ỦY BAN NHÂN DÂN HUYỆN CỦ CHI
TRƯỜNG TRUNG CẤP NGHỀ CỦ CHI

GIÁO TRÌNH
MÔN HỌC/MÔ ĐUN: LẬP TRÌNH CĂN BẢN
NGÀNH/NGHỀ: KỸ THUẬT SỮA CHỮA & LẮP RÁP MÁY TÍNH
TRÌNH ĐỘ: TRUNG CẤP

Ban hành kèm theo Quyết định số: 89/QĐ-TCN ngày 15 tháng 08 năm 2024
của Hiệu trưởng Trường Trung cấp nghề Củ Chi

Củ Chi, năm 2024

Lời giới thiệu

Giáo trình là một trong ba yếu tố quyết định chất lượng dạy nghề. Nhằm đảm bảo tính thống nhất trong quản lý và thực hiện chương trình giáo trình trong thời gian tới tại trường Trung cấp nghề Củ Chi, để từng bước nâng cao chất lượng đào tạo.

Giáo trình "Lập trình căn bản" được biên soạn theo đơn vị bài học. Nội dung của giáo trình được nhóm biên soạn, xây dựng dựa trên cơ sở chi tiết hóa chương trình môn học "Lập trình căn bản" trình độ trung cấp nghề, đồng thời có sự tham khảo các tài liệu, cập nhật các nội dung mới và những kinh nghiệm thực tế giảng dạy.

Nội dung chính của giáo trình được chia thành 06 bài, bao gồm các nội dung:

- Bài 1: Tổng quan ngữ lập trình
- Bài 2: Các thành phần cơ bản của ngôn ngữ lập trình
- Bài 3: Các cấu trúc điều khiển
- Bài 4: Hàm và thủ tục
- Bài 5: Dữ liệu kiểu tập hợp, mảng và bản ghi
- Bài 6: Dữ liệu kiểu chuỗi

Giáo trình môn học "Lập trình căn bản" được dùng làm tài liệu giảng dạy và học tập cho giáo viên và học sinh sinh viên nghề lắp ráp cài đặt & sữa chữa máy tính tại trường Trung cấp nghề Củ Chi, cũng như mọi đối tượng quan tâm đến lĩnh vực nghề máy tính.

Tác giả biên soạn giáo trình "Lập trình căn bản" xin chân thành cảm ơn những ý kiến góp ý đánh giá vô cùng quý giá của các đồng nghiệp, và những ý kiến phản biện khoa học của các Nhà giáo, các nhà quản lý và các Doanh nghiệp trong Hội đồng nghiệm thu giáo trình, để cuốn giáo trình "Lập trình căn bản" được hoàn thiện ra mắt phục vụ cho quá trình dạy và học.

……, ngày … tháng … năm 2024

Tham gia biên soạn

MỤC LỤC

CHƯƠNG TRÌNH MÔN HỌC

(Kèm theo Thông tư số: 03/2017/TT-BLĐXH ngày 01/03/2017
Của Bộ trưởng Bộ Lao động – Thương binh và Xã hội)

Tên môn học: LẬP TRÌNH CĂN BẢN

Mã số môn học: MH 11

Thời gian môn học: 45 giờ *(Lý thuyết: 15 giờ; Thực hành: 27 giờ; Kiểm tra: 3 giờ)*

I. VỊ TRÍ, TÍNH CHẤT MÔN HỌC:

- Vị trí: Môn học được bố trí sau khi học sinh học xong các môn học chung, tin văn phòng.
- Tính chất: Là môn học lý thuyết cơ sở nghề bắt buộc.

II. MỤC TIÊU MÔN HỌC: Về kiến thức:

- Trình bày được khái niệm về lập trình máy tính;
- Mô tả được ngôn ngữ lập trình: cú pháp, công dụng của các câu lệnh;
- Phân tích được chương trình: xác định nhiệm vụ chương trình;
- Thực hiện được các thao tác trong môi trường phát triển phần mềm: biên tập chương trình, sử dụng các công cụ, điều khiển, thực đơn lệnh trợ giúp, gỡ rối, bẫy lỗi,...
- Mô tả được bài toàn thực tiễn vào máy tính, dữ liệu vào, dữ liệu ra, xử lí mã nguồn.

Về kỹ năng:

- Thực hành viết chương trình từ căn bản đến bài toàn thực tiễn;
- Phân tích bài toán, xác định rõ dữ liệu vào-ra và xử lí mã nguồn;
- Tìm lỗi, xử lí lỗi, đọc các thông báo của trình biên dịch;
- Thao tác sử dụng các chức năng của trình biên dịch;
- Kỹ năng sử dụng máy tính thành thạo hơn.

Về năng lực tự chủ và trách nhiệm:

- Rèn luyện lòng yêu nghề, tư thế tác phong công nghiệp , tính kiên trì, sáng tạo trong công việc, trao đổi học hỏi bạn bè, làm việc nhóm, trách nhiệm với môn học, nội quy thực hành, khả năng tự tìm hiểu.
- Rèn luyện trách nhiệm tự học, tự tìm hiểu thêm về môn học.
- Thực hiện an toàn sử dụng máy tính, điện, trách nhiệm với trang thiết bị phòng học.
- Bố trí làm việc khoa học đảm bảo an toàn cho người và phương tiện học tập.

CHƯƠNG 1. TỔNG QUAN NGÔN NGỮ LẬP TRÌNH

Giới thiệu:

Bài này nhằm giới thiệu cho học sinh những kiến thức tổng quan về ngôn ngữ lập trình. Cách thao tác và làm quen với ngôn ngữ lập trình. Đơn cử trong giáo trình này sử dụng ngôn ngữ lập trình căn bản C.

1. Mục tiêu:

- Trình bày được các khái niệm về lập trình;
- Trình bày được lịch sử phát triển, ứng dụng của ngôn ngữ lập trình;
- Làm quen môi trường phát triển phần mềm;
- Sử dụng được hệ thống trợ giúp từ help file;
- Thực hiện các thao tác an toàn với máy tính.

2. Nội dung:

Ngôn ngữ C ra đời năm 1970 – phát minh bởi Dennis Ritchie được thử nghiệm và chạy trên nền tảng Unix và được ban hành chuẩn hóa lần đầu tiên vào năm 1989 (C89 or C90). Sau đó tiếp tục được chuẩn hóa với nhiều phiên bản khác nhau như C95, C99, C11.

2.1 Giới thiệu các khái niệm cơ bản về lập trình:

Ngôn ngữ hướng cấu trúc;

Ngôn ngữ sách tay (hỗ trợ nhiều trình dịch);

Phục vụ viết hệ điều hành, lập trình nhúng…;

Viết trình biên dịch;

Các trình dịch cho C tạo ra nhanh chóng trên các nền tảng mới => lưu động.

Ví dụ: Viết chương trình hiển thị cụm từ "Hello world" lên màn hình

```
#include <stdio.h>  // Needed to perform IO operations
 int main()
{                          // Program entry point
   printf("Hello, world!\n");  // Says Hello
   return 0;            // Terminate main()
}                         // End of main()
```

2.2 Giới thiệu lịch sử phát triển và ứng dụng của ngôn ngữ lập trình:

GCC <Gnu Compiler Collection>

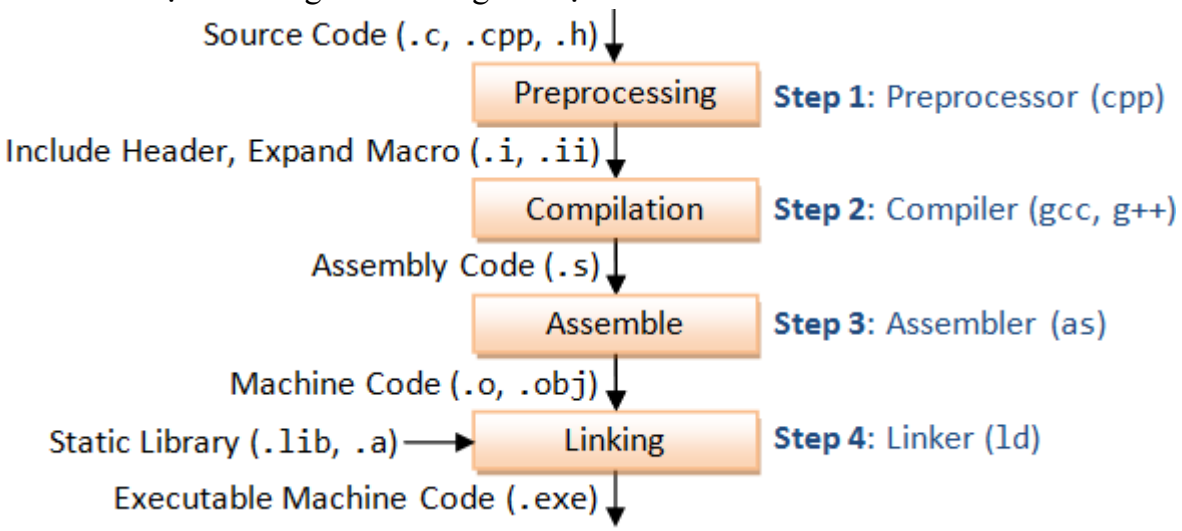
- Bộ trình dịch GNU: tập hợp các trình dịch được thiết kế cho nhiều ngôn ngữ khác nhau.
- Tên gốc GCC: CNU C Compiler (trình dịch C của GNU).
- Hiện nay GCC hỗ trợ đầu vào cho nhiều ngôn ngữ và tương thích với rất nhiều nền tảng.

GCC cho ngôn ngữ lập trình C

- GCC: cho phép thực hiện các quá tình tiền xử lý, biên dịch, lắp ráp và liên kết để tạo ra một chương trình.
- Sử dụng các tùy chọn cho phép chúng ra thực hiện các bước hay dừng lại ở giai đoạn trung gian (Ví dụ: tùy chọn -c : chỉ biên dịch và không liên kết).

GCC command options:

- Biên dịch chương trình có 4 giai đoạn :



- Lưu ý:

  Việc sử dụng các options có tác dụng (công dụng) điều khiển các loại đầu ra như: file thực thi, tập tin đối tượng (.o, .obj); file lắp ráp hay quá trình tiền xử lý nguồn.

  Với bất kỳ tập tin đầu vào thì tên hậu tố của file sẽ xác định cách thức biên dịch.

  Với các file đầu vào không xác định được hậu tố thì được xác định là đầu vào cho giai đoạn liên kết.

  Có thể chỉ định tệp tin đầu vào là ngôn ngữ gì. (thay thế cho việc trình dịch mặc định lựa chọn dựa theo hậu tố)

Make – makefile là gì?

- Công cụ hỗ trợ biên dịch
- Ngôn ngữ kịch bản
- Hỗ trợ việc xây dựng lại khi có thay đổi (chỉ tác động lên các thành phần phù hợp).
- Cú pháp:

```
#comment
target: dependency1 dependency2 ...
<tab> command
```

- Ví dụ quá trình liên kết 2 file đối tượng (program.o & mylib.o) thành file thực thi (program):

```
program: program.o mylib.o
<tab> gcc -o program program.o mylib.o
```

GDB + DDD

GDB (GNU DEBUGGER)

- Tên gọi tập tin thực thi.
- Là một chuẩn debugger (gỡ lỗi) cho hệ điều hành GNU. Được viết năm 1986 bởi Richard Stallman.
- Cho phép kiểm soát quá trình hoạt động của một chương trình.

- GDB có thể thực hiện 4 tác vụ chính:
  Bắt đầu chương trình, xác định các tác nhân ảnh hưởng tới chương trình.
  Kiểm tra những gì đã diễn ra khi chương trình đã dừng.
  Làm chương trình dừng lại với các điều kiện.
  Thay đổi bất cứ điều gì trong chương trình, bạn có thể thử nghiệm các trường hợp khi debug.

DDD (DATA DISPLAY DEBUGGER) <GNU DDD>

Là giao diện đồ họa người dùng cho phép gỡ lỗi chương trình dựa trên nền tảng sử dụng GDB. Sử dụng DDD cho phép hiển thị dữ liệu đồ họa, tương tác với chương trình trực quan. Sử dụng DDD có thể suy luận về chương trình thông qua dữ liệu của chương trình. Phiên bản đầu tiên của DDD được viết năm 1990 với Andreas Zeller.

IDE (Integrated Development Environment)

IDE tạm dịch: Môi trường tích hợp phát triển

Là một ứng dụng phần mềm cung cấp cơ sở toàn diện để lập trình phát triển phần mềm. Một IDE thông thường bao gồm một trình soạn thảo, xây dựng các công cụ tự động build và gỡ lỗi.

Ví dụ: Eclipse, CodeBlock, Netbean…

Lib (Library : Thư viện)

Là một bộ các phương thức, đối tượng được đóng gói để sử dụng. Sử dụng thư viện để chia sẻ mã nguồn giúp ích cho việc tái sử dụng chương trình.

- Có 2 loại thư viện:
  Thư viện liên kết động (Dynamic linking library) – Tham chiếu hàm
  Thư viện liên kết tĩnh (Static linking library) – Nạp vào chương trình
- Ưu điểm:
  Loại 1:
  - Kích thước nhỏ gọn
  - Dễ nâng cấp khi thay đổi thư viện

  + Loại 2:
  - Gọi hàm nhanh hơn
  - Phân bố mã nhị phân dễ hơn

2.3 Làm quen môi trường phát triển phần mềm:

Ngày nay, khoa học máy tính thâm nhập vào mọi lĩnh vực. Tự động hóa hiện đang là ngành chủ chốt điều hướng sự phát triển thế giới. Bất cứ ngành nghề nào cũng cần phải hiểu biết ít nhiều về Công nghệ Thông tin và lập trình nói chung. Cụ thể, C là một ngôn ngữ lập trình cấp cao mà mọi lập trình viên cần phải biết. Vì thế, trong giáo trình này, chúng ta sẽ nghiên cứu chi tiết cấu trúc ngôn ngữ C. Đầu tiên chúng ta tìm hiểu sự khác nhau của những khái niệm: Lệnh (Command), Chương trình (Program) và Phần mềm (Software).

Hoạt động của máy tính

Khi một máy tính được khởi động, nó sẽ tự động thực thi một số tiến trình và xuất kết quả ra màn hình. Điều này diễn ra thế nào? Câu trả lời đơn giản là nhờ vào Hệ điều hành cài đặt bên trong máy tính. Hệ điều hành (operating system) được xem như phần

mềm hệ thống. Phần mềm này khởi động máy tính và thiết lập các thông số ban đầu trước khi trao quyền cho người dùng. Để làm được điều này, hệ điều hành phải được cấu tạo từ một tập hợp các chương trình. Mọi chương trình đều cố gắng đưa ra lời giải cho một hay nhiều bài toán nào đó. Mọi chương trình cố gắng đưa ra giải pháp cho một hay nhiều vấn đề. Mỗi chương trình là tập hợp các câu lệnh giải quyết một bài toán cụ thể. Một nhóm lệnh tạo thành một chương trình và một nhóm các chương trình tạo thành một phần mềm.

Để rõ hơn, chúng ta hãy xem xét một thí dụ : Một người bạn đến nhà chúng ta chơi và được mời món sữa dâu. Anh ta thấy ngon miệng và muốn xin công thức làm. Chúng ta hướng dẫn cho anh ta làm như sau :

1. Lấy một ít sữa.
2. Đổ nước ép dâu vào.
3. Trộn hỗn hợp này và làm lạnh.

Bây giờ nếu bạn của chúng ta theo những chỉ dẫn này, họ cũng có thể tạo ra món sữa dâu tuyệt vời.

Chúng ta hãy phân tích chỉ thị (lệnh) ở trên:

- Lệnh đầu tiên : Lệnh này hoàn chỉnh chưa ? Nó có trả lời được câu hỏi lấy sữa ‘ở đâu’ ?.
- Lệnh thứ hai : Một lần nữa, lệnh này không nói rõ nước ép dâu để ‘ở đâu’.

May mắn là bạn của chúng ta đủ thông minh để hiểu được công thức pha chế nói trên, dù rằng còn nhiều điểm chưa rõ ràng. Do vậy nếu chúng ta muốn phổ biến cách làm, chúng ta cần bổ sung các bước như sau :

1. Rót một ly sữa vào máy trộn.
2. Đổ thêm vào một ít nước dâu ép.
3. Đóng nắp máy trộn
4. Mở điện và bắt đầu trộn
5. Dừng máy trộn lại
6. Nếu đã trộn đều thì tắt máy, ngược lại thì trộn tiếp.
7. Khi đã trộn xong, rót hỗn hợp vào tô và đặt vào tủ lạnh.
8. Để lạnh một lúc rồi lấy ra dùng.

So sánh hai cách hướng dẫn nêu trên, hướng dẫn thứ hai chắc chắn hoàn chỉnh, rõ ràng hơn, ai cũng có thể đọc và hiểu được.

Tương tự, máy tính cũng xử lý dữ liệu dựa vào tập lệnh mà nó nhận được. Đương nhiên các chỉ thị đưa cho máy vi tính cũng cần phải hoàn chỉnh và có ý nghĩa rõ ràng. Những chỉ thị này cần phải tuân thủ các quy tắc:

1. Tuần tự
2. Có giới hạn
3. Chính xác.

Mỗi chỉ thị trong tập chỉ thị được gọi là “câu lệnh” và tập các câu lệnh được gọi là “chương trình”.

Chúng ta hãy xét trường hợp chương trình hướng dẫn máy tính cộng hai số.

Các lệnh trong chương trình có thể là :

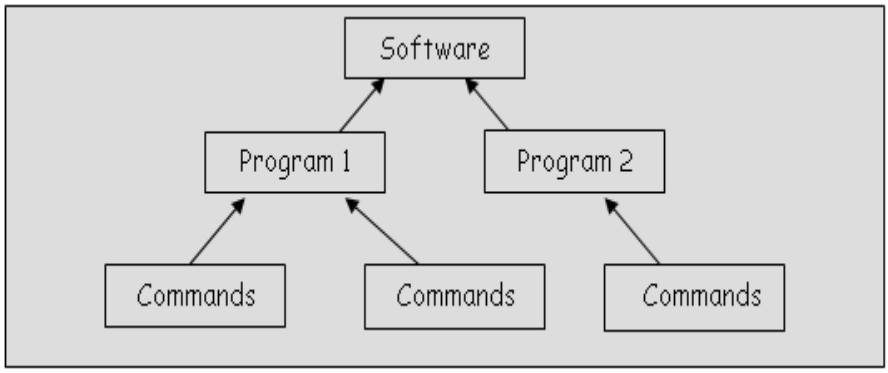
1. Nhập số thứ nhất và nhớ nó.
2. Nhập số thứ hai và nhớ nó.
3. Thực hiện phép cộng giữa số thứ nhất và số thứ hai, nhớ kết quả phép cộng.
4. Hiển thị kết quả.
5. Kết thúc.

Tập lệnh trên tuân thủ tất cả các quy tắc đã đề cập. Vì vậy, tập lệnh này là một chương trình và nó sẽ thực hiện thành công việc cộng hai số trên máy tính.

- **Ghi chú:** Khả năng nhớ của con người được biết đến như là trí nhớ, khả năng nhớ dữ liệu được đưa vào máy tính được gọi là "bộ nhớ". Máy tính nhận dữ liệu tại một thời điểm và làm việc với dữ liệu đó vào thời điểm khác, nghĩa là máy tính ghi dữ liệu vào trong bộ nhớ rồi sau đó đọc ra để truy xuất các giá trị dữ liệu và làm việc với chúng.

Khi khối lượng công việc giao cho máy tính ngày càng nên nhiều và phức tạp thì tất cả các câu lệnh không thể được đưa vào một chương trình, chúng cần được chia ra thành một số chương trình nhỏ hơn. Tất cả các chương trình này cuối cùng được tích hợp lại để chúng có thể làm việc với nhau. Một tập hợp các chương trình như thế được gọi là phần mềm.

Mối quan hệ giữa ba khái niệm câu lệnh, chương trình và phần mềm có thể được biểu diễn bằng sơ đồ trong hình 1.1:



Hình 1.1: Phần mềm, chương trình và câu lệnh

2.4 Sử dụng sự trợ giúp từ help file về cú pháp lệnh, về cú pháp hàm, các chương trình mẫu:

2.4.1 Cú pháp lệnh:

- **Statement:** Một Statement trong lập trình tương ứng với một hành động, một lệnh. Nó phải được kết thúc bởi dấu chấm phẩy (**;**).
- **Preprocessor Directive:** "#include" được gọi là một *chỉ thị tiền xử lý* và không phải là một Statement. Một chỉ thị tiền xử lý bắt đầu với dấu thăng (#), nó được xử lý trước khi biên dịch chương trình. Một tiền xử lý KHÔNG kết thúc bởi dấu chấm phẩy.
- **Block:** Một khối nhóm các Statement được đặt trong dấu ngoặc kép { }. Như ví dụ Hello.c có một block chứa phần thân của hàm main(). Không cần phải đặt dấu chấm phẩy khi kết thúc một block.
- **Comments:** Được gọi là chú thích. Comments KHÔNG thực thi và được bỏ qua bởi trình biên dịch. Các comments nhằm cung cấp những giải thích hữu ích về chương trình.

- **Whitespaces:** Blank, tab và newline (\n) được gọi chung là whitespaces (khoảng trắng). Các whitespaces lớn sẽ được bỏ qua, nghĩa là chỉ cần một whitespaces để phân cách các thẻ. Nhưng whitespaces giúp chương trình bạn trở nên sáng sủa và mạch lạc hơn.
- **Case Sensitivity:** Ngôn ngữ C phân biệt các trường hợp cụ thể. Một ROSE không phải là Rose và cũng không phải là rose. Điều này có nghĩa là trong C phân biệt chữ thường và chữ hoa.

2.4.2 Cú pháp hàm:

Một hàm C phải bao gồm một kiểu trả về (kiểu đó trả về `void` nếu không có giá trị trả về), một tên xác định, một danh sách các tham số để trong ngoặc đơn (nếu danh sách này không có tham số nào thì ghi là `void` bên trong dấu ngoặc), sau đó là khối các câu lệnh (hay khối mã) và/hay các câu lệnh `return`. (Nếu kiểu trả về là `void` thì mệnh đề này không bắt buộc phải có. Ngược lại, cũng không bắt buộc chỉ có một câu lệnh `return` mà tùy theo kỹ thuật, người lập trình có thể dẫn dòng mã sao cho mọi hướng chẻ nhánh đều được trả về đúng kiểu.)

```
<kiểu_trả_về> tên_hàm(<danh sách tham số>)
{
  <các_câu_lệnh>
  return <biến (hay giá trị) có kiểu là kiểu_trả_về>;
}
```

Trong đó, `<danh sách tham số>` của `N` biến thì được khai báo như là kiểu dữ liệu và tách rời nhau bởi dấu phẩy `,`:

```
<kiểu_dữ_liệu> var1, var2,..., varN;
```

Toàn bộ danh sách này được đặt trong ngoặc đơn ngay sau tên_hàm.

Ví dụ:

Hàm `add` tính tổng hai số có kiểu **integer**, hàm `abs` tính trị tuyệt đối của số có kiểu integer, và chương trình (hàm main) hiển thị hai dòng `1 + 1 = 2` và `absolute value of-2 is 2`

```
#include <stdio.h>; //Chú giải: dòng này khai báo thư viện là stdio.h
int add(int x, int y)
{
        return x + y;

}
int abs(int x)
{
        if (x > 0) return x;
        if (x < 0) return -x;
        if (x == 0) return 0;
```

```
/* đây chỉ là thí dụ cho thấy C có khả năng dùng nhiều hơn 1 câu lệnh
<code>return</code> hoàn toàn có thể dùng các câu lệnh khác đơn giản hơn.*/
 }
 int main(void)
 {
        {
            int z;
            int y;
            printf("nhap z:");
            scanf("%d",&z);
            printf("nhap y:");
            scanf("%d",&y);
            printf("%d + %d = %d\n", z, y,add(z, y));
            printf ("gia tri tuyet doi cua %d la %d", y, abs(y));
        }
}
```

2.4.3 Các chương trình mẫu:

Chương trình đếm số ký tự trong một chuỗi ASCII

```
#include <stdio.h>
#include <ctype.h>
#include<conio.h>
void main()
{
  char chuoi[80];
  int i = 0, count = 0;
  printf("\nNhap vao mot chuoi bat ky : ");
  gets(chuoi);
  while (chuoi[i] != 0)
  {
    if (isalpha(chuoi[i++]))
      count++;
  }
  printf("So ky tu trong chuoi = %d", count);
  getch();
}
```

Giải hệ PT bậc nhất

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

```
  printf("\nGiai phuong trinh bac nhat AX + B = 0");
  printf("\nCho biet ba he so A B : ");
  scanf("%f%f", &a, &b);
  if (a==0)
    if (b!=0)
      printf("Phuong trinh vo nghiem");
    else
      printf("Phuong trinh co nghiem khong xac dinh");
  else
    printf("Dap so cua phuong trinh tren = %f", -b/a);
  getch();
}
```

Biểu diễn số dưới dạng bit

```
#include <stdio.h>
#include <conio.h>
void main()
{
  unsigned int mang[24], i;
  int bit[16], k, index;
  printf("\nNhap vao 23 gia tri nguyen : ");
  for (i=0; i<23; i++)
    scanf("%d",&mang[i]);
  printf("       FEDCBA9876543210");
  for (i=0; i<23; i++)
  {
     k = mang[i];
     for (index = 0; index < 16; index++)
       bit[index] = 0;
     index = 15;
     while (k)
     { bit[index--] = k%2;       k /= 2;      }
     printf("\n%5d  ",mang[i]);
     for (index=0; index<16; index++)
       if (bit[index] == 1)
         printf("*");
       else
         printf("-");
  }
  getch();
}
```

CHƯƠNG 2. CÁC THÀNH PHẦN CƠ BẢN CỦA NGÔN NGỮ LẬP TRÌNH

Giới thiệu

Mỗi ngôn ngữ lập trình thường có ba thành phần cơ bản: bảng chữ cái, cú pháp và ngữ nghĩa.

- **Bảng chữ cái**: Là tập các kí hiệu dùng để viết chương trình.

+ Trong ngôn ngữ lập trình bảng chữ cái gồm: Các chữ cái trong bảng chữ cái tiếng Anh, các chữ số 0 -> 9 và một số kí tự đặc biệt.

- **Cú pháp**: là bộ qui tắc dùng để viết chương trình.
- **Ngữ nghĩa**: xác định ý nghĩa thao tác cần phải thực hiên, ứng với tổ hợp kí tự dựa vào ngữ cảnh của nó
- Cú pháp cho biết cách viết một chương trình hợp lệ, còn ngữ nghĩa xác định ý nghĩa của các tổ hợp kí tự trong chương trình.
- Lỗi cú pháp được chương trình dịch phát hiện và thông báo cho người lập chương trình biết, chỉ có các chương trình không còn lỗi cú pháp mới có thể được dịch sang ngôn ngữ máy.
- Lỗi ngữ nghĩa chỉ được phát hiện khi thực hiện chương trình trên dữ liệu cụ thể.

1. Mục tiêu:

- Trình bày và sử dụng được hệ thống kí hiệu và từ khóa;
- Mô tả được các kiểu dữ liệu;
- Trình bày được và vận dụng được các loại biến, hằng biểu thức cho từng chương trình cụ thể;
- So sánh được các lệnh, khối lệnh;
- Thực hiện được việc chạy chương trình;
- Thực hiện các thao tác an toàn với máy tính.

2. Nội dung:

2.1. Hệ thống từ khóa và kí hiệu được dùng trong ngôn ngữ lập trình:

Trước hết dạng chương trình duy nhất mà máy tính có thể thực thi trực tiếp là ngôn ngữ máy hay mã máy. Nó có dạng dãy các số nhị phân, thường được ghép nhóm thành byte 8 bit cho các hệ xử lý 8/16/32/64 bit. Nội dung byte thường biểu diễn bằng đôi số hex. Để có được bộ mã này ngày nay người ta dùng ngôn ngữ lập trình để viết ra chương trình ở dạng văn bản và dùng trình dịch để chuyển sang mã máy .

