**I. Tìm hiểu về lập trình C cơ bản.**

**1.      Ngôn ngữ C là gì, ưu và nhược điểm, phạm vi ứng dụng?**

Ngôn ngữ lập trình C là gì

Ngôn ngữ lập trình C là ngôn ngữ lập trình có mục đích chung, liên quan chặt chẽ đến cách máy tính hoạt động. Hiểu cách hoạt động của bộ nhớ máy tính là một khía cạnh quan trọng của ngôn ngữ lập trình C. Mặc dù nhiều người nghĩ rằng C là một ngôn ngữ khá khó học nhưng thực tế C là một ngôn ngữ rất đơn giản, với khả năng mạnh mẽ.

C là một ngôn ngữ rất phổ biến, và nó là ngôn ngữ của nhiều ứng dụng như Windows, trình thông dịch Python, Git và nhiều thứ khác nữa.

Tại sao nên học lập trình C

 Ngôn ngữ lập trình C đã tồn tại được 30 năm và có rất nhiều mã nguồn. Điều này có nghĩa là có rất nhiều thứ để học hỏi và rất nhiều thứ để sử dụng. Như vậy có nghĩa là nó đã được nghiên cứu 1 cách rõ ràng và bạn có thể tìm thấy nhiều hướng dẫn có sẵn. Ngoài ra, với C, bạn nhận được rất nhiều ý kiến mạnh mẽ kết hợp với thông tin chi tiết mà bạn có thể hiểu được.

Lập trình C để làm gì

Bạn sẽ hiểu cách một máy tính hoạt động. Hiểu C cho phép bạn viết các chương trình mà bạn chưa bao giờ nghĩ là có thể làm ra trước đó (hoặc ít nhất, bạn sẽ có hiểu biết rộng hơn về kiến trúc và lập trình máy tính nói chung).

C là ngôn ngữ lập trình của lingua.

Hầu như tất cả các ngôn ngữ lập trình bậc cao như Java, Python, JavaScript,… có thể giao tiếp với lập trình C. Ngoài ra, nó là một ngôn ngữ tốt để thể hiện những ý tưởng chung trong lập trình. Không quan trọng nếu người bạn đang nói chuyện không biết C, bạn vẫn có thể truyền đạt ý tưởng của mình theo cách họ có thể hiểu được.

Cơ hội để làm việc trên các dự án nguồn mở có tác động đến hàng triệu người.

 Lúc đầu, bạn có thể bỏ qua thực tế rằng C là một ngôn ngữ quan trọng. Nếu bạn cần phát triển một ứng dụng di động, bạn cần Java (cho Android), Swift và Objective C (cho iOS). Và có hàng chục ngôn ngữ như C#, PHP, Ruby, Python để xây dựng ứng dụng web.

Python được sử dụng để tạo phạm vi rộng cho các ứng dụng. Và C được sử dụng để tạo Python. Nếu bạn muốn đóng góp cho Python, bạn cần biết lập trình C để làm việc trên trình thông dịch Python tác động đến hàng triệu lập trình viên Python. Đây chỉ là một ví dụ. Một số lượng lớn các phần mềm mà bạn sử dụng ngày nay được cung cấp bởi C. Một số dự án nguồn mở lớn hơn trong đó lập trình C được sử dụng là Linux Kernel, Python Interpreter, SQLite Database.

Một ngôn ngữ khác thường được sử dụng cho dự án mã nguồn mở lớn là C++. Nếu bạn biết C và C++, bạn có thể đóng góp cho các dự án nguồn mở lớn tác động đến hàng trăm triệu người.

Ưu điểm:

1. Ngôn ngữ lập trình C là một ngôn ngữ mạnh, mềm dẻo và có thể truy nhập vào hệ thống, nên thường được sử dụng để viết hệ điều hành, các trình điều khiển thiết bị, đồ họa, có thể xây dựng các phần mềm ngôn ngữ khác , …
2. Ngôn ngữ lập trình C có cấu trúc module, từ đó ta có thể phân hạch hay chia nhỏ chương trình để tăng tính hiệu quả, rõ ràng, dễ kiểm tra trong chương trình.
3. Gần với ngôn ngữ máy với hiệu suất cao, hỗ trợ trực tiếp cho phần cứng, phổ biến trong lĩnh vực nhúng.

Nhược điểm:

Cú pháp phức tạp, ít hỗ trợ cho lập trình hướng đối tượng và dễ gặp lỗi bảo mật.

Lập trình C còn không có khả năng kiểm tra thời gian chạy: Ngôn ngữ C thường không hiển thị lỗi mã sau mỗi dòng mã. Thay vào đó, tất cả các lỗi sẽ được trình biên dịch trình bày sau khi chương trình được viết. Nhược điểm này gây khó khăn trong việc kiểm tra mã đối với chương trình lớn.

Tính năng Namespace: C thiếu các tính năng của vùng tên, có nghĩa là không thể sử dụng lại cùng một tên biến trong một phạm vi. Điều này đồng nghĩa với việc  không thể khai báo hai biến có cùng tên.

Không có khái niệm về hàm hủy và hàm tạo: Vì C không hướng đối tượng nên nó không cung cấp các tính năng hàm tạo và hàm hủy. Việc tạo hoặc hủy một biến trong C phải được thực hiện thủ công thông qua hàm hoặc bằng các phương tiện khác.

Phạm vi ứng dụng của ngôn ngữ lập trình C:

* Ngôn ngữ lập trình C được sử dụng khá rộng rãi trong [lập trình game](https://aptech.fpt.edu.vn/lap-trinh-game.html) các hệ thống nhúng
* Ngôn ngữ C được dùng để phát triển System Apps
* Ngôn ngữ C còn có thể phát triển các ứng dụng của máy tính để bàn
* Ngôn ngữ C được dùng để phát triển các phần mềm ứng dụng nổi tiếng như trình duyệt Chromium và Adobe
* Ngôn ngữ C được dùng để phát triển hệ điều hành Windows của Microsoft và Symbian, OSX của Apple
* Ngoài ra, C còn được dùng để phát triển máy tính bàn, sản xuất trình biên dịch và được sử dụng rộng rãi trong các ứng dụng IOT

**2.   Biến, hằng, toán tử, macro, kiểu dữ liệu**

2.1. Biến

Biến (variables) là các giá trị được lưu trữ trong vùng nhớ dữ liệu và thay đổi được khi chạy chương trình.

