Funtion scope

Global scope meaning function is visible or using in multiple translate unit.

Internal scope means function is only using in single translate unit.

Single translate unit là 1 file.c. (o hay c?)

Mỗi file.o hay .s hay .c, mỗi file đó đều là 1 single translate unit.

Mỗi 1 **file.o** là 1 translate unit.

Sử dụng trong muttiple là sử dụng trong các file .c

Single translate chỉ sử dụng trong file .c mà chứa internal scope của function đó.

Function chỉ sử dụng trong 1 file thì có cần đưa lên file.h ko? (có).

Funtion được complie mà chỉ sử dụng trong file thì có nghĩa nó là internal scope và chỉ được biên dịch nội trong cái translate unit.

Function hay quan trọng hơn là symbol của nó về bản chất, complier ko hiểu cái này là function cái nào là biến nhưng nó sẽ hiểu cái tên của nó.

Cái **tên** đó nó sẽ được gọi là **symbol**, đại diện cho **địa chỉ hay object nằm ở đó**, nhưng complier ko quan tâm object đó là gì mà chỉ quan tâm 1 object mà có bộ nhớ nhất định, có địa chỉ nhất định.

Tất cả function của mình chỉ có 1 function có symbol. Nhưng những function nào tồn tại **internal scope**, tức là chỉ có trong file đó thì ko có symbol vì nó ko cần nhảy đến, trỏ đến, ko cần con trỏ trỏ đến nó nên cái tên của nó ko mang ý nghĩa là địa chỉ.

Cái symbol này gần như là để đại diện cho 1 file nào đấy và symbol dùng để liên kết các file đấy và khi quá trình link xuất hiện hay trong complier viết tắt là chữ ld thì link sẽ cần thông tin liên quan đến các symbol này và sẽ collect các symbol này với nhau và nó sẽ đặt vào những phần địa chỉ.

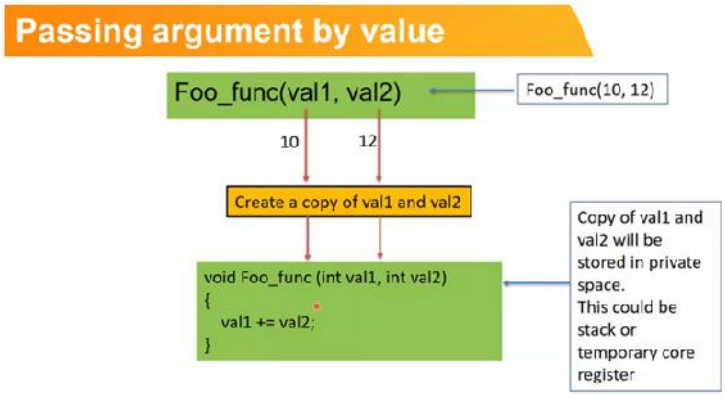
Ví dụ file A.c chứa 1 symbol, file B có hàm gọi đến symbol hay chính xác hơn là Symbol này là symbol của hàm đấy, thì nó sẽ tạo 1 liên kết từ file B đến file A này thông qua symbol này, cái function hay symbol của function này, cái scope của nó sẽ chuyển thành scope global chứa trong nhiều translate unit và cụ thể hơn là function B sẽ tạo 1 reference hay 1 link liên kết đến file A và sẽ tìm symbol trong file A. file A chứa định nghĩa của function đấy, file B chứa reference đến symbol đấy, cái đấy ngta gọi là global scope hay sẽ tạo 1 liên kết giữa file b và file a.

Lúc complie có 1 key word extern trong file B thì có nghĩa là ta đã tạo 1 liên kết từ fileB đến fileA. Nếu trong file A có nhỡ tay xoá định nghĩa của function đấy hay của symbol

đấy thì đến phần link sẽ báo ngay 1 lỗi là undefi symbol nghĩa là file B đang tìm symbol đó trong file A, thực ra nó tìm cả project nhưng vấn đề là nó ko tìm thấy cái này ở đâu cả thì nó sẽ báo lỗi symbol hay function name là tôi hiểu cái này là gì (linking).