Khi kỹ thuật điện toán ra đời chưa có ngôn ngữ lập trình dạng đại diện nào, thì phải lập trình trực tiếp bằng mã máy. Dãy byte viết ra được đục lỗ lên phiếu đục lỗ (punched card) và nhập qua máy đọc phiếu tới máy tính. Sau đó chương trình có thể được ghi vào băng/đĩa từ để sau này nhập nhanh vào máy tính. Ngôn ngữ máy được gọi là "ngôn ngữ lập trình thế hệ 1" (1GL, first-generation programming languages).

Sau đó các mã lệnh được thay thế bằng các tên gợi nhớ và lập trình được ở dạng văn bản (text) rồi dịch sang mã máy. Hợp ngữ (assembly languages) ra đời, là "ngôn ngữ lập trình thế hệ 2" (2GL, second-generation programming languages). Lập trình thuận lợi hơn, khi dịch có thể liên kết với thư viện chương trình con ở cả dạng macro (đoạn chưa dịch) và lẫn mã đã dịch. Hợp ngữ hiện được dùng là ngôn ngữ bậc thấp (low-

level programming languages) để tinh chỉnh ngôn ngữ bậc cao thực hiện truy nhập trực tiếp phần cứng cụ thể trong việc lập trình hệ thống, tạo các hiệu ứng đặc biệt cho chương trình.

Ngôn ngữ bậc cao (high-level programming languages) hay "ngôn ngữ lập trình thế hệ 3" (3GL, third-generation programming languages) ra đời vào những năm 1950. Đây là các ngôn ngữ hình thức, dùng trong lập trình máy điện toán và không lệ thuộc vào hệ máy tính cụ thể nào. Nó giải phóng người lập trình ứng dụng làm việc trong hệ điều hành xác định mà không phải quan tâm đến phần cứng cụ thể. Các ngôn ngữ được phát triển liên tục với các dạng và biến thể mới, theo bước phát triển của kỹ thuật điện toán.

Đối với ngôn ngữ bậc cao thì định nghĩa ngôn ngữ lập trình theo [Loud 94], T.3 là: Ngôn ngữ lập trình là một hệ thống được ký hiệu hóa để miêu tả những tính toán (qua máy tính) trong một dạng mà cả con người và máy đều có thể đọc và hiểu được.

Theo định nghĩa ở trên thì một ngôn ngữ lập trình phải thỏa mãn được hai điều kiện cơ bản sau:

Dễ hiểu và dễ sử dụng đối với người lập trình, để có thể dùng để giải quyết nhiều bài toán khác nhau.

Miêu tả một cách đầy đủ và rõ ràng các tiến trình (tiếng Anh: process), để chạy được trên các hệ máy tính khác nhau.

Một tập hợp các chỉ thị được biểu thị qua ngôn ngữ lập trình nhằm mục đích thực hiện các thao tác máy tính nào đó được gọi là một chương trình. Khái niệm này còn có những tên khác như chương trình máy tính hay chương trình điện toán.

Lưu ý: chương trình được viết cho máy vi tính thường được gọi là phần mềm máy tính. Ví dụ: chương trình Microsoft Word là một cách gọi chung chung; cách gọi phần mềm Microsoft Word chỉ rõ hơn nó là một chương trình ứng dụng.

Khái niệm lập trình dùng để chỉ quá trình con người tạo ra chương trình máy tính thông qua ngôn ngữ lập trình. Người ta còn gọi đó là quá trình mã hoá thông tin tự nhiên thành ngôn ngữ máy. Từ viết mã cũng được dùng trong nhiều trường hợp để chỉ cùng một ý.

Như vậy, theo định nghĩa, mỗi ngôn ngữ lập trình cũng chính là một chương trình, nhưng nó có thể được dùng để tạo nên các chương trình khác. Văn bản được viết bằng ngôn ngữ lập trình để tạo nên chương trình được gọi là mã nguồn.

Thao tác chuyển đổi từ mã nguồn thành chuỗi các chỉ thị máy tính được thực hiện tương tự như việc chuyển đổi qua lại giữa các ngôn ngữ tự nhiên của con người. Các thao tác này gọi là biên dịch, hay ngắn gọn hơn là dịch. Nếu quá trình dịch diễn ra đồng thời với quá trình thực thi, ta gọi đó là thông dịch; nếu diễn ra trước, ta gọi đó là biên dịch. Phần mềm dịch tương ứng được gọi là phần mềm thông dịch và phần mềm biên dịch.

Một phần mềm thông dịch là một phần mềm có khả năng đọc, chuyển mã nguồn của một ngôn ngữ và ra lệnh cho máy tính tiến hành các tính toán dựa theo cú pháp của ngôn ngữ.

Một phần mềm biên dịch hay ngắn gọn hơn trình biên dịch là phần mềm có khả năng chuyển mã nguồn của một ngôn ngữ ban đầu sang dạng mã mới thường là một ngôn ngữ cấp thấp hơn.

Ngôn ngữ cấp thấp nhất là một chuỗi các chỉ thị máy tính mà có thể được thực thi trực tiếp bởi máy tính (thông qua các theo tác trên vùng nhớ). Trước đây, hầu hết các trình biên dịch cũ phải dịch từ mã nguồn sang bộ mã phụ (các tệp có dang *.obj) rồi mới tạo ra tập tin thực thi. Ngày nay, hầu hết các trình biên dịch đều có khả năng dịch mã nguồn trực tiếp thành các tập tin thực thi hay thành các dạng mã khác thấp hơn, tuỳ theo yêu cầu của người lập trình.

Điểm khác nhau giữa thông dịch và biên dịch là: trình thông dịch dịch từng câu lệnh theo yêu cầu thực thi và chương trình đích vừa tạo ra sẽ không được lưu lại; trong khi đó, trình biên dịch sẽ dịch toàn bộ chương trình, cho ra chương trình đích được lưu lại trong máy tính rồi mới thực hiện chương trình.

Một chương trình máy tính có thể được thực thi bằng cách biên dịch, thông dịch, hoặc phối hợp cả hai.

Để đạt được yêu cầu về độ chính xác và tính hiệu quả, mã viết ra nhiều khi khó đọc ngay cả với chính người viết ra mã đó, chưa kể tới người khác. Chính vì lý do đó, mọi tài liệu, hướng dẫn lập trình đều khuyên nên thêm các chú giải vào mã nguồn trong quá trình viết. Các chú giải giúp người khác rất nhiều trong việc đọc hiểu mã nguồn; đối với chương trình phức tạp, chú giải là thành phần vô cùng quan trọng trong mã nguồn.

Đặc điểm chung của ngôn ngữ lập trình

Mỗi ngôn ngữ lập trình có thể được xem như là một tập hợp của các chi tiết kỹ thuật chú trọng đến cú pháp, từ vựng, và ý nghĩa của ngôn ngữ.

Những chi tiết kỹ thuật này thường bao gồm:

- Dữ liệu và cấu trúc dữ liệu
- Câu lệnh và dòng điều khiển
- Các tên và các tham số

Các cơ chế tham khảo và sự tái sử dụng

Đối với các ngôn ngữ phổ biến hoặc có lịch sử lâu dài, người ta thường tổ chức các hội thảo chuẩn hoá nhằm tạo ra và công bố các tiêu chuẩn chính thức cho ngôn ngữ đó, cũng như thảo luận về việc mở rộng, bổ sung cho các tiêu chuẩn trước đó. Ví dụ: Với ngôn ngữ C++, hội đồng tiêu chuẩn ANSI C++ và ISO C++ đã tổ chức đến 13 cuộc hội thảo để điều chỉnh và nâng cấp ngôn ngữ này. (Xem thêm Comeau.Computing). Đối với các ngôn ngữ lập trình web như JavaScript, ta có chuẩn ECMA, W3C.

Kiểu dữ liệu:

Một hệ thống đặc thù mà theo đó các dữ liệu được tổ chức sắp xếp trong một chương trình gọi là hệ thống kiểu của ngôn ngữ lập trình. Việc thiết kế và nghiên cứu các hệ thống kiểu được biết như là lý thuyết kiểu.

Nhiều ngôn ngữ định nghĩa sẵn các kiểu dữ liệu thông dụng như:

- integer: rất thông dụng, được dùng để biểu diễn các số nguyên.
- char: biểu diễn các ký tự đơn lẻ.

- string: biểu diễn chuỗi các ký tự, hay còn gọi là chuỗi, để tạo thành câu hay cụm từ.

Ví dụ: trong C/C++, kiểu số nguyên thông dụng có tên là int và chiếm 4 byte trong hầu hết trình dịch 32-bit; kiểu chuỗi là một dãy các char, với ký tự NULL (hay '\0') ở vị trí chuỗi kết thúc – dãy có thể dài hơn chuỗi nó lưu trữ.

Ngôn ngữ có kiểu tĩnh là ngôn ngữ xác định trước kiểu cho tất cả dữ liệu được khai báo trong mã nguồn tại thời điểm dịch. Các giá trị của biến chỉ có thể ở một/một số kiểu cụ thể nào đó và ta chỉ có thể thực hiện một số thao tác nhất định trên chúng.

Ví dụ: trong C, ta không thể dùng phép tính + trên kiểu dữ liệu string (tức là char * hay char []). Hầu hết các ngôn ngữ có kiểu tĩnh thông dụng như C, C++, Java, D, Delphi, và C# đều đòi hỏi người lập trình kê khai rõ ràng kiểu của dữ liệu. Những người ủng hộ việc này cho rằng nó sẽ giúp ngôn ngữ rõ ràng hơn.

Các ngôn ngữ có kiểu tĩnh lại được chia ra thành hai loại:

Ngôn ngữ kiểu khai báo, tức là sự thông báo của biến và hàm đều được khai báo riêng về kiểu của nó.

Ví dụ điển hình của loại này là Pascal, Java, C, hay C++.

Còn lại là ngôn ngữ loại suy đoán kiểu. Trong đó các biến và hàm có thể không cần được khai báo từ trước. Linux BASH và PHP là hai ví dụ trong những kiểu này. Suy đoán kiểu là một cơ chế mà ở đó các đặc tả về kiểu thường có thể bị loại bỏ hoàn toàn nếu có thể được, nhằm giúp cho trình dịch dễ dàng tự đoán các kiểu của các giá trị từ ngữ cảnh mà các giá trị đó được sử dụng. Ví dụ một biến được gán giá trị 1 thì trình dịch loại suy đoán kiểu không cần khai báo riêng rằng đó là một kiểu integer. Các ngôn ngữ suy đoán kiểu linh hoạt hơn trong sử dụng, đặc biệt khi chúng lắp đặt sự đa dạng hoá các tham số. Ví dụ của ngôn ngữ loại này là Haskell, MUMPS và ML.

Các ngôn ngữ có kiểu động là ngôn ngữ mà các kiểu chỉ được gán lên các dữ liệu trong thời gian chương trình được thực thi. Điều này có mặt lợi là người lập trình không cần phải xác định kiểu dữ liệu nào hết, đồng thời có thêm lợi thế là có thể gán nhiều hơn một kiểu dữ liệu lên các biến. Tuy nhiên, vì ngôn ngữ có kiểu động xem tất cả các vai trò của dữ liệu trong chương trình là có thể chuyển hóa được, do vậy các phép toán không đúng (như là cộng các tên, hay là xếp thứ tự các số theo thứ tự đánh vần) sẽ không tạo ra các lỗi cho đến lúc nó được thi hành—mặc dù vẫn có một số cài đặt cung cấp vài dạng kiểm soát tĩnh cho các lỗi hiển nhiên.

Ví dụ của các ngôn ngữ này là Objective-C, Lisp, JavaScript, Tcl, Prolog, Python và Ruby.

Các ngôn ngữ có kiểu mạnh không cho phép dùng các giá trị của kiểu này như là một kiểu khác. Chúng rất chặt chẽ trong việc phát hiện sự dùng sai kiểu. Việc phát hiện này sẽ xảy ra ở thời gian thi hành (run-time) đối với các ngôn ngữ có kiểu động và xảy ra ở thời gian dịch đối với các ngôn ngữ có kiểu tĩnh.

ADA, Java, ML và Oberon là các ví dụ của ngôn ngữ có kiểu mạnh. Ngược lại, ngôn ngữ có kiểu yếu không quá khắt khe trong các quy tắc về kiểu hoặc cho phép một cơ chế tường minh để xử lý các vi phạm. Thường nó cho phép hành xử các biểu

hiện chưa được định nghĩa trước, các vi phạm về sự phân đoạn (segmentation), hay là các biểu hiện không an toàn khác khi mà các kiểu bị gán giá trị một cách không đúng.

C, ASM, C++, Tcl và Lua là các ví dụ của ngôn ngữ có kiểu yếu.

Lưu ý: Các khái niệm về kiểu mạnh hay yếu có tính tương đối. Java là ngôn ngữ có kiểu mạnh đối với C nhưng yếu đối với ML. Tùy theo cách nhìn mà các khái niệm đó được dùng, nó tương tự như việc xem ngôn ngữ ASM là ở cấp thấp hơn ngôn ngữ C; trong khi Java lại là ngôn ngữ ở mức cao hơn C.

Hai khái niệm tĩnh và mạnh cũng không đối lập nhau. Java là ngôn ngữ có kiểu mạnh và tĩnh. C là ngôn ngữ có kiểu yếu và tĩnh. Trong khi đó, Python là ngôn ngữ có kiểu mạnh và động. Tcl lại là ngôn ngữ có kiểu yếu và động. Cũng nên biết trước rằng có nhiều người đã dùng sai các khái niệm trên và cho rằng kiểu mạnh là kiểu tĩnh cộng với mạnh. Lầm lẫn hơn, họ còn cho rằng ngôn ngữ C có kiểu mạnh mặc dù rằng C không hề bắt nhiều loại lỗi về việc dùng sai kiểu. Cấu trúc dữ liệu.

Hầu hết các ngôn ngữ đều cung cấp các cách thức để lắp ráp các cấu trúc dữ liệu phức tạp từ các kiểu sẵn có và để liên kết các tên với các kiểu mới kết hợp (dùng các kiểu mảng, danh sách, hàng đợi, ngăn xếp hay tập tin).

Các ngôn ngữ hướng đối tượng cho phép lập trình viên định nghĩa các kiểu dữ liệu mới gọi là đối tượng. trong nội bộ các đối tượng đó có riêng các hàm và các biến (và thường được gọi theo thứ tự là các phương thức và các thuộc tính). Một chương trình có định nghĩa các đối tượng sẽ cho phép các đối tượng đó thực thi như là các chương trình con độc lập nhưng lại tương tác nhau. Các tương tác này có thể được thiết kế trong lúc viết mã để mô hình hóa và mô phỏng theo đời sống thật của các đối tượng. Nói một cách đơn giản, các ngôn ngữ hướng đối tượng đã được cho thêm sức sống để có riêng những tính năng hoạt động và tương tác với thế giới bên ngoài. Ngoài ra, các đối tượng còn có thêm các đặc tính như là thừa kế và đa hình. Điều này là một ưu thế trong việc dùng ngôn ngữ loại này để mô tả các đối tượng của thế giới thực.

Các mệnh lệnh và dòng điều khiển:

Khi dữ liệu đã được định rõ, máy tính phải được chỉ thị làm thế nào để tiến hành các phép toán trên dữ liệu đó. Những mệnh đề cơ bản có thể được cấu trúc thông qua việc sử dụng các từ khóa (đã được định nghĩa bởi ngôn ngữ lập trình) hoặc là có thể tạo thành từ việc dùng và kết hợp các cấu trúc ngữ pháp hay cú pháp đã được định nghĩa. Những mệnh đề cơ bản này gọi là các câu lệnh.

Tùy theo ngôn ngữ, các câu lệnh có thể được kết hợp với nhau theo trật tự nào đó. Điều này cho phép thiết lập được các chương trình thực hiện được nhiều chức năng. Xa hơn, ngoài các câu lệnh để thay đổi và điều chỉnh dữ liệu, còn có những kiểu câu lệnh dùng để điều khiển dòng xử lý của máy tính như là phân nhánh, định nghĩa bởi nhiều trường hợp, vòng lặp, hay kết hợp các chức năng. Đây là các thành tố không thể thiếu của một ngôn ngữ lập trình.

Các tên và các tham số:

Muốn cho chương trình thi hành được thì phải có phương pháp xác định được các vùng trống của bộ nhớ để làm kho chứa dữ liệu. Phương pháp được biết nhiều nhất là thông qua tên của các biến. Tùy theo ngôn ngữ, các vùng trống gián tiếp có thể bao

gồm các tham chiếu, mà thật ra, chúng là các con trỏ (pointer) chỉ đến những vùng chứa khác của bộ nhớ, được cài đặt trong các biến hay nhóm các biến. Phương pháp này gọi là đặt tên kho nhớ. Tương tự với phương pháp đặt tên kho nhớ, là phương pháp đặt tên những nhóm của các chỉ thị. Trong hầu hết các ngôn ngữ lập trình, đều có cho phép gọi đến các macro hay các chương trình con như là các câu lệnh để thi hành nội dung mô tả trong các macro hay chương trình con này thông qua tên. Việc dùng tên như thế này cho phép các chương trình đạt tới một sự linh hoạt cao và có giá trị lớn trong việc tái sử dụng mã nguồn (vì người viết mã không cần phải lặp lại những đoạn mã giống nhau mà chỉ việc định nghĩa các macro hay các chương trình con.)

Các tham chiếu gián tiếp đến các chương trình khả dụng hay các bộ phận dữ liệu đã được xác định từ trước cho phép nhiều ngôn ngữ định hướng ứng dụng tích hợp được các thao tác khác nhau.

Cơ chế tham khảo và việc tái sử dụng mã nguồn

Mỗi ngôn ngữ lập trình đều có một bộ các cú pháp quy định việc lập trình sao cho mã nguồn được thực thi. Theo đó, mỗi nhà sản xuất ngôn ngữ lập trình sẽ cung cấp một bộ các cấu trúc ngữ pháp cho các câu lệnh, một khối lượng lớn các từ vựng quy ước được định nghĩa từ trước, và một số lượng các thủ tục hay hàm cơ bản. Ngoài ra, để giúp lập trình viên dễ dàng sử dụng, nhà sản xuất còn phải cung cấp các tài liệu tra cứu về đặc tính của ngôn ngữ mà họ phát hành. Những tài liệu tra cứu này bao gồm hầu hết các đặc tả, tính chất, các tên (hay từ khoá) mặc định, phương pháp sử dụng, và nhiều khi là các mã nguồn để làm ví dụ. Do sự không thống nhất trong các ý kiến về việc thiết kế và sử dụng từng ngôn ngữ nên có thể xảy ra trường hợp mã nguồn của cùng một ngôn ngữ chạy được cho phần mềm dịch này nhưng không tương thích được với phần mềm dịch khác. Ví dụ là các mã nguồn C viết cho Microsoft C (phiên bản 6.0) có thể không chạy được khi dùng trình dịch Borland (phiên bản 4.5) nếu không biết cách thức điều chỉnh. Đây cũng là nguyên do của các kỳ hội nghị chuẩn hóa ngôn ngữ lập trình. Ngoài công việc chính là phát triển ngôn ngữ đặc thù, hội nghị còn tìm cách thống nhất hóa ngôn ngữ bằng cách đưa ra các tiêu chuẩn, các khuyến cáo thay đổi về ngôn ngữ trong tương lai hay các đổi mới về cú pháp của ngôn ngữ.

Những đổi mới về tiêu chuẩn của một ngôn ngữ mặt khác lại có thể gây ra các hiệu ứng phụ. Đó là việc mã nguồn của một ngôn ngữ dùng trong phiên bản cũ không tương thích được với phần mềm dịch dùng tiêu chuẩn mới hơn. Đây cũng là một việc cần lưu tâm cho những người lập trình. Trường hợp điển hình nhất là việc thay đổi phiên bản về ngôn ngữ Visual Basic của Microsoft. Các mã nguồn của phiên bản 6.0 có thể sẽ không dịch được nếu dùng phiên bản mới hơn. Lý do là nhà thiết kế đã thay đổi kiến trúc của VisualBasic để nâng cao và cung cấp thêm các chức năng mới về lập trình theo định hướng đối tượng cho ngôn ngữ này.

Thay vào việc tái sử dụng mã nguồn thì cũng có các hướng phát triển khác nhằm tiết kiệm công sức cho người lập trình mà hai hướng chính là:

Việc ra đời của các bytecode mà điển hình là ngôn ngữ Java. Với Java thì mã nguồn sẽ được dịch thành một ngôn ngữ trung gian khác gọi là bytecode. Mã của bytecode một lần nữa sẽ được phần mềm thông dịch thực thi, phần mềm này gọi là

máy ảo. Các máy ảo được cài đặt sẵn trên các máy tính và được cung cấp miễn phí. Tùy theo hệ điều hành mà có thể cài đặt máy ảo thích hợp. Do đó, cùng một nguồn Java bytecode có thể chạy trong bất cứ hệ điều hành nào miễn là hệ điều hành đó có cài đặt sẵn máy ảo Java. Việc này tiết kiệm rất nhiều công sức cho lập trình viên vì họ không phải viết mã Java khác nhau cho mỗi hệ điều hành.

Tận dụng tính chất thừa kế của các lớp (class) trong các ngôn ngữ hướng đối tượng. Theo kiểu thiết kế này, một đối tượng có thể thụ hưởng các đặc tính mà các thế hệ trước của chúng đã có. Do đó, khi phát triển phần mềm mới theo cấu trúc của các lớp, người ta chỉ cần tạo thêm các lớp con (subclass) có nhiều tính năng mới hơn. Điều này giúp giảm bớt công sức vì không phải phát triển lại từ đầu. (Lưu ý: Java cũng là một loại ngôn ngữ hướng đối tượng nên nó có luôn ưu thế này.)

Triết lý của các thiết kế

Tùy theo mục đích của ngôn ngữ mà chúng được thiết kế để tạo điều kiện giải quyết những vấn đề mà ngôn ngữ đó hướng tới. Những chức năng này làm cho một ngôn ngữ có thể tiện lợi để dùng phát triển loại phần mềm này nhưng có thể khó để phát triển loại phần mềm khác.

Hầu hết các ngôn ngữ đòi hỏi sự chính xác cao về mặt cú pháp. Các ngôn ngữ không cho phép có lỗi. Mặc dù vậy, một số ít ngôn ngữ cũng cho phép tự điều chỉnh trong một mức độ khá cao, khi đó chương trình tự viết lại để xử lý những trường hợp mới. Các ngôn ngữ như Prolog, PostScript và các thành viên trong họ ngôn ngữ Lisp có khả năng này. Trong ngôn ngữ MUMPS, kỹ thuật này gọi là tái biên dịch động. Các phần mềm mô phỏng và nhiều máy ảo (virtual machine) khai thác kỹ thuật này để có hiệu suất cao.

Một yếu tố liên quan đến triết lý thiết kế là có một số ngôn ngữ vì muốn tạo sự dễ dàng cho người mới dùng, đã không phân biệt việc viết chữ hoa hay không. Pascal và Basic là hai ngôn ngữ không phân biệt việc một ký tự có viết hoa hay không, trái lại trong C/C++, Java, PHP, Perl, BASH đều bắt buộc phải bảo đảm việc viết đúng y hệt như lúc khai báo cho các tên.

Các dạng câu lệnh:

Câu lệnh là một thành tố quan trọng nhất của mọi ngôn ngữ lập trình. Tùy theo ngôn ngữ các câu lệnh đều phải tuân theo các trật tự sắp xếp của các từ khóa, tham số, biếnvà các định danh khác như các macro, hàm, thủ tục cũng như các quy ước khác. Tập hợp trật tự và quy tắc đó tạo thành cú pháp của ngôn ngữ lập trình. Các dạng câu lệnh bao gồm Định nghĩa: Dạng câu lệnh này cho phép xác định một kiểu dữ liệu mới hay một hằng. Lưu ý là trong các ngôn ngữ lập trình định hướng đối tượng thì mỗi lớp đều có thể là một kiểu dữ liệu mới do đó việc tạo ra một lớp mới tức là đã dùng câu lệnh kiểu định nghĩa.

Ví dụ: Trong C hay C++, câu lệnh #define PI 3.1415927 sẽ cho phép định nghĩa tên (macro) PI với giá trị không đổi là 3,1415927.

Khai báo: Cũng gần giống như dạng định nghĩa, dạng khai báo cho phép người lập trình chính thức thông báo về sự ra đời của một biến, hay một tên (tên hàm chẳng hạn). Thông thường, đối với ngôn ngữ tĩnh, tên hàm hay biến mới đều phải có phần cho biết

kiểu dữ liệu của biến hay hàm. Tuy nhiên, điều này không bắt buộc với ngôn ngữ động. Ngoài ra, các khai báo đôi khi còn cho phép các biến gán một giá trị ban đầu nhưng thường thì việc này cũng không bắt buộc. (Xem thêm loại câu lệnh gán giá trị). Đối với nhiều ngôn ngữ thì việc khai báo có thể cho phép chương trình đó được cấp thêm một phần bộ nhớ dự trữ riêng cho các biến (hay các đối tượng) đăng ký tên trong câu lệnh khai báo.

Ví dụ:

Trong Java hay C/C++, câu lệnh int line_number = 0; thuộc loại khai báo

Trong Perl hay PHP, câu lệnh $my_var; thuộc loại khai báo

Gán giá trị là loại câu lệnh cho phép viết giá trị cụ thể vào các biến. Có thể có các giới hạn khác nhau trong việc gán giá trị này (chẳng hạn như phải tương thích về kiểu dữ liệu hay trường hợp nếu biến có các kiểu đặc biệt thì phải dùng đến các hàm hay các thủ tục để gán giá trị cho chúng).

Ví dụ:

Trong ASM, câu lệnh mov AX, 21h sẽ gán giá trị 21h lên thanh AX

Trong Java hay C/C++, câu lệnh i=j; sẽ gán giá trị đang có của biến j cho biến i

Kết hợp: Hầu hết các ngôn ngữ đều cho phép thiết lập câu lệnh mới từ nhiều câu lệnh. Lưu ý: Cần dựa theo cú pháp của từng ngôn ngữ để làm việc này.

Ví dụ:

Trong văn lệnh BASH hai câu lệnh xóa các tệp có đuôi txt rm -f *.txt và câu lệnh mkdir newfolder tạo một thư mục trống có tên 'newfolder' có thể được ghép nhau thành dãy câu có dạng rm -f *.txt; mkdir newfolder. Thứ tự thực hiện các câu lệnh thành phần sẽ đi từ trái sang phải.

Điều kiện: Loại câu lệnh này dùng để chẻ nhánh dòng điều khiển của ngôn ngữ. Thường từ khóa hay được dùng nhất là "if", "else", và "else if". Ngoài ra, một số ngôn ngữ có thể dùng thêm dạng câu lệnh phân nhánh đặc biệt cho trường hợp có nhiều phân nhánh (thường từ khóa bắt đầu câu lệnh điều kiện kiểu này có thể là "switch" hay là "case".)

Ví dụ: Trong Java hay C/C++, câu lệnh

```
if (x==1) { y = x; }
else { y = x + 3; }
```

là loại câu lệnh điều kiện

Vòng lặp: Dùng để lặp lại các câu lệnh giống nhau cho các đối tượng hay các biến trong một số hữu hạn lần. Từ khóa thường gặp nhất trong các ngôn ngữ là "for" và "while".

Ví dụ: Trong Java hay C/C++, câu lệnh

```
for (int n=1; n!=5; ++n) { value *= n }
```

sẽ lần lượt tính giá trị value = value * n làm 4 lần với các giá trị của biến n lần lượt là 1,2,3,4. Giá trị sau cùng nhận về của value sẽ là (value * 4!).

Gọi loại lệnh này dùng để thực thi các hàm, các thủ tục, hay các macro đã được định nghĩa sẵn bởi các thư viện hay bởi người lập trình.