Tính chất của mỗi biến:

* Tên biến: Dùng để phân biệt với một biến khác trong cùng một khối { }
* Kiểu biến: Xác định kích thước (tính bằng số byte) của biến trong bộ nhớ. Các kiểu biến trong C/C++ được cho trong bảng sau:

|  |  |  |
| --- | --- | --- |
| Kiểu | Kích thước | Miền giá trị |
| char | 1 | -128 ... 127 |
| short | 2 | -32,768 ... 32,767 |
| int | 4 | –2,147,483,648 ... 2,147,483,647 |
| unsigned | 4 | 0 … 4,294,967,295​ |
| long | 4 | –2,147,483,648 ... 2,147,483,647 |
| long long | 8 | –9,223,372,036,854,775,808 ... 9,223,372,036,854,775,807​ |
| float | 4 | ±3.4E±38​​ |
| double | 8 | ±1.7E±308​ |
| long double | 16 | ±1.18973E±4932 |

Khai báo sử dụng biến:

* Cú pháp khai báo:

          kiểu\_dữ\_liệu tên\_biến;

 Hoặc: kiểu\_dữ\_liệu tên\_biến1, ..., tên\_biếnN;

* Vd: int x; Float y,z,t;

Khai báo và khởi tạo giá trị cho biến:

* Cú pháp:

          kiểu\_dữ\_liệu tên\_biến = giá\_trị\_ban\_đầu;

 Hoặc: kiểu\_dữ\_liệu biến1=giá\_trị1, biếnN=giá\_trịN;

- Vd: int a=5; Float x=6, y=3.4;

c) Cách đặt tên biến trong C

Trong ngôn ngữ C, mỗi biến có một tên gọi, nhưng không phải muốn đặt tên thế nào tùy theo ý thích của bạn cũng được. Dưới đây là một số nguyên tắc khi đặt tên cho biến:

* Tên biến chỉ có thể chứa chữ cái (viết hoa hoặc viết thường), chữ số và ký tự \_ (gạch dưới).
* Tên biến phải bắt đầu bằng một chữ cái.
* Không được sử dụng khoảng trắng hoặc các ký tự đặc biệt như !, @, #, ...
* Bạn cũng không được phép sử dụng chữ cái mang dấu trọng âm. (ví dụ é è ê...).
* Tên biến không được trùng với những từ khóa trong C.
* Và một điều hết sức quan trọng mà bạn cần phải nắm đó là trong ngôn ngữ C có sự khác nhau giữa chữ thường và chữ in hoa: chieu\_dai, CHIEU\_DAI và CHieu\_DaI là tên của 3 biến khác nhau trong ngôn ngữ C.

2.2 Hằng

a)

Hằng (constants) là các giá trị không thay đổi được khi chạy chương trình

|  |  |
| --- | --- |
| Phân biệt các hằng | |
| 5 | Hằng số nguyên thuộc kiểu int |
| 05 | Hằng số nguyên thuộc kiểu int (hệ đếm 8) |
| 0x05 | Hằng số nguyên thuộc kiểu int (hệ đếm 16) |
| 5u | Hằng số nguyên thuộc kiểu unsigned (không dấu) |
| 5.0 | Hằng số thực thuộc kiểu double |
| '5' | Hằng ký tự có giá trị số (mã ASCII) bằng 53 |
| "5" | Hằng xâu ký tự gồm ký tự '5' và ký tự NULL |

|  |  |
| --- | --- |
| Một vài ký tự đặc biệt | |
| \n | Ký tự xuống dòng |
| \t | Ký tự TAB |
| \' | Ký tự nháy đơn |
| \" | Ký tự nháy kép |
| \" | Ký tự nháy kép |
| \\ | Ký tự xổ trái |
| \0 | Ký tự NULL (kết thúc xâu ký tự) |

b) Khai báo hằng:

* Cách 1: Dùng từ khóa #define
* Cú pháp:

# define tên\_hằng giá\_trị

* Ví dụ:

#define NGUYEN THI NHAT LE 50

#define BKHN 100

* Cách 2: Dùng từ khóa const
* Cú pháp:

const kiểu\_dữ\_liệu tên\_hằng = giá\_trị;

* Ví dụ:

const int MAX\_SINH\_VIEN = 74;

const char BKHN[20] = “vua hoc vua choi";

2.3 Toán tử

1. Các toán tử tăng, giảm

|  |  |  |  |
| --- | --- | --- | --- |
| Toán tử | Tên gọi | Biểu thức | Tương đương |
| ++ | cộng trước | b = ++a | a = a + 1, b = a |
| ++ | cộng sau | b = a++ | b = a, a = a + 1 |
| -- | trừ trước | b = --a | a = a - 1, b = a |
| -- | trừ sau | b = a-- | b = a, a = a - 1 |

1. Các toán tử so sánh

|  |  |  |  |
| --- | --- | --- | --- |
| Toán tử | Tên gọi | Biểu thức | Giá trị |
| == | bằng với | 7 == 3 | 0 |
| != | khác với | 7 != 3 | 1 |
| > | lớn hơn | 7 > 3 | 1 |
| < | nhỏ hơn | 7 < 3 | 0 |
| >= | lớn hơn hoặc bằng | 7 >= 3 | 1 |
| <= | nhỏ hơn hoặc bằng | 7 <= 3 | 0 |

1. Các toán tử logic

|  |  |  |  |
| --- | --- | --- | --- |
| Toán tử | Tên gọi | Biểu thức | Ý nghĩa |
| && | và | ('z' >= x && x >= 'a') | kiểm tra trong vùng ký tự thường |
| || | hoặc | (x < '0' || '9' < x) | kiểm tra ngoài vùng chữ số |
| ! | phủ định | !(x != 0) | tương đương: x == 0 |

1. Các toán tử xử lý bit

|  |  |  |  |
| --- | --- | --- | --- |
| Toán tử | Tên gọi | Biểu thức | Giá trị |
| & | AND | 2 & 5 | 0 |
| | | OR | 2 | 5 | 7 |
| ~ | NOT | ~5 | -6 |
| ^ | XOR | 2 ^ 5 | 7 |
| >> | dịch phải | 5 >> 1 | 2 |
| << | dịch trái | 5 << 1 | 10 |

1. Toán tử kiểm tra

(biểu thức logic ? biểu thức nếu đúng : biểu thức nếu sai)

2.4 Macro

* Marco là 1 tên bất kì (do lập trình viên đặt tên) trỏ tới 1 khối lệnh thực hiện một chức năng nào đó.
* Trong quá trình tiền xử lí (pre-processor), các macro được sử dụng trong chương trình được thay thế bởi các khối câu lệnh tương ứng.
* Định nghĩa macro bằng lệnh #define

2.5 Kiểu dữ liệu

Các hàm vào ra cơ bản

* Hàm printf()
* Cú pháp:  printf(xâu\_định\_dạng, [danh\_sách\_tham\_số]);

Xâu định dạng: Là xâu dùng để quy định cách thức hiển thị dữ liệu ra

màn hình máy tính. Các xâu định dạng chứa:

\*) Các kí tự thông thường: Được hiển thị ra màn hình.

\*) Các nhóm kí tự định dạng: Xác định quy cách hiển thị các tham số

trong phần danh\_sách\_tham\_số.

\*) Các ký tự điều khiển: Dùng để tạo các hiệu ứng hiển thị đặc biệt

như xuống dòng ('\n') hay sang trang ('\f')...

