1. Tìm hiểu về lập trình C cơ bản.
2. Ngôn ngữ C là gì, ưu và nhược điểm, phạm vi ứng dụng?

C là một ngôn ngữ lập trình phổ biến nhất thế giới, là ngôn ngữ đơn giản và linh hoạt khi sử dụng. Nó là một ngôn ngữ lập trình có cấu trúc độc lập và được sử dụng rộng rãi để viết các ứng dụng, hệ điều hành như Windows và nhiều chương trình phức tạp khác như Oracle database, Git, Python Interpreter,…

Ngoài ra, rất nhiều lập trình viên khi **học lập trình C** đều ví C là “ngôn ngữ mẹ”. Bởi C là cơ sở, nền tảng cho các ngôn ngữ khác và nếu lập trình viên **học lập trình C** giỏi thì các ngôn ngữ khác như C++, C#, Java đều có thể chinh phục dễ dàng.

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

A variable in C language is the name associated with some memory location to store data of different types. ( Biến )

Cách định dạng biến

Data\_type + Variable\_Name = value ; Với định dạng 1 biến

Data\_type + var1, var2, var3 ; Với định dạng nhiều biến

Ví dụ : int number = 1 ;

Constant – Hằng số, chúng ta không thể thay đổi giá trị của hàm số sau khi chúng được định nghĩa bởi 1 câu lệnh

Const data\_type var\_name = value

Ví dụ : const float num1 = 14.8 ;

int main ()

{

const int number = 10 ;

printf (“ The number is : %d \n “ , number ) ;

return 0 ;

}

// C program to demonstrate const specifier

#include <stdio.h>

int main()

{

const int num = 1;

num = 5; // Modifying the value

return 0;

}

Sẽ sai vì num đã thay đổi

Toán tử là một phần quan trọng trong ngôn ngữ lập trình C. Nó là một biểu tượng hay ký hiệu hoạt động trên một giá trị hoặc một biến.

Ví dụ : + là một toán tử để khai triển phép cộng

Int number = 200 + 300 ;

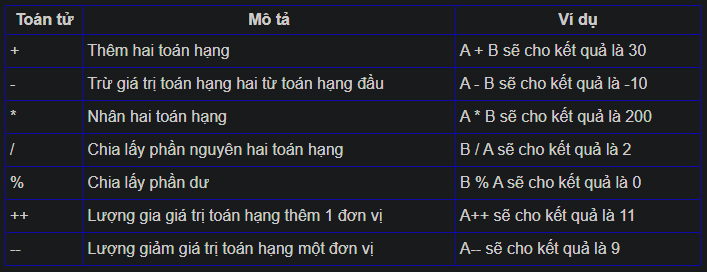
Mặc dù toán tử + thường được dùng để cộng hai giá trị với nhau, như ở ví dụ trên, nó cũng có thể được dùng để kết hợp một biến và một giá trị hoặc hai biến. Ví dụ:

int sum1 = 100 + 50; // 150 (100 + 50)

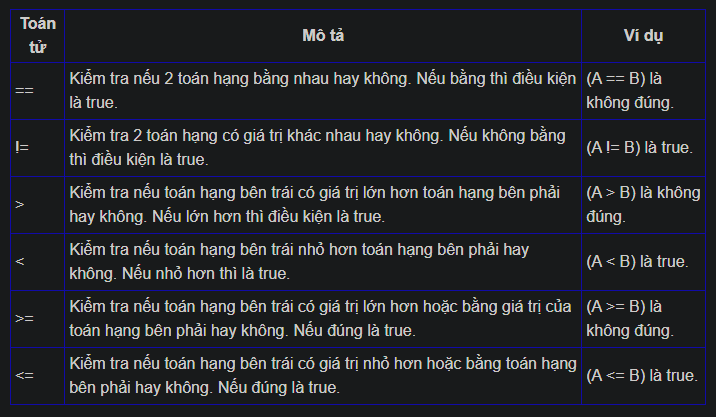
int sum2 = sum1 + 250; // 400 (150 + 250)

int sum3 = sum2 + sum2; // 800 (400 + 400)

