Keyword cho biến trong C

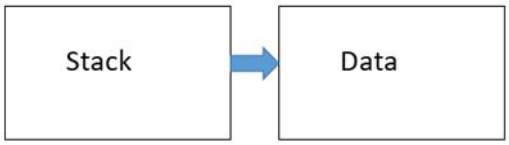
4 từ khoá: static, extern, register, volatile

Static dùng cho nhiều ngôn ngữ: C++, java. Với ngôn ngữ C, từ khoá này sẽ thay đổi, điều khiển thời gian sống của 1 biến, sau phần lớp lưu trữ thì ta đã biết, ở lớp lưu trữ, đại diện các phần đó là thời gian sống của 1 biến, scope 1 biến và vùng nhớ của biến lưu trữ.

Từ khoá này khi mà khai báo 1 biến thì nó sẽ thay đổi những gì của biến?

Static sẽ áp dụng cho global variables (biến toàn cục), chúng ta có static của function và chúng ta có static cho local variables

Ví dụ về static của local variables, khi có từ khoá static cho biến local variables. Như chúng ta đã biết nếu 1 biến local ko được khai báo 1 biến static thì biến local này sẽ được cấp phát tại vùng nhớ stack nhưng khi chúng ta khai báo từ khoá static thì biến local này sẽ được khai báo trong vùng data.

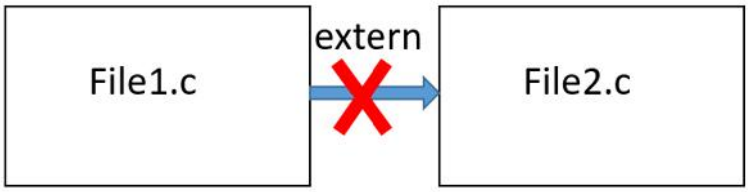


Và như chúng ta đã biết, khi 1 từ khoá biến local bình thường thì có thời gian sống được bắt đầu từ khi hàm được gọi và kết thúc khi hàm hoạt động xong (thoát khỏi hàm đó).

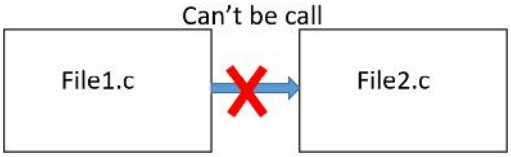
Khi có từ khoá static thì biến local nó sẽ được từ khoá static khi mà được gọi (khai báo trước 1 biến local) thì nó sẽ được cấp phát từ lúc hàm đó được gọi đầu tiên và sẽ được free đi khi chương trình kết thúc, có nghĩa là khi mà hàm khai báo biến static này dù có kết thúc hàm đó nhưng chương trình ko kết thúc

Khi lưu trữ trong biến **static** này sẽ vẫn còn lưu trữ lại các giá trị khi mà lần sau đó gọi khi mà **ko cần khai báo lại nữa**.

Với biến static global thì với biến này sẽ ko thay đổi thời gian sống của 1 biến nhưng nó sẽ quy định tầm vực của biến đó sẽ chỉ được sử dụng ở trong file code khai báo biến global đấy & khi đã có từ khoá global thì sẽ ko extern được biến này sang các file code khác



Từ khoá static cho function, function này chỉ được call ở trong chính các file mà khai báo hàm đấy thôi, hàm này sẽ ko được call từ các file khác dù cho có include.



