential`

Sau khi chạy lệnh trên, chúng ta có thể mở bất cứ trình soạn thảo nào để code (như gedit, vim, hay sublime-text,...) sau đó dịch, chạy chương trình bằng terminal. Chúng ta cũng có thể download và cài Code::Blocks trên Linux để dùng rất tiện, hoặc dùng lệnh cài: `sudo apt-get install codeblocks`

Để dễ và thống nhất trong quá trình học tập, C_Free, Dev-C++ được lựa chọn sử dụng.

3.1 Giới thiệu cơ bản về Dev-C++

a. Giao diện Dev-C++



Hình 1.2. Giao diện Dev-C++ ver. 5.11

Trong hình trên, có một số phần chính được đánh dấu bằng các số với ý nghĩa sau:

1. Nút tạo file mới
2. Nút mở một file đã có
3. Nút biên dịch và chạy chương trình
4. Vùng soạn thảo code (mã chương trình)
5. Vùng hiển thị lỗi nếu có.

b. Tạo chương trình đầu tiên

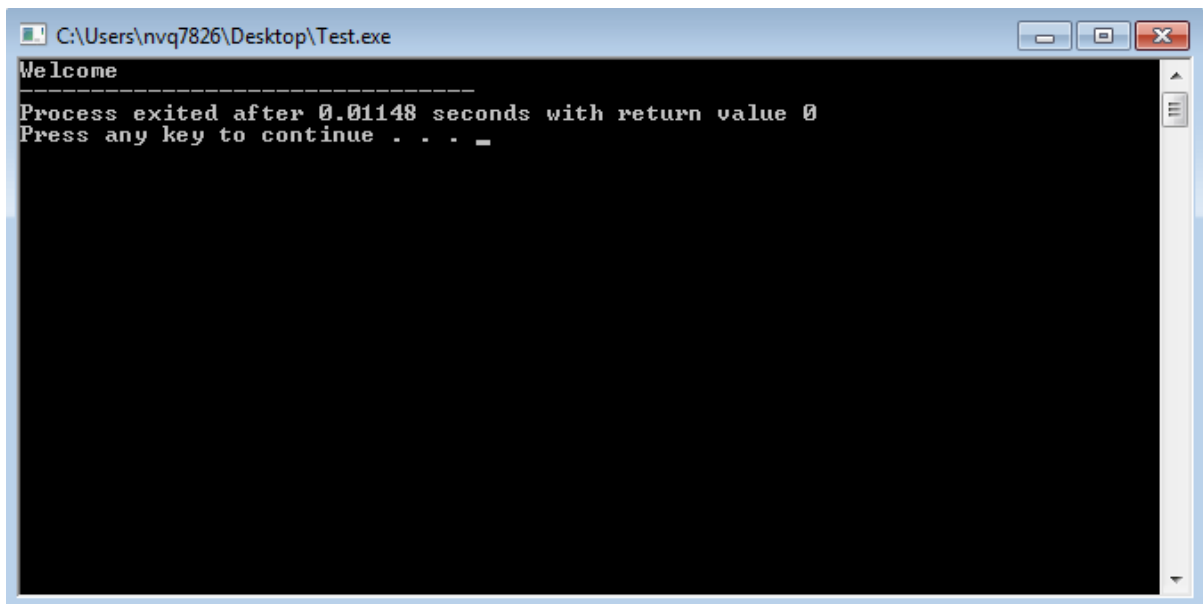
B1: Tạo 1 file mới.

B2: Gõ đoạn code sau vào vùng soạn thảo.

```
1    #include <stdio.h>
2
3    int main()
4    {
5        printf("Welcome");
6        return 0;
7    }
```

B3: Lưu lại với tên file là Test.c

B4: Click nút chạy chương trình và bạn sẽ thấy một màn hình màu đen hiện lên như sau:



Hình1.3. Kết quả chương trình sau khi thực thi.

Trong đó các bạn có thể thấy chữ Welcome hiện lên, đó chính là dòng chữ được in ra khi dùng lệnh printf. Phần dưới đường kẻ ngang là do Dev-C tự sinh ra.

Giải thích về code:

- Dòng 1: Chứa phát biểu tiền xử lý `#include <stdio.h>`. Vì trong chương trình này sử dụng các lệnh trong thư viện của C là `printf`, do đó chúng ta cần phải có khai báo của hàm thư viện này để báo cho trình biên dịch C biết. Nếu không khai báo chương trình sẽ báo lỗi. Thư viện `stdio.h` viết tắt của standard input output (std – i – o) là thư viện nhập xuất chuẩn).
- Dòng 3: `int main()` là thành phần chính của mọi chương trình C. Mọi chương trình C đều bắt đầu thi hành từ hàm `main`. Cặp dấu ngoặc `()` cho biết đây là khối hàm (function). Hàm `main()` có từ khóa `int` đầu tiên cho biết hàm này trả về giá trị kiểu nguyên (`int`).
- Dòng 4 và 7: cặp dấu ngoặc móc `{ }` giới hạn thân của hàm. Thân hàm bắt đầu bằng dấu `{` và kết thúc bằng dấu `}`.
- Dòng 5: `printf ("Welcome");`, chỉ thị cho máy in ra chuỗi ký tự nằm trong nháy kép `" "`. Hàng này được gọi là một câu lệnh, kết thúc một câu lệnh trong C phải là dấu chấm phẩy (`;`).
- Dòng 6: `return 0;` Trả về giá trị kiểu nguyên là 0 theo như đúng ban đầu là khai báo `int main()`.

Lưu ý:

1. Trong chương trình này chúng ta không dùng thư viện `conio.h` vì trong chuẩn C không có thư viện này, và từ đó cũng không dùng được `getch()` để dừng màn hình quan sát kết quả, chúng ta đã thay bằng lệnh `system("pause");` trong thư viện `stdlib.h`
2. Khi dùng `return` để trả về giá trị của hàm. Chúng ta có thể bỏ qua lệnh này chương trình vẫn chạy nhưng về chuẩn là sai, trả về 1 cũng sai, tóm lại là trả về 0. Nếu trả về một số khác không thì hệ thống máy tính sẽ hiểu là chương trình này của bạn sau khi chạy phát sinh ra cần đó lỗi.

Bây giờ, chúng ta mở folder chứa file Test.c vừa tạo ra, chúng ta thấy một file có tên Test.exe (gọi tắt là file exe), đây chính là file chạy chương trình. Có thể click chuột vào file

exe này là chạy được chương trình trên hoặc bạn có thể copy file exe này sang máy tính khác thì vẫn có thể chạy được, nó giống như chúng ta copy game từ máy này sang máy khác vậy.

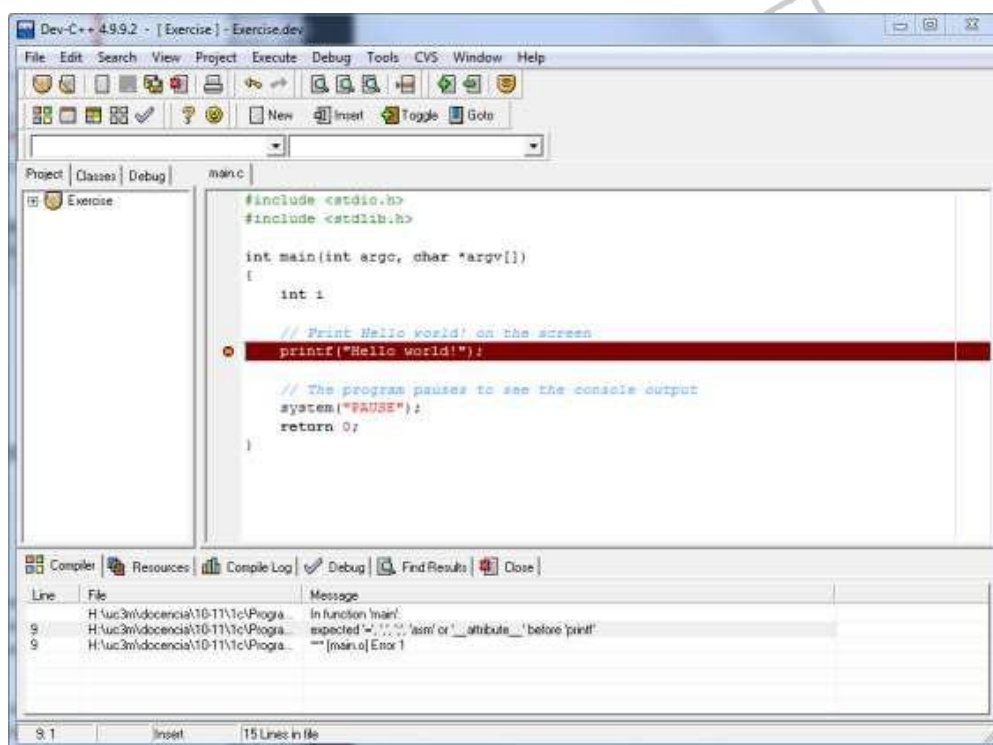
Từ đây ta có:

- File Test.c là file chúng ta tạo ra và viết các lệnh để máy tính hiểu, file này gọi là file mã nguồn.
- File Test.exe là file sinh ra khi chúng ta ấn nút Chạy chương trình, file này gọi là file thực thi.

Lưu ý: Các chương trình sau này chúng ta làm sẽ chủ yếu chạy trên màn hình đen như trên (gọi là màn hình console), tuy không có giao diện đẹp mắt nhưng đây là phần lõi của lập trình, học tốt phần này chúng ta mới có thể làm được các chương trình có giao diện đồ họa và các ngôn ngữ khác cũng hầu hết dựa vào phần lõi này

3.2. Debug chương trình

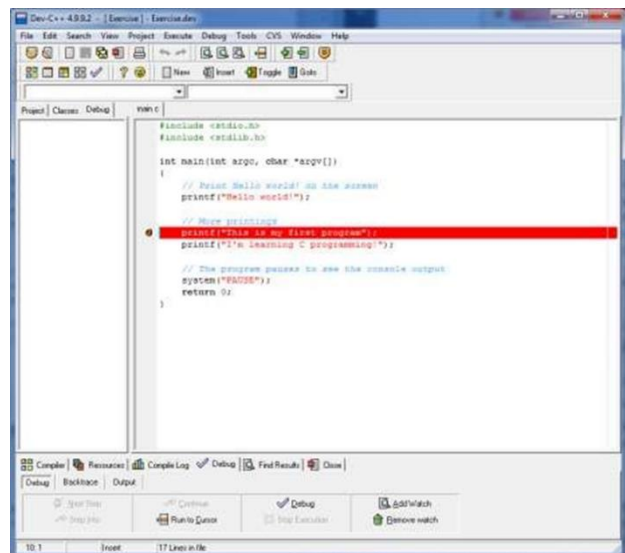
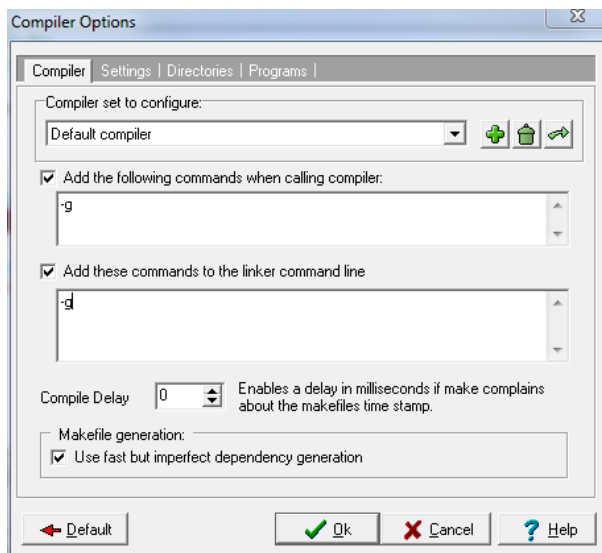
Lỗi biên dịch là lỗi được phát hiện bởi trình biên dịch. Để sửa lỗi biên dịch, chúng ta nên giải quyết lỗi đầu tiên và sau đó biên dịch lại chương trình một lần nữa, vì các lỗi tiếp theo thường là các phần sai của mã mà trình biên dịch không thể thực thi do lỗi đầu tiên.



Hình 1.4. Lỗi phát sinh khi biên dịch

Dev-C ++ gạch dưới màu đỏ dòng mã nơi lỗi biên dịch đã được phát hiện. Compile tab mô tả chi tiết các lỗi, Compile Log hiển thị thông báo lỗi do trình biên dịch chương trình. Việc phát triển một chương trình đúng về mặt cú pháp không có nghĩa là nó thực hiện đúng theo yêu cầu. Các chương trình phức tạp dễ xảy ra lỗi và thông thường lỗi logic được thực hiện trong mã nguồn không dẫn đến hành vi như mong đợi. Do đó, có thể cần phải thay đổi mã nguồn và lặp lại các bước biên dịch.

IDE bao gồm một trình gỡ lỗi, là một công cụ hỗ trợ để tìm lỗi. Để sử dụng trình gỡ lỗi, cần phải sửa đổi các tùy chọn trình biên dịch để bao gồm thông tin gỡ lỗi trong đối tượng và các tệp thực thi (Tools > Compiler Options > Compiler). Điều này được thực hiện bằng cách thêm tham số -g vào các lệnh gọi tới trình biên dịch và trình liên kết.



Hình 1.5. Chế độ debug chương trình.

Khi chương trình được biên dịch và liên kết với các tùy chọn này, chúng ta có thể chạy chương trình ở chế độ gỡ lỗi bằng cách nhấp vào nút Debug (F8). Chế độ này cho phép chạy chương trình từng bước (các hướng dẫn được thực hiện từng bước một), dừng việc thực thi tại các điểm ngắt (breakpoints) hoặc xem giá trị của một biến tại bất kỳ thời điểm thực thi nào.

Điểm ngắt (breakpoints)

Để tạo điểm ngắt, chúng ta có thể thực hiện nhiều cách:

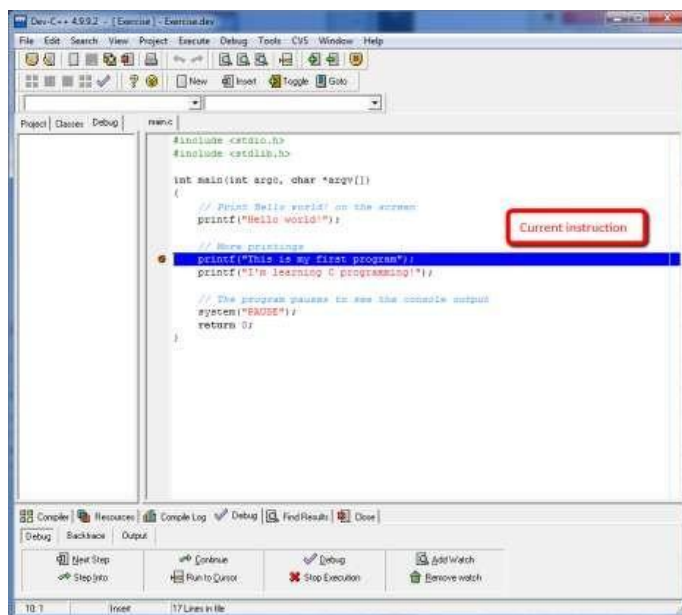
- chọn dòng mã và gõ Ctrl + F5;
- nhấp chuột phải vào dòng và chọn Toggle Breakpoint ;
- nhấp vào thanh dọc màu xám ở bên trái của mã. Hướng dẫn sẽ được đánh dấu màu đỏ.

Khi chương trình chạy ở chế độ Debug (F8), việc thực thi bị tạm dừng tại dòng này. Từ thời điểm này, việc thực hiện có thể tiếp tục bình thường hoặc từng bước. Hướng dẫn hiện tại được tô màu xanh lam.

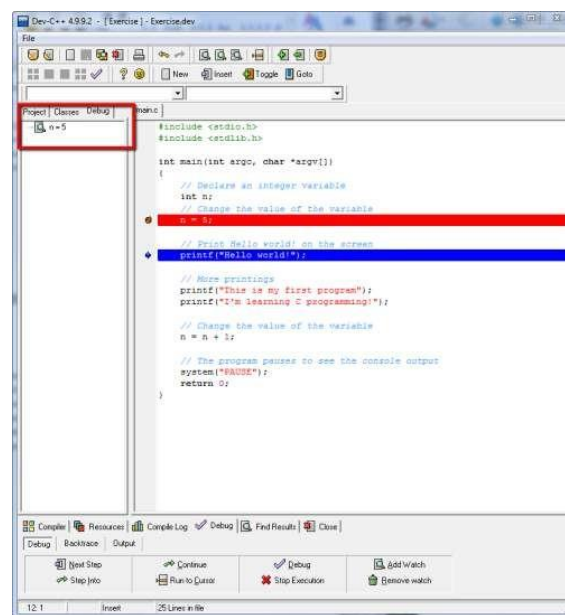
Các gỡ lỗi tab của kết quả chương trình cửa sổ lệnh gỡ lỗi của Dev-C ++. Như mô tả trong hình sau, có thể tiếp tục thực hiện, chạy chương trình từng bước, chạy đến con trỏ, v.v.

Lưu ý: Có thể bắt đầu thực thi một chương trình ở chế độ Gỡ lỗi mà không cần xác định bất kỳ điểm ngắt nào bằng cách sử dụng Run to Cursor.

Chế độ Gỡ lỗi cho phép tham khảo hoặc xem giá trị của một biến bất kỳ lúc nào trong quá trình thực thi chương trình. Để xem một biến, chương trình phải được dừng lại (với một điểm ngắt hoặc sau khi sử dụng Run to Cursor). Trong trường hợp này, chúng ta có thể chọn Add Watch trong Debug tab (cách khác, F4 or Debug > Add Watch) và nhập biến. Giá trị của các biến đã xem được liệt kê trong tab Gỡ lỗi của cửa sổ File Explorer ; nếu các giá trị này được thay đổi trong chương trình, thay đổi sẽ được hiển thị.



Hình 1.6a. Tạm dừng tại dòng lệnh trong debug.



Hình 1.6b Xem giá trị biến số trong kiểu debug.

BÀI TẬP

1. Lập trình là gì?
2. Ngôn ngữ lập trình là ngôn ngữ như thế nào?
3. Chương trình là gì?
4. Các bước xây dựng chương trình?
5. Thực hiện quá trình cài đặt phần mềm, viết chương trình “Hello” và thao tác debug chương trình.

BÀI 2 CƠ BẢN VỀ GIẢI THUẬT

Mục tiêu:

Kết thúc bài học này, sinh viên có thể:

- Hiểu rõ khái niệm giải thuật (algorithms)
- Biết sử dụng các ký hiệu dùng trong lưu đồ (flowchart)
- Vẽ lưu đồ (flowchart)
- Đọc hiểu được lưu đồ (flowchart)
- Sử dụng mã giả (pseudocode) để mô tả bài toán

1. Các bước giải quyết vấn đề

Chúng ta thường gặp phải những bài toán. Để giải quyết những bài toán đó, chúng ta cần hiểu chúng trước rồi sau đó mới hoạch định các bước cần làm. Giả sử chúng ta muốn đi từ phòng học đến quán ăn tự phục vụ ở tầng hầm. Để thực hiện việc này chúng ta cần **hiểu** nó rồi tìm ra **các bước giải quyết** trước khi thực thi các bước đó:

BUƯỚC 1 : Rời phòng

BUƯỚC 2 : Đến cầu thang

BUƯỚC 3 : Xuống tầng hầm

BUƯỚC 4 : Đi tiếp đến quán ăn tự phục vụ

Thủ tục trên liệt kê tập hợp các bước thực hiện được xác định rõ ràng cho việc giải quyết vấn đề. Một tập hợp các bước như vậy gọi là giải thuật (Algorithm hay gọi tắt là algo). Một giải thuật (còn gọi là thuật toán) có thể được định nghĩa như là một thủ tục, công thức hay cách giải quyết vấn đề. Nó gồm một tập hợp các bước giúp đạt được lời giải.

Qua phần trên, chúng ta thấy rõ ràng để giải quyết được một bài toán, trước tiên ta phải hiểu bài toán đó, kể đến chúng ta cần tập hợp tất cả những thông tin liên quan tới nó. Bước kế sẽ là xử lý những mâu thuẫn thông tin đó. Cuối cùng, chúng ta cho ra lời giải của bài toán đó.

Giải thuật chúng ta có là một tập hợp các bước được liệt kê dưới dạng ngôn ngữ đơn giản. Rất có thể rằng các bước trên do hai người khác nhau viết vẫn tương tự nhau nhưng ngôn ngữ dùng diễn tả các bước có thể khác nhau. Do đó, cần thiết có những phương pháp chuẩn mực cho việc viết giải thuật để mọi người dễ dàng hiểu nó. Chính vì vậy , giải thuật được viết bằng cách dùng hai phương pháp chuẩn là mã giả (pseudocode) và lưu đồ (flowchart).

Cả hai phương pháp này đều dùng để xác định một tập hợp các bước cần được thi hành để có được lời giải. Liên hệ tới vấn đề đi đến quán ăn tự phục vụ trên, chúng ta đã vạch ra một kế hoạch (thuật toán) để đến đích. Tuy nhiên, để đến nơi, chúng ta phải cần thi hành những bước này thật sự. Tương tự, mã giả và lưu đồ chỉ đưa ra những bước cần làm. Lập trình viên phải viết mã cho việc thực thi những bước này qua việc dùng một ngôn ngữ nào đó.

Chi tiết về mã giả và lưu đồ được trình bày dưới đây.

1.1 Mã giả (pseudocode)

Nhớ rằng mã giả không phải là mã thật. Mã giả sử dụng một tập hợp những từ tương tự như mã thật nhưng nó không thể được biên dịch và thực thi như mã thật. Chúng ta hãy xem xét mã giả qua ví dụ sau. Ví dụ này sẽ hiển thị câu 'Hello World!'.

Ví dụ 1:

```
BEGIN  
    DISPLAY 'Hello World!'  
END
```

Qua ví dụ trên, mỗi đoạn mã giả phải bắt đầu với từ BEGIN hoặc START, và kết thúc với từ END hay STOP. Để hiển thị giá trị nào đó, từ DISPLAY hoặc WRITE được dùng. Khi giá trị được hiển thị là một giá trị hằng (không đổi), trong trường hợp này là (Hello World), nó được đặt bên trong dấu nháy. Tương tự, để nhận một giá trị của người dùng, từ INPUT hay READ được dùng.

Để hiểu điều này rõ hơn, chúng ta xem xét ví dụ 2, ở ví dụ này ta sẽ nhập hai số và máy sẽ hiển thị tổng của hai số.

Ví dụ 2:

```
BEGIN  
    INPUT A, B  
    DISPLAY A + B  
END
```

Trong đoạn mã giả này, người dùng nhập vào hai giá trị, hai giá trị này được lưu trong bộ nhớ và có thể được truy xuất như là A và B theo thứ tự. Những vị trí được đặt tên như vậy trong bộ nhớ gọi là biến. Chi tiết về biến sẽ được giải thích trong phần sau của chương này. Bước kế tiếp trong đoạn mã giả sẽ hiển thị tổng của hai giá trị trong biến A và B.

Tuy nhiên, cũng đoạn mã trên, ta có thể bổ sung để lưu tổng của hai biến trong một biến thứ ba rồi hiển thị giá trị biến này như trong ví dụ 3 sau đây.

Ví dụ 3:

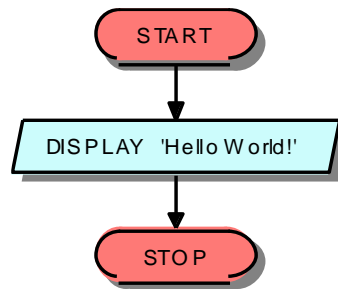
```
BEGIN  
    INPUT A, B  
    C = A + B  
    DISPLAY C  
END
```

Một tập hợp những chỉ thị hay các bước trong mã giả thì được gọi chung là một cấu trúc. Có ba loại cấu trúc : tuần tự, chọn lựa và lặp lại. Trong đoạn mã giả ta viết ở trên, chúng ta dùng cấu trúc tuần tự. Chúng được gọi như vậy vì những chỉ thị được thi hành tuần tự, cái này sau cái khác và bắt đầu từ điểm đầu tiên. Hai loại cấu trúc còn lại sẽ được đề cập trong những chương sau.

1.2 Lưu đồ (Flowcharts)

Một lưu đồ là một hình ảnh minh họa cho giải thuật. Nó vẽ ra biểu đồ của luồng chỉ thị hay những hoạt động trong một tiến trình. Mỗi hoạt động như vậy được biểu diễn qua những ký hiệu.

Để hiểu điều này rõ hơn, chúng ta xem lưu đồ trong hình 2.1 dùng hiển thị thông điệp truyền thống 'Hello World!'.



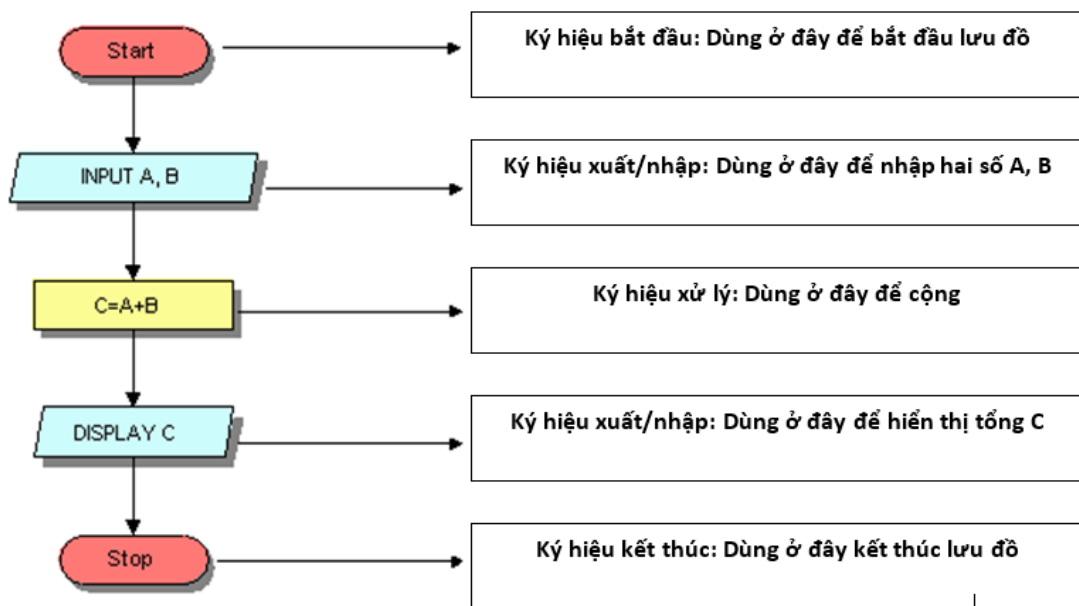
Hình 2.1: Lưu đồ

Lưu đồ giống với đoạn mã giả là cùng bắt đầu với từ BEGIN hoặc START, và kết thúc với từ END hay STOP. Tương tự, từ khóa DISPLAY được dùng để hiển thị giá trị nào đó đến người dùng. Tuy nhiên, ở đây, mọi từ khóa thì nằm trong những ký hiệu. Những ký hiệu khác nhau mang một ý nghĩa tương ứng được trình bày ở bảng trong Hình 2.2.

| Biểu Tượng | Mô Tả |
|------------|--|
| | Bắt đầu hay kết thúc chương trình |
| | Những bước tính toán |
| | Các lệnh xuất hay nhập |
| | Quyết định và rẽ nhánh |
| | Bộ nối hai phần trong chương trình (đầu nối) |
| | Dòng chảy |

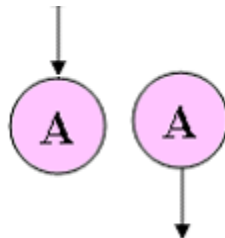
Hình 2.2: Ký hiệu trong lưu đồ

Ta hãy xét lưu đồ cho ví dụ 3 như ở Hình 2.3 dưới đây.



Hình 2.3: Lưu đồ cộng hai số

Tại bước mà giá trị của hai biến được cộng và gán cho biến thứ ba thì xem như là một xử lý và được trình bày bằng một hình chữ nhật. Lưu đồ mà chúng ta xét ở đây là đơn giản. Thông thường, lưu đồ trải rộng trên nhiều trang giấy. Trong trường hợp như thế, biểu tượng bộ nối được dùng để chỉ điểm nối của hai phần trong một chương trình nằm ở hai trang kế tiếp nhau. Vòng tròn chỉ sự nối kết và phải chứa ký tự hoặc số như ở hình 2.4. Như thế, chúng ta có thể tạo liên kết giữa hai lưu đồ chưa hoàn chỉnh.