Có rất nhiều phép toán tử trong C, để đơn giản và dễ hiểu hãy định dạng A = 10 B= 20



Toán tử quan hệ

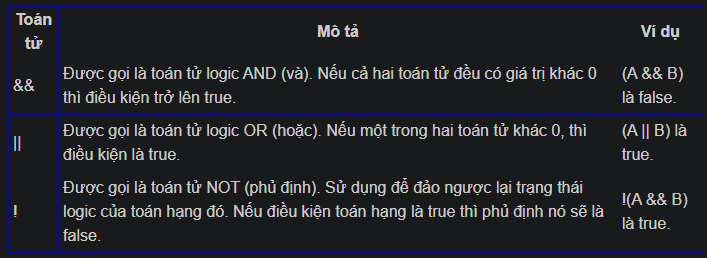


Toán tử logic

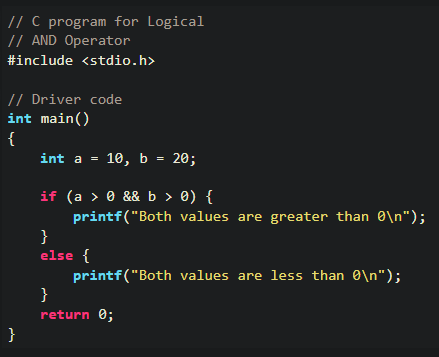
Chúng ta có 3 hàm toán tử Logic trong C ( Logical Operator )

* Logical AND &&
* Logical OR ||
* Logical NOT !

Với A = 1 B = 0



Ví dụ : Input



Output : Both values are greater than 0

Macro

A macro is a piece of code in a program that is replaced by the value of the macro. Macro is defined by #define directive. Whenever a macro name is encountered by the compiler, it replaces the name with the definition of the macro. Macro definitions need not be terminated by a semi-colon(; .

Ví dụ :

Input :

// C program to illustrate macros

#include <stdio.h>

// Macro definition

#define LIMIT 5

// Driver Code

int main()

{

// Print the value of macro defined

printf("The value of LIMIT"

" is %d",

LIMIT);

return 0;

}

Output :

The value of LIMIT is 5

Các kiểu Macro

1, Object-like Macros

2, 2. Chain Macros: Macros inside macros are termed as chain macros. In chain macros first of all parent macro is expanded then the child macro is expanded.

Ví dụ cho chain macro

#include <stdio.h>

#define INSTAGRAM FOLLOWERS

#define FOLLOWERS 138

int main()

{

// Print the message

printf("Geeks for Geeks have %dK"

" followers on Instagram",

INSTAGRAM);

return 0;

}

Output

Geeks for Geeks have 138K followers on Instagram

3, Multi-line Macros

// C program to illustrate macros

#include <stdio.h>

// Multi-line Macro definition

#define ELE 1, \

2, \

3

// Driver Code

int main()

{

// Array arr[] with elements

// defined in macros

int arr[] = { ELE };

// Print elements

printf("Elements of Array are:\n");

for (int i = 0; i < 3; i++) {

printf("%d ", arr[i]);

}

return 0;