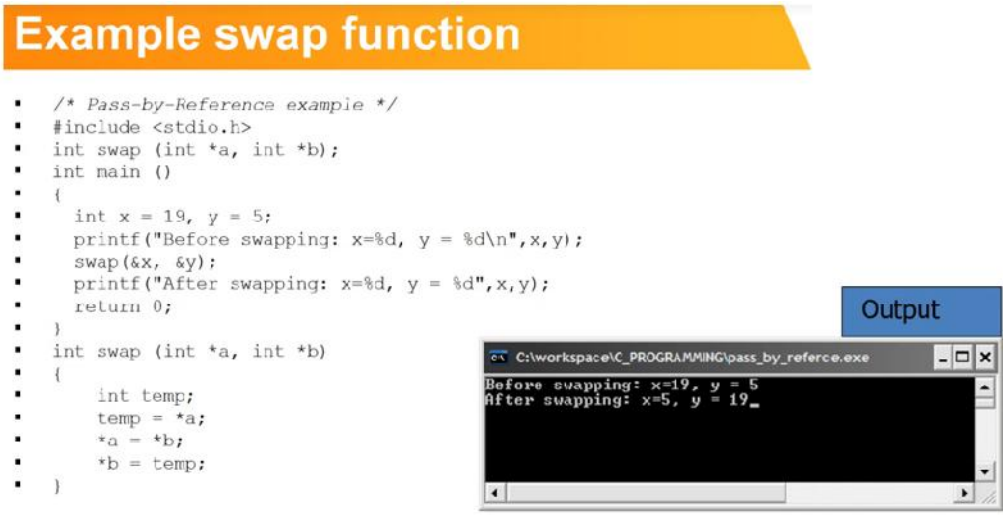
Tham trị, tham chiếu hay là pass argument hay reference.

Chú ý **private stack** là bộ nhớ riêng của core.



Khi mà tạo ra 1 function con mà cần truyền giá trị vào mà mình ko thể thay đổi nó, còn việc lưu trữ nó thì ko thể lưu trữ vào 2 thằng này được, khi mà parse 2 giá trị của 2 tham số này vào private register này thì nó sẽ tạo ra 1 bản ghi nhỏ trên bộ nhớ riêng và bản ghi đó chỉ để tính toán chứ ko để lưu trữ, nếu dùng để lưu trữ thì cuối cùng nó vẫn bị giải phóng

Là các thanh ghi R0 R1 R2 R3

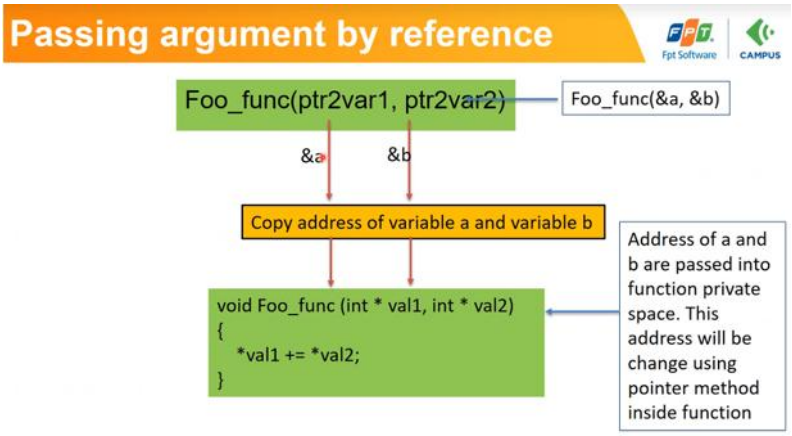


Tạo 1 function mà mình truyền giá trị của 2 biến vào và kết thúc function đó thì 2 giá trị này đổi chỗ cho nhau.

Cách chuyển là chúng ta sẽ không truyền giá trị của x và y mà là truyền địa chỉ của x và y vào. Trong function swap là nó sẽ dùng con trỏ để truy cập địa chỉ truyền vào và ở đây mình chỉ có thông tin về địa chỉ và giá trị thực sự sẽ bị con trỏ modify đi.