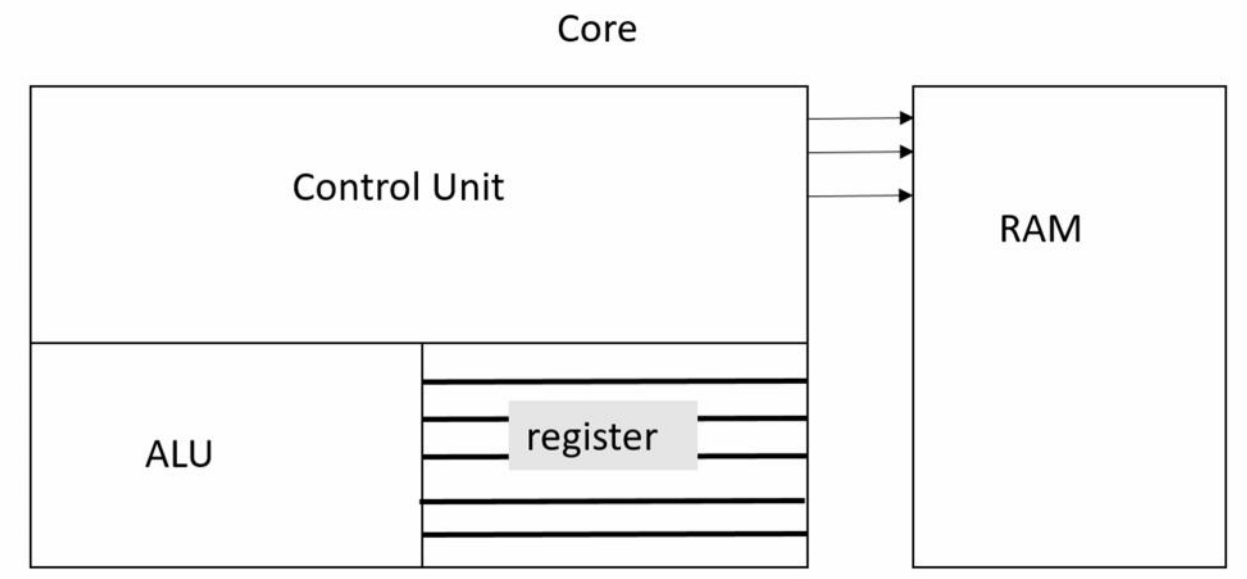
Extern là từ khoá giúp chúng ta sử dụng extern 1 biến ở trong C, là khi 1 từ khoá (từ khoá extern sẽ chỉ áp dụng cho 1 biến global) thì như bình thường ta thấy 1 biến global sẽ chỉ được nhận biết trong file source code và được khai báo biến global đó thôi.

Giả sử trong chương trình có nhiều file code và các file code này muốn sử dụng 1 biến được khai báo tại 1 file code nào đó thì chúng ta sẽ sử dụng từ khoá **extern**, với từ khoá extern này chúng ta sẽ được áp dụng cho 2 đối tượng đó là **extern biến** và **extern hàm.**

Extern biến, khi các file khác chỉ cần extern variable và chúng ta đã có thể sử dụng được biến mà khai báo ở hàm trước, các hàm này sẽ phải được sử dụng trong header file.

Extern hàm, khi các hàm muốn được sử dụng trong các file code khác thì chúng ta sẽ include file header file vào và sử dụng từ khoá extern cho biến được khai báo cho cái header file đó thì file source code trong chương trình sẽ sử dụng được hàm được khai báo, sử dụng trong file header file được include vào

Từ khoá **register** ít khi được sử dụng nhưng quan trọng.



Từ khoá này khi đứng trước 1 biến nó sẽ nói cho chương trình biết rằng cái biến chúng ta khai báo được cấp phát trong vùng thanh ghi của CPU thay vì khai báo cấp phát trên vùng nhớ bởi vì thường trong CPU của chúng ta có 3 phần CU, ALU và các thanh ghi,

các thanh ghi này giúp chúng ta có thể tính toán, lưu trữ các kết quả của bộ phận tính toán logic và số học từ bộ phận ALU ra.

Như vậy với việc mà khi chương trình tính toán (thành phần ALU tính toán) thì họ sẽ lấy các số hạng, các toán tử, các toán hạng ở trên các thanh ghi. Để hoạt động được, chương trình phải load các biến từ trong vùng nhớ vào thanh ghi.

Với các biến có thể chúng ta cần đến **tốc độ nhanh hơn** thay vì việc phải load từ thanh ghi, chúng ta sẽ phải khai báo bằng từ khoá **register** để biến này sẽ được cấp phát trực tiếp đến các thanh ghi và nó sẽ tăng tốc độ tính toán của biến đó lên.

Từ khoá register thường sẽ ko được sử dụng nhiều trong các driver trong nhúng nhưng vẫn có và rất hay được sử dụng ở?

Từ khoá **volatile**, với từ khoá này thể hiện rõ khi 1 biến thường được trình biên dịch tối ưu hoá (ở bài optimize).