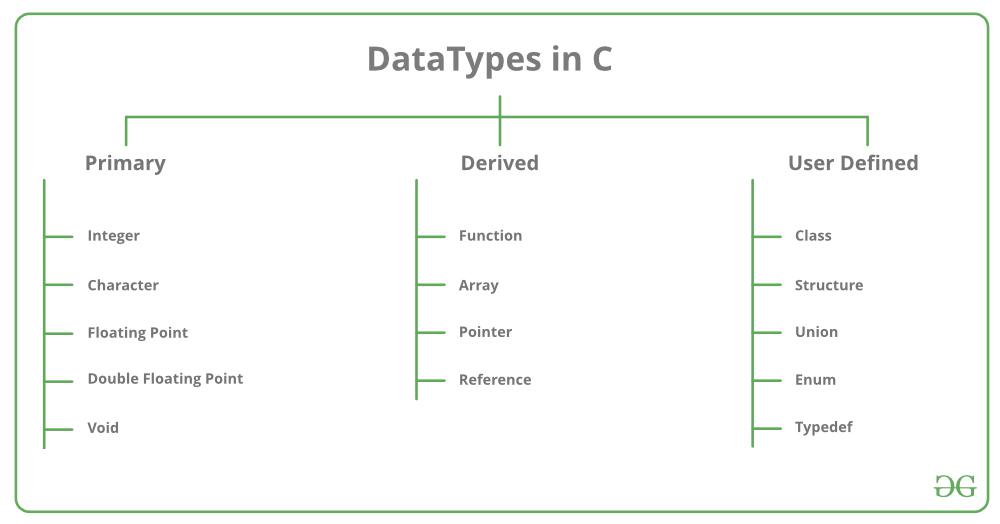
}

Output

Elements of Array are:

1 2 3

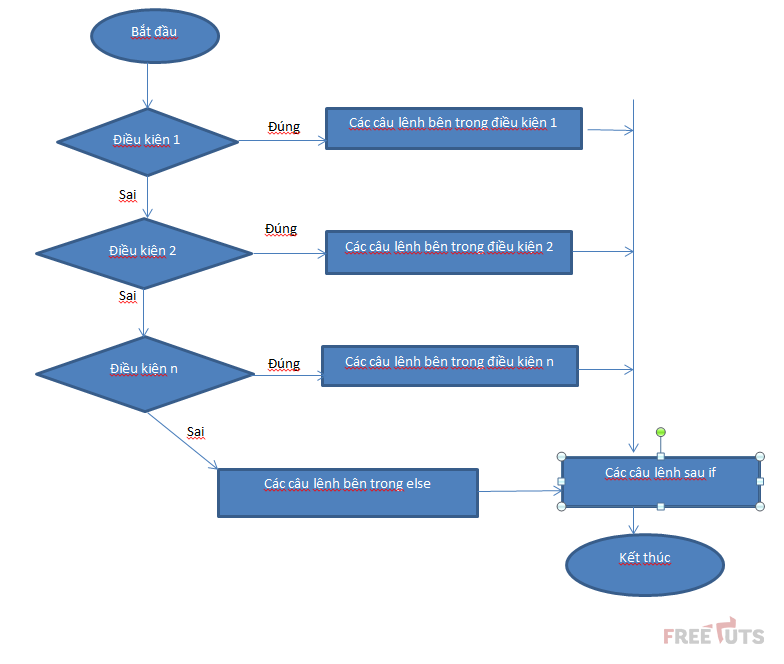
Các kiểu dữ liệu trong lập trình C



1. Các cấu trúc điều kiện, vòng lặp.

Câu điều kiện if-else:

  -  Được sử dụng để thực hiện một hành động nào đó nếu điều kiện đúng (true), và một hành động khác nếu điều kiện sai(false).



**Vòng lặp for:**

    Được sử dụng để lặp lại một khối lệnh một số lần nhất định. Cú pháp của vòng lặp for như sau:

**for** (initialization; condition; increment/decrement) {

// code to execute repeatedly

}

Trong đó:

– Initialization: là lệnh khởi tạo giá trị ban đầu cho biến sử dụng trong vòng lặp.

– Condition: là điều kiện để vòng lặp tiếp tục hoặc kết thúc.

– Increment/decrement: là lệnh tăng hoặc giảm giá trị của biến sử dụng trong vòng lặp.

