
IMIC
TECHNOLOGY
Khởi nguồn thành công

General Introduction of C

Mai Trong Nhan (Mr.)
Lecturer
+84 (0) 385.852.284
mainhan.imic@gmail.com

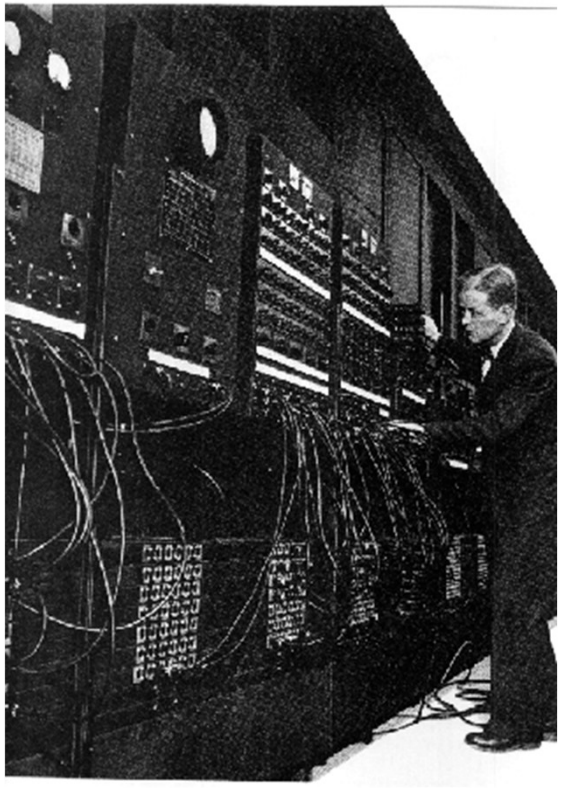
Overview of programming languages

Overview of Programming Languages

What are Computer Programming Languages?

Computer programming languages allow us to give instructions to a computer in a language the computer understands. Just as many human-based languages exist, there are an array of computer programming languages that programmers can use to communicate with a computer. The portion of the language that a computer can understand is called a “binary.” Translating programming language into binary is known as “compiling.” Each language, from C Language to Python, has its own distinct features, though many times there are commonalities between programming languages

Developing History of Computer Languages.



1883: Algorithm for the Analytical Engine: Created by Ada Lovelace for Charles Babbage's Analytical Engine to computer Bernoulli numbers, it's considered to be the first computer programming language.

1949: Assembly Language: First widely used in the Electronic Delay Storage Automatic Calculator, assembly language is a type of low-level computer programming language that simplifies the language of machine code, the specific instructions needed to tell the computer what to do.

1972: C: Developed by Dennis Ritchie at Bell Labs, C is considered by many to be the first high-level language. A high-level computer programming language is closer to human language and more removed from the machine code. C was created so that an operating system called Unix could be used on many different types of computers. It has influenced many other languages, including Ruby, C#, Go, Java, JavaScript, Perl, PHP, and Python.

1983: C++: C++ is an extension of the C language and was developed by Bjarne Stroustrup. It is one of the most widely used languages in the world. C++ is used in game engines and high-performance software like Adobe Photoshop. Most packaged software is still written in C++

Cấu trúc chương trình C

Chương trình C: Hello World

Chương trình C đầu tiên: Hello World

Một chương trình C bao gồm những phần sau đây:

+ Các lệnh tiền xử lý
+ Các hàm
+ Các biến
+ Các lệnh và biểu thức
+ Các comment

```c
#include <stdio.h>
int main()
{
/* Day la chuong trinh C dau tien */
                printf("Hello, World! \n");
                return 0;
 }
```

#include <stdio.h> là lệnh tiền xử lý, nhắc nhở bộ biên dịch C thêm tệp stdio.h trước khi biên dịch.

int main() là hàm main, nơi chương trình bắt đầu.

/*...*/ là dòng comment được bỏ qua bởi bộ biên dịch compiler và được dùng để thêm các chú thích cho chương trình. Đây được gọi là phần comment của chương trình.

printf(...) là một hàm chức năng khác của ngôn ngữ C , in ra thông điệp "Hello, World!" hiển thị trên màn hình.

return 0; kết thúc hàm chính và trả về giá trị 0.

Một số quy tắc cần nhớ khi viết chương trình

- **Quy tắc 1:** Mỗi câu lệnh có thể viết trên một hay nhiều dòng nhưng phải được kết thúc bằng dấu ; (dấu chấm phẩy)

  Một câu lệnh có thể viết trên nhiều dòng, ta đặt thêm dấu \ trước khi xuống dòng.

  VD:

```
printf(“ C THUC HANH \
        LAM QUEN VOI BAN”);
```

- **Quy tắc 2:** Là quy tắc viết các lời ghi chú. Các lời ghi chú cần được đặt giữa cặp dấu /* … */ hoặc dấu //

  Những lời chú thích sẽ được compiler làm “làm ngơ” khi biên dịch chương trình.

- **Quy tắc 3:** Trong một chương trình có thể có nhiều hàm nhưng chỉ duy nhất 1 hàm main

Các khái niệm cơ bản

Tập ký tự dùng trong ngôn ngữ lập trình C

Tập ký tự dùng trong ngôn ngữ lập trình C.

- Cũng như ngôn ngữ tự nhiên, mọi ngôn ngữ lập trình được xây dựng từ một bộ ký tự được coi là bản chữ cái của ngôn ngữ đó. Các ký tự này được kết hợp với nhau theo quy tắc để tạo thành các câu lệnh (statement). Một chương trình sẽ gồm nhiều câu lệnh được diễn đạt theo logic của một thuật tóa nào đó để giải quyết một bài toán cụ thể.

  ngôn ngữ C được xây dựng trên bộ ký tự sau:

  * 26 chữ cái tiếng Anh viết hoa: A, B, C,. . . ,Z

  * 26 chữ cái tiếng Anh viết thường: a, b, c,. . . ,z

  * 10 chữ số của hệ thập phân: 0, 1, 2, . . ., 9

  * Các ký hiệu toán học như: + - * / = ( )

  * Ký tự gạch dưới: _

  * Các ký hiệu đặt biệt khác như: . , ; : [ ] { } ?

➔ Chú ý khi viết chương trình ta không được sử dụng bất kỳ ký hiệu nào khác ngoài tập hợp ký tự nói trên.

.

Tập ký tự dùng trong ngôn ngữ lập trình C.

- Từ khóa là những từ dành riên của ngôn ngữ lập trình được định nghĩa trước với ý nghĩa hoàn toàn xác định. Từ khóa thường được dùng để khai báo biến định nghĩa các kiểu dữ liệu, định nghĩa ra các toán tử, các hàm và các câu lệnh....
- Một số từ khóa dùng cho C:

asm	break	case	cdecl	char	const	continue	default
do	double	else	enum	extern	far	float	for
goto	huge	if	int	interrupt	long	near	pascal
register	return	short	signed	sizeof	static	struct	switch
typedef	union	unsigned	void	volatile	while		

- ➔ Không được dùng từ khóa để đặt tên cho các hằng, biến, mảng,...
- *** Từ khóa phải được viết bằng chữ thường. VD: không được viết INT hay Int mà phải viết int.