Hình 2.4: Bộ nối

Bởi vì lưu đồ được sử dụng để viết chương trình, chúng cần được trình bày sao cho mọi lập trình viên hiểu chúng dễ dàng. Nếu có ba lập trình viên dùng ba ngôn ngữ lập trình khác nhau để viết mã, bài toán họ cần giải quyết phải như nhau. Trong trường hợp này, mã giả đưa cho lập trình viên có thể giống nhau mặc dù ngôn ngữ lập trình họ dùng và tất nhiên là cú pháp có thể khác nhau. Nhưng kết quả cuối cùng là một. Do đó, cần thiết phải hiểu rõ bài toán và mã giả phải được viết cẩn thận. Chúng ta cũng kết luận rằng mã giả độc lập với ngôn ngữ lập trình.

Vài điểm cần thiết khác phải chú ý khi vẽ một lưu đồ :

- Lúc đầu chỉ tập trung vào khía cạnh logic của bài toán và vẽ các luồng xử lý chính của lưu đồ
- Một lưu đồ phải có duy nhất một điểm bắt đầu (START) và một điểm kết thúc (STOP).
- Không cần thiết phải mô tả từng bước của chương trình trong lưu đồ mà chỉ cần các bước chính và có ý nghĩa cần thiết.

Chúng ta tuân theo những cấu trúc tuần tự, mà trong đó luồng thực thi chương trình đi qua tất cả các chỉ thị bắt đầu từ chỉ thị đầu tiên. Chúng ta có thể bắt gặp các điều kiện trong chương trình, dựa trên các điều kiện này hướng thực thi của chương trình có thể rẽ nhánh. Những cấu trúc cho việc rẽ nhánh như là cấu trúc chọn lựa, cấu trúc điều kiện hay rẽ nhánh. Những cấu trúc này được đề cập chi tiết sau đây:

➤ Cấu trúc IF (Nếu)

Cấu trúc chọn lựa cơ bản là cấu trúc 'IF'. Để hiểu cấu trúc này chúng ta hãy xem xét ví dụ trong đó khách hàng được giảm giá nếu mua trên 100 đồng. Mỗi lần khách hàng trả tiền, một đoạn mã chương trình sẽ kiểm tra xem lượng tiền trả có quá 100 đồng không?. Nếu đúng thế thì sẽ giảm giá 10% của tổng số tiền trả, ngược lại thì không giảm giá.

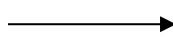
Điều này được minh họa sơ lược qua mã giả như sau:

IF khách hàng mua trên 100 thì giảm giá 10%

Cấu trúc dùng ở đây là câu lệnh IF. Hình thức chung cho câu lệnh IF (cấu trúc IF) như sau:

IF Điều kiện

Các câu lệnh



Phần thân của cấu trúc IF

END IF

Một cấu trúc 'IF' bắt đầu là IF theo sau là điều kiện. Nếu điều kiện là đúng (thỏa điều kiện) thì quyền điều khiển sẽ được chuyển đến các câu lệnh trong phần thân để thực thi. Nếu điều kiện sai (không thỏa điều kiện), những câu lệnh ở phần thân không được thực thi và chương trình nhảy đến câu lệnh sau END IF (chấm dứt cấu trúc IF). Cấu trúc IF phải được kết thúc bằng END IF.

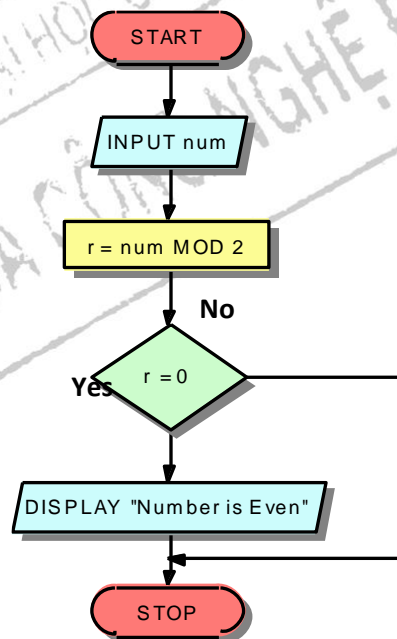
Chúng ta xem ví dụ 4 cho cấu trúc IF.

Ví dụ 4: Yêu cầu: Kiểm xem một số là chẵn hay không và hiển thị thông điệp báo nếu đúng là số chẵn, ta xử lý như sau :

```
BEGIN
    INPUT num
    r = num MOD 2
    IF r=0
        Display "Number is even"
    END IF
END
```

Đoạn mã trên nhập một số từ người dùng, thực hiện toán tử MOD (lấy phần dư) và kiểm tra xem phần dư có bằng 0 hay không. Nếu bằng 0 hiển thị thông điệp, ngược lại thoát ra.

Lưu đồ cho đoạn mã giả trên thể hiện qua hình 2.5.



Hình 2.5 : Kiểm tra số chẵn

Cú pháp của lệnh IF trong C như sau:

```
if (Điều kiện)
{
    Câu lệnh
}
```


➤ Cấu trúc IF...ELSE

Trong ví dụ 4, sẽ hay hơn nếu ta cho ra thông điệp báo rằng số đó không là số chẵn tức là số lẻ thay vì chỉ thoát ra. Để làm điều này ta có thể thêm câu lệnh IF khác để kiểm tra xem trường hợp số đó không chia hết cho 2. Ta xem ví dụ 5.

Example 5:

```
BEGIN
    INPUT num
    r = num MOD 2
    IF r=0
        DISPLAY "Even number"
    END IF

    IF r<>0
        DISPLAY "Odd number"
    END IF
END
```

Ngôn ngữ lập trình cung cấp cho chúng ta cấu trúc **IF...ELSE**. Dùng cấu trúc này sẽ hiệu quả và tốt hơn để giải quyết vấn đề. Cấu trúc **IF ...ELSE** giúp lập trình viên chỉ làm một phép so sánh và sau đó thực thi các bước tùy theo kết quả của phép so sánh là True (đúng) hay False (sai). Cấu trúc chung của câu lệnh IF...ELSE như sau:

IF Điều kiện

Câu lệnh 1

ELSE

Câu lệnh 2

END IF

Cú pháp của cấu trúc if...else trong C như sau:

if(Điều kiện)

{

Câu lệnh 1

}

else

{

Câu lệnh 2

}

Nếu điều kiện thỏa (True), câu lệnh 1 được thực thi. Ngược lại, câu lệnh 2 được thực thi. Không bao giờ cả hai được thực thi cùng lúc. Vì vậy, đoạn mã tối ưu hơn cho ví dụ tìm số chẵn được viết ra như ví dụ 6.

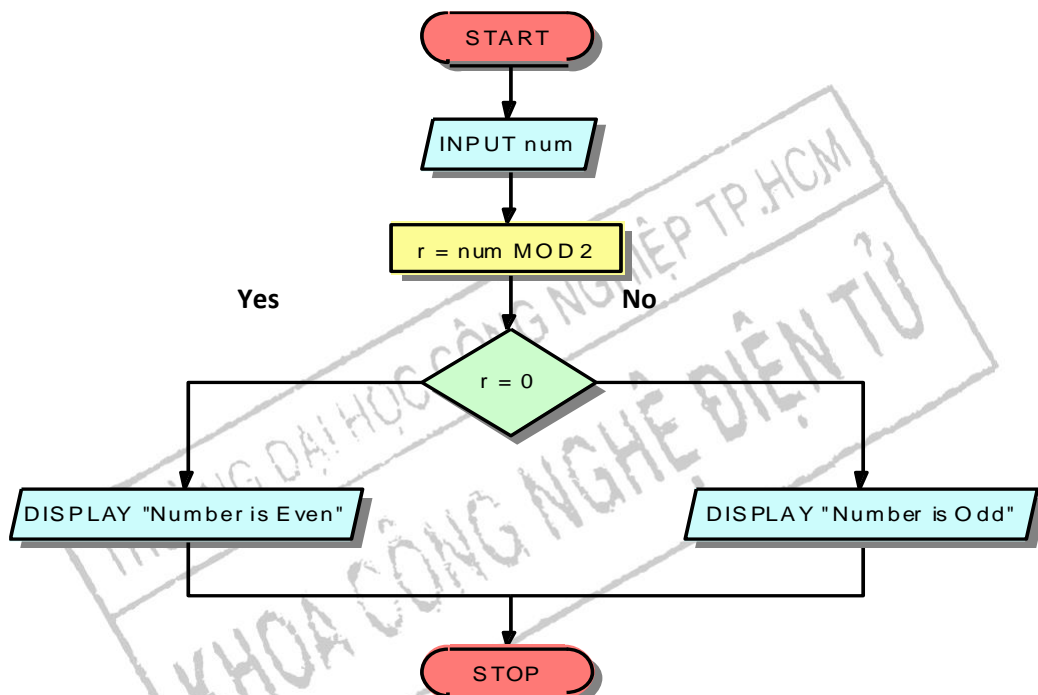
Ví dụ 6:

```

BEGIN
    INPUT num
    r = num MOD 2
    IF r = 0
        DISPLAY "Even Number"
    ELSE
        DISPLAY "Odd Number"
    END IF
END

```

Lưu đồ cho đoạn mã giả trên thể hiện qua Hình 2.6.



Hình 2.6: Số chẵn hay số lẻ

➤ Đa điều kiện sử dụng AND/OR

Cấu trúc IF...ELSE làm giảm độ phức tạp, gia tăng tính hữu hiệu. Ở một mức độ nào đó, nó cũng nâng cao tính dễ đọc của mã. Các thí dụ IF chúng ta đã đề cập đến thời điểm này thì khá đơn giản. Chúng chỉ có một điều kiện trong IF để đánh giá. Thỉnh thoảng chúng ta phải kiểm tra cho hơn một điều kiện, thí dụ: Để xem xét nhà cung cấp có đạt MVS (nhà cung cấp quan trọng nhất) không?, một công ty sẽ kiểm tra xem nhà cung cấp đó đã có làm việc với công ty ít nhất 10 năm không? và đã có tổng doanh thu ít nhất 5,000,000 không?. Hai điều kiện thỏa mãn thì nhà cung cấp được xem như là một MVS. Do đó toán tử AND có thể được dùng trong câu lệnh 'IF' như trong ví dụ 7 sau:

Ví dụ 7:

```

BEGIN
    INPUT yearsWithUs
    INPUT bizDone

```

```
IF yearsWithUs >= 10 AND bizDone >=5000000
```

```
    DISPLAY “Classified as an MVS”
```

```
ELSE
```

```
    DISPLAY “A little more effort required!”
```

```
END IF
```

```
END
```

Ví dụ 7 cũng khá đơn giản, vì nó chỉ có 2 điều kiện. Ở tình huống thực tế, chúng ta có thể có nhiều điều kiện cần được kiểm tra. Nhưng chúng ta có thể dễ dàng dùng toán tử AND để nối những điều kiện lại giống như ta đã làm ở trên.

Bây giờ, giả sử công ty trong ví dụ trên đổi quy định, họ quyết định đưa ra điều kiện dễ dàng hơn. Như là : Hoặc làm việc với công ty trên 10 năm hoặc có doanh số (giá trị thương mại, giao dịch) từ 5,000,000 trở lên. Vì vậy, ta thay thế toán tử AND bằng toán tử OR. Nhớ rằng toán tử OR cho ra giá trị True (đúng) nếu chỉ cần một điều kiện là True.

➤ Cấu trúc IF lồng nhau

Một cách khác để thực hiện ví dụ 7 là sử dụng cấu trúc IF lồng nhau. Cấu trúc IF lồng nhau là câu lệnh IF này nằm trong câu lệnh IF khác. Chúng ta viết lại ví dụ 7 sử dụng cấu trúc IF lồng nhau ở ví dụ 8 như sau:

Ví dụ 8:

```
BEGIN
```

```
    INPUT yearsWithUs
```

```
    INPUT bizDone
```

```
    IF yearsWithUs >= 10
```

```
        IF bizDone >=5000000
```

```
            DISPLAY “Classified as an MVS”
```

```
        ELSE
```

```
            DISPLAY “A little more effort required!”
```

```
        END IF
```

```
    ELSE
```

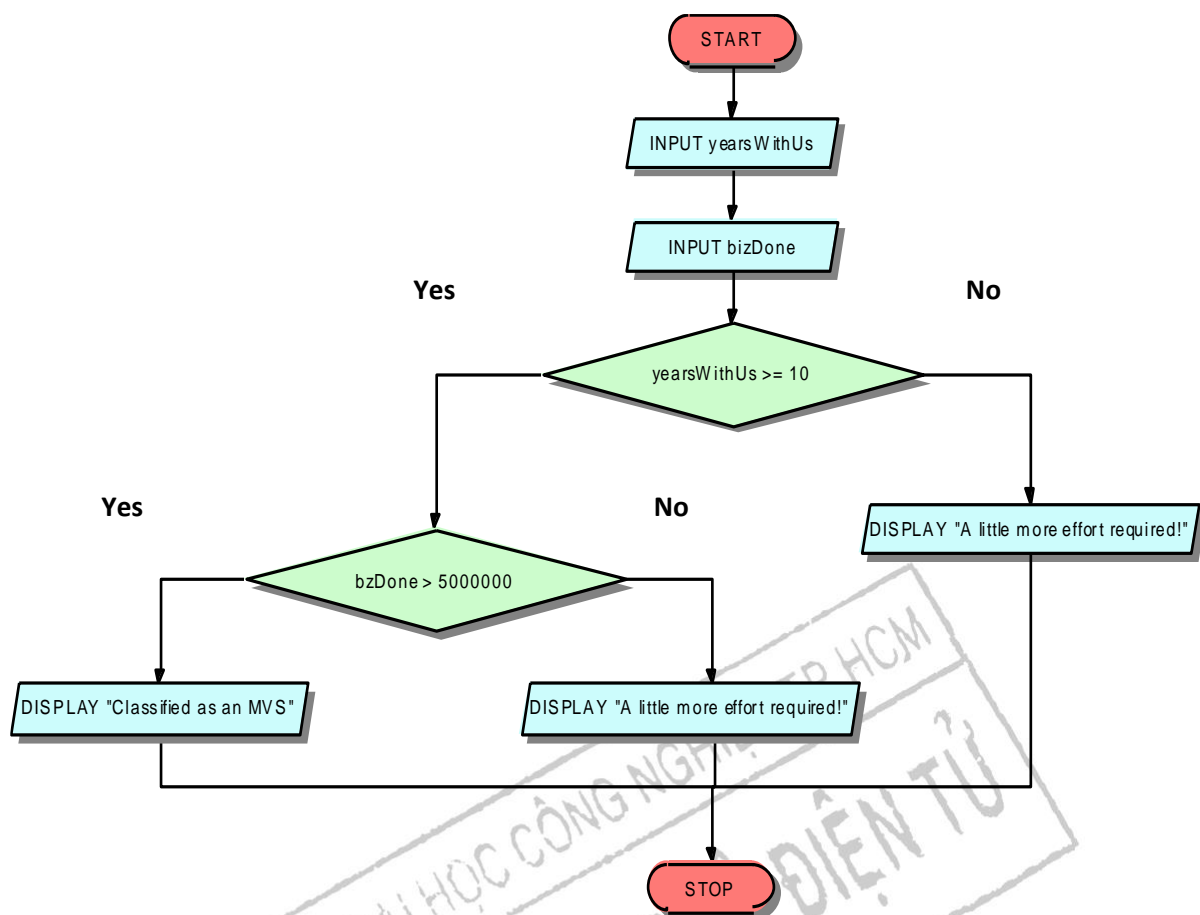
```
        DISPLAY “A little more effort required!”
```

```
    END IF
```

```
END
```

Đoạn mã trên thực hiện cùng nhiệm vụ nhưng không có ‘AND’. Tuy nhiên, chúng ta có một lệnh IF (kiểm tra xem bizDone lớn hơn hoặc bằng 5,000,000 hay không?) bên trong lệnh IF khác (kiểm tra xem yearsWithUs lớn hơn hoặc bằng 10 hay không?). Câu lệnh IF đầu tiên kiểm tra điều kiện thời gian nhà cung cấp làm việc với công ty có lớn hơn 10 năm hay không. Nếu dưới 10 năm (kết quả trả về là False), nó sẽ không công nhận nhà cung cấp là một MVS; Nếu thỏa điều kiện nó xét câu lệnh IF thứ hai, nó sẽ kiểm tra tới điều kiện bizDone lớn hơn hoặc bằng 5,000,000 hay không. Nếu thỏa điều kiện (kết quả trả về là True) lúc đó nhà cung cấp được xem là một MVS, nếu không thì một thông điệp báo rằng đó không là một MVS.

Lưu đồ cho mã giả của ví dụ 8 được trình bày qua hình 2.7.



Hình 2.7: Câu lệnh IF lồng nhau

Mã giả trong trường hợp này của cấu trúc IF lồng nhau tại ví dụ 8 chưa hiệu quả. Câu lệnh thông báo không thỏa điều kiện MVS phải viết hai lần. Hơn nữa lập trình viên phải viết thêm mã nên trình biên dịch phải xét hai điều kiện của lệnh IF, do đó lãng phí thời gian. Ngược lại, nếu dùng toán tử AND chỉ xét tới điều kiện của câu lệnh IF một lần. Điều này không có nghĩa là cấu trúc IF lồng nhau nói chung là không hiệu quả. Nó tùy theo tình huống cụ thể mà ta dùng nó. Có khi dùng toán tử AND hiệu quả hơn, có khi dùng cấu trúc IF lồng nhau hiệu quả hơn. Chúng ta sẽ xét một ví dụ mà dùng cấu trúc IF lồng nhau hiệu quả hơn dùng toán tử AND.

Một công ty định phân lương cơ bản cho công nhân dựa trên tiêu chuẩn như trong bảng 2.1.

Bảng 2.1: Lương cơ bản

| Grade | Experience | Salary |
|-------|------------|--------|
| E | 2 | 2000 |
| E | 3 | 3000 |
| M | 2 | 3000 |
| M | 3 | 4000 |

Vì vậy, nếu một công nhân được xếp loại là E và có hai năm kinh nghiệm thì lương là 2000, nếu ba năm kinh nghiệm thì lương là 3000.

Mã giả dùng toán tử AND cho vấn đề trên như ví dụ 9:

Ví dụ 9:

BEGIN

```

INPUT grade
INPUT exp
IF grade ="E" AND exp =2
    salary=2000
ELSE
    IF grade = "E" AND exp=3
        salary=3000
    END IF
END IF

```

```

IF grade ="M" AND exp =2
    salary=3000
ELSE
    IF grade = "M" AND exp=3
        salary=4000
    END IF
END IF
END

```

Câu lệnh IF đầu tiên kiểm tra xếp loại và kinh nghiệm của công nhân. Nếu xếp loại là E và kinh nghiệm là 2 năm thì lương là 2000, ngoài ra nếu xếp loại E, nhưng có 3 năm kinh nghiệm thì lương là 3000. Nếu cả 2 điều kiện không thỏa thì câu lệnh IF thứ hai cũng tương tự sẽ kiểm tra điều kiện xếp loại và kinh nghiệm cho công nhân để phân định lương.

Giả sử xếp loại của một công nhân là E và có hai năm kinh nghiệm. Lương người đó sẽ được tính theo mệnh đề IF đầu tiên. Phần còn lại của câu lệnh IF thứ nhất được bỏ qua. Tuy nhiên, điều kiện tại mệnh đề IF thứ hai sẽ được xét và tất nhiên là không thỏa, do đó nó kiểm tra mệnh đề ELSE của câu lệnh IF thứ 2 và kết quả cũng là False. Đây quả là những bước thừa mà chương trình đã xét qua. Trong ví dụ, ta chỉ có hai câu lệnh IF bởi vì ta chỉ xét có hai loại là E và M. Nếu có khoảng 15 loại thì sẽ tốn thời gian và tài nguyên máy tính cho việc tính toán thừa mặc dù lương đã xác định tại câu lệnh IF đầu tiên. Đây dứt khoát không phải là mã nguồn hiệu quả.

Bây giờ chúng ta xét mã giả dùng cấu trúc IF lồng nhau đã được sửa đổi trong ví dụ 10.

Ví dụ 10:

```

BEGIN
    INPUT grade
    INPUT exp
    IF grade="E"
        IF exp=2
            salary = 2000
        ELSE
            IF exp=3

```

```

        salary=3000
    END IF
END IF
ELSE
    IF grade="M"
        IF exp=2
            Salary=3000
        ELSE
            IF exp=3
                Salary=4000
            END IF
        END IF
    END IF
END IF
END IF
END

```

Đoạn mã trên nhìn khó đọc. Tuy nhiên, nó đem lại hiệu suất cao hơn. Chúng ta xét cùng ví dụ như trên. Nếu công nhân được xếp loại là E và kinh nghiệm là 2 năm thì lương được tính là 2000 ngay trong bước đầu của câu lệnh IF. Sau đó, chương trình sẽ thoát ra vì không cần thực thi thêm bất cứ lệnh ELSE nào. Do đó, không có sự lãng phí và đoạn mã này mang lại hiệu suất cho chương trình và chương trình chạy nhanh hơn.

➤ Vòng lặp

Một chương trình máy tính là một tập các câu lệnh sẽ được thực hiện tuần tự. Nó có thể lặp lại một số bước với số lần lặp xác định theo yêu cầu của bài toán hoặc đến khi một số điều kiện nhất định được thỏa. Chẳng hạn, ta muốn viết chương trình hiển thị tên của ta 5 lần. Ta xét mã giả dưới đây.

Ví dụ 11:

```

BEGIN
    DISPLAY "Scooby"
    DISPLAY "Scooby"
    DISPLAY "Scooby"
    DISPLAY "Scooby"
    DISPLAY "Scooby"
END

```

Nếu để hiển thị tên ta 1000 lần, nếu ta viết DISPLAY "Scooby" 1000 lần thì rất tốn công sức. Ta có thể tinh giản vấn đề bằng cách viết câu lệnh DISPLAY chỉ một lần, sau đó đặt nó trong cấu trúc vòng lặp, và chỉ thị máy tính thực hiện lặp 1000 lần cho câu lệnh trên. Ta xem mã giả của cấu trúc vòng lặp trong ví dụ 12 như sau:

Ví dụ 12:

Do loop 1000 times

DISPLAY “Scooby”

End loop

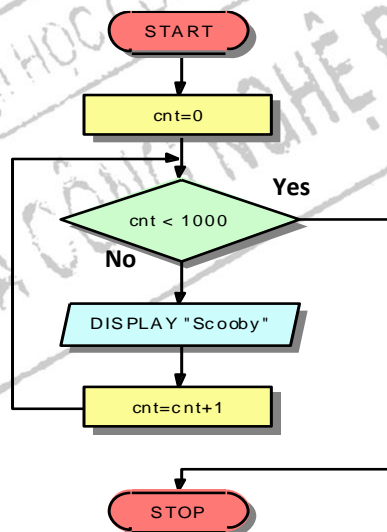
Những câu lệnh nằm giữa Do loop và End loop (trong ví dụ trên là lệnh DISPLAY) được thực thi 1000 lần. Những câu lệnh này cùng với các lệnh **do loop** và **end loop** được gọi là cấu trúc vòng lặp. Cấu trúc vòng lặp giúp lập trình viên phát triển thành những chương trình lớn trong đó có thể yêu cầu thực thi hàng ngàn câu lệnh. **Do loop...end loop** là một dạng thức tổng quát của vòng lặp.

Ví dụ sau là cách viết khác nhưng cũng dùng cấu trúc vòng lặp.

Ví dụ 13:

```
BEGIN
  cnt=0
  WHILE (cnt < 1000)
  DO
    DISPLAY “Scooby”
    cnt=cnt+1
  END DO
END
```

Lưu đồ cho mã giả trong ví dụ 13 được vẽ trong Hình 2.8.



Hình 2.8: Cấu trúc vòng lặp

Chú ý rằng Hình 2.8 không có ký hiệu đặc biệt nào để biểu diễn cho vòng lặp. Chúng ta dùng ký hiệu phân nhánh để kiểm tra điều kiện và quản lý hướng đi của của chương trình bằng các dòng chảy (flow_lines).

BÀI TẬP

Thực hiện vẽ lưu đồ (flowchart) và viết mã giả cho các bài toán sau.

1. Giải phương trình bậc 2: $ax^2 + bx + c = 0$
2. Hiển thị số chẵn từ 0 tới n với n nhập vào (có kiểm tra điều kiện $n > 0$)
3. Hiển thị các số nguyên tố từ 0 tới n (có kiểm tra điều kiện $n > 0$)
4. Tính $n!!$
 $= 1 * 3 * 5 * \dots * n$ (nếu n lẻ)
 $= 2 * 4 * 6 * \dots * n$ (nếu n chẵn)
5. Nhập 1 số nguyên n tìm ra số m sao cho: $1 + 2 + 3 + \dots + m \leq n$



BÀI 3: CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ C

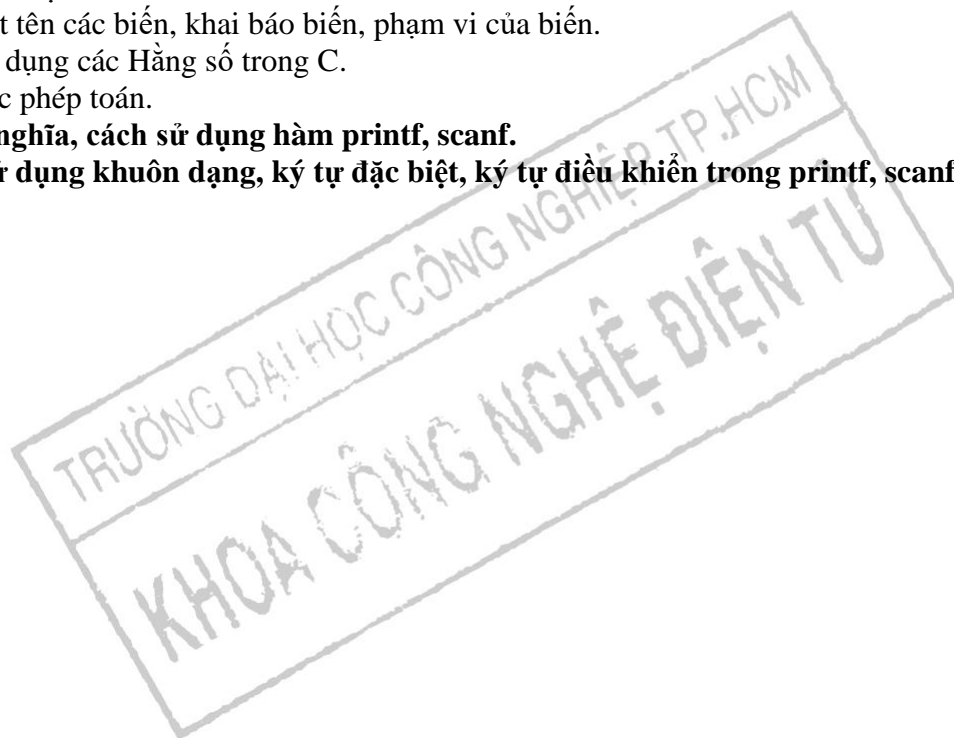
Mục tiêu:

Sau khi hoàn tất bài này sinh viên sẽ hiểu và vận dụng các kiến thức kỹ năng cơ bản sau:

- Khái niệm từ khóa trong ngôn ngữ C.
- Các kiểu dữ liệu.
- Cách tạo chú thích.
- Đặt tên các biến, khai báo biến, phạm vi của biến.
- Sử dụng các hằng số trong C.
- Biểu thức và phép toán.
- Vào ra dữ liệu cơ bản.