Danh sách tham số: Danh sách các biến sẽ được hiển thị giá trị lên

màn hình theo cách thức được quy định trong xâu\_định\_dạng.

• Một số nhóm kí tự định dạng phổ biến

• Mỗi nhóm kí tự định dạng chỉ dùng cho một kiểu dữ liệu, còn một kiểu

dữ liệu có thể hiển thị theo nhiều cách khác nhau nên có nhiều nhóm

kí tự định dạng khác nhau.

• Nếu giữa nhóm kí tự định dạng và tham số tương ứng không phù hợp

với nhau thì sẽ hiển thị ra kết quả không như ý.

A close-up of a text

Description automatically generatedA close up of a text

Description automatically generated

* Hàm Scanf():
* Cú pháp:

scanf(xâu\_định\_dạng, [danh\_sách\_địa\_chỉ]);

* VD:

scanf("%d%f", &a, &b);

Địa chỉ của một biến được viết bằng cách đặt dấu & trước tên biến.

A close-up of a text

Description automatically generated

**3.  Các cấu trúc điều kiện, vòng lặp.**

3.1 Cấu trúc điều kiện

1. Cấu trúc IFA diagram of a diagram

   Description automatically generated

* Cú pháp:

if (bieu\_thuc\_dieu\_kien)

     lenh;

1. Cấu trúc IF… ELSE

A diagram of a diagram

Description automatically generated

* Cú pháp:

if (bieu\_thuc\_đieu\_kien)

        lenh\_1;

else

        lenh\_2;

1. Cấu trúc SWITCH… CASE

A diagram of a flowchart

Description automatically generated

* Cú pháp:

switch (biểu thức) {

case hằng\_1:

    khối biểu thức 1

    break;

case hằng\_2:

    khối biểu thức 2

    break;

case hằng\_n:

    khối biểu thức n

    break;

default:

    khối biểu thức

    cho các trường hợp còn lại

}

3.2 Vòng lặpA diagram of a diagram

Description automatically generated

1. Cấu trúc FOR

* Cú pháp:

for (biểu thức khởi tạo; biểu thức logic; biểu thức biến đổi) {

    khối biểu thức

}

A diagram of a diagram

Description automatically generated

1. Cấu trúc WHILE

* Cú pháp:

while (biểu thức logic) {

    khối biểu thức

}

     - Kiểm tra điều kiện vòng lặp (tức là giá trị của biểu thức)

 trước rồi  mới thực hiện lệnh.

     - Các lệnh sau while có thể không được thực hiện lần nào.

c) Cấu trúc DO…WHILEA diagram of a diagram

Description automatically generated

* Cú pháp:

do {

    khối biểu thức

} while (biểu thức logic);

    − Thực hiện lệnh trước rồi mới kiểm tra dieu\_kien của

 vòng lặp.

   − Các lệnh sau do được thực hiện ít nhất 1 lần dù bieu\_thuc

có giá trị như thế nào.

Câu lệnh thay đổi cấu trúc lập trình

* continue

• Bỏ qua việc thực hiện các câu lệnh nằm sau lệnh continue trong

thân vòng lặp.

• Chuyển sang thực hiện một vòng lặp mới

* break

• Thoát khỏi vòng lặp ngay cả khi biểu thức điều kiện của vòng lặp

vẫn còn được thỏa mãn.

**4.   Cấu trúc dữ liệu tự định nghĩa: Struct, union**

4.1 STRUCT

Struct hay cấu trúc là một kiểu dữ liệu mà tự bạn định nghĩa ra bằng cách gộp nhiều kiểu dữ liệu có sẵn lại nhằm mục đích có thể mô tả nhiều trường thông tin của đối tượng bạn cần lưu

Cú pháp khai báo :

struct ten\_struct{

     data\_type1 data\_field1;

     data\_type2 data\_field2;

     ....

     data\_typen data\_fieldn;

};

Trong struct bạn sẽ liệt kê các trường thông tin mà bạn cần lưu cho cấu trúc bạn đang xây dựng kèm theo kiểu dữ liệu của trường thông tin đó

b) UNION

Union cũng là kiểu dữ liệu mà người dùng tự định nghĩa tương tự như struct, union cũng có các thành phần dữ liệu để mô tả thông tin của đối tượng mà bạn đang muốn lưu trữ

Tuy nhiên điểm khác biệt với struct đó là struct thì các thành phần của nó có thể lưu trữ giá trị đồng thời tại cùng một thời điểm còn union thì ngược lại, tại mỗi thời điểm bạn chỉ có một thuộc tính của union lưu trữ 1 giá trị nào đó.

Tất cả các thành phần của union sẽ chia sẻ chung 1 vùng nhớ

Cú pháp :

union union\_name{

    data\_type1 member1;

    data\_type2 member2;

    data\_type3 member3;

    ....

    data\_typeN memberN;

};

Để truy cập các thành phần của union bạn cũng dùng toán tử '.' với biến union bình thường hoặc toán tử '->' với kiểu con trỏ union

c) So sánh Struct và Union

Union và struct giống nhau bao gồm :

* Đều là kiểu dữ liệu mà người dùng tự định nghĩa ra nhằm phục vụ các bài toán thực tế
* Có các thuộc tính thành phần
* Truy cập thuộc tính thông qua toán tử '.' hoặc toán tử '->' đối với biến kiểu con trỏ

Sự khác nhau giữa 2 kiểu dữ liệu người dùng định nghĩa được liệt kê dưới đây :

|  |  |
| --- | --- |
| Struct | Union |
| K Kích thước của struct thường lớn hơn hoặc bằng tổng tất cả các kích thước của các thuộc tính thành phần | Kích thước của union bằng với kích thước của thuộc tính có kích thước lớn nhất của nó |
| St Struct có thể lưu trữ đồng thời nhiều giá trị cho các thuộc tính thành phần ở cùng một thời điểm | Chỉ có 1 thuộc tính thành phần của union có thể lưu giá trị tại 1 thời điểm |
| K Khai báo với từ khóa struct | K Khai báo với từ khóa union |

**5. Con trỏ, mảng.**

 5.1 Mảng

1. Khái niệm:

 • Mảng là một tập hợp hữu hạn các phần tử có cùng kiểu dữ liệu được lưu trữ kế tiếp nhau trong bộ nhớ.

 • Các phần tử trong mảng có cùng tên (và cũng là tên mảng) nhưng phân biệt với nhau ở chỉ số cho biết vị trí của chúng trong mảng.

1. Khai báo

* Cú pháp: kiểu\_dữ\_liệu tên\_mảng [kích\_thước\_mảng];

Trong đó:

kiểu\_dữ\_liệu là kiểu dữ liệu của các phần tử trong mảng.

tên\_mảng là tên của mảng.

kích\_thứớc\_mảng cho biết số phần tử trong mảng.