Bản chất của swap là nó sẽ truyền địa chỉ của thằng A và B vào chứ nó ko truyền giá trị của thằng A và thằng B và khi mình có địa chỉ của thằng A và B thì mình có thể dùng con trỏ để mình modify dữ liệu lên đấy.

Bản chất con trỏ là 1 biến lưu trữ địa chỉ mà nó trỏ đến mà khi mình có địa chỉ đó thì mình có thể dùng phương pháp dấu * để mình có thể modify giá trị mà nó trỏ đến



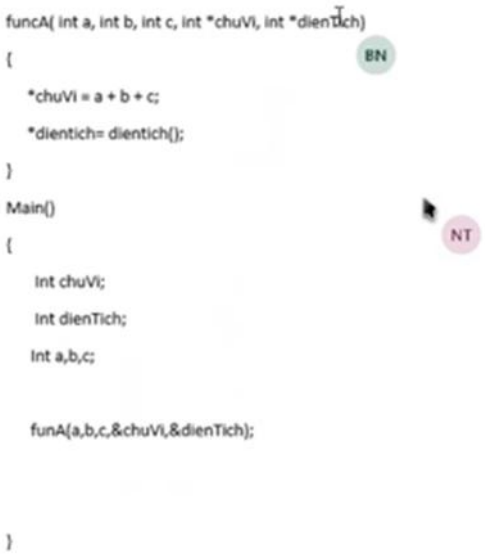
Lừa complier là sẽ bảo tôi chỉ truyền địa chỉ của thằng này vào đây nhưng bên trong dùng chính địa chỉ của val1 để thay đổi giá trị của chính nó thì nó được gọi là phương pháp pass argument by reference tức là tham chiếu. khi mà truyền địa chỉ của thằng A và B vào đấy thì nó sẽ tạo 1 bản ghi copy giá trị của thằng a và b này. Giá trị của a và b hiện tại là địa chỉ của a và b thì dẫn đến việc bên trong sẽ thay đổi những gì liên quan đến những gì mà nó trỏ đến thì thực tế là trên bộ nhớ nó cũng thay đổi theo

Nếu đổi dấu * thành dấu & ở void Foo_func (int &val1, int &val2) thì C ko complie được.

Int *val1, bản chất là mình đang defination function là kiểu con trỏ, trỏ đến kiểu dữ liệu là int, tức là mình truyền vào địa chỉ, địa chỉ là kiểu int 32 bit (hệ điều hành 32 bit) còn với function Foo_func(&a, &b) thì mình phải truyền địa chỉ thực sự, 1 cái value thực sự. còn int *val1 bắt buộc là 1 địa chỉ có kiểu vì c bao giờ tất cả dữ liệu được định nghĩa trong function cũng phải có kiểu, định dạng.

Bài tập: cho hình tam giác, viết 1 function trả về giá trị chu vi và diện tích của hình tam giác đó.

Truyền tham chiếu trực tiếp vào hàm đó



```
funcA( int a, int b, int c, int *chuVi, int *dienTich)
{
   *chuVi = a + b + c;
   *dientich= dientich();
}
Main()
{
    Int chuVi;
    Int dienTich;
   Int a,b,c;

   funA(a,b,c,&chuVi,&dienTich);


}
```

Với cách này, có thể xuất output thẳng vào function này

Int funcA ()

{

Return x;

}

Chỉ return được 1 giá trị

Cách cơ bản: trả luôn thẳng về giá trị được lưu trong ô nhớ mà thằng người dùng ngta truyền vào. Để làm theo cách này người dùng chuẩn bị 2 biến (ô nhớ) để lưu giá trị ấy và việc của ngta là truyền địa chỉ của 2 ô nhớ vào trong function của mình và function của mình có trách nhiệm modify 2 cái ô nhớ đó bằng cái địa chỉ mà thằng này sẽ truyền vào. Có nghĩa là function này có nhược điểm là nếu ng dùng ko chuẩn bị sẵn bộ nhớ để lưu trữ cái đấy thì nó sẽ ko chạy được, ưu điểm là tích bao nhiêu phần output cũng được.