Tên (identifier)

- Tên được dùng để xác định các đối tượng khác nhau trong một chương trình. Chúng ta có:
  - Tên hằng
  - Tên biến.
  - Tên mảng.
  - Tên hàm.
  - Tên con trỏ.
  - Tên cấu trúc
  - Tên nhãn. . .

Tên là một khái niệm rất quan trọng. Tên được đặt theo quy tắc sau:

- Tên là một dãy các ký tự: chữ, số và ký tự "_" (dấu gạch dưới)

- Ký tự đầu của tên phải là chữ hoặc ký tự _ (dấu gạch dưới)

- Không được trùng với từ khóa.

*** **Chú ý**: *Trong tên, các chữ hoa và chữ thường được xem là khác nhau. VD: tên biến AB khác với aB. Trong C thường dùng chữ hoa để đặt cho các hằng và dùng chữ thường để đặt tên cho hầu hết các đối tượng khác như biến, mảng, hàm, cấu trúc. Tuy nhiên đây không phải là điều bắt buộc*

Các khái niệm cơ bản

Khai báo và toán tử gán

Khai báo và toán tử gán

- Toán tử gán có dạng:

```
b = bt;
```

- Trong đó b là một biến hoặc một biểu thức toán học nào đó. Trước tiên chương trình sẽ thực hiện biểu thức bt sau đó lấy giá trị biểu thức bt gán cho biến b.

VD:
```
int a;
a = 10;
a = 2*a  -  2;
```

+ Đầu tiên sẽ gán giá trị 10 cho biến a.

+ Sau đó sẽ thực hiện biểu thức 2*a – 2 (ta được kết quả là 18)

+ Sau đó gán giá trị 18 cho biến a.

➔ Ta cần chú ý và phân biệt toán tử gán với khái niệm đẳng thức trong toán học.

Toán tử gán là một lệnh trong chương trình. Nó ra lệnh cho máy tính lấy giá trị của biểu thức bỏ vào địa chỉ của biến	Đẳng thức trong toán học biểu thị mối quan hệ giữa 2 vế của đẳng thức

KHAI BÁO BIẾN

Mai Trong Nhan (Mr.)
Lecturer
nhan.mai@imic.edu.vn

1. CÁCH THỂ HIỆN SỐ TRONG NGÔN NGỮ LẬP TRÌNH C

Trong ngôn ngữ lập trình nói chung, ngôn ngữ C/C++ nói riêng. Chúng ta có 4 cách thể hiện 1 con số. Theo dạng hệ 2, hệ 8, hệ 10, hệ 16 (hoặc BIN, OCT, DEC, HEX)

Hệ 2 (bin): một số nguyên được biểu diễn bằng 2 chữ số là 0 và 1

Hệ 8 (Oct): một số nguyên được biểu diễn bằng 8 chữ số là 0, 1, 2, 3, 4, 5, 6, 7

Hệ 10 (Dec): đây là cách thể hiện phổ biến nhất mà chúng ta hay dùng - một số nguyên được biểu diễn bằng 10 chữ số là 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

Hệ 16 (Hex): một số nguyên được biểu diễn bằng 16 chữ số là 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B C, D, E, F

Ví dụ: Các cách biểu diễn 1 số có cùng giá trị:
Hệ 2: 0b1010
Hệ 10: 10
Hệ 16: 0x0A

2. Đơn vị đo lường thông tin - Cấu trúc của một vùng nhớ

Bit: là đơn vị nhỏ nhất của bộ nhớ máy tính. 2 ký hiệu được sử dụng trong máy tín là 0 và 1

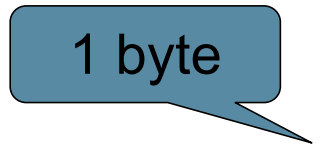
Byte: 1 Byte = 8 Bit

Kilobyte (Kb): 1 Kb = 1024 byte

MegaByte (Mb): 1 Mb = 1024 Kb

GigaByte (Gb): 1 Gb = 1024 Mb

TeraByte (Tb): 1 Tb = 1024 Gb



0x01	0	0	0	0	1	0	1	0
0x02	0	0	0	0	0	0	0	0
0x03	0	0	0	0	0	0	0	0
0x04	0	0	0	0	0	0	0	0
0x05	0	0	0	0	0	0	0	0
0x06	0	0	0	0	0	0	0	0
0x07	0	0	0	0	0	0	0	0
0x08	0	0	0	0	0	0	0	0
0x09	0	0	0	0	0	0	0	0
0x0A	0	0	0	0	0	0	0	0
0x0B	0	0	0	0	0	0	0	0
0x0C	0	0	0	0	0	0	0	0
0x0D	0	0	0	0	0	0	0	0
0x0E	0	0	0	0	0	0	0	0
0x0F	0	0	0	0	0	0	0	0
0x10	0	0	0	0	0	0	0	0
0x11	0	0	0	0	0	0	0	0
0x12	0	0	0	0	0	0	0	0
0x13	0	0	0	0	0	0	0	0
0x14	0	0	0	0	0	0	0	0

3. Kiểu dữ liệu

Kiểu dữ liệu	Kích thước	Phạm vi biểu diễn
unsigned char	1 byte	0 –> 255
char	1 byte	(-128) –> 127
unsigned int	4 byte	0 -> 4294967295
int	4 byte	-2147483646 -> 2147483647
float	4 byte	3.4E-38 -> 3.4E+38
double	8 byte	1.7E-308 -> 1.7E+308
long double	10 byte	3.4E-4932 -> 1.1E4932

Cú pháp khai báo biến.

Kiểu_dữ_liệu tên_biến [= giá_trị_khởi_tạo];

Toán tử

Toán tử

I. Toán tử số học

Phép toán	Ý nghĩa	Ví dụ
+	Cộng	A + B
-	Trừ	A – B
*	Nhân	A * B
/	Chia	A / B
%	Chia lấy dư	A % B

II. Toán tử thao tác bit

Phép toán	Ý nghĩa	Ví dụ
&	AND	A&B
|	OR	A|B
^	XOR	A^B
<<	Dịch trái	A<<4
>>	Dịch phải	A>>4

- *** Chú ý: Đối với 2 số nguyên, Phép chia sẽ chặt cụt phần phân. VD: 11/3 = 3

1&1=1 1|1=1 1^1=0
1&0=0 1|0=1 1^0=1
0&1=0 0|1=1 0^1=1
0&0=0 0|0=0 0^0=0

Toán tử

III. Toán tử so sánh

Phép toán	Ý nghĩa	Ví dụ
>	Có lớn hơn không?	A > B 3>7 trả về giá trị 0 (false)
>=	Có lớn hơn hay bằng không?	A>=B 8>=8 có giá trị 1(true)
<	Có nhỏ hơn không?	A<B 9<9 có giá trị là 0(false)
<=	Có nhỏ hơn hay bằng không?	A<=B 3<=10 có giá trị 1(true)
==	Có bằng nhau không	A==B 3==9 có giá trị 0(false)
!=	Có khác nhau không?	3!=9 có giá trị 1(true)