VD:

int mang\_nguyen[10]; // khai báo mảng 10 phần tử có kiểu dữ liệu int float mang\_thuc[4]; // khai báo mảng 4 phần tử có kiểu dữ liệu float char mang\_ki\_tu[6]; // khai báo mảng 6 phần tử có kiểu dữ liệu char

* Mảng một chiều
* Đn: Mỗi phần tử của mảng cũng là một mảng khác => Giống vector trong toán học
* VD: Mảng 2 chiều: int a[6][5]; Mảng 3 chiều: int b[3][4][5];

1. Sử dụng mảng

  Để truy cập vào một phần tử của mảng thông qua tên mảng và chỉ số của phần tử đó

* Cú pháp:

           ten\_mang[chi\_so\_cua\_phan\_tu]

VD1:

 int mang\_nguyen[3];

Trong đó:

mang\_nguyen[0]: Phần tử thứ 1.

mang\_nguyen[1]: Phần tử thứ 2.

mang\_nguyen[2]: Phần tử thứ 3.

VD2:

int a[6][5];

Trong đó:

 - a[0] là phần tử đầu tiên của mảng, là 1 mảng

            - Phần tử đầu tiên của mảng a[0] là a[0][0],...

 - …

 - a[2][3] sẽ là phần tử thứ 4 của phần tử thứ 3 của a.

 - a[i][j] sẽ là phần tử thứ j+1 của a[i], mà phần tử a[i] lại là phần tử thứ i+1 của a.

1. Nhập dữ liệu cho mảng

Nhập dữ liệu cho từng phần của mảng

* VD1: Dùng hàm scanf() hoặc lệnh gán thông thường

float a[10]; // khai bao mot mang so thuc co 10 phan tu

int i; // Nhap tu ban phim mot so thuc va gan gia tri so thuc do

 // cho phan tu thu 2 cua mang, tuc la a[1]

 scanf(“%f”,&a[1]); // Gán giá trị cho phần tử a[2] a[2] = a[1] + 5;

* VD2: Dùng lệnh for

int b[10];

int i;

// Nhap gia tri tu ban phim cho tat ca cac phan tu cua mang

for(i = 0; i < 10; i++) {

 printf(“\n Nhap gia tri cho b[%d]”, i);

 scanf(“%d”,&b[i]);

}

1. Xuất dữ liệu cho mảng

Dùng hàm printf()

5.2 Con trỏ

1. Khái niệm con trỏ

* Địa chỉ và giá trị của một biến
* Bộ nhớ như một dãy các byte nhớ.
* Các byte nhớ được xác định một cách duy nhất qua một địa chỉ.
* Biến được lưu trong bộ nhớ.
* Khi khai báo một biến:

• Chương trình dịch sẽ cấp phát cho biến đó một số ô nhớ liên tiếp đủ để chứa nội dung của biến. Ví dụ một biến số nguyên (int) được cấp phát 2 byte.

 • Địa chỉ của một biến chính là địa chỉ của byte đầu tiên trong số đó.

* Một biến luôn có hai đặc tính:

• Địa chỉ của biến.

• Giá trị của biến.

* Khái niệm:

Con trỏ là một biến mà giá trị của nó là địa chỉ của một vùng nhớ.

* Cú pháp:

Kieu\_du\_lieu \*ten\_bien\_con\_tro;

VD:

int i = 3;

int \*p;

p = &i;

 => Một con trỏ chỉ có thể trỏ tới một đối tượng cùng kiểu.

1. Toán tử & và \*

* Toán tử &: Trả về địa chỉ của biến.
* Toán tử \*: Trả về giá trị chứa trong vùng nhớ được trỏ bởi con trỏ.
* Cả hai toán tử \* và & có độ ưu tiên cao hơn tất cả các toán tử số học ngoại trừ toán tử đảo dấu.

1. Gán giá trị cho con trỏ

Biến con trỏ được gán bởi địa chỉ của biến khác

 pointer\_variable = &variable;

       Biến con trỏ được gán bởi một con trỏ khác( tốt nhất là cùng kiểu)

                        pointer\_variable2 = pointer\_variable;

Giá trị NULL cũng có thể được gán đến một biến con trỏ bằng số 0:

            pointer\_variable = 0;

Các biến cũng có thể được gán giá trị thông qua con trỏ của chúng

\*pointer\_variable = 10;

1. Các phép toán số học có thể thực hiện trên con trỏ là:

* Cộng con trỏ với một số nguyên (int, long) và kết quả là một con trỏ cùng kiểu.
* Trừ con trỏ với một số nguyên và kết quả là một con trỏ cùng kiểu.
* Trừ hai con trỏ cùng kiểu cho nhau, kết quả là một số nguyên. Kết quả này nói lên khoảng cách (số phần tử thuộc kiểu dữ liệu của con trỏ) ở giữa hai con trỏ.

 => phép toán cộng hai con trỏ, và nhân chia, lấy phần dư trên con trỏ là không hợp lệ.

1. Con trỏ void

* Khai báo:

  void \*con\_trỏ;

* Đây là con trỏ đặc biệt, con trỏ không có kiểu, nó có thể nhận giá trị là địa chỉ của một biến thuộc bất kỳ kiểu dữ liệu nào.
* Con trỏ void được dùng làm đối để nhận bất kỳ địa chỉ nào từ tham số của các lời gọi hàm.
* Các lệnh sau đây là hợp lệ: void \*p, \*q; int x = 21; float y = 34.34; P = &x; q = &y;

1. Con trỏ và mảng 1 chiều

* a chính là một hằng địa chỉ và là địa chỉ của phần tử đầu tiên của mảng. Với một mảng có tên là a thì a là một địa chỉ và a có giá trị bằng &a[0].
* Xét khai báo mảng tab gồm 10 số nguyên sau:

 int a[10], \*p;

nếu p = a; khi đó

p+1 sẽ trỏ tới phần tử cùng kiểu ngay sau phần tử đầu tiên của mảng, chính là a[1], nghĩa là \*(p+1) chính là a[1].

p+2 sẽ trỏ tới a[2], hay \*(p+2) là a[2]

 ....

p+i sẽ trỏ tới a[i]

 => Dùng con trỏ để duyệt các phần tử trong mảng

* Ví dụ:

#define N 5

int tab[5] = {1, 2, 6, 0, 7};

main() {

 int i;

 int \*p;

 p = tab;

 for (i = 0; i < N; i++){

    printf(" %d \n",\*p);

    p++;

}

}

* \*(p+i) để chỉ nội dung phần tử thứ i tính từ đầu mảng (trỏ bởi p) Cụ thể là p[i] có vai trò như \*(p+i) và cũng là phần tử thứ i của mảng được trỏ bởi p.
* Lưu ý:

+ Con trỏ luôn cần phải được khởi tạo, hoặc bằng cách gán cho nó một địa chỉ nào đó, hoặc qua thao tác cấp phát động bộ nhớ.

