

Print value of b which is equal a variable below on line 5

Print address of a variable below on line 4

Declare a pointer variable called iPtr pointing to an int

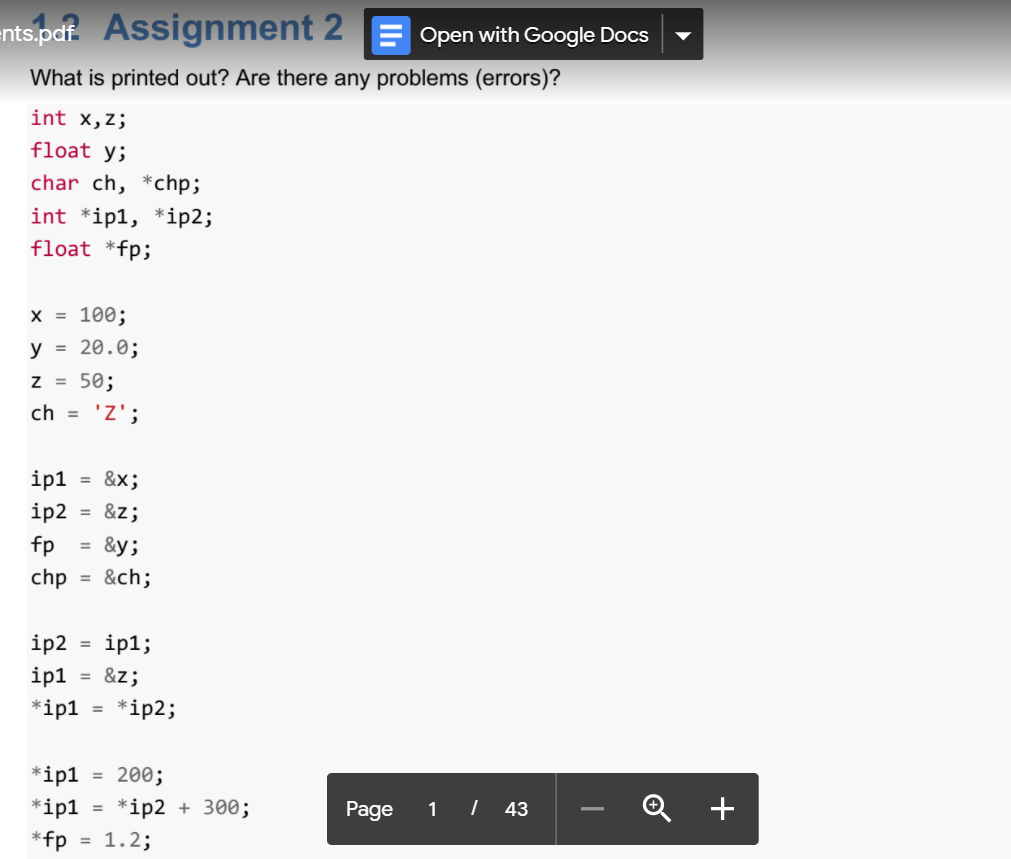
Int pointer pointing to an int value

Print Address of a variable on line 9

Print the value of a variable: 3 on line 8

Print address of b variable on line 6

**Above statement is valid and will compile successfully. There is no error**



The value of pointer ip1 is equal to that of ip2 (=x=100)

\*ip2=100; plus 300 equal 400. \*ip1=400

Re-assign a new number to the pointer

Ip1 pointer pointing to the address of z variable

(The address) Ip2 pointer pointing to ip1 pointer (=&x)