Phép toán logic

- Trong C sử dụng ba phép toán logic:
  - Phép PHỦ ĐỊNH MỘT NGÔI !
  - Phép VÀ &&
  - Phép HOẶC ||

A	!A
Khác 0	0 (false)
Bằng 0	1 (true)

A	B	A&&B	A||B
Khác 0	Khác 0	1(true)	1(true)
Khác 0	Bằng 0	0(false)	1(true)
Bằng 0	Khác 0	0(false)	1(true)
Bằng 0	Bằng 0	0(false)	0 (false)

*** Hãy cho biết sự khác nhau giữa toán tử logic và toán tử thao tác bit

Thứ tự ưu tiên toán tử

loại	Toán tử	Thứ tự ưu tiên
Postfix	() [] -> . ++ - -	Trái sang phải
Unary	+ - ! ~ ++ - - (type)* & sizeof	Phải sang trái
Tính nhân	* / %	Trái sang phải
Tính cộng	+ -	Trái sang phải
Dịch chuyển	<< >>	Trái sang phải
Quan hệ	< <= > >=	Trái sang phải
Cân bằng	== !=	Trái sang phải
Phép AND bit	&	Trái sang phải
Phép XOR bit	^	Trái sang phải
Phép OR bit	|	Trái sang phải
Phép AND logic	&&	Trái sang phải
Phép OR logic	||	Trái sang phải
Điều kiện	?:	Phải sang trái
Gán	= += -= *= /= %=>>= <<= &= ^= |=	Phải sang trái

Một số biểu thức, lệnh quan trọng trong C

Chuyển đổi kiểu giá trị

- **Chuyển đổi kiểu trong biểu**

  - Quy tắc: Khi chuyển hai toán hạng trong một phép toán có kiểu dữ liệu khác nhau thì kiểu thấp hơn sẽ được nâng giá trị có kiểu cao hơn.

  Ví dụ:

  *1.5 * (11 / 3) = ?*

  *1.5 * 11 / 3 = ?*

  *(11.0 / 3) * 1.5 = ?*

- **Các phép chuyển đổi kiểu**

  - Các phép chuyển đổi kiểu cũng được thực hiện thông qua phép gán
  - Giá trị của vế phải được chuyển sang kiểu của vế trái đó là kiểu của kết quả.

- **Ép kiểu:**

  (type) (biểu thức)

Phép toán tăng giảm

C đưa ra hai phép toán một ngôi để tăng và giảm các biến (nguyên và số thực). Toán tử ++ sẽ cộng 1 vào toán hạng của nó, toán tử giảm – sẽ trừ đi 1. Chẳng hạn nếu n đang có giá trị bằng 5:

- Sau phép tính ++n, n có giá trị 6
- Sau phép tính --n, n có giá trị 4.

Dấu phép toán ++ và – có thể đứng trước hoặc sau toán hạng, như vậy có thể viết:

++n n++ --n n—

Sự khác nhau của ++n và n++ ở chổ:

- trong phép n++ thì n được tăng sau khi giá trị nó được sử dụng, còn trường hợp phép ++n thì n được tăng trước khi nó được sử dụng. Tương tụ cho –n và n--

Biểu thức điều kiện:

Biểu thức điều kiện là biểu thức có dạng:

e1?e2:e3

Trong đó e1, e2, e3 là các biểu thức nào đó. Giá trị của biểu thức điều kiện bằng giá trị của e2 nếu e1 khác 0 (đúng) và bằng giá trị e3 nếu e1 bằng không (false)

Bài tập:

Viết chương trình tách 0xABCD thành 2 biến 8 bit 0xAB và 0xCD

Xây dựng hàm và sử dụng hàm

Xây dựng hàm và cách hoạt động của hàm.

```
Kiểu ten_ham(khai báo đối)
{
        Khai báo các biến cục bộ;
        các câu lệnh;
        [return[biểu thức];]
}
```

```
void hoan_vi(uint8_t x, uint8_t y)
{
        uint8_t temp;
        temp = x;
        x = y;
        y = temp;
}
```

- Đối số và biến cục bộ đều có phạm vi trong cùng một hàm nên đối và biến cục bộ cần khai báo có tên khác nhau.
- Đối và biến cục bộ đều là các biến tự động, nghĩa là chúng được cấp bộ nhớ khi ham được xét đến và chũng sẽ bi xóa đi trước khi ra khỏi hàm.

Chú ý rằng: *có thể đặt tên biến cục bộ và đối số trùng với các biến bên ngoài hàm.*

tên của hàm chính là địa chỉ chứa các dòng lệnh thực hiện hàm.

```
void main()
{
        uint8_t x = 10;
        uint8_t y = 20;
        hoan_vi(x, y);
        printf("x: %d, y: %d", x, y);
}
```

Con trỏ

Con trỏ

- Con trỏ là một biến dùng để chứa địa chỉ.
- Có nhiều loại địa chỉ nên cũng có nhiều kiểu con trỏ tương ứng.

```
Kiểu * tên_con_trỏ;
```

Quy tắc sử dụng con trỏ trong các biểu thức

Ta có thể sử dụng tên con trỏ hoặc dạng khai báo của nó trong các biểu thức :

VD: khai báo một con trỏ

```
int *px;
```

+ Cách 1: Sử dụng tên con trỏ (px). Con trỏ cũng là một biến nên khi tên của nó xuất hiện trong một biểu thức thì giá trị của nó sẽ được sử dụng trong biểu thức này.

lưu ý:

Giá trị của con trỏ là một địa chỉ của một biến nào đó.

Khi tên con trỏ đứng ở bên trái của một toán tử gán thì giá trị biểu thức bên phải phải là một địa chỉ

+ Cách 2: Sử dụng dạng khai báo của con trỏ (*px). Nó hoạt động như một biến thông thường.

Hàm có đối số là con trỏ

```
#include “stdio.h”
void hoanVi(uint8_t *px, uint_8 *py)
{
        uint8_t temp;
        temp = *px;
        *px = *py;
        *py = temp;
}
void main()
{
        uint8_t x, y;
        x = 1;
        y = 2;
        hoanVi(&x, &y);
        printf(“x: %d, y: %d”, x, y);
}
```

- Nếu đối của hàm là một con trỏ thì tham số thực tương ứng phải là địa chỉ của biến muốn truyền vào.
- Khi đó, địa chỉ của biến được truyền cho đối của con trỏ tương ứng.

Chú ý: Trong các đối số của hàm, ta có 2 loại:

Loại 1: Là các đối số dùng để chứa dữ liệu vào (input)

Loại 2: Là các đối số dùng để xuất dữ liệu ra (output)

➔ *Thực tập: sử dụng hàm giải phương trình bậc 2. Thay đổi hàm giaiPhuongTrinh*
uint8_t giaiPhuongTrinh(uint8_t *a, uint8_t *b, uint8_t *c, uint8_t *x1, uint8_t *x2)

Con trỏ hàm

Cách khai báo con trỏ hàm.

```
kiểu (*ten_con_tro_ham)(kiểu)
```

Tác dụng của con trỏ hàm.