Nội dung:

- Các từ khóa trong ngôn ngữ C.
- Các kiểu dữ liệu.
- Cách tạo chú thích.
- Đặt tên các biến, khai báo biến, phạm vi của biến.
- Sử dụng các Hằng số trong C.
- Các phép toán.
- **Ý nghĩa, cách sử dụng hàm printf, scanf.**
- **Sử dụng khuôn dạng, ký tự đặc biệt, ký tự điều khiển trong printf, scanf.**



1. Các từ khoá trong C.

1.1. Lý thuyết.

Từ khóa là các từ dành riêng (reserved words) của C mà người lập trình có thể sử dụng nó trong chương trình tùy theo ý nghĩa của từng từ. Ta không được dùng từ khóa để đặt cho các tên của riêng mình. Các từ khóa của C:

asm • auto • break • case • cdecl • char • class • const • continue • _cs • default • delete • do double • _ds • else • enum • _es • extern • _export • far • _fastcall • float • for • friend • goto • huge • if • inline • int • interrupt • _loadds • long • near • new • operator • pascal • private • protected • public • register • return • _saveregs • _seg • short • signed • sizeof • _ss • static • struct • switch • template • this • typedef • union • unsigned • virtual • void • volatile • while

Các từ khóa phải viết bằng chữ thường

1.2. Tên.

Khái niệm tên rất quan trọng trong quá trình lập trình, nó không những thể hiện rõ ý nghĩa trong chương trình mà còn dùng để xác định các đại lượng khác nhau khi thực hiện chương trình. Tên thường được đặt cho hằng, biến, mảng, con trỏ, nhãn... Chiều dài tối đa của tên là 32 ký tự. Tên biến hợp lệ là một chuỗi ký tự liên tục gồm: **Ký tự chữ, số và dấu gạch dưới**. Ký tự đầu của tên phải là chữ hoặc dấu gạch dưới. Khi đặt tên không được đặt trùng với các từ khóa.

Ví dụ 1 :

Các tên đúng: delta, a_1, Num_ODD, Case

Các tên sai:

3a_1 (ký tự đầu là số)

num-odd (sử dụng dấu gạch ngang)

int (đặt tên trùng với từ khóa)

del ta (có khoảng trắng)

f(x) (có dấu ngoặc tròn)

Lưu ý: Trong C, tên phân biệt chữ hoa, chữ thường

Ví dụ 2 : number khác Number

case khác Case

(case là từ khóa, do đó ta đặt tên là Case vẫn đúng)

2. Kiểu dữ liệu.

2.1. Kiểu dữ liệu cơ bản trong C là: **char, int, float, double**.

| TT | Kiểu dữ liệu (Type) | Kích thước (Length) | Miền giá trị (Range) |
|----|---------------------|---------------------|-----------------------------------|
| 1 | unsigned char | 1 byte | 0 đến 255 |
| 2 | char | 1 byte | – 128 đến 127 |
| 3 | enum | 2 bytes | – 32,768 đến 32,767 |
| 4 | unsigned int | 2 bytes | 0 đến 65,535 |
| 5 | short int | 2 bytes | – 32,768 đến 32,767 |
| 6 | int | 4 bytes | – 32,768 đến 32,767 |
| 7 | unsigned long | 4 bytes | 0 đến 4,294,967,295 |
| 8 | long | 4 bytes | – 2,147,483,648 đến 2,147,483,647 |

| | | | |
|----|-------------|----------|--|
| 10 | float | 8 bytes | $3.4 * 10^{-38}$ đến $3.4 * 10^{38}$ |
| 11 | double | 10 bytes | $1.7 * 10^{-308}$ đến $1.7 * 10^{308}$ |
| | long double | | $3.4 * 10^{-4932}$ đến $1.1 * 10^{4932}$ |

2.2. Ghi chú.

Trong khi lập trình cần phải ghi chú để giải thích các biến, hằng, thao tác xử lý giúp cho chương trình rõ ràng dễ hiểu, dễ nhớ, dễ sửa chữa và để người khác đọc vào dễ hiểu. Trong C có các ghi chú sau: // hoặc /* nội dung ghi chú */

```
void main()
{
    int a, b; //khai bao bien t kieu int
    a = 1; //gan 1 cho a
    b = 3; //gan 3 cho b
    /* thuat toan tim so lon nhat la
    neu a lon hon b, thi a lon nhat
    nguoc lai b lon nhat */
    if (a > b) printf("max: %d", a);
    else printf("max: %d", b);
}
```

Khi biên dịch chương trình, C gộp cặp dấu ghi chú sẽ không dịch ra ngôn ngữ máy. Tóm lại, đối với ghi chú dạng // dùng để ghi chú một hàng và dạng /* */ có thể ghi chú một hàng hoặc nhiều hàng.

3. Biến.

Cú pháp

Kiểu dữ liệu *Danh sách tên biến;*

- Kiểu dữ liệu: Theo bảng trên

Danh sách tên biến: Gồm các tên biến có cùng kiểu dữ liệu, mỗi tên biến cách nhau dấu phẩy (,), cuối cùng là dấu chấm phẩy (;).

-Khi khai báo biến nên đặt tên biến theo quy tắc Hungarian Notation

Ví dụ 4 :

```
int ituai;           //khai báo biến ituai có kiểu int
float fTrongluong;    //khai báo biến fTrongluong có kiểu long
char ckitu1, ckitu2; //khai báo biến ckitu1, ckitu2 có kiểu char
```

Các biến khai báo trên theo quy tắc Hungarian Notation. Nghĩa là biến **ituai** là kiểu **int**, ta thêm chữ i (kí tự đầu của kiểu) vào đầu tên biến tuoi để trong quá trình lập trình hoặc sau này xem lại, sửa chữa... ta dễ dàng nhận ra biến ituai có kiểu **int** mà không cần phải di chuyển đến phần khai báo mới biết kiểu của biến này. Tương tự cho biến **fTrongluong**, ta nhìn vào là biết ngay biến này có kiểu **float**.

Vừa khai báo vừa khởi gán

Có thể kết hợp việc khai báo với toán tử gán để biến nhận ngay giá trị cùng lúc với khai báo.

Ví dụ 5 :

+ **Khai báo trước, gán giá trị sau:**

```
void main()
{
    int a, b, c;
    a = 1;
    b = 2;
    c = 5;
    ...
}
```

+ **Vừa khai báo vừa gán giá trị:**

```
void main()
{
    int a = 1, b = 2, c = 5;
    ...
}
```

Phạm vi của biến.

Khai báo biến ngoài (biến toàn cục): Vị trí biến đặt bên ngoài tất cả các hàm, cấu trúc... Các biến này có ảnh hưởng đến toàn bộ chương trình. Chu trình sống của nó là bắt đầu chạy chương trình đến lúc kết thúc chương trình. Khai báo biến trong (biến cục bộ): Vị trí biến đặt bên trong hàm, cấu trúc.... Chỉ ảnh hưởng nội bộ bên trong hàm, cấu trúc đó... Chu trình sống của nó bắt đầu từ lúc hàm, cấu trúc được gọi thực hiện đến lúc thực hiện xong.

4. Hằng (Constant).

Là đại lượng không đổi trong suốt quá trình thực thi của chương trình. Hằng có thể là một chuỗi ký tự, một ký tự, một con số xác định. Chúng có thể được biểu diễn hay định dạng (Format) với nhiều dạng thức khác nhau.

4.1. Hằng số thực.

Số thực bao gồm các giá trị kiểu *float*, *double*, *long double* được thể hiện theo 2 cách sau:

- Cách 1: Sử dụng cách viết thông thường mà chúng ta đã sử dụng trong các môn Toán, Lý, ... Điều cần lưu ý là sử dụng dấu thập phân là dấu chấm (.);

Ví dụ: 123.34-223.3333.00-56.0

- Cách 2: Sử dụng cách viết theo số mũ hay số khoa học. Một số thực được tách làm 2 phần, cách nhau bằng ký tự e hay E

Phần giá trị: là một số nguyên hay số thực được viết theo cách 1.

Phần mũ: là một số nguyên

Giá trị của số thực là: Phần giá trị nhân với 10 mũ phần mũ.

Ví dụ: $1234.56e-3 = 1.23456$ (là số $1234.56 * 10^{-3}$)

$-123.45E4 = -1234500$ (là $-123.45 * 10^4$)

4.2. Hằng số nguyên.

Số nguyên gồm các kiểu *int* (2 bytes), *long* (4 bytes) được thể hiện theo những cách sau.

- Hằng số nguyên 2 bytes (*int*) hệ thập phân: Là kiểu số mà chúng ta sử dụng thông thường, hệ thập phân sử dụng các ký số từ 0 đến 9 để biểu diễn một giá trị nguyên.

Ví dụ: 123 (một trăm hai mươi ba), -242 (trừ hai trăm bốn mươi hai).

- Hằng số nguyên 2 byte (int) hệ bát phân: Là kiểu số nguyên sử dụng 8 ký số từ 0 đến 7 để biểu diễn một số nguyên.

Cách biểu diễn: 0<các ký số từ 0 đến 7>

Ví dụ : 0345 (số 345 trong hệ bát phân)

-020 (số -20 trong hệ bát phân)

4.3. Hằng ký tự.

Hằng ký tự là một ký tự riêng biệt được viết trong cặp dấu nháy đơn (''). Mỗi một ký tự tương ứng với một giá trị trong bảng mã ASCII. Hằng ký tự cũng được xem như trị số nguyên.

Ví dụ: 'a', 'A', '0', '9'

Chúng ta có thể thực hiện các phép toán số học trên 2 ký tự (thực chất là thực hiện phép toán trên giá trị ASCII của chúng)

4.4. Hằng chuỗi ký tự.

Hằng chuỗi ký tự là một chuỗi hay một xâu ký tự được đặt trong cặp dấu nháy kép (").

Ví dụ: "Nhap mon lap trinh C", "Khoa CNTT", "BMDTMT"

Chú ý:

1. Một chuỗi không có nội dung "" được gọi là chuỗi rỗng.
2. Khi lưu trữ trong bộ nhớ, một chuỗi được kết thúc bằng ký tự NULL ('\0': mã Ascii là 0).
3. Để biểu diễn ký tự đặc biệt bên trong chuỗi ta phải thêm dấu \ phía trước

Ví dụ: "I'm a student" phải viết "I\\m a student"

"Day la ky tu \"dac biet\"" phải viết "Day la ky tu \\\"dac biet\\\""

5. Biểu thức và phép toán.

5.1. Biểu thức (Expressions)

Một biểu thức là tổ hợp các toán tử và toán hạng. Toán tử thực hiện các thao tác như cộng, trừ, so sánh v.v... Toán hạng là những biến hay những giá trị mà các phép toán được thực hiện trên nó. Trong ví dụ $a + b$, "a" và "b" là toán hạng và "+" là toán tử. Tất cả kết hợp lại là một biểu thức.

Trong quá trình thực thi chương trình, giá trị thực sự của biến (nếu có) sẽ được sử dụng cùng với các hằng có mặt trong biểu thức. Việc đánh giá biểu thức được thực hiện nhờ các toán tử. Vì vậy, mọi biểu thức trong C đều có một giá trị.

Ví dụ 1.

2
x
3 + 7
 $2 \times y + 5$
 $2 + 6 \times (4 - 2)$
 $z + 3 \times (8 - z)$

Ví dụ 2:

a + b
 $b = 1 + 5 * 2/i$

$a = 6 \% (7 + 1)$
 $x++ * 2/4 + 5 - \text{power}(i, 2)$

Toán hạng sử dụng trong biểu thức có thể là hằng số, biến, hàm.

5.2. Phép toán.

Trong có 4 nhóm toán tử chính sau đây:

a. Phép toán số học

| | |
|---|--|
| $+$: cộng $-$: trừ $*$: nhân $/$: chia | áp dụng trên tất cả các toán hạng có kiểu dữ liệu char, int, float, double (kể cả long, short, unsigned) |
| $\%$: lấy phần dư | áp dụng trên các toán hạng có kiểu dữ liệu char, int, long |

* Thứ tự ưu tiên:

| | | | |
|----------------|-------|-------------|--------|
| Đảo dấu $+, -$ | $()$ | $*, / , \%$ | $+, -$ |
|----------------|-------|-------------|--------|

Ví dụ 2:

$10\%4 = 2$ (10 chia 4 dư 2); $9\%3 = 0$ (9 chia 3 dư 0)

$3 * 5 + 4 = 19$

$6 + 2 / 2 - 3 = 4$

$-7 + 2 * ((4 + 3) * 4 + 8) = 65$

- Chỉ sử dụng cặp ngoặc $()$ trong biểu thức, cặp ngoặc đơn được thực hiện theo thứ tự ưu tiên từ trong ra ngoài.

b. Phép quan hệ

$>$: lớn hơn

$>=$: lớn hơn hoặc bằng

$<$: nhỏ hơn

$<=$: nhỏ hơn hoặc bằng

$==$: bằng

$!=$: khác

* Thứ tự ưu tiên: $>, >=, <, <=, ==, !=$

- Kết quả của phép toán quan hệ là số nguyên kiểu int, bằng 1 nếu đúng, bằng 0 nếu sai.

Phép toán quan hệ ngoài toán hạng được sử dụng là số còn được sử dụng với kiểu dữ liệu char.

* Thứ tự ưu tiên giữa toán tử số học và toán tử quan hệ

| | |
|----------------|-----------------|
| Toán tử số học | Toán tử quan hệ |
|----------------|-----------------|

Ví dụ 3:

$4 > 10 \rightarrow$ có giá trị 0 (sai)

$4 >= 4 \rightarrow$ có giá trị 1 (đúng)

$3 == 5 \rightarrow$ có giá trị 0 (sai)

$2 <= 1 \rightarrow$ có giá trị 0 (sai)

$6 - 3 < 4 \rightarrow$ có giá trị 1 (đúng), tương đương $(6 - 3) < 4$

$-2 * -4 < 3 + 2 \rightarrow$ có giá trị 0 (sai), tương đương $(-2 * -4) < (3 + 2)$

c. Phép toán luận lý

$!$: NOT (phép phủ định)

$\&\&$: AND (phép và)

$\|$: OR (phép hoặc)

| Toán hạng a | Toán hạng b | !a | a && b | a b |
|-------------|-------------|----------|----------|----------|
| Khác 0 | Khác 0 | 0 (sai) | 1 (đúng) | 1 (đúng) |
| Khác 0 | Bằng 0 | 0 (sai) | 0 (sai) | 1 (đúng) |
| Bằng 0 | Khác 0 | 1 (đúng) | 0 (sai) | 1 (đúng) |
| Bằng 0 | Bằng 0 | 1 (đúng) | 0 (sai) | 0 (sai) |

* Thứ tự ưu tiên: ! && ||

Ví dụ 4:

| | |
|-----------|-----------------------|
| !(2 <= 1) | → có giá trị 1 (đúng) |
| 5 && 10 | → có giá trị 1 (đúng) |
| !6 | → có giá trị 0 (sai) |
| 1 && 0 | → có giá trị 0 (sai) |
| 1 0 | → có giá trị 1 (đúng) |

* Thứ tự ưu tiên giữa các toán tử:

| | | | | |
|---|----------------|-----------------|----|--|
| ! | Toán tử số học | Toán tử quan hệ | && | |
|---|----------------|-----------------|----|--|

d. Phép toán trên bit (bitwise)

& : và (AND)

| : hoặc (OR)

^ : hoặc loại trừ (XOR)

>> : dịch phải

<< : dịch trái

~ : đảo

| Bit a | Bit b | ~a | a & b | a b | a ^ b |
|-------|-------|----|-------|-------|-------|
| 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 |

Ví dụ 5:

a = 13 → đổi ra hệ nhị phân → 1101

b = 10 → đổi ra hệ nhị phân → 1010

1101 1101 1101

& 1010 | 1010 ^ 1010

= 1000 = 1111 = 0111

= 8 = 15 = 7 (dạng thập phân)

a = 1235 → đổi ra hệ nhị phân → 0100 1101 0011

b = 465 → đổi ra hệ nhị phân → 0001 1101 0001

0100 1101 0011 0100 1101 0011 0100 1101 0011

& 0001 1101 0001 | 0001 1101 0001 ^ 0001 1101 0001

= 0000 1101 0001 = 0101 1101 0011 = 0101 0000 0010

= 209 = 1491 = 1282 (dạng thập phân)

e. Các phép toán khác

+ Phép toán gán

Phép gán là thay thế giá trị hiện tại của biến bằng một giá trị mới.

Các phép gán: =, +=, -=, *=, /=, %=, <<=, >>=, &=, |=, ^=.

Ví dụ 6: ta có giá trị $i = 3$

$$i = i + 3 \rightarrow i = 6$$

$$i += 3 \rightarrow i = 6 \equiv i = i + 3$$

$$i *= 3 \rightarrow i = 9 \equiv i = i * 3$$

+ Phép toán tăng, giảm: ++, --

Toán tử ++ sẽ cộng thêm 1 vào toán hạng của nó, toán tử -- sẽ trừ đi 1.

Ví dụ 7: ta có giá trị $n = 6$

+ Sau phép tính ++n hoặc n++, ta có $n = 7$.

+ Sau phép tính --n hoặc n--, ta có $n = 5$.

* Sự khác nhau giữa ++n và n++, --n và n--

+ Sau phép tính $x = ++n + 2$, ta có $x = 9$. (n tăng 1 cộng với 2 rồi gán cho x)

+ Sau phép tính $x = n++ + 2$, ta có $x = 8$. (n cộng với 2 gán cho x rồi mới tăng 1)

5.3. Độ ưu tiên của các phép toán.

| Độ ưu tiên | Các phép toán | Trình tự kết hợp |
|------------|-----------------------------------|------------------|
| 1 | () [] -> | Trái sang phải |
| 2 | ! ~ & * - ++ -- (type) sizeof | Phải sang trái |
| 3 | * / % | Trái sang phải |
| 4 | + - | Trái sang phải |
| 5 | << >> | Trái sang phải |
| 6 | < <= > >= | Trái sang phải |
| 7 | == != | Trái sang phải |
| 8 | & | Trái sang phải |
| 9 | ^ | Trái sang phải |
| 10 | | Trái sang phải |
| 11 | && | Trái sang phải |
| 12 | | Trái sang phải |
| 13 | ? : | Phải sang trái |
| 14 | = += -= *= /= %= <<= >>= &= ^= = | Phải sang trái |
| 15 | , | Trái sang phải |

* Lưu ý:

- Phép đảo (–) ở dòng 2, phép trừ (–) ở dòng 4
- Phép lấy địa chỉ (&) ở dòng 2, phép AND bit (&) ở dòng 8
- Phép lấy đối tượng con trỏ (*) ở dòng 2, phép nhân (*) ở dòng 3.

6. Xuất/Nhập dữ liệu

6.1. Hàm printf

Kết xuất dữ liệu được định dạng.

Cú pháp

printf ("chuỗi định dạng" [, đối mục 1, đối mục 2, ...]);

Khi sử dụng hàm phải khai báo tiền xử lý `#include <stdio.h>`

- printf: tên hàm, **phải viết bằng chữ thường**.
- **đối mục 1,...**: là các mục dữ kiện cần in ra màn hình. Các đối mục này có thể là biến, hằng hoặc biểu thức phải được định trị trước khi in ra.
- chuỗi định dạng: được đặt trong cặp nháy kép (" "), gồm 3 loại:
 - + Đối với chuỗi kí tự ghi như thế nào in ra giống như vậy.
 - + Đối với những kí tự chuyển đổi dạng thức cho phép kết xuất giá trị của các đối mục ra màn hình tạm gọi là mã định dạng. Sau đây là các dấu mô tả định dạng:
 - %c : Kí tự đơn
 - %s : Chuỗi
 - %d : Số nguyên thập phân có dấu
 - %f : Số chấm động (ký hiệu thập phân)
 - %e : Số chấm động (ký hiệu có số mũ)
 - %g : Số chấm động (%f hay %g)
 - %x : Số nguyên thập phân không dấu
 - %u : Số nguyên hex không dấu
 - %o : Số nguyên bát phân không dấu
 - l : Tiền tố dùng kèm với %d, %u, %x, %o để chỉ số nguyên dài (ví dụ %ld)
 - + Các ký tự điều khiển và ký tự đặc biệt
 - \n : Nhảy xuống dòng kế tiếp canh về cột đầu tiên.
 - \t : Canh cột tab ngang.
 - \r : Nhảy về đầu hàng, không xuống hàng.
 - \a : Tiếng kêu bip.
 - \\ : In ra dấu \
 - \" : In ra dấu "
 - \' : In ra dấu '
 - %% : In ra dấu %

Ví dụ 1:

```
printf("Bai hoc C dau tien. \n");
```

→ ký tự điều khiển
→ chuỗi ký tự

Kết quả in ra màn hình: Bai hoc C dau tien.

Ví dụ 2:

```
printf("Ma dinh dang \\\" in ra dau \\' . \n");
```

→ ký tự điều khiển
→ ký tự đặc biệt
→ chuỗi ký tự

Kết quả in ra màn hình: Ma dinh dang\"in ra dau'

Ví dụ 3: Giả sử i có giá trị i = 5

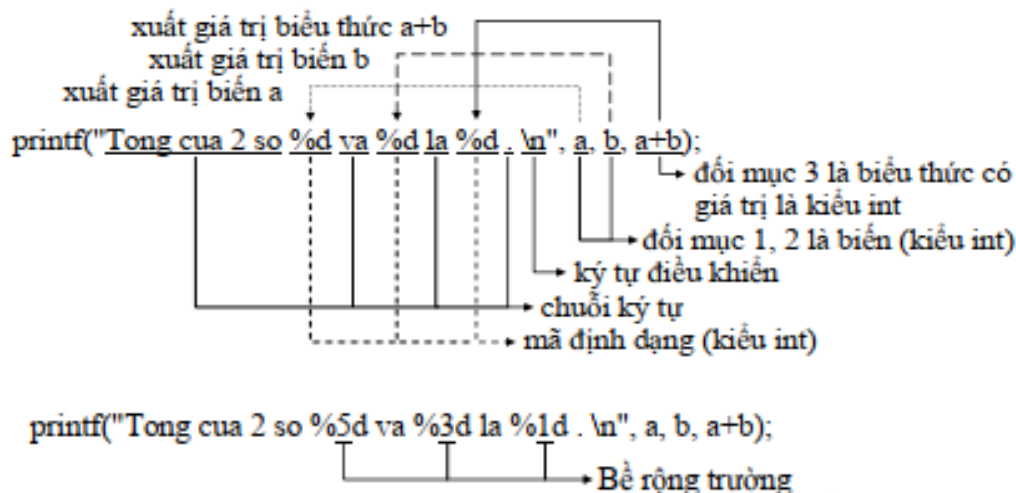
```
printf("So ban vua nhap la: %d . \n", i);
```

→ xuất giá trị biến i
→ đối mục là biến (kiểu int)
→ ký tự điều khiển
→ chuỗi ký tự
→ mã định dạng (kiểu int)

Kết quả in ra màn hình: so bạn vừa nhập: 5

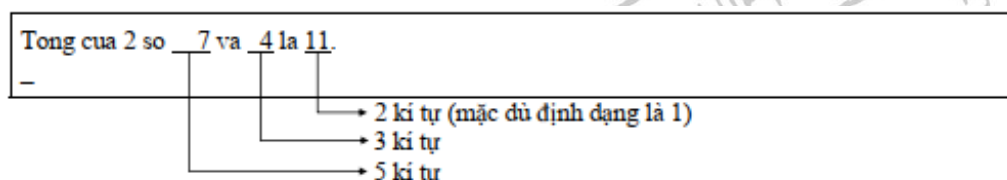
Ví dụ 4: giả sử biến a có giá trị = 7 và b có giá trị = 4

Kết quả in ra màn hình: Tổng của 2 số 7 và 4 là 11.



Ví dụ 5: Mở rộng ví dụ 4.

Kết quả in ra màn hình:



6.2. Hàm scanf

Định dạng khi nhập liệu.

Cú pháp

scanf ("chuỗi định dạng", đối mục 1, đối mục 2,...);

Khi sử dụng hàm phải khai báo tiền xử lý #include <stdio.h>

- **scanf**: tên hàm, **phải viết bằng chữ thường**.
- **khung định dạng**: được đặt trong cặp nháy kép (" ") là hình ảnh dạng dữ liệu nhập vào.
- **đối mục 1, ...**: là danh sách các đối mục cách nhau bởi dấu phẩy, mỗi đối mục sẽ tiếp nhận giá trị nhập vào.

Ví dụ 6: scanf("%d",&i);

- Nhập vào 12abc, biến i chỉ nhận giá trị 12. Nhập 3.4 chỉ nhận giá trị 3.

Ví dụ 7: scanf("%d%d", &a, &b);

- Nhập vào 2 số a, b phải cách nhau bằng **khoảng trắng** hoặc **enter**.

Ví dụ 8: scanf("%d/%d/%d", &ngay, &thang, &nam);

- Nhập vào ngày, tháng, năm theo dạng ngày/thang/nam (20/12/2002)

Ví dụ 9: scanf("%d%*c%d%*c%d", &ngay, &thang, &nam);

- Nhập vào ngày, tháng, năm với dấu phân cách /, -, ...; ngoại trừ số.

Ví dụ 10: scanf("%2d%2d%4d", &ngay, &thang, &nam);

- Nhập vào ngày, tháng, năm theo dạng dd/mm/yyyy.

BÀI TẬP:

1. Giả sử a, b, c là biến kiểu `int` với $a = 8, b = 3$ và $c = 5$. Xác định giá trị các biểu thức sau:

| | | | | | |
|-------------|--|-----------------------|--|-------------------------|--|
| $a + b + c$ | | $a \% c * 2$ | | $a * (a \% b)$ | |
| $a / b - c$ | | $2 * b + 3 * (a - c)$ | | $a * (b + (c - 4 * 3))$ | |
| $a + c / a$ | | $c * (b / a)$ | | $5 * a - 6 / b$ | |
| $a \% b$ | | $(a * b) \% c$ | | $5 \% b \% c$ | |

2. Giả sử x, y, z là biến kiểu `float` với $x = 8.8, y = 3.5$ và $z = 5.2$. Xác định giá trị các biểu thức sau:

| | | | | | |
|-----------------------|--|-------------------|--|---------------------------------|--|
| $x + y + z$ | | $z / (y + x)$ | | $x / y - z * y$ | |
| $5 * y + 6 * (x - z)$ | | $(z / y) + x$ | | $2.5 * x / z - (y + 6)$ | |
| x / z | | $2 * y / 3 * z$ | | $5 * 6 / ((x + y) / z)$ | |
| $x \% z$ | | $2 * y / (3 * z)$ | | $x / y * (6 + ((z - y) + 3.4))$ | |

3. Cho chương trình C với các khai báo và khởi tạo các biến như sau:

`int i = 8, j = 5;`

`float x = 0.005, y = -0.01;`

`char c = 'c', d = 'd';`

Hãy xác định giá trị trả về của các biểu thức sau:

| | | | |
|--|--|--|--|
| $(3 * i - 2 * j) \% (4 * d - c)$ | | $c < d$ | |
| $2 * ((i / 4) + (6 * (j - 3)) \% (i + j - 4))$ | | $x >= 0$ | |
| $(i - 7 * j) \% (c + 3 * d) / (x - y)$ | | $x < y$ | |
| $-(i + j) * -1$ | | $j != 6$ | |
| <code>++i</code> | | $c == 99$ | |
| <code>i++</code> | | $d != 100$ | |
| $i++ + 5$ | | $5 * (i + j + 1) > 'd'$ | |
| $++i + 5$ | | $(3 * x + y) == 0$ | |
| <code>j--</code> | | $2 * x + (y == 0)$ | |
| <code>--j</code> | | $!(i < j)$ | |
| $j-- + i$ | | $!(d == 100)$ | |
| $--j--5$ | | $!(x < 0)$ | |
| <code>++x</code> | | $(i > 0) \&\& (j < 6)$ | |
| <code>y--</code> | | $(i > 0) !! (j < 5)$ | |
| $i >= j$ | | $(x > y) \&\& (i > 0) \parallel (j < 5)$ | |

4. Cho chương trình có các khai báo biến và khởi tạo như sau:

`int i = 8, j = 5, k;`

`float x = 0.005, y = -0.01, z;`

char a, b, c = 'c', d = 'd';

Xác định giá trị các biểu thức gán sau:

| | | | | | |
|---------------------|--|------------------------|--|-----------------------------------|--|
| $k = (i + j * 4)$ | | $z = i / j$ | | $i \% = j$ | |
| $x = (x + y * 1.2)$ | | $a = b = d$ | | $i += (j - 3)$ | |
| $i = j$ | | $y -= x$ | | $k = (j == 5) ? i : j$ | |
| $k = (x + y)$ | | $x *= 2$ | | $k = (j > 5) ? i : j$ | |
| $k = c$ | | $i /= j$ | | $i += j *= i /= 2$ | |
| $i = j = 1.1$ | | $i += 2$ | | $a = (c < d) ? c : d$ | |
| $z = k = x$ | | $z = (x >= 0) ? x : 0$ | | $i -= (j > 0) ? j : 0$ | |
| $k = z = x$ | | $z = (y >= 0) ? y : 0$ | | $i = (i * 9 * (3 + (8 * j / 3)))$ | |

- Viết chương trình đọc và 2 số nguyên và in ra kết quả của phép (+), phép trừ (-), phép nhân (*), phép chia (/). Nhận xét kết quả chia 2 số nguyên.
- Viết chương trình nhập vào bán kính hình cầu, tính và in ra diện tích, thể tích của hình cầu đó.
Hướng dẫn: $S = 4\pi R^2$ và $V = (4/3)\pi R^3$.
- Viết chương trình nhập vào một số a bất kỳ và in ra giá trị bình phương (a^2), lập phương (a^3) của a và giá trị a^4 .
- Viết chương trình đọc từ bàn phím 3 số nguyên biểu diễn ngày, tháng, năm và xuất ra màn hình dưới dạng "ngày/tháng/năm" (chỉ lấy 2 số cuối của năm).
- Viết chương trình nhập vào số giây từ 0 đến 86399, đổi số giây nhập vào thành dạng "gio:phut:giay", mỗi thành phần là một số nguyên có 2 chữ số. (Ví dụ: 11:11:20)

BÀI 4. CẤU TRÚC Rẽ NHÁNH CÓ ĐIỀU KIỆN

(Cấu trúc chọn)

Mục tiêu

Sau khi hoàn tất bài này học viên sẽ hiểu và vận dụng các kiến thức kỹ năng cơ bản sau:

- Ý nghĩa lệnh, khối lệnh.
- Cú pháp, ý nghĩa, cách sử dụng lệnh if, lệnh switch.
- Một số bài toán sử dụng lệnh if, switch thông qua các ví dụ.
- So sánh, đánh giá một số bài toán sử dụng lệnh if hoặc switch.
- Cách sử dụng các cấu trúc lồng nhau.