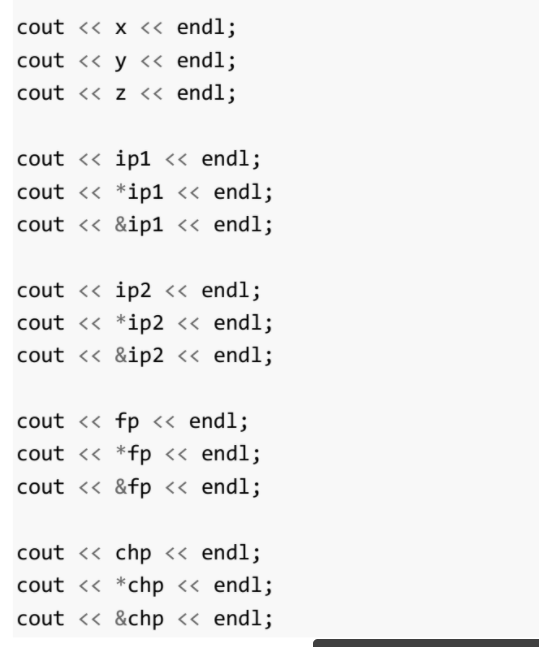
You can change the address stored in a pointer

Int pointer pointing to an int value

Float pointer pointing to an float value

We assign values to variables

Declare a pointer variable called ip1, ip2 as a pointer of type



Print the address of fp variable

Print the address of ch variable

Print the value of the chp (‘Z’)

Print the address of the chp

100 I explained as above

Print the address of z cz ip1=&z;

Print the value of ip1 variable (400)

1.2

100

1.2

Print the address of y variable

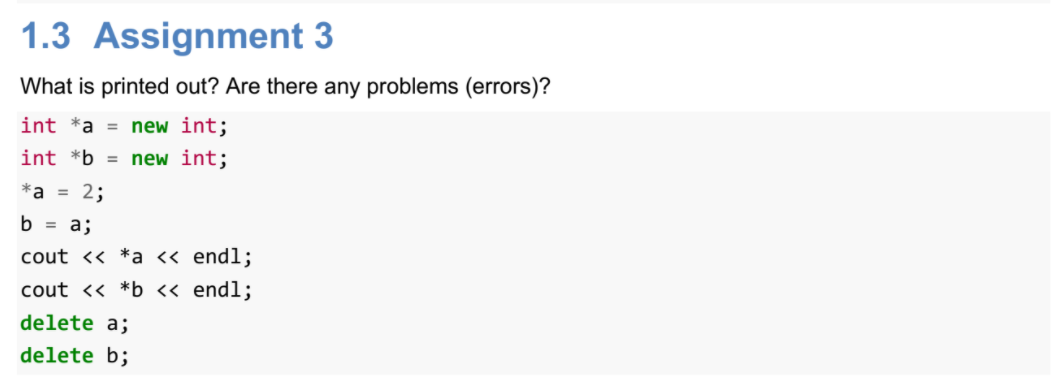
Print the address of ip2 variable

Print the address of ip2 which is pointing to the address of variable x (&x)

Print the address of ip1 variable

=400=\*ip1 because ip1=&z;

There is no error!



return the memory to the operating system.

Error: Vẫn chạy được nhưng có lỗi quản lý vùng nhớ do vùng nhớ mà b tạo ra tại dòng lệnh : int \*b = new int;

không có con trỏ nào nắm giữ vì sau đó b giữ giá trị của a.

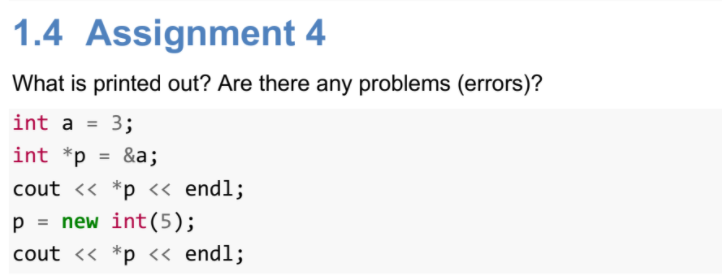
Fix: Xoá dòng cuối

put a value in that memory location

In ra giá trị bên trong địa chỉ mà con trỏ a đang giữ (=>2),

Giá trị của con trỏ b là địa chỉ của vùng nhớ mà con trỏ a nắm giữ, được gán tại dòng lệnh : b = a;