Ví dụ: Trong C/C++, câu lệnh printf("Hello, world!\n");

gọi hàm cho sẵn nhằm hiển thị dòng chữ "Hello, world!<dấu xuống hàng>"

Các định hướng dịch hay còn gọi là các chỉ thị tiền xử lý: Ngoài các thành tố trên, các nhà sản xuất các phần mềm dịch (đặc biệt là các trình dịch) còn có thể cung cấp thêm các dạng câu lệnh không trực tiếp tham gia vào việc tính toán trên các dữ liệu của chương trình nhưng lại trực tiếp điều khiển các dòng chuyển dịch mã ở thời điểm dịch cũng như là hướng dẫn các trình dịch cách xử lý, tìm nguồn mã bổ sung, cách dùng thư viện, hay các cài đặt đặc biệt cho một loại hệ điều hành hay cho một loại phần cứng nào đó. Các câu lệnh này có thể tùy thuộc vào nhà sản xuất phần mềm chuyển dịch cung cấp.

Ví dụ: Trong C/C++ các câu lệnh

#ifndef MY_LIB

#include "my_code.h"

#endif

sẽ kiểm tra nếu tên MY_LIB chưa được định nghĩa trước đây trong chương trình thì sẽ tiếp tục đọc tệp my_code.h (để nhận vào các định nghĩa, hay các khai báo có trong tệp my_code.h rồi tiếp tục dịch mã.)

Chú giải Các câu lệnh loại này không tham gia vào bất kỳ hoạt động nào trong quá trình dịch nghĩa là các phần mềm dịch sẽ bỏ qua các dòng này. Tuy nhiên, các câu lệnh loại chú giải có giá trị văn bản. Người ta thường dùng chúng để ghi chú các kỹ thuật, các tính năng hay những điều cần nhớ để sau này khi đọc lại mã nguồn thì có thể hiểu được người lập trình đã làm gì.

Thí du: Trong Java, C/C++, PHP các câu chú giải có thể bắt đầu bởi dấu "//"

//hàm "SUM(n,r,m)" dùng để tính tổng số tiền có được khi gửi ngân hàng

// n=số tháng, r=lãi suất trong năm, m=vốn gửi ban đầu sẽ là hai câu lệnh chú giải.

Lưu ý: để hiểu rõ hơn và sử dụng thuần thục các dạng câu lệnh thì người lập trình nên tham khảo các tài liệu giảng giải riêng về từng ngôn ngữ.

Chương trình con và macro

Một chương trình con (còn được gọi là hàm, thủ tục, hay thủ tục con) là một chuỗi mã để thực thi một thao tác đặc thù nào đó như là một phần của chương trình lớn hơn. Đây là các câu lệnh được nhóm vào một khối và được đặt tên và tên này tùy theo ngôn ngữ có thể được gán cho một kiểu dữ liệu. Những khối mã này có thể được tập trung lại làm thành các thư viện phần mềm. Các chương trình con có thể được gọi ra để thi hành (thường là qua tên của chương trình con đó). Điều này cho phép các chương trình dùng tới những chương trình con nhiều lần mà không cần phải lặp lại các khối mã giống nhau một khi đã hoàn tất việc viết mã cho các chương trình con đó chỉ một lần.

Trong một số ngôn ngữ, người ta lại phân biệt thành 2 kiểu chương trình con: Hàm (function) dùng để chỉ các chương trình con nào có giá trị trả về (trong một kiểu dữ liệu nào đó) thông qua tên của hàm.

Thủ tục (subroutine) dùng để mô tả các chương trình con được thi hành và không có giá trị trả về.

Tuy nhiên, trong nhiều ngôn ngữ khác như C chẳng hạn thì không có sự phân biệt này và chỉ có một khái niệm hàm. Để mô tả các hàm không trả về giá trị (tương đương với khái niệm thủ tục) thì người ta có thể gán cho kiểu dữ liệu của hàm đó là void.

Lưu ý: trong các ngôn ngữ hướng đối tượng, mỗi một đối tượng hay một thực thể (instance), tùy theo quan điểm, có thể được xem là một chương trình con hay một biến vì bản thân nội tại của thực thể đó có chứa các phương thức và cả các dữ liệu có thể trả lời cho các lệnh gọi từ bên ngoài. Macro được hiểu là tên viết tắt của một tập các câu lệnh. Như vậy, trong những chương trình có các khối câu lệnh giống nhau thì người ta có thể định nghĩa một macro cho khối đại diện và có thể dùng tên của macro này trong lúc viết mã thay vì phải viết cả khối câu lệnh mỗi lần khối này xuất hiện lặp lại. Một cách trừu tượng, thì macro là sự thay thế một dạng thức văn bản xác định bằng việc định nghĩa của một (hay một bộ) quy tắc. Trong quá trình dịch, các phần mềm dịch sẽ tự động thay các macro này trở lại bằng các mã mà nó viết tắt cho, rồi mới tiếp tục dịch. Như vậy, các mã này được điền trả lại trong thời gian dịch. Một số ngôn ngữ có thể cho các macro được phép khai báo và sử dụng tham số. Như vậy về vai trò macro giống hệt như các chương trình con.

Các điểm khác nhau quan trọng giữa một chương trình con và một macro bao gồm: Mã của chương trình con vẫn được dịch và để riêng ra. Cho tới khi một chương trình con được gọi ở thời điểm thi hành, thì các mã đã dịch sẵn của chương trình con này mới được lắp vào dòng chạy của chương trình.

Trong khi đó, sau khi dịch, các macro sẽ không còn tồn tại. Trong chương trình đã được dịch, tại các vị trí có tên của macro thì các tên này được thay thế bằng khối mã (đã dịch) mà nó đại diện.

Cách viết mã dùng chương trình con sau khi dịch xong sẽ tạo thành các tập tin ngắn hơn so với cách viết dùng macro. Ngược lại khi máy tính tải lên thì một phần mềm có cách dùng macro ít tốn tính toán của CPU hơn là phần mềm đó phát triển bằng phương pháp gọi các chương trình con.

Biến, hằng, tham số, và đối số

Một biến (variable) là một tên biểu thị cho một số lượng, một ký hiệu hay một đối tượng. Thêm vào đó, một biến sẽ được dành sẵn chỗ (phần của bộ nhớ) để chứa số lượng, ký hiệu hay đối tượng đó. Trong lúc chương trình được thi hành thì các biến của chương trình sẽ có thể thay đổi giá trị hoặc không thay đổi gì cả. Hơn nữa, một biến có thể bị thay đổi cả lượng bộ nhớ mà nó đang chiếm hữu (do người lập trình hay do phần mềm dịch ra lệnh). Trường hợp biến này không được gán giá trị hay có gán giá trị nhưng không được sử dụng vào các tính toán thì nó chỉ chiếm chỗ trong bộ nhớ một cách vô ích. Mỗi biến sẽ có tên của nó và có thể có kiểu xác định. Tùy theo ngôn ngữ, một biến có thể được khai báo ở vị trí nào đó trong mã nguồn và cũng tùy ngôn ngữ, tùy phần mềm dịch và cách thức lập trình mà một biến có thể được tạo nên (cùng với chỗ chứa) hay bị xóa bỏ tại một thời điểm nào đó trong lúc thực thi chương trình. Việc các biến bị xóa bỏ là để tiết kiệm bộ nhớ cũng như làm tốt hơn việc quản lý phần bộ nhớ mà đôi khi một chương trình chỉ được cấp bởi đăng ký với hệ điều hành.

Quá trình tồn tại của một biến gọi là đời sống của biến. Trong nhiều trường hợp đời sống của một biến chỉ xảy ra trong nội bộ một hàm, một thủ tục hay trong một khối mã.

Một hằng (constant) là một giá trị số hay ký hiệu được gán cho một tên xác định. Khác với biến, hằng không bao giờ thay đổi giá trị. Vì lý do tiện lợi trong việc viết mã, thường đời sống của một hằng lâu dài hơn một biến và có khi nó tồn tại trong suốt toàn bộ thời gian thi hành của chương trình. Trong nhiều trường hợp hằng có thể được xác định kiểu hay không. (C++ là ngôn ngữ cho phép có cả hai cách định nghĩa hằng có kiểu hay không có kiểu và câu lệnh để tạo ra hai loại này là hoàn toàn khác nhau). Nếu một biến hoàn toàn không thay đổi giá trị của nó trong mọi tình huống thì vai trò của biến này tương đương với một hằng.

Khác với biến, tham số (parameter) cũng là các tên được các chương trình con hay macro dùng để tính toán. Khi được gọi thì chương trình con, hay macro sẽ đòi hỏi các tên này phải được gán giá trị cụ thể trước khi tiến hành tính toán.

Các giá trị được gán lên cho các tham số để một chương trình con hay macro thi hành gọi là các đối số (argument). Một cách đơn giản, các đối số là các giá trị thông tin hay dữ liệu cung cấp cho các chương trình con hay macro trước khi tính toán.

Các tham số giống biến ở chỗ chúng thường có kiểu xác định. Bên trong chương trình con, hay macro, các tham số thường đóng vai trò của hằng nhưng trong nhiều trường hợp khác chúng vẫn có thể hoạt động như các biến và điều này cũng phụ thuộc vào các đặc tính của mỗi ngôn ngữ.

Nếu nhìn toàn bộ chương trình như một hàm lớn thì tham số của hàm này gọi là tham số của chương trình và các tham số của chương trình này có thể tương tác với các chương trình khác và ngược lại. Một cách đơn giản thì tham số là các dữ liệu truyền đi giữa các chương trình hay các hàm, thủ tục hay macro.

Từ vựng quy ước

Từ vựng quy ước là những dãy các ký tự hay ký hiệu (thường tạo thành các chữ có ý nghĩa) nối nhau và được một ngôn ngữ cho sử dụng như là tên, giá trị hay một luật nào đó. Người viết mã nên tránh sử dụng các từ quy ước này vào việc đặt tên (cho các biến, hàm, hay các đối tượng khác) để tránh không gây ra các lỗi dạng ambiguity (nghĩa là từ dùng có nhiều nghĩa khiến cho phần mềm dịch không biết phải chọn cách nào). Tuy nhiên, tuỳ theo từng trường hợp mà một tên mới đặt ra trùng với các tên đã quy định có được chấp nhận hay không và việc chấp nhận này sẽ có hiệu ứng phụ gì.

Thí dụ:

Trong C thì việc viết #define MYVALUE 10; thì dãy ký tự "#define" sẽ là một từ vựng quy ước (thuộc về câu lệnh dạng định nghĩa)

Trong C/C++ nếu dùng từ int để khai báo như là tên của một biến chẳng hạn như unsigned int; thì lập tức khai báo này sẽ bị trình dịch bắt lỗi.

Từ khóa:

```
#A is height B is radius
def cone (a,b):
```

```
formula=(3.14*.33*a)*(b*b)
return formula
```

Công cụ tô màu cú pháp (syntax highlighting) dùng màu sắc để giúp lập trình viên thấy nhiệm vụ của các từ khóa, số, và dòng chú thích (comment) trong mã nguồn. Chương trình này được viết trong ngôn ngữ Python, nó tính ra thể tích của hình nón.

Từ khóa trong ngôn ngữ lập trình là các từ hay ký hiệu mà đã được ngôn ngữ đó gán cho một ý nghĩa xác định. Người lập trình sẽ không được phép dùng lại các từ khóa này dưới một ý nghĩa khác. Thường các từ khóa này được ngôn ngữ xác định dùng trong các kiểu dữ liệu cơ bản hay trong các dòng điều khiển. Ví dụ một số từ khóa trong C và C++: auto, float, return, char, if else, static, void...

Các tên chuẩn hay tên cho trước

Ngoài các từ khóa, một ngôn ngữ lập trình còn có khối lượng khá lớn các tên đã được định nghĩa hay được gán cho các ý nghĩa chuyên biệt gọi là các tên chuẩn. Các tên này có thể được dùng lại cho một ý nghĩa khác tùy theo người viết mã. Trong nhiều trường hợp sẽ phải có một cơ chế gọi để phân biệt là người lập trình muốn ám chỉ các tên đã bị tái dụng này dưới ý nghĩa nguyên thủy hay dưới ý nghĩa mới. Thường các tên được phép định nghĩa lại nằm trong hai loại chính:

Các hàm hay thủ tục chuẩn.

Các biến toàn cục (global)

Ví dụ: Trong C thì sin là tên của một hàm tính giá trị sin (trong thư viện math.h) nhưng người lập trình hoàn toàn có thể định nghĩa lại hàm này để cho nó có chức năng khác.

Trong văn lệnh BASH thì biến toàn cục $PATH có thể được định nghĩa lại để dùng như là một biến địa phương.

Các ký hiệu: Trong mỗi ngôn ngữ đều cung cấp một hệ thống ký hiệu hay ký tự có ý nghĩa riêng. Tùy theo ngôn ngữ mà các ký hiệu này được phép định nghĩa lại hay không. Những ký hiệu được đùng trong hai trường hợp thường thấy nhất là:

Dùng để chỉ các phép toán.

Dùng trong cú pháp. Trường hợp này thì các ký hiệu này giữ vai trò tương tự như các dấu chấm câu trong các ngôn ngữ tự nhiên.

Ví dụ:

Trong C/C++/Java/PHP thì các dấu ký hiệu '+', '-', '*', '/', '=' được dùng trong các phép toán theo thứ tự là cộng, trừ, nhân, chia và phép toán gán giá trị.

Trong C thì các dấu '+', '-', '*', '/',... là không thể dùng lại cho ý nghĩa khác. Trong khi đó nếu dùng C++ thì người lập trình hoàn toàn có khả năng định nghĩa chúng lại thành những phép toán mới theo ý riêng và áp dụng cho các đối tượng mà người lập trình mong muốn (chẳng hạn như dùng phương pháp "quá tải toán tử").

Trong C, C++, PHP, Perl, Java và Pascal thì kết thúc các câu lệnh đơn giản thường bắt buộc phải dùng dấu ';'. Và điều này thì không nhất thiết nếu dùng văn lệnh BASH. Dấu ';' này giữ vai trò tương tự như dấu '.' trong Việt ngữ hay Anh ngữ. (Có điều là đại đa số các ngôn ngữ lập trình sẽ tuyệt đối không cho phép việc viết sai cú pháp.)

Các luật cấm và ngoại lệ

Mỗi ngôn ngữ, do hạn chế của môi trường và bản thân ngôn ngữ cũng như do mục tiêu sử dụng, có thể có một số luật cấm mà người lập trìnhkhông thể vi phạm. Những luật cấm này có thể có những cách xử lý khác nhau như là:

Nhiều ngôn ngữ cho phép dùng các câu lệnh đặc biệt để lập trình viên có toàn quyền xử lý lỗi và thường được gọi là ngoại lệ (hay exception). Những ngoại lệ này nếu không xử lý đúng mức sẽ có thể gây ra những sai sót trong thời gian thi hành hay ngay cả trong thời gian dịch. Dĩ nhiên, người viết mã có thể tùy theo tình huống mà viết các câu lệnh rẽ nhánh tránh không để cho mã vi phạm các lỗi. Hay là dùng các câu lệnh xử lý các ngoại lệ này.

Một số ngôn ngữ không cung cấp khả năng xử lý ngoại lệ thì người viết mã buộc phải tự mình phán đoán hết các tình huống có thể vi phạm lỗi và dùng câu lệnh điều kiện để loại trừ.

Các loại lỗi về ngôn ngữ khi lập trình thường xảy ra là:

Lỗi cú pháp

Vi phạm khi đặt hay gọi tên biến và hàm: Lỗi loại này thường rất dễ tìm ra trong lúc phát triển mã. Thường người ta có thể đọc lại các bảng tham chiếu về ngôn ngữ để tránh sai cú pháp mẫu (prototype) của hàm hay tránh dùng các ký tự đặc biệt bị cấm không cho dùng trong khi đặt tên. Trong không ít trường hợp người lập trình có thể đã định nghĩa cùng một tên cho nhiều hơn một đối tượng khác nhau và lại có giá trị toàn cục. Trong nhiều trường hợp chúng tạo thành lỗi ý nghĩa.

Lỗi chính tả: người viết mã có thể viết hay gọi sai tên hàm, tên biến. Trong nhiều ngôn ngữ có kiểu tĩnh thì các lỗi này sẽ rất dễ bị phát hiện. Còn đối với ngôn ngữ có kiểu động hay có kiểu yếu thì nó có thể dẫn đến sai sót nghiêm trọng vì bản thân phần mềm dịch không hề phát hiện ra.

Vượt quá khả năng tính toán: Bản thân máy tính và hệ điều hành cũng có rất nhiều giới hạn về phần cứng, phần mềm và các đặc điểm chuyên biệt. Khi người lập trình yêu cầu máy làm quá khả năng sẽ gây ra các lỗi mà đôi khi không xác định được như Lỗi thời gian (timing error) thường thấy trong các hệ thống đa luồng hay đa nhiệm.

Lỗi chia cho 0: Bản thân phần cứng máy tính sẽ ở trạng thái bất định khi thực hiện phép chia cho 0; trong nhiều trường hợp, mã sau khi dịch mới phát hiện ra trong lúc thi hành và được đặt tên là lỗi division by 0.

Dùng hay gọi tới các địa chỉ hay các thiết bị mà bản thân máy hay hệ điều hành đang thực thi lại không có hay không thể đạt tới. Đây là trường hợp rất khó lường. Bởi vì thường ngươi lập trình có thể viết mã trên một máy nhưng lại cho thi hành trong các máy khác và các máy này lại không thỏa mãn các yêu cầu. Để giảm trừ các lỗi loại này thường người lập trình nên xác định trước các điều kiện mà phần mềm làm ra sẽ hỗ trợ.

Ví dụ: trong nhiều phần mềm ngày nay ở trong vỏ hộp đều được ghi rõ các yêu cầu về vận tốc, bộ nhớ tối thiểu, và quan trọng là hệ điều hành nào mà phần mềm đó hỗ trợ.

Gán sai dữ liệu: Tức là dùng một dữ liệu có kiểu khác với kiểu của biến để gán cho biến đó một cách không chủ ý. Đối với các ngôn ngữ tĩnh hay có kiểu mạnh thì lỗi này

dễ tìm thấy hơn. Còn những ngôn ngữ động hay ngôn ngữ có kiểu yếu thì lỗi tạo ra sẽ có thể khó phát hiện và thường xảy ra lúc thi hành.

Các lỗi biên: Lỗi biên thường xảy ra khi người viết mã không chú ý đến các giá trị ở biên của các biến, các hàm. Những lỗi để thấy có thể là:

Gán giá trị của một số (hay một chuỗi) lên một biến mà nó vượt ngoài sự cho phép của định nghĩa.

Ví dụ: Gán một giá trị lớn hơn 255 cho một biến có kiểu là short trong ngôn ngữ C tạo nên các lỗi khi biến chạy trong vòng lặp đạt giá trị ở biên.

Ví dụ: đoạn mã C/C++ sau đây sẽ gây ra lỗi biên—Chia cho 0

```
for (m=10; m >= 0, m--) {x= 8+ 2/m; }
```

Lỗi ý nghĩa

Lỗi về quản lý bộ nhớ. Trong nhiều loại ngôn ngữ người lập trình có thể xin đăng ký một lượng nào đó của bộ nhớ để dùng làm chỗ chứa giá trị cho một biến (một hàm hay một đối tượng). Thường thì sau khi dùng xong người viết mã phải có phần lệnh trả về các phần bộ nhớ mà nó đã đăng ký dùng. Nếu không, sự trả về này chỉ xảy ra ở giai đoạn kết thúc việc thi hành. Trong nhiều trường hợp, số lượng bộ nhớ xin đăng ký quá nhiều và không được dùng đúng chỗ có thể làm cho máy kiệt quệ về mặt tài nguyên bộ nhớ và gây ra treo máy. Điển hình nhất là việc xin đăng ký các phần của bộ nhớ trong các vòng lặp lớn để gán cho các đối tượng bên trong vòng lặp nhưng không trả về sau khi sử dụng. Người ta thường gọi lỗi kiểu này là lỗi rò rỉ bộ nhớ (memory leaking).

Sai sót trong thuật toán: Trước khi viết một chương trình, để giảm thiểu sai sót về mặt lập luận thì người ta có nhiều biện pháp để làm giảm lỗi trong đó có các phương pháp vẽ lưu đồ, vẽ sơ đồ khối, hay viết mã giả. Những biện pháp này nhằm tạo nên các thuật toán để giải quyết vấn đề. Tuy nhiên, một thuật toán không chặt chẽ, xử lý không rốt ráo mọi trường hợp có thể xảy ra, không dự đoán được sự thay đổi trong lúc thi hành thì có thể tạo nên các lỗi và các lỗi này thường khó thấy bởi vì nó chỉ xảy ra ở những chỗ, những thời điểm mà người lập trình không ngờ trước. Một trong những phương pháp đơn giản làm giảm thiểu lỗi thuật toán là phải chú ý xử lý mọi tình huống khi dùng câu lệnh điều kiện (hay chẻ nhánh) mặc dù có thể có các trường hợp tưởng như hiển nhiên.

Lỗi về lập luận: Đây có thể xem là trường hợp đặc biệt của sai sót trong thuật toán. Trong các biểu thức tính giá trị, đôi khi không quen dùng đại số Bool (nhất là khi dùng luật De Morgan để phủ định một biểu thức phức tạp) nên người lập trình có thể tính toán sai, hay định nghĩa sai các phép toán. Do đó, giá trị trả về của các biểu thức logic hay biểu thức nhị phân sẽ bị sai trong một vài trường hợp hay toàn bộ biểu thức. Trong những tình huống như vậy phần mềm dịch sẽ không thể nào phát hiện ra cho đến khi chương trình được thi hành và lọt vào tình huống tính sai của người lập trình.

Các thành tố đặc trưng của ngôn ngữ OOP

Bài chi tiết: Lập trình hướng đối tượng OOP là chữ viết tắt của Object Oriented Programming có nghĩa là Lập trình hướng đối tượng được phát minh năm 1965 bởi Ole-Johan Dahl và Kristen Nygaard trong ngôn ngữ Simula. So với phương pháp lập trình cổ điển, thì triết lý chính bên trong loại ngôn ngữ loại này là để

tái dụng các khối mã nguồn và cung ứng cho các khối này một khả năng mới: chúng có thể có các hàm (gọi là các phương thức) và các dữ liệu (gọi là thuộc tính) nội tại. Khối mã như vậy được gọi là đối tượng. Các đối tượng thì độc lập với môi trường và có khả năng trả lời với yêu cầu bên ngoài tùy theo thiết kế của người lập trình. Với cách xây dựng này, mỗi đối tượng sẽ tương đương với một chương trình riêng có nhiều đặc tính mới mà quan trọng nhất là tính đa hình, tính đóng, tính trừu tượng và tính thừa kế.

Thừa kế

Đây là đặc tính cho phép tạo các đối tượng mới từ đối tượng ban đầu và lại có thể có thêm những đặc tính riêng mà đối tượng ban đầu không có. Cơ chế này cho phép người lập trình có thể tái sử dụng mã nguồn cũ và phát triển mã nguồn mới bằng cách tạo ra các đối tượng mới thừa kế đối tượng ban đầu.

Đa hình

Tính đa hình được thể hiện trong lập trình hướng đối tượng rất đặc biệt. Người lập trình có thể định nghĩa một thuộc tính (chẳng hạn thông qua tên của các phương thức) cho một loạt các đối tượng gần nhau nhưng khi thi hành thì dùng cùng một tên gọi mà sự thi hành của mỗi đối tượng sẽ tự động xảy ra tương ứng theo từng đối tượng không bị nhầm lẫn.

Ví dụ: khi định nghĩa hai đối tượng "hinh_vuong" và "hinh_tron" thì có một phương thức chung là "chu_vi". Khi gọi phương thức này thì nếu đối tượng là "hinh_vuong" nó sẽ tính theo công thức khác với khi đối tượng là "hinh_tron".

Trừu tượng

Đặc tính này cho phép xác định một đối tượng trừu tượng, nghĩa là đối tượng đó có thể có một số đặc điểm chung cho nhiều đối tượng nhưng bản thân đối tượng này có thể không có các biện pháp thi hành.

Ví dụ: người lập trình có thể định nghĩa đối tượng "hinh" hoàn toàn trừu tượng không có đặc tính mà chỉ có các phương thức được đặt tên chẳng hạn như "chu_vi", "dien_tich". Để thực thi thì người lập trình buộc phải định nghĩa thêm các đối tượng cụ thể chẳng hạn định nghĩa "hinh_tron" và "hinh_vuông" dựa trên đối tượng "hinh" và hai định nghĩa mới này sẽ thừa kế mọi thuộc tính và phương thức của đối tượng "hinh".

Đóng

Tính đóng ở đây dược hiểu là các dữ liệu (thuộc tính) và các hàm (phương thức) bên trong của mỗi đối tượng sẽ không cho phép người gọi dùng hay thay đổi một cách tự do mà chỉ có thể tương tác với đối tượng đó qua các phương thức được người lập trình cho phép. Tính đóng ở đây có thể so sánh với khái niệm "hộp đen", nghĩa là người ta có thể thấy các hành vi của đối tượng tùy theo yêu cầu của môi trường nhưng lại không thể biết được bộ máy bên trong thi hành ra sao.

Một số thành tố thường thấy khác của một ngôn ngữ lập trình hiện đại

Nhiều ngôn ngữ lập trình hiện đại, nhất là các ngôn ngữ viết cho Windows, thường có cung cấp thêm một số lượng rất lớn các thư viện bao gồm nhiều hàm để hỗ trợ giao diện người dùng và các thiết bị đầu cuối.

Giao diện đồ họa

Các ngôn ngữ chuẩn thường không đề cập tới sự cung cấp thư viện giúp cho việc thiết lập giao diện đồ họa (graphic interface). Nhưng hầu hết trong các ngôn ngữ hiện đại mà nhà sản xuất cung cấp cho các hệ điều hành đều có thêm thư viện các hàm và các biến toàn cục có thể dùng để nhanh chóng viết mã có giao diện phù hợp.

Ví dụ như GDK (cho Linux), Java (cho mọi hệ), Visual C/C++/C# (cho Windows),... Và các thư viện này ngày nay đã trở thành các thành tố không thể thiếu cho người lập trình.

Điều khiển theo sự kiện

Tương tự trên, triết lý đằng sau của việc điều khiển theo sự kiện là để hỗ trợ cho việc đồng bộ sử dụng cùng lúc nhiều thiết bị đầu cuối như là chuột, bàn phím, máy in... Việc nhận một mệnh lệnh từ chuột hay từ bàn phím phải được lập tức đồng bộ và thay đổi giao điện tức thời để cập nhật hoá.

Thời gian thực

Bản thân một ngôn ngữ sẽ không nói rõ là có hỗ trợ cho tính năng này hay không. Phản ứng và cập nhật dữ liệu theo thời gian thực là một hướng phát triển nhằm đáp ứng các nhu cầu đồng bộ hoá nhanh dữ liệu mà chúng có thể chia sẻ cho nhiều nơi hay là để thỏa mãn nhu cầu cần thiết đồng bộ hóa dữ liệu của các dịch vụ (ngân hàng, hàng không và quân sự chẳng hạn).

Hỗ trợ hệ điều hành

Ngoài các hỗ trợ cho các giao diện thì ngày nay hầu hết các hệ điều hành (Linux/UNIX, Netware và Windows) đều có khả năng đa luồng (multithreading) hay đa nhiệm (multitasking). Những khả năng này nâng cao hiệu quả của máy tính. Các ngôn ngữ, do đó thường có thêm các hàm, thủ tục hay các biến cho phép người lập trình tận dụng chúng. Việc viết mã cho kiến trúc đa luồng và đa nhiệm không đơn giản như viết mã cho các hệ thống thông thường. Người lập trình ngoài kỹ năng viết mã, còn phải luyện tập cách xử lý và đồng bộ nhiều thao tác được thi hành đồng thời trong một chương trình mà không gây ra ách tắc hay vi phạm các nguyên tắc quản lý bộ nhớ hay các quy tắc lập trình theo đa luồng hay đa nhiệm.