Nếu như biến đó mà vì 1 lý do nào đó, khi chúng ta lập trình, với các lập trình trên hệ thống, support từng task, **nhiều luồng** hoặc có các **chương trình ngắt** trong hệ thống, với các **biến toàn cục** này có thể sử dụng trong các chương trình ngắt hoặc là trong các task khác nhau của chương trình lớn thì việc biến này sẽ bị thay đổi bất cứ lúc nào trong chương trình, khi có ngắt hoặc khi có 1 task trích xuất đến nó nhưng 1 trình biên dịch thường ko nhận biết được phần này cho nên nếu ko khai báo từ khoá volatile, **1 lúc nào đấy, biến này đã bị thay đổi** trong các task khác hoặc trong ngắt nhưng **chương trình sẽ không cập nhật giá trị cho biến này**.

→Như vậy sẽ dẫn đến sự **sai sót giá trị của biến toàn cục** này. Để khắc phục tình trạng như vậy người ta sẽ sử dụng từ khoá **volatile**.

Với từ khoá volatile này, trước khi biến đó được sử dụng trong chương trình thì hệ thống sẽ cập nhật lại giá trị của nó, bỏ qua phần tối ưu hoá trình biên dịch.

Các user case của các biến kết hợp các từ khoá này

Về sau sẽ gặp.

Kiểu dữ liệu có cấu trúc trong C

Về C có các kiểu dữ liệu nguyên thuỷ, đó là int, float, double, char, void thế nhưng các bài toán trên thực tế lại thường thường sẽ ko quy ngay về các kiểu dữ liệu cơ bản trong C được.

Ví dụ bài toán quản lý sinh viên của 1 lớp: việc mà dữ liệu đầu vào đó chính là sinh viên, các thành phần của sinh viên ví dụ như tên tuổi, điểm số, quê quán, giới tính, lúc đó mới là các giá trị có thể quy về.

Để dễ dàng quản lý các kiểu dữ liệu thực tế trong ngôn ngữ lập trình C cung cấp cho chúng ta 1 kiểu dữ liệu đó là **kiểu dữ liệu có cấu trúc**, với kiểu dữ liệu có cấu trúc này có 3 cấu trúc hay sử dụng: **structure variable, union và enum**.

Với kiểu dữ liệu có cấu trúc được định nghĩa như sau:

Một cấu trúc bao gồm các thành phần dữ liệu, trong đó các thành phần dữ liệu này có thể là **ko cần cùng kiểu dữ liệu** cơ bản với nhau và chúng ta có thể group chúng lại thành một group.

Các kiểu dữ liệu có cấu trúc này thì chúng ta có 1 cấu trúc, ví dụ như là cấu trúc về 1 quyển sách, có thể là tên của quyển sách, tác giả của quyển sách và số lần tái bản của quyển sách.

Đây là cấu trúc mà các kiểu dữ liệu mà được định nghĩa quay về các kiểu dữ liệu cơ bản, đó là 1 mảng ký tự, chính là tên của 1 quyển sách, 1 mảng ký tự tiếp theo lưu trữ tên tác giả của quyển sách và 1 phần tử tiếp theo là số lần tái bản thì có kiểu dữ liệu là kiểu int.

Để định nghĩa 1 kiểu dữ liệu có cấu trúc mới gồm khai báo có 2 thành phần

```
Struct cat
{
        Char bk_name [25];
        Char author[20];
        Int edn;/*biến kiểu int để define số lần tái bản*/
        Float price;
};
```

Từ khoá **struct**, **cat** là tên của kiểu dữ liệu mình bắt đầu định nghĩa mới, {} là để đóng gói các phần tử của chúng ta (struct element or struct member)

Khi khai báo kiểu dữ liệu có cấu trúc này thì chúng ta dùng từ struct

Khi define kiểu dữ liệu struct như trên thì hệ thống chưa cấp phát bộ nhớ, lúc này ta đã có kiểu dữ liệu có cấu trúc là kiểu cat.