+ Một (tên) mảng là một hằng địa chỉ, nó không bao giờ có thể nằm ở vế trái của một phép gán như con trỏ, cũng như chấp nhận các phép toán số học trên nó ví dụ như tab++;

**6.  Hàm, tham chiếu, tham trị**

   6.1 Hàm

         Hàm (Function) - một đoạn chương trình có thể được gọi nhiều lần trong đoạn chương trình khác hoặc trong chính nó (hàm đệ quy).

Đầu vào (Inputs) - các thành phần không cố định trong các biểu thức của hàm.

Đầu ra (Outputs) - kết quả nhận được sau khi kết thúc hàm.

Ưu điểm

* Giảm dung lượng chương trình
* Thuận tiện trong việc chỉnh sửa
* Cho phép module hóa chương trình

1. Khái niệm chương trình con

* Các đoạn chương trình lặp đi lặp lại ở nhiều vị trí khác nhau

=>thay thế các đoạn chương trình đó bằng các chương trình con tương ứng

và gọi chúng khi cần thiết, thay vì phải viết lại cả đoạn chương trình đó.

* Các chương trình con được tạo sẵn: sin(), cos(),exp(), pow(), sqrt()…..

 => Chương trình con tiêu chuẩn

* Bài toán lớn và phức tạp => chia thành các bài toán nhỏ và tương đối

độc lập=>lập các chương trình con => ghép các chương trình lại về bài

toán ban đầu

=> phương pháp lập trình cấu trúc.

1. Phân loại chương trình con

* 2 loại chương trình con
* hàm (function) : trả về giá trị
* thủ tục (procedure) : không trả về giá trị
* Ngôn ngữ C chỉ cho phép chương trình con là Hàm
* Hàm luôn có giá trị để trả về.
* Nếu không có giá trị để trả về (giống Thủ tục) => khai báo hàm có giá trị

trả về là “ không là kiểu giá trị nào cả” => kiểu void trong C

1. Hàm khai báo

Khai báo của một hàm được chia làm 2 phần:

- Dòng đầu hàm:

[<kiểu giá trị trả về>] <tên hàm>([<danh sách tham số>,...])

- Thân hàm: là tập hợp các khai báo và câu lệnh đặt trong cặp dấu ngoặc nhọn

{

<Các khai báo>

...

<Các câu lệnh>

}

Trong thân hàm có ít nhất một lệnh return.

Ví dụ:

int giai\_thua(int a) {

int ket\_qua;

int i;

ket\_qua = 1;

for(i = 0;i<a;i++)

ket\_qua = ket\_qua \* i;

if(a < 0) ket\_qua = -1;

if(a == 0) ket\_qua = 0;

return ket\_qua;

}

Các thành phần của dòng đầu hàm

* Dòng đầu hàm là các thông tin được trao đổi giữa bên trong và bên ngoài hàm.
* Dòng đầu hàm phân biệt các hàm với nhau, hay nói cách khác không được có
* 2 hàm có dòng đầu hàm giống nhau.
* Các thành phần
* Kiểu dữ liệu trả về của hàm

     + Tên hàm

     + Tham số của hàm

     + Lệnh return

1. Sử dụng hàm

Cú pháp:

<tên hàm> ([danh sách các tham số])

Lưu ý:

- Nếu có nhiều tham số trong danh sách tham số thì các tham số được phân

cách với nhau bằng dấu phẩy

- Cho dù hàm có một, nhiều hay không có tham số thì vẫn luôn luôn cần cặp

ngoặc đơn đứng sau tên hàm

d) Phân loại biến và sử dụng trong chương trình

\*) Phân loại:

* Biến cục bộ ( biến địa phương – local variable):Là các biến được khai báo

trong lệnh khối hoặc trong thân chương trình con. Việc khai báo các biến cục

bộ phải được đặt trước phần câu lệnh trong lệnh khối hoặc trong chương trình

con.

* Biến toàn cục (global variable): Là biến được khai báo trong chương trình

chính. Vị trí khai báo của biến toàn cục là sau phần khai báo tệp tiêu đề và khai

báo hàm nguyên mẫu.

\*) Lưu ý:

o Hàm main() cũng chỉ là một chương trình con, nhưng nó là chương trình con

đặc biệt ở chỗ chương trình được bắt đầu thực hiện từ hàm main().

o Biến khai báo trong hàm main() không phải là biến toàn cục mà là biến cục bộ

của hàm main().

1. Con trỏ hàm trong C

* Con trỏ hàm là gì?
* Con trỏ trong C: là một biến được dùng để lưu trữ địa chỉ của dữ liệu

trong bộ nhớ máy tính

* Con trỏ hàm trong C: lưu trữ địa chỉ của một hàm và được sử dụng để

truy cập vào địa chỉ của hàm, cũng như thực thi các xử lý bên trong hàm

đó.

* Khai báo con trỏ hàm trong C
* Cú pháp:

type (\*fp) ( para\_type\_1, para\_type\_2, para\_type\_3,...);

trong đó: fp là tên con trỏ hàm,

type là kiểu của con trỏ

các para\_type là kiểu của các đối số (parameter) được truyền vào hàm.

* Lưu ý: kiểu type của con trỏ phải giống với kiểu giá trị trả về của hàm mà nó gán địa chỉ.
* Gán địa chỉ hàm cho con trỏ
* Cú pháp:

fp = funct;

trong đó: fp là tên con trỏ hàm

funct là tên hàm (function) cần gán địa chỉ.

* Gọi một hàm bằng con trỏ
* Cú pháp:

(\*fp) ( para\_1, para\_2, para\_3,...);

 Trong đó:

fp là tên con trỏ hàm, và các paralà các đối số (parameter) được truyền vào hàm.

6.2 Tham chiếu

Tham chiếu (Reference) là một loại biến trong C++ được thiết kế để tránh những rắc rối của con trỏ trong ngôn ngữ C. Có thể hiểu tham chiếu như là đại diện của duy nhất một biến khác.

kiểu& tên\_tham\_chiếu = biểu thức;

Trong cấu trúc hàm, tham chiếu được sử dụng trong các trường hợp sau:

* Kiểu trả về của một hàm đọc/ghi
* Là một tham biến

6.3 Tham trị

Truyền tham trị là truyền cho đối số một bản sao.

**II.         Embedded C**

**1.      Sự giống và khác nhau giữa C thông thường với Embedded C**

Thực tế không có nhiều khác biệt giữa C và Embedded C ngoài một vài phần mở rộng và môi trường hoạt động. Cả C và Embedded C đều là các tiêu chuẩn ISO có cú pháp, kiểu dữ liệu, hàm gần như giống nhau, v.v.

Embedded C về cơ bản là một phần mở rộng cho Standard C Programming với các tính năng bổ sung như Addressing I/O, multiple memory addressing và fixed-point arithmetic.. v.v

C Programming thường được sử dụng để phát triển các ứng dụng máy tính để bàn trong khi Embedded C được sử dụng để phát triển các ứng dụng dựa trên Vi điều khiển.

