Bài 5: Storage Classes : Extern - Static - Volatile – Register

Là các biến đặc biệt

5.1.Extern

Đặt vấn đề:

- Giả sử folder code của bạn có 3 file c (test.c, test1.c, test2.c). Bạn đang làm việc với test.c. Bạn có thể dùng #include để copy toàn bộ mã nguồn code của test1.c và test2.c => test.c từ đó có thể sử dụng tất cả các biến và hàm của 2 file này.

  #include “test1.c”
  #include “test2.c”
- Nhược điểm : là copy nhưng mã nguồn không cần thiết vào chương trình => làm nó nặng hơn, tốn bộ nhớ hơn. Để chỉ sử dụng 1 biến nào của test1, test2 thì ta dùng từ khóa extern.
- Tuy nhiên thông thường người ta cũng không khai báo extern trong các file.c mà người ta tạo ra các file .h (test1.h, test2.h) và sẽ khai báo extern trong file .h để dùng chung cho các file. Trong các file .c sẽ thực hiện include các file .h này vào. Tại sao lại như vậy ????
- Bởi file .h hoạt động như file thử viện. trong đây chỉ chứa macro. Chúng giống như việc bạn tạo 1 mục lục , gạch đầu dòng ra tên các biến các hàm sẽ sử dụng mà không truyền bất cứ giá trị cụ thể cho các biến đó và Cũng không khai báo cụ thể hàm đó đó thực hiện như thế nào => file .h sẽ nhẹ hơn, ít dòng lệnh hơn => việc include các file.h vào file.c cũng sẽ không gây cho code chương trình phình to hơn => chương trình sẽ nhẹ hơn, ít dòng code hơn, chạy sẽ nhanh hơn
- Notes: câu lệnh #include nó sẽ thực hiện copy toàn bộ mã nguồn của file này vào file kia.
- Khi khai báo 1 file header .h, nó chỉ liệt kê ra các biến, hàm thì phải khai báo 1 file .c trùng tên để triển khai thực triển khai code cho hàm biến đó trong File .h. ví dụ: tạo file test.h, thì file tạo thêm file test.c để triển khai code.

Ví dụ:

Hàm Main:

```c
#include <stdio.h>
#include "test1.h"
#include "test2.h"

int main(int argc, char const *argv[])
{
   display1();
   display2();
```

```
    a = 50;
    b = 20;

    printf("value a = %d \n", a);
    printf("value a = %d \n", b);

     return 0;
}
```

test1.h

```
// File .h là cac file làm việc ở cấp độ macro, nó sẽ truyền giá trị cụ thể cho biến hoặc hàm
// file header như mục lục liệt kê các biến, hàm sẽ sử dụng cho file.c, trong file.c cụ thể hóa biến hàm này.
// biến extern thường được sử dụng trong các file .h, ít dùng trong các file .c
#ifndef test1_h
#define test1_h

extern int a; // khai báo extern khi muốn biến đó dùng chung cho các file.

void display1();

#endif
```

test1.c

```
- #include <stdio.h>
- #include "test1.h"
- int a = 10;
-
- void display1(){
-
-     printf("this is test1.c file \n");
-
- }
-
```

- Test2.h

```
- / File .h là cac file làm việc ở cấp độ macro, nó sẽ truyền giá trị cụ thể cho biến hoặc hàm
- // file header như mục lục liệt kê các biến, hàm sẽ sử dụng cho file.c, trong file.c cụ thể hóa biến hàm này.
- // biến extern thường được sử dụng trong các file .h, ít dùng trong các file .c
- #ifndef test2_h  // nếu chưa định nghĩa test2.h
- #define test2_h  // định nghĩa file test2.h, 2 câu lệnh này tránh tạo trùng file
```

```
-
-   extern int b;
-
-   void display2();
-
-   #endif
```

test2.c

```
-   #include <stdio.h>
-   #include "test1.h"
-   int b = 9;
-
-   void display2(){
-
-       printf("this is test2.c file \n");
-
-   }
-
```

Kết quả chạy

```
extern_main.c:13:4: note: include <stdio.h> or provide a declaration of 'printf'
PS E:\3.ELECTRONIC\.HALA-EMBBED PROGRAMMING_GOOD\C & C++ ADVANCE\EXCERCISE\Bai 5_Storage Classes\Extern> gcc
PS E:\3.ELECTRONIC\.HALA-EMBBED PROGRAMMING_GOOD\C & C++ ADVANCE\EXCERCISE\Bai 5_Storage Classes\Extern> ./e
this is test1.c file
this is test2.c file
value a = 50
value a = 20
PS E:\3.ELECTRONIC\.HALA-EMBBED PROGRAMMING_GOOD\C & C++ ADVANCE\EXCERCISE\Bai 5_Storage Classes\Extern>
```

5.2. Static

Biến static (biến tĩnh) có đặc điểm khai báo ở đâu sử dụng ở đó.