Lưu ý: Hầu hết các hệ điều hành hỗ trợ kiến trúc đa luồng hay đa nhiệm đều có khả năng thực thi những chương trình được tạo ra từ mã viết theo kiểu thông thường mà không đá động tới các chức năng đa luồng hay đa nhiệm. Điểm khác nhau là khi không dùng tới các ưu điểm đa luồng hay đa nhiệm thì chương trình đó sẽ không tận dụng được ưu thế phần cứng và phần mềm hỗ trợ (thường thì chương trình đó chạy chậm hơn)

#include <conio.h> cho các hàm getch(), putch(), clrscr(), gotoxy() ...

#include <stdio.h> cho các hàm khác như gets(), fflus(), fwrite(), scanf()...

ở gần chỗ bắt đầu chương trình. Tệp stdio.h định nghĩa các macro và biến cùng các hàm dùng trong thư viện vào/ra. Dùng dấu ngoặc < và > thay cho các dấu nháy thông thường để chỉ thị cho trình biên dịch tìm kiếm tệp trong danh mục chứa thông tin tiêu đề chuẩn.

2.2. Các kiểu dữ liệu cơ bản: kiểu số, ký tự, chuỗi, ...

2.2.1. Dữ liệu và kiểu dữ liệu

Máy tính ngày nay cho phép làm việc với hầu hết tất cả các loại dữ liệu mà con người phải xử lý hàng ngày, ví dụ một số loại dữ liệu có thể kể đến như: số thực, số nguyên, ký tự, xâu ký tự, logic, ngày tháng. Ngôn ngữ lập trình thường cho phép làm việc với một số (hoặc tất cả) các loại dữ liệu kể trên; và mỗi loại dữ liệu được thể hiện bằng từ khóa mô tả kiểu dữ liệu *(data type*) đó, các kiểu dữ liệu này được gọi là các kiểu dữ liệu cơ bản hay nguyên thủy của ngôn ngữ.

Bảng 1-1: **Các kiểu cơ bản trong** *C*

Kiểu dữ liệu	Tên kiểu (từ khóa)	Số byte	Miền giá trị	
			Từ	Đến
Ký tự	*char*	*1*	*-128*	*128*
	unsigned char	*1*	*0*	*255*
Số nguyên	*int*	*2*	*-32767*	*32768*
	unsigned int	*2*	*0*	*65535*
	short	*2*	*-32767*	*32768*
	long	*4*	-2^{15}	$2^{15}-1$
Số thực	*float*	*4*	$\pm 10^{-37}$	$\pm 10^{38}$
	double	*8*	$\pm 10^{-307}$	$\pm 10^{+308}$

Trong *C*, các kiểu dữ liệu cơ bản bao gồm: kiểu ký tự, kiểu số nguyên, kiểu số thực; nội dung cơ bản về các kiểu dữ liệu này được thể hiện như Bảng 1-1.

Trong ngôn ngữ lập trình, mỗi kiểu dữ liệu được quan tâm ở các khía cạnh: tên gọi - để sử dụng khi khai báo; số byte trong bộ nhớ cần dùng cho mỗi biến - để sử dụng hợp lý bộ nhớ của máy tính; miền giá trị - để xem xét về sự phù hợp của miền giá trị với đối tượng mà nó mô tả; và các phép toán (*operator*) có thể thực hiện đối với kiểu dữ liệu đó.

Ngoài các kiểu dữ liệu cơ bản thì hầu hết các ngôn ngữ đều định nghĩa sẵn hoặc cho phép người lập trình tự định nghĩa các kiểu dữ liệu phức tạp hơn từ những kiểu cơ bản; các kiểu này được gọi chung là kiểu dữ liệu có cấu trúc. Các kiểu dữ liệu có cấu trúc sẽ được trình bày trong các phần sau của tài liệu.

2.2.2. Kiểu ký tự

Bảng mã ASCII là một bảng quy ước về mã cho các chữ cái La tinh, những ký tự, kí hiệu thường dùng khác. Ký tự, kí hiệu, qui ước, chữ cái trong bảng mã ASCII được gọi chung là ký tự (*Character*). Người ta đưa ra bảng mã ASCII để thống nhất về cách mã hóa đối với các ký tự được sử dụng trên máy tính, và như mô tả trong tên bảng mã, **ASCII** - **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange, thì nó được hiểu là mã tiêu chuẩn dùng trong việc trao đổi thông tin của Mỹ.

Bảng mã ASCII chuẩn có 128 ký tự bao gồm các ký tự điều khiển (không hiển thị), và các ký tự hiển thị được. Bảng mã ASCII mở rộng có 256 ký tự bao gồm 128 ký tự của bảng mã ASCII chuẩn và 128 ký tự mở rộng gồm nhiều ký tự có dấu, ký tự trang trí khác. Bảng 95 ký tự hiển thị được của bảng mã ASCII được cho như Bảng 1-2.

Trong *C* kiểu ký tự được hiểu theo hai cách, một là ký tự trong bảng mã ASCII và hai là mã ASCII của ký tự đó. Ví dụ, nếu *ch* là một biến kiểu ký tự thì lệnh gán *ch* =

'A' hoặc *ch = 65* có ý nghĩa như nhau vì mã ASCII của ký tự 'A' là 65 (xem Bảng 1-2). Theo Bảng 1-1 ta thấy có hai loại dữ liệu kiểu kí tự là *char* với miền giá trị từ -128 đến 127 và *unsigned char* (kí tự không dấu) với miền giá trị từ 0 đến 255. Trường hợp một biến được gán giá trị vượt ra ngoài miền giá trị của kiểu thì giá trị của biến sẽ được tính theo mã bù - *(256 - c)*. Ví dụ nếu gán cho *char ch* giá trị 179 (vượt khỏi miền giá trị đã được qui định của *char*) thì giá trị thực sự được lưu trong máy sẽ là - *(256 - 179) = -77*.

Bảng 1-2: **Các kí tự hiển thị được trong bảng mã ASCII**

Mã	Ký tự	Mã	Ký tự	Mã	Ký tự	Mã	Ký tự	Mã	Ký tự
32	SP	51	3	70	F	89	Y	108	l
33	!	52	4	71	G	90	Z	109	m
34	"	53	5	72	H	91	[	110	n
35	#	54	6	73	I	92	\	111	o
36	$	55	7	74	J	93	]	112	p
37	%	56	8	75	K	94	^	113	q
38	&	57	9	76	L	95	_	114	r
39	'	58	:	77	M	96	`	115	s
40	(	59	;	78	N	97	a	116	t
41	)	60	<	79	O	98	b	117	u
42	*	61	=	80	P	99	c	118	v
43	+	62	>	81	Q	100	d	119	w
44	,	63	?	82	R	101	e	120	x
45	-	64	@	83	S	102	f	121	y
46	.	65	A	84	T	103	g	122	z
47	/	66	B	85	U	104	h	123	{
48	0	67	C	86	V	105	i	124	|
49	1	68	D	87	W	106	j	125	}
50	2	69	E	88	X	107	K	126	~

Một ví dụ về việc sử dụng kiểu *char* và *unsigned char* trong *C* như sau:

```
char c, d;              // c, d duoc phep gan gia tri tu -128 đen 127
unsigned char e,f;      // e đuoc phep gan gia tri tu 0 đen 255
c = 65 ; d = 179;       // d có giá tri ngoài mien cho phép
e = 179; f = 330;       // f có giá tri ngoài mien cho phép
printf("%c,%d",c,c);    // in ra chu cái 'A' và giá tri so 65
printf("%c,%d",d,d);    // in ra là kí tu '|' và giá tri so -77
printf("%c,%d",e,e);    // in ra là kí tu '|' và giá tri so 179
printf("%c,%d",f,f);    // in ra là kí tu 'J' và giá tri so 74
```

Từ ví dụ trên ta thấy một biến nếu được gán giá trị ngoài miền giá trị cho phép sẽ dẫn đến kết quả không theo suy nghĩ thông thường. Do vậy nên ước lượng chính xác nhu cầu và miền giá trị của kiểu dữ liệu khi sử dụng; ví dụ nếu muốn sử dụng biến có

giá trị từ 128 .. 255 thì nên khai báo biến dưới dạng kí tự không dấu (*unsigned char*).

Với nhiều ngôn ngữ thì ký tự được hiểu là những ký tự của bảng mã ASCII.

2.2.3. Kiểu số nguyên

Trong *C*, các số nguyên được phân chia thành bốn kiểu khác nhau với các miền giá trị tương ứng được cho trong Bảng 1-1. Các kiểu số nguyên gồm có số nguyên ngắn (*short*), số nguyên (*int*), số nguyên không dấu (*unsigned int*) sử dụng 2 byte và số nguyên dài (*long*) sử dụng 4 byte. Kiểu số nguyên còn được chia thành hai loại có dấu và không dấu với cùng số byte phải dùng nhưng có miền giá trị khác nhau; mục đích của việc này là nhằm tạo sự mền dẻo, thuận tiện cho người sử dụng có thể tiết kiện được tài nguyên máy tính mà vẫn giải quyết được vấn đề đặt ra.

Các phép toán trong *C* có thể thực hiện trên số nguyên được đề cập trong các nội dung sau của tài liệu.

2.2.4. Kiểu số thực

Để sử dụng số thực ta cần khai báo kiểu *float* hoặc *double*, miền giá trị của các biến kiểu này được mô tả trong Bảng 1-1. Các giá trị số kiểu *double* được gọi là số thực với độ chính xác gấp đôi vì với kiểu dữ liệu này máy tính có cách biểu diễn khác so với kiểu *float* để đảm bảo số số lẻ sau một số thực có thể tăng lên nhằm thể hiện sự chính xác cao hơn so với số kiểu *float*. Tuy nhiên, trong các bài toán thông dụng thường ngày độ chính xác của số kiểu *float* là đủ dùng.

2.3 Hằng, biến, hàm, các phép toán và biểu thức

2.3.1. Biến

Biến là đối tượng được tạo ra trong chương trình dùng để lưu trữ giá trị thuộc một kiểu nào đó cần thiết cho quá trình xử lý và tính toán. Giá trị của biến có thể thay đổi trong quá trình tính toán. Có thể khẳng định biến là đối tượng quan trọng nhất trong chương trình, không chỉ đơn thuần dùng để lưu trữ dữ liệu mà việc tổ chức các biến còn quyết định đến tính hiệu quả của thuật toán và chương trình. Điều này được khẳng định qua tiêu đề cuốn sách "Giải thuật + Cấu trúc dữ liệu = Chương trình" của giáo sư Niklaus Emil Wirth, câu nói được hiểu như một định lý trong lĩnh vực lập trình.

Trong *C*, biến phải được khai báo trước khi sử dụng. Việc khai báo nhằm cung cấp các thuộc tính của một biến như: tên gọi (tên biến), kiểu của biến, và vị trí khai báo biến đó cũng có ý nghĩa nhất định. Thông thường với nhiều ngôn ngữ lập trình tất cả các biến phải được khai báo ngay từ đầu chương trình, tuy nhiên để thuận tiện *C* cho phép khai báo biến ngay bên trong chương trình. Với *C*, khi cần, người sử dụng có thể khai báo một biến tại bất kỳ vị trí nào trong chương trình để sử dụng.

2.3.1.1. Khai báo

Trong *C*, khai báo biến được viết theo cú pháp mô tả sau đây.

Cú pháp:

Kiểu Tên_biến_1 [,Tên_biến_2, Tên_biến_3 …];

Trong đó:

- *Kiểu*: Là từ khóa qui định cho kiểu dữ liệu cần dùng. Từ khóa tương ứng với các kiểu dữ liệu cơ bản được đề cập trong Bảng 1-1;

- *Tên_biến_1*: Tên biến do người sử dụng tự đặt. Tên biến phải được viết theo quy tắc viết tên đã mô tả trong phần 1.1.3 - Tên gọi. Quy tắc cơ bản là không được chứa khoảng trống, không bắt đầu bằng chữ số, và không (nên) dùng chữ cái Tiếng Việt có dấu.
- Khi cần khai báo nhiều biến có cùng kiểu thì có thể liệt kê tên các biến đó, *Tên_biến_2, Tên biến_3, ...* , sau *Tên_biến_1* và mỗi biến cách nhau bởi một dấu phảy (,).
- Trong *C* dấu chấm phảy (;) cuối lệnh trên là bắt buộc.

Ví dụ xét hai dòng lệnh sau:

```
int a, b, c;
float delta;
```

Hai lệnh trên khai báo các biến *a, b, c* có kiểu là số nguyên và *delta* có kiểu là số thực. Các khai báo trong ví dụ trên không xác định giá trị ban đầu cho tất cả các biến, khi được tạo ra chúng nhận một giá trị (ngẫu nhiên) nào đó. Trong *C* cho phép khai báo biến kết hợp với việc xác định giá trị ban đầu cho biến đó theo cú pháp mô tả dau đây.

Cú pháp:

```
Kiểu     Tên_biến_1 = giá_trị_1 [,Tên_biến_2 = giá_trị_2, …];
```

Trong đó:

- *Kiểu, Tên_biến_1, Tên_biến_2 ...* được hiểu như trên;
- *giá_trị_1, giá_trị_2 …* là các **biểu thức** cho kết quả thuộc miền giá trị của *Tên_biến_1, Tên_biến_2 ...*

Chú ý: Khi đặt tên biến, ngoài việc phải tuân theo quy tắc đặt tên của ngôn ngữ, nên đặt sao cho tên đó có tính gợi nhớ để thuận tiện cho sử dụng về sau. Tuy vậy thì không nên dùng tên biến quá dài, bởi nó dễ làm khổng kềnh các biểu thức về sau. Biến *delta* trong ví dụ trên được đặt tên như vậy với ý định dùng nó để lưu trữ giá trị delta theo cách tính nghiệm của phương trình bậc hai.

2.3.1.2. Phạm vi tác động của biến

Như đã đề cập ở trên, mỗi biến khi khai báo ngoài những thuộc tính đã biết như tên, kiểu thì vị trí trong chương trình của biến đó cũng có ý nghĩa nhất định. Ý nghĩa về vị trí khai báo của biến chính là phạm vi tác động của biến. Một biến xuất hiện trong chương trình có thể được sử dụng bởi hàm này nhưng không được bởi hàm khác hoặc bởi cả hai, điều này phụ thuộc chặt chẽ vào vị trí nơi biến được khai báo. Một nguyên tắc đầu tiên là biến sẽ có tác dụng kể từ vị trí nó được khai báo cho đến hết khối lệnh chứa nó; khối lệnh ở đây được hiểu là một lệnh hợp thành, một hàm, một chương trình.

2.3.2. Hằng

Hằng, cũng giống như biến, là những đối tượng được tạo ra trong chương trình dùng để lưu trữ giá trị thuộc một kiểu nào đó cần thiết cho quá trình tính toán và xử lý. Tuy nhiên khác với biến là giá trị của hằng được xác định khi khai báo và không thể thay đổi được trong chương trình.

Tương ứng với mỗi kiểu dữ liệu ngôn ngữ lập trình qui định cách biểu diễn hằng

của kiểu đó. Các hằng thường dùng trong *C* gồm có: hàng ký tự, hàng số nguyên, hằng số thực, hằng xâu ký tự.

2.3.2.1. Hằng số nguyên

Ngôn ngữ *C* cho phép viết hằng số nguyên ở dạng thập phân, bát phân và thập lục phân.

- **Dạng thập phân**: Dùng 10 chữ số 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 để biểu diễn như chúng ta vẫn sử dụng hàng ngày; ví dụ số 2012, 365, -328, 40000. Với các hằng nguyên kể trên, số 40000 vượt quá giới hạn số nguyên *int* (lớn nhất 32767) khi đó C tự động dùng kiểu *long* để biểu diễn. Trường hợp muốn chủ động biểu diễn số nguyên dạng *long* thì thêm vào chữ cái *l* hay *L* sau cách viết thông thường; ví dụ 2012*l*, 365*L*.
- **Dạng bát phân**: Dùng các chữ số 0, 1, 2, 3, 4, 5, 6, 7 để biểu diễn. Không giống cách biểu diễn thông thường của số thập phân, *C* quy ước việc biểu diễn số bát phân như sau:
- [dấu] 0 <chữ số>

  Trong đó

  [dấu]: để chỉ ra số âm (dấu -) hay dương (dấu + hoặc không dấu);

  0: là ký hiệu bắt buộc;

  <*chữ số*>: các chữ số 0 đến 7.

  Ví dụ: 0345 là số 345 trong hệ bát phân (= $3*8^2 + 4*8^1 + 5*8^0$ = 229 trong hệ thập phân).
- **Dạng thập lục phân**: Sử dụng 10 ký số từ 0 đến 9 và 6 ký tự A, B, C, D, E ,F để biểu diễn. *C* quy ước việc biểu diễn số thập lục phân như sau:
- [dấu] 0x <ký tự>

  Trong đó:

  dấu: để chỉ ra số âm (dấu -) hay dương (dấu + hoặc không dấu);

  0x: là ký hiệu bắt buộc;

  Ký tự: gồm các chữ số 0 đến 9 và 6 chữ cái A, B, C, D, E, F.

  *Ví dụ: 0x34F là số 34F trong hệ thập lục phân (= 3*162 + 4*161 + 15*160 = 847* trong hệ thập phân).

Chú ý: Ngoài những điều trên cần lưu ý:

- Hằng số nguyên 4 byte (*long*): Số long (số nguyên dài) được biểu diễn như số *int* trong hệ thập phân và kèm theo ký tự l hoặc L. Một số nguyên nằm ngoài miền giá trị của số *int* (2 *bytes*) là số long (4 *bytes*) ; ví dụ: 45345L hay 45345l hay 45345.
- Các hằng số còn lại: Viết như cách viết thông thường (không có dấu phân cách giữa 3 số) .

2.3.2.2. Hằng số thực

Số thực bao gồm các giá trị kiểu *float, double, long double* được thể hiện theo hai 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 -223.333 3.00 -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* có dạng *men* hay *mEn*, trong đó: *m* là phần định trị là một số nguyên hay số thực được viết theo cách 1; *n* là phần mũ 10 là một số nguyên. Khi đó giá trị của số thực là: $m*10^n$, ví dụ: 1234.56e-3 = 1.23456 (là số $1234.56 * 10^{-3}$), hoặc -123.45E4 = -1234500 (là $-123.45 *10^4$).

2.3.2.3. 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’*.

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).

2.3.2.4. 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: "*Ngon ngu lap trinh C*", "*Cong hoa xa hoi chi nghia Viet Nam*".

Chú ý:

- Một chuỗi không có nội dung "" được gọi là chuỗi rỗng;
- Khi lưu trữ trong bộ nhớ, một chuỗi được kết thúc bằng ký tự *NULL* (‘\0’: mã ASCII là 0);
- 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\"*"".

2.3.2.5. Khai báo

Hằng tham gia vào các biểu thức tính toán theo hai cách: 1) là một giá trị cụ thể trong biểu thức, 2) được xác định thông qua tên hằng. Khi hằng không xuất hiện nhiều lần thì nên sử dụng chúng theo cách 1, và ngược lại, khi hằng dùng đến nhiều lần thì nên sử dụng theo cách 2.

Ví dụ, xét đoạn chương trình sau:

```
...
delta = b*b - 4*a*c;
...
x1= (-b+sqrt(delta))/(2*a);
x2= (-b-sqrt(delta))/(2*a);
...
```

Trong ví dụ trên, số *4* trong dòng lệnh đầu, số *2* trong hai dòng lệnh dưới là các hằng số được dùng trực tiếp trong các biểu thức.

Để sử dụng theo cách 2 cần khai báo hằng theo cú pháp sau:

Cú pháp:

```
#define   Tên_hằng    giá_trị;
```

Hoặc:

```
const    Kiểu  Tên_hằng = giá_trị;
```

Trong đó:

- *Kiểu*: Là từ khóa qui định cho kiểu dữ liệu cần dùng; như phần trước đã nêu các kiểu dữ liệu cơ bản có thể là *char, unsigned char, int, float, long, double* …;
- *Tên hằng*: Do người sử dụng tự định nghĩa, lưu ý rằng tên hằng phải tuân theo qui tắc đặt tên qui định bởi ngôn ngữ lập trình.
- *giá_trị*: Giá trị thuộc miền giá trị của Kiểu đối với khai báo dạng *const*.

Ví dụ có thể khai báo các hằng để sử dụng như sau:

```
#define so_Pi 3.1415
const int Max = 1001;
const char* monhoc = "Ky thuat lap trinh";
```

2.3.3. Biểu thức

Biểu thức là dãy kí hiệu kết hợp giữa các toán hạng (*operand*), toán tử (*operator*) và cặp dấu () theo một qui tắc nhất định. Các toán hạng là hằng, biến, hàm; toán tử là các phép toán số học, phép so sánh, phép logic, các phép toán trên tập ký tự và xâu ký tự. Ví dụ biểu thức tính delta: *b*b - 4*a*c*, biểu thức tính nghiệm của phương trình bậc hai *(-b+sqrt(delta))/(2*a)*.

Trong các ngôn ngữ lập trình phép toán trên các kiểu dữ liệu giống nhau nói chung là giống nhau, chúng thường khác nhau ở một số khía cạnh nhỏ như có hay không một toán tử nào đó mà ngôn ngữ khác không có. Đối với kiểu dữ liệu số, *C* cung cấp một cách phong phú nhất các phép toán như các toán tử tăng, giảm, toán tử *bitwise* ... so với ngôn ngữ khác do vậy phần nội dung các phép toán sau đây là định nghĩa bởi ngôn ngữ lập trình *C*.

2.3.3.1. Các phép toán số học

- ***Các phép toán + (cộng), - (trừ), * (nhân), / (chia)***: được hiểu theo nghĩa thông thường trong số học. Kết quả trả về của các phép chia, *a/b*, phụ thuộc vào kiểu của các toán hạng *a* và *b;* nếu cả *a* và *b* là nguyên thì kết quả phép chia là lấy phần nguyên, nếu một trong hai toán hạng là thực thì kết quả phép chia trả về giá trị thực. Ví dụ *16/3 = 5*, nhưng *16.0/3 = 16/3.0 = 16.0/3.0 = 5.3*.
- ***Phép toán % (lấy phần dư)***: *a % b* trả về kết quả là phần dư của phép chia *a* cho *b*. Ví dụ *16 % 3 =1, 6 % 2 =0*.
- ***Toán tử ++ (tăng) và -- (giảm)***: Đây là các phép toán một ngôi, toán tử ++ thêm 1 vào toán hạng của nó và -- trừ bớt 1; nói cách khác: *++ x* giống như *x = x + 1* và *-- x* giống như *x = x - 1*. Cả hai toán tử tăng và giảm đều có thể đặt trước hoặc sau toán hạng là một biến. Ví dụ: *x = x + 1* có thể viết *x ++* (hay *++ x*). Tuy nhiên giữa việc đặt trước hoặc đặt sau có một khác biệt nhỏ; khi một toán tử tăng hay giảm đứng trước toán hạng của nó, thì giá trị của toán hạng sẽ được tăng hay giảm trước khi tham gia vào tính giá trị của biểu thức; nếu toán tử đi sau toán hạng, thì giá trị của toán hạng được tăng hay giảm sau khi nó tham gia vào tính giá trị của biểu thức. Ví dụ, nếu *x = 10* thì lệnh *y = ++ x* sẽ cho *y* giá trị bằng *11* và sau đó *x* được tăng lên 1 giá trị; nhưng nếu thực hiện lệnh *y = x ++* thì *y* sẽ nhận giá trị bằng *10*, sau đó *x* sẽ được tăng lên 1 đơn vị.

2.3.3.2. Các phép so sánh và logic

Đây là các phép toán mà giá trị trả lại là đúng hoặc sai. Nếu giá trị của biểu thức là

đúng thì nó nhận giá trị 1, ngược lại là sai thì biểu thức nhận giá trị 0. Nói cách khác 1 và 0 là giá trị cụ thể của hai khái niệm "đúng", "sai" ở trong *C*; mở rộng hơn *C++* quan niệm một giá trị bất kỳ khác 0 là "đúng" và giá trị 0 là "sai", đây được hiểu là hai hằng logic trong *C*.

- ***Các phép so sánh*** là các phép toán hai ngôi và ký hiệu của chúng như sau: == (bằng nhau), *!*= (khác nhau), > (lớn hơn), < (nhỏ hơn), >= (lớn hơn hoặc bằng), <= (nhỏ hơn hoặc bằng). Ví dụ, *a == b* là phép so sánh bằng, nó trả về kết quả đúng (*true*) nếu giá trị *a* bằng giá trị của *b*, trả về kết quả sai (*false*) trong tình huống ngược lại. Lưu ý trong phép so sánh thì hai toán hạng phải có cùng kiểu.
- ***Các phép logic*** bao gồm các phép toán: *&&* (*AND*), || (*OR*), và *!* (*NOT*), trong đó *&&* và || là các phép toán hai ngôi và *!* là phép toán một ngôi. Bảng giá trị các phép toán logic được cho như Bảng 1.3.

*Bảng 1-3:***Bảng giá trị các phép toán logic**

a	b	a&&b	a||b	!a
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Ví dụ có thể viết các biểu thức logic sử dụng phép so sánh và logic như sau:

```
3 && (4 > 5)              // = 0 vì có hạng thức (4>5) sai
(3 >= 1) && (7)           // = 1 vì cả hai hạng thức cùng đúng
!1                        // = 0
! (4 + 3 < 7)             // = 1 vì (4+3<7) bằng 0
5 || (4 >= 6)             // = 1 vì có một hạng thức (5) đúng
(5 < !0) || (4 >= 6)      // = 0 vì cả hai hạng thức đều sai
```

2.3.3.3. Các toán tử bitwise

Trong ngôn ngữ máy tính, các toán tử *bitwise* thực hiện tính toán (theo từng bit) trên một hoặc hai chuỗi bit, hoặc trên các số nhị phân. Với nhiều loại máy tính, việc thực thi các phép toán thao tác bit thường nhanh hơn so với khi thực thi các phép toán cộng, trừ, nhân, hoặc chia. Trong C/C++ các phép toán bitwise gồm có: *&* (*AND*), *|* (*OR*), *^* (*XOR*), *~* (*NOT*), *>>* (*dịch phải*), *<<* (*dịch trái*). Các toán hạng trong phép toán bitwise phải là dạng số nguyên.