**Vòng lặp while:**

    Cho phép lặp lại một khối mã cho đến khi một điều kiện đã cho là sai. Cú pháp của vòng lặp while như sau:

while (condition) {

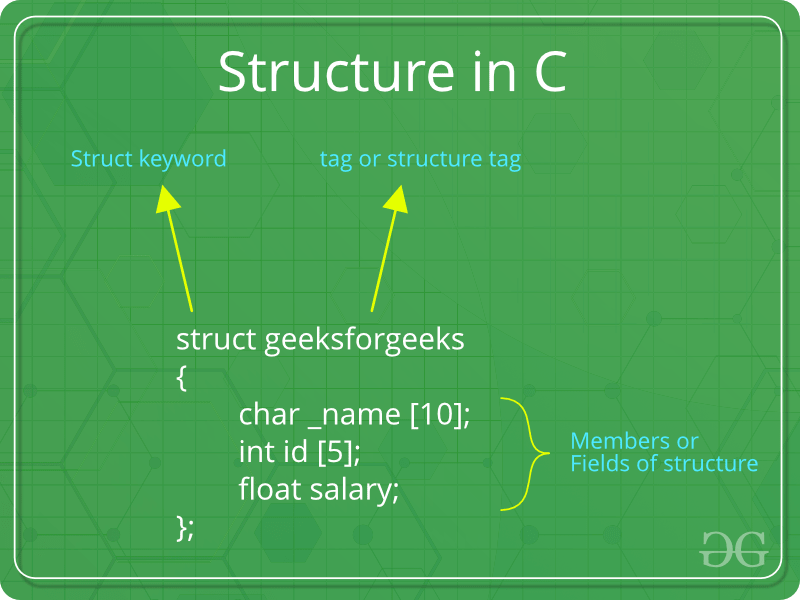
// block of code to be executed repeatedly

}

 Trong đó, condition là biểu thức điều kiện. Nếu giá trị của condition là true, khối mã bên trong vòng lặp while sẽ được lặp lại, ngược lại nếu condition là false thì vòng lặp sẽ kết thúc.

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

The structure in C is a user-defined data type that can be used to group items of possibly different types into a single type. The struct keyword is used to define the structure in the C programming language. The items in the structure are called its member and they can be of any valid data type.



1 ví dụ cho struct

#**include** <stdio.h>

#**include** <string.h>

struct Books

{

char tieude[50];

char tacgia[50];

char chude[100];

int id;

};

int main( )

{

struct Books Book1; /\* Khai bao Book1 la cua kieu Book \*/

struct Books Book2; /\* Khai bao Book2 la cua kieu Book \*/

/\* thong tin chi tiet quyen sach thu nhat \*/

strcpy( Book1.tieude, "Lap trinh C");

strcpy( Book1.tacgia, "Pham Van At");

strcpy( Book1.chude, "Ngon ngu lap trinh C");

Book1.id = 1234567;

/\* thong tin chi tiet quyen sach thu hai \*/

strcpy( Book2.tieude, "Toi thay hoa vang tren co xanh");

strcpy( Book2.tacgia, "Nguyen Nhat Anh");

strcpy( Book2.chude, "Van hoc");

Book2.id = 6677028;

/\* hien thi thong tin Book1 \*/

printf( "Tieu de cua Book1 la: %s\n", Book1.tieude);

printf( "Tac gia cua Book1 la: %s\n", Book1.tacgia);

printf( "Chu de cua Book1 la: %s\n", Book1.chude);

printf( "ID cua Book1 la: %d\n", Book1.id);

/\* hien thi thong tin Book2 \*/

printf( "Tieu de cua Book2 la: %s\n", Book2.tieude);

printf( "Tac gia cua Book2 la: %s\n", Book2.tacgia);

printf( "Chu de cua Book2 la: %s\n", Book2.chude);

printf( "ID cua Book2 la: %d\n", Book2.id);