**2.      Lưu ý khi lập trình C embedded**

Đặc điểm đối với hệ thống nhúng:

* ROM và RAM hạn chế.
* Lập trình phụ thuộc phần cứng.
* Cần đáp ứng chính xác về thời gian (hàm xử lý ngắt, tác vụ…)
* Nhiều kiểu pointer (far/rom/ui/paged/…)
* Một số keywords và token đặc biệt (@, interrupt, tiny,..)

Để phát triển tốt phần mềm nhúng bằng ngôn ngữ C cần nắm vững.

* Thiết kế kiến trúc phần mềm hợp lý.
* Thành thạo sử dụng các tool và debugging
* Data types native support
* Các thư viện chuẩn.
* Phân biệt rõ về simple code với eficient code.

Một số điểm có thể tạo ra “sự khác biệt”

* Inline assembly
* Hàm xử lý ngắt.
* Assembly language generation
* Thư viện chuẩn
* Startup code
* Sử dụng các từ khóa near và far để tăng hiệu suất của biến khi biến nằm ở các vùng nhớ gần hoặc xa vùng đang sử dụng

**3.      C memory management**

**Quản lý bộ nhớ** hoặc phân bổ bộ nhớ là quá trình các chương trình máy tính được gán cho không gian bộ nhớ vật lý hoặc ảo. Bộ nhớ máy tính là nguồn tài nguyên hữu hạn cần được quản lý hiệu quả.

Việc cấp phát bộ nhớ có thể được thực hiện trước hoặc tại thời điểm thực hiện chương trình. Có hai kỹ thuật cấp phát bộ nhớ: cấp phát bộ nhớ tĩnh và cấp phát bộ nhớ động.

**Phân bổ bộ nhớ tĩnh**

Trong kiểu phân bổ này, trình biên dịch phân bổ một lượng bộ nhớ cố định trong thời gian biên dịch và hệ điều hành bên trong sử dụng cấu trúc dữ liệu được gọi là ngăn xếp để quản lý bộ nhớ.

Yêu cầu bộ nhớ chính xác phải được biết trước vì một khi bộ nhớ được cấp phát thì không thể thay đổi được.

**Phân bổ bộ nhớ động**

Trong kiểu phân bổ này, bộ nhớ hệ thống được quản lý trong thời gian chạy. Quản lý bộ nhớ động trong ngôn ngữ lập trình C được thực hiện bằng cách sử dụng các hàm malloc(), calloc(), realloc()và free(). Bốn hàm này được định nghĩa trong <stdlib.h>tệp tiêu đề thư viện chuẩn C. Nó sử dụng không gian heap của bộ nhớ hệ thống.

|  |  |
| --- | --- |
| **Hàm** | **Mô tả** |
| malloc() | Phân bổ kích thước yêu cầu của byte và trả về một byte đầu tiên của không gian được phân bổ. |
| calloc() | Phân bổ không gian cho một phần tử mảng, khởi tạo về 0 và sau đó trả về một con trỏ tới bộ nhớ. |
| free() | Giải phóng không gian được phân bổ trước đó. |
| realloc() | Thay đổi kích thước của không gian được phân bổ trước đó. |

**4.      Compilation process, toolchain, compiler, linker.**

Quy trình dịch là quá trình chuyển đổi từ ngôn ngữ bậc cao (NNBC) (C/C++, Pascal, Java, C#…) sang ngôn ngữ đích (ngôn ngữ máy) để máy tính có thể hiểu và thực thi. Ngôn ngữ lập trình C là một ngôn ngữ dạng biên dịch. Chương trình được viết bằng C muốn chạy được trên máy tính phải trải qua một quá trình biên dịch để chuyển đổi từ dạng mã nguồn sang chương trình dạng mã thực thi. Quá trình được chia ra làm 4 giai đoạn chính:

* Giai đoàn tiền xử lý (Pre-processor)
* Giai đoạn dịch NNBC sang Asembly (Compiler)
* Giai đoạn dịch asembly sang ngôn ngữ máy (Asember)
* Giai đoạn liên kết (Linker)

1. Giai đoạn tiền xử lý – Preprocessor  
   Giai đoạn này sẽ thực hiện:

* Nhận mã nguồn
* Xóa bỏ tất cả chú thích, comments của chương trình
* Chỉ thị tiền xử lý (bắt đầu bằng #) cũng được xử lý

Ví dụ: chỉ thị #include cho phép ghép thêm mã chương trình của một tệp tiêu để vào mã nguồn cần dịch. Các hằng số được định nghĩa bằng #define sẽ được thay thế bằng giá trị cụ thể tại mỗi nơi sử dụng trong chương trình.

1. Công đoạn dịch Ngôn Ngữ Bậc Cao sang Assembly

* Phân tích cú pháp (syntax) của mã nguồn NNBC
* Chuyển chúng sang dạng mã Assembly là một ngôn ngữ bậc thấp (hợp ngữ) gần với tập lệnh của bộ vi xử lý.

1. Công đoạn dịch Assembly

* Dich chương trình => Sang mã máy 0 và 1
* Một tệp mã máy (.obj) sinh ra trong hệ thống sau đó.

1. Giai đoạn Linker

* Trong giai đoạn này mã máy của một chương trình dịch từ nhiều nguồn (file .c hoặc file thư viện .lib) được liên kết lại với nhau để tạo thành chương trình đích duy nhất
* Mã máy của các hàm thư viện gọi trong chương trình cũng được đưa vào chương trình cuối trong giai đoạn này.
* Chính vì vậy mà các lỗi liên quan đến việc gọi hàm hay sử dụng biến tổng thể mà không tồn tại sẽ bị phát hiện. Kể cả lỗi viết chương trình chính không có hàm main() cũng được phát hiện trong liên kết.

**Kết thúc quá trình tất cả các đối tượng được liên kết lại với nhau thành một chương trình có thể thực thi được (executable hay .exe) thống nhất.**

Toolchain là một tập hợp các công cụ và thành phần phần mềm được sử dụng để phát triển các ứng dụng ngôn ngữ lập trình C. Những công cụ này rất cần thiết để viết, biên dịch, liên kết và gỡ lỗi mã C.