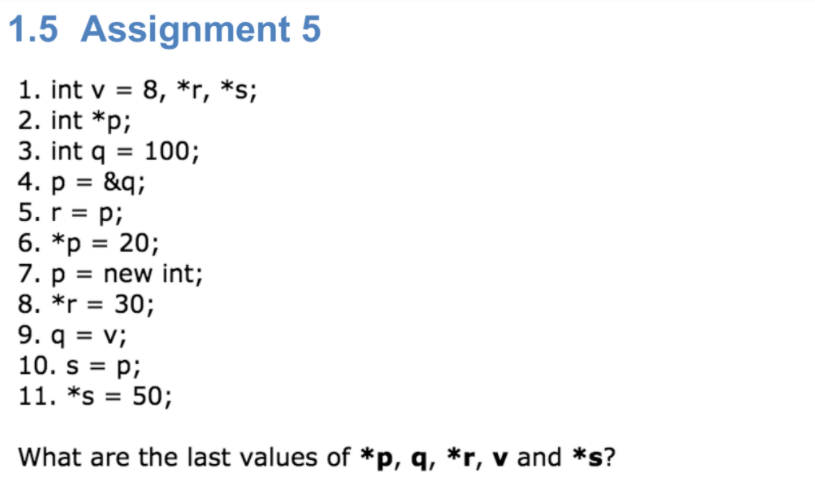
Dynamically allocate an int and assign its address to pointer



In ra giá trị 5 vì p được gán với giá trị khởi tạo ngay khi khai  
báo tại dòng p=new int(5)

In ra giá trị lưu trữ tại địa chỉ con trỏ trỏ tới, tức là 3

Trỏ con trỏ p đến địa chỉ của biến a



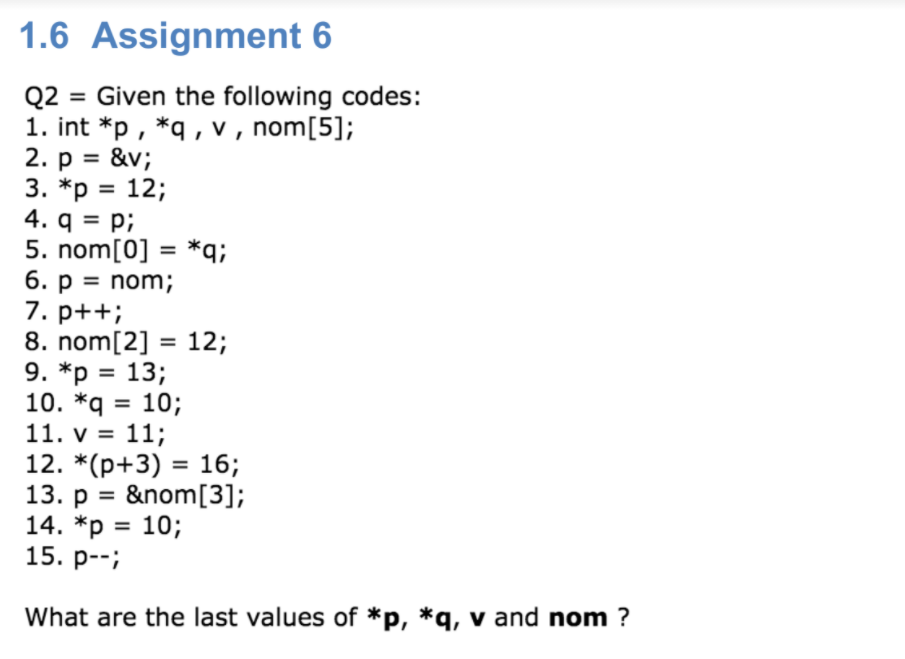
1. Dòng 7: p= new int => \*p là giá trị rác vì con trỏ p trỏ tới một vùng nhớ chưa được khởi tạo