- ***Toán tử &*** là toán tử hai ngôi dạng *a & b*, thực hiện phép AND từng cặp bit của *a* và *b* từ trái qua phải. Phép *AND bit* có kết quả như sau:

```
0 & 0 = 0
0 & 1 = 0
1 & 0 = 0
1 & 1 = 1
```

Ví dụ cho *a = 5*, biểu diễn bit của nó là 0101, *b = 3* và biểu diễn bit của nó là 0011 khi đó kết quả phép *a & b* được thực hiện như sau:

```
a       =       0101
```

b	=	0011
a & b	=	0001

- ***Toán tử*** **|** là toán tử hai ngôi dạng *a | b*, thực hiện phép *OR* từng cặp bit của *a* và *b* từ trái qua phải. Phép *OR bit* có kết quả như sau:

0 | 0 = 0
0 | 1 = 1
1 | 0 = 1
1 | 1 = 1

Ví dụ cho *a = 5*, biểu diễn bit của nó là 0101, *b = 3* và biểu diễn bit của nó là 0011 khi đó kết quả phép *a | b* được thực hiện như sau:

a	=	0101
b	=	0011
a | b	=	0111

- ***Toán tử*** **!** là toán tử một ngôi dạng *!a*, thực hiện phép *NOT* từng bit của *a*. Phép *NOT bit* có kết quả như sau:

! 0 = 1
! 1 = 0

Ví dụ cho *a = 5*, biểu diễn bit của nó là 0101 khi đó kết quả phép *!a* được thực hiện như sau:

a	=	0101
!a	=	1010

- ***Toán tử*** **^** là toán tử hai ngôi dạng *a^b*, thực hiện phép *XOR* từng cặp bit của *a* và *b* từ trái qua phải. Phép *XOR bit* có kết quả như sau:

0 ^ 0 = 0
0 ^ 1 = 1
1 ^ 0 = 1
1 ^ 1 = 0

Ví dụ cho *a=5*, biểu diễn bit của nó là 0101, *b=8* và biểu diễn bit của nó là 1000 khi đó kết quả phép *a^b* được thực hiện như sau:

a	=	0101
b	=	1000
a ^ b	=	1101

- ***Toán tử*** **>>** là toán tử hai ngôi dạng *a>>b*, trong đó *b* là số bước dịch chuyển qua phải các biểu diễn bit của *a*.Ví dụ cho *a=23* (*char*), biểu diễn bit của nó là 00010111 khi đó kết quả phép *a>>1* được thực hiện như sau:

Vị trí bit		7 6 5 4 3 2 1 0
a	=	0 0 0 1 0 1 1 1
a >> 1		-> 0 0 0 0 1 0 1 1 1 ->
a	=	0 0 0 0 1 0 1 1

- ***Toán tử*** **<<** là toán tử hai ngôi dạng *b<<a*, trong đó *b* là số bước dịch chuyển qua trái các biểu diễn bit của *a*.Ví dụ cho *a = 23* (*char*), biểu diễn bit của nó là 00010111 khi đó kết quả phép *1<<a* được thực hiện như sau:

Vị trí bit	7 6 5 4 3 2 1 0
a =	0 0 0 1 0 1 1 1
1 << a	<- 0 0 0 1 0 1 1 1 0 <-
a =	0 0 1 0 1 1 1 0

2.3.3.4. Một số toán tử khác

- ***Toán tử ? và : (dấu hai chấm)***: *C* cung cấp một cặp toán tử mạnh để thay thế cho một lệnh *if .. then .. else* là cặp toán tử *?* (hỏi chấm) cùng với *:* (hai chấm) theo dạng *E1 ? E2 : E3*, trong đó *E1* là biểu thức logic còn *E2, E3* là các biểu thức có cùng kiểu với biến. Cấu trúc *E1 ? E2 : E3* được thực hiện như sau: trước tiên *E1* được ước lượng, nếu đúng *E2* được ước lượng và nó trở thành giá trị của biểu thức; nếu *E1* sai, thì *E3* được ước lượng và trở thành giá trị của biểu thức. Ví dụ, đoạn lệnh sau:

```
x = 10;
y = x > 9 ? 100 : 200;
```

Sẽ gán cho *y* giá trị 100, và đoạn mã trên có thể thay bằng lệnh *if* như sau:

```
x = 10
if (x < 9) {y = 100;} else {y = 200;}
```

- ***Toán tử dấu, (dấu phẩy)***: Toán tử dấu phẩy được sử dụng để kết hợp các biểu thức lại với nhau. Bên trái của toán tử dấu phẩy luôn được xem là kiểu *void*. Điều đó có nghĩa là biểu thức bên phải trở thành giá trị của tổng các biểu thức được phân cách bởi dấu phẩy. Ví dụ biểu thức *x = (y = 3, y + 1)*, trước hết gán *3* cho *y* rồi gán *4* cho *x*; cặp dấu ngoặc đơn là cần thiết vì toán tử dấu phẩy có độ ưu tiên thấp hơn toán tử gán.
- ***Cách viết tắt khi thực hiện toán tử gán***: Khi thực hiện phép gán, C/C++ cho phép viết tắt chẳng hạn thay vì viết *x = x + 10* ta có thể viết *x +=10*; toán tử += báo cho chương trình dịch biết để tăng giá trị của *x* lên 10. Cách viết này làm việc trên tất cả các toán tử nhị phân (phép toán hai ngôi) của C/C++. Tổng quát: phép gán *(Biến) = (Biến) (Toán tử) (Biểu thức)* có thể được viết tắt dạng *(Biến) (Toán tử)= (Biểu thức).*

2.3.3.5. Thứ tự ưu tiên của các phép toán

Nếu có nhiều cặp ngoặc lồng nhau thì cặp trong cùng (sâu nhất) được tính trước. Các phép toán trong cùng một lớp có độ ưu tiên theo thứ tự: lớp nhân (*, /, &&), lớp cộng (+, −, ||). Nếu các phép toán có cùng thứ tự ưu tiên thì chương trình sẽ thực hiện từ trái sang phải. Các phép gán có độ ưu tiên cuối cùng và được thực hiện từ phải sang trái.

Trong C/C++, các toán tử được thực hiện theo thứ tự ưu tiên giảm dần theo thứ tự sau:

() []	*Cao nhất*
! ~ ++ -- (Kiểu) * &	
* / %	
+ -	
<< >>	
< <= > >=	
&	
^	

|	
&&	
||	
?:	
= += -= *= /=	
,	*Thấp nhất*

2.4 Các lệnh, khối lệnh:

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

Ví dụ 1:

```
{
char ten[30];
printf("\n Nhap vao ten cua ban:");
scanf("%s", ten);
printf("\n Chao Ban %s",ten);
}
```

Ví dụ 2:

```
#include <stdio.h>
#include<conio.h>
int main ()
{ /*đây là đầu khối*/
char ten[50];
printf("Xin cho biet ten cua ban !");
scanf("%s",ten);
getch();
return 0;
} /*đây là cuối khối*/
```

Một khối lệnh có thể chứa bên trong nó nhiều khối lệnh khác gọi là khối lệnh lồng nhau. Sự lồng nhau của các khối lệnh là không hạn chế.

Minh họa:

```
{
… lệnh;
{
… lệnh;
{
… lệnh;
}
… lệnh;
}
… lệnh;
}
```

Lưu ý về phạm vi tác động của biến trong khối lệnh lồng nhau:

- Trong các khối lệnh khác nhau hay các khối lệnh lồng nhau có thể khai báo các biến cùng tên.

Ví dụ 1:

```
{
… lệnh;
    {
    int a,b; /*biến a, b trong khối lệnh thứ nhất*/
            … lệnh;
    }
…lệnh;
    {
    int a,b; /*biến a,b trong khối lệnh thứ hai*/
            … lệnh;
    }
}
```

Ví dụ 2:

```
{
int a, b; /*biến a,b trong khối lệnh "bên ngoài"*/
… lệnh;
    { int a,b; /*biến a,b bên trong khối lệnh con*/}
}
```

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

Ví dụ:

```
{
    int a, b, c;
    …lệnh;
    {
    int c, d;
    …lệnh;
    }
}
```

2.5 Thực thi chương trình, nhập dữ liệu, nhận kết quả

Bài toán: Cho một mảng *NxM*, phần tử *k* và cột *c* phần tử kiểu nguyên cho trước. Xác định sự xuất hiện của phần tử *k* trên cột *c*.

Ý tưởng: Ban đầu trạng thái chưa tìm thấy. Duyệt trên cột *c* của ma trận, nếu xuất hiện chuyển trạng thái tìm thấy.

***Thuật toán*:**

Dữ liệu vào: arrData[N][M] phần tử, giá trị k, và cột c.

Dữ liệu ra: Xuất hiện của k trên c.

```
    found=0;
    for(i=1 -> N)
            if(arData[i][c]==k)
    found=1;
```

***Cài đặt chương trình*:**

```
#include <conio.h>
#include <stdio.h>
#define N 5
#define M 3
int main()
{
    int arrData[N][M]={{3,5,1},{2,4,5},{1,4,3},{3,2,5},{2,3,1}};
    int c=1;
    int k=3;
    int i, found=0;
    for(i=0;i<=N-1;i++)
    {
            if(arrData[i][c]==k)
            {
                    found=1;
            }
    }
    if(found==1)
    {
            printf("Tim thay gia tri %d tren cot %d cua ma tran",k,c);
    }
    else
    {
            printf("Khong tim thay %d tren cot %d cua ma tran", k, c);
    }
    getch();
    return 0;
}
```

CHƯƠNG 3. CÁC CẤU TRÚC ĐIỀU KHIỂN

Giới thiệu

Một chương trình bao gồm nhiều câu lệnh. Thông thường các câu lệnh được thực hiện một cách lần lượt theo thứ tự mà chúng được viết ra. Các cấu trúc điều khiển cho phép thay đổi trật tự nói trên, do đó máy có thể nhảy thực hiện một câu lệnh khác ở một ví trí trước hoặc sau câu lệnh hiện thời.

Xét về mặt công dụng, có thể chia các cấu trúc điều khiển thành các nhóm chính:

- Nhảy không có điều kiện.
- Rẽ nhánh.
- Tổ chức chu trình.
- Ngoài ra còn một số toán tử khác có chức năng bổ trợ như break, continue.

1. Mục tiêu:

- Trình bày được lệnh có cấu trúc;
- Vận dụng được các lệnh cấu trúc: cấu trúc lựa chọn, cấu trúc lặp xác định và lặp vô định;
- Vận dụng được các lệnh bẻ vòng lặp;
- Thực hiện các thao tác an toàn với máy tính.

2. Nội dung chương:

2.1 Khái niệm về lệnh cấu trúc

Một chương trình bao gồm nhiều câu lệnh. Thông thường các câu lệnh được thực hiện một cách lần lượt theo thứ tự mà chúng được viết ra. Các cấu trúc điều khiển cho phép thay đổi trật tự nói trên, do đó máy có thể nhảy thực hiện một câu lệnh khác ở một ví trí trước hoặc sau câu lệnh hiện thời.

Xét về mặt công dụng, có thể chia các cấu trúc điều khiển thành các nhóm chính:

- Nhảy không có điều kiện.
- Rẽ nhánh.
- Tổ chức chu trình.
- Ngoài ra còn một số toán tử khác có chức năng bổ trợ như break, continue.

2.2 Các lệnh cấu trúc lựa chọn

Các cấu trúc suy luận cơ bản của giải thuật

Giải thuật được thiết kế theo ba cấu trúc suy luận cơ bản sau đây:

+ Tuần tự (Sequential):

Các công việc được thực hiện một cách tuần tự, công việc này nối tiếp công việc kia.

+ Cấu trúc lựa chọn (Selection)

Lựa chọn một công việc để thực hiện căn cứ vào một điều kiện nào đó. Có một số dạng như sau:

- *Cấu trúc 1*: Nếu < điều kiện> (đúng) thì thực hiện <công việc>

- *Cấu trúc 2*: Nếu < điều kiện> (đúng) thì thực hiện <công việc 1>, ngược lại (điều kiện sai) thì thực hiện <công việc 2>

- *Cấu trúc 3*: Trường hợp < i> thực hiện <công việc i>

+ Cấu trúc lặp (Repeating)

Thực hiện lặp lại một công việc không hoặc nhiều lần căn cứ vào một điều kiện nào đó.

Có hai dạng như sau:

- *Lặp xác định*: là loại lặp mà khi viết chương trình, người lập trình đã xác định được công việc sẽ lặp bao nhiêu lần.
- *Lặp không xác định*: là loại lặp mà khi viết chương trình người lập trình chưa xác định được công việc sẽ lặp bao nhiêu lần. Số lần lặp sẽ được xác định khi chương trình thực thi.

Trong một số trường hợp người ta cũng có thể dùng các cấu trúc này để diễn tả một giải thuật.

2.2.1 Cấu trúc có điều kiện

+ Lệnh if-else

Toán tử if cho phép lựa chọn chạy theo một trong hai nhánh tuỳ thuộc vào sự bằng không và khác không của biểu thức. Nó có hai cách viết sau :

+ Lệnh if-else

if (biểu thức)khối lệnh 1;/* Dạng một */	if (biểu thức)khối lệnh 1;elsekhối lệnh 2 ;/* Dạng hai */

Hoạt động của biểu thức dạng 1

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau lệnh if trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện ngay các lệnh tiếp sau lệnh if trong chương trình.

Hoạt động của biểu thức dạng 2

Máy tính giá trị của biểu thức. Nếu biểu thức đúng (biểu thức có giá trị khác 0) máy sẽ thực hiện khối lệnh 1 và sau đó sẽ thực hiện các lệnh tiếp sau khối lệnh 2 trong chương trình. Nếu biểu thức sai (biểu thức có giá trị bằng 0) thì máy bỏ qua khối lệnh 1 mà thực hiện khối lệnh 2 sau đó thực hiện tiếp các lệnh tiếp sau khối lệnh 2 trong chương trình.

Ví dụ: Chương trình nhập vào hai số a và b, tìm max của hai số rồi in kết quả lên màn hình. Chương trình có thể viết bằng cả hai cách trên như sau :

```
#include "stdio.h"
main()
{
    float a,b,max;
    printf("\n Cho a=");
    scanf("%f",&a);
    printf("\n Cho b=");
    scanf("%f",&b);
    max=a;
    if (b>max) max=b;
```

```
        printf(" \n Max cua hai so a=%8.2f va b=%8.2f la Max=%8.2f",a,b,max);
}

#include "stdio.h"
main()
{
        float a,b,max;
        printf("\n Cho a=");
        scanf("%f",&a);
        printf("\n Cho b=");
        scanf("%f",&b);
        if (a>b) max=a;
        else max=b;
                printf("\n Max cua hai so a=%8.2f va b=%8.2f la Max=%8.2f",a,b,max);
}
```

Sự lồng nhau của các toán tử if:

C cho phép sử dụng các toán tử if lồng nhau có nghĩa là trong các khối lệnh (1 và 2) ở trên có thể chứa các toán tử if - else khác. Trong trường hợp này, nếu không sử dụng các dấu đóng mở ngoặc cho các khối thì sẽ có thể nhầm lẫn giữa các if-else.

Chú ý là máy sẽ gắn toán tử else với toán tử if không có else gần nhất.

Chẳng hạn như đoạn chương trình ví dụ sau :

```
if ( n>0 )/* if thứ nhất*/
        if ( a>b )/* if thứ hai*/
                z=a;
        else
                z=b;
```

thì else ở đây sẽ đi với if thứ hai.

Đoạn chương trình trên tương đương với:

```
if (n>0)/* if thứ nhất*/
        {
                if ( a>b )/* if thứ hai*/
                        z=a;
                else
                        z=b;
        }
```

Trường hợp ta muốn else đi với if thứ nhất ta viết như sau:

```
if ( n>0 )/* if thứ nhất*/
        {
                if ( a>b )/* if thứ hai*/
                        z=a;
        }
else
```

```
        z=b;
```

Lệnh else-iKhi muốn thực hiện một trong n quyết định ta có thể sử dụng cấu trúc sau:

```
if ( biểu thức 1 )
   khối lệnh 1;
else
   if ( biểu thức 2 )
        khối lệnh 2;
        else
                if ( biểu thức n-1 )
                        khối lệnh n-1;
                else
                        khối lệnh n;
```

Trong cấu trúc này, máy sẽ đi kiểm tra từ biểu thức 1 trở đi đến khi gặp biểu thức nào có giá trị khác 0.

Nếu biểu thức thứ i (1,2, ...n-1) có giá trị khác 0, máy sẽ thực hiện khối lệnh i, rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Nếu trong cả n-1 biểu thức không có biểu thức nào khác 0, thì máy sẽ thực hiện khối lệnh n rồi sau đó đi thực hiện lệnh nằm tiếp theo khối lệnh n trong chương trình.

Ví dụ: Chương trình giải phương trình bậc hai.

```
#include "stdio.h"
void main()
{
float a,b,c,d,x1,x2;
printf("\n Nhap a, b, c:");
scanf("%f%f%f,&a&b&c);
d=b*b-4*a*c;
if (d<0.0)
        printf("\n Phuong trinh vo nghiem ");
else
        if (d==0.0)
                printf("\n Phuong trinh co nghiem kep x1,2=%8.2f",-b/(2*a));
        else
        {
                printf("\n Phuong trinh co hai nghiem ");
                printf("\n x1=%8.2f",(-b+sqrt(d))/(2*a));
                printf("\n x2=%8.2f",(-b-sqrt(d))/(2*a));
        }
```

Lệnh nhảy không điều kiện - toán tử goto

Nhãn có cùng dạng như tên biến và có dấu : đứng ở phía sau. Nhãn có thể được gán cho bất kỳ câu lệnh nào trong chương trình

Ví dụ: ts: s=s++; thì ở đây **ts** là nhãn của câu lệnh gán s=s++.

Toán tử goto có dạng: goto nhãn;

Khi gặp toán tử này máy sẽ nhảy tới câu lệnh có nhãn viết sau từ khoá goto.

Khi dùng toán tử goto cần chú ý:

Câu lệnh goto và nhãn cần nằm trong một hàm, có nghĩa là toán tử goto chỉ cho phép nhảy từ vị trí này đến vị trí khác trong thân một hàm và không thể dùng để nhảy từ một hàm này sang một hàm khác.

Không cho phép dùng toán tử goto để nhảy từ ngoài vào trong một khối lệnh. Tuy nhiên việc nhảy từ trong một khối lệnh ra ngoài là hoàn toàn hợp lệ.

Ví dụ như đoạn chương trình sau là sai.

```
goto n1;
{
    n1: printf("\n Gia tri cua N la: ");
}
```

Ví dụ : Tính tổng s=1+2+3+....+10

```
#include "stdio.h"
main()
{
        int s,i;
        i=s=0;
        tong:++i;
        s=s+i;
        if (i<10) goto tong;
                printf("\n tong s=%d",s);
}
```

Cấu trỳc rẽ nhỏnh - toỏn tử switch

Là cấu trỳc tạo nhiều nhỏnh đặc biệt. Nú căn cứ vào giỏ trị một biểu thức nguyờn để để chọn một trong nhiều cỏch nhảy.

Cấu trỳc tổng quỏt của nú là:

```
switch (biểu thức nguyên)
{
   case n1
        khối lệnh 1
   case n2
        khối lệnh 2
        .....
   case nk
        khối lệnh k
        [default khối lệnh k+1]
}
```

Với ni là các số nguyên, hằng ký tự hoặc biểu thức hằng. Các ni cần có giá trị khác nhau. Đoạn chương trình nằm giữa các dấu { } gọi là thân của toán tử switch. default là một thành phần không bắt buộc phải có trong thân của switch.

Sự hoạt động của toán tử switch phụ thuộc vào giá trị của biểu thức viết trong dấu ngoặc () như sau :

Khi giá trị của biểu thức này bằng ni, máy sẽ nhảy tới các câu lệnh có nhãn là case ni.

Khi giá trị biểu thức khác tất cả các ni thì cách làm việc của máy lại phụ thuộc vào sự có mặt hay không của lệnh default như sau:

Khi có default máy sẽ nhảy tới câu lệnh sau nhãn default.

Khi không có default máy sẽ nhảy ra khỏi cấu trúc switch.

Máy sẽ nhảy ra khỏi toán tử switch khi nó gặp câu lệnh break hoặc dấu ngoặc nhọn đóng cuối cùng của thân switch. Ta cũng có thể dùng câu lệnh goto trong thân của toán tử switch để nhảy tới một câu lệnh bất kỳ bên ngoài switch.

Khi toán tử switch nằm trong thân một hàm nào đó thì ta có thể sử dụng câu lệnh return trong thân của switch để ra khỏi hàm này (lệnh return sẽ đề cập sau).

Khi máy nhảy tới một câu lệnh nào đó thì sự hoạt động tiếp theo của nó sẽ phụ thuộc vào các câu lệnh đứng sau câu lệnh này. Như vậy nếu máy nhảy tới câu lệnh có nhãn case ni thì nó có thể thực hiện tất cả các câu lệnh sau đó cho tới khi nào gặp câu lệnh break, goto hoặc return. Nói cách khác, máy có thể đi từ nhóm lệnh thuộc case ni sang nhóm lệnh thuộc case thứ ni+1. Nếu mỗi nhóm lệnh được kết thúc bằng break thì toán tử switch sẽ thực hiện chỉ một trong các nhóm lệnh này.

Ví dụ : Lập chương trình phân loại học sinh theo điểm sử dụng cấu trúc switch:

```
#include "stdio.h"
   main()
   {
   int diem;
   tt: printf("\nVao du lieu :");
   printf("\n Diem =");
   scanf("%d",&diem);
   switch (diem)
   {
   case 0:
   case 1:
   case 2:
   case 3:printf("Kem\n");break;
   case 4:printf("Yeu\n");break;
   case 5:
   case 6:printf("Trung binh\n");break;
   case 7:
   case 8:printf("Kha\n");break;
   case 9:
   case 10:printf("Gioi\n");break;
   default:printf(Vao sai\n);
    }
```

```
    printf("Tiep tuc 1, dung 0 :")
    scanf("%d",&diem);
    if (diem==1) goto tt;
    getch();
    return;
}
```

2.3 Các câu lệnh lặp

2.3.1 Cấu trúc lặp

Cấu trúc lặp với toán tử while và for

2.3.1.1 Cấu trúc lặp với toán tử while

Toán tử while dùng để xây dựng chu trình lặp dạng :

while (biểu thức)

Lệnh hoặc khối lệnh;

Như vậy toán tử while gồm một biểu thức và thân chu trình. Thân chu trình có thể là một lệnh hoặc một khối lệnh.

Hoạt động của chu trình như sau :

Máy xác định giá trị của biểu thức, tuỳ thuộc giá trị của nó máy sẽ chọn cách thực hiện như sau :

Nếu biểu thức có giá trị 0 (biểu thức sai), máy sẽ ra khỏi chu trình và chuyển tới thực hiện câu lệnh tiếp sau chu trình trong chương trình.

Nếu biểu thức có giá trị khác không (biểu thức đúng), máy sẽ thực hiện lệnh hoặc khối lệnh trong thân của while. Khi máy thực hiện xong khối lệnh này nó lại thực hiện xác định lại giá trị biểu thức rồi làm tiếp các bước như trên.

Trong các dấu ngoặc ( ) sau while chẳng những có thể đặt một biểu thức mà còn có thể đặt một dãy biểu thức phân cách nhau bởi dấu phảy. Tính đúng sai của dãy biểu thức được hiểu là tính đúng sai của biểu thức cuối cùng trong dãy.

Bên trong thân của một toán tử while lại có thể sử dụng các toán tử while khác. bằng cách đó ta đi xây dựng được các chu trình lồng nhau.

Khi gặp câu lệnh break trong thân while, máy sẽ ra khỏi toán tử while sâu nhất chứa câu lệnh này.

Trong thân while có thể sử dụng toán tử goto để nhảy ra khỏi chu trình đến một vị trí mong muốn bất kỳ. Ta cũng có thể sử dụng toán tử return trong thân while để ra khỏi một hàm nào đó.

Ví dụ :

Chương trình tính tích vô hướng của hai véc tơ x và y :

Cách 1 :

```
#include "stdio.h"
    float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
    main()
    {
    float s=0;
    int i=-1;
```

```
    while (++i<4)
    s+=x[i]*y[i];
    printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
    }
```

Cách 2 :

```
#include "stdio.h"
    float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
    main()
    {
    float s=0;
    int i=0;
    while (1)
    {
    s+=x[i]*y[i];
    if (++i>=4) goto kt;
    }
    kt:printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
    }
```

Cách 3 :

```
#include "stdio.h"
    float x[]={2,3.4,4.6,21}, y[]={24,12.3,56.8,32.9};
    main()
    {
    float s=0;
    int i=0;
    while (s+=x[i]*y[i], ++i<=3 );
    printf("\n Tich vo huong hai vec to x va y la :%8.2f",s);
    }
```

2.3.1.1 Cấu trúc lặp với toán tử for

Toán tử for dùng để xây dựng cấu trúc lặp có dạng sau :

for (biểu thức 1; biểu thức 2; biểu thức 3)

Lệnh hoặc khối lệnh ;

Toán tử for gồm ba biểu thức và thân for. Thân for là một câu lệnh hoặc một khối lệnh viết sau từ khoá for. Bất kỳ biểu thức nào trong ba biểu thức trên có thể vắng mặt nhưng phải giữ dấu ; .

Thông thường biểu thức 1 là toán tử gán để tạo giá trị ban đầu cho biến điều khiển, biểu thức 2 là một quan hệ logic biểu thị điều kiện để tiếp tục chu trình, biểu thức ba là một toán tử gán dùng để thay đổi giá trị biến điều khiển.

Hoạt động của toán tử for

Toán tử for hoạt động theo các bước sau :

Xác định biểu thức 1

Xác định biểu thức 2

Tuỳ thuộc vào tính đúng sai của biểu thức 2 để máy lựa chọn một trong hai nhánh :

Nếu biểu thức hai có giá trị 0 (sai), máy sẽ ra khỏi for và chuyển tới câu lệnh sau thân for.

Nếu biểu thức hai có giá trị khác 0 (đúng), máy sẽ thực hiện các câu lệnh trong thân for.

Tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình.

Nếu biểu thức 2 vắng mặt thì nó luôn được xem là đúng. Trong trường hợp này việc ra khỏi chu trình for cần phải được thực hiện nhờ các lệnh break, goto hoặc return viết trong thân chu trình.

Trong dấu ngoặc tròn sau từ khoá for gồm ba biểu thức phân cách nhau bởi dấu ;. Trong mỗi biểu thức không những có thể viết một biểu thức mà có quyền viết một dãy biểu thức phân cách nhau bởi dấu phảy. Khi đó các biểu thức trong mỗi phần được xác định từ trái sang phải. Tính đúng sai của dãy biểu thức được tính là tính đúng sai của biểu thức cuối cùng trong dãy này.

Trong thân của for ta có thể dùng thêm các toán tử for khác, vì thế ta có thể xây dựng các toán tử for lồng nhau.

Khi gặp câu lệnh break trong thân for, máy ra sẽ ra khỏi toán tử for sâu nhất chứa câu lệnh này. Trong thân for cũng có thể sử dụng toán tử goto để nhảy đến một ví trí mong muốn bất kỳ.

Ví dụ 1:

Nhập một dãy số rồi đảo ngược thứ tự của nó.

Cách 1:

```
#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
int i,j;
float c;
for (i=0,j=n-1;i<j;++i,--j)
{
c=x[i];x[i]=x[j];x[j]=c;
}
fprintf(stdprn,“\n Day so dao la \n\n”);
for (i=0;i<n;++i)
fprintf(stdprn,“%8.2f”,x[i]);
}
```

Cách 2 :

```
#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
```

```
{
int i,j;
float c;
for (i=0,j=n-1;i<j;c=x[i],x[i]=x[j],x[j]=c,++i,--j)
fprintf(stdprn,“\n Day so dao la \n\n”);
for (i=0;++i<n;)
fprintf(stdprn,“%8.2f”,x[i]);
}
```

Cách 3 :

```
#include “stdio.h”
float x[]={1.3,2.5,7.98,56.9,7.23};
int n=sizeof(x)/sizeof(float);
main()
{
int i=0,j=n-1;
float c;
for ( ; ; )
{
c=x[i];x[i]=x[j];x[j]=c;
if (++i>--j) break;
}
fprintf(stdprn,“\n Day so dao la \n\n”);
for (i=-1;i++<n-1; fprintf(stdprn,“%8.2f”,x[i]));
}
```

Ví dụ 2:

Tính tích hai ma trận mxn và nxp.

```
c#include "stdio.h"
float x[3][2],y[2][4],z[3][4],c;
main()
{
int i,j;
printf("\n nhap gia tri cho ma tran X ");
for (i=0;i<=2;++i)
for (j=0;j<=1;++j)
{
printf("\n x[%d][%d]=",i,j);
scanf("%f",&c);
x[i][j]=c;
}
printf("\n nhap gia tri cho ma tran Y ");
for (i=0;i<=1;++i)
for (j=0;j<=3;++j)
```

```
{
printf("\n y[%d][%d]=",i,j);
scanf("%f",&c);
y[i][j]=c;
}
for (i=0;i<=3;++i)
for (j=0;j<=4;++j)
z[i][j]
}
```

2.3.2 Chu trình do-while

Khác với các toán tử while và for, việc kiểm tra điều kiện kết thúc đặt ở đầu chu trình, trong chu trình do while việc kiểm tra điều kiện kết thúc đặt cuối chu trình. Như vậy thân của chu trình bao giờ cũng được thực hiện ít nhất một lần.