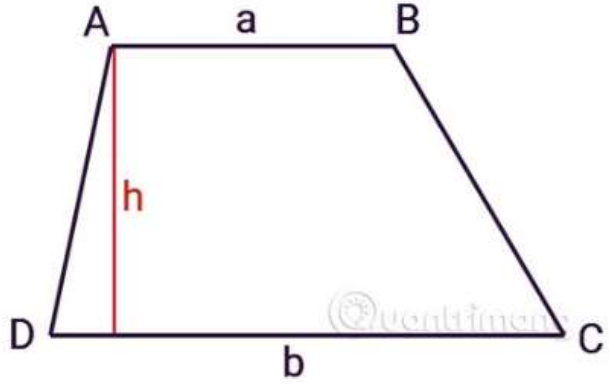
Con trỏ hàm dùng để chứa địa chỉ của hàm. Muốn vậy ta thực hiện phép gán tên hàm cho con trỏ hàm. Để phép gán có nghĩa thì kiểu hàm và kiểu con trỏ phải tương thích. Sau phép gán ta có thể sử dụng tên con trỏ hàm thay cho tên hàm.

Đối con trỏ hàm

C cho phép thiết kế các hàm mà đối số lại là một tên của hàm khác ➔ khi đó đối số phải là con trỏ hàm

Thực hành: Lặp trình hàm tính tích phân của hàm f(x) trên đoạn [a,b] theo phương pháp hình thang với tần số lấy mẫu 1000 lần.

Có hình thang ABCD với độ dài đáy AB là a, đáy CD là b và chiều cao h.



Công thức chung tính diện tích hình thang: **trung bình cộng 2 cạnh đáy nhân với chiều cao giữa 2 đáy.**

$$S_{ABCD} = \frac{a+b}{2} \times h$$

Mảng (array)

Mảng

- Mỗi biến chỉ có thể chứa một giá trị. Để chứa một dãy ký tự ta có thể sử dụng nhiều biến, nhưng đây không phải là cách tiện lợi. Mảng sinh ra để giải quyết bài toán trên.

Mảng có thể hiểu là một tập hợp nhiều phần tử có cùng một kiểu dữ liệu và có chung một tên

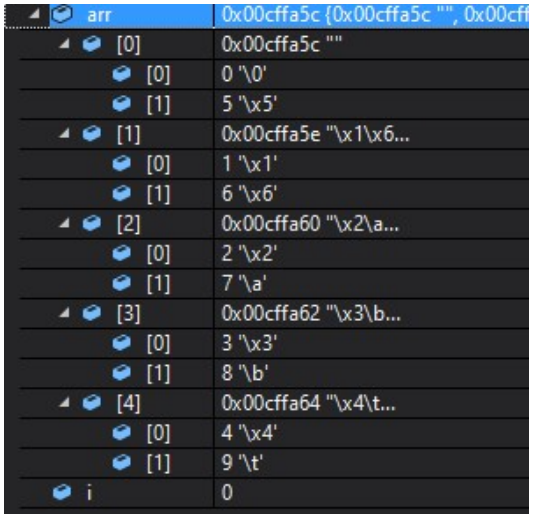
Mỗi phần tử mảng có vai trong như một biến và chứa được một giá trị.

Cách khai báo mảng:

```
uint8_t mang[2];
uint8_t mang[];
uint8_t mang[2][3]
```

- Các phần tử của mảng được cấp phát các khoảng nhớ liên tiếp nhau trong bộ nhớ. Nói cách khác các phần tử mảng có địa chỉ liên tiếp nhau.
- Các phần tử của mảng hai chiều được sắp xếp theo hàng.

```
uint8_t arr[5][2];
for (int i = 0; i < 2; i++)
{
    for (int j = 0; j < 5; j++)
    {
        arr[j][i] = j + 5 * i;
    }
}
```



Mối liên hệ giữa con trỏ và mảng

- Trong C có mối quan hệ chặt chẻ giữa con trỏ và mảng: các phần tử của mảng có thể được xác định nhờ chỉ số hoặc thông qua con trỏ.

```
uint8_t arr[20];
```

Lúc đó hệ điều hành sẽ bố trí cho mảng arr 20 khoảng nhớ (1 khoảng 1 byte – 8bit) liên tiếp nhau. Như vậy nếu biết địa chỉ của một phần tử nào đó của mảng a thì sẽ dễ dang suy ra địa chỉ của các phâng tử khác.

Điều đặt biệt: tên mảng là một hằng địa chỉ. Nó chính là địa chỉ của phần tử *đầu tiên* của mảng

Như vậy:

```
arr tương đương với &arr[0]
arr+i tương đương với &arr[i]
*(arr+i) tương đương với arr[i]
```

Mảng, con trỏ và xâu ký tự

Ta có, xâu ký tự là một dãy ký tự đặt trong hai dấu nháy kép. Ví dụ:

```
"hello world"
```

Khi gặp một xâu kí tự, máy sẽ cấp phát một khoảng nhớ cho mảng kiểu *char* đủ lớn để chứa các ký tự có trong xâu và chứa thêm kí tự '\0' (ký tự này được dùng để báo kết thúc một xâu ký tự). Mỗi ký tự của xâu được chứa trong một phần tử của mảng.

Chú ý: cũng giống như tên mảng, tên của xâu ký tự là một hằng địa chỉ biểu diễn địa chỉ đầu của mảng chứa nó. Vì vậy nếu ta khai báo *st* kiểu con trỏ *char.* Thì khi đó ta có thể có phép gán:

```
char *st;
st = "hello world";
```

Cấp phát bộ nhớ.

Cấp phát bộ nhớ

Các hàm được khai báo trong thư viện *alloc.h*

1. **calloc** cấp phát vùng nhớ cho n đối tượng kích thước size byte

```
void *calloc(unsigned n, unsigned size)
```

Công dụng : cấp phát vùng nhớ n*size byte. Nếu thành công hàm sẽ trả về địa chỉ đầu vùng nhớ được cấp, khi không đủ bộ nhớ cấp phát hàm sẻ trả vê NULL

2. **malloc** cấp phát vùng nhớ cho n byte.

```
void *malloc(unsigned n)
```

Công dụng : cấp phát vùng nhớ n byte. Nếu thành công hàm sẽ trả về địa chỉ đầu vùng nhớ được cấp, khi không đủ bộ nhớ cấp phát hàm sẻ trả vê NULL

3. **free** giải phóng vùng nhớ đã cấp phát.

```
void free(void *ptr)
```

Công dụng :Giải phóng vùng nhớ (đã được cấp phát bởi calloc hoặc malloc) do con trỏ ptr trỏ tới