2. q = 8 vì tại dòng 9: q=v mà v=8

3. \*r = 30 vì dòng 8: \*r=30

4. v=8 vì v được khởi tạo bằng 8 mà các lệnh tiếp theo không liên quan đến v

5. \*s = 50 vì dòng lệnh cuối cùng tác động đến \*s=50



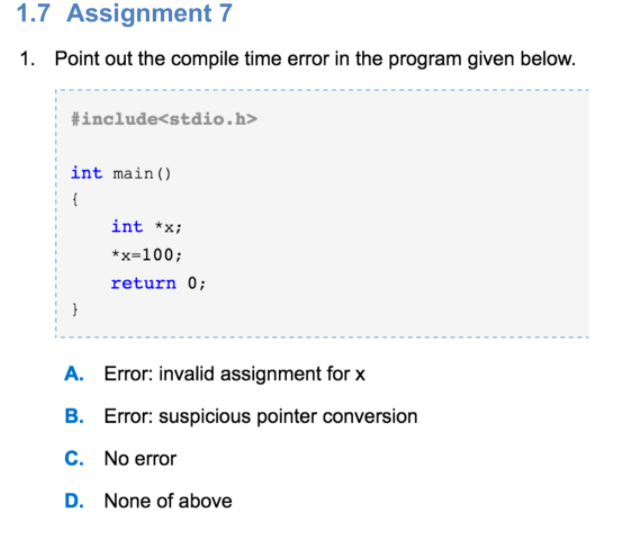
1. \*p = 12 vì tại dòng 13 (p=&now[3]) và dòng 15 (p--) thì p trỏ đến địa chỉ nom[2] mà nom[2]=12 =>\*p=12

2. \*q = 11 vì tại dòng 4 (q=p) mà p giữ vị trí của v tại dòng 2. Mà v=11.

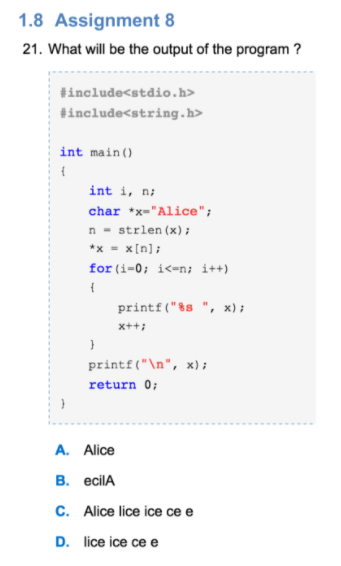
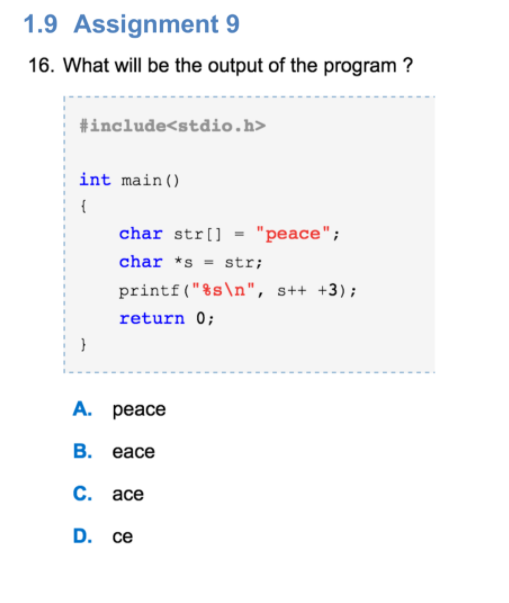
=> \*q=11

3. v=11 tại dòng thứ 11

4. nom là địa chỉ unknown đang chứa mảng nom



Đáp án A. Vì x chưa trỏ vào vùng nhớ nào để lưu giá trị   
mà đã gán =100.