Ưu điểm thứ nhất là sẽ trả về rất nhiều parameter chẳng hạn

Ưu điểm thứ 2 là nếu bài tập này liên quan đến bài tập swap(a,b), void add(a,b){ a +=b;} là lưu vào a, có nghĩa là có khả năng modify chính trên input truyền vào, tức là input cũng là output luôn.

Swap (a,b), input là a và b, output là a và b

Đối với add(a,b) intput là a và b, output là tổng a+b lưu ở (bắt buộc có chỗ mà nó trả về), nếu ko thể lưu được thì nó ko có output, nếu như hàm nó returrn thì người dùng ở ngoài phải có 1 biến để đón đầu cái đó hoặc là output có thể sử dụng ở phép tính khác

Nếu output ko lưu ở đâu cả thì có nghĩa là nó ko có output, chứ ko phải là phép tổng, output ở đây là a

```
#include <stdio.h>

/*Khai báo giá tri chứa tổng và tích */
struct tinhTongTich{
    float ChuVi;
    float DienTich;
};

struct tinhTongTich DienTichChuVi (int a, int b, int c){
   struct tinhTongTich A;
   A.DienTich = (a*b)/2;
   A.ChuVi = a+b+c;
}
```

```
int main(){
    int a;
    int b;
    int c;

    printf("Nhap gia tri cac canh cua tam giac; ");
    scanf("%d", &a,&b, &c);

    struct tinhTongTich B = DienTichChuVi(a,b,c);
    printf("Dien tich la: %f", &B.DienTich);
    printf("Chu vi la: %f", &B.ChuVi);
    return 0;
}
```

Cách 2 vẫn là tham chiếu nhưng mà thay vì có nhiều wrapper thì ngta sẽ tạo ra 1 kiểu dữ liệu có thể là struct hay mảng sẽ dùng con trỏ để modify trên từng dữ liệu, tức là ngta sẽ truyền dữ liệu bắt đầu từ kiểu dữ liệu đấy và nó sẽ dùng con trỏ để modify dữ liệu trên cái ô nhớ có sẵn đấy. cái này nó yêu cầu kiểu dữ liệu khởi tạo từ trước.

Cách này thì hay dùng trong các thư viện, tức là để **hạn chế số lượng parameter truyền vào quá lớn** dẫn tới việc bộ nhớ tạm của cpu ko chứa nổi thì ngta sử dụng con trỏ duy nhất nhưng mà con trỏ đó là con trỏ kiểu struct.

Rules trong lập trình, thứ nhất là ko tạo 1 function quá 4 đến 5 parameter, nếu hơn thì performance sẽ chậm.

Nếu trong trường hợp cần tạo nhiều thế thì ngta dùng kiểu struct và trong trường hợp sử dụng đến tham chiếu thì người ta sẽ dùng con trỏ trỏ đến struct.

Struct có nhược điểm là kiểu dữ liệu của nó thường phức hợp và thường người dùng sử dụng thư viện ko nhớ trên struct đấy dẫn tới việc nó gây khó khăn cho người dùng nên người ta cũng hạn chế dùng struct trong khi người dùng muốn dùng kiểu struct đó ngta phải define kiểu typedef.

Khi mà ta design 1 cái thư viện làm sao cho người dùng dễ sử dụng nhất, người dùng ngta chỉ cần inlcude 1 file header duy nhất thì nó có thể sử dụng luôn.

Nhưng mà trong file header đó phải có định nghĩa về kiểu, liên quan đến các internal data và phải export cho người dùng để ngta sử dụng điều này và người dùng có khả năg truy cập đến các phần bộ nhớ data đấy.

Lỡ như người dùng modify data đó thì cả thư viện ko chính xác, dẫn tới ngta có thể trách chương trình của mình là vớ vẩn → có nhiều vấn đề khi mình expore quá nhiều dữ liệu external cho người dùng nên việc sử dụng tham chiếu cũng rất có lợi tại vì chỉ truyền vào đấy thôi, các dữ liệu global thì người ta ko thể access lên được.

Để tiếp cận các loại thư viện, các dữ liệu cdata hay những dữ liệu dạng global thì người dùng ko có cách nào ngta có thể truy cập vào được, mục đích là giấu nó đi. Mục đích là người dùng ko được export, share biến global giữa các file là vì thế