Nội dung

1. Lệnh if

Câu lệnh if cho phép lựa chọn một trong hai nhánh tùy thuộc vào giá trị của biểu thức luận lý là đúng (true) hay sai (false) hoặc khác không hay bằng không.

a. Dạng 1 (if thiếu)

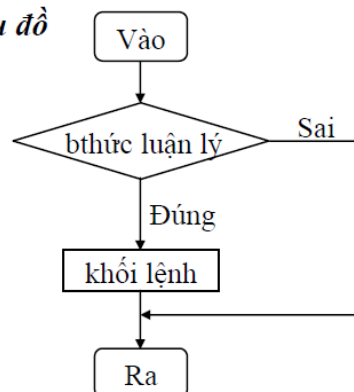
Quyết định sẽ thực hiện hay không một khối lệnh.

Cú pháp lệnh

if (biểu thức luận lý)

khối lệnh;

- Lưu đồ



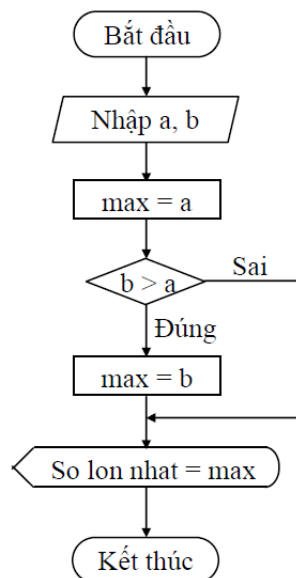
Hình 4.1. Lưu đồ lệnh if thiếu.

Ví dụ 1: Viết chương trình nhập vào 2 số nguyên a, b. Tìm và in ra số lớn nhất.

a. Phác họa lời giải

Trước tiên ta cho giá trị a là giá trị lớn nhất bằng cách gán a cho max (max là biến được khai báo cùng kiểu dữ liệu với a, b). Sau đó so sánh b với a, nếu b lớn hơn a ta gán b cho max và cuối cùng ta được kết quả max là giá trị lớn nhất.

b. Mô tả bằng lưu đồ



Hình 4.2. Lưu đồ giải thuật cho ví dụ 1.

c. Viết chương trình

```

#include <stdio.h>
#include <conio.h>
void main(void)
{
    int ia, ib, imax;
    printf("Nhap vao so a: ");
    scanf("%d", &ia);
    printf("Nhap vao so b: ");
    scanf("%d", &ib);
    imax = ia;
    if (ib > ia)
        imax = ib;
    printf("So lon nhat = %d.\n", imax);
    getch();
}
  
```

b. Dạng 2 (if đủ)

Quyết định sẽ thực hiện 1 trong 2 khối lệnh cho trước.

Cú pháp lệnh

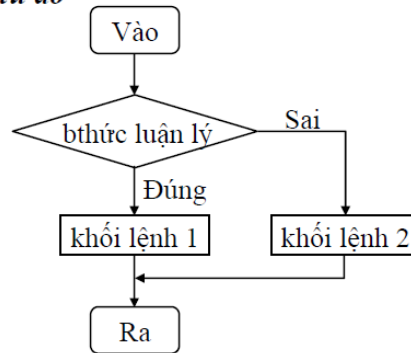
if (biểu thức luận lý)

khối lệnh 1;

else

khối lệnh 2;

- **Lưu đồ**



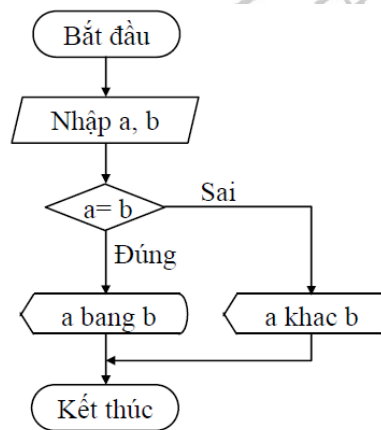
Hình 4.3. Lưu đồ giải thuật hàm if đủ

Ví dụ 2: Viết chương trình nhập vào 2 số nguyên a, b. In ra thông báo "a bằng b" nếu $a = b$, ngược lại in ra thông báo "a khác b".

a. Phác họa lời giải

So sánh a với b, nếu a bằng b thì in ra câu thông báo "a bằng b", ngược lại in ra thông báo "a khác b".

b. Mô tả bằng lưu đồ



Hình 4.4. Lưu đồ giải thuật cho ví dụ 2.

c. Viết chương trình

```

#include <stdio.h>
#include <conio.h>
void main(void)
{
    int ia, ib;
    printf("Nhap vao so a: ");
    scanf("%d", &ia);
    printf("Nhap vao so b: ");
    scanf("%d", &ib);
    if (ia == ib)
        printf("a bang b\n");
  
```

```

else
printf("a khác b\n");
getch();
}

```

c. Cấu trúc else if

Quyết định sẽ thực hiện 1 trong n khối lệnh cho trước.

Cú pháp lệnh

```

if (biểu thức luận lý 1)
khối lệnh 1;
else if (biểu thức luận lý 2)
khối lệnh 2;
...
else if (biểu thức luận lý n-1)
khối lệnh n-1;
else
khối lệnh n;

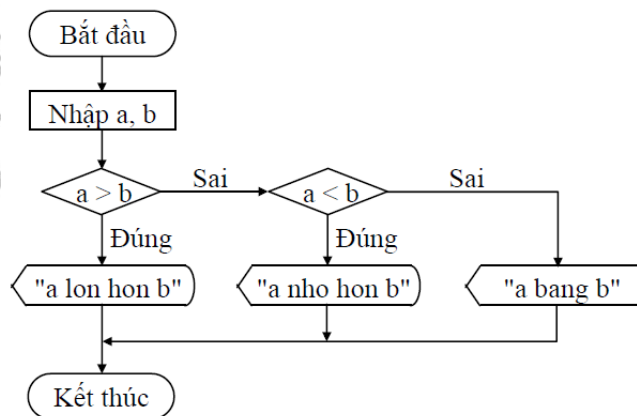
```

Ví dụ 3: Viết chương trình nhập vào 2 số nguyên a, b. In ra thông báo "a lớn hơn b" nếu $a > b$, in ra thông báo "a nhỏ hơn b" nếu $a < b$, in ra thông báo "a bằng b" nếu $a = b$.

a. Phác họa lời giải

Trước tiên so sánh a với b. Nếu $a > b$ thì in ra thông báo "a lớn hơn b", ngược lại nếu $a < b$ thì in ra thông báo "a nhỏ hơn b", ngược với 2 trường hợp trên thì in ra thông báo "a bằng b".

b. Mô tả bằng lưu đồ



Hình 4.5. Lưu đồ giải thuật cho ví dụ 3.

c. Viết chương trình

```

#include <stdio.h>
#include <conio.h>
void main(void)
{
    int ia, ib;
    printf("Nhap vao so a: ");
    scanf("%d", &ia);

```

```

printf("Nhap vao so b: ");
scanf("%d", &ib);
if (ia>ib)
printf("a lon hon b.\n");
else if (ia<ib)
printf("a nho hon b.\n");
else
printf("a bang b.\n");
getch();
}

```

d. Cấu trúc if lồng:

Quyết định sẽ thực hiện 1 trong n khối lệnh cho trước.

Cú pháp lệnh

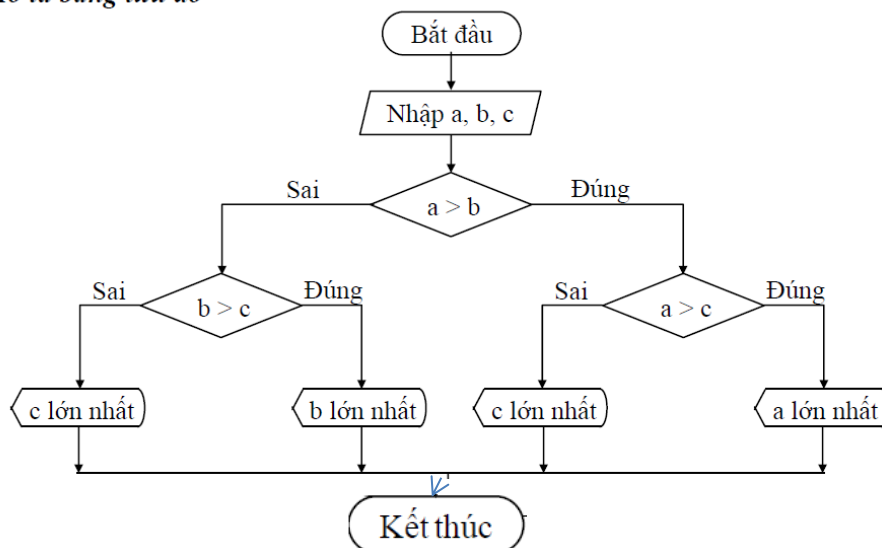
Cú pháp là một trong 3 dạng trên, nhưng trong 1 hoặc nhiều khối lệnh bên trong phải chứa ít nhất một trong 3 dạng trên gọi là cấu trúc if lồng nhau.

Ví dụ 4: Viết chương trình nhập vào 3 số nguyên a, b, c. Tìm và in ra số lớn nhất.

Phác họa lời giải

Trước tiên bạn so nếu $a > b$, mà $a > c$ thì a lớn nhất, ngược lại c lớn nhất, còn nếu $a \leq b$, mà $c > b$ thì b lớn nhất, ngược lại c lớn nhất.

Mô tả bằng lưu đồ



Hình 4.6. Lưu đồ giải thuật cho ví dụ 4.

Viết chương trình

```

#include <stdio.h>
#include <conio.h>
void main(void)
{

```



```

int ia, ib, ic;
printf("Nhap vao so a: ");
scanf("%d", &ia);
printf("Nhap vao so b: ");
scanf("%d", &ib);
    printf("Nhap vao so c: ");
scanf("%d", &ic);
if (ia > ib)
if (ia > ic)
printf("%d lon nhat.\n", ia);
else
printf("%d lon nhat.\n", ic);
else
if (ib > ic)
printf("%d lon nhat.\n", ib);
else
printf("%d lon nhat.\n", ic);
getch();
}

```

Lệnh switch

Lệnh switch cũng giống cấu trúc else if, nhưng nó mềm dẻo hơn và linh động hơn nhiều so với sử dụng if. Tuy nhiên, nó cũng có mặt hạn chế là kết quả của biểu thức phải là giá trị hằng nguyên (có giá trị cụ thể). Một bài toán sử dụng lệnh switch thì cũng có thể sử dụng if, nhưng ngược lại còn tùy thuộc vào giải thuật của bài toán.

1. Cấu trúc switch...case (switch thiếu)

Chọn thực hiện 1 trong n lệnh cho trước.

Cú pháp lệnh

```

switch (biểu thức)
{
case giá trị 1 : lệnh 1;
    break;
case giá trị 2 : lệnh 2
    break;
...
case giá trị n : lệnh n;
    [break;]
}

```

Ví dụ 5: Viết chương trình nhập vào tháng và in ra quý. (tháng 1 -> quý 1, tháng 10 -> quý 4)

a. Phác họa lời giải

Nhập vào giá trị tháng, kiểm tra xem tháng có hợp lệ (trong khoảng 1 đến 12). Nếu hợp lệ in ra quý tương ứng (1->3: quý 1, 4->6: quý 2, 7->9: quý 3, 10->12: quý 4).

b. Viết chương trình

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
    int ithang;
    printf("Nhap vao thang: ");
    scanf("%d", &ithang);
    if (ithang > 0 && ithang <= 12)
        switch(ithang)
        {
            case 1:
            case 2:
            case 3: printf("Quy 1.\n");
                    break;
            case 4:
            case 5:
            case 6: printf("Quy 2.\n");
                    break;
            case 7:
            case 8:
            case 9: printf("Quy 3.\n");
                    break;
            case 10:
            case 11:
            case 12: printf("Quy 4.\n");
                    break;
        };
    else
        printf("Thang khong hop le.\n");
    getch();
}
```

b. Cấu trúc switch...case...default (switch đủ)

Chọn thực hiện 1 trong $n + 1$ lệnh cho trước.

Cú pháp lệnh

```
switch (biểu thức)
{
    case giá trị 1 : lệnh 1;
        break;
    case giá trị 2 : lệnh 2;
        break;
    ...
    case giá trị n : lệnh n;
        break;
    default : lệnh;
        [break;]
}
```

Ví dụ 6: Viết lại chương trình ở **Ví dụ 5**

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
    int ithang;
    printf("Nhap vao thang: ");
    scanf("%d", &ithang);
    switch(ithang)
    {
        case 1: case 2: case 3 : printf("Quy 1.\n");
            break;
        case 4: case 5: case 6: printf("Quy 2.\n");
            break;
        case 7: case 8: case 9: printf("Quy 3.\n");
            break;
        case 10: case 11: case 12: printf("Quy 4.\n");
            break;
        default : printf("Ban phai nhap vao so trong khoang 1..12\n");
    };
}
```

```
    getch();  
}
```

BÀI TẬP

Sử dụng lệnh if

1. Viết chương trình nhập vào số nguyên dương, in ra thông báo số chẵn hay lẻ.
2. Viết chương trình nhập vào 4 số nguyên. Tìm và in ra số lớn nhất.
3. Viết chương trình giải phương trình bậc 2: $ax^2 + bx + c = 0$, với a, b, c nhập vào từ bàn phím.

Sử dụng lệnh switch

1. Viết chương trình nhập vào tháng, in ra tháng đó có bao nhiêu ngày.

Hướng dẫn: Nhập vào tháng

Nếu là tháng 1, 3, 5, 7, 8, 10, 12 thì có 30 ngày

Nếu là tháng 4, 6, 9, 11 thì có 31 ngày

Nếu là tháng 2 và là năm nhuận thì có 29 ngày ngược lại 28 ngày
(Năm nhuận là năm chia chắn cho 4)

2. Viết chương trình trò chơi One-Two-Three ra cái gì ra cái này theo điều kiện:

- Búa (B) thắng Kéo, thua Giấy.
- Kéo (K) thắng Giấy, thua Búa.
- Giấy (G) thắng Búa, thua Kéo.

BÀI 5: CẤU TRÚC VÒNG LẶP

Mục tiêu:

Sau khi hoàn tất bài này sinh viên sẽ hiểu và vận dụng các kiến thức kỹ năng cơ bản sau:

- Ý nghĩa, cách hoạt động của vòng lặp.
- Cú pháp, ý nghĩa, cách sử dụng lệnh for, while, do...while.
- Ý nghĩa và cách sử dụng lệnh break, continue.
- Một số bài toán sử dụng lệnh for, while, do...while thông qua các ví dụ.
- Cấu trúc vòng lặp lồng nhau.

Nội dung:

- Cú pháp và cách dùng vòng lặp for
- Cú pháp và cách dùng vòng lặp while
- Cú pháp và cách dùng vòng lặp do...while
- Cú pháp và cách dùng lệnh break
- Cú pháp và cách dùng lệnh continue



1. Tóm tắt lý thuyết về cấu trúc vòng lặp

1.1. Vòng lặp for

a. Cách sử dụng: vòng lặp *for* là vòng lặp xác định, thực hiện lặp lại một số lần xác định của một hay nhiều hành động

b. Cú pháp lệnh:

```
for (biểu thức 1; biểu thức 2; biểu thức 3)
    khối lệnh;
```

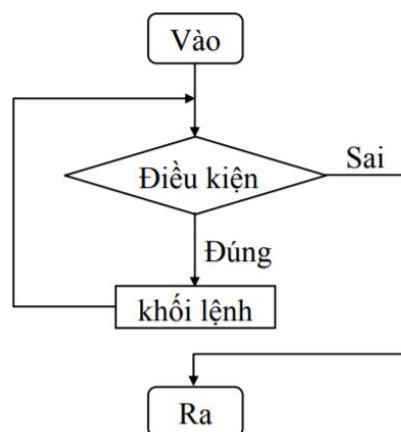
Giải thích:

- + Biểu thức 1: khởi tạo giá trị ban đầu cho biến điều khiển.
- + Biểu thức 2: là quan hệ logic thể hiện điều kiện tiếp tục vòng lặp.
- + Biểu thức 3: phép gán dùng thay đổi giá trị biến điều khiển.
- + Nếu khối lệnh bao gồm từ 2 lệnh trở lên thì phải đặt trong dấu { }

Lưu ý:

- + Nếu biểu thức 2 không có, vòng for được xem là luôn luôn đúng. Muốn thoát khỏi vòng lặp for phải dùng một trong 3 lệnh break, goto hoặc return.
- + Trong thân for (khối lệnh) có thể chứa một hoặc nhiều cấu trúc điều khiển khác.
- + Khi gặp lệnh break, cấu trúc lặp sâu nhất sẽ thoát ra.
- + Trong thân for có thể dùng lệnh goto để thoát khỏi vòng lặp đến vị trí mong muốn.
- + Trong thân for có thể sử dụng return để trở về một hàm nào đó.
- + Trong thân for có thể sử dụng lệnh continue để chuyển đến đầu vòng lặp (bỏ qua các câu lệnh còn lại trong thân).

c. Lưu đồ:



Giải thích: kiểm tra điều kiện, nếu đúng thì thực hiện khối lệnh, lặp lại kiểm tra điều kiện, nếu sai thoát khỏi vòng lặp.

1.2. Vòng lặp while

a. Cách sử dụng

Vòng lặp *while* thực hiện lặp lại một hay nhiều hành động trong khi biểu thức còn đúng.

b. Cú pháp lệnh:

```
while (biểu thức)
khởi lệnh;
```

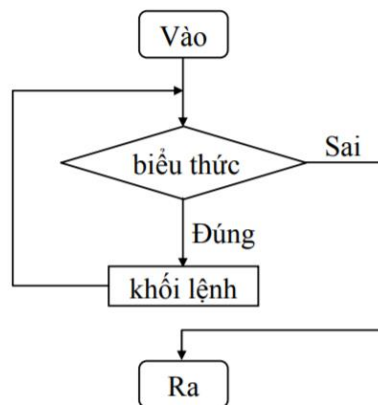
Giải thích:

- + Biểu thức: có thể là một biểu thức hoặc nhiều biểu thức con. Nếu là nhiều biểu thức con thì cách nhau bởi dấu phẩy (,) và tính đúng sai của biểu thức được quyết định bởi biểu thức con cuối cùng.
- + Nếu khối lệnh bao gồm từ 2 lệnh trở lên thì phải đặt trong dấu { }

Lưu ý:

- + Trong thân while (khối lệnh) có thể chứa một hoặc nhiều cấu trúc điều khiển khác.
- + Trong thân while có thể sử dụng lệnh continue để chuyển đến đầu vòng lặp
- + Muốn thoát khỏi vòng lặp while tùy ý có thể dùng các lệnh break, goto, return như lệnh for.

c. Lưu đồ:



Giải thích:

Trước tiên biểu thức được kiểm tra, nếu sai thì kết thúc vòng lặp while (khối lệnh không được thi hành 1 lần nào), nếu đúng thực hiện khối lệnh. Lặp lại kiểm tra biểu thức.

1.3. Vòng lặp do...while

a. Cách sử dụng

Vòng lặp *do....while* thực hiện lặp lại một hay nhiều hành động cho đến khi biểu thức sai.

b. Cú pháp lệnh:

```
do
khởi lệnh;
while (biểu thức);
```

Giải thích:

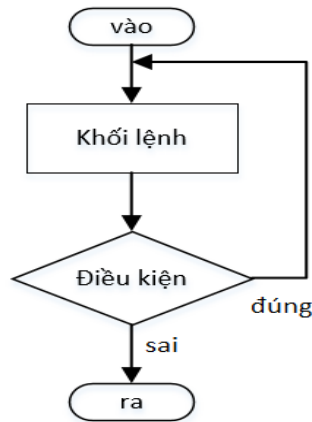
- + Biểu thức: có thể là một biểu thức hoặc nhiều biểu thức con. Nếu là nhiều biểu thức con thì cách nhau bởi dấu phẩy (,) và tính đúng sai của biểu thức được quyết định bởi biểu thức con cuối cùng.
- + Nếu khối lệnh bao gồm từ 2 lệnh trở lên thì phải đặt trong dấu { }

Lưu ý:

- + Trong thân while (khối lệnh) có thể chứa một hoặc nhiều cấu trúc điều khiển khác.

- + Trong thân while có thể sử dụng lệnh `continue` để chuyển đến đầu vòng lặp
- + Muốn thoát khỏi vòng lặp while tùy ý có thể dùng các lệnh `break`, `goto`, `return` như lệnh `for`.

c. Lưu đồ:



Giải thích:

Thực hiện khởi lệnh. Kiểm tra biểu thức. Nếu đúng thì lặp lại thực hiện khởi lệnh. Nếu sai thì kết thúc vòng lặp (khởi lệnh được thi hành 1 lần).

1.4. lệnh break

a. Cách sử dụng

- Lệnh `break` dùng để thoát khỏi vòng lặp không xác định điều kiện dừng hoặc bạn muốn dừng vòng lặp theo điều kiện do bạn chỉ định.
- Lệnh `break` để thoát khỏi vòng lặp thường sử dụng phối hợp với lệnh `if`.
- Lệnh `break` thường dùng trong `for`, `while`, `do...while`, `switch`.

b. Cú pháp lệnh:

```
break;
```

1.5. lệnh continue

a. Cách sử dụng

- Lệnh `continue` thi hành quyền điều khiển sẽ trao qua cho biểu thức điều kiện của vòng lặp gần nhất. Nghĩa là lộn ngược lên đầu vòng lặp, tất cả những lệnh đi sau trong vòng lặp chứa `continue` sẽ bị bỏ qua không thi hành.
- Lệnh `continue` thường dùng trong vòng lặp `for`, `while`, `do...while`.

b. Cú pháp lệnh:

```
continue;
```

2. Các ví dụ về cấu trúc vòng lặp

2.1. Ví dụ về vòng lặp for

Ví dụ 1: Viết chương trình in ra 3 lần câu "Vi dụ su dung vong lap for"

Bài giải

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

```
#include <conio.h>
#define MSG "Vi du su dung vong lap for.\n"
void main(void)
{
    int i; for(i = 1; i<=3; i++)
        printf("%s", MSG);
    getch();
}
```

Ví dụ 2: Viết chương trình nhập vào 3 số và tính tổng

Bài giải

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
    int i, in, is;
    is = 0;
    for(i = 1; i<=3; i++)
    {
        printf("Nhap vao so thu %d :", i);
        scanf("%d", &in);
        is = is + in;
    }
    printf("Tong: %d", is);
    getch();
}
```

Ví dụ 3: Viết chương trình nhập vào số nguyên n. Tính tổng các giá trị lẻ từ 0 đến n.

Bài giải

```
#include <stdio.h>
#include <conio.h>
void main(void)
{
    int i, in, is = 0;
    printf("Nhap vao so n: ");
    scanf("%d", &in);
    is = 0;
    for(i = 0; i<=in; i++)
```

```

    {
        if (i % 2 != 0)                //neu i la so le
            is = is + i;                //hoac is += i;
        }
    printf("Tong: %d", is);
    getch();
}

```

2.2. Ví dụ về vòng lặp while

Ví dụ 1: Viết chương trình in ra câu "Vi dụ su dung vong lap while" 3 lần.

Bài giải

```

#include <stdio.h>
#include <conio.h>
#define MSG "Vi dụ su dung vong lap while.\n"
void main (void)
{
    int i = 0;
    while (i++ < 3)
        printf("%s", MSG);
    getch();
}

```

Ví dụ 2: Viết chương trình tính tổng các số nguyên từ 1 đến n, với n được nhập vào từ bàn phím.

Bài giải

```

#include <stdio.h>
#include <conio.h>
void main(void)
{
    int i = 0, in, is = 0;
    printf("Nhap vao so n: ");
    scanf("%d", &in);
    while (i++ < in) is = is + i;    //hoac is += i;
    printf("Tong: %d", is);
    getch();
}

```

2.3. Ví dụ về vòng lặp do...while

Ví dụ 1: Viết chương trình kiểm tra password.

Bài giải

```

#include <stdio.h>
#include <conio.h>
# define PASSWORD 12345
void main(void)
{
    int in;
    do
    {
        printf("Nhap vao password: ");
        scanf("%d", &in);
    }
    while (in != PASSWORD)
}

```

Ví dụ 2: Viết chương trình nhập vào năm hiện tại, năm sinh. In ra tuổi
 Bài giải

```

#include <stdio.h>
#include <conio.h>
# define CHUC "Chuc ban vui ve (: >\n"
void main(void)
{
    unsigned char choi;
    int inamhtai,
    inamsinh;
    do
    {
        printf("Nhap vao nam hien tai: ");
        scanf("%d", &inamhtai);
        printf("Nhap vao nam sinh: ");
        scanf("%d", &inamsinh);
        printf("Ban %d tuoi, %s", inamhtai - inamsinh, CHUC);
        printf("Ban co muon tiep tuc? (Y/N)\n");
        choi = getch();
    }
    while (choi == 'y' || choi == 'Y');
}

```

BÀI TẬP

Bài tập 1: Viết chương trình in ra bảng mã ASCII

Bài tập 2: Viết chương trình tính tổng bậc 3 của N số nguyên đầu tiên.

Bài tập 3: Viết chương trình nhập vào một số nguyên rồi in ra tất cả các ước số của số đó.

Bài tập 4: Viết chương trình vẽ một tam giác cân bằng các dấu *

Bài tập 5: Viết chương trình tính tổng nghịch đảo của N số nguyên đầu tiên theo công thức

$$S = 1 + 1/2 + 1/3 + \dots + 1/N$$

Bài tập 6: Viết chương trình tính tổng bình phương các số lẻ từ 1 đến N.

Bài tập 7: Viết chương trình nhập vào N số nguyên, tìm số lớn nhất, số nhỏ nhất.

Bài tập 8: Viết chương trình nhập vào N rồi tính giai thừa của N.

Bài tập 9: Viết chương trình tìm USCLN, BSCNN của 2 số.

Bài tập 10: Viết chương trình vẽ một tam giác cân rộng bằng các dấu *.

Bài tập 11: Viết chương trình nhập vào một số và kiểm tra xem số đó có phải là số nguyên tố hay không?

Bài tập 12: Viết chương trình tính x.n với x, n được nhập vào từ bàn phím.

Bài tập 13: Viết chương trình nhập vào 1 số từ 0 đến 9. In ra chữ số tương ứng. Ví dụ: nhập vào số 5, in ra "Năm".

Bài tập 14: Viết chương trình phân tích một số nguyên N thành tích của các thừa số nguyên tố.

Bài tập 15: Viết chương trình tìm ước số chung lớn nhất và bội số chung nhỏ nhất của 2 số nguyên.

Bài tập 16: Viết chương trình tính dân số của một thành phố sau 10 năm nữa, biết rằng dân số hiện nay là 6.000.000, tỉ lệ tăng dân số hàng năm là 1.8%.

Bài tập 17: Viết chương trình tìm các số nguyên gồm 3 chữ số sao cho tích của 3 chữ số bằng tổng 3 chữ số. Ví dụ: $1*2*3 = 1+2+3$.

Bài tập 18: Viết chương trình tìm các số nguyên a, b, c, d khác nhau trong khoảng từ 0 tới 10 thỏa mãn điều kiện $a*d*d = b*c*c*c$

Bài tập 19: Viết chương trình lặp đi lặp lại các công việc sau:

- Nhập vào một ký tự trên bàn phím.
- Nếu là chữ thường thì in ra chính nó và chữ HOA tương ứng.
- Nếu là chữ HOA thì in ra chính nó và chữ thường tương ứng.
- Nếu là ký số thì in ra chính nó.
- Nếu là một ký tự điều khiển thì kết thúc chương trình

Bài tập 20: Viết chương trình nhập vào N số nguyên, đếm xem có bao nhiêu số âm, bao nhiêu số dương và bao nhiêu số không.

BÀI 6. HÀM

Mục tiêu:

Sau khi hoàn tất bài này sinh viên sẽ hiểu và vận dụng các kiến thức kỹ năng cơ bản sau:

- Khái niệm, cách khai báo về hàm.
- Cách truyền tham số, tham biến, tham trị.
- Sử dụng biến cục bộ, toàn cục trong hàm.
- Sử dụng con trỏ hàm

Nội dung:

- Hàm là một đoạn chương trình con thực hiện một khối lượng công việc cụ thể.
- Các hàm được sử dụng để rút gọn cho một chuỗi các chỉ thị được thực hiện lặp đi lặp lại nhiều lần trong khi thực thi chương trình .

1. Tóm tắt lý thuyết về Hàm

1.1. Cấu trúc hàm:

Cú pháp tổng quát của một hàm trong C như sau:

```
type_specifier function_name(arguments)
{
    body of the function
}
```

Trong đó :

- **type_specifier**: xác định kiểu dữ liệu của giá trị mà hàm sẽ trả về.
- **function_name**: tên hàm hợp lệ được gán cho định danh của hàm
- **arguments** : các đối số xuất hiện trong cặp dấu ngoặc () được gọi là các tham số hình thức, các tham số cách nhau bởi dấu phẩy, hàm có thể có tham số hoặc không. Tất cả các tham số của hàm phải được khai báo một cách riêng biệt.

Để viết một chương trình có sử dụng hàm ta phải khai báo hàm trước hàm main() và có thể định nghĩa hàm sau hàm main().

Để gọi hàm ta dùng cú pháp :

```
function_name(arguments1, arguments2, arguments3...);
```

Nếu hàm không có đối số ta gọi hàm theo cú pháp sau:

```
function_name();
```

Ví dụ 1.1: Chương trình xây dựng hàm tính bình phương của các số từ 1 đến 10

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

```
int squarer();
```

```
void main(void)
```

```
{
```

```
    int i;
```

```

        for(i=1; i<=10; i++)
            printf("\n squarer of %d is %d", i, squarer(i));
    }
    int squarer(int x)
    {
        int j;
        j=x*x;
        return(j);
    }

```

- Hàm squarer() được khai báo trước hàm main() và được định nghĩa sau hàm main().
- Dữ liệu được truyền từ hàm main() đến hàm squarer()
- Hàm squarer() sử dụng đối số là x, trong chương trình chính đối số x chính là biến i
- Lệnh return() ngay lập tức chuyển điều khiển từ hàm trở về chương trình gọi.
- Giá trị đặt trong cặp dấu ngoặc () theo sau lệnh return được trả về cho chương trình gọi.
- **type_specifier** là không bắt buộc nếu kiểu của giá trị trả về là một số nguyên hoặc nếu không có giá trị trả về
- Tuy nhiên, để tránh sự không nhất quán, một kiểu dữ liệu nên được xác định.

1.2. Khai báo hàm:

Ví dụ 2.1:

```

#include <stdio.h>
void address();
void main(void){
    ...
    address()
    ...
}
address(){
    ...
}

```

Khai báo kiểu a

```

#include <stdio.h>
void main(void){
    ...
    address()
    ...
}
address(){
    ...
}

```

Khai báo kiểu b

- Khai báo kiểu a: việc khai báo hàm address() là bắt buộc khi hàm được sử dụng trước khi nó được định nghĩa
- Khai báo kiểu b:
 - + Hàm address() được gọi trước khi nó được định nghĩa
 - + Một số trình biên dịch C sẽ thông báo lỗi nếu hàm không được khai báo trước khi gọi
 - + Điều này còn được gọi là sự khai báo không tương minh

1.3. Các biến trong Hàm :

- Biến cục bộ
 - + Được khai báo bên trong một hàm
 - + Được tạo tại điểm vào của một khối và bị hủy tại điểm ra khỏi khối đó

- Tham số hình thức
 - + Được khai báo trong định nghĩa hàm như là các tham số
 - + Hoạt động như một biến cục bộ bên trong một hàm
- Biến toàn cục
 - + Được khai báo bên ngoài tất cả các hàm
 - + Lưu các giá trị tồn tại suốt thời gian thực thi của chương trình

1.4. Lệnh return :

Có hai cách để thoát khỏi hàm : thực hiện đến câu lệnh cuối cùng hoặc gặp lệnh return đầu tiên. Một hàm có thể chứa nhiều lệnh return.

Để trả về giá trị cho hàm ta dùng lệnh return

Cú pháp :

return expression;

Thông thường có ba loại hàm cơ bản :

- Tính toán và trả về một giá trị.
- Hàm thao tác trên thông tin và trả về một giá trị cho biết thao tác có thành công hay không.
- Hàm không trả về một giá trị nào. Tất cả các hàm không trả về giá trị đều khai báo kiểu void.

1.5. Gọi hàm :

- Truyền bằng giá trị (by value): Sao chép giá trị của đối số vào tham số của chương trình con. Trong trường hợp này mọi thay đổi tham số không ảnh hưởng đến đối số.

- Truyền bằng tham chiếu (by reference): Để gửi một tham chiếu đến hàm ta sử dụng con trỏ, địa chỉ của đối số sẽ được truyền đến tham số. Khi đó mọi thay đổi trên tham số sẽ thay đổi đối số.

Ví dụ 5.1 : Minh họa cách truyền tham số cho hàm bằng giá trị, chương trình xây dựng hàm tính bình phương của một số nguyên

```
#include<stdio.h>
int squarer(int x);
void main(void)
{
    int t=10;
    printf(" %d ^2=%d",t, squarer(t));
}
```

```
int squarer(int x)
{
    x=x*x;
    return x;
}
```

Trong ví dụ trên giá trị của đối số t=10 được sao chép đến tham số x. Câu lệnh gán x*x chỉ có biến x thay đổi, trong khi giá trị t vẫn giữ nguyên.

Ví dụ 5.2: Minh họa cách truyền tham số cho hàm bằng tham chiếu, chương trình xây dựng hàm hoán đổi giá trị của hai biến

```
#include<stdio.h>
void swap(int *x, int *y);
void main(void)
```

```

{
    int i, j;
    i=10;
    j=20;
    printf(" truooc khi doi i=%d, j=%d, i, j");
    swap(&i,&j);
    printf(" sau khi doi i=%d, j=%d, i, j");
}
void swap(int *x, int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

```

Trong ví dụ trên hàm swap() dùng để đổi giá trị cho hai biến số nguyên, sau khi thoát khỏi hàm swap() hai biến i và j phải đổi giá trị cho nhau nghĩa là i và j phải thay đổi giá trị không còn giữ giá trị như lúc đầu. Để thực hiện điều này ta phải sử dụng cơ chế truyền tham số theo tham chiếu

- Sự lồng nhau của lời gọi hàm.

Ví dụ 5.3:

| | |
|--|---|
| <pre> void main(void) { palindrome(); } </pre> | <pre> palindrome() { ... getstr(); reverse(); cmp(); ... } </pre> |
|--|---|

1.6. Con trỏ hàm :

- Hàm có một địa chỉ trong bộ nhớ và có thể gán địa chỉ này cho con trỏ.
- Địa chỉ này là điểm vào của hàm và nó là địa chỉ được sử dụng khi gọi hàm
- Nếu có một con trỏ đến hàm, khi đó ta có thể gọi hàm thông qua con trỏ này.

Cú pháp :

type (*name)(arguments);

Ví dụ : float(*f)(float x) : khai báo một con trỏ hàm có tên f, con trỏ hàm này chỉ có thể nhận địa chỉ của những hàm có kiểu trả về là kiểu float và hàm này chỉ có một tham số kiểu float.

Ta có thể nhận địa chỉ của hàm bằng cách sử dụng tên hàm mà không có các tham số

Ví dụ 6.1: Minh họa cách sử dụng một con trỏ hàm để gọi nhiều hàm khác nhau

```

#include<stdio.h>
#include<math.h>
void main(void)
{

```

```

double(*f) double(d); /*khai bao con tro ham*/
double x=3.14159;
/*goi ham sin thong qua con tro f*/
f=sin;
printf(" sin(%0.41f)=%0.41f\n", x, f(x));
/*goi ham cos thong qua con tro f*/
f=cos;
printf(" cos(%0.41f)=%0.41f\n", x, f(x));
}

```

- Trong ví dụ trên ta khai báo một con trỏ hàm f. Con trỏ hàm này chỉ nhận địa chỉ những hàm có kiểu trả về là double và nhận một tham số kiểu double.
- Câu lệnh f=sin; f=cos; gán địa chỉ hàm sin(x), cos(x) trong thư viện math.h vào con trỏ f, sau đó gọi hàm sin(x), cos(x) thông qua con trỏ này.

2. Các ví dụ về Hàm:

Ví dụ 1: Viết chương trình tạo ra 2 dòng dấu *, mỗi dòng là 19 dấu *. Minh họa cách khai báo và gọi hàm line()

```

#include <stdio.h>
#include <conio.h>
// khai bao prototype
void line();
// ham in 1 dong dau *
void line()
{
    int i;
    for(i = 0; i < 19; i++)
        printf("*");
    printf("\n");
}

```

```

void main(void)
{
    line();
    printf("* Minh hoa ve ham *");
    line();
    getch();
}

```

Ví dụ 2: Viết chương trình xây dựng hàm tính số mũ của hai số nguyên.

```

#include <stdio.h>
#include <conio.h>
// khai bao prototype
int power(int ix, int in);
// ham tinh so mu
int power(int ix, int in)
{
    int i, ip = 1;
    for(i = 1; i <= in; i++)
        ip *= ix;
    return ip;
}
for(i = 0; i < 19; i++)
    printf("*");
printf("\n");
}

```

```

void main(void)
{
    printf("2 mu 2 = %d.\n", power(2, 2));
    printf("2 mu 3 = %d.\n", power(2, 3));
    getch();
}

```

- Hàm power có hai tham số truyền vào là ix, in có kiểu int và kiểu trả về cũng có kiểu int.

- return ip: trả về giá trị sau khi tính toán
- Hai tham số ix, in của hàm power là dạng truyền tham trị.

Ví dụ 3: Viết chương trình xây dựng hàm đổi số phút thành giờ :phút

```
#include <stdio.h>
#include <conio.h>
// khai bao prototype
void time(int &ig, int &ip) ;
// ham doi phut thanh gio:phut
void time(int &ig, int &ip)
{
    ig = ip / 60;
    ip %= 60;
}
```

```
void main(void)
{
    int igio, iphut;
    printf("Nhap vao so phut : ");
    scanf("%d", &iphut);
    time(igio, iphut);
    printf("%02d:%02d\n", igio, iphut);
    getch();
}
```

Hàm time() có hai tham số truyền vào là ig, ip có kiểu int. 2 tham số này có toán tử địa chỉ & đi trước cho biết 2 tham số này là dạng truyền tham biến.

Ví dụ 4: Ví dụ minh họa tham số dạng tham biến và tham trị

```
void thamtri(int ix, int iy)
{
    ix += 1; //cong ix them 1
    iy += 1; //cong iy them 1
}
void thambien(int &ix, int &iy)
{
    ix += 1; //cong ix them 1
    iy += 1; //cong iy them 1
}
```

```
void main(void)
{
    int ia = 5, ib = 5;
    thamtri(ia, ib);
    printf("a = %d, b = %d", ia, ib);
    thambien(ia, ib);
    printf("a = %d, b = %d", ia, ib);
}
```

Đối với hàm sử dụng lệnh return bạn chỉ có thể trả về duy nhất 1 giá trị mà thôi. Để có thể trả về nhiều giá trị sau khi gọi hàm bạn sử dụng hàm truyền nhiều tham số dạng tham biến.

Ví dụ 5: Ví dụ minh họa cách sử dụng biến toàn cục trong hàm

```
#include <stdio.h>
#include <conio.h>
// khai bao prototype
void oddeven();
void negative();
//khai bao bien toan cuc
int inum;
void main(void)
{
    printf("Nhap vao 1 so nguyen : ");
    scanf("%d", &inum);
    oddeven();
    negative();
    getch();
}
```

```
// ham kiem tra chan le
void oddeven()
{
    if (inum % 2)
        printf("%d la so le.\n", inum);
    else
        printf("%d la so chan.\n", inum);
}
//ham kiem tra so am
void negative()
{
    if (inum < 0)
        printf("%d la so am.\n", inum);
    else
        printf("%d la so duong.\n", inum);
}
```

BÀI TẬP

1. Viết hàm tính $n!$
2. Viết hàm tính tổng $S = 1+2+...+n$.
3. Viết hàm kiểm tra số nguyên tố.
4. Viết hàm tính số hạng thứ n trong dãy Fibonacci.
5. Viết hàm tìm số lớn nhất trong 2 số.



BÀI 7: CẤU TRÚC DỮ LIỆU KIỂU MẢNG

1. MẢNG MỘT CHIỀU:

1.1 Khái niệm:

Mảng là một tập hợp nhiều phần tử (biến) có cùng một kiểu và chung một tên được sắp xếp liên tiếp nhau trong bộ nhớ.

Mỗi phần tử của mảng có một đại lượng xác định vị trí tương đối của phần tử đó so với các phần tử khác trong mảng, gọi là chỉ số.

1.2. Khai báo mảng một chiều:

Cú pháp:

<Kiểu> <Tên mảng>[<Kích thước>];

Mỗi phần tử của mảng được truy nhập thông qua tên mảng cùng với chỉ số đặt giữa hai ngoặc vuông.

Chỉ số là một số nguyên được đánh số từ 0 đến <Kích thước> - 1.

Ví dụ:

```
int a[6];
```

Câu lệnh trên khai báo một mảng a gồm 6 phần tử kiểu int, bao gồm:

```
a[0], a[1], a[2], a[3], a[4], a[5]
```

1.3. Khởi tạo mảng một chiều:

Để khởi tạo mảng ta liệt kê danh sách các giá trị của chúng trong cặp dấu ngoặc “{” và “}”.

Kích thước mảng, nếu có, không được nhỏ hơn số giá trị có trong danh sách khởi tạo.

Ví dụ:

```
int a[5] = {20, -5, 8, 40, 10};
```

```
int b[] = {100, -25, 18, 10, 18};
```

Ví dụ 1: Nhập vào một số nguyên dương n ($1 \leq n \leq 7$). Tùy theo n, hãy in ra các từ Sun, Mon, ..., Sat tương ứng.

```
void main()
```

```
{
```

```
    char *a[7] = {"Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"}; //Khởi tạo  
    mảng các chuỗi
```

```
    int n;
```

```
    <Nhập n>
```

```
    if(n >= 1 && n <= 7)
```

```
        printf("%s\n", a[n - 1]);
```

```
    else printf("DL nhap khong hop le\n");
```

```
}
```

Ví dụ 2: Nhập tháng và năm. Hãy tính và in ra số ngày trong tháng.

```
void main()
```

```

{
    int m, y;
    <Nhập m, y>
    int a[12] = { 31, Nhuan(y)? 29 : 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    /*Khởi tạo mảng các số nguyên*/
    if(m > 0 && m < 13 && y > 0)
        printf("Số ngày trong tháng %d năm %d là %d\n", m, y, a[m - 1]);
    else
        printf("DL nhập không hợp lệ\n");
}

```

1.4. Nhập xuất mảng một chiều:

Ví dụ:

```

#define SIZE 20
void main()
{
    int a[SIZE]; /*khai báo mảng gồm SIZE phần tử kiểu int*/
    int n; // Lưu số phần tử thực có của mảng
    int i;
    do {
        printf("Nhập số phần tử:");
        scanf("%d", &n);
    } while(n < 1 || n > SIZE);

    // Nhập dữ liệu cho mảng
    for(i = 0; i < n; i++) {
        printf("pt thu %d:", i);
        scanf("%d", &a[i]);
    }

    // Xuất dữ liệu của mảng
    for(i = 0; i < n; i++)
        printf("%d\t", a[i]);
    printf("\n");
}

```

1.5. Dùng mảng một chiều làm tham số truyền cho hàm:

Tên mảng là một hằng địa chỉ và nó chính là địa chỉ phần tử đầu tiên của mảng.

Khi dùng tên mảng làm tham số thực truyền cho hàm thì thực chất là địa chỉ phần tử đầu tiên của mảng được truyền cho hàm và như vậy tham số hình thức tương ứng trong định nghĩa hàm phải viết dưới dạng con trỏ.

Ví dụ:

```
#define SIZE 20
// khai báo các nguyên mẫu hàm
void Nhap(int *a, int *n);
void Xuat(int *a, int n);
// Định nghĩa các hàm
void Nhap(int *a, int *n)
{
    do {
        printf("Nhap so phan tu:");
        scanf("%d", &(*n));
    } while(*n < 1 || *n > SIZE);
    for(int i = 0; i < *n; i++) {
        printf("pt thu %d:", i);
        scanf("%d", &a[i]);
    }
}

void Xuat(int *a, int n) {
    for(int i = 0; i < n; i++)
        printf("%d\t", a[i]);
    printf("\n");
}

void main() {
    int a[SIZE], n;
    Nhap(a, &n);
    Xuat(a, n);
}
```

Lưu ý: Tham số hình thức tương ứng với tham số thực là tên mảng cũng có thể viết như sau:

```
void Nhap(int a[], int *n);
void Xuat(int a[], int n);
```

1.6. Bài tập:

Đối với các bài tập trong chương này, trong chương trình luôn phải có hai hàm nhập và xuất mảng một chiều nguyên.

Bài tập có hướng dẫn

1. Viết chương trình thực hiện các chức năng sau:
 - a. Tính tổng các phần tử có chữ số đầu tiên là chữ số lẻ.
 - b. Tính tổng các phần tử là số chính phương.

Hướng dẫn:

//Câu 1a

```
int ChuSoDauTienLe(int x) {  
    while (x >= 10)  
        x /= 10 ;  
    return (x % 2 != 0 ? 1 : 0) ;  
}
```

```
int TongChuSoDauTienLe(int a[], int n) {  
    int ret = 0;  
    for(i = 0; i < n; i++)  
        if(ChuSoDauTienLe(a[i]))  
            ret += a[i];  
    return ret;  
}
```

//Câu 1b

```
int TongChinhPhuong(int a[], int n) {  
    int ret = 0;  
    for(i = 0; i < n; i++)  
        if(ChinhPhuong(a[i]))  
            ret += a[i];  
    return ret;  
}
```

2. Viết chương trình thực hiện các chức năng sau:
 - a. Đếm số phần tử có chữ số đầu tiên là chữ số lẻ.
 - b. Đếm số phần tử là số chính phương.
 - c. Đếm số phần tử phân biệt, các phần tử trùng nhau chỉ đếm một lần.

Hướng dẫn:

//Câu 2a

```
int DemChuSoDauTienLe(int a[], int n) {  
    int ret = 0;  
    for(i = 0; i < n; i++)  
        if(ChuSoDauTienLe(a[i]))  
            ret++;  
    return ret;  
}
```

//Câu 2b

```
int DemChinhPhuong(int a[], int n) {  
    int ret = 0;  
    for(i = 0; i < n; i++)  
  
        if(ChinhPhuong(a[i]))  
            ret++;  
}
```

```

        return ret;

    }

//Câu 2c
int DemPhanBiet(int a[], int n) {
    int ret = 0;
    for(i = 0; i < n; i++) {
        if(TimKiem(a, i, a[i]) == -1) /*Không tìm thấy a[i] trong
            mảng a gồm i phần tử */
            ret++;
    }
    return ret;
}

```

3. Viết chương trình thực hiện các chức năng sau:
- Kiểm tra mảng có chứa số dương hay không?
 - Kiểm tra mảng có đối xứng hay không?
 - Kiểm tra mảng có tất cả các cặp phần tử đứng cạnh nhau có giá trị khác nhau hay không?

Hướng dẫn:

```

//Câu 3a
int CoDuong(int a[], int n) {
    int ret = 0; //Mảng không có số dương
    int i;
    for(i = 0; i < n - 1; i++)
        if(a[i] > 0) {
            ret = 1; // Mảng có số dương
            break;
        }
    return ret;
}

//Câu 3b
int DoiXung(int a[], int n) {
    int ret = 1; //Mảng đối xứng
    int i;
    for(i = 0; i < n/2; i++)
        if(a[i] != a[n - 1 - i])
            { //Mảng không đối xứng
                ret = 0;
                break;
            }
}

```

```

        return ret;
    }
//Câu 3c
int CapKhacNhau(int a[], int n) {
    int ret = 1;           //Mảng có cặp khác nhau
    int i;
    for(i = 0; i < n - 1; i++)
        if(a[i] == a[i + 1]) {
            ret = 0;       //Mảng có cặp không khác nhau
            break;
        }
    return ret;
}

```

4. Viết chương trình thực hiện các chức năng sau:
- Sắp xếp các phần tử là số chính phương tăng dần, các phần tử khác giữ nguyên vị trí.
 - Sắp xếp nửa mảng đầu tăng dần, nửa mảng sau giảm dần.

Hướng dẫn:

```

//Câu 4a
void SXCPTang(int a[], int n) {
    int i, j;
    for(i = 0; i < n-1; i++)
        for(j = i+1; j < n; j++)
            if(ChinhPhuong(a[i]) && ChinhPhuong(a[j]) && a[i] > a[j]) {
                int tam = a[i];
                a[i] = a[j];
                a[j] = tam;
            }
}

```

```

//Câu 4b
void SXNuaTangNuaGiam(int a[], int n) {
    SapXepDayConTang(a, n, 0, n/2);
    SapXepDayConGiam(a[], n, n/2 + 1, n - 1)
}

```

5. Viết chương trình thực hiện các chức năng sau:
- Tìm vị trí phần tử nguyên tố nhỏ nhất.
 - Tìm tất cả các cặp phần tử có tổng là số chính phương.

Hướng dẫn:

//Câu 6a

```

int TimVTNTMin(int a[], int n) {
    int i, j, ret;
    //Tìm vị trí phần tử nguyên tố đầu tiên
    for(i = 0; i < n; i++)
        if(NguyenTo(a[i])) break;
    if(i >= n) //Không có phần tử nguyên tố
        ret = -1;
    else { //Có phần tử nguyên tố
        //Tìm vị trí nguyên tố min
        ret = i;
        for(j = i + 1; j < n; j++)
            if(NguyenTo(a[j]) && a[j] < a[ret])
                ret = j;
    }
    return ret;
}

```

//Câu 6b

```

void TimCapCoTongLaCP(int a[], int n) {

    int qh = 0;
    int i, j;
    for(i = 0; i < n - 1; i++)
        for(j = i + 1; j < n; j++)
            if(ChinhPhuong(a[i] + a[j])) {
                printf("(%d, %d)\n", a[i], a[j]);
                qh = 1;
            }
    if(qh == 0)
        printf("Khong tim thay cap phan tu nao co quan he\n");
}

```

6. Viết chương trình thêm một giá trị x vào trong mảng có thứ tự tăng sao cho vẫn giữ nguyên tính tăng của mảng.

Hướng dẫn:

```

void ThemVaoMangTang(int a[], int
    *n, int x) { int i;
    //Tìm vị trí phần tử đầu tiên lớn hơn
    hoặc bằng x for(i = 0; i < n; i++)
        if(a[i] >= x) break;

    //Thêm x
    vào vị trí i
    Them(a,
    &(*n), i, x);
}

```

7. Viết các hàm thực hiện các chức năng sau:

- a. Xóa phần tử là số chính phương đầu tiên.
- b. Xóa tất cả phần tử là số chính phương.

Hướng dẫn:

//Câu 7a

```
int TimCPDauTien(int a[], int n) {
    int ret = -1;
    int i;
    for(i = 0; i < n; i++)
        if(ChinhPhuong(a[i])) {
            ret = i; break;
        }
    return ret;
}
```

```
void XoaCPDauTien(int a[], int *n) {
```

```
    int vt = TimCPDauTien(a, *n);
    if(vt != -1) {
        Xoa(a, &(*n), vt);
    }
    else <Không tìm thấy số CP>
```

```
}
```

//Câu 7b

```
void XoaTatCaCP(int a[], int *n) {
    int vt;
    while((vt = TimCP(a, *n)) != -1) {
        Xoa(a, &(*n), vt);
    }
}
```

8. Viết các hàm thực hiện các chức năng sau:
 - a. Tạo mảng b từ mảng a sao cho mảng b chỉ chứa các phần tử phân biệt.
 - b. Tạo mảng b từ mảng a sao cho mảng b chỉ chứa các phần tử là số chính phương.

Hướng dẫn:

//Câu 8a

```
void TaoMangChuaPhanBiet(int a[], int n, int b[], int *m) { *m = 0;
    for(int i = n; i > 0; i--)
        if(TimKiem(a, i, a[i]) == -1) { //Không tìm thấy
            b[*m] = a[i];
            (*m)++;
        }
}
```

//Câu 8b

```
void TaoMangChuaCP(int a[], int n, int b[], int *m) {
    *m = 0;
    for(int i = 0; i < n; i++)
```

```

        if(NguyenTo(a[i])) {
            b[*m] = a[i];
            (*m)++;
        }
    }
}

```

BÀI TẬP

9. Viết các hàm thực hiện các chức năng sau:
 - a. Tính Tổng các phần tử có chữ số tận cùng là 6 và chia hết cho 6.
 - b. Tính tổng các phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng.
10. Viết các hàm thực hiện các chức năng sau:
 - a. Đếm số phần tử có chữ số tận cùng là 6 và chia hết cho 6.
 - b. Đếm số phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng.
 - c. Đếm số phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng phân biệt, các phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng trùng nhau chỉ đếm một lần.
 - d. Đếm tần suất xuất hiện của các phần tử. Chẳng hạn với mảng gồm các phần tử: 12 34 12 34 43 12 5 thì tần suất xuất hiện các phần tử là 12 (3 lần), 34 (2 lần), 12 (2 lần), 5 (1 lần).
11. Viết các hàm thực hiện các chức năng sau:
 - a. Kiểm tra mảng có gồm toàn số dương hay không?
 - b. Kiểm tra mảng có thứ tự giảm hay không?
 - c. Kiểm tra mảng có được sắp thứ tự hay không?
 - d. Kiểm tra mảng có đan xen âm dương hay không?
 - e. Kiểm tra mảng có tất cả bộ 3 phần tử đứng cạnh nhau lập thành cấp số cộng hay không?
12. Viết các hàm thực hiện các chức năng sau:
 - a. Sắp xếp các phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng tăng dần, các phần tử khác giữ nguyên vị trí.
 - b. Trộn hai mảng tăng dần thành một mảng được sắp thứ tự tăng dần.
13. Viết hàm tìm vị trí của phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng nhỏ nhất.
14. Viết các hàm thực hiện các chức năng sau:

Xóa phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng cuối cùng.

Xóa tất cả phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng.
15. Viết các hàm thực hiện các chức năng sau:
 - a. Tạo mảng b từ mảng a sao cho mảng b chỉ chứa các phần tử là số hoàn chỉnh/số nguyên tố/ số đối xứng.
 - b. Tạo mảng b từ mảng a sao cho mảng b chỉ chứa các phần tử là số hoàn chỉnh/số nguyên tố/ số đối xứng phân biệt.

2. MẢNG HAI CHIỀU:

2.1. Khái niệm :

Mảng hai chiều thực chất là mảng một chiều trong đó mỗi phần tử của mảng lại là mảng một chiều, và được truy xuất bởi hai chỉ số dòng và cột.

2.2. Khai báo mảng hai chiều:

Cú pháp:

<Kiểu> <tên mảng>[<Kích thước dòng>][<Kích thước cột>];

Mỗi phần tử của mảng được truy nhập thông qua tên mảng cùng với hai chỉ số: chỉ số dòng (bắt đầu từ 0 đến <Kích thước dòng> – 1) và chỉ số cột (bắt đầu từ 0 đến <Kích thước cột> – 1).

Ví dụ: `int a[2][3]`

Khai báo mảng hai chiều gồm 6 phần tử kiểu `int`, bao gồm:

`a[0][0]` `a[0][1]` `a[0][2]`

`a[1][0]` `a[1][1]` `a[1][2]`

2.3. Khởi tạo mảng hai chiều:

Ví dụ 1:

```
int a[2][3] = {
    { 11, 12, 13 },
    { 21, 22, 23 } };
```

Ví dụ 2: Khởi tạo có thể không cần chỉ ra kích thước dòng

```
double b[][3] = {
    { 6, 7.8, 8 },
    { 3, 12.6, 4 },
    { 6.5, 20, 7 } };
```

Ví dụ 3: Sử dụng mảng hai chiều khởi tạo để vẽ chữ H bằng các dấu “*”

```
#define SIZE1 5
```

```
#define SIZE2 5
```

```
void main() {
```

```
    int a[SIZE1][SIZE2] = {
        { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }, { 1, 1, 1, 1, 1 },
        { 1, 0, 0, 0, 1 }, { 1, 0, 0, 0, 1 }
    };
```

```
    for(i = 0; i < SIZE1; i++) {
        for(j = 0; j < SIZE2; j++) {
            if(a[i][j] == 1) printf("*")
            else printf(" "); //Khoảng trắng
        }
        printf("\n");
    }
```

2.4. Nhập xuất mảng hai chiều:

Ví dụ:

```
#define SIZE1 4
```

```
#define SIZE2 6
```

```
void main() {
```

```
    float a[SIZE1][SIZE2]; /*Khai báo mảng hai chiều gồm SIZE1*SIZE2 phần tử
    kiểu thực*/
```

```
    int sd, sc; //Số dòng và số cột
```

```
    int i, j;
```

```
    float tam;
```

```
    do {
```

```
        printf("Nhap so dong va so cot:");
```

```
        scanf("%d%d", &sd, &sc);
```

```
    } while(sd < 1 || sd > SIZE1 || sc < 1 || sc > SIZE2);
```

```
    //Nhập dữ liệu mảng từ bàn phím
```

```
    for(i = 0; i < sd; i++)
```

```
        for(j = 0; j < sc; j++) {
```

```
            printf("pt thu [%d][%d]:", i, j);
```

```
            scanf("%f", &tam);
```

```
            a[i][j] = tam;
```

```

    }
//Xuất dữ liệu mảng ra màn hình
    for(i = 0; i < sd; i++) {
        for(j = 0; j < sc; j++)
            printf("%.2f\t", a[i][j]);
        printf("\n");
    }
}

```

Lưu ý: Trong C, đối với các phần tử mảng hai chiều không nguyên ta không thể sử dụng toán tử lấy địa chỉ.

2.5. Dùng mảng hai chiều làm tham số truyền cho hàm:

Tên mảng hai chiều cũng là một hằng địa chỉ và nó chính là địa chỉ phần tử đầu tiên của mảng. Khi dùng tên mảng làm tham số thực truyền cho hàm thì thực chất là địa chỉ phần tử đầu tiên của mảng được truyền cho hàm và như vậy tham số hình thức tương ứng trong định nghĩa hàm phải viết dưới dạng con trỏ.

Ví dụ:

```

#define SIZE1 4
#define SIZE2 6
// khai báo các nguyên mẫu hàm
void Nhap(float (*a)[SIZE2], int *sd, int *sc);
void Xuat(float (*a)[SIZE2], int sd, int sc);
// Định nghĩa các hàm
void Nhap(float (*a)[SIZE2], int *sd, int *sc)
{
    int i, j; float tam;
    do {
        printf("Nhap so dong va so cot:");
        scanf("%d%d", &(*sd), &(*sc));
    } while(*sd < 1 || *sd > SIZE1 || *sc < 1 || *sc > SIZE2);
    for(i = 0; i < *sd; i++)
        for(j = 0; j < *sc; j++) {
            printf("pt thu [%d][%d]:", i, j);
            scanf("%f", &tam);
            a[i][j] = tam;
        }
}

void Xuat(float (*a)[SIZE2], int sd, int sc) {
    int i, j;
    for(i = 0; i < sd; i++) {
        for(j = 0; j < sc; j++)
            printf("%.2f\t", a[i][j]);
        printf("\n");
    }
}

void main() {
    float a[SIZE1][SIZE2];
    int sd, sc;
    Nhap(a, &sd, &sc); Xuat(a, sd, sc);
}

```

Lưu ý: Tham số hình thức tương ứng với tham số thực là tên mảng hai chiều cũng có thể viết như sau:

```
void Nhap(float a[][SIZE2], int *sd, int *sc);
```

```
void Xuat(float a[][SIZE2], int sd, int sc);
```

2.6. Các bài toán cơ bản trên mảng hai chiều:

- Duyệt theo từng dòng từ trên xuống dưới, với mỗi dòng duyệt từ trái sang phải

```
for(i = 0; i < sd; i++)
    for(j = 0; j < sc; j++) {
        //Xử lý a[i][j]
        ...
    }
```

- Duyệt theo từng cột từ trái sang phải, với mỗi cột duyệt từ trên xuống dưới

```
for(j = 0; j < sc; j++)
    for(i = 0; i < sd; i++) {
        //Xử lý a[i][j]
        ...
    }
```

- Duyệt các phần tử nằm trên cùng một dòng hay trên cùng một cột

Duyệt các phần tử trên dòng có chỉ số k ($0 \leq k < sd$)

```
for(i = 0; i < sc; i++) {
    <Xử lý a[k][i]> }
```

Duyệt các phần tử trên cột có chỉ số k ($0 \leq k < sc$)

```
for(i = 0; i < sd; i++) {
    <Xử lý a[i][k]> }
```

2.7. Mảng vuông (ma trận vuông):

a. Tính chất

Mảng vuông là mảng hai chiều có số dòng = số cột = n, n gọi là cấp ma trận. Sau đây là một số đặc tính của ma trận vuông:

Các phần tử nằm trên đường chéo chính là các phần tử $a[i][i]$ ($0 \leq i < n$)

Các phần tử nằm trên đường chéo phụ là các phần tử $a[i][n - 1 - i]$ ($0 \leq i < n$)

Các phần tử nằm trong nửa mảng vuông phía trên đường chéo chính là các phần tử $a[i][j]$ với $i \leq j$

Các phần tử nằm trong nửa mảng vuông phía dưới đường chéo chính là các phần tử $a[i][j]$ với $i \geq j$

Các phần tử nằm trong nửa mảng vuông phía trên đường chéo phụ là các phần tử $a[i][j]$ với $i + j \leq n - 1$

Các phần tử nằm trong nửa mảng vuông phía dưới đường chéo phụ là các phần tử $a[i][j]$ với $i + j \geq n - 1$

b. Duyệt mảng vuông:

Duyệt các phần tử nằm trên ĐCC

```
for(i = 0; i < n; i++)
    <Xử lý a[i][i]>
```

Duyệt các phần tử nằm trên ĐCP

```
for(i = 0; i < n; i++)
    <Xử lý a[i][n - 1 - i]>
```

Duyệt các phần tử nằm trong nửa mảng vuông.

Phía trên ĐCC

```
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        if(i <= j) {
            <Xử lý a[i][j]> }
```

Phía dưới ĐCC

```
for(i = 0; i < n; i++)
    for(j = 0; j < n; j++)
        if(i >= j) {
```

<Xử lý a[i][j]> }

Phía trên ĐCP

```
for(i = 0; i < n; i++)  
    for(j = 0; j < n; j++)  
        if(i + j <= n - 1)  
            <Xử lý a[i][j]>
```

Phía dưới ĐCP

```
for(i = 0; i < n; i++)  
    for(j = 0; j < n; j++)  
        if(i + j >= n - 1)  
            <Xử lý a[i][j]>
```

2.8. Bài tập:

Đối với các bài tập trong chương này, trong chương trình luôn phải có hai hàm nhập và xuất mảng hai chiều nguyên.

□ Bài tập có hướng dẫn

1. Viết các hàm thực hiện các chức năng sau:

- Tính tổng các phần tử trên dòng thứ k.
- Tính trung bình cộng các phần tử là số chính phương.

Hướng dẫn:

//Câu 1a

```
int Tong(int a[][SIZE2], int sc, int k) {  
    int ret = 0;  
    for(i = 0; i < sc;  
        i++)  
        ret += a[k][i];  
    return ret;  
}
```

//Câu 1b

```
float TBCong(int a[][SIZE2], int sd, int sc) {  
    int dem = 0, s = 0;  
    float ret;  
    for(int i = 0; i < sd; i++)  
        for(int j = 0; j < sc; j++)  
            if(chinhPhuong(a[i][j])) {  
                s += a[i][j];  
                dem++;  
            }  
  
    if(dem > 0)  
        ret = (float)s / dem;  
    else ret = -1; //Ma trận không chứa số chính phương  
    return ret;  
}
```

2. Viết các hàm thực hiện các chức năng sau:

- Đếm số phần tử là số chính phương.
- Đếm số phần tử là số chính phương trên dòng thứ k.

Hướng dẫn

//Câu 2a

```
int DemCP (int a[][SIZE2], int sd, int sc) {
```

```

int ret = 0;
int i, j;
for(i = 0; i < sd; i++)
    for(j = 0; j < sc; j++) {
        if(chinhPhuong(a[i][j]))
            ret++;
    }
return ret;
}

```

//Câu 2b

```

int DemCPTrenDong(int a[][SIZE2], int sc, int k) {
    int ret = 0;
    for(int i = 0; i < sc; i++) {
        if(chinhPhuong(a[k][i]))
            ret++;
    }
    return ret;
}

```

3. Viết các hàm thực hiện các chức năng sau:
 - a. Kiểm tra ma trận có phần tử dương hay không.
 - b. Kiểm tra dòng thứ k của ma trận có tăng dần/giảm dần hay không.

Hướng dẫn:

//Câu 3a

```

int CoDương(int a[][SIZE2], int sd, int sc) {
    int i, j;
    for(i = 0; i < sd - 1; i++)
        for(j = 0; j < sc - 1; j++) {
            if(a[i][j] > 0) {
                return 1; //Ma trận có phần tử dương
            }
        }
    return 0; //Ma trận không có phần tử dương
}

```

//Câu 3b

```

int DongTang(int a[][SIZE2], int sc, int k) {
    int ret = 1; //Ma trận có dòng thứ k tăng
    for(int i = 0; i < sc - 1; i++)
        if(a[k][i] > a[k][i+1]) {
            ret = 0; //Ma trận có dòng thứ k không tăng
            break;
        }
    return ret;
}

```

4. Viết các hàm thực hiện các chức năng sau:

- a. Liệt kê các dòng có chứa ít nhất một phần tử là số chính phương.
- b. Liệt kê các dòng có chứa phần tử gồm toàn số chính phương.

Hướng dẫn:

//Câu 4a

```
void LietKeDongCoCP(int a[][SIZE2], int sd, int sc) {
    int cp; //Cờ
    int i, j;
    for(i = 0; i < sd; i++) {
        //Kiểm tra dòng thứ i có chứa số chính phương không?
        cp = 0;
        for(j = 0; j < sc; j++) {
            if(ChinhPhuong(a[i][j])) {
                cp = 1;
                break;
            }
        }

        //In dòng có số chính phương
        if(cp) {
            for(j = 0; j < sc; j++)
                printf("%d\t", a[i][j]);
            printf("\n");
        }
    }
}
```

//Câu 4b

```
void LietKeDongToanCP(int a[][SIZE2], int sd, int sc) {
    int cp; //Cờ
    int i, j;
    for(i = 0; i < sd; i++) {
        /*Kiểm tra dòng thứ i có chứa các phần tử gồm toàn chính
        phương không? */
        cp = 1;
        for(j = 0; j < sc; j++) {
            if(ChinhPhuong(a[i][j]) == 0) {
                cp = 0;
                break;
            }
        }

        //In dòng có số chính phương
        if(cp) {
            for(j = 0; j < sc; j++)
                printf("%d\t", a[i][j]);
            printf("\n");
        }
    }
}
```

```

    }
}

```

5. Viết các hàm thực hiện các chức năng sau:
- Tìm vị trí phần tử lớn nhất.
 - Tìm vị trí phần tử lớn nhất dòng thứ k.
 - Tìm vị trí phần tử dương nhỏ nhất dòng thứ k.

Hướng dẫn:

//Câu 5a

```

void TimVTMax(int a[][SIZE2], int sd, int sc, int
    *csdong, int *cscot) { int i, j;
    *csdong = 0; *cscot = 0;
    for(i = 0; i < sd; i++)
        for(j = 0; j < sc; j++)
            if(a[i][j] > a[*csdong][*cscot]) {
                *csdong = i;
                *cscot = j;
            }
}

```

//Câu 5b

```

int TimVTMaxTrenDong(int a[][SIZE2], int sc, int k,
    int *csdong, int *cscot) {
    int i, j;
    *csdong = k;
    *cscot = 0;
    for(i = 1; i < sc; i++) {
        if(a[k][i] > a[k][*cscot])
            *cscot = i;
    }
    return ret; //Phần tử max nằm ở dòng thứ ret (cột thứ k)
}

```

6. Viết các hàm thực hiện các chức năng sau:
- Sắp xếp các phần tử trên dòng thứ k tăng dần từ trái sang phải.
 - Sắp xếp các cột của ma trận sao cho tổng giá trị của mỗi cột tăng dần từ trái sang phải.

Ma trận nhập vào

| | | | |
|---|---|---|---|
| 2 | 4 | 6 | 0 |
| 9 | 1 | 2 | 6 |

11 5 8 6

Ma trận sau khi s.xếp

| | | | |
|---|---|---|---|
| 4 | 0 | 6 | 2 |
| 1 | 6 | 2 | 9 |

5 6 8 11

- Sắp xếp ma trận tăng dần trên mỗi dòng từ trái sang phải và từ trên xuống dưới.

Ma trận nhập vào

| | | | |
|---|---|---|---|
| 2 | 4 | 5 | 9 |
| 9 | 7 | 1 | 8 |
| 5 | 9 | 0 | 1 |
| 3 | 8 | 7 | 6 |

Ma trận sau khi s.xếp

| | | | |
|---|---|---|---|
| 0 | 1 | 1 | 2 |
| 3 | 4 | 5 | 5 |
| 6 | 7 | 7 | 8 |
| 8 | 9 | 9 | 9 |

Hướng dẫn:

//Câu 6a

```
void SapXepDong(int a[][SIZE2], int sc, int k) {  
    int i, j;
```

```
    for(i = 0; i < sc-1; i++)
```

```
        for(j = i+1; j < sc; j++)
```

```
            if(a[k][i] > a[k][j]) {
```

```
                int tam = a[k][i];
```

```
                a[k][i] = a[k][j];
```

```
                a[k][j] = tam;
```

```
            }
```

```
    }
```

//Câu 6b

```
void HoanViCot(int a[][SIZE2], int sd, int i, int j) {
```

```
    for(int k = 0; k < sd; k++) {
```

```
        int tam = a[k][i];
```

```
        a[k][i] = a[k][j];
```

```
        a[k][j] = tam;
```

```
    }
```

```
}
```

```
void SXTongCotTang(int a[][SIZE2], int sd, int sc) {
```

```
    for(int i = 0; i < sc-1; i++)
```

```
        for(int j = i+1; j < sc; j++) {
```

```
            int s1 = TongCot(a, sd, i);
```

//Tổng các phần tử trên cột i

```
            int s2 = TongCot(a, sd, j);
```

//Tổng các phần tử trên cột j

```
            if(s1 > s2)
```

```
                HoanViCot(a, sd, i, j);
```

//Hoán vị cột i và cột j

```
        }
```

```
    }
```

//Câu 6c

```
void SapXepCau6d(int a[][SIZE2], int sd, int sc) {
```

```
    int b[SIZE1*SIZE2];
```

```

int m = 0;
//Xây dựng mảng một chiều b từ mảng hai chiều a cho trước
for(i = 0; i < sd; i++)
    for(j = 0; j < sc; j++) {
        b[m] = a[i][j]; m++;
    }
//Sắp xếp mảng một chiều b tăng dần
SapXepTang(b, m);
//Đặt các phần tử trong b ngược trở về a
m = 0;

for(i = 0; i < sd; i++)
    for(j = 0; j < sc; j++) {
        a[i][j] = b[m]; m++;
    }
}

```

□ Bài tập luyện tập

7. Viết các hàm thực hiện các chức năng sau:
 - a. Tính tổng/tích các phần tử trên cột thứ k.
 - b. Tính trung bình cộng các phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng.
8. Viết các hàm thực hiện các chức năng sau:
 - a. Đếm số phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng.
 - b. Đếm số phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng trên cột thứ k.
9. Viết các hàm thực hiện các chức năng sau:
 - a. Kiểm tra ma trận có gồm toàn phần tử dương hay không.
 - b. Kiểm tra cột thứ k của ma trận có tăng dần/giảm dần hay không.
10. Viết các hàm thực hiện các chức năng sau:
 - a. Liệt kê các cột có chứa phần tử là số hoàn chỉnh/số nguyên tố/số đối xứng.
 - b. Liệt kê các cột có chứa phần tử gồm toàn là số chính phương/số hoàn chỉnh/số nguyên tố/số đối xứng.
11. Viết các hàm thực hiện các chức năng sau:
 - a. Tìm vị trí phần tử nhỏ nhất.
 - b. Tìm vị trí phần tử nhỏ nhất trên cột thứ k.
 - c. Tìm vị trí phần tử âm lớn nhất trên cột thứ k.
12. Viết các hàm thực hiện các chức năng sau:
 - a. Sắp xếp các phần tử trên cột thứ k tăng dần từ trên xuống dưới.
 - b. Sắp xếp các dòng của ma trận sao cho tổng giá trị của mỗi dòng tăng dần từ trên xuống dưới.
13. Viết các hàm thực hiện các chức năng sau trên mảng vuông
 - a. Đếm các phần tử là số chính phương/số hoàn chỉnh/số nguyên tố/số đối xứng nằm trên đường chéo phụ.
 - b. Đếm các phần tử là số chính phương/số hoàn chỉnh/số nguyên tố/số đối xứng nằm trong nửa mảng vuông phía dưới đường chéo chính/nửa mảng vuông phía trên đường chéo phụ/ nửa mảng vuông phía dưới đường chéo phụ.
 - c. Tìm phần tử lớn nhất/nhỏ nhất trên đường chéo phụ.
 - d. Tìm phần tử âm lớn nhất/dương nhỏ nhất trên đường chéo phụ.

e. Sắp xếp các phần tử trên đường chéo phụ theo thứ tự tăng/giảm dần từ trên hướng xuống/từ dưới hướng lên.

3. CHUỖI:

3.1/ Khái niệm:

Chuỗi được xem như một mảng 1 chiều gồm các phần tử kiểu char. Ngoài ra ký hiệu kết thúc chuỗi được quy ước là ký tự ‘\0’ (ký tự có mã ASCII là 0) đặt ở cuối chuỗi. Như vậy một mảng ký tự gồm n phần tử sẽ lưu được tối đa n – 1 ký tự.

3.2/ Khai báo chuỗi:

Cú pháp:

```
char <Tên chuỗi>[<Kích thước>];
```

Ví dụ:

```
char s[10];
```

Khai báo một chuỗi s có 10 phần tử kiểu char, nhưng s chỉ chứa tối đa được 9 ký tự.

3.3/ Khởi tạo chuỗi:

Khi khởi tạo chuỗi có thể chỉ ra hoặc không chỉ ra kích thước chuỗi. Ngoài ra cũng có thể dùng một biến con trỏ kiểu char để khởi tạo một chuỗi.

Ví dụ 1: Chỉ ra kích thước khi khởi tạo chuỗi: `char s[10] = "Hello";`

Ví dụ 2: Không chỉ ra kích thước khi khởi tạo chuỗi: `char s[] = "Hello";`

Ví dụ 3: Dùng biến con trỏ kiểu char để khởi tạo chuỗi: `char *p = "Hello";`

3.4/ Hàm nhập chuỗi:

a. Khái niệm về stdin:

stdin là một vùng nhớ đệm dùng để lưu dữ liệu từ bàn phím. Các hàm scanf, gets đều nhận dữ liệu từ stdin. Cần phân biệt hai trường hợp:

Nếu trên stdin có đủ dữ liệu thì các hàm trên sẽ nhận một phần dữ liệu mà chúng yêu cầu. Phần dữ liệu còn lại (chưa được nhận) vẫn ở trên stdin.

Nếu trên stdin không đủ dữ liệu theo yêu cầu của các hàm, thì máy tạm dừng để người dùng đưa thêm dữ liệu từ bàn phím lên stdin (cho đến khi nhấn phím Enter)

b. Hàm nhập chuỗi:

```
char *gets(char *s);
```

Nhập một dãy ký tự từ stdin cho đến khi gặp ‘\n’ (ký tự ‘\n’ bị loại khỏi stdin). Dãy ký tự được bổ sung thêm ký tự ‘\0’ và đặt vào vùng nhớ do s trỏ đến. Hàm trả về con trỏ chỉ đến chuỗi nhận được.

Chú ý: Hàm scanf có để lại ký tự ‘\n’ trên stdin, ký tự này sẽ làm trôi hàm gets (nếu có) sau đó. Để hàm gets không bị trôi thì phải khử ký tự ‘\n’ trong stdin trước khi gọi hàm gets bằng cách dùng hàm fflush(stdin) để làm sạch stdin.

Ví dụ:

```
void main() {  
    int tuoi;  
    char ten[31];
```

```

printf("Nhap tuoi:");
scanf("%d", &tuoi);
fflush(stdin); //Làm sạch stdin
printf("Nhap ten:");
gets(ten);
}

```

5/ Một số hàm thao tác chuỗi:

Các hàm sau đây được định nghĩa sẵn trong thư viện string.h

int strlen(const char *s);

Trả về độ dài chuỗi s.

char *strcat(char *s1, const char *s2);

Ghép s2 vào s1 và trả về con trỏ đến s1.

char *strcpy(char *s1, const char *s2);

Chép s2 đè lên s1 và trả về con trỏ đến s1.

int strcmp(const char *s1, const char *s2);

So sánh s1 và s2. Hàm trả về giá trị âm nếu s1 nhỏ hơn s2, giá trị 0 nếu s1 bằng s2 và giá trị dương nếu s1 lớn hơn s2.

int stricmp(const char *s1, const char *s2);

Hàm làm việc tương tự như strcmp nhưng không phân biệt chữ hoa với chữ thường.

char *strchr(const char *s, int c);

Tìm sự xuất hiện đầu tiên của c trong s. Nếu tìm thấy hàm trả về con trỏ đến ký tự tìm được trong s, nếu không hàm trả về giá trị NULL.

char *strstr(const char *s1, const char *s2);

Tìm sự xuất hiện đầu tiên của s2 trong s1. Nếu tìm thấy hàm trả về con trỏ đến chuỗi con tìm được trong s1, nếu không hàm trả về giá trị NULL.

6/ Bài tập:

☐ Bài tập có hướng dẫn

- Viết chương trình kiểm tra một chuỗi có đối xứng hay không?

Ví dụ: Các chuỗi level, radar, dad, ... là các chuỗi đối xứng

Hướng dẫn

```

int DoiXung(char *s) {
    int len = strlen(s); int ret = 1;
    for(int i = 0; i <= len/2; i++)
        if(s[i] != s[len - 1 - i]) {
            ret = 0; break;
        }
}

```

```

return ret;
}

```

2. Viết chương trình tìm chuỗi đảo ngược của một chuỗi.

Ví dụ: Chuỗi “Lap trinh C” có chuỗi đảo ngược là “C hnirt paL”

Hướng dẫn

//Dữ liệu đầu vào là s1, dữ liệu đầu ra là s2 đảo ngược

```
void TimDaoNguoc(char *s1, char *s2) {
```

```

    int len =
    strlen(s1);
    int j = 0;

    for(int i = len - 1; i >=
        0; i--) { s2[j] =
        s1[i];
        j++;
    }

```

```

    s2[j] = 0; //Thêm ký tự ‘\0’ và cuối chuỗi s2
}

```

3. Viết chương trình đếm số từ trong một chuỗi. Qui ước các từ trong chuỗi cách nhau ít nhất một khoảng trắng.

Ví dụ: Chuỗi “Lap trinh C” có 3 từ

Hướng dẫn

```

int DemTu(char *s) {
    int len = strlen(s);
    int ret = 0;
    int i = 0;
    while(i < len) {
        //Đi qua các ký tự trắng
        while(s[i] == ' ' && i < len)
            i++;
        if(i < len) {
            //Đi qua các ký tự khác trắng
            while(s[i] != ' ' && i < len)
                i++;
            ret++;
        }
    }
}

```

```

    return ret;
}

```

4. Viết chương trình in mỗi từ của chuỗi trên một dòng.

Ví dụ: Chuỗi “Lập trình C” sẽ được in ra:

Lập
trình
C

```

void InTu(char *s) {
    int i = 0, j;
    char tu[20];
    int len = strlen(s);
    while(i < len) {
        //Đi qua các khoảng trắng
        while(s[i] == ' ' && i < len)
            i++;
        if(i < len) {
            //Đi qua các ký tự khác trắng
            j = 0;
            while(s[i] != ' ' && i < len) {
                tu[j] = s[i];
                j++; i++;
            }
            tu[j] = 0;
            printf("%s\n", tu);
        }
    }
}

```

5. Viết chương trình đếm số lần xuất hiện của mỗi ký tự từ A đến Z có trong một chuỗi (không phân biệt chữ hoa hay chữ thường).

Ví dụ: Với chuỗi “Hello World” ta sẽ có:

D: 1 lần
E: 1 lần
H: 1 lần
L: 3 lần
...