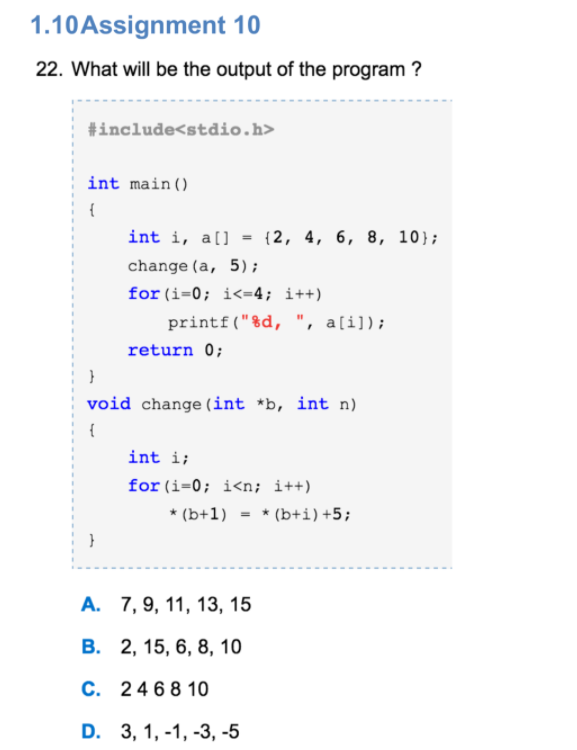


Không có đáp án vì vùng nhớ lưu trữ chuỗi kí tự "Alice" là vùng nhớ hằng, giá trị bên trong vùng nhớ này không thể thay đổi, trong khi đó lệnh \*x = x[n]; cố gắng thay đổi giá trị bên trong vùng nhớ hằng.

Không có cái gì được in ra trên console

Câu D

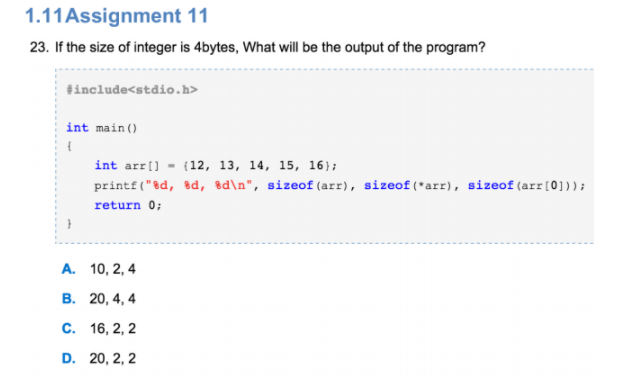
Vì con trỏ s được trỏ tới vị trí của chuỗi str, nên \*s = str[0], khi gọi lệnh printf(“%s”,s) sẽ được chuỗi giống chuỗi str, nên ở dòng lệnh printf(“%s\n”,s++ +3) đã làm con trỏ s tăng lên 3 ô nhớ bằng với str[3], nên sẽ in ra chuỗi str kể từ vị trí str[3].



Chọn B

Vì hàm change làm thay đổi giá trị của a[1] vì mảng a được truyền vào hàm và dòng lệnh tác động đến phần tử a[1] là: \*(b + 1) = \*(b + i) + 5

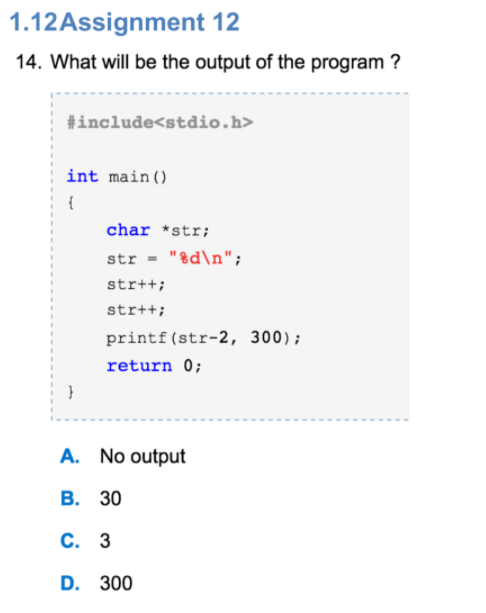
tương đương với b[1] = b[i] + 5; và tới khi vòng lặp kết thúc ta có b[1] = b[4] + 5 ; suy ra b[1] = 15. Và mảng b tương đương với mảng a vì mảng a được truyền vào hàm nên a[1] cũng bằng 15. Vậy nên ta có kết quả in ra console là: 2, 15, 6, 8, 10.