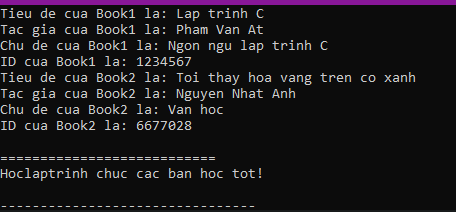
printf("\n===========================\n");

printf("Hoclaptrinh chuc cac ban hoc tot! \n");

return 0;

}

Output



Union trong C

Một Union là dữ liệu đặc biệt trong ngôn ngữ C cho phép bạn dự trữ các kiểu dữ liệu khác nhau trong cùng một vùng nhớ. Bạn có thể định nghĩa Union với rất nhiều tham số, nhưng chỉ một thành phần chứa giá trị tại một thời điểm. Union cung cấp một cách hiệu quả cho việc sử dụng một vùng nhớ cho nhiều mục đích.

union a

{

int i;

float f;

char chuoi[50];

} tenbien;

1. Con trỏ, mảng.

Pointers are one of the core components of the C programming language. A pointer can be used to store the memory address of other variables, functions, or even other pointers. The use of pointers allows low-level memory access, dynamic memory allocation, and many other functionality in C.

As the pointers in C store the memory addresses, their size is independent of the type of data they are pointing to. This size of pointers in C only depends on the system architecture.

Pointer Declaration

In pointer declaration, we only declare the pointer but do not initialize it. To declare a pointer, we use the ( \* ) dereference operator before its name.

Example

int \*ptr;

The pointer declared here will point to some random memory address as it is not initialized. Such pointers are called wild pointers.

Pointer Initialization

Pointer initialization is the process where we assign some initial value to the pointer variable. We generally use the ( & ) addressof operator to get the memory address of a variable and then store it in the pointer variable.

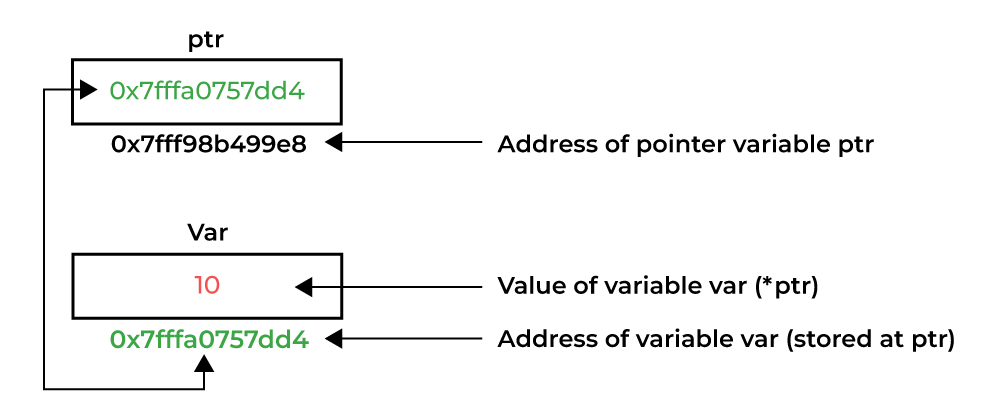
Example

int var = 10;

int \* ptr;

ptr = &var;

We can also declare and initialize the pointer in a single step. This method is called pointer definition as the pointer is declared and initialized at the same time.



1. Hàm, tham chiếu, tham trị

Truyền tham trị là truyền cho đối số một bản sao, nghĩa là giá trị của biến đó sẽ được sao chép sang biến tham số truyền vào hàm. Đây cũng là cách chúng ta sử dụng hàm bình thường.