```

#include <ctype.h>
#define NUM_ALPHABET 26          //Số ký tự trong bảng chữ cái
void ThongKe(char *s) {
    int dem[NUM_ALPHABET];       //Mảng đếm số lần xuất hiện ký tự
    //Khởi tạo mảng đếm
    for(i = 0; i < NUM_ALPHABET; i++)
        dem[i] = 0;

    //Đếm số lần xuất hiện ký tự
    int len = strlen(s);
    for(i = 0; i < len; i++)
        if(toupper(s[i]) >= 'A' && toupper(s[i]) <= 'Z')
            dem[toupper(s[i]) - 'A']++;

    //In kết quả
    for(i = 0; i < NUM_ALPHABET; i++)
        printf("ký tự %c: %d lan\n", i + 'A', dem[i]);
}

```

6. Viết một hàm stringcpy có chức năng tương tự như hàm **strcpy** của thư viện.

Hướng dẫn

```

void stringcpy(char *s1, char *s2) {
    int i = 0;
    while((s1[i] = s2[i]) != 0)
        i++;
}

```

□ **Bài Tập luyện tập**

7. Viết chương trình xóa đi ký tự 'e' có trong một chuỗi nhập từ bàn phím.

Ví dụ:

Chuỗi ban đầu = "Que huong la chum khe ngọt"

Chuỗi sau khi xóa = "Qu huong la chum kh ngọt"

8. Viết chương trình xóa từ đầu tiên có trong một chuỗi nhập từ bàn phím.

Ví dụ:

Chuỗi ban đầu = "Que huong la chum khe ngọt"

Chuỗi sau khi xóa = "huong la chum khe ngọt"

9. Viết một hàm stringlen có chức năng tương tự như hàm **strlen** của thư viện.
 10. Viết một hàm stringcmp có chức năng tương tự như hàm **strcmp** của thư viện.
 11. Viết một hàm stringcat có chức năng tương tự như hàm **strcat** của thư viện.

12. Viết một hàm stringchar có chức năng tương tự như hàm **strchar** của thư viện.
13. Viết một hàm stringstring có chức năng tương tự như hàm **strchar** của thư viện.



CHƯƠNG 8A: KIỂU DỮ LIỆU CẤU TRÚC

Mục đích:

Trong quá trình xây dựng ứng dụng, ngoài các kiểu dữ liệu do ngôn ngữ lập trình C hỗ trợ, thì C còn cho phép các lập trình viên có thể tự định nghĩa các kiểu dữ liệu riêng của mình. Các kiểu dữ liệu này được gọi là kiểu dữ liệu do người dùng định nghĩa “Kiểu dữ liệu người dùng” hay kiểu dữ liệu hướng người dùng.

Thực tế điều gì xảy ra nếu khi xây dựng chương trình cần định nghĩa một kiểu dữ liệu, nó tập dữ liệu mà mỗi thành phần có kiểu dữ liệu khác nhau ví dụ như SinhVien là một tập dữ liệu bao gồm : Tên, ngày sinh, giới tính và số điện thoại. Lúc đó các lập trình viên không thể sử dụng mảng để giải quyết vấn đề này được.

Vì vậy kiểu dữ liệu cấu trúc cho phép lập trình viên tự định nghĩa để giải quyết theo yêu cầu của bài toán dựa vào những kiểu dữ liệu cơ bản được cài đặt sẵn trong ngôn ngữ lập trình.

Khái niệm

Kiểu cấu trúc (struct) là một kiểu dữ liệu do người dùng định nghĩa bằng cách gom nhóm các kiểu dữ liệu cơ bản có sẵn trong C thành một kiểu dữ liệu phức hợp nhiều thành phần.

Định nghĩa kiểu dữ liệu

Cú pháp:

struct < tên cấu trúc >{

Các kiểu dữ liệu thành phần ;

};

Ví dụ 1: Kiểu dữ liệu SINHVIEN gồm các thành phần:

- Mã số sinh viên: chuỗi tối đa 10 ký tự
- Họ và tên: chuỗi tối đa 25 ký tự
- Ngày tháng năm sinh: chuỗi tối đa 15 ký tự
- Điểm trung bình: kiểu float

```
struct SINHVIEN{  
    char mssv[10];  
    char hoten[25];  
    char ngaysinh[15];  
    float dtb;  
};
```

Kiểu dữ liệu cấu trúc có thể lồng nhau:

Ví dụ 2: Định nghĩa kiểu dữ liệu của học sinh HOCSINH gồm:

- Mã số học sinh (MSHS): chuỗi có tối đa 5 ký tự.
- Họ tên (hoten): chuỗi có tối đa 30 ký tự.
- Ngày tháng năm sinh (ngaysinh): kiểu DATE.
- Địa chỉ (diachi): chuỗi có tối đa 50 ký tự.
- Giới tính: chuỗi có tối đa 3 ký tự

- Điểm trung bình (diemtb): kiểu float

Ta định nghĩa kiểu HOCSINH như sau:

```
struct DATE
{
    char thu[5];
    unsigned char ngay;
    unsigned char thang;
    int nam;
}

struct HOCSINH {
    char MSHS[6];
    char hoten[31];
    struct DATE ngaysinh;
    char diachi[51];
    unsigned char phai[4];
    float diemtb;
};
```

Khai báo

Khi ta định nghĩa kiểu dữ liệu tức là ta có một kiểu dữ liệu mới, muốn sử dụng ta phải khai báo biến. Cú pháp khai báo kiểu dữ liệu cũng giống như cách khai báo của các kiểu dữ liệu chuẩn.

Cú pháp: **struct < tên cấu trúc > < tên biến >** hoặc
< tên cấu trúc > < tên biến >

Ví dụ 3:

```
struct nhanvien
{
    int manv;
    char hoten[25];
};

struct nhanvien nv; hoặc nhanvien nv;
```

Có thể vừa tạo cấu trúc vừa khai báo:

```
struct nhanvien
{
    int manv;
    char hoten[25];
}nv;
```

Ngoài ra còn có thể khai báo theo kiểu con trỏ như sau:

struct < tên cấu trúc > * < tên biến > ;

Ví dụ 4:

```
DATE *y;           //Khai báo con trỏ kiểu cấu trúc DATE
y = (DATE *) malloc (sizeof (DATE)) ;
```

Truy xuất

Để truy xuất một thành phần dữ liệu nào đó bên trong cấu trúc ta có 2 trường hợp truy xuất như sau :

- Biến x là một biến cấu trúc thông thường, ta dùng toán tử dấu chấm “.”

Cú pháp :

< Tên cấu trúc >.< Biến thành phần >;

Ví dụ 5:

```
DATE x;           //khai bao bien x kieu DATE
x.ngay = 5; //gan ngay bang 5
```

- Biến x là một biến con trỏ, ta dùng toán tử mũi tên “->” (Gồm dấu trừ ‘-’ và dấu lớn hơn ‘>’).

Cú pháp :

< Tên cấu trúc > -> < Biến thành phần >;

Ví dụ 6:

```
DATE *x; //khai bao bien x kieu con tro DATE
x -> ngay = 5;           //gan ngay bang 5
```

Đối với kiểu dữ liệu có struct lồng nhau phải truy cập đến thành phần cuối cùng có kiểu dữ liệu cơ bản.

Ví dụ 7: Giả sử, có kiểu HOCSINH như trên

HOCSINH hs; //khai bao bien hs kieu HOCSINH

Muốn in học sinh A sinh vào tháng mấy ta phải truy cập như sau:

```
printf(“Thang sinh cua hoc sinh A la: %d”,(hs.ngaysinh).thang);
```

Ví dụ minh họa:

Viết chương trình nhập vào tọa độ hai điểm trong mặt phẳng và tính tổng hai tọa độ này.

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

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

```
struct DIEM //khai bao mot kieu du lieu DIEM gom toa do x va y
```

```
{
```

```
    int x;
```

```
    int y;
```

```
};
```

```
void Nhap (DIEM &d)
```

```
{
```

```

printf ("\nNhap vao tao do diem\n");
printf ("Tung do : ");
scanf ("%d", & d.x);
printf ("Hoanh do : ");
scanf ("%d", & d.y);
}
void Xuat (DIEM d)
{
    printf ("\nToa do diem : (%d, %d)", d.x, d.y);
}
DIEM Tong (DIEM d1, DIEM d2)
{
    DIEM temp;
    temp.x = d1.x + d2.x;
    temp.y = d1.y + d2.y;
    return Temp;
}
void main ()
{
    DIEM A, B, AB; //khai bao 3 diem A, B, AB;
    Nhap (A);
    Xuat (A);
    Nhap (B);
    Xuat (B);
    printf ("\n Tong cua hai diem vua nhap la : ");
    AB = Tong (A, B);
    Xuat (AB);
    getch ();
}

```

Mảng cấu trúc

- Cách khai báo tương tự như mảng một chiều hay ma trận (Kiểu dữ liệu bây giờ là kiểu dữ liệu có cấu trúc).
- Cách truy cập phần tử trong mảng cũng như truy cập trên mảng một chiều hay ma trận. Nhưng do từng phần tử có kiểu cấu trúc nên phải chỉ định rõ cần lấy thành phần nào, tức là phải truy cập đến thành phần cuối cùng có kiểu là dữ liệu cơ bản (xem lại bảng các kiểu dữ liệu cơ bản).

Nguyên tắc viết chương trình có mảng cấu trúc

Do kiểu dữ liệu có cấu trúc thường chứa rất nhiều thành phần nên khi viết chương trình loại này thường code khác dài vì vậy chúng ta cần phân tích kỹ lưỡng trước khi làm:

- Xây dựng hàm xử lý cho một kiểu cấu trúc.
- Để xử lý cho mảng cấu trúc, ta gọi lại hàm xử lý cho một kiểu cấu trúc đã được xây dựng bằng cách dùng vòng lặp.

Ví dụ 8: Cho một lớp học gồm n học sinh ($n < 50$). Thông tin của một học sinh được mô tả ở ví dụ 2. Hãy viết chương trình nhập và xuất danh sách học sinh sau đó đếm xem có bao nhiêu học sinh được lên lớp (Điều kiện được lên lớp là điểm trung bình > 5.0).

Phân tích:

- Trước hết ta phải xây dựng hàm nhập và xuất cho 1 học sinh.
- Xây dựng hàm nhập và xuất ngày tháng năm (Kiểu dữ liệu DATE).
- Sau đó mới xây dựng hàm nhập và xuất cho danh sách học sinh.
- Thực hiện hàm đếm số học sinh lên lớp.

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

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

```
#define MAX 50
```

```
struct DATE
```

```
{
```

```
    char thu[5];
```

```
    unsigned char ngay;
```

```
    unsigned char thang;
```

```
    int nam;
```

```
};
```

```
struct HOCSINH
```

```
{
```

```
    char MSHS[8];
```

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

```
    struct DATE ngaysinh;
```

```
    unsigned char phai[4];
```

```
    float diemtb;
```

```
}
```

```
voidNhapNamSinh(DATE &d);
```

```
voidXuatNamSinh(DATE d);
```

```
voidNhapIHS (HOCSINH &hs);
```

```
voidXuatIHS (HOCSINH hs);
```

```
voidNhapDSHS(HOCSINH dshs[], int &n);
```

```

void XuatDSHS(HOCSINH dshs[], int n);
int DemHSLenLop(HOCSINH dshs[], int n);
void main()
{
    HOCSINH dshs[MAX]; //Khai báo mảng dshs gồm có tối đa 50 học sinh
    int n, sohsdau;
    NhapDSHS(dshs, n);
    XuatDSHS(dshs, n);
    sohsdau = DemHSLenLop(dshs, n);
    printf("\nSố lượng học sinh được lên lớp là: %d", sohsdau);
    getch();
}
void NhapNamSinh(DATE &d)
{
    printf("\nNhập vào ngày: "); scanf("%u", &d.ngay);
    printf("\nNhập vào tháng: "); scanf("%u", &d.thang);
    printf("\nNhập vào năm: "); scanf("%d", &d.nam);
}
void XuatNamSinh(DATE d){
    printf("%02u / %02u / %4d", d.ngay, d.thang, d.nam);
}
void Nhap1HS(HOCSINH &hs)
{
    float d;
    fflush(); //Xóa vùng đệm
    printf("\nNhập mã số học sinh: ");
    gets(hs.MSHS);
    printf("\nNhập họ tên học sinh: ");
    gets(hs.hoten);
    printf("\nNhập ngày tháng năm sinh: ");
    fflush(); //Xóa vùng đệm
    NhapNamSinh (hs. ngay sinh);
    printf("\nPhái: ");
    gets(hs.phai);
    printf("\nNhập vào điểm trung bình: ");
    fflush(); //Xóa vùng đệm

```



```

scanf("%f", &d); //Nhập vào biến tạm d sau đó gán vào hs.diemtb hs.diemtb=d;
}

void NhapDSHS(HOCSINH dshs[], int &n)
{
    printf("\nNhập vào số lượng học sinh: ");
    scanf("%d", &n);
    for(int i=0; i<n; i++)
    {
        printf("\nNhập vào thông tin của học sinh thu %d:\n", i+1);
        Nhap1HS(dshs[i]); //Goi ham nhap thông tin 1 học sinh
    }
}

void Xuat1HS(HOCSINH hs)
{
    printf("\nMã số học sinh: %s", hs.MSHS);
    printf("\nHọ tên học sinh: %s", hs.hoten);
    printf("\nNgày tháng năm sinh: ");
    XuatNamSinh(hs.ngaysinh);
    printf("\nPhái: %s", hs.phai);
    printf("\nĐiểm trung bình: %2.2f", hs.diemtb);
}

void XuatDSHS(HOCSINH dshs[], int n)
{
    for(int i=0; i<n; i++)
    {
        printf("\n\nThông tin học sinh thu %d:", i+1);
        Xuat1HS(dshs[i]); //Goi ham xuat thông tin 1 học sinh
    }
}

int DemHSLenLop(HOCSINH dshs[], int n)
{
    int d=0;
    for(int i=0; i<n; i++)
        if(dshs[i].diemtb>=5.0)
            d++;
    return d;
}

```

- **Lưu ý:**

Khi nhập một chuỗi trong màn hình console, ta phải có thao tác xóa bộ nhớ đệm bàn phím. Nếu không có thể thấy rằng kết quả nhập chuỗi bị sai hoặc trôi đi mất. Trong quá trình chạy chương trình ta sẽ phải nhập bằng bàn phím, mọi ký tự bạn gõ vào bàn phím (kể cả ký tự Enter \n) đều được đẩy vào bộ nhớ đệm trước khi được gán vào biến. Nếu trước đó bạn có nhập số bằng **scanf** hoặc **cin**, chúng chỉ nhận số chứ không nhận được ký tự Enter, và ký tự Enter vẫn còn trong bộ nhớ đệm. Đến khi nhập chuỗi, các hàm nhập chuỗi nhận được ký tự Enter thì dừng

nhập luôn và chương trình vẫn chạy tiếp. Điều này khiến kết quả bị sai. Vì vậy ta thường sử dụng hàm `flushall()` dùng để xóa bộ đệm

BÀI TẬP

1. Viết chương trình sử dụng kiểu cấu trúc để hiển thị giờ, phút, giây ra màn hình, và tính khoảng cách giữa 2 mốc thời gian.
2. Viết chương trình sử dụng kiểu cấu trúc để hiển thị ngày, tháng, năm ra màn hình, và tính khoảng cách giữa 2 ngày.
3. Viết chương trình khai báo kiểu dữ liệu thể hiện một số phức. Sử dụng kiểu này để viết hàm tính tổng, hiệu, tích của hai số phức.
4. Viết chương trình khai báo kiểu dữ liệu để biểu diễn một phân số. Hãy viết hàm thực hiện những công việc sau:
 - Tính tổng, hiệu, tích, thương hai phân số
 - Rút gọn phân số
 - Quy đồng hai phân số
 - So sánh hai phân số
5. Viết chương trình khai báo kiểu dữ liệu để biểu diễn một điểm trong hệ tọa độ Oxy. Hãy viết hàm thực hiện các công việc sau:
 - Tìm những điểm đối xứng của nó qua tung độ, hoành độ, tọa độ tâm
 - Hãy tính tổng, hiệu, tích của hai điểm trong mặt phẳng tọa độ Oxy
 - Tính khoảng cách giữa hai điểm.
6. Viết chương trình tạo một mảng các số phức. Hãy viết hàm tính tổng, tích các số phức có trong mảng.

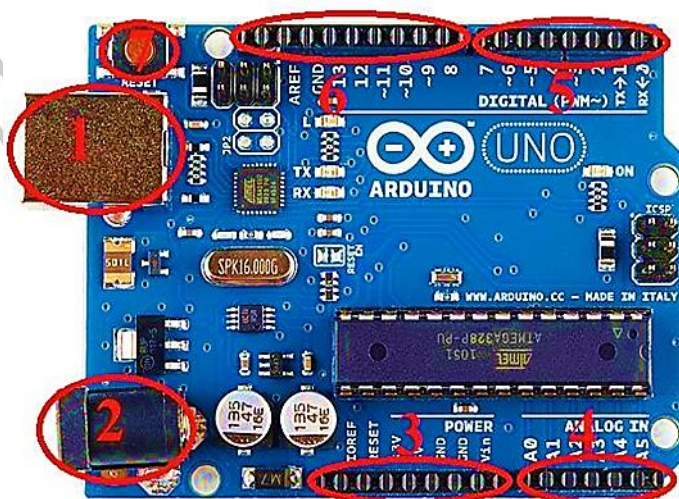
BÀI 8B. LẬP TRÌNH ỨNG DỤNG - ARDUINO

1. Tổng quan về Arduino

Arduino là một bo mạch vi xử lý được dùng để lập trình tương tác với các thiết bị phần cứng như cảm biến, động cơ, đèn hoặc các thiết bị khác. Đặc điểm nổi bật của Arduino là môi trường phát triển ứng dụng cực kỳ dễ sử dụng, với một ngôn ngữ lập trình có thể học một cách nhanh chóng ngay cả với người ít am hiểu về điện tử và lập trình. Và điều làm nên hiện tượng Arduino chính là mức giá rất thấp và tính chất nguồn mở từ phần cứng tới phần mềm. Arduino Uno là sử dụng chip Atmega328. Nó có 14 chân digital I/O, 6 chân đầu vào (input) analog, thạch anh dao động 16Mhz. Một số thông số kỹ thuật như sau :

| Chip | ATmega328 |
|--------------------------------------|--|
| Điện áp cấp nguồn | 5V |
| Điện áp đầu vào (input) (kiến nghị) | 7-12V |
| Điện áp đầu vào(giới hạn) | 6-20V |
| Số chân Digital I/O | 14 (có 6 chân điều chế độ rộng xung PWM) |
| Số chân Analog (Input) | 6 |
| DC Current per I/O Pin | 40 mA |
| DC Current for 3.3V Pin | 50 mA |
| Flash Memory | 32KB (ATmega328) với 0.5 KB sử dụng bootloader |
| SRAM | 2 KB (ATmega328) |
| EEPROM | 1 KB (ATmega328) |
| Xung nhịp | 16 MHz |

✓ Sơ đồ chân của Arduino.



Hình 8.1: Arduino Uno.

✓ Kết nối qua cổng USB (1).

Arduino sử dụng cáp USB để giao tiếp với máy tính. Thông qua cáp USB chúng ta có thể Upload chương trình cho Arduino hoạt động, ngoài ra USB còn là nguồn cho Arduino.

✓ Nguồn (2 và 3).

Khi không sử dụng USB làm nguồn thì chúng ta có thể sử dụng nguồn ngoài thông qua jack cắm 2.1mm (cực dương ở giữa) hoặc có thể sử dụng 2 chân V_{in} và GND để cấp nguồn cho Arduino. Bo mạch hoạt động với nguồn ngoài ở điện áp từ 5 – 20 volt. Chúng ta có thể cấp một áp lớn hơn tuy nhiên chân 5V sẽ có mức điện áp lớn hơn 5 volt. Và nếu sử dụng nguồn lớn hơn 12 volt thì sẽ có hiện tượng nóng và làm hỏng bo mạch. Khuyết cáo nên dùng nguồn ổn định là 5 đến dưới 12 volt.

Chân 5V và chân 3.3V (Output voltage) : các chân này dùng để lấy nguồn ra từ nguồn mà chúng ta đã cung cấp cho Arduino. Lưu ý : không được cấp nguồn vào các chân này vì sẽ làm hỏng Arduino.

Chân 0V (mass) : GND

✓ **Chip Atmega328.**

Chip Atmega328 Có 32K bộ nhớ flash trong đó 0.5k sử dụng cho bootloader. Ngoài ra còn có 2K SRAM, 1K EEPROM.

✓ **Input và Output (4, 5 và 6).**

Arduino Uno có 14 chân digital với chức năng input và output sử dụng các hàm *pinMode()*, *digitalWrite()* và *digitalRead()* để điều khiển các chân này tôi sẽ đề cập chúng ở các phần sau.

Cũng trên 14 chân digital này chúng ta còn một số chân chức năng đó là:

- Serial : chân 0 (Rx), chân 1 (Tx). Hai chân này dùng để truyền (Tx) và nhận (Rx) dữ liệu nối tiếp TTL. Chúng ta có thể sử dụng nó để giao tiếp với cổng COM của một số thiết bị hoặc các linh kiện có chuẩn giao tiếp nối tiếp.
- PWM (pulse width modulation): các chân 3, 5, 6, 9, 10, 11 trên bo mạch có dấu “~” là các chân PWM chúng ta có thể sử dụng nó để điều khiển tốc độ động cơ, độ sáng của đèn...
- SPI : 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK), các chân này hỗ trợ giao tiếp theo chuẩn SPI.
- I2C: Arduino hỗ trợ giao tiếp theo chuẩn I2C. Các chân A4 (SDA) và A5 (SCL) cho phép chúng ta giao tiếp giữa Arduino với các linh kiện có chuẩn giao tiếp là I2C.

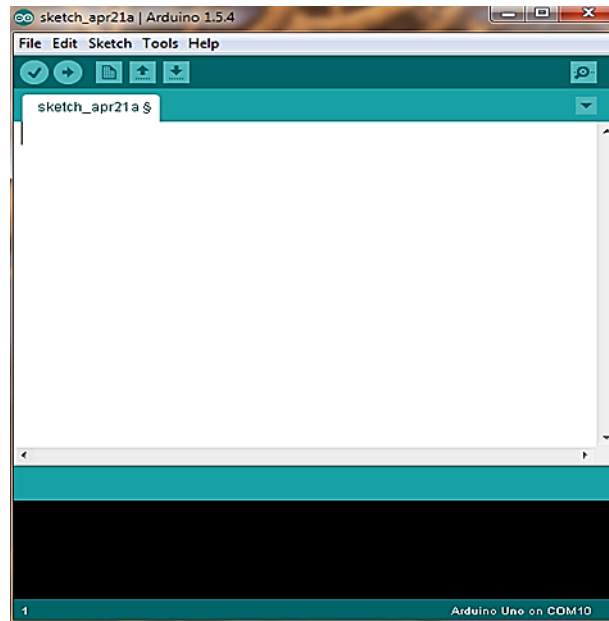
✓ **Reset (7): dùng để reset Arduino.**

2. Cài đặt chương trình Arduino IDE và Driver cho Arduino

2.1. Cài đặt chương trình Arduino IDE

Truy cập vào trang web <http://arduino.cc/en/Main/Software> và tải về chương trình Arduino IDE phù hợp với hệ điều hành của máy mình bao gồm Windows, Mac OS hay Linux. Đối với Windows có bản cài đặt (.exe) và bản Zip, đối với Zip thì chỉ cần giải nén và chạy chương trình không cần cài đặt.

Sau khi cài đặt xong thì giao diện chương trình như sau:



Hình 8.2: Arduino IDE.

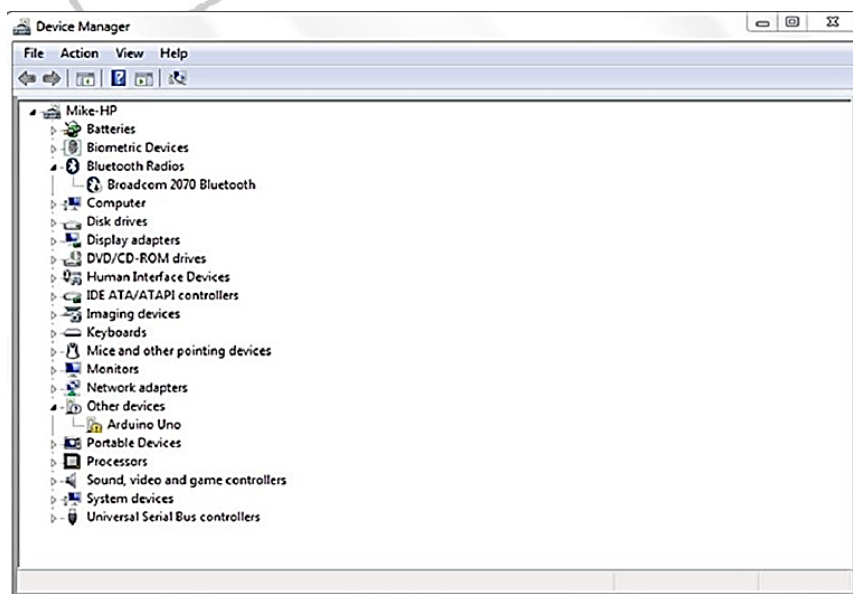
✓ Cài đặt Driver

Sử dụng cáp USB kết nối Arduino với máy tính, lúc này bạn sẽ thấy đèn led power của bo sáng. Máy tính sẽ nhận dạng thiết bị và bạn sẽ nhận được thông báo: “*Device driver software was not successfully installed*”



Hình 8.3: Driver Software Installation.

Bây giờ, click vào *Start Menu* chọn *Control Panel* kế đến chọn *System and Security*, click *System* và sau đó chọn *Device Manager*.



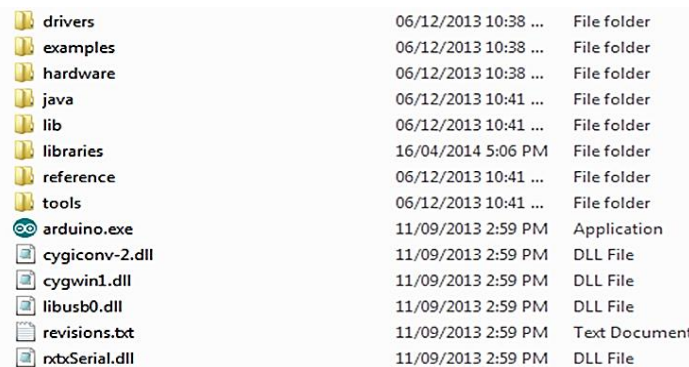
Hình 8.4: Device Manager.

Chúng ta sẽ thấy cảnh báo màu vàng thiếu driver trên Arduino. Click chuột phải trên Arduino Uno icon sau đó chọn “Update Driver Software”



Hình 8.5: Right click và chọn “Update Driver Software”.

Sau đó, chọn đường dẫn tới folder “driver” nơi mà phần mềm Arduino được lưu trữ.

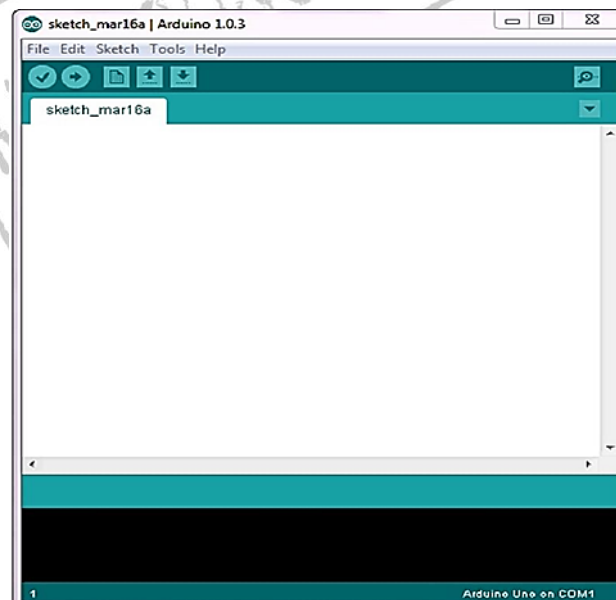


Hình 8.6: Driver.

Click “Next” Windown tự động cài đặt driver, qua trình cài đặt driver hoàn tất.

2.2. Arduino IDE

Arduino IDE là nơi để soạn thảo code, kiểm tra lỗi và upload code cho arduino







Hình 8.7: Arduino IDE.

Arduino Toolbar: có một số button và chức năng của chúng như sau :



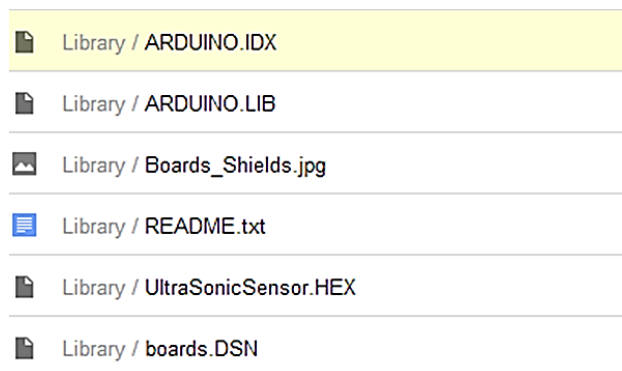
Hình 8.8: Arduino Toolbar.

- Verify : kiểm tra code có lỗi hay không 
- Upload: nạp code đang soạn thảo vào Arduino 
- New, Open, Save : Tạo mới, mở và Save sketch 
- Serial Monitor : Đây là màn hình hiển thị dữ liệu từ Arduino gửi lên máy tính 

2.3. Hướng dẫn cài đặt bản mô phỏng Arduino trên Proteus.

Để mô phỏng được Arduino trên proteus thì chúng ta cần phải download thư viện arduino cho proteus. Để có được thư viện này các bạn cần truy cập vào trang web:

<http://blogembarcado.blogspot.com/search/label/Proteus>



Hình 9: Thư viện mô phỏng Arduino.

Sau khi download về các bạn chép 2 file ARDUINO.IDX và ARDUINO.LIB vào thư mục:

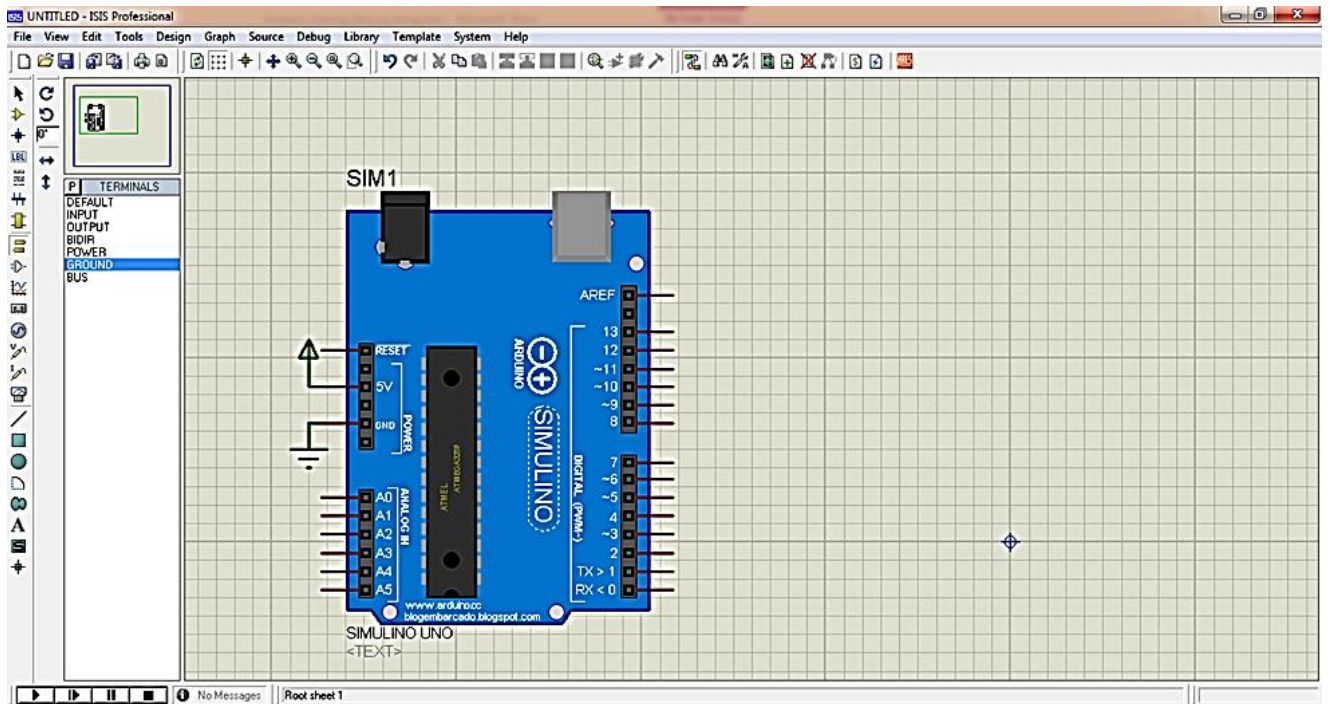
- Proteus 7:

C:\Program Files (hoặc x86) \Labcenter Electronics\Proteus 7 Professional\LIBRARY

- Proteus 8:

C:\Program Files (hoặc x86) \Labcenter Electronics\Proteus 8 professional\Data\LIBRARY

Trong thư viện này hỗ trợ 5 loại board Arduino khác nhau trong đó gồm có Arduino Uno, MEGA, NANO, LILYPAD và UNO SMD và một cảm biến siêu âm Untrasonic. Sau khi chép xong chúng ta khởi động Proteus lên vào thư viện linh kiện bằng cách bấm phím P và gõ từ khoá là ARDUINO chúng sẽ hiện ra danh sách các board hiện có ở đây tôi chọn Arduino Uno.



Hình 8.10: Mô phỏng Arduino bằng Proteus.

Lưu ý chúng ta cần phải cấp nguồn vào 2 chân 5V và Gnd trên mạch như hình trên.

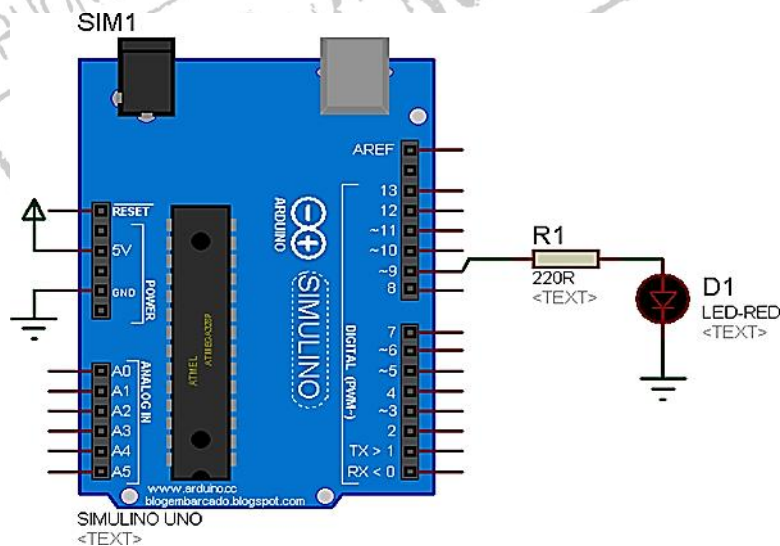
3. Một số ví dụ lập trình Arduino

3.1. Project 1: Led nhấp nháy (protues và board)

Sau đây chúng ta tạo project nhấp nháy led thời gian delay là 1 giây sử dụng proteus để mô phỏng.

✓

Sơ đồ mạch:



Hình 8.11: Led nhấp nháy.


✓

Code chương trình.