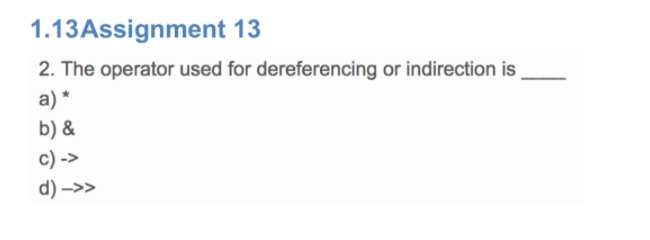
Chọn B

Vì hàm change làm thay đổi giá trị của a[1] vì mảng a được truyền vào hàm và dòng lệnh tác động đến phần tử a[1] là: \*(b + 1) = \*(b + i) + 5

tương đương với b[1] = b[i] + 5; và tới khi vòng lặp kết thúc ta có b[1] = b[4] + 5 ; suy ra b[1] = 15. Và mảng b tương đương với mảng a vì mảng a được truyền vào hàm nên a[1] cũng bằng 15. Vậy nên ta có kết quả in ra console là: 2, 15, 6, 8, 10.



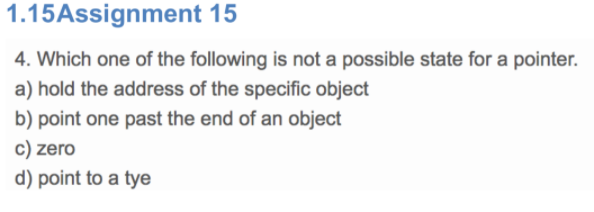
chọn D, vì trong hàm printf chứa tham số là str-2 mà lúc trước ta đã tăng str lên 2 đơn vị nhớ nên sẽ tương đương với vị trí đầu chuỗi, tham số str-2 chứa thẻ định dạng %d được truyền vào số nguyên là 300 tương ứng với tham số thứ 2 của hàm printf.

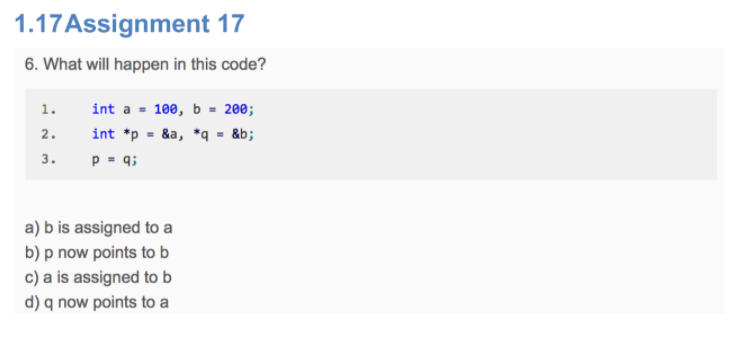


Chọn a, vì toán tử dùng để truy xuất vào giá trị bên trong địa chỉ mà con trỏ nắm giữ là \*