Trước khi các chuỗi công cụ phát triển phần mềm hiện đại tồn tại, quá trình tạo ra phần mềm rất khác biệt và thô sơ hơn. Việc phát triển phần mềm trong những ngày đầu được đặc trưng bởi các nhiệm vụ thủ công và tốn nhiều công sức, các công cụ hạn chế và tập trung vào lập trình dành riêng cho phần cứng. Dưới đây là một số khía cạnh chính của việc phát triển phần mềm trước các chuỗi công cụ như chúng ta biết ngày nay:

* **Mã hóa thủ công:**Các lập trình viên viết phần mềm hoàn toàn bằng mã máy hoặc hợp ngữ, là những ngôn ngữ cấp thấp, dành riêng cho phần cứng. Quá trình này rất dễ xảy ra lỗi và tốn thời gian vì các lập trình viên phải xử lý trực tiếp các biểu diễn nhị phân của lệnh và địa chỉ bộ nhớ.
* **Không có ngôn ngữ cấp cao:**Các ngôn ngữ lập trình cấp cao như C, Python hoặc Java không tồn tại. Các lập trình viên phải làm việc trực tiếp với phần cứng, điều này đòi hỏi sự hiểu biết sâu sắc về kiến ​​trúc máy tính cụ thể.
* **Công cụ gỡ lỗi hạn chế:**Gỡ lỗi là một quá trình đầy thử thách và tốn thời gian. Các lập trình viên thường dựa vào các câu lệnh in hoặc kiểm tra nội dung bộ nhớ theo cách thủ công để xác định và sửa lỗi trong mã của họ.
* **Hợp tác hạn chế:**Việc hợp tác giữa các lập trình viên gặp nhiều thách thức do thiếu hệ thống kiểm soát phiên bản và các công cụ phát triển hợp tác. Các nhóm thường làm việc riêng biệt trên các phần khác nhau của dự án và sau đó tích hợp mã của họ theo cách thủ công.
* **Hạn chế về phần cứng:**Các máy tính đời đầu có bộ nhớ và khả năng xử lý hạn chế. Các lập trình viên phải viết mã hiệu quả cao để tận dụng tối đa các tài nguyên sẵn có.
* **Chu kỳ phát triển dài:**Việc phát triển các ứng dụng phần mềm thậm chí tương đối đơn giản cũng có thể mất nhiều thời gian do tính chất thủ công của việc mã hóa và gỡ lỗi.

**Có hai loại toolchains**

* + Chuỗi công cụ gốc:
* Chuỗi công cụ gốc được thiết kế và tối ưu hóa cho một nền tảng phần cứng hoặc phần mềm cụ thể. Chúng dự định chạy trên cùng một nền tảng mà chúng tạo mã.
* Sử dụng các chuỗi công cụ này trong nhiều ví dụ như ứng dụng máy tính để bàn hoặc trò chơi thiết kế sẽ chạy trên máy tính.
* **Ví dụ:**Các trình biên dịch như GCC (Bộ sưu tập trình biên dịch GNU) hoặc Microsoft Visual C++ là các chuỗi công cụ gốc. Họ biên dịch mã để chạy cụ thể trên hệ thống máy chủ nơi trình biên dịch được cài đặt.
  + Chuỗi công cụ đa nền tảng:
* **Nhắm mục tiêu cho nhiều nền tảng:**Chuỗi công cụ đa nền tảng được thiết kế để hoạt động trên một nền tảng (máy chủ) nhưng tạo mã có thể chạy trên các nền tảng mục tiêu khác nhau. Chúng được sử dụng khi bạn cần phát triển phần mềm có thể triển khai trên nhiều môi trường phần cứng hoặc phần mềm khác nhau như vi điều khiển dựa trên ARM Cortex-M. Sử dụng rộng rãi loại chuỗi công cụ này trong các ứng dụng hệ thống nhúng và chuỗi công cụ trình biên dịch chéo nhận biết bản đồ bộ nhớ và chi tiết về kiến trúc nền tảng phù hợp với việc thực hiện mục tiêu của họ.
* **Ví dụ:**Trình biên dịch GCC cho ARM, đầu ra sẽ chạy trong (Bộ vi điều khiển dựa trên ARM).
* Chuỗi công cụ đa nền tảng được sử dụng rộng rãi trên các hệ thống nhúng, chúng ta sẽ xử lý GNU Arm Embedded Toolchain, chuỗi công cụ này chạy trên các bộ vi điều khiển dựa trên bộ xử lý ARM.

https://www.linkedin.com/pulse/buildingcompilation-process-using-gnu-arm-toolchain-mohamed-ali#:~:text=Toolchain%20is%20a%20set%20of,linking%2C%20and%20debugging%20C%20code.

**5.      Make, build system.**

**Xác định yêu cầu:** Đầu tiên, bạn cần phải hiểu rõ yêu cầu của hệ thống mà bạn muốn xây dựng. Điều này bao gồm việc xác định chức năng, hiệu suất, yêu cầu về kích thước, tiêu thụ năng lượng, và các yếu tố khác.

**Lựa chọn phần cứng:** Dựa trên yêu cầu của hệ thống, bạn có thể chọn các vi điều khiển (microcontroller) phù hợp. Cân nhắc các yếu tố như hiệu suất, tính năng, kích thước, tiêu thụ năng lượng, và sẵn có của các công cụ phát triển.

**Chọn ngôn ngữ lập trình và môi trường phát triển:** C cũng như C++, là lựa chọn phổ biến cho việc lập trình hệ thống nhúng. Xác định môi trường phát triển phù hợp, bao gồm trình biên dịch và các công cụ phát triển tích hợp.

**Thiết kế phần mềm:** Xác định cấu trúc phần mềm, bao gồm việc phân chia chương trình thành các module, viết mã nguồn và thử nghiệm từng phần.

**Thiết kế phần cứng:** Thiết kế mạch điện và PCB (Printed Circuit Board) nếu cần thiết. Điều này bao gồm việc chọn linh kiện, vẽ mạch và kiểm tra tính toàn vẹn của mạch.

**Phát triển và kiểm tra:** Lập trình và triển khai phần mềm trên hệ thống nhúng. Tiến hành kiểm tra, sửa lỗi và tối ưu hóa mã nguồn.

**Kiểm tra tích hợp:** Kiểm tra toàn bộ hệ thống, bao gồm cả phần cứng và phần mềm, để đảm bảo hoạt động đúng đắn và đáp ứng các yêu cầu.

**Triển khai và bảo trì:** Triển khai hệ thống vào môi trường sản xuất và tiếp tục bảo trì, cập nhật và sửa lỗi theo thời gian.

**6.      8bit/16bit/32bit machine**

**8 bit:** Máy 8-bit có khả năng xử lý dữ liệu với các giá trị từ 0 đến 255 (2^8 - 1). Đây là các máy tính cổ điển như các vi xử lý Intel 8080, Zilog Z80, hoặc MOS 6502. Mặc dù ít phổ biến hơn trong các ứng dụng hiện đại, nhưng vẫn được sử dụng trong các hệ thống nhúng đơn giản hoặc trong việc điều khiển các thiết bị cơ bản.

**16 bit:** Máy 16-bit có khả năng xử lý dữ liệu với các giá trị từ 0 đến 65,535 (2^16 - 1). Các ví dụ bao gồm Intel 8086/8088 và các vi xử lý Motorola 68000. Máy 16-bit thường được sử dụng trong các hệ thống nhúng đòi hỏi tính toán đơn giản đến trung bình.