```
int ledPin = 9;

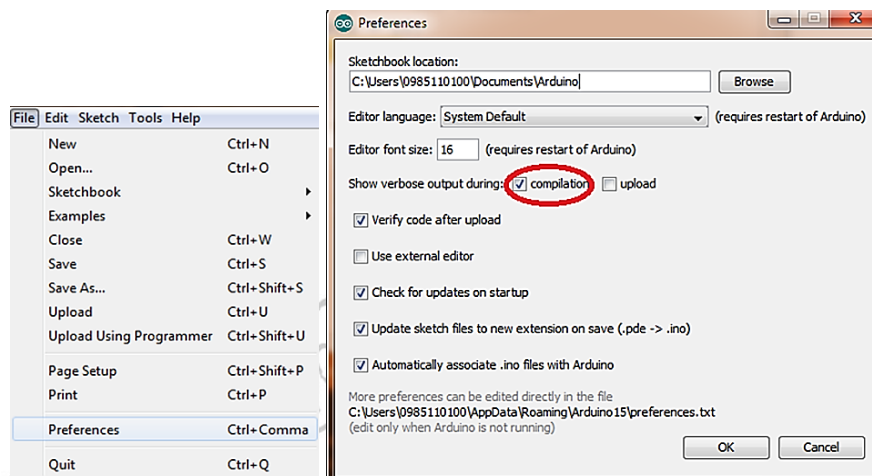
void setup() {
  pinMode(ledPin, OUTPUT);
}
```

```
void loop() {
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```


Sau khi gõ code vào chương trình soạn thảo bạn cần click vào  để kiểm tra lỗi.

✓ Tạo File Hex.

Chúng ta cần phải có file Hex để cung cấp cho proteus và khi bấm play chương trình mới hoạt động được. Cách tạo file Hex trên Arduino IDE như sau: Click vào File chọn Preferences. Các bạn check vào compilation và OK.

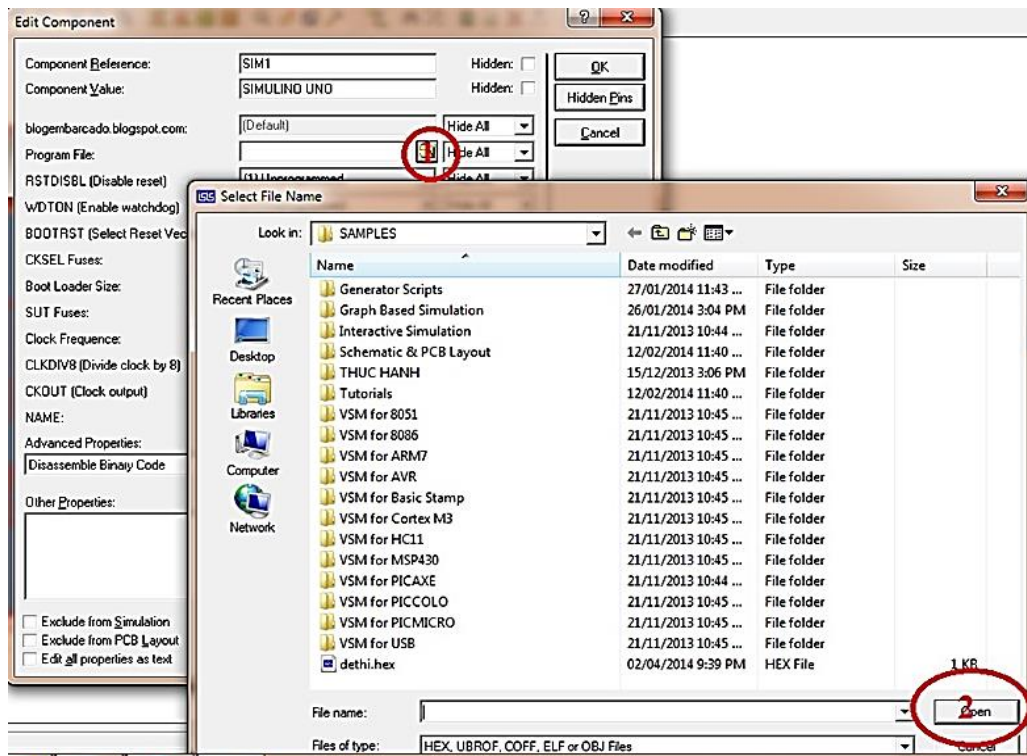


Hình 8.12: Click Preferences và Check compilation

Sau đó tiếp tục bấm  . Chương trình sẽ tạo file hex được lưu ở đường dẫn như hình dưới. Hãy chép file hex ra một thư mục nào đó sau đó mở proteus lên và double click vào Arduino Uno.

```
C:\Users\098511~1\AppData\Local\Temp\build9118448727674380763.tmp\Blink.cpp.hex
Sketch uses 1,116 bytes (3%) of program storage space. Maximum is 32,256 bytes.
Global variables use 11 bytes (0%) of dynamic memory, leaving 2,037 bytes for local variables. Maximum is 2,048 bytes.
11
```

Hình 8.13: Đường dẫn chứa file hex.



Hình 8.14: Add file Hex cho Proteus.

Bấm vào vị trí số 1 và chọn nơi lưu file hex ở trên chọn tiếp Open, OK và Play. Các bạn sẽ thấy led nhấp nháy tắt và sáng thời gian delay là 1s.

✓ Giải thích chương trình.

```
int ledPin = 9;
```

Khai báo một giá trị biến integer là *ledPin* = 9.

```
void setup() {
```

```
pinMode(ledPin, OUTPUT); }
```

Trong Arduino sketch cần phải có hàm `setup()` và `loop()` nếu không có thì chương trình báo lỗi. Hàm `Setup()` chỉ chạy một lần kể từ khi bắt đầu chương trình. Hàm này có chức năng thiết lập chế độ vào, ra cho các chân digital hay tốc độ baud cho giao tiếp Serial, v.v.

Cấu trúc của hàm `pinMode()` là như sau:

`pinMode(pin, Mode);` *pin* : là vị trí chân digital; *Mode*: là chế độ vào (INPUT), ra (OUTPUT).

```
pinMode(ledPin, OUTPUT);
```

Lệnh này thiết lập chân số 9 trên board là chân ngõ ra (*OUTPUT*). Nếu không khai báo “`int ledPin = 9;`” thì bạn có thể viết cách sau nhưng ý nghĩa không thay đổi: `pinMode(9, OUTPUT);`

Bắt buộc khai báo một hàm `loop()` trong Arduino IDE. Hàm này là vòng lặp vô hạn

```
digitalWrite( ledPin, HIGH);
```

Lệnh này có ý nghĩa là xuất ra chân digital có tên là *ledPin* (chân 9) mức cao (*HIGH*), mức cao tương ứng là 5 volt.

```
delay(1000);
```

Lệnh này tạo một khoảng trễ với thời gian là 1 giây. Trong hàm `delay()` của IDE thì 1000 tương ứng với 1 giây.

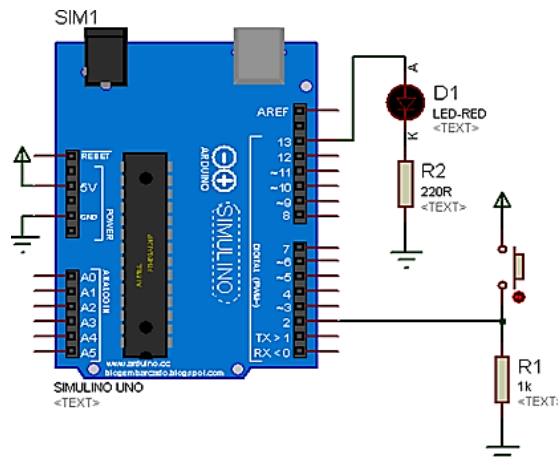
digitalWrite(ledPin, LOW);

Cũng giống như *digitalWrite(ledPin, HIGH);* lệnh này xuất ra chân *ledPin* mức thấp (LOW) tức là 0 volt. Và tiếp tục là một hàm delay().

Như vậy chúng ta có thể thấy chương trình sẽ thực hiện tắt sáng led liên tục không ngừng trừ khi ta ngắt nguồn.

3.2. Project 2 : Đèn sáng khi nhấn phím.

✓ Sơ đồ mạch:



Hình 8.15: Đèn sáng khi nhấn phím

✓ Code chương trình :

```
const int buttonPin = 2;
const int ledPin = 13;
int buttonState = LOW;
void setup() {
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);
}
void loop(){
    buttonState = digitalRead(buttonPin);
    if (buttonState == HIGH) {
        digitalWrite(ledPin, HIGH);
    }
    else {
        digitalWrite(ledPin, LOW);
    }
}
```

✓ Giải thích chương trình :

Trước tiên ta khai báo hai biến để lưu trữ vị trí chân của phím nhấn và led:

const int buttonPin = 2;


```
const int ledPin = 13;
```

```
int buttonState = LOW;
```

Phím nhấn sẽ ở vị trí chân số 2 và led chân số 13. Ta khai báo một biến trạng thái của phím nhấn tên là buttonState.

```
pinMode(ledPin, OUTPUT);
```

```
pinMode(buttonPin, INPUT);
```

Trong hàm *setup()* là khai chế độ (Mode) cho chân button và chân led. Chân button là chân ngõ vào và chân led là chân ngõ ra.

```
buttonState = digitalRead(buttonPin);
```

Trong hàm *loop()* ta có câu lệnh đầu tiên là gán giá trị đọc được từ chân button (chân 2) cho biến *buttonState*. *buttonState* sẽ có giá trị 0 nếu như button không được nhấn và có giá trị 1 nếu được nhấn. Bằng cách sử dụng hàm *digitalRead()* ta có thể kiểm tra được các chân digital đang ở mức cao hay thấp.

```
if (buttonState == HIGH) {
```

```
    digitalWrite(ledPin, HIGH);
```

```
}
```

Sau khi đọc được giá trị có ở chân *buttonPin* (chân 2) ta kiểm tra xem là button có nhấn hay không. Nếu có tức là *buttonState* = *HIGH* thì lúc này ta bật led bằng lệnh *digitalWrite()*.

```
else {
```

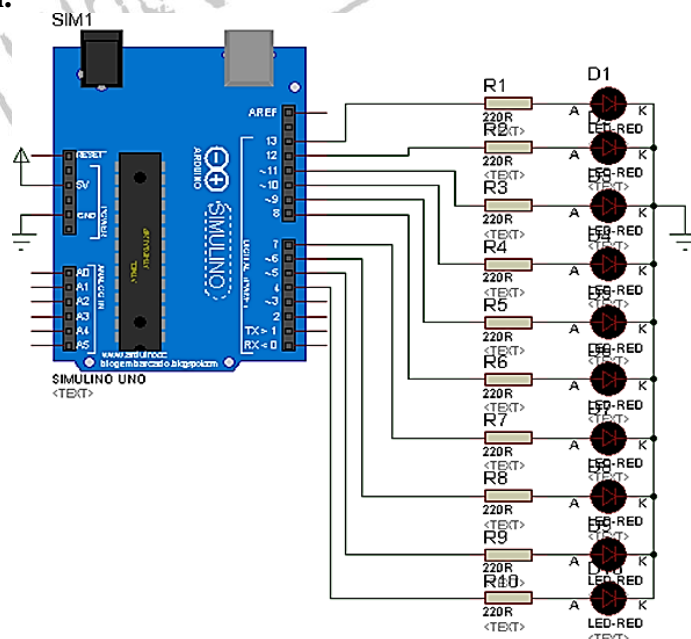
```
    digitalWrite(ledPin, LOW);
```

```
}
```

Ngược lại thì ta một lần nữa sử dụng hàm *digitalWrite()* để tắt led.

3.3. Project 3 : Led sáng dần từ led 1 đến led 10 và ngược lại.

✓ Sơ đồ mạch.



Hình 8.16: Led sáng dần từ led 1 đến led 10 và ngược lại.

✓ Code chương trình.

```

byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
int direction = 1;
int currentLED = 0;
void setup() {
    for (int x=0; x<10; x++) {
        pinMode(ledPin[x], OUTPUT);
    }
}
void loop() {
    for (int x=0; x<10; x++) {
        digitalWrite(ledPin[x], LOW);
    }
    digitalWrite(ledPin[currentLED], HIGH);
    currentLED += direction;
    if (currentLED == 9) {
        direction = -1;
    }
    if (currentLED == 0) {
        direction = 1;
    }
    delay(500);
}

```

✓ Giải thích chương trình.

Trong Project, chúng ta sử dụng 10 chân digital để điều khiển 10 led, để cho chương trình ngắn gọn thì ở đây tôi sử dụng mảng 1 chiều gồm 10 phần tử trong đó chứa 10 vị trí chân led mà ta sử dụng trong project

```
byte ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

Tiếp tục khai báo 2 biến integer là :

```
int direction = 1;
```

```
int currentLED = 0;
```

Trong hàm *setup()*, sử dụng một vòng lặp để định nghĩa mode cho các chân led.

Tiếp theo là hàm *loop()*, đầu tiên tắt tất cả các led bằng các câu lệnh:

```
for (int x=0; x<10; x++) {
```

```
    digitalWrite(ledPin[x], LOW);
```

```
}
```

Sau đó cho sáng led đầu tiên bằng câu lệnh :

```
digitalWrite(ledPin[currentLED], HIGH);
```

Vì ta đã khai báo $currentLED = 0$ nên mảng sẽ truy xuất phần tử đầu tiên trong mảng có giá trị là 4 vì thế led ở vị trí chân digital số 4 sẽ sáng.

```
currentLED += direction;
```

Tăng $currentLED$ lên 1 đơn vị ($direction = 1$). Vòng lặp tiếp theo sẽ là led ở chân digital 5 sáng và cứ như thế cho đến led ở chân số 13 sáng, thì lúc này $currentLED == 9$, câu lệnh “ $if (currentLED == 9) \{ direction = -1; \}$ ” sẽ thực hiện và led sẽ sáng ngược lại từ led 10 xuống led thứ 1.

Hai câu lệnh sau dùng để quy định chiều sáng của led là tăng dần hay giảm dần. Nếu là Led thứ 10 sáng thì tiếp theo sẽ giảm xuống led thứ 9 và ngược lại nếu led thứ 0 sáng thì chu kỳ tiếp theo led 1 sẽ sáng.

```
if (currentLED == 9) {
```

```
    direction = -1;
```

```
}
```

```
if (currentLED == 0) {
```

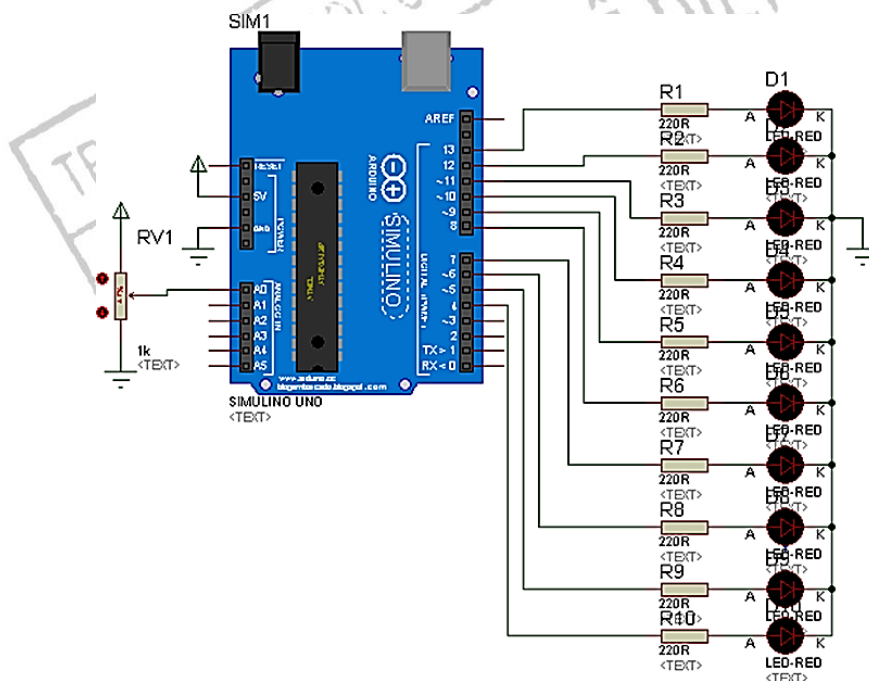
```
    direction = 1;
```

```
}
```

3.4. Project 4 : Led sáng dần từ led 1 đến led 10 và ngược lại thời gian delay thay đổi được.

✓ Sơ đồ mạch :

Trong project này có thêm một biến trở tại chân A0 dùng để điều chỉnh thông số hàm delay.



Hình 8.17: Led sáng dần từ led 1 đến led 10 thời gian delay thay đổi được.

✓ Code chương trình.

```
int ledPin[] = {4, 5, 6, 7, 8, 9, 10, 11, 12, 13};
```

```
int direction = 1;
```

```
int currentLED = 0; int potPin = 0;
```

```
unsigned long changeTime;
```



```

void setup() {
    for (int x=0; x<10; x++) {
        pinMode(ledPin[x], OUTPUT);
    }
}

void loop() {
    int delayvalue= analogRead(potPin);
    for (int x=0; x<10; x++) {
        digitalWrite(ledPin[x], LOW);
    }
    digitalWrite(ledPin[currentLED], HIGH);
    currentLED += direction;
    if (currentLED == 9) {
        direction = -1;
    }
    if (currentLED == 0) {
        direction = 1;
    }
    delay(delayvalue);
}

```



Giải thích chương trình.

Chương trình ta chỉ thêm và thay đổi một vài câu lệnh mà thôi ngoài ra không khác gì nhiều so với project 3, các câu lệnh đó như sau :

```
int potPin = 0 ;
```

```
int delayvalu= analogRead(potPin);
```

```
delay(delayvalue);
```

Đầu tiên chúng ta khai báo một biến chứa vị trí chân biến trở kết nối đó là vị trí A0

```
int potPin = 0 ;
```

Đọc giá trị từ chân analog A0 bằng câu lệnh *analogRead(potPin)* và gán nó cho biến *delayvalue*. Arduino có 6 chân đầu vào analog đánh dấu từ A0 đến A5 với 10 bit chuyển đổi từ analog sang digital (ADC). Nghĩa là chân analog có thể đọc được các giá trị điện áp từ 0 đến 5 volt tương ứng với các số integer từ 0 (0 volt) đến 1023 (5 volt).

Trong project này chúng ta cần thiết lập thời gian delay bằng cách điều chỉnh biến trở. Ta sử dụng câu lệnh *delay(delayvalue)* để tạo thời gian trễ. Nếu ta điều chỉnh biến trở sao cho điện áp đầu vào chân analog là 5 volt thì *delayvalue* sẽ có giá trị là 1023 (hơn 1 giây), nếu là 2,5 volt thì *delayvalue* sẽ là 511. Hãy điều chỉnh biến trở ta sẽ thấy thời gian delay thay đổi hoặc là nhanh dần hoặc là chậm dần.

Lưu ý : đối với các chân analog chúng ta không cần thiết lập chế độ vào ra bằng hàm *pinMode* như các chân digital. Mặc định các chân analog là input.

3.5. Project 5: Giao tiếp với máy tính (protues và Board).

Trong phần này, chúng ta tìm hiểu cách để giao tiếp giữa Arduino với máy tính thông qua chuẩn giao tiếp nối tiếp không đồng bộ UART. Điều khiển bật tắt bằng cách gửi lệnh từ máy tính.

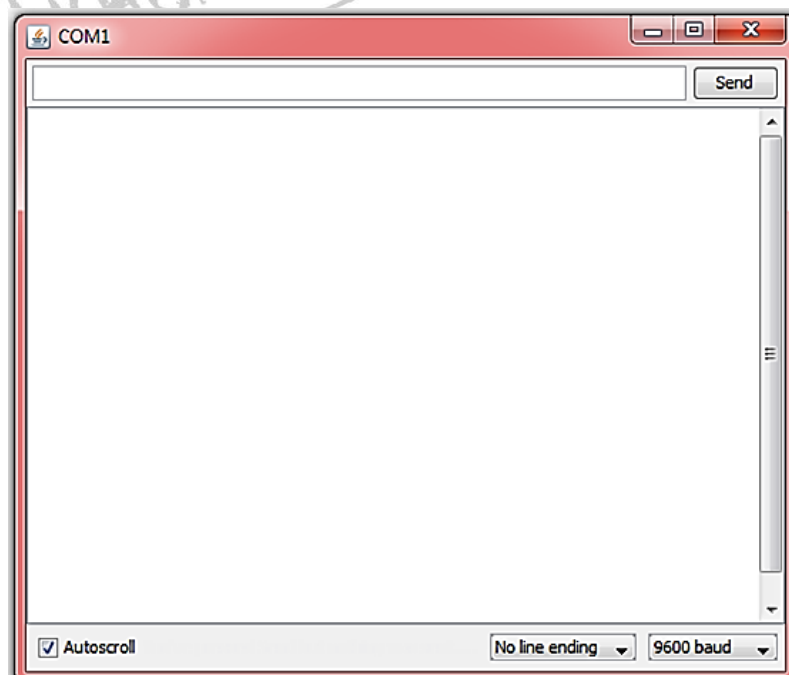
✓ Sơ đồ mạch:

Dùng cáp USB kết nối Arduino với máy tính. Led nối với chân 13 thông qua điện trở 220 ohm.

✓ Code chương trình:

```
int ledpin =13;
void setup() {
    Serial.begin(9600);
    pinMode(ledpin,OUTPUT);
}
void loop() {
    if( Serial.available(>0)
        char setupled = Serial.read();
        switch (setupled) {
            case '1': { digitalWrite(ledpin,HIGH); break;
            }
            case '0': { digitalWrite(ledpin,LOW); break;
            }
        }
    }
```

✓ Giải thích chương trình:



Hình 8.18: Serial Monitor.

Để có thể điều khiển được led bật tắt chúng ta cần mở Serial monitor. Trong chương trình ta cần chú ý tới các câu lệnh sau:

`Serial.begin(9600);`

Câu lệnh này thiết lập tốc độ truyền dữ liệu của chúng ta là 9600 bps. Chúng ta có thể thiết lập các tốc độ khác như 300, 1200, 2400, 4600, 9600, 19200, 57600, 115200. Cần lưu ý rằng để tốc độ truyền giữa máy tính và thiết bị phải giống nhau, nếu không thì dữ liệu nhận được sẽ bị lỗi.

Trong vòng lặp **`loop()`** chúng ta có câu lệnh:

`Serial.available()>0`

Câu lệnh này dùng để kiểm tra xem có dữ liệu truyền tới hay không. Ngoài ra **`Serial.available()`** còn trả về cho chúng ta số ký tự đã được truyền tới Arduino

`char setupled = Serial.read();`

Khi dữ liệu được truyền tới Arduino thì dữ liệu sẽ được lưu vào bộ nhớ đệm. Chúng ta khai báo biến **`setupled`** với kiểu dữ liệu char và dùng hàm **`Serial.read()`** để truy xuất dữ liệu trong bộ nhớ đệm và lưu vào trong nó. Như vậy ký tự đầu tiên trong chuỗi ký tự được truyền tới sẽ được gán vào **`setupled`**. Dùng hàm switch-case để kiểm tra, nếu là “1” thì sáng led, nếu là “0” thì tắt led, các trường hợp còn lại thì không làm gì.

BÀI TẬP

- 4.1. Sử dụng **board arduino** kết nối máy tính qua cổng USB. Viết chương trình cộng/ trừ/ nhân/ chia 2 số nguyên (integer) trên arduino. Cho biết giá trị 2 số nguyên nhập vào từ serial monitor và kết quả phép cộng hiển thị trên serial monitor.
- 4.2. Sử dụng **board arduino** kết nối máy tính qua cổng USB. Viết chương trình cộng/ trừ/ nhân/ chia 2 số thực (float) trên arduino. Cho biết giá trị 2 số thực nhập vào từ serial monitor và kết quả phép cộng hiển thị trên serial monitor.
- 4.3. Sử dụng **board arduino** kết nối máy tính qua cổng USB. Viết chương trình cho arduino thực hiện nhiệm vụ đếm **giờ, phút, giây**. Giá trị giờ, phút, giây được hiển thị trên serial monitor. Ngoài ra, giá trị giờ, phút, giây có thể được cài đặt ban đầu từ serial monitor.