Kiểu cấu trúc

Cấu trúc - Struct

- Trước khi xây dựng một hoặc một số cấu trúc có cùng một kiểu ta cần phải mô tả kiểu có nó. Điều này cũng tương tự như việc phải thiết kế một kiểu nhà trước khi xây dựng những cân nhà thực sự ở những điểm khác nhau. Khi định nghĩa một kiểu cáu trúc cần chỉ ra: tên của kiểu cấu trúc và các thành phần của nó. Việc này được thực hiện theo mẫu sau:

```
struct tên_cấu_trúc
{
        //khai báo các thành phần của nó
};
```

Trong đó: thành phần khai báo có thể là biến, mảng, nhóm bit, hoặc một cấu trúc khác mà kiểu có nó đã được định nghĩa từ trước.

```
struct ngay
{
        int ngay_thu;
        char ten_thang[10];
        int nam;
};
```

```
struct nhan_cong
{
        char ten[15];
        char dia_chi[20];
        double bac_luong;
        struct ngay ngay_sinh;
        struct ngay _vao_cty
};
```

Khởi gán cho cấu trúc

```
struct ngay
{
        int ngay_thu;
        int thang;
        int nam;
        char *ten_thang;
};

struct ngay A = {4, 7, 1990, “thang bay”};
```

Kiểu cấu trúc

- Ngôn ngữ lập trình C cung cấp một từ khóa **typedef** để sử dụng để cung cấp kiểu cho một tên mới.

```
typedef unsigned char BYTE;
```

- Sau khi định nghĩa kiểu này, định danh BYTE có thể được sử dụng như tên viết tắt của kiểu dữ liệu **unsigned char**

- Chúng ta cũng có thể sử dụng typedef để định nghĩa một kiểu cấu trúc

```
typedef struct
{
        //khai báo các thành phần của nó
} tên_kiểu_cấu_trúc;
```

```
typedef struct
{
        int ngay_thu;
        char ten_thang[10];
        int nam;
} ngay;
```

```
stypedef truct
{
        char ten[15];
        char dia_chi[20];
        double bac_luong;
        ngay ngay ngay_sinh;
        ngay ngay_vao_cty
} nhan_cong;
```

Truy cập đến các phần tử của cấu trúc

- Để truy nhập tới một thành phần cơ bản của một cấu trúc ta sử dụng một trong các cách viết sau:

```
struct tên_cấu_trúc
{
        tên_thành_phần_1;
        struct tên_cấu_trúc
        {
                tên_thành_phần;
        }
};
```

```
tên_cấu_trúc.tên_thành_phần_1
tên_cấu_trúc.tên_cấu_trúc.tên_thành_phần
```

Con trỏ cấu trúc

```
struct ngay
{
        int ngay_thu;
        char ten_thang[10];
        int nam;
};

struct nhan_cong
{
        char ten[15];
        char dia_chi[20];
        double bac_luong;
        struct ngay ngay_sinh;
        struct ngay ngay_vao_co_quan;
};
```

- Khi đó con trỏ cấu trúc kiểu nhan_cong có thể được khai báo cùng với biến và mảng như sau:

```
struct nhan_cong x, *px;
```

đối với struct x cách truy nhập chúng ta đã thảo luật ở phần trên.

Nhưng đối với struct *px. Nó được khai báo theo kiểu con trỏ. thì cách dùng có chút khác biệt:

```
(*px).bac_luong;

px->bac_luong;
```

Cách 1:

```
Tên_con_trỏ->tên_thành_phần
```

Cách 2:

```
(*Tên_con_trỏ).tên_thành_phần
```

Hàm trên các cấu trúc

Đối số của hàm có thể là:

+ Biến cấu trúc. Khi đó tham số thực tương ứng là một giá trị của cấu trúc.

+ Con trỏ cấu trúc: Khi đó tham số thực tương ứng là địa chỉ của biến cấu trúc.

Hàm có thể trả về giá trị:

+ Giá trị cấu trúc.

+ Con trỏ cấu trúc.

*** Xét ví dụ: tìm nghiệm của phương trình bậc 2.
sử dụng return về một kiểu cấu trúc chứa 2 biến x1, x2

Kiểu cấu trúc

Kiểu hợp (union)

Hợp (uinon) là gì: cũng giống như cấu trúc, hợp gồm nhiều thành phần nhưng chúng khác nhau ở chỗ: Các thành phần của cấu trúc có những vùng nhớ khác nhau, còn các thành phần của hợp được cấp phát một vùng nhớ chung. Độ dài của hợp bằng độ dài của thành phần có bộ nhớ lớn nhất.

Khai báo: Việc khai báo, định nghĩa kiểu hợp, cũng như cách truy nhập đến các thành phần hợp được thực hiện hoàn toàn tương tự như đối với cấu trúc. một cấu trúc có thể có chứa thành phần kiểu hợp, và một kiểu hợp cũng có thể chưa thành phần cấu trúc

*** Xét ví dụ: sử dụng kiểu hợp để tách 1 biến 16bit thành 2 biến 8 bit

Thao tác trên các tệp tin.

Mai Trong Nhan (Mr.)
Lecturer
+84 (0) 385.852.284
nhan.mai@imic.edu.com

Kiểu nhập xuất nhị phân và văn bản

Trước khi giới thiệu các kiểu xuất nhập nhị phân và văn bản chúng ta cũng cần biết cấu trúc chung của một tệp tin (file) trên đĩa. Một tệp tin (dù nó được xây dựng bằng cách nào) đơn giản chỉ là một dãy các byte (có giá trị từ 0 đến 255) ghi trên đĩa. Số byte của dãy chính là độ dài của tệp.

Bảo toàn dữ liệu: Trong quá trình nhập xuất dữ liệu không bị biến đổi. Dữ liệu ghi trên tệp theo các byte nhị phân như trong bộ nhớ.

Mã kết thúc tệp: Trong khi đọc, nếu gặp cuối tệp thì ta nhận được ký tự kết thúc tệp EOF (định nghĩa trong stdio.h bằng -1) và hàm feof cho giá trị khác không. Tại sao lại chọn số -1 làm mã kết thúc têp?

-> Lý do rất đơn giản: nếu chưa gặp cuối tệp thì sẽ đọc được một byte có giá trị từ 0 – 255. Như vậy giá trị 1 sẽ không trùng với bất kỳ ký byte nào đọc được từ file

Kiểu văn bản: Kiểu nhập xuất văn bản chỉ khác kiểu nhị phân khi xử lý ký tự chuyển dòng (0x0a) và ký tự mã (26, 0x1A). Đối với các ký tự khác, hai kiểu đều đọc ghi như nhau.