**32 bit:** Máy 32-bit có khả năng xử lý dữ liệu với các giá trị từ 0 đến 4,294,967,295 (2^32 - 1). Đây là một tiêu chuẩn phổ biến trong máy tính hiện đại. Các CPU 32-bit như Intel 80386, ARM Cortex-M3 hoặc các phiên bản sau của vi xử lý Motorola 68000 thường được sử dụng trong các hệ thống nhúng phức tạp hơn và các máy tính cá nhân.

**7.      Tổ chức firmware, kiến trúc phần mềm**

*7.1 Tổ chức firmware*

*7.2 Kiến trúc phần mềm*

Phần mềm hệ thống nhúng là các chương trình được viết, thiết kế, lập trình chuyên biệt cho các thiết bị số và hoạt động song song với các thiết bị đó. Hiện nay có 6 loại kiến trúc phần hệ thống nhúng phổ biến là:

* Vòng lặp kiểm soát đơn giản
* Hệ thống ngắt điều khiển
* Đa nhiệm tương tác
* Đa nhiệm ưu tiên
* Vi nhân (Microkernel) và nhân ngoại (Exokernel)
* Nhân khối (monolithic kernels)

1. **Vòng lặp kiểm soát đơn giản**

* Một vòng lặp kiểm soát là một hệ thống quản lý quá trình được thiết kế để duy trì một quá trình diễn biến tại một vị trí mong muốn.
* Theo thiết kế này thì phần mềm được tổ chức thành một vòng lặp đơn giản. Vòng lặp gọi đến các chương trình con, mỗi chương trình con quản lý một phần của hệ thống phần cứng hoặc phần mềm.
* Mỗi bước trong vòng lặp sẽ hoạt động cùng với các bước khác để quản lý hệ thống.

**b) Hệ thống ngắt điều khiển**

* Hệ thống nhúng thường được điều khiển bằng các ngắt, các tác vụ của hệ thống nhúng được kích hoạt bởi các loại sự kiện khác nhau.
* Ví dụ: một ngắt có thể được sinh ra bởi một bộ định thời sau một chu kỳ được định nghĩa trước, hoặc bởi sự kiện khi cổng nối tiếp nhận được một byte nào đó.
* Hệ thống ngắt điều khiển thường được sử dụng trong các hệ thống có bộ quản lý sự kiện đơn giản, ngắn gọn và cần độ trễ thấp. Thường được thực hiện một tác vụ đơn giản trong một vòng lặp chính.
* Đôi khi, các tác vụ phức tạp hơn sẽ được thêm vào một cấu trúc hàng đợi trong bộ quản lý ngắt để được vòng lặp xử lý sau đó. Lúc này, hệ thống gần giống với kiểu nhân đa nhiệm với các tiến trình rời rạc.

**c) Phần mềm hệ thống nhúng đa nhiệm tương tác**

* Hệ thống đa nhiệm không ưu tiên gần giống với kỹ thuật của vòng lặp kiểm soát đơn giản, chỉ khác ở chỗ trừ việc vòng lặp này được ẩn giấu thông qua một giao diện lập trình API.
* Các nhà lập trình định nghĩa một loạt các nhiệm vụ, mỗi nhiệm vụ chạy tỏng một môi trường riêng của nó, khi không cần thực hiện nhiệm vụ đó thì nó gọi đến các tiến trình con tạm nghỉ bằng cách gọi “pause”, “wait”, “yield”,…
* Đối với hệ thống đa nhiệm tương tác, việc thêm một phần mềm mới được thực hiện dễ dàng hơn bằng cách lập trình một tác vụ mới hoặc thêm vào hàng đợi thông dịch.

**d) Đa nhiệm ưu tiên**

* Hệ thống đa nhiệm ưu tiên thường có một đoạn mã ở mức thấp thực hiện việc chuyển đổi giữa các tác vụ khác nhau thông qua một bộ định thời.
* Đoạn mã này thường nằm ở mức mà hệ thống được coi là có một hệ điều hành và vì thế cũng gặp phải tất cả những phức tạp trong việc quản lý đa nhiệm.
* Bất kỳ tác vụ nào có thể phá hủy dữ liệu của một tác vụ khác đều cần phải được tách biệt một cách chính xác. Việc truy cập tới các dữ liệu chia sẻ có thể được quản lý bằng một số kỹ thuật đồng bộ hóa như hàng đợi thông điệp (message queues), các phương thức truyền tin thị giác (semaphores)…
* Hệ thống này thường được đưa ra là sử dụng một hệ điều hành thời gian thực. Lúc đó, các nhà lập trình có thể tập trung vào việc phát triển các chức năng của thiết bị chứ không cần quan tâm đến các dịch vụ của hệ điều hành nữa.

**e) Vi nhân (Microkernel) và nhân ngoại (Exokernel)**

* **Vi nhân (Microkernet)**
  + Vi nhân (Microkernet) là một bước tiếp cận gần hơn tới khái niệm hệ điều hành thời gian thực. Lúc này, nhân hệ điều hành thực hiện việc cấp phát bộ nhớ và chuyển CPU cho các luồng thực thi.
  + Còn các tiến trình người dùng sử dụng các chức năng chính như hệ thống file, giao diện mạng lưới,…
  + Kiến trúc này thường được áp dụng trong các hệ thống mà việc chuyển đổi và giao tiếp giữa các tác vụ là nhanh.
* **Nhân ngoại (Exokernel)**
  + Nhân ngoại (Exokernel) là tiến hành giao tiếp hiệu quả bằng cách sử dụng các lời gọi chương trình con thông thường.
  + Phần cứng và toàn bộ phần mềm trong hệ thống luôn đáp ứng và có thể được mở rộng bởi các ứng dụng.

**f) Nhân khối (Monolithic kernels)**

Nhân khối đầy đủ với các khả năng phức tạp được chuyển đổi để phù hợp với môi trường nhúng. Giúp các nhà lập trình có được một môi trường giống với hệ điều hành trong các máy để bàn như Microsoft Windows, Linux,…

Tuy nhiên, nhân khối đòi hỏi đáng kể các tài nguyên phàn cứng làm tăng chi phí cho hệ thống. Mặc dù ch phí phần cứng tăng nhưng các loại hệ thống nhúng này đang phát triển rất mạnh. Bởi vì:

* Hệ thống có cổng kết nối đến các chip nhúng thông dụng.
* Cho phép sử dụng lại các đoạn mã sẵn như các trình điều khiển thiết bị, web servers, firewalls,…
* Hệ thống có thể phát triển với một tập nhiều loại đặc tính, chức năng. Sau khi phân phối sản phẩm, hệ thống có thể được cấu hình để loại bỏ một số chức năng không cần thiết.
* Có chế độ người dùng để dễ dàng chạy các ứng dụng và gỡ rối, quy trình được thực hiện dễ dàng hơn, lập trình có tính linh động.

Một số nhân khối thông dụng hiện nay là Embedded Linux và Windows CE, đặc biệt là trong các thiết bị nhúng mạnh như Wireless router hoặc hệ thống định vị GPS.

**III. Embedded C for 8051**