**Nói đơn giản truyền tham trị là truyền giá trị.**

Ví dụ : Input

#include <stdio.h>

void incValue(int x)

{

x++;

printf("Bien trong ham = %d \n", x);

}

int main ()

{

int a = 10;

incValue(a);

printf("Bien ngoai ham ham = %d \n", a);

return 0;

}

Output :

Bien trong ham = 11

Bien ngoai ham ham = 10

Lỗi thường gặp khi dùng cách truyền tham trị là ta khai báo lại 1 lần nữa tham số truyền vào, ví dụ:

void changeValue(int x)

{

int x; // lỗi

x = 2;

}

**Truyền tham chiếu là gì?**

Truyền tham chiếu chính là cách chúng ta truyền cho nó một bản gốc thông qua địa chỉ ‘&‘. Nghĩa là giá trị của biến tham số truyền vào hàm, sẽ là giá trị của biến truyền vao hàm đó.

**Nói đơn giản truyền tham chiếu là truyền địa chỉ.**

Xét ví dụ:

#include <stdio.h>

void incValue(int &x)

{

x++;

printf("Bien trong ham = %d \n", x);

}

int main ()

{

int a = 10;

incValue(a);

printf("Bien ngoai ham ham = %d \n", a);

return 0;

}

Kết quả

Bien trong ham = 11

Bien ngoai ham ham = 11

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

C thông thường và Embedded C là hai biến thể của ngôn ngữ lập trình C, nhưng chúng có một số điểm khác biệt quan trọng. Dưới đây là sự giống và khác biệt giữa chúng:

Sự Giống Nhau:

Cú pháp: Cả hai đều sử dụng cú pháp C chuẩn, bao gồm các cấu trúc điều khiển, hàm, biến, v.v.

Kiểu dữ liệu và toán tử: Các kiểu dữ liệu và toán tử trong Embedded C cũng giống như trong C thông thường.

Tiêu chuẩn: Cả hai thường tuân thủ các tiêu chuẩn quốc tế như ANSI C hoặc ISO C.

Sự Khác Biệt:

Môi trường phát triển: C thông thường thường được sử dụng cho các ứng dụng máy tính thông thường, trong khi Embedded C được sử dụng chủ yếu cho việc phát triển phần mềm cho các thiết bị nhúng như vi điều khiển, cảm biến, bộ điều khiển và vi xử lý.

Hạn chế tài nguyên: Trong Embedded C, do tài nguyên hạn chế trên các thiết bị nhúng, nên các quy tắc về quản lý bộ nhớ, tối ưu hóa mã và quản lý tài nguyên phần cứng thường được thực hiện một cách cẩn thận hơn.

Thư viện chuẩn: Các thư viện chuẩn trong C thông thường thường không hoàn toàn phù hợp với việc lập trình nhúng. Trong Embedded C, các thư viện thường phải được tùy chỉnh hoặc thay thế bởi các thư viện nhỏ gọn và tối ưu hơn để phù hợp với yêu cầu của hệ thống nhúng.

Điều khiển phần cứng: Embedded C thường cần sử dụng các hàm và thủ tục đặc biệt để điều khiển phần cứng như đọc ghi dữ liệu từ các cổng GPIO, tương tác với các thiết bị giao tiếp như SPI, I2C, UART, v.v.

Tối ưu hóa và độ tin cậy: Embedded C thường được tối ưu hóa để sử dụng ít tài nguyên hơn và chạy ổn định trên các thiết bị nhúng có khả năng xử lý thấp.

Trong tóm tắt, Embedded C và C thông thường có nhiều điểm tương đồng nhưng cũng có những sự khác biệt quan trọng, đặc biệt là trong môi trường phát triển và quản lý tài nguyên phần cứng.

1. **Lưu ý khi lập trình C embedded**

Khi lập trình C cho các hệ thống nhúng (embedded), có một số điểm quan trọng mà bạn cần lưu ý để đảm bảo rằng mã của bạn hoạt động một cách chính xác và hiệu quả trên các thiết bị nhúng. Dưới đây là một số lưu ý quan trọng:

Hiểu rõ phần cứng: Cần phải hiểu rõ về phần cứng mà bạn đang làm việc trên đó. Điều này bao gồm kiến thức về vi xử lý, các giao tiếp ngoại vi (ví dụ: UART, SPI, I2C), bộ nhớ và các ràng buộc phần cứng khác.