Mã chuyển dòng: Khi ghi, một ký tự **LF** (0x0a) được chuyển thành 2 ký tự **CR** (mã 13, 0x0D) và **LF**

Khi đọc, 2 ký tự liên tiếp **CR** và **LF** trên tệp chỉ cho ta một ký tự **LF.**

Một số hàm thao tác với tệp (file)

Hàm fopen: Mở tệp

+ Dạng hàm: FILE *fopen(const char *tên_tệp, const char *kiểu);

Kiểu	Ý nghĩa
“r”	Mở một file để đọc. File phải tồn tại
"w"	Tạo một file trống để ghi. Nếu một file với cùng tên đã tồn tại, nội dung của nó bị tẩy đi và file được xem như một file trống mới
"a"	Phụ thêm (append) tới một file. Với các hoạt động ghi, phụ thêm dữ liệu tại cuối file. File được tạo nếu nó chưa tồn tại

Công dụng: Hàm dùng để mở tệp. Nếu thành công hàm sẽ trả về con trỏ kiểu FILE chứa địa chỉ tệp mới mở. Nếu hàm có lỗi thì sẽ trả về giá trị NULL.

Một số hàm thao tác với tệp (file)

Hàm fclose: đóng tệp.

+ Dạng hàm:

int fclose(FILE *fp);

+ Đối:

fp là con trỏ tương ứng với tệp cần đóng.

+ Công dụng:

- Đẩy dữ liệu còn trong vùng nhớ đệm lên đĩa (đối với đang ghi)
- Xóa vùng nhớ đệm (Khi đang đọc)
- Giải phòng fp để nó có thể dùng cho tệp khác. Nếu hành công hàm có giá trị 0, Nếu thất bại hàm sẽ trả về giá trị âm.

Một số hàm thao tác với tệp (file)

Các hàm nhập xuất ký tự dùng được cả trong kiểu nhị phân và ký tự, nhưng tác dụng có những điểm khác nhau.

Hàm int fgetc(FILE *fp); hoặc fgetc(FILE *fp)

Công dụng: Hàm đọc một ký tự từ file fp. Nếu thành công hàm cho mã đọc được (có giá trị từ 0 đến 255). Nếu gặp cuối tệp hay có lỗi hàm sẽ trả về giá trị EOF(-1) (end of file).

Hàm fputc(int ch, FILE *fp); hoặc fputc(int ch, FILE *fp)

Công dụng: Hàm ghi lên file một ký tự có mã bằng m = ch%256;

Trong đó ch được xem là số nguyên không dấu. Nếu thành công hàm cho mã ký tự được ghi, trái lại hàm sẽ trả về (return) EOF.

Chú ý: Trong kiểu văn bản, nếu m = 10 (0x0A) thì hàm sẽ ghi lên tệp hai ký tự là 0x0D, 0x0A.

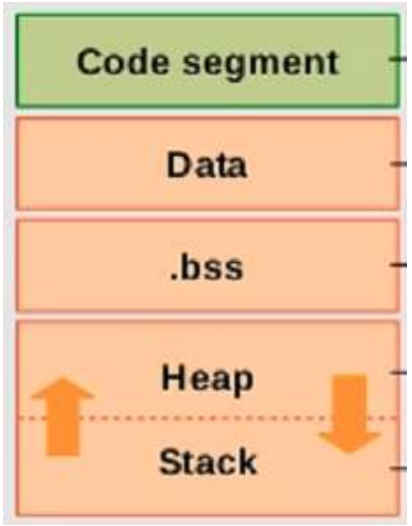
Lưu trữ dữ liệu tổ và tổ chức bộ nhớ chương trình

Mai Trong Nhan (Mr.)
Lecturer
+84 (0) 385.852.284
mainhan.imic@gmail.com

Bộ nhớ chương trình

Bộ nhớ chương trình được chia làm 5 phần chính: (thứ tự từ địa chỉ thấp đến địa chỉ cao):

0x0000



+ Text: Vùng mã lệnh và hằng. Vùng nhớ Text chỉ chịu sự chi phối của hệ điều hành, các tác nhân khác không thể can thiệp trực tiếp đến phân vùng này

+ BSS: Vùng cấp phát tỉnh, bao gồm những biến: được khai báo ở static, biến toàn cục (global variable) nhưng chưa được khởi tạo giá trị ban đầu.

+ Data: giống như vùng nhớ .BSS, nhưng nó dùng để lưu những biến có khởi tạo giá trị ban đầu.

0xFFFF

+ HEAP: Vùng cấp phát động. Được dùng để cấp phát vùng nhớ thông qua kỹ thuật dynamic memory allocation (ví dụ: sử dụng hàm malloc trong C, hoặc sử dụng new trong C++)

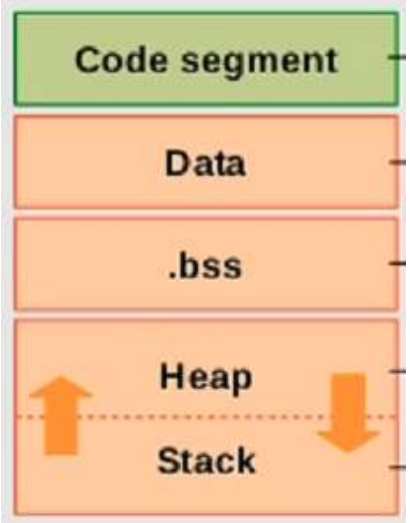
+ STACK: Vùng ngăn xếp. (chứa các đối tượng cục bộ)

*** Nhận xét:** *vùng TEXT vùng BSS và Data có độ lớn cố định trong suốt quá trình hoạt động của chương trình Vùng HEAP và STACK có độ lớn thay đổi – có thể co dãn trong quá trình hoạt động của chương trình.*

Biến toàn cục và biến cục bộ

- **Biến toàn cục:**
- Được khai báo bên ngoài hàm.
- Được lưu trữ tại vùng cấp phát tĩnh.
- Tồn tại trong suốt thời gian làm việc của chương trình
- Khi chưa khởi tạo thì giá trị bằng 0

- **Biến cục bộ:**
- Được khai báo bên trong hàm.
- Được lưu trữ tại vùng stack (ngăn xếp)
- Tồn tại từ lúc hàm được gọi đến lúc kết thúc hàm.
- Khi chưa khởi tạo thì giá trị không xác định.



Từ khóa bộ nhớ trong C

Từ khóa "*auto*"

Được dùng để đặt trước khai báo một biến bên trong hàm để chỉ rõ tính cục bộ của biến được khai báo.

→ Vì các biến khai báo trong hàm đương nhiên nó là biến cục bộ, nên từ khóa này là không cần thiết và rất ít khi được dùng.

Từ khóa "*extern*"

Được sử dụng khi một chương trình được viết trên nhiều file. Dùng từ khóa "extern" để cho compiler biết biến đó đã được khai báo ở một file nào đó. Chỉ cần gọi và thực hiện. Không cần cấp phát một vùng nhớ mới.

Từ khóa "*static*"

Dùng để khai báo các biến tĩnh cục bộ, hoặc các biến tĩnh toàn cục.

Biến tĩnh là gì?:

+ Được lưu trữ trong .BSS. (giống như biến toàn cục).

+ Nếu không khởi tạo thì giá trị mặc định là 0 (không phải là không xác định)

Phạm vi sử dụng:

+ Được OS cấp phát vùng nhớ 1 lần duy nhất.