Chu trình do while có dạng sau :

do

Lệnh hoặc khối lệnh;

while (biểu thức);

Lệnh hoặc khối lệnh là thân của chu trình có thể là một lệnh riêng lẻ hoặc là một khối lệnh.

Hoạt động của chu trình như sau

Máy thực hiện các lệnh trong thân chu trình.

Khi thực hiện xong tất cả các lệnh trong thân của chu trình, máy sẽ xác định giá trị của biểu thức sau từ khoá while rồi quyết định thực hiện như sau :

Nếu biểu thức đúng (khác 0) máy sẽ thực hiện lặp lại khối lệnh của chu trình lần thứ hai rồi thực hiện kiểm tra lại biểu thức như trên.

Nếu biểu thức sai (bằng 0) máy sẽ kết thúc chu trình và chuyển tới thực hiện lệnh đứng sau toán tử while.

Những điều lưu ý với toán tử while ở trên hoàn toàn đúng với do while.

Ví dụ :

Đoạn chương trình xác định phần tử âm đầu tiên trong các phần tử của mảng x.

```
#include "stdio.h"
float x[5],c;
main()
{
int i=0;
printf("\n nhap gia tri cho ma tran x ");
for (i=0;i<=4;++i)
{
printf("\n x[%d]=",i);
scanf("%f",&c);
y[i]=c;
}
```

```
do
++i;
while (x[i]>=0 && i<=4);
if (i<=4)
printf("\n Phan tu am dau tien = x[%d]=%8.2f",i,x[i]);
else
printf("\n Mang khong có phan tu am ");
}
```

2.4 Các lệnh chuyển điều khiển

2.4.1 Các lệnh điều khiển vòng lặp

Các lệnh điều khiển vòng lặp thay đổi sự thực thi lệnh từ dãy thông thường của nó. Khi sự thực thi lệnh rời khỏi một phạm vi, tất cả các đối tượng tự động mà được tạo ra trong phạm vi đó bị hủy.

C hỗ trợ các lệnh điều khiển vòng lặp sau đây:

Lệnh điều khiển	Mô tả
Lệnh break	Kết thúc **vòng lặp** hoặc lệnh **switch** và chuyển sang thực thi vòng lặp hoặc lệnh switch ngay sau nó.
Lệnh continue	Khi gặp lệnh này thì chương trình sẽ bỏ qua các câu lệnh ở dưới nó (trong cùng một câu lệnh lặp) để thực hiện vòng lặp mới.
Lệnh goto	Chuyển tới lệnh được gán. Mặc dù vậy, nó được khuyên rằng không nên sử dụng lệnh goto trong chương trình của bạn.

2.5 Kết hợp các cấu trúc điều khiển trong chương trình

2.5.1 Câu lệnh break

Câu lệnh break cho phép ra khỏi các chu trình với các toán tử for, while và switch. Khi có nhiều chu trình lồng nhau, câu lệnh break sẽ đưa máy ra khỏi chu trình bên trong nhất chứa nó không cần điều kiện gì. Mọi câu lệnh break có thể thay bằng câu lệnh goto với nhãn thích hợp.

Ví dụ :

Biết số nguyên dương n sẽ là số nguyên tố nếu nó không chia hết cho các số nguyên trong khoảng từ 2 đến căn bậc hai của n. Viết đoạn chương trình đọc vào số nguyên dương n, xem n có là số nguyên tố.

```
# include "stdio.h"
# include "math.h"
unsigned int n;
main()
{
int i,nt=1;
printf("\n cho n=");
scanf("%d",&n);
for (i=2;i<=sqrt(n);++i)
```

```
if ((n % i)==0)
{
nt=0;
break;
}
if (nt)
printf("\n %d la so nguyen to",n);
else
printf("\n %d khong la so nguyen to",n);
}
```

2.5.2 Câu lệnh continue

Trái với câu lệnh break, lệnh continue dùng để bắt đầu một vòng mới của chu trình chứa nó. Trong while và do while, lệnh continue chuyển điều khiển về thực hiện ngay phần kiểm tra, còn trong for điều khiển được chuyển về bước khởi đầu lại (tức là bước : tính biểu thức 3, sau đó quay lại bước 2 để bắt đầu một vòng mới của chu trình).

Lệnh continue chỉ áp dụng cho chu trình chứ không áp dụng cho switch.

Ví dụ :

Viết chương trình để từ một nhập một ma trận a sau đó :

Tính tổng các phần tử dương của a.

Xác định số phần tử dương của a.

Tìm cực đại trong các phần tử dương của a.

```
#include "stdio.h"
float a[3[4];
main()
{
int i,j,soptd=0;
float tongduong=0,cucdai=0,phu;
for (i=0;i<3;++i)
for (j=0;i<4;++j)
{
printf("\n a[%d][%d]=",i,j );
scanf("%f",&phu);
a[i][j]=phu;
if (a[i][j]<=0) continue;
tongduong+=a[i][j];
if (cucdai<a[i][j]) cucdai=a[i][j];
++soptd;}
printf("\n So phan tu duong la : %d",soptd);
printf("\n Tong cac phan tu duong la : %8.2f",tongduong);
printf("\n Cuc dai phan tu duong la : %8.2f",cucdai);
}
```

2.6 Kiểm tra

a) Tính tổng các phần tử của mảng *N* giá trị nguyên

Bài toán: Cho một mảng *N* phần tử kiểu nguyên cho trước, tìm tổng các giá trị của mảng.

Ý tưởng: Khởi tạo giá trị ban đầu bằng không. Duyệt các phần tử cộng giá trị vào biến lưu trữ.

Thuật toán:

Dữ liệu vào: arrData[N] phần tử

Dữ liệu ra: cSum tổng các giá trị của mảng

Mô tả:

```
cSum=0;
for(i=1 -> N)
        cSum=cSum+arrData[i];
```

Cài đặt chương trình:

Theo thuật toán phần tử đầu tiên của mảng là *1*, và duyệt từ *1* đến *N*, nhưng trong cài đặt ngôn ngữ C do chỉ số phần tử của mảng được xét từ *0* đến *N-1* nên trong chương trình vòng lặp sẽ tiến hành từ giá trị *1* đến giá trị *N-1*;

```
#include <stdio.h>
#define N 5
int main()
{
    int arrData[N]={3,5,1,2,4};
    int i, cSum;
    cSum=0;
    for(i=0;i<=N-1;i++) //phan tu ket thuc cua mang la N-1
    {
        cSum=cSum+arrData[i];
    }
    printf("Tong gia tri:%d",cSum);
    return 0;
}
```

b) Kiểm tra tính dương của dãy

Bài toán: Cho một mảng *N* phần tử kiểu nguyên cho trước, kiểm tra xem dãy (các phần tử của mảng) có là số dương (>0) hay không.

Ý tưởng: Giả sử ban đầu dãy chứa các phần tử dương. Duyệt các phần tử của dãy, nếu phát hiện phần tử không thỏa mãn điều kiện dương, thay đổi trạng thái và kết thúc kiểm tra.

Thuật toán:

Dữ liệu vào: arrData[N] phần tử

Dữ liệu ra: Kết luận về tính dương của dãy

```
iPosi=1;
for(i=1 -> N)
        if(arrData[i]<=0)
```

```
                    iPosi=0;
        if(iPosi==1)
                Kết luận dãy dương
        else
                Kết luận dãy không dương
```

Cài đặt chương trình:

Tương tự như giải thích ở trên, trong cài đặt ở đây chỉ số mảng là *0* đến *N-1*, vì vậy vòng lặp sẽ thực hiện từ *0* đến *N-1*;

```
#include <stdio.h>
#define N 5
int main()
{
    int iPosi;
    int arrData[N]={3,5,1,2,4};
    int i;
    iPosi=1;
    for(i=0;i<=N-1;i++)
    {
        if(arrData[i]<=0)
        {
            iPosi=0;
            break;
        }
    }
    if(iPosi==1)
    {
        printf("Day toan duong");
    }
    else
    {
        printf("Day khong thoa man toan duong");
    }
    getch();
    return 0;
}
```

BÀI TẬP

Bài 1 : Tìm số lớn nhất trong 2 số thực

Bài 2 : Tìm số lớn nhất trong 4 số nguyên

Bài 3 : Viết chương trình giải phương trình bậc 2

Bài 4 : Viết chương trình tính diện tích của hình học sơ cấp

Bài 5 : Viết chương trình in ra phần tử âm đầu tiên

CHƯƠNG 4. HÀM VÀ THỦ TỤC

Giới thiệu

Hàm và thủ tục là một trong những cấu trúc cực kỳ quan trọng trong lập trình. Việc sử dụng hàm và thủ tục trong lập trình sẽ diễn ra rất thường xuyên. Vì vậy nắm, hiểu "hàm và thủ tục là gì? cách khai báo và sử dụng hàm trong C" sẽ đóng vai trò then chốt trong việc học lập trình của bạn. Vậy thì bây giờ, chúng ta hãy cùng nhau tìm hiểu vấn đề này nhé.

1. Mục tiêu:

- Trình bày được khái niệm hàm, thủ tục;
- Trình bày được qui tắc xây dựng hàm, thủ tục và vận dụng được khi thiết kế xây dựng chương trình;
- Phân biệt được cách sử dụng tham số, tham biến;
- Sử dụng được các lệnh kết thúc và lấy giá trị trả về của hàm;
- Thực hiện các thao tác an toàn với máy tính

2. Nội dung:

2.1 Khái niệm chương trình con

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

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

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

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

Ví dụ 1: Ta có hàm max để tìm số lớn giữa 2 số nguyên a, b như sau:

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

Ví dụ 2: Ta có chương trình chính (hàm main) dùng để nhập vào 2 số nguyên a,b và in ra màn hình số lớn trong 2 số

```
#include <stdio.h>
#include <conio.h>
int max(int a, int b)
{
return (a>b) ? a:b;
}
```

```
int main()
{
int a, b, c;
printf("\n Nhap vao 3 so a, b,c ");
scanf("%d%d%d",&a,&b,&c);
printf("\n So lon la %d",max(a, max(b,c)));
getch();
return 0;
}
```

2.1.1 Hàm thư viện

Hàm thư viện là những hàm đã được định nghĩa sẵn trong một thư viện nào đó, muốn sử dụng các hàm thư viện thì phải khai báo thư viện trước khi sử dụng bằng lệnh *#include <tên thư viện.h>*

Một số thư viện:

alloc.h, assert.h, bcd.h, bios.h, complex.h, conio.h, ctype.h, dir.h, dirent.h, dos.h, errno.h, fcntl.h, float.h, fstream.h, grneric.h, graphics.h, io.h, iomanip.h, iostream.h, limits.h, locale.h, malloc.h, math.h, mem.h, process.h, setjmp.h, share.h, signal.h, stdarg.h, stddef.h, stdio.h, stdiostr.h, stdlib.h, stream.h, string.h, strstrea.h, sys\stat.h, sys\timeb.h, sys\types.h, time.h, values.h,

Ý nghĩa của một số thư viện thường dùng:

1. stdio.h : Thư viện chứa các hàm vào/ ra chuẩn (**st**an**d**ard **i**nput/**o**utput). Gồm các hàm **printf(), scanf(), getc(), putc(), gets(), puts(), fflush(), fopen(), fclose(), fread(), fwrite(), getchar(), putchar(), getw(), putw()…**

2. conio.h : Thư viện chứa các hàm vào ra trong chế độ DOS (DOS console). Gồm các hàm **clrscr(), getch(), getche(), getpass(), cgets(), cputs(), putch(), clreol(),…**

3. math.h: Thư viện chứa các hàm tính toán gồm các hàm **abs(), sqrt(), log(). log10(), sin(), cos(), tan(), acos(), asin(), atan(), pow(), exp(),…**

4. alloc.h: Thư viện chứa các hàm liên quan đến việc quản lý bộ nhơ. Gồm các hàm **calloc(), realloc(), malloc(), free(), farmalloc(), farcalloc(), farfree(), …**

5. io.h: Thư viện chứa các hàm vào ra cấp thấp. Gồm các hàm **open(), _open(), read(), _read(), close(), _close(), creat(), _creat(), creatnew(), eof(), filelength(), lock(),…**

6. graphics.h: Thư viện chứa các hàm liên quan đến đồ họa. Gồm **initgraph(), line(), circle(), putpixel(), getpixel(), setcolor(), …**

Muốn sử dụng các hàm thư viện thì ta phải xem cú pháp của các hàm và sử dụng theo đúng cú pháp (xem trong phần trợ giúp của Turbo C).

2.1.2 Hàm người dùng

Hàm người dùng là những hàm do người lập trình tự tạo ra nhằm đáp ứng nhu cầu xử lý của mình.

2.2 Cấu trúc chương trình có sử dụng chương trình con

+ Định nghĩa hàm

Cấu trúc của một hàm tự thiết kế:

<*kiểu kết quả*> Tên hàm ([< *kiểu t số*> < *tham số* >][,< *kiểu t số* >< *tham số* >][…])

{

[Khai báo biến cục bộ và các câu lệnh thực hiện hàm]

[return [<Biểu thức>];]

}

+ Giải thích:

- *Kiểu kết quả:* là kiểu dữ liệu của kết quả trả về, có thể là : int, byte, char, float, void… Một hàm có thể có hoặc không có kết quả trả về. Trong trường hợp *hàm không có kết quả trả về ta nên sử dụng kiểu kết quả là void*.

- *Kiểu t số*: là kiểu dữ liệu của tham số.

- *Tham số:* là tham số truyền dữ liệu vào cho hàm, một hàm có thể có hoặc không có tham số. *Tham số này gọi là tham số hình thức, khi gọi hàm chúng ta phải truyền cho nó các tham số thực tế*. Nếu có nhiều tham số, mỗi tham số phân cách nhau dấu phẩy (,).

- Bên trong thân hàm (phần giới hạn bởi cặp dấu {}) là các khai báo cùng các câu lệnh xử lý. Các khai báo bên trong hàm được gọi là các khai báo cục bộ trong hàm và các khai báo này chỉ tồn tại bên trong hàm mà thôi.

- Khi định nghĩa hàm, ta thường sử dụng câu lệnh return để trả về kết quả thông qua tên hàm.

Lệnh return dùng để thoát khỏi một hàm và có thể trả về một giá trị nào đó.

Cú pháp:

return ; /*không trả về giá trị*/

return <biểu thức>; /*Trả về giá trị của biểu thức*/

return (<biểu thức>); /*Trả về giá trị của biểu thức*/

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

Ví dụ 1: Viết hàm tìm số lớn giữa 2 số nguyên a và b

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

Ví dụ 2: Viết hàm tìm ước chung lớn nhất giữa 2 số nguyên a, b. Cách tìm: đầu tiên ta giả sử UCLN của hai số là số nhỏ nhất trong hai số đó. Nếu điều đó không đúng thì ta giảm đi một đơn vị và cứ giảm như vậy cho tới khi nào tìm thấy UCLN

```
int ucln(int a, int b)
{
int u;
if (a<b)
u=a;
```

```
else
u=b;
while ((a%u !=0) || (b%u!=0))
u--;
return u;
}
```

Sử dụng hàm

Một hàm khi định nghĩa thì chúng vẫn chưa được thực thi trừ khi ta có một lời gọi đến hàm đó.

Cú pháp gọi hàm: <Tên hàm>([Danh sách các tham số])

Ví dụ: Viết chương trình cho phép tìm ước số chung lớn nhất của hai số tự nhiên.

```
#include<stdio.h>
unsigned int ucln(unsigned int a, unsigned int b)
{
unsigned int u;
if (a<b)
u=a;
else
u=b;
while ((a%u !=0) || (b%u!=0))
u--;
return u;
}
int main()
{
unsigned int a, b, UC;
printf("Nhap a,b: ");scanf("%d%d",&a,&b);
UC = ucln(a,b);
printf("Uoc chung lon nhat la: ", UC);
return 0;
}
```

Lưu ý: Việc gọi hàm là một phép toán, không phải là một phát biểu.

Nguyên tắc hoạt động của hàm

Trong chương trình, khi gặp một lời gọi hàm thì hàm bắt đầu thực hiện bằng cách chuyển các lệnh thi hành đến hàm được gọi. Quá trình diễn ra như sau:

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

2.2.1 Truyền tham số cho hàm:

Mặc nhiên, việc truyền tham số cho hàm trong C là truyền theo giá trị; nghĩa là các giá trị thực (tham số thực) không bị thay đổi giá trị khi truyền cho các tham số hình thức

Ví dụ 1: Giả sử ta muốn in ra nhiều dòng, mỗi dòng 50 ký tự nào đó. Để đơn giản ta viết một hàm, nhiệm vụ của hàm này là in ra trên một dòng 50 ký tự nào đó. Hàm này có tên là InKT.

```
#include <stdio.h>
#include <conio.h>
void InKT(char ch)
{
int i;
for(i=1;i<=50;i++) printf("%c",ch);
printf("\n");
}
int main()
{
char c = 'A';
InKT('*'); /* In ra 50 dau * */
InKT('+');
InKT(c);
return 0;
}
```

Lưu ý:

- Trong hàm InKT ở trên, biến ch gọi là tham số hình thức được truyền bằng giá trị (gọi là tham trị của hàm). Các tham trị của hàm coi như là một biến cục bộ trong hàm và chúng được sử dụng như là dữ liệu đầu vào của hàm.

- Khi chương trình con được gọi để thi hành, tham trị được cấp ô nhớ và nhận giá trị là bản sao giá trị của tham số thực. Do đó, mặc dù tham trị cũng là biến, nhưng việc thay đổi giá trị của chúng không có ý nghĩa gì đối với bên ngoài hàm, không ảnh hưởng đến chương trình chính, nghĩa là không làm ảnh hưởng đến tham số thực tương ứng.

Ví dụ 2: Ta xét chương trình sau đây:

```
#include <stdio.h>
#include <conio.h>
int hoanvi(int a, int b)
{
int t;
t=a; /*Đoạn này hoán vị giá trị của 2 biến a, b*/
a=b;
b=t;
printf("\Ben trong ham a=%d , b=%d",a,b);
return 0;
```

```
}
int main()
{
int a, b;
clrscr();
printf("\n Nhap vao 2 so nguyen a, b:");
scanf("%d%d",&a,&b);
printf("\n Truoc khi goi ham hoan vi a=%d ,b=%d",a,b);
hoanvi(a,b);
printf("\n Sau khi goi ham hoan vi a=%d ,b=%d",a,b);
getch();
return 0;
}
```

Kết quả thực hiện chương trình:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Giải thích:

- Nhập vào 2 số 6 và 5 (a=6, b=5)
- Trước khi gọi hàm hoán vị thì a=6, b=5
- Bên trong hàm hoán vị a=5, b=6
- Khi ra khỏi hàm hoán vị thì a=6, b=5

* Lưu ý

Trong đoạn chương trình trên, nếu ta muốn sau khi kết thúc chương trình con giá trị của a, b thay đổi thì ta phải đặt tham số hình thức là các con trỏ, còn tham số thực tế là địa chỉ của các biến.

Lúc này mọi sự thay đổi trên vùng nhớ được quản lý bởi con trỏ là các tham số hình thức của hàm thì sẽ ảnh hưởng đến *vùng nhớ đang được quản lý bởi tham số thực tế tương ứng* (cần để ý rằng vùng nhớ này chính là các biến ta cần thay đổi giá trị).

Người ta thường áp dụng cách này đối với các dữ liệu đầu ra của hàm.

Ví dụ: Xét chương trình sau đây:

```
#include <stdio.h>
#include <conio.h>
long hoanvi(long *a, long *b)
/* Khai báo tham số hình thức *a, *b là các con trỏ kiểu long */
{
long t;
t=*a; /*gán nội dung của x cho t*/
*a=*b; /*Gán nội dung của b cho a*/
*b=t; /*Gán nội dung của t cho b*/
printf("\n Ben trong ham a=%ld , b=%ld",*a,*b);
/*In ra nội dung của a, b*/
return 0;
}
```

```
int main()
{
long a, b;
clrscr();
printf("\n Nhap vao 2 so nguyen a, b:");
scanf("%ld%ld",&a,&b);
printf("\n Truoc khi goi ham hoan vi a=%ld ,b=%ld",a,b);
hoanvi(&a,&b); /* Phải là địa chỉ của a và b */
printf("\n Sau khi goi ham hoan vi a=%ld ,b=%ld",a,b);
getch();
return 0;
}
```

Kết quả thực hiện chương trình:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

Giải thích:

- Nhập vào 2 số 5, 6 (a=5, b=6)
- Trước khi gọi hàm hoanvi thì a=5, b=6
- Trong hàm hoanvi (khi đã hoán vị) thì a=6, b=5
- Khi ra khỏi hàm hoán vị thì a=6, b=6

Lưu ý: Kiểu con trỏ và các phép toán trên biến kiểu con trỏ sẽ nói trong phần sau.

2.2.2 Hàm đệ quy

Định nghĩa

Một hàm được gọi là đệ quy nếu bên trong thân hàm có lệnh gọi đến chính nó.

Ví dụ: Người ta định nghĩa giai thừa của một số nguyên dương n như sau:

n!=1* 2 * 3 *…* (n-1) *n = (n-1)! *n (với 0!=1)

Như vậy, để tính n! ta thấy nếu n=0 thì n!=1 ngược lại thì n!=n * (n-1)!

Với định nghĩa trên thì hàm đệ quy tính n! được viết:

```
#include <stdio.h>
#include <conio.h>
/*Hàm tính n! bằng đệ quy*/
unsigned int giaithua_dequy(int n)
{
if (n==0)
return 1;
else
return n*giaithua_dequy(n-1);
}
/*Hàm tính n! không đệ quy*/
unsigned int giaithua_khongdequy(int n)
{
unsigned int kq,i;
kq=1;
```

```
for (i=2;i<=n;i++)
kq=kq*i;
return kq;
}
int main()
{
int n;
clrscr();
printf("\n Nhap so n can tinh giai thua ");
scanf("%d",&n);
printf("\nGoi ham de quy: %d != %u",n,giaithua_dequy(n));
printf("\nGoi ham khong de quy: %d != %u",
n,giaithua_khongdequy(n));
getch();
return 0;
}
```

Đặc điểm cần lưu ý khi viết hàm đệ quy

- Hàm đệ quy phải có 2 phần:

Phần dừng hay phải có trường hợp nguyên tố. Trong ví dụ ở trên thì trường hợp n=0 là trường hợp nguyên tố.

Phần đệ quy: là phần có gọi lại hàm đang được định nghĩa. Trong ví dụ trên thì phần đệ quy là n>0 thì n! = n * (n-1)!

- Sử dụng hàm đệ quy trong chương trình sẽ làm chương trình dễ đọc, dễ hiểu và vấn đề được nêu bật rõ ràng hơn. Tuy nhiên trong đa số trường hợp thì hàm đệ quy tốn bộ nhớ nhiều hơn và tốc độ thực hiện chương trình chậm hơn không đệ quy.

- Tùy từng bài có cụ thể mà người lập trình quyết định có nên dùng đệ quy hay không (có những trường hợp không dùng đệ quy thì không giải quyết được bài toán).

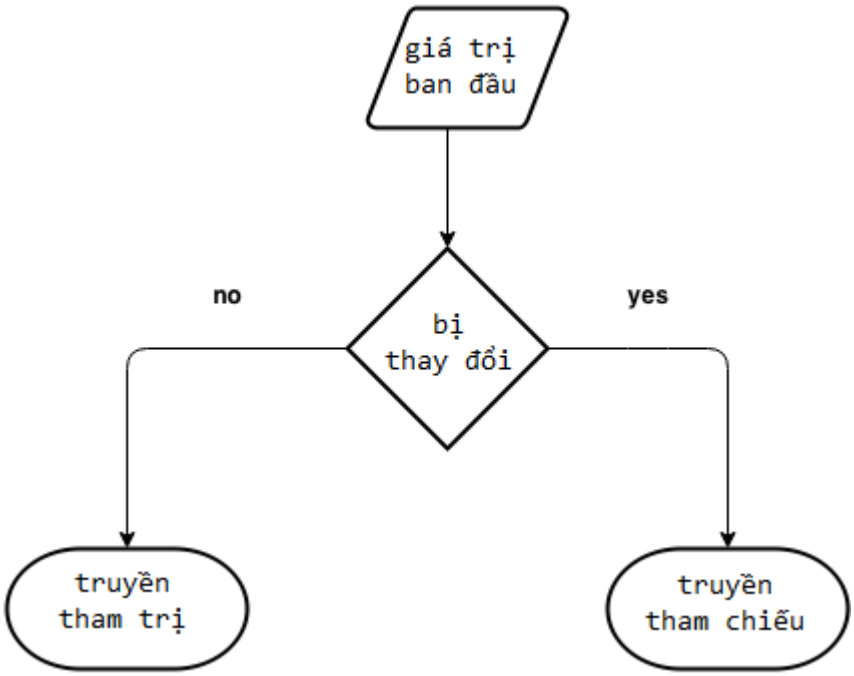
2.3 Các hàm và thủ tục trong ngôn ngữ lập trình

2.4 Tham trị và tham biến

Một điều khá quan trọng trong C đó là **truyền tham chiếu (tham biến) và tham trị vào một phương thức (Function)**.

Truyền tham chiếu là truyền địa chỉ ô nhớ của biến, do đó khi thay đổi giá trị của biến bên trong phương thức thì giá trị của biến cũng bị thay đổi bên ngoài phương thức.

Truyền tham trị là truyền giá trị của biến (không phải là địa chỉ ô nhớ), khi đó phương thức sẽ tự động tạo ra một địa chỉ ô nhớ mới để lưu trữ giá trị này, do đó nó chỉ được thay đổi trong phương thức hiện hành và giá trị của biến không bị thay đổi bên ngoài phương thức hiện hành.



Ví dụ truyền tham trị:

```
#include<stdio.h>
void change(int num) {
    num = num + 1;
}
int main() {
    int x = 100;
    printf("Truoc khi goi phuong thuc x = %d \n", x);
    change(x); // truyen tham tri vao phuong thuc
    printf("Sau khi goi phuong thuc x = %d \n", x);
    return 0;
}
```

Kết quả:

Truoc khi goi phuong thuc x = 100
Sau khi goi phuong thuc x = 100

Trong ví dụ trên, giá trị của biến x không bị thay đổi bên ngoài phương thức change(), mặc dù bên trong phương thức change() chúng ta đã cố gắng thay đổi bằng cách tăng m lên 1.

Ví dụ truyền tham chiếu:

Lưu ý: để hiểu về truyền tham chiếu trong C, bạn phải có hiểu biết về ***con trỏ (pointer) trong C***.

```
#include<stdio.h>
void change(int *num) {
    *num = *num + 1;
```

```
}
int main() {
   int x = 100;
   printf("Truoc khi goi phuong thuc x = %d \n", x);
   change(&x); // truyen tham chieu vao phuong thuc
   printf("Sau khi goi phuong thuc x = %d \n", x);
   return 0;
}
```

Kết quả:

Truoc khi goi phuong thuc x = 100
Sau khi goi phuong thuc x = 101

Trong ví dụ trên, Giá trị của biến x bị thay đổi cả bên trong và bên ngoài phương thức change().

2.5 Biến toàn cục và biến địa phương

Một chương trình viết trong ngôn ngữ C là một dãy các hàm, trong đó có một hàm chính (hàm main()). Hàm chia các bài toán lớn thành các công việc nhỏ hơn, giúp thực hiện những công việc lặp lại nào đó một cách nhanh chóng mà không phải viết lại đoạn chương trình. Thứ tự các hàm trong chương trình là bất kỳ, song chương trình bao giờ cũng đi thực hiện từ hàm main(). Trong C chương trình con chỉ tồn tại dới dạng hàm chứ không có thủ tục.

Hàm có thể xem là một đơn vị độc lập của chương trình. Các hàm có vai trò ngang nhau, vì vậy không có phép xây dựng một hàm bên trong các hàm khác.

Hiểu nôm na một chương trình gồm các hàm trong đó hàm main() là giám đốc.sử dụng được tất cả các nhân viên(các hàm con : thực hiện các công việc nhất định) và khi giám đốc cần có thể sai khiến.