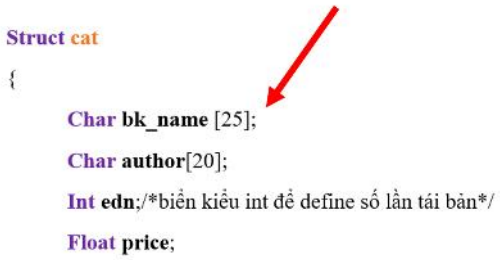
Khi kiểu cat được khai báo là struct cat book1; thì hệ thống sẽ cấp phát cho chúng ta 1 vùng nhớ dùng để lưu trữ các thành phần của kiểu dữ liệu có cấu trúc của chúng ta, các thành phần này được cấp phát liên tiếp nhau.

```
Struct cat book1, book2;
Struct cat book1;
```

Struct cat book2;

Truy xuất đến từng thành phần của struct ntn sử dụng dấu (.) và chúng ta có

[Struct_name].[element_name]



Stru **ct_name** là tên của biến có kiểu dữ liệu được khai báo với kiểu dữ liệu có cấu trúc, ví dụ đã khai báo **struct cat book1**, để truy xuất đến từng thành phần của book1 chúng ta sẽ sử dụng đó là **book1**.[bk_name].

[bk_name] là 1 trường trong kiểu dữ liệu mà cat được khai báo phía trên.

Khi ta truy xuất đến các thành phần của 1 kiểu dữ liệu có cấu trúc (sử dụng toán tử "**.**" hoặc toán tử this "->"), lúc này các thành phần của kiểu dữ liệu có cấu trúc chúng ta đã có các kiểu dữ liệu cơ bản, lúc này ta áp dụng các quy tắc nhập xuất, các quy tắc dẫn xuất tương tự như kiểu dữ liệu cơ bản đã được học ở phần 1.

Size của kiểu dữ liệu có cấu trúc thì bằng bao nhiêu?

Khi 1 cấu trúc được khai báo và 1 biến được khai báo thì các thành phần của biến đó sẽ được cấp phát các vùng nhớ liên tiếp nhau, như vậy thì tổng size của cả 1 cấu trúc 1 biến của kiểu dữ liệu có cấu trúc có giá trị là bao nhiêu, có phải là tổng số byte được cấp phát cho các thành phần trong biến hay không.

Sử dụng Typedef

Phần 2 là ngoài việc định nghĩa 1 kiểu dữ liệu có cấu trúc mới thì ngoài ra cũng có sử dụng type define để định nghĩa 1 kiểu dữ liệu mới thay vì các kiểu dữ liệu cơ bản hay kiểu dữ liệu có cấu trúc

Typedef type name;

Typedef float deci;

Sau này khi muốn sử dụng kiểu dữ liệu deci thì chúng ta chỉ **sử dụng kiểu dữ liệu deci để khai báo như biến float.**

Kiểu dữ liệu có cấu trúc có size là bao nhiêu? (đó là với struct nhưng union thì khác)

Data structure alignment là gì?

Khi nào thì sử dụng các kiểu dữ liệu có cấu trúc?

Kiểu dữ liệu có cấu trúc union:

Nội dung tìm hiểu:

Các biến UNION. Kiểu union là gì? Define union ra sao?	Khai báo biến union thế nào? Size of union là gì? Khi nào thì sử dụng các biến union này?

1 biến **union** là kiểu dữ liệu đặc biệt trong ngôn ngữ C giúp chúng ta lưu trữ các data type khác nhau trong cùng 1 vùng nhớ với các kiểu dữ liệu khác nhau.

Mục đích: chúng ta có thể sử dụng định nghĩa các biến **union** này với nhiều thành phần khác nhau như kiểu dữ liệu có cấu trúc nhưng chỉ 1 thành phần được lưu trữ và được giữ lại trong 1 khoảng thời gian.

Tại 1 thời điểm chỉ có thể lưu trữ 1 thành phần của biến cấu trúc **union** trên vùng nhớ và **union** sẽ cung cấp 1 cách mà người dùng có thể dùng 1 vùng nhớ cho nhiều nội dung khác nhau. ***(Các thành phần dùng chung 1 vùng nhớ)***

Như đã biết thì kiểu dữ liệu có cấu trúc struct bên trên thì nó cũng có nhiều thành phần nhưng **mỗi thành phần của struct sẽ được lưu trữ ở các vùng nhớ liên tiếp**, nó **riêng biệt về vùng nhớ**, còn union thì sẽ là dùng chung 1 vùng nhớ

Cách define 1 biến **union**

Union [union tag]
{
Member defination;
Member defination;
……
Member defination;
}
[one or more union variables];
	Trong đó
	[union tag]: tên của union
	Ngoài ra có thể khai báo các biến của union ở phía sau.
	Để khai báo biến union chúng ta sử dụng từ khoá
	Union <union tag> <union variables>;

[union tag] hay là **[union name**] là tên của union variable

Quay lại vấn đề: struct và union khác nhau ở chỗ nào?