+ Phạm vi sử dụng: trong hàm. Nhưng, Nó vẫn dữ giá trị khi ra khỏi hàm và giá trị này được giữ mỗi khi thực hiện trở lại. (nghĩa là: khi kết thúc hàm. Vùng nhớ không được trả lại cho OS)

Từ khóa bộ nhớ trong C

Từ khóa "*const*"

+ Dùng để khai báo và khởi tạo giá trị cho các biến toàn cục và biến cục bộ. Mà sau này giá trị của nó không được phép thay đổi bởi các lệnh trong chương trình.

→Nó thường được gọi là cách khai báo hằng.

Từ khóa "volatile"

Được dùng để cho OS biết giá trị của biến đó có thể được thay đổi theo một các nào đó không được mô tả trong chương trình. Chẳng hạn như các ngắt của DOS hay ROM BIOS. Biến voltatile thường là các biến toàn cục được dùng trong các thủ tục ngắt.

Tiền xử lý

Mai Trong Nhan (Mr.)
Lecturer
+84 (0) 385.852.284
mainhan.imic@gmail.com

How to make a program for computer

Step 1: Write the source codes (.c) and header files (.h).

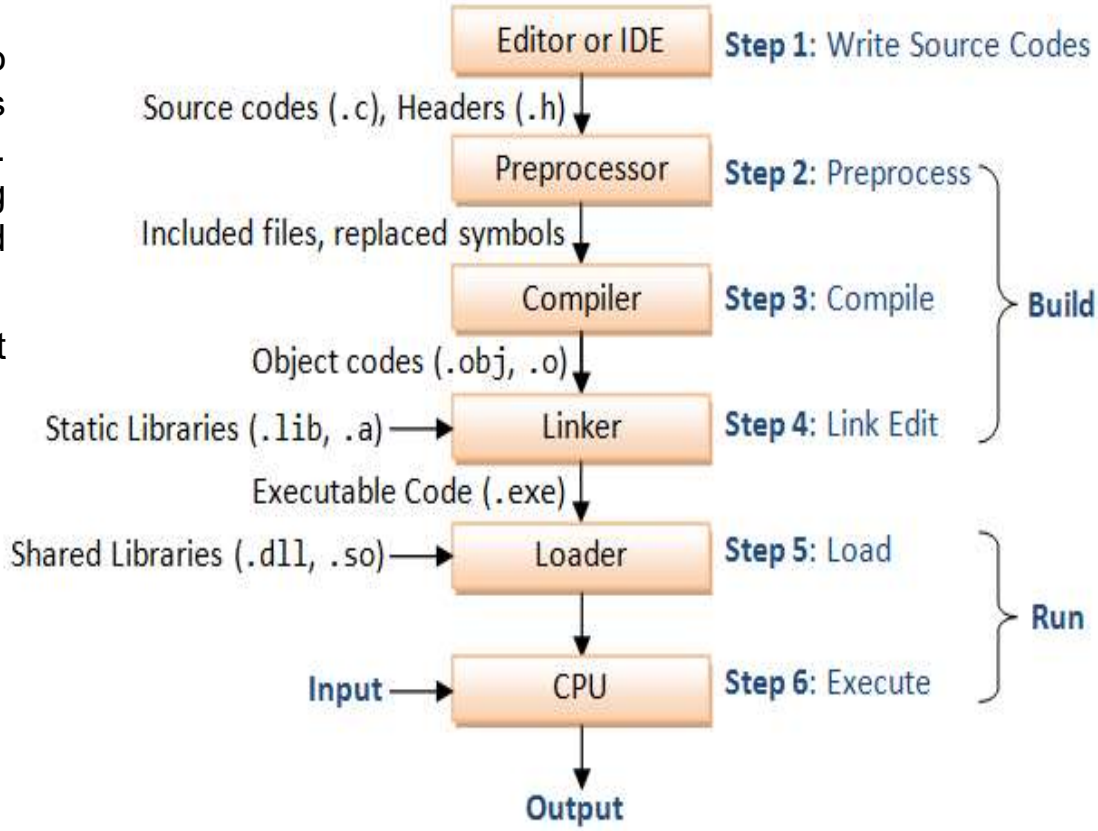
Step 2: Pre-process the source codes according to the *preprocessor directives*. The preprocessor directives begin with a hash sign (#), such as #include and #define. They indicate that certain manipulations (such as including another file or replacement of symbols) are to be performed BEFORE compilation

Step 3: Compile the pre-processed source codes into object codes (.obj, .o).

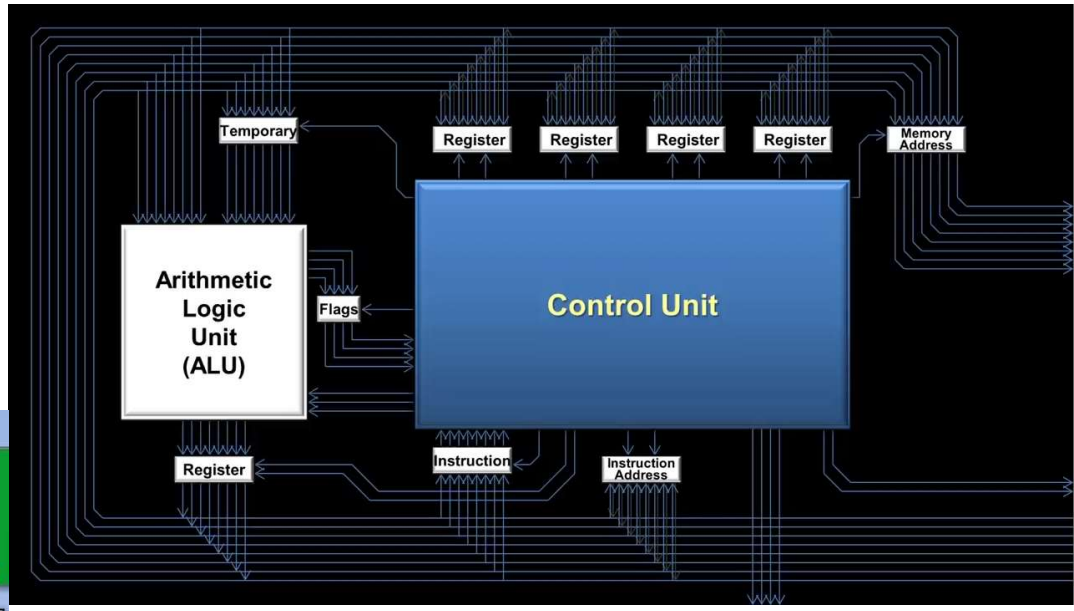
Step 4: Link the compiled object codes with other object codes and the library object codes (.lib, .a)to produce the executable code (.exe)