Chọn D

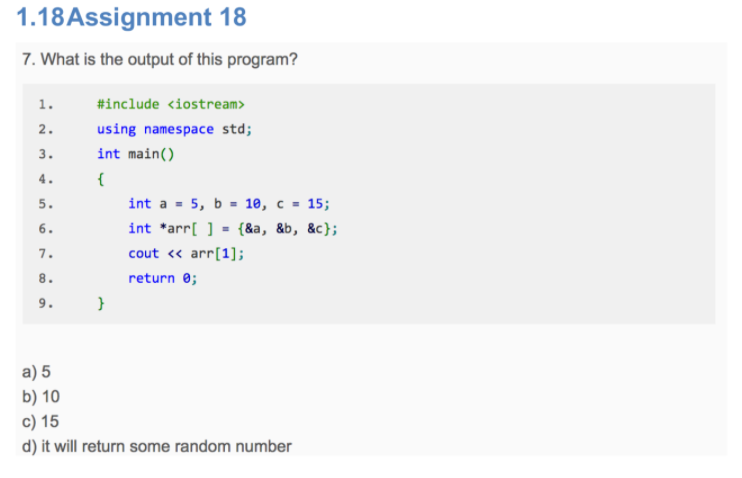
Vì con trỏ không thể trỏ đến 1 kiểu





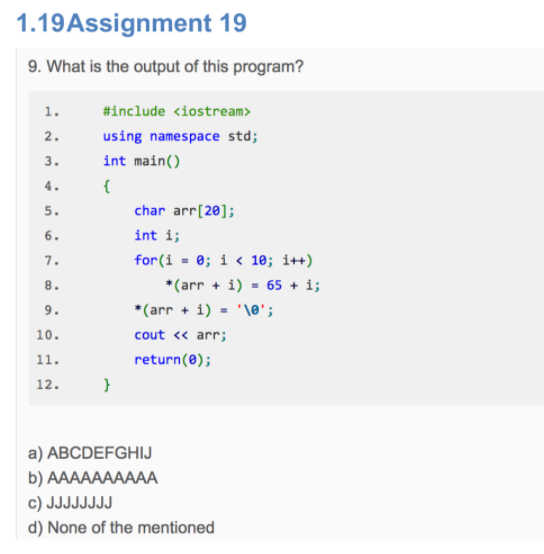
Chọn D

Vì con trỏ không thể trỏ đến 1 kiểu



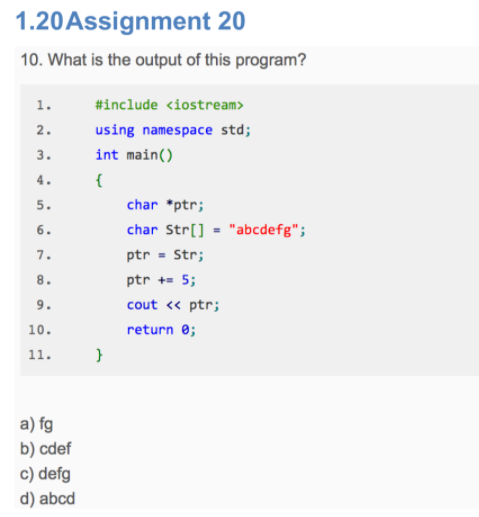
Chọn d

Vì dòng lệnh cout << arr[1]; sẽ in ra địa chỉ của biến b là một giá trị nào đó tương ứng với địa chỉ của biến b.



Chọn a

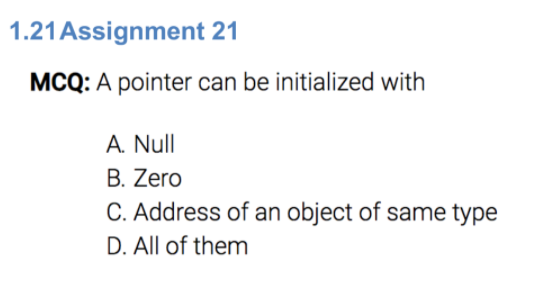
Vì dòng lệnh cout << arr; tương đương với in chuỗi kí tự bên trong arr ra màn hình, mà giá trị của chuỗi này được thiết lập ở các dòng : \*(arr + i) = 65 + i; tương ứng với kí tự thứ 6 trong bảng mã ASCII trở lên, \*(arr + i) = ‘\0’; thiết lập kí tự kết thúc chuỗi để khi cout << arr; ta sẽ được giá trị in ra là kí tự ở vị trí từ đầu chuỗi đến là kí tự ở vị trí kết thúc chuỗi.



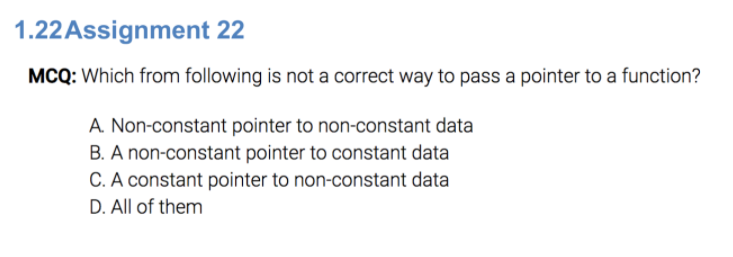
Chọn a

Vì tại dòng lệnh ptr = Str tương đương với trỏ con trỏ ptr đến đầu chuỗi Str. Sau đó dòng lệnh ptr += 5; làm con trỏ này tăng lên 5 ô nhớ sẽ tương ứng với Str[5]. Khi in ptr ra ta sẽ được chuỗi từ Str[5] đến kí tự kết thúc chuỗi tương đương với “fg”.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