2.5.1 Khai báo và định nghĩa hàm

Xây dựng một hàm bao gồm: khai báo kiểu hàm, đặt tên hàm, khai báo các đối và đưa ra câu lệnh cần thiết để thực hiện yêu cầu đề ra cho hàm. Một hàm được viết theo mẫu sau:

```
<Kiểu_trả_về>  <tên_hàm> ( [khai báo các tham số hình thức])
{
//Khai báo các biến cục bộ
//Các câu lệnh
return  [biểu thức];
}
```

Giải thích:

- <Kiểu_trả_về>: giá trị kiểu dữ liệu của dữ liệu sẽ trả về cho hàm
- <tên_hàm>: tên của hàm mà bạn muốn định nghĩa, được đặt theo qui tắc đặt tên của C
- [khai báo các tham số hình thức]: các tham số hình thức và kiểu của chúng

- [Khai báo các biến cục bộ]: khai báo các biến cục bộ, các biến này chỉ có tác dụng trong nội bộ hàm
- [return]: là lệnh thực hiện gán giá trị trả về cho hàm
- [biểu thức]: là giá trị trả về cho hàm, có thể là biến, hằng, biểu thức nhưng phải có giá trị xác định và có kiểu dữ liệu là kiểu đã khai báo cho hàm.

Ví dụ 1: Hàm tìm giá trị lớn nhất giữa hai giá trị

```
#include <stdio.h>
int tim_max(int a, int b)
{
        if(a>=b)
                return a;
        else
                return b;
}
int main()
{
int Max = tim_max (5,8);
printf ("Max = %d",Max);
return 0;
}
```

kết quả :Max = 8

Lưu ý:

- Hàm có thể có giá trị trả về hoặc không, giá trị trả về phải cùng kiểu với kiểu trả về đã khai báo hàm. Nếu hàm không có giá trị trả về thì đặt từ khóa **void** trước tên hàm để báo hiệu là hàm không cần giá trị trả về cho hàm.
- Khi hàm khai báo không có kiểu ở trước nó thì nó được mặc định là kiểu int.
- Không nhất thiết phải khai báo nguyên mẫu hàm. Nhưng nói chung nên có vì nó cho phép chương trình biên dịch phát hiện lỗi khi gọi.
- Nguyên mẫu của hàm thực chất là dòng đầu tiên của hàm thêm vào dấu;. Tuy nhiên, trong nguyên mẫu có thể bỏ tên các tham số hình thức.

ví dụ :

```
#include <stdio.h>
#include <conio.h>
void A ()
{printf ("Toi la A");}
void B ()
{printf ("nToi la B");}
int main ()
{
//Goi ham A
A();
//Goi ham B
```

```
B();
getch ();
return 0;
}
```
kết quả :
Toi la A
Toi la B

2.5.2 Lời gọi hàm

Cú pháp:

tên hàm ([Danh sách các tham số thực])

Danh sách các tham số thực phải bằng số tham số hình thức và lần lượt chúng có kiểu tương ứng với nhau.

Ví dụ

```
#include <stdio.h>
#include <conio.h>
// khai bao prototype
// ham so sanh a và b
int tim_max(int a, int b)
 {
if(a>=b)
return a;
else
return b; }
int main()
{
int a=5, b=7;
printf("Max là %d ",tim_max(a,b));
getch();
}
```

2.5.3 Tham số hình thức, tham số thực và biến cục bộ

Các tham số dùng khi khai báo hàm được gọi là tham số hình thức. Các tham số được cung cấp cho hàm khi gọi hàm là tham số thực. Tham số thực có thể là một biểu thức, trong khi tham số hình thức thì không thể là 1 biểu thức. Dãy các tham số thực phải tương ứng về kiểu với tham số hình thức.

Có những hàm không cần có tham số. Vì vậy, khi khai báo ta có thể dùng từ khóa void để báo rằng hàm không cần tham số.

ví dụ : hàm tra bảng cửu chương 2

```
void in_cuuchuong2(void)
{
        for(int i=1;i<=10;i++)
                printf("2 x %d = %dn", i, i*2);
}
```

Biến toàn cục là biến được khai báo ngoài tất cả các hàm, kể cả hàm main. biến toàn cục có thể được sử dụng trong cả chương trình

Biến cục bộ là biến chỉ có phạm vi hoạt động trọng nội bộ hàm, được khia báo bên trong hàm. Do tham số thực và biến cục bộ đều có phạm vi hoạt động trong cùng một hàm nên tham số thực và biến cục bộ cần có tên khác nhau.

Tham số hình thức và **biến cục bộ** có thể trùng tên với các đại lượng ngoài hàm mà không gây ra nhầm lẫn nào.

Khi một hàm được gọi tới, việc đầu tiên là giá trị của các tham số thực được gán cho các tham số hình thức. Như vậy các tham số hình thức chính là các bản sao của các tham số thực. Hàm chỉ làm việc trên các tham số hình thức.

Các tham số hình thức có thể bị biến đổi trong thân hàm, còn các tham số thực thì không bị thay đổi.

ví dụ :

```
#include <stdio.h>
#include <conio.h>
// khai bao prototype
int power(int, int);
void main(void)
{
printf("2 mu 2 = %d.n", power(2, 2));
printf("2 mu 3 = %d.n", power(2, 3));
getch();
}
// ham tinh so mu
int power(int ix, int in)
{
int i, ip = 1;
for(i = 1; i <= in; i++)
ip *= ix; //tương đương với ip=ip*ix
return ip; //giá trị trả về cho hàm
}
```

kết quả :

```
2 mu 2 = 4.
2 mu 3 = 8.
```

Giải thích chương trình:

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

Dòng lệnh: return ip, trả về giá trị sau khi tính toán

Hai tham số ix, in của hàm power là dạng truyền tham trị.

2.5.4 Quy tắc hoạt động của hàm

Khi gặp một lời gọi hàm thì nó sẽ bắt đầu được thực hiện. Nói cách khác, khi máy gặp lời gọi hàm ở một vị trí nào đó trong chương trình, máy sẽ tạm dời chỗ đó và chuyển đến hàm tương ứng. Quá trình đó diễn ra theo trình tự sau:

- Cấp phát bộ nhớ cho các biến cục bộ.
- Gán giá trị của các tham số thực cho các tham số hình thức tương ứng.
- Thực hiện các câu lệnh trong thân hàm.
- Khi gặp câu lệnh return hoặc dấu } cuối cùng của thân hàm thì máy sẽ xoá các tham số hình thức, biến cục bộ và ra khỏi hàm.

Nếu trở về từ một câu lệnh return có chứa biểu thức thì giá trị của biểu thức được gán cho hàm. Giá trị của hàm sẽ được sử dụng trong các biểu thức chứa nó.

2.6 Kiểm tra

BÀI TẬP

1. Viết hàm tìm số lớn nhất trong hai số. Áp dụng tìm số lớn nhất trong ba số a, b, c với a, b, c nhập từ bàn phím.

2. Viết hàm tìm UCLN của hai số a và b. Áp dụng: nhập vào tử và mẫu số của một phân số, kiểm tra xem phân số đó đã tối giản hay chưa.

3. Viết hàm in n ký tự c trên một dòng. Viết chương trình cho nhập 5 số nguyên cho biết số lượng hàng bán được của mặt hàng A ở 5 cửa hàng khác nhau. Dùng hàm trên vẽ biểu đồ so sánh 5 giá trị đó, mỗi trị dùng một ký tự riêng.

4. Viết một hàm tính tổng các chữ số của một số nguyên. Viết chương trình nhập vào một số nguyên, dùng hàm trên kiểm tra xem số đó có chia hết cho 3 không. Một số chia hết cho 3 khi tổng các chữ số của nó chia hết cho 3.

5. Tam giác Pascal là một bảng số, trong đó hàng thứ 0 bằng 1, mỗi một số hạng của hàng thứ n+1 là một tổ hợp chập k của n ($C_n^k = \frac{k!}{(n-k)!}$)

Tam giác Pascal có dạng sau:

1 (hàng 0)
1 1 (hàng 1)
1 2 1 (hàng 2)
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1 (hàng 6)
.....

Viết chương trình in lên màn hình tan giác Pascal có n hàng (n nhập vào khi chạy chương trình) bằng cách tạo hai hàm tính giai thừa và tính tổ hợp.

6. Yêu cầu như câu 5 nhưng dựa vào tính chất sau của tổ hợp:

$$C_n^k = C_{n-1}^{k-1} + C_{n-1}^k$$

để hình thành thuật toán là: tạo một hàm tổ hợp có hai biến n, k mang tính đệ quy như sau:

$$ToHop(n,k)=\begin{cases}1 \text{ nếu } k=0 \text{ hoặc } k=n\\ ToHop(n-1,k-1)+ToHop(n-1,k) \text{ nếu } 1<k<n\end{cases}$$

7. Viết chương trình tính các tổng sau:

a) $S= 1 + x + x^2 + x^3 + \ldots + x^n$

b) $S= 1 - x + x^2 - x^3 + \ldots (-1)^n x^n$

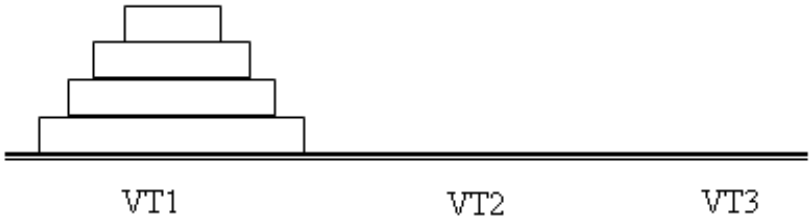
c) $S= 1 + x/1! + x^2/2! + x^3/3! + \ldots + x^n/n!$

Trong đó n là một số nguyên dương và x là một số bất kỳ được nhập từ bàn phím khi chạy chương trình.

8. Viết chương trình in dãy Fibonacci đã nêu trong bằng phương pháp dùng một hàm Fibonacci F có tính đệ quy.

$$F_n=\begin{cases}1, \text{nếu } n=1\\ 2, \text{nếu } n=2\\ F_{n-1}+F_{n-2}\end{cases}$$

9. Bài toán tháp Hà Nội: Có một cái tháp gồm n tầng, tầng trên nhỏ hơn tầng dưới (hình vẽ). Hãy tìm cách chuyển cái tháp này từ vị trí thứ nhất sang vị trí thứ hai thông qua vị trí trung gian thứ ba. Biết rằng chỉ được chuyển mỗi lần một tầng và không được để tầng lớn trên tầng nhỏ.



10. Viết chương trình phân tích một số nguyên dương ra thừa số nguyên tố.

CHƯƠNG 5. DỮ LIỆU KIỂU TẬP HỢP, MẢNG VÀ BẢN GHI

Giới thiệu

Trong khi làm việc với bất kỳ ngôn ngữ lập trình nào, bạn cần sử dụng các kiểu biến đa dạng để lưu giữ thông tin. Các biến, không gì khác ngoài các vị trí bộ nhớ được dành riêng để lưu giá trị. Nghĩa là, khi bạn tạo một biến, bạn dành riêng một số không gian trong bộ nhớ cho biến đó.

Bạn có thể thích lưu thông tin của các kiểu dữ liệu (Data Type) đa dạng như *Character, Wide Character, integer, floating-point, double floating point, Boolean,...* Dựa trên kiểu dữ liệu của một biến, hệ thống sẽ cấp phát bộ nhớ và quyết định những gì có thể được lưu giữ trong bộ nhớ dành riêng đó.

1. Mục tiêu:

- Trình bày được khái niệm tập hợp, mảng và bản ghi;
- Thực hiện cách khai báo, gán giá trị cho tập hợp, mảng, bản ghi;
- Thực hiện các phép toán trên tập hợp, mảng và bản ghi;
- Thực hiện các thao tác an toàn với máy tính.

2. Nội dung:

2.1 Kiểu tập hợp, các phép toán trên tập hợp

2.1.1 Tổng quan về tập hợp

+ Khái niệm tập hợp

Tập hợp là một khái niệm cơ bản trong toán học. Tập hợp được dùng để mô hình hoá hay biểu diễn một nhóm bất kỳ các đối tượng trong thế giới thực cho nên nó đóng vai trò rất quan trọng trong mô hình hoá cũng như trong thiết kế các giải thuật.

Khái niệm tập hợp cũng như trong toán học, đó là sự tập hợp các thành viên (members) hoặc phần tử (elements). Tất cả các phần tử của tập hợp là khác nhau. Tập hợp có thể có thứ tự hoặc không có thứ tự, tức là, có thể có quan hệ thứ tự xác định trên các phần tử của tập hợp hoặc không. Tuy nhiên, trong chương này, chúng ta giả sử rằng các phần tử của tập hợp có thứ tự tuyến tính, tức là, trên tập hợp S có quan hệ < và = thoả mãn hai tính chất:

Với mọi a,b ∈ S thì a<b hoặc b<a hoặc a=b

Với mọi a,b,c ∈ S, a<b và b<c thì a<c

+ Kiểu dữ liệu trừu tượng tập hợp

Cũng như các kiểu dữ liệu trừu tượng khác, các phép toán kết hợp với mô hình tập hợp sẽ tạo thành một kiểu dữ liệu trừu tượng là rất đa dạng. Tùy theo nhu cầu của các ứng dụng mà các phép toán khác nhau sẽ được định nghĩa trên tập hợp. Ở đây ta đề cập đến một số phép toán thường gặp nhất như sau

Thủ tục **UNION(A,B,C)** nhận vào 3 tham số là A,B,C; Thực hiện phép toán lấy hợp của hai tập A và B và trả ra kết quả là tập hợp C = A ∪B.

Thủ tục **INTERSECTION(A,B,C)** nhận vào 3 tham số là A,B,C; Thực hiện phép toán lấy giao của hai tập A và B và trả ra kết quả là tập hợp C = A ∩ B.

Thủ tục **DIFFERENCE(A,B,C)** nhận vào 3 tham số là A,B,C; Thực hiện phép toán lấy hợp của hai tập A và B và trả ra kết quả là tập hợp C = A\B

Hàm **MEMBER(x,A)** cho kết quả kiểu logic (đúng/sai) tùy theo x có thuộc A hay không. Nếu x ∈ A thì hàm cho kết quả là 1 (đúng), ngược lại cho kết quả 0 (sai).

Thủ tục **MAKENULLSET(A)** tạo tập hợp A tập rỗng

Thủ tục **INSERTSET(x,A)** thêm x vào tập hợp A

Thủ tục **DELETESET(x,A)** xoá x khỏi tập hợp A

Thủ tục **ASSIGN(A,B)** gán A cho B (tức là B:=A)

Hàm **MIN(A)** cho phần tử bé nhất trong tập A

Hàm **EQUAL(A,B)** cho kết quả TRUE nếu A=B ngược lại cho kết quả FALSE

+ Cài đặt tập hợp

+ Cài đặt tập hợp bằng vector Bit

Hiệu quả của một cách cài đặt tập hợp cụ thể phụ thuộc vào các phép toán và kích thước tập hợp. Hiệu quả này cũng sẽ phụ thuộc vào tần suất sử dụng các phép toán trên tập hợp. Chẳng hạn nếu chúng ta thường xuyên sử dụng phép thêm vào và loại bỏ các phần tử trong tập hợp thì chúng ta sẽ tìm cách cài đặt hiệu quả cho các phép toán này. Còn nếu phép tìm kiếm một phần tử xảy ra thường xuyên thì ta có thể phải tìm cách cài đặt phù hợp để có hiệu quả tốt nhất.

Ở đây ta xét một trường hợp đơn giản là khi toàn thể tập hợp của chúng ta là tập hợp con của một tập hợp các số nguyên nằm trong phạm vi nhỏ từ 1.. n chẳng hạn thì chúng ta có thể dùng một mảng kiểu Boolean có n phần tử để cài đặt tập hợp (ta gọi là vectơ bít), bằng cách cho phần tử thứ i của mảng này giá trị TRUE nếu i thuộc tập hợp hoặc cho mảng lưu kiểu 0-1. Nếu nội dung phần tử trong mảng tại vị trí i là 1 nghĩa là i tồn tại trong tập hợp và ngược lại, nội dung là 0 nghĩa là phần tử i đó không tồn tại trong tập hợp.

Ví dụ: Giả sử các phần tử của tập hợp được lấy trong các số nguyên từ 1 đến 10 thì mỗi tập hợp được biểu diễn bởi một mảng một chiều có 10 phần tử với các giá trị phần tử thuộc kiểu logic. Chẳng hạn tập hợp A={1,3,5,8} được biểu diễn trong mảng có 10 phần tử như sau:

1	2	3	4	5	6	7	8	9	10
1	0	1	0	1	0	0	1	0	0

Cách biểu diễn này chỉ thích hợp trong điều kiện là mọi thành viên của tất cả các tập hợp đang xét phải có giá trị nguyên hoặc có thể đặt tương ứng duy nhất với số nguyên nằm trong một phạm vi nhỏ. Có thể dễ dàng nhận thấy khai báo cài đặt như sau

```
const maxlength = 100;
// giá trị phần tử lớn nhất trong tập hợp số nguyên không âm
typedef int SET [maxlength];
```

+ Tạo một tập hợp rỗng

Để tạo một tập hợp rỗng ta cần đặt tất cả các nội dung trong tập hợp từ vị trí 0 đến vị trí maxlength đều bằng 0. Câu lệnh được viết như sau :

```
void makenull(SET a)
{
int i;
```

```
for(i=0;i<maxlength;i++)
a[i]=0;
}
```

Biểu diễn tập hợp bằng vectơ bít tạo điều kiện thuận lợi cho các phép toán trên tập hợp. Các phép toán này có thể cài đặt dễ dàng bằng các phép toán Logic trong ngôn ngữ lập trình. Chẳng hạn thủ tục UNION(A,B,C) và thủ tục INTERSECTION được viết như sau :

```
void SET_union (SET a,SET b,SET c)
{
int i;
for (i=0;i<maxlength;i++)
if ((a[i]==1)||(b[i]==1)) c[i]=1;
else c[i]=0;
}
void SET_intersection (SET a,SET b, SET c)
{
int i;
for (i=0;i<maxlength;i++)
if ((a[i]==1)&&(b[i]==1)) c[i]=1;
else c[i]=0;
}
```

Các phép toán giao, hiệu,... được viết một cách tương tự. Việc kiểm tra một phần tử có thuộc tập hợp hay không, thủ tục thêm một phần tử vào tập hợp, xóa một phần tử ra khỏi tập hợp cũng rất đơn giản và xem như bài tập.

Cài đặt bằng danh sách liên kết

Tập hợp cũng có thể cài đặt bằng danh sách liên kết, trong đó mỗi phần tử của danh sách là một thành viên của tập hợp. Không như biểu diễn bằng vectơ bít, sự biểu diễn này dùng kích thước bộ nhớ tỉ lệ với số phần tử của tập hợp chứ không phải là kích thước đủ lớn cho toàn thể các tập hợp đang xét. Hơn nữa, ta có thể biểu diễn một tập hợp bất kỳ. Mặc dù thứ tự của các phần tử trong tập hợp là không quan trọng nhưng nếu một danh sách liên kết có thứ tự nó có thể trợ giúp tốt cho các phép duyệt danh sách. Chẳng hạn nếu tập hợp A được biểu diễn bằng một danh sách có thứ tự tăng thì hàm MEMBER(x,A) có thể thực hiện việc so sánh x một cách tuần tự từ đầu danh sách cho đến khi gặp một phần tử y ≥ x chứ không cần so sánh với tất cả các phần tử trong tập hợp.

Một ví dụ khác, chẳng hạn ta muốn tìm giao của hai tập hợp A và B có n phần tử. Nếu A,B biểu diễn bằng các danh sách liên kết chưa có thứ tự thì để tìm giao của A và B ta phải tiến hành như sau:

for (mỗi x thuộc A)

{ Duyệt danh sách B xem x có thuộc B không. Nếu có thì x thuộc giao của hai tập hợp A và B; }

Rõ ràng quá trình này có thể phải cần đến n x m phép kiểm tra (với n,m là độ dài của A và B).

Nếu A,B được biểu diễn bằng danh sách có thứ tự tăng thì đối với một phần tử e∈A ta chỉ tìm kiếm trong B cho đến khi gặp phần tử x ≥ e. Quan trọng hơn nếu f đứng ngay sau e trong A thì để tìm kiếm f trong B ta chỉ cần tìm từ phần tử x trở đi chứ không phải từ đầu danh sách lưu trữ tập hợp B.

Khai báo

```
typedef int ElementType;
typedef struct Node
{
ElementType Data;
Node * Next;
};
typedef Node * Position;
typedef Position SET;
```

Thủ tục UNION

```
//C= hop cua hai tap hop A,B
void UnionSET(SET A, SET B, SET *C)
{
Position p;
MakeNullSET(C);
p=First(A);
while (p!=EndSET(A))
{ InsertSET (Retrieve(p,A),*C);
p=Next(p,A);
}
p=First(B);
while (p!=EndSET (B))
{ if (Member(Retrieve(p,B),*C)==0)
InsertSET (Retrieve(p,B),*C);
p=Next(p,B);
}
}
```

Thủ tục INTERSECTION

```
// C=giao cua hai tap hop A,B
void IntersectionSET(SET A, SET B, SET *C)
{
Position p;
MakeNullSET(C);
p=First(A);
while (p!=EndSET(A))
{ if (Member(Retrieve(p,A),B)==1)
```

```
InsertSET (Retrieve(p,A),*C);
p=Next(p,A);
}
}
```

Phép toán hiệu có thể viết tương tự (xem như bài tập). Phép ASSIGN(A,B) chép các các phần tử của tập A sang tập B, chú ý rằng ta không được làm bằng lệnh gán đơn giản B:=A vì nếu làm như vậy hai danh sách biểu diễn cho hai tập hợp A,B chỉ là một nên sự thay đổi trên tập này kéo theo sự thay đổi ngoài ý muốn trên tập hợp kia. Toán tử MIN(A) chỉ cần trả ra phần tử đầu danh sách (tức là A->Next->Data). Toán tử DELETESET là hoàn toàn giống như DELETE_LIST. Phép INSERTSET(x,A) cũng tương tự INSERT_LIST tuy nhiên ta phải chú ý rằng khi xen x vào A phải đảm bảo thứ tự của danh sách.

Thêm phần tử vào tập hợp

```
// Them phan tu vao tap hop co thu tu
void InsertSET(ElementType X, SET L)
{
Position T,P;
int finish=0;
P=L;
while ((P->Next!=NULL)&&(finish==0))
if (P->Next->Data<=X)
P=P->Next;
else finish=1;
// P dang luu tru vi tri de xen phan tu X vao
T=(Node*)malloc(sizeof(Node));
T->Data=X;
T->Next=P->Next;
P->Next=T;
}
```

Xoá phần tử ra khỏi tập hợp:

```
void DeleteSET(ElementType X, SET L)
{
Position T,P=L;
int finish=0;
while ((P->Next!=NULL)&& (finish==0))
if (P->Next->Data<X) P=P->Next;
else finish=1;
if (finish==1)
if(P->Next->Data==X)
{ T=P->Next;
P->Next=T->Next;
free(T);
```

```
}
}
```

Kiểm tra sự hiện diện của phần tử trong tập hợp:

Hàm kiểm tra xem phần tử X có thuộc tập hợp hay không. Hàm trả về giá trị 1 nếu phần tử X đó thuộc tập hợp và ngược lại, hàm trả về giá trị 0.

```
int Member(ElementType X, SET L)
{
Position P;
int Found = 0;
P = First(L);
while ((P != EndSET(L)) && (Found == 0))
if (Retrieve(P,L) == X) Found = 1;
else P = Next(P, L);
return Found;
}
```

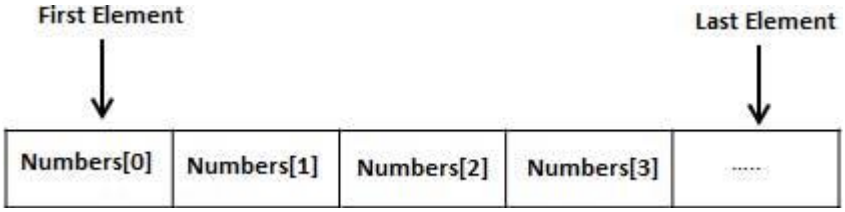
2.2 Khái niệm mảng, khai báo mảng, gán giá trị

2.2.1 Khái niệm mảng

Ngôn ngữ lập trình C cung cấp cấu trúc dữ liệu gọi là **mảng**, được lưu trữ trong một tập hợp các dữ liệu cùng kiểu với độ dài cố định. Một mảng được sử dụng để lưu trữ tập hợp dữ liệu, nhưng nó rất hữu dụng nếu bạn nghĩ về một mảng các biến với cùng một kiểu.

Thay vì khai báo biến một cách rời rạc, như biến number0, number1,… và number99, bạn có thể khai báo một mảng các giá trị như numbers[0], numbers[1] và … numbers[99] để biểu diễn các giá trị riêng biệt. Một thành phần cụ thể của mảng có thể được truy cập qua index (chỉ số).

Tất cả mảng đều bao gồm các vị trí nhớ liền kề nhau. Địa chỉ thấp nhất tương ứng với thành phần đầu tiền và địa chỉ cao nhất tương ứng với thành phần cuối cùng của mảng.



2.2.2 khai báo mảng

Để khai báo một mảng trong ngôn ngữ C, chương trình xác định kiểu của biến và số lượng các phần tử được yêu cầu bởi biến đó như sau:

Kieu Ten_mang [Kich_co_mang];

Đây là **mảng một chiều**. **Kich_co_mang** phải là một số nguyên lớn hơn 0 và **Kieu** phải hợp lệ trong ngôn ngữ C. Ví dụ, khai báo một mảng 10 phần tử gọi là *sohangban* với kiểu int, sử dụng câu lệnh sau đây:

int sohangban[10];

Bây giờ *sohangban* là một biến mảng có thể đủ chỗ chứa *10* phần tử int.

2.2.3 Khởi tạo mảng và gán giá trị

Bạn có thể khởi tạo mảng trong C hoặc từng phần tử một hoặc sử dụng một câu lệnh như dưới đây:

int sohangban[5] = {34, 56, 23, 124, 67};

Số lượng các giá trị trong dấu ngoặc kép {} không được lớn hơn số lượng phần tử khai báo trong dấu ngoặc vuông [].