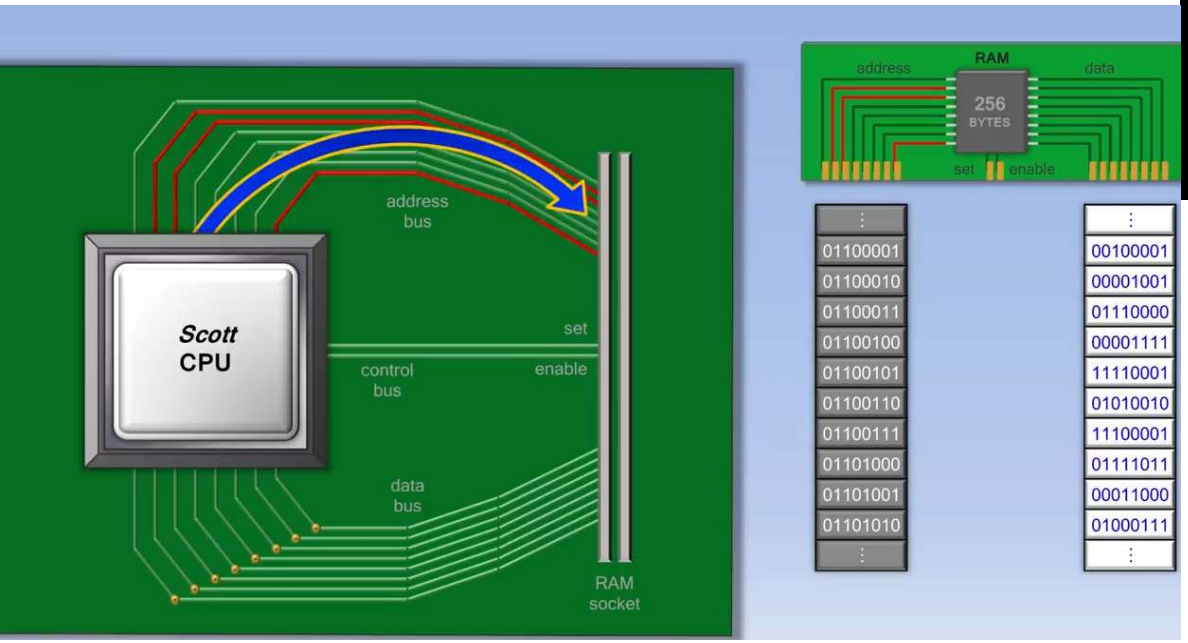
Step 5: Load the executable code into computer memory.

Step 6: Run the executable code



How to a program run in computer





Tiền xử lý define:

#define

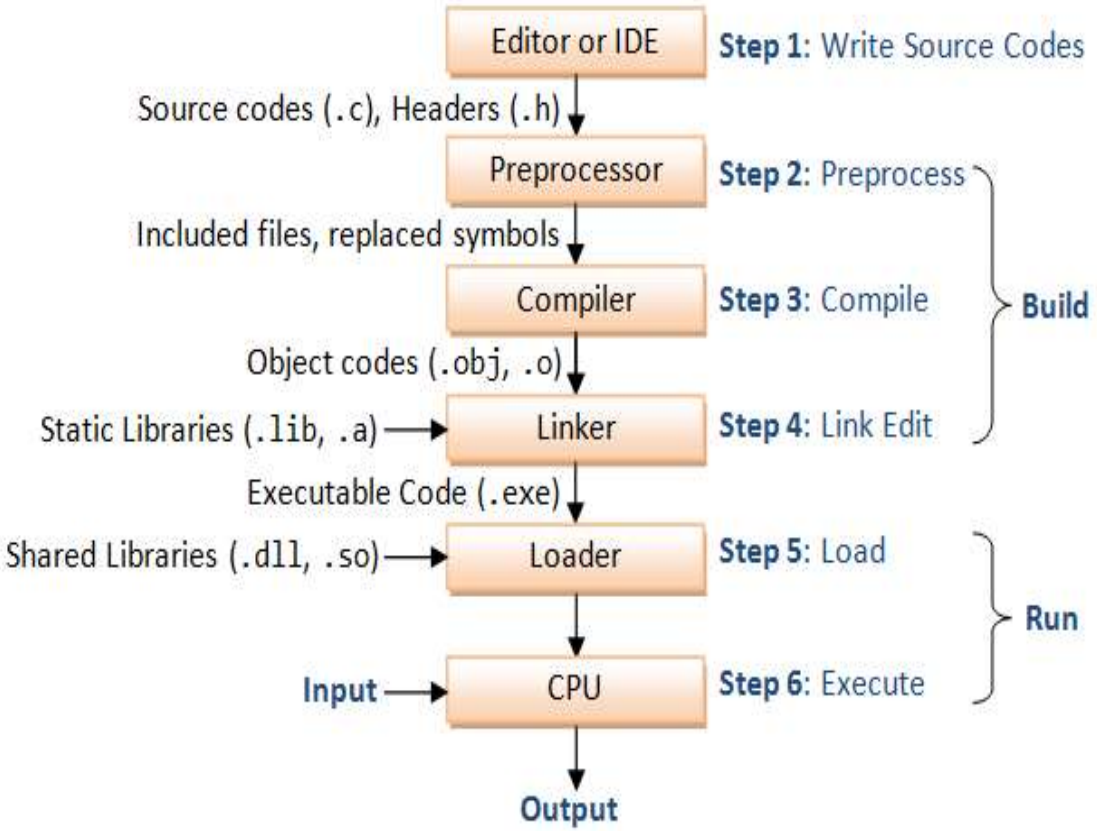
Cho phép tạo ra các macro.

```
#define tên_macro
#define tên_macro giá_trị_macro
```

Có tác dụng thay thế tên_macro bằng giá_trị_macro. Trước khi chương trình được biên dịch thành mã bin: xét VD:

```
#define A 10
#define B 20
#define C A+B
#define D C*10
```

Vậy D có kết quả bao nhiêu.?



Các chỉ thị biên dịch có điều kiện #if, #ifdef và #ifndef.

Có tác dụng lựa chọn đoạn chương trình để build.

```
#if điều_kiện_1
    đoạn chương trình 1;
#elif điều_kiện_2
    đoạn chương trình 2;
#else
       đoạn chương trình 3;
#endif
```

Nếu “điều_kiện_1” đúng thì đoạn_chương_trình 1 được build. Nếu điều_kiện_2 đúng thì đoạn_chương_trình_2 được lựa chọn để build. Nếu điều_kiện_1 và điều_kiện_2 đều sai thì đoạn_chương_trình_3 được lựa chọn để build.

```
#ifdef macro
    đoạn chương trình 1;
#else
       đoạn chương trình 2;
#endif
```

Nếu macro đã được định nghĩa bởi #define macro thì đoạn chương trình 1 được chọn để build. Còn nếu không được định nghĩa thì đoạn chương trình 2 được chọn để build

```
#ifndef macro
    đoạn chương trình 1;
#else
       đoạn chương trình 2;
#endif
```

Ngược lại với #ifdef.

Nếu macro không được định nghĩa thì chọn đoạn chương trình 1 để build, nếu macro được định nghĩa thì chọn đoạn chương trình 2 để build

Chỉ thị #error

- Chỉ thị #error được dùng để dừng biên dịch và in ra bản tin báo lỗi.
- Có nghĩ compiler sẽ ngừng lại khi gặp #error.

Thank you.