Bản chất của biến union là gì?

Biến union là biến lưu trữ các thành phần của chương trình trên cùng 1 vùng nhớ.

→**Size của union** chính là size của phần tử có độ lớn nhất trong khai báo union đấy.

Giả sử 1 union có kiểu char kiểu int, kiểu float và kiểu double. Size của union đấy bằng size của phần tử lớn nhất đó chính là phần tử có kiểu double, được cấp phát là 8 byte.

Tương tự như kiểu dữ liệu có cấu trúc, chúng ta truy xuất đến từng thành phần của union chúng ta cũng sẽ sử dụng dấu '.'

Ví dụ khi sử dụng biến **union**

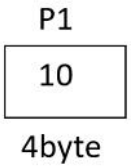
```
#include <stdio.h>
#include <string.h>
#include <math.h>

Union Point
{
        Int a;
        Int b;
};
```

```
Int main()
{
        Float distance, t1, t2;
        Union Point p1,p2;
        P1.a = 10;
        P1.b=20;
        P2.a=30;
        P2.b = 40;
        T1 = pow((p2.a-p1.a),2);
        T2 = pow((p2.b-p1.b),2);
        Distance = sqrt(t1-t2);
        Printf("distance : %4.2f",distance);
        Return 0;
}
```

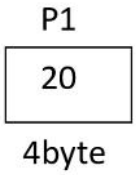
Chúng ta khai báo biến **union** có **union tag** là Point (name), gọi là kiểu Point, có các kiểu int a và int b

Với kiểu **Point** này khi chúng ta khai báo union P1, P2 thì P1 được cấp phát là kiểu int là 4 byte và P2 được cấp phát với kiểu int là 4 byte.

Tại thời điểm đầu tiên, P1.a = 10, **vùng nhớ của union có giá trị bằng 10** và nó đang lưu trữ giá trị của phần tử a



tiếp theo chúng ta có P1.b =20, lúc này tại vùng nhớ của P1 sẽ lưu trữ giá trị của member b.



Chúng ta có P2.a = 30, P2.b = 40, lúc này P2 sẽ lưu trữ giá trị của biến b và khi chúng ta tính giá trị pow …. Và chúng ta tính distance ra nó sẽ in ra giá trị 28.28.

Nhìn vào P2.a – P1.a = 30 – 20 = 0 hay P2.b – P1.b =40 - 20. Lúc này P1.a này vẫn truy xuất được (chỉ vào p2.a) nhưng giá trị của nó ko phải bằng 10 mà nó chính bằng 20 bởi vì chúng ta đang **lưu trữ biến b phía sau khi cấp phát biến a rồi.**

Tương tự P2.b cũng thế nó cũng sẽ lưu trữ biến b chứ ko lưu trữ của biến a nữa

Kiểu enum

Enum là gì? define của enum ra làm sao? khai báo của enum như thế nào?	size của enum bằng bn các user case của enum

Enum là 1 thành phần, 1 kiểu giúp cho người dùng có thể define trong ngôn ngữ C, nó sẽ giúp chúng ta assign 1 cái name trở thành 1 constant.

```
int zone3 = 0;
/*Vừa điều khiển lái xe, vừa nhận tín hiệu từ ESP*/
void TaskDrive  (void *pvParameters )
{
  int turn_right = 1, turn_left = 2;
  int bienluu = 0;
  uint32_t ulNotifiedValue;
  enum States{FORWARD,REVERSE,TURN,STOP};
  States state=FORWARD;
  for(;;)
  {
    switch(state)
    {
      case FORWARD:
```

Giúp cho chương trình của chúng ta có thể dễ dàng đọc được chương trình ấy hơn, dễ maintain hơn, chương trình dùng enum để tránh **hard code** trong C.

(Một hardcode là một phần của một chương trình máy tính mà không thể được thay đổi trong bất kỳ cách nào ngoại trừ bằng cách thay đổi mã nguồn của chương trình riêng của mình.)