5.2.1.Static local variables

Khi static được sử dụng với local variables (biến cục bộ) - khai báo biến trong một hàm). static cục bộ sẽ giữ nguyên địa chỉ của biến hoặc hàm suốt chương trình. Nhờ vậy giá trị của biến sau mỗi lần tính toán, gọi hàm sẽ không bị reset về giá trị gán ban đầu.

```
#include <stdio.h>
int *ptr = NULL;
void CountFunc1() {
    int a = 5;  // 0x0a : STACK (phân vùng trong bộ nhớ)
    printf("a: %d\n", ++a); // Note: ++count thì sẽ + lên trươc khi in ra, còn
nếu việt count++ thì nó sẽ thực hiện in trước khi+
}
void CountFunc2() {
    static int count = 5; // 0x09 : BSS
    ptr = &count;
    printf("address: %p, Count: %d\n", ptr, ++count);
}
void CountFunc3() {
    static int c = 0; // 0x0F : Data

    printf(" c: %d\n", ++c);
}
/*
- Tùy cách khai báo biến mà chương trinh C sẽ cấp cho phân vùng khác nhau
(Stack, BSS, Data), mỗi phân vùng lại có 1 đặc điểm riêng
- Nếu biên trong phân vùng Stack thì biến count được cấp địa chỉ trong phân
vùng stack. đặc điểm phân vùng này sẽ thu hồi địa chỉ
    ngay sau khi kết thúc hàm. bởi vậy CountFunc1() in ra 3 con 6.
- Nếu dùng static khai báo cho biến, và gán giá trị khác 0 (CountFunc2) thì
biến đó sẽ vào phân vung BSS
    Đặc điểm: BSS chỉ thu hồi địa chỉ bộ nhớ sau khi kết thúc toàn bộ chương
trinh, hoặc tắt vscode đi nó mới thu hồi. Địa chỉ tồn tại xuyên suốt chương
trình
    bởi vậy chương trình in ra là 6,7,8
- Nếu dùng static khai báo và giá trị = 0 thì nó sẽ phân vào phân vùng Data.

*/

int main() {
    CountFunc1();  // In ra "Count: 6"
    CountFunc1();  // In ra "Count: 6"
    CountFunc1();  // In ra "Count: 6"
    printf("\n");
```

```
    CountFunc2();  // In ra "Count: 6"
    CountFunc2();  // In ra "Count: 7"
    CountFunc2();  // In ra "Count: 8"
    printf("\n");

    CountFunc3();  // In ra "Count: 1"
    CountFunc3();  // In ra "Count: 2"
    CountFunc3();  // In ra "Count: 3"

    return 0;
}
```

Ket quả

```
staticlocal_var && "e:\3.ELECTRONIC\.HALA-EMBBED PROGRAM
a: 6
a: 6
a: 6

address: 00007FF7379A3000, Count: 6
address: 00007FF7379A3000, Count: 7
address: 00007FF7379A3000, Count: 8

c: 1
c: 2
c: 3

[Done] exited with code=0 in 0.247 seconds
```

5.2.2.Static global variables

Khi static được sử dụng với global variables (biến toàn cục) - khai báo biến bên ngoài hàm), nó hạn chế phạm vi của biến đó chỉ sử dụng được trong file nguồn đó. Các file khác không thể lấy ra sử dụng được biến hoặc hàm đó.

Ứng dụng: dùng để thiết kế các file thư viện. (file .h), những hàm, biến trung gian không muốn người dùng tác động tới, thì khai báo static.

File motor.h

```
#ifndef __MOTOR_H
#define __MOTOR_H

typedef struct {
    void (*start)(int gpio);
   void (*stop)(int gpio);
   void (*changeSpeed)(int gpio, int speed);
} MotorController;

typedef int PIN;

static void startMotor(PIN pin);
static void stopMotor(PIN pin);
void changeSpeedMotor(PIN pin, int speed);

// khi khai báo toàn cục static void như này thì các hàm startMotr, stopMotor
chỉ sử dụng được trong file motor.h, và motor.c này
// các file như main.c cũng không thể gọi ra hoặc khai báo 1 hàm trùng tên ""
startMotr, stopMotor" này được

void init_motor(MotorController *motorName);


#endif
```

File motor.c

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


// General function
void startMotor(PIN pin) {
   printf("Start motor at PIN %d\n", pin);
}

void stopMotor(PIN pin) {
   printf("Stop motor at PIN %d\n", pin);
}
```

```
void changeSpeedMotor(PIN pin, int speed) {
   printf("Change speed at PIN %d: %d\n", pin, speed);
}