Quản lý bộ nhớ: Trong các hệ thống nhúng, tài nguyên như bộ nhớ thường rất hạn chế. Bạn cần phải biết cách quản lý bộ nhớ một cách hiệu quả, tránh lãng phí tài nguyên và tránh tràn bộ nhớ.

Tiết kiệm năng lượng: Một số ứng dụng nhúng yêu cầu tiết kiệm năng lượng. Sử dụng các kỹ thuật như sleep modes và điều chỉnh tần số xung để giảm tiêu thụ năng lượng của hệ thống.

Xử lý lỗi: Trong một số trường hợp, việc xử lý lỗi trở nên phức tạp hơn trên các hệ thống nhúng. Việc sử dụng các cơ chế báo cáo lỗi và xử lý lỗi một cách cẩn thận là rất quan trọng.

Không sử dụng các thư viện tiêu tốn tài nguyên quá nhiều: Một số thư viện tiêu tốn quá nhiều tài nguyên như bộ nhớ hoặc tài nguyên xử lý. Thay vào đó, hãy sử dụng các thư viện nhỏ gọn và tối ưu hơn.

Thiết kế modul và tái sử dụng mã: Sử dụng các kỹ thuật thiết kế phần mềm tốt để tạo ra mã dễ bảo trì, mở rộng và tái sử dụng.

Kiểm tra và gỡ lỗi: Việc kiểm tra và gỡ lỗi trở nên khó khăn hơn trên các hệ thống nhúng. Sử dụng các công cụ gỡ lỗi như JTAG hoặc UART và đảm bảo rằng bạn có cơ chế ghi log hoặc báo cáo lỗi.

Chú ý đến thời gian thực: Trong một số ứng dụng, việc đảm bảo rằng mã của bạn hoạt động trong thời gian thực là rất quan trọng. Cân nhắc về việc sử dụng các ngắt (interrupts) và lập lịch (scheduling) để đảm bảo tính chính xác về thời gian.

Tuân thủ các chuẩn và quy định: Tuân thủ các chuẩn và quy định như MISRA C là rất quan trọng để đảm bảo tính ổn định và an toàn của hệ thống.

Bảo mật: Bảo mật cũng là một vấn đề quan trọng trong lập trình nhúng, đặc biệt là khi làm việc với các hệ thống kết nối mạng. Đảm bảo rằng bạn cân nhắc về việc bảo mật trong quá trình phát triển.

Nhớ rằng, việc lập trình cho các hệ thống nhúng đòi hỏi kiến thức chuyên sâu và sự cẩn thận. Hãy luôn thử nghiệm mã của bạn trên nhiều môi trường và điều kiện khác nhau để đảm bảo tính ổn định và đáng tin cậy của nó.

1. **C memory management**

A typical memory representation of a C program consists of the following sections.

Text segment (i.e. instructions)

Initialized data segment

Uninitialized data segment (bss)

Heap

Stack



1. Compilation process, toolchain, compiler, linker.
2. Make, build system.
3. 8bit/16bit/32bit machine
4. Tổ chức firmware, kiến trúc phần mềm.
5. Embedded C for 8051.

Tham khảo khóa học.

<https://www.youtube.com/watch?v=Fu-0Yc4ZS-w&list=PLhFjtzzUovr-YW6vlzkiUJRo88T4deV23&ab_channel=H%E1%BB%8CCIT-%C4%90I%E1%BB%86NT%E1%BB%AC>

1. Layout PCB.
2. Tìm hiểu cách vẽ mạch in bằng Altium trên youtube
3. Thực hiện vẽ Kit 8051 với các ngọai vi như: GPIO, LED, BUTTON, UART, khối nạp