Ví dụ về việc khai báo 1 enum

/*! @brief GPIO direction definition*/

Typedef enum _Gpio_pin_direction

{

kGpioDigitalInput =0U,

kGpioDigitalOutput =1U

}

Gpio_pin_direction_t;

Ngoài ra ở đây cũng **dùng từ khoá typedef có nghĩa là chúng ta đã định nghĩa 1 kiểu enum mới có nghĩa là Gpio_pin_direction_t.**

Thì với phần này, giả sử 1 giá trị input hoặc output chẳng hạn thì thường trong thanh ghi họ sẽ quy định là giá trị 0 hay giá trị 1 thôi. Như vậy khi lập trình chúng ta ghi giá trị 0 hay giá trị 1 thì nó rất là khó hiểu và cần đọc lại thanh ghi để xem nó là 0 hay là 1, 1 là out hay 0 là out cho nên chúng ta định nghĩa luôn:

kGpioDigitalInput =0U,

kGpioDigitalOutput =1U

chúng ta có thể sử dụng từ khoá kGpioDigitalInput =0U thay cho việc sử dụng số 1 và 0

enum week (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday)
int main()
{
enum week today;
today = Friday;
printf(“day %d”,today+1);
return 0;
}
	Nếu như trong 1 trường enum chúng ta ko define rõ ràng giá trị của nó thì nó sẽ tự động assign giá trị cho các member và các constant bắt đầu từ 0 về sau.
	
	Khi khai báo **enum week today;**
	week là tên của enum, biến enum có tên là today, giả sử chúng ta cho today = Friday

<u>Bonus</u>

GPIO CLR_PDDR (base, 1U << pin); đưa giá trị 1 vào thanh ghi GPIO CLR_PDDR

Casting - ép kiểu trong C, ta có các thành phần khác nhau

Type casting, tất cả các object trong C đều có kiểu dữ liệu cơ bản (char, int, float, double), các kiểu con trỏ cơ bản (char, int, float, double)

Trong các biểu thức có thể sử dụng nhiều kiểu dữ liệu cơ bản khác nhau nhưng ở trong c ta sẽ thấy là C thì họ phải làm thế nào để các biểu thức của mình có thể tính toán được với các kiểu khác nhau như vậy thì ở đây chúng ta có 1 kiến thức được gọi là ép kiểu.

Có 2 loại C **automatic ép kiểu** (hệ thống khi biên dịch sẽ tự ép kiểu cho mình) và **người dùng ép kiểu**, tức là trong code, mình chủ đôkng ép kiểu trong biểu thức của mình.

Các quy tắc để ép kiểu

Int/int thì kết quả là int 2/4 = 0

Float/float thì kết quả là float 2.0/4.0=0.5

Ép kiểu với các biểu thức có kiểu dữ liệu khác nhau thì nó sẽ tang level của kiểu dữ liệu ấy lên

Char < int < long < float < double

Float /int thì kiểu int sẽ tự động change sang kiểu float float/float =float

2.0 /4= 0.5

Ép kiểu người dùng

Các biểu thức gán như sau

Tăng level biểu thức bên phải lên hoặc là giảm level đi theo kiểu dữ liệu bên trái

Ví dụ int i

Float f=1.23

i = f → f được ép kiểu tạm thời là kiểu int và I có giá trị bằng 1

f = i → biểu thức bên phải sẽ tạm thời được coi là kiểu dữ liệu float bằng kiểu dữ liệu bên trái

Với cách ép kiểu tự động như thế này, đôi khi chúng ta giá trị thực bị sai sót, nhầm lẫn

Int I =3

Float f

Ép kiểu f = i thì i sẽ bị mặc định ép kiểu float, do float sau dấu . có độ chính xác là 6 chữ số nên suy ra có thể I thay đổi còn 2.999995.

Ép kiểu bởi người dùng

Có cấu trúc kiểu ()

Int x =1, x= 2

Float f1 = x1/x2; x1 /x2 là kiểu int/int -> kết quả kiểu int, sau đó kết quả này được tạm thời hiểu là kiểu float

X1 = 1; x2 = 2

½ = 0

Sau đó 0 từ int sang float là 0.0

Float f2 = (float)x1/x2 -> f2 = 0.5, riêng x1 được ép kiểu float, x2 bị nâng 1 level lên

Float f3 = (float)(x1/x2); biểu thức 2 kiểu int/int kết quả trả về kiểu int rồi bị ép sang float