void init_motor(MotorController *motorName)
{
   motorName->start = startMotor;
   motorName->stop = stopMotor;
   motorName->changeSpeed = changeSpeedMotor;
}
```

5.2.3.Static trong class

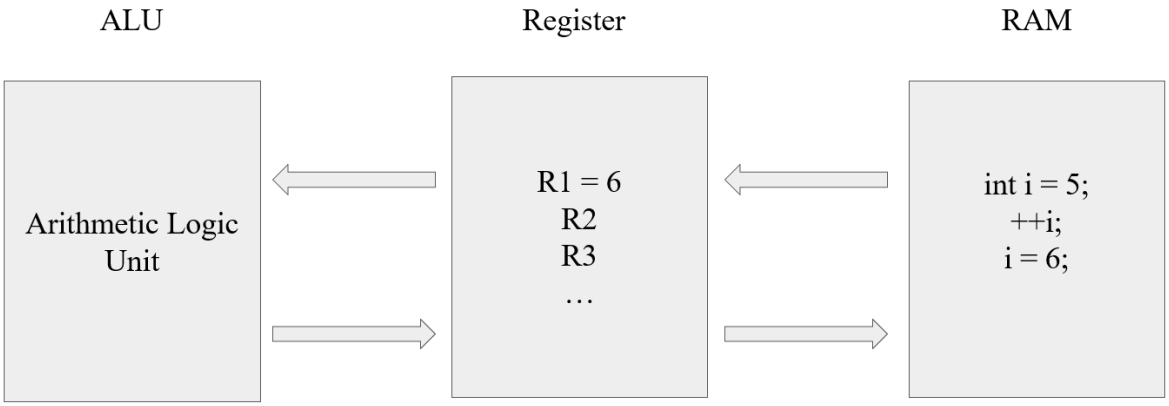
Khi một thành viên của lớp được khai báo là static, nó thuộc về lớp chứ không thuộc về các đối tượng cụ thể của lớp đó. Các đối tượng của lớp sẽ chia sẻ cùng một bản sao của thành viên static, và nó có thể được truy cập mà không cần tạo đối tượng. Nó thường được sử dụng để lưu trữ dữ liệu chung của tất cả đối tượng.

5.3 Register

Xét quy trình tính toán của bộ ALU.
Đầu tiên khi bạn khai báo biến i=5, ++ i, thì RAM cấp bộ nhớ để lưu biến i và phép +.
Sau đó giá trị 5 sẽ được ghi vào 1 thanh ghi còn trống nào đó,
Phép + cũng lưu vào thanh ghi còn trống.
Sau đó các dữ liệu này chuyển qua ALU để tính toán, sau khi tính toán xong, kết quả được ghi vào 1 thanh ghi còn trống nào đó, rồi nó mới trả vào địa chỉ của biến trên RAM lúc đó biến
May tính có thể có vài chục thanh ghi khác nhau R1…Rxx. Máy tính 64 bit mỗi thanh ghi sẽ có 64 bit 0xFFFF FFFF FFFF FFFF. Bằng việc đảo các bit dữ liệu trên thanh ghi (0->1, 1>0) mà có thể lưu trữ được giá của biến nào đó.
Ý tưởng: để cho việc thực thi được nhanh hơn thì ta ghi thẳng dữ liệu vào thanh ghi, để việc tính toán được nhanh hơn. Để làm được điều này thì dùng biến register.
Đặc điểm:

- Register không dùng được cho biến toàn cục
- Không dùng được con trỏ để đọc giá trị của nó.

Register



Ví dụ: Chương trình đo thời gian thực hiện của một vòng lặp for, khi không dùng register và dùng register.

```c
// chương trình đo thời gian thực hiện vòng lặp for
#include <stdio.h>
#include <time.h>

int main() {
    // Lưu thời điểm bắt đầu
    clock_t start_time = clock();
    register int i;

    // Đoạn mã của chương trình
    for (i = 0; i < 2000000; ++i) {
        // Thực hiện một số công việc bất kỳ
    }

    // Lưu thời điểm kết thúc
    clock_t end_time = clock();

    // Tính thời gian chạy bằng miligiây
    double time_taken = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;

    printf("Thoi gian chay cua chuong trinh: %f giay\n", time_taken);

    return 0;
}
```

Kết quả nếu không dùng register, thời gian chạy của chương trình là 0.007s.
Ghi thẳng dữ liệu vào thanh ghi thì thời gian chạy chương trình chỉ còn 0.001s (nhanh hơn 7 lần)

```
[Running] cd "e:\3.ELECTRONIC\.HALA-EMBBED PROGRAMMING_GOOD\C & C++ ADVANCE\EXCERCISE\B
"e:\3.ELECTRONIC\.HALA-EMBBED PROGRAMMING_GOOD\C & C++ ADVANCE\EXCERCISE\Bai 5_Storage
Thoi gian chay cua chuong trinh: 0.007000 giay

[Done] exited with code=0 in 0.353 seconds

[Running] cd "e:\3.ELECTRONIC\.HALA-EMBBED PROGRAMMING_GOOD\C & C++ ADVANCE\EXCERCISE\B
"e:\3.ELECTRONIC\.HALA-EMBBED PROGRAMMING_GOOD\C & C++ ADVANCE\EXCERCISE\Bai 5_Storage
Thoi gian chay cua chuong trinh: 0.001000 giay

[Done] exited with code=0 in 0.323 seconds
```

5.3. Volatile

Từ khóa volatile trong ngôn ngữ lập trình C được sử dụng để báo hiệu cho trình biên dịch rằng một biến có thể thay đổi ngẫu nhiên, ngoài sự kiểm soát của chương trình. Việc này ngăn chặn trình biên dịch tối ưu hóa hoặc xóa bỏ các thao tác trên biến đó, giữ cho các thao tác trên biến được thực hiện như đã được định nghĩa.