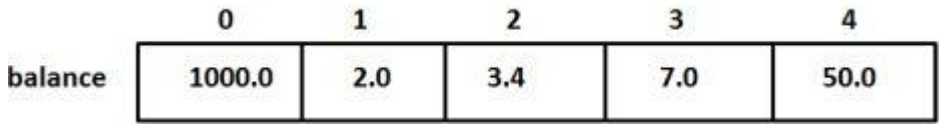
Nếu bạn bỏ sót kích cỡ mảng thì mảng đó đủ lớn để giữ các giá trị được khởi tạo:

int sohangban[] = {34, 56, 23, 124, 67};

Bạn sẽ tạo chính xác một chuỗi có giá trị giống hệt chuỗi bên trên bằng cách gán từng phần tử một. Dưới đây là một ví dụ khi gán giá trị cho một phần tử của mảng:

sohangban[4] = 67;

Câu lệnh bên trên gán giá trị thứ 5 của mảng giá trị 67. Tất cả các mảng đều có chỉ số (index) đầu tiên bằng 0, đây được gọi là chỉ số cơ bản và phần tử cuối cùng của mảng có chỉ số bằng độ lớn của mảng trừ đi 1. Dưới đây là cách biểu diễn hình họa cho chuỗi khai báo bên trên thông qua chỉ số:



2.2.4 Truy cập các phần tử mảng

Một mảng được truy cập bởi cách đánh chỉ số trong tên của mảng. Dưới đây là một cách truy cập một giá trị của mảng:

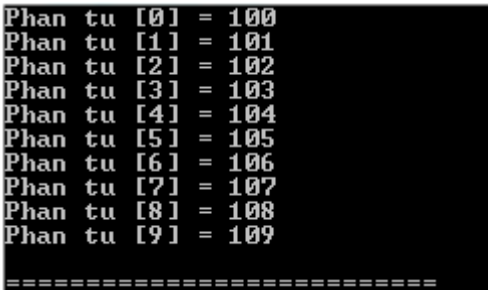
int luonghangban = sohangban[9];

Câu lệnh trên lấy phần tử thứ 10 của mảng và gán giá trị này cho biến *luonghangban*. Dưới đây là một ví dụ về việc sử dụng với tất cả mô tả bên trên:

```
#include <stdio.h>
int main ()
{
  int n[ 10 ]; /* mang n gom 10 so nguyen */
  int i,j;
  /* khoi tao cac phan tu trong mang ve gia tri 0 */
  for ( i = 0; i < 10; i++ )
  {
    n[ i ] = i + 100; /* Thiet lap phan tu tai vi tri i thanh i + 100 */
  }
  /* hien thi gia tri cac phan tu trong mang */
  for (j = 0; j < 10; j++ )
  {
    printf("Phan tu [%d] = %d\n", j, n[j] );
  }
  printf("\n==========================\n");
  printf("Chuc cac ban hoc tot! \n");
  return 0;
```

}

Biên dịch và chạy chương trình C trên sẽ cho kết quả:



2.2.5 Chi tiết về mảng trong C

Mảng là một phần rất quan trọng trong ngôn ngữ C. Dưới đây là những định nghĩa quan trọng liên quan đến một mảng cụ thể mà được trình bày rõ ràng hơn cho các lập trình viên C:

Khái niệm	Miêu tả
Mảng đa chiều trong C	C hỗ trợ các mảng đa chiều. Mẫu đơn giản nhất của mảng này là mảng hai chiều.
Truyền mảng cho hàm như là tham số trong C	Bạn có thể truyền tới hàm một điểm trỏ chỉ tới một mảng bởi xác định tên mảng chứ không phải là một chỉ số.
Trả về mảng từ một hàm trong C	C cho phép một hàm có thể trả về một mảng.
Trỏ tới một mảng trong C	Bạn có thể trỏ tới phần tử đầu tiên của mảng một cách đơn giản chỉ bởi xác định tên mảng đó, chứ không phải một chỉ số.

2.3 Mảng nhiều chiều

Ngôn ngữ C hỗ trợ các mảng đa chiều. Dưới đây là mẫu chung của một khai báo mảng đa chiều:

kieu_du_lieu ten_mang[kichco_1][kichco_2]...[kichco_N];

Ví dụ, khai báo sau tạo một mảng số nguyên 3 chiều: 5. 10. 4:

int mangbachieu[5][10][4];

2.3.1 Mảng hai chiều trong C

Mẫu đơn giản nhất của mảng đa chiều là mảng hai chiều. Một mảng hai chiều về bản chất là danh sách của các mảng một chiều. Để khai báo một mảng hai chiều integer với kích cỡ x, y, bạn nên viết như sau:

kieu_du_lieu ten_mang [x][y];

Ở đây, **kieu_du_lieu** có thể là bất kỳ kiểu dữ liệu có hiệu lực nào và **ten_mang** sẽ là một định danh C có hiệu lực. Một mảng hai chiều có thể như là một bảng mà có x hàng và y cột. Một mảng hai chiều **a** chứa 3 hàng và 4 cột có thể được hiển thị như sau:

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Như vậy, mỗi phần tử trong mảng **a** được định danh bởi một tên phần tử trong kiểu mẫu **a[i][j]**, với a là tên mảng và i, j là các subscript – chỉ số được xác định duy nhất mỗi phần tử trong a.

2.3.2 Khởi tạo mảng hai chiều trong C

Các mảng đa chiều có thể được khởi tạo bởi xác định các giá trị trong dấu móc vuông cho mỗi hàng. Sau đây là một hàng với 3 hàng và mỗi hàng chứa 4 cột.

```
int a[3][4] = {
 {0, 1, 2, 3} ,   /*  khoi tao cho hang co chi muc la 0 */
 {4, 5, 6, 7} ,   /*  khoi tao cho hang co chi muc la 1 */
 {8, 9, 10, 11}   /*  khoi tao cho hang co chi muc la 2 */
};
```

Các dấu ngoặc ôm, mà chỉ các hàng giá trị là tùy ý. Khởi tạo sau là tương đương với ví dụ trên:

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

2.3.3 Truy cập các phần tử của mảng hai chiều trong C

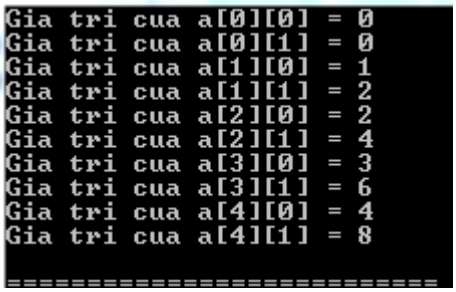
Các phần tử mảng hai chiều được truy cập bởi sử dụng các chỉ số, ví dụ chỉ số hàng và chỉ số cột. Ví dụ:

```
int ten_bien = a[2][3];
```

Lệnh trên sẽ truy cập vào phần tử thứ 4 từ hàng thứ 3 của mảng. Bạn có thể kiểm tra lại nó trong sơ đồ trên. Bây giờ chúng ta xem xét ví dụ dưới đây, chúng tôi đã sử dụng các vòng lặp lồng vào nhau để xử lý một mảng hai chiều:

```
#include <stdio.h>
 int main ()
{   /* mang sau co 5 hang va 2 cot */
  int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
  int i, j;
  /* hien thi gia tri cac phan tu trong mang */
  for ( i = 0; i < 5; i++ )
  {      for ( j = 0; j < 2; j++ )
    {
      printf("Gia tri cua a[%d][%d] = %d\n", i,j, a[i][j] );
    }
  }
  printf("\n==========================\n");
  return 0;
}
```

Biên dịch và chạy chương trình C trên sẽ cho kết quả:



Như phần trên đã giải thích, bạn có thể có các mảng với số chiều bất kỳ, nhưng hầu hết các mảng bạn tạo ra sẽ là một hoặc hai chiều.

2.4 Kiểu bản ghi

Dữ liệu kiểu bản ghi dùng để mô tả các đối tượng có cùng một số thuộc tính mà các thuộc tính có thể có các kiểu dữ liệu khác nhau.

- Kiểu bản ghi là một kiểu dữ liệu có cấu trúc. Một bản ghi gồm các thành phần (gọi là trường), khác với các kiểu dữ liệu có cấu trúc khác (mảng và xâu), các trường có thể thuộc các kiểu dữ liệu khác nhau.

- Kiểu bản ghi cho phép mô tả nhiều đối tượng cỏ cùng một số thuộc tính, có hữu ích cho nhiều bài toán quản lí.

- Ngôn ngữ lập trình đưa ra quy tắc, cách thức xác định:

+ Tên kiểu bản ghi.

+ Tên các thuộc tính (trường).

+ Kiểu dữ liệu của mỗi trường.

+ Cách khai báo biến.

+ Cách tham chiểu đến trường.

Ví dụ: Ta muốn lưu thông tin về sinh viên. Dữ liệu cần lưu trữ bao gồm:

+ Họ tên (Kiểu String).

+ Giới tính (Kiếu String).

+ Điểm số (Kiểu real).

Để lưu trữ nhiều sinh viên ta có thể sử dụng các cách:

+ Khai báo một mảng bản ghi. Mỗi bản ghi sẽ chứa thông tin về một sinh viên (họ tên,giới tính,điểm số).

+ Khai báo 3 mảng 2 mảng String để lưu tên và giới tính 1 mảng kiểu real để lưu điểm số.

Rõ ràng cách đầu tiên thuật tiện để lưu trữ hơn cách thứ hai khi số thuộc tính lớn.

2.4.1. Khai báo

Ta cần khai báo tên kiểu bản ghi, tên các thuộc tính, kiểu dữ liệu của mỗi thuộc tính.

Cú pháp:

Type<tên kiểu bản ghi> = record

<tên trường 1>:<kiểu trường 1>;

……..

<tên trường 2>:<kiểu trường 2>;

```
                End;
```

Sau khi có kiểu bản ghi, biến kiểu bản ghi có thể được khai báo như sau:

```
Var
    <tên biến bản ghi>:<tên kiểu bản ghi>;
```

Ví dụ:

```
    Type
        Hocsinh=record
            Hoten:string[30];
            Ngaysinh:string[10];
            Gioitinh:Boolean;
            Toan,Tin,Van,Anh:real;
    End;
    Var
    A,b:Hocsinh;
    Lop:array[1..60] of Hocsinh;
```

Để tham chiếu đến thuộc tính nào ta sự dụng dấu.

Ví dụ:

Tham chiếu đến họ tên của Hocsinh A. Ta sử dụng

A.Hoten

Ta có thể coi A.Hoten như một biến string[30] . Ta hoàn toàn có thể thực hiện so sánh ,gán giá trị...

2.4.2. Gán giá trị.

Có 2 cách:

+ Dùng lệnh gán trực tiếp: A:=B (A với B là 2 biến bản ghi).

+ Gán giá trị cho từng trường: A.X:=C hoặc nhập từ bàn phím.

Ví dụ:

Một lớp gồm N (N<=60) học sinh. Cần quản lí học sinh với các thuộc tính như họ và tên, ngày sinh

```
program xep_loai;
uses crt;
const max= 60;
type Hocsinh = record
hoten: string[30];
ngaysinh: string [10];
Diachi: string[50] ;
Toan, Van: real;
Xeploai : char;
end;
var
Lop: array [ 1..max] of hocsinh;
N,i: byte;
Begin
```

```
clrscr;
write('So luong hoc sinh trong lop N=  ') ; readln(N);
for i:= 1 to N do
begin
writeln( 'Nhap so lieu ve hoc sinh thu',i,': ');
Write ('Ho va ten: '); readln (lop [i]. hoten);
Write (' Ngay sinh : '); readln (lop [i].ngaysinh);
Write (' Dia chi : '); readln (lop [i].Diachi);
Write ('Diem Toan : '); readln (lop [i]. Toan);
Write ('Diem Van : '); readln (lop [i]. Van);
If Lop [i]. Toan+Lop [i]. Van >=18
then Lop [i]. xeploai:='A';
if    (Lop[i].Toan+Lop[i].Van>=14)    and
(Lop [i]. Toan+Lop [i]. Van <18)
then Lop [i]. xeploai:='B';
if    (Lop[i].Toan+Lop[i].Van>=10)    and
(Lop [i]. Toan+Lop [i]. Van >=14)
then Lop [i]. xeploai:='C';
if    (Lop[i].Toan+Lop[i].Van<=10)
then Lop[i].xeploai:='D';
end;
clrscr;
writeln ('Danh sach xep loai hoc sinh trong lop: ' );
for i:=1 to N do
writeln (Lop[i].Hoten:30,' – Xep loai : ', Lop[i]. Xeploai);
readln
End.
```

2.5 Kiểm tra

BÀI TẬP

Bài 1 : Khởi tạo một mảng và sắp xếp giảm mảng đó

Bài 2 : Nhập N là số phần tử của dãy. Sau đó nhập 1 dãy số gồm N phần tử. In ra:

- Phần tử nhỏ nhất
- Phần tử lớn nhất
- Tổng các phần tử chẵn
- Số lượng các phần tử lẻ

Bài 3: Nhập 1 dãy số gồm N phần tử, các phần tử đó nằm trong khoảng từ 0..9. Đếm số lượng các số 0,1,2,…9 trong dãy số đó

CHƯƠNG 6. DỮ LIỆU KIỂU CHUỖI

Giới thiệu

Chuỗi được xem như là một mảng 1 chiều gồm các phần tử có kiểu char như ký tự, con số và bất cứ ký tự đặc biệt như +, -, *, /, $, #, … Theo quy ước, một chuỗi sẽ được kết thúc bởi ký tự null (‘\0’ : kí tự rỗng còn được gọi là ký tự NULL trong bảng mã ASCII).

Các hằng chuỗi ký tự được đặt trong cặp dấu nháy kép "".

1. Mục tiêu:

Trình bày được khái niệm dữ liệu kiểu chuỗi kí tự ;
Biết sử dụng dữ liệu kiểu chuỗi trong chương trình;
Áp dụng được các phép toán trên chuỗi;
Vận dụng được các hàm xử lý chuỗi để xử lý;
Thực hiện các thao tác an toàn với máy tính.

2. Nội dung:

2.1 Khai báo và các phép toán

Chuỗi trong ngôn ngữ lập trình C thực chất là mảng một chiều của các ký tự mà kết thúc bởi một ký tự **null** '\0'.

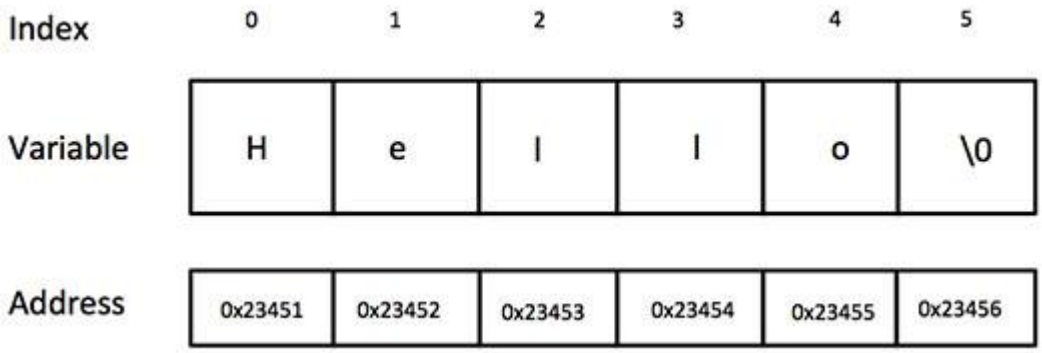
Phần khai báo và khởi tạo dưới đây tạo ra một chuỗi bao gồm một từ "Hello". Để giữ các giá trị null tại cuối của mảng, cỡ của mảng các ký tự bao gồm một chuỗi phải nhiều hơn số lượng các ký tự trong từ khóa "Hello".

char loichao[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

Nếu bạn theo quy tắc khởi tạo các chuỗi, bạn có thể viết lệnh như sau:

char loichao[] = "Hello";

Dưới đây là phần biểu diễn ô nhớ cho đoạn chuỗi trên trong ngôn ngữ C/C++:

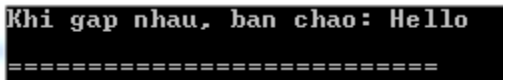


Thực tế, bạn không đặt ký tự **null** tại vị trí cuối cùng của biến hằng số. Bộ biên dịch C tự động thêm '\0' tại ví trí cuối cùng của chuỗi khi nó khởi tạo chuỗi. Cùng thử ví dụ in ra chuỗi sau đây:

```
#include <stdio.h>
int main ()
{
  char loichao[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
  printf("Khi gap nhau, ban chao: %s\n", loichao );
  printf("\n==========================\n");
```

```
  return 0;
}
```

Biên dịch và chạy chương trình C trên sẽ cho kết quả:



Ngôn ngữ C hỗ trợ một dãy rộng rãi các hàm để thao tác các chuỗi kết thúc là null:

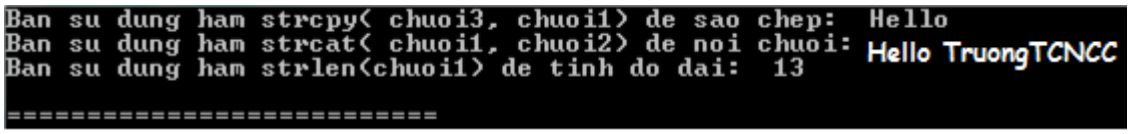
STT	Hàm & Mục đích
1	**strcpy(s1, s2);** Sao chép chuỗi s2 cho chuỗi s1.
2	**strcat(s1, s2);** Nối chuỗi s2 vào cuối chuỗi s1.
3	**strlen(s1);** Trả về độ dài của chuỗi s1.
4	**strcmp(s1, s2);** Trả về 0 nếu s1 và s2 là như nhau; nhỏ hơn 0 nếu s1<s2; lớn hơn 0 nếu s1>s2.
5	**strchr(s1, ch);** Trả về con trỏ tới vị trí đầu tiên của ch trong s1.
6	**strstr(s1, s2);** Trả về con trỏ tới vị trí đầu tiên của chuỗi s2 trong chuỗi s1.

Dưới đây là ví dụ cho việc sử dụng một vài hàm bên trên:

```
#include <stdio.h>
#include <string.h>  /* thu vien cho cac ham xu ly chuoi*/
int main ()
{
   char chuoi1[12] = "Hello";
   char chuoi2[12] = "TruongTCNCC";
   char chuoi3[12];
   int  dodai ;
   /* sao chep chuoi1 vao trong chuoi3 */
   strcpy(chuoi3, chuoi1);
   printf("Ban su dung ham strcpy( chuoi3, chuoi1) de sao chep:  %s\n", chuoi3 );
   /* noi hai chuoi: chuoi1 va chuoi2 */
   strcat( chuoi1, chuoi2);
   printf("Ban su dung ham strcat( chuoi1, chuoi2) de noi chuoi:   %s\n", chuoi1 );
```

```
  /* tinh do dai cua chuoi1 sau khi noi chuoi */
  dodai = strlen(chuoi1);
  printf("Ban su dung ham strlen(chuoi1) de tinh do dai:  %d\n", dodai );
  printf("\n==========================\n");
  return 0;
}
```

Biên dịch và chạy chương trình C trên sẽ cho kết quả:



Bạn có thể tìm thấy một danh sách đầy đủ các hàm liên quan tới chuỗi trong Thư viện tiêu chuẩn C.

2.2 Nhập, xuất chuỗi

2.2.1 Nhập chuỗi

Để nhập một chuỗi ký tự từ bàn phím, ta sử dụng hàm gets()

Cú pháp: **gets(<Biến chuỗi>)**

Ví dụ:

```
char Ten[20];
gets(Ten);
```

Ta cũng có thể sử dụng hàm scanf() để nhập dữ liệu cho biến chuỗi, tuy nhiên lúc này ta chỉ có thể nhập được một chuỗi không có dấu khoảng trắng.

Ngoài ra, hàm gets() (trong stdio.h) cũng được sử dụng để nhập chuỗi.

2.2.2 Xuất chuỗi

Để xuất một chuỗi (biểu thức chuỗi) lên màn hình, ta sử dụng hàm puts().

Cú pháp: **puts(<Biểu thức chuỗi>)**

Ví dụ: Nhập vào một chuỗi và hiển thị trên màn hình chuỗi vừa nhập.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char Ten[12];
    char queQuan[15];
    printf("Nhap Ten: ");fflush(stdin);gets(Ten);
   printf("Nhap Que Quan: ");fflush(stdin);gets(queQuan);
   printf("Chuoi vua nhap: ");puts(Ten);puts(queQuan);
   getch();
   return 0;
}
```

Ngoài ra, ta có thể sử dụng hàm printf(), puts() (trong conio.h) để hiển thị chuỗi lên màn hình.

2.3 Các hàm làm việc với chuỗi

Chú ý khi sử dụng các hàm này ta phải khai báo thư viện #include "string.h"

2.3.1. Hàm strcpy:

Công dụng: sao chép chuỗi nguồn vào chuỗi đích.

Cấu trúc:

char*strcpy(char *dich, **char** *nguon);

Có nghĩa là khi ta nhập vào một dãy các kí tự ở chuỗi nguồn thì nó sẽ sao chép tất cả các kí tự vừa nhập vào cái chuỗi đích.

ví dụ như sau:

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
   char A[255],B[255];
   printf("Nhap chuoi: ");
   gets(A);
   strcpy(B,A);
   printf("Chuoi dich: ");
   puts(B);
   getch();
   return 0;
}
```

Chương trình trên khi ta nhập vào mảng A một dãy các kí tự là “abc” thì khi gap hàm strcpy(B,A); thì nó sẽ copy 3 kí tự “abc” từ mảng A vào mảng B.

Nếu chúng ta muốn copy n kí tự từ chuỗi nguồn vào chuỗi đích ta dùng hàm sau:

2.3.2.Hàm strncpy:

Công dụng: sao chép n kí tự đầu tiên của chuỗi nguồn vào chuỗi đích.

Cấu trúc:

char ***strncpy(char** *dich, **char** *nguon,**int** n);

2.3.3.Hàm strlen:

Công dụng :cho biết độ dài của chuỗi s

Cấu trúc:

int strlen(char *s)

Ví dụ: Sử dụng hàm strlen xác định độ dài một chuỗi nhập từ bàn phím.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
```

```
    char Chuoi[255];
    int Dodai;
    printf("Nhap chuoi: ");
    gets(Chuoi);
    Dodai = strlen(Chuoi);
    printf("Chuoi vua nhap:");
    puts(Chuoi);
    printf("Co do_dai
          %d",Dodai);
    getch();
    return 0;
}
```

2.3.4.Hàm strcat:

Công dụng: ghép chuỗi nguồn vào sau chuỗi đích.

Cấu trúc:

char *strcat(**char** *dich,**char** *nguon)

Ví dụ: Nhập vào họ lót và tên của một người, sau đó in cả họ và tên của họ lên màn hình.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
    char HoLot[30], Ten[12];
    printf("Nhap Ho Lot: ");
    gets(HoLot);
    printf("Nhap Ten: ");
    gets(Ten);
    strcat(HoLot,Ten); /* Ghep Ten vao HoLot*/
    printf("Ho ten la: ");
    puts(HoLot);
    getch();
    return 0;
}
```

2.3.5.Hàm strncat:

Công dụng: ghép n kí tự đầu tiên của chuỗi vào sau chuỗi đích

Cấu trúc:

char ***strncat**(**char** *dich,**char** *nguon,**int** n);

2.3.6.Hàm strcmp:

Công dụng: so sánh 2 chuỗi s1 và s2

Cấu trúc:

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

Hàm sẽ trả về 1 trong các giá trị sau:
Giá trị âm nếu chuỗi s1 nhỏ hơn chuỗi s2
Giá trị 0 nếu hai chuỗi bằng nhau
Giá trị dương nếu chuỗi s1 lớn hơn chuỗi s2
Ví dụ:

```
char *chu1 = "aaa", *chu2= "bbb", *chu3 = "aaa";
strcmp(chu1, chu2); //kết quả trả về - 1
strcmp(chu1, chu3); //kết quả trả về 0
strcmp(chu2, chu3); //kết quả trả về 1
```

ví dụ minh họa đây:

```
/*Nhap danh sach ten va sap xep theo thu tu tang dan*/
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define MAXNUM 5
#define MAXLEN 10
int main(void)
{
char ten[MAXNUM][MAXLEN]; //mang chuoi
char *c[MAXNUM]; //mang con tro tro den chuoi
char *ct;
int i, j, n = 0;
//nhap danh sach ten
while (n < MAXNUM)
{
printf("Nhap vao ten nguoi thu %d: ",n + 1);
gets(ten[n]);
c[n++] = ten[n]; //con tro den ten
}
//sap xep danh sach theo thu tu tang dan
for (i = 0; i < n - 1; i ++)
for (j = i + 1; j < n; j ++)
if (strcmp(c[i], c[j]) > 0)
{
ct = c[i];
c[i] = c[j];
c[j] = ct;
}
//In danh sach da sap xep
printf("Danh sach sau khi sap xep:\n");
for (i = 0; i < n; i ++)
printf("Ten nguoi thu %d : %s\n", i + 1, c[i]);
```

```
getch();
}
```

2.3.7.Hàm strlwr:

Công dụng: chuyển tất cả các kí tự chuỗi về chữ thường

Cấu trúc:

```
char *strlwr(char *s);
```

2.3.8.Hàm strupr :

Công dụng: chuyển tất cả các kí tự chuỗi thường về chữ hoa

Cấu trúc:

```
char *strupr(char *s)
```

Ví dụ: Viết chương trình nhập vào một chuỗi ký tự từ bàn phím. Sau đó sử dụng hàm strupr(); để chuyển đổi chúng thành chuỗi chữ hoa.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
   char Chuoi[255],*s;
   printf("Nhap chuoi: ");
   gets(Chuoi);
   s=strupr(Chuoi) ;
   printf("Chuoi chu hoa:");
   puts(s);
   getch();
   return 0;
}
```

2.3.9.Hàm strrev :

Công dụng: đảo ngược chuỗi kí tự

Cấu trúc:

```
char *strrev(char *s);
```

2.3.10.Hàm strchr:

Công dụng: trả về địa chỉ vị trí xuất hiện đầu tiên của kí tự ch trong chữ s và sẽ trả về giá trị NULL trong trường hợp không tìm thấy.

Cấu trúc:

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

2.3.11.Hàm strrchr:

Cấu trúc:

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

Công dụng: trả về địachỉ vị trí xuất hiện cuối cùng của kí tự ch trong chuỗi s. Nếu không tìm thấy hàm sẽ trả về giá trị NULL

2.3.12.Hàm strstr:

Công dụng: trả về địa chỉ vị trí xuất hiện đầu tiên của chuỗi s1 trong chuỗi s và sẽ trả về giá trị NULL trong trường hợp không tìm thấy.

Cấu trúc:

char ***strstr**(**char** *s, **char** *s1);

Ví dụ: Viết chương trình sử dụng hàm strstr() để lấy ra một phần của chuỗi gốc bắt đầu từ chuỗi "hoc".

```
#include<conio.h>
#include<stdio.h>
#include<string.h>
int main()
{
   char Chuoi[255],*s;
   printf("Nhap chuoi: ");
   gets(Chuoi);
   s=strstr(Chuoi,"hoc");
   printf("Chuoi trich ra:");
   puts(s);
   getch();
   return 0;
}
```

2.3.13. Hàm memset

Công dụng: Set num byte nhớ từ vị trí được trỏ tới bằng giá trị value

Cấu trúc:

void ***memset** (**void** *ptr, **int** value, **size_t** num);

2.3.14. Hàm memcpy

Công dụng: Chép num byte từ vị trí mà source trỏ tới đến vị trí mà destination trỏ tới

Cấu trúc:

void ***memcpy** (**void** *destination, **const void** *source, **size_t** num);

2.3.15. Hàm memcmp

Công dụng: So sánh giá trị các vùng nhớ mà ptr1 và ptr2 trỏ tới theo từng byte, sẽ dừng lại khi so sánh đủ num byte. Trả về -1 khi byte đầu tiên mà không trùng nhau của 2 vùng so sánh của ptr1 nhỏ hơn ptr2, trả về 0 khi 2 vùng nhớ bằng nhau, trả về 1 khi byte đầu tiên mà không trùng nhau của 2 vùng so sánh của ptr1 lớn hơn ptr2

Cấu trúc:

int memcmp(**const void** *ptr1, **const void** *ptr2, **size_t** num);

2.3.16.Hàm stricmp:

Công dụng : So sánh 2 chuỗi không phân biệt chữ hoa chữ thường , hàm trả về tương tự strcmp.

Cấu trúc :

int stricmp (**const char** * string1, **const char** * string2);

Ví dụ:

Ví dụ này sử dụng stricmp () để so sánh hai chuỗi.

```
#include <stdio.h>
#include <string.h>
int main (void)
{
  / * So sánh hai chuỗi như là chữ thường * /
  if (0 == stricmp ( "hello", "Hello"))
   if (0 == stricmp("hello", "HELLO"))
     printf("The strings are equivalent.\n");
  else
     printf("The strings are not equivalent.\n");
  return 0;
}
```

2.4 Kiểm tra

BÀI TẬP

1. Viết chương trình in một chuỗi trong C
2. Viết chương trình tìm độ dài chuỗi trong C (không sử dụng hàm)
3. Viết chương trình để đếm số lần xuất hiện của một ký tự trong chuỗi
4. Viết chương trình để đếm số nguyên âm, phụ âm trong chuỗi
5. Viết chương trình Sắp xếp các ký tự của chuỗi trong C

Tài liệu cần tham khảo:

[1]. Quách Tuấn Ngọc. NXB Thống kê tái bản.

[2]. Các trang web, diễn đàn về lập trình C

[3]. Trang web Vietjack.com