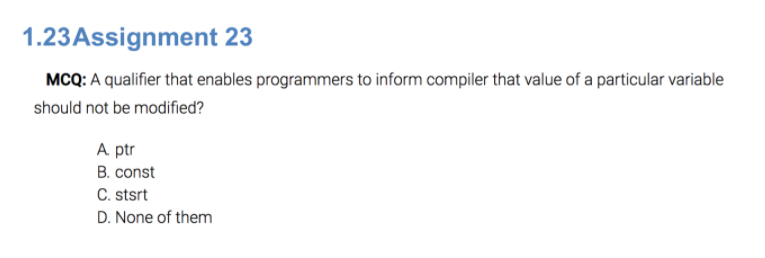


* Chọn D, vì một con trỏ có thể chứa giá trị Null, 0, hoặc vị trí của một đối tượng có kiểu giữ liệu tương ứng.



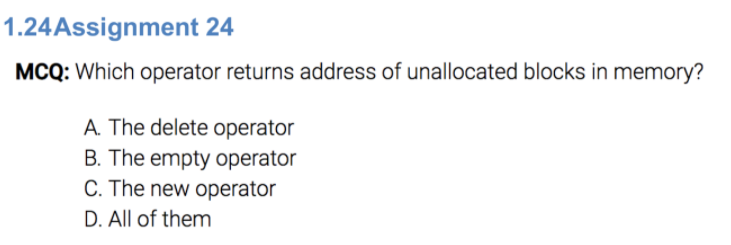
* Chọn B, vì nếu không sử dụng con trỏ hằng để trỏ đến một vùng nhớ là có giá trị là hằng số thì sẽ gây lỗi.

-----------------------------------------------------------------------------------------------------------------------------------------------------------



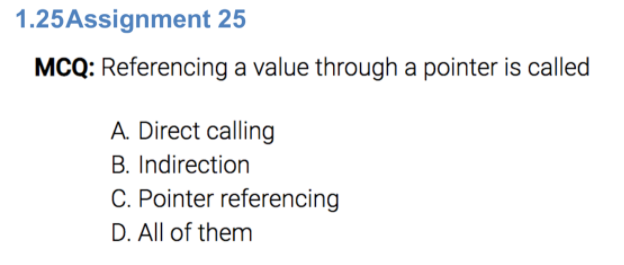
-> Chọn B, thêm const phía trước kiểu dữ liệu lúc khai báo biến sẽ làm cho giá trị biên trong biến đó không thể thay đổi.

-----------------------------------------------------------------------------------------------------------------------------------------------------------



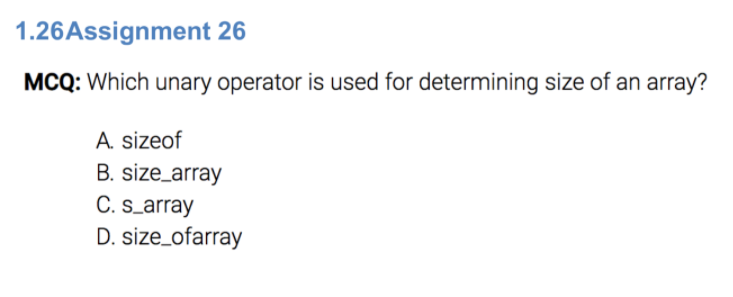
-> chọn C, sử dụng toán tử new để cấp phát vùng nhớ cho con trỏ, kiểu dữ liệu trả về của toán tử này là địa chỉ của vùng nhớ được cấp phát.

-----------------------------------------------------------------------------------------------------------------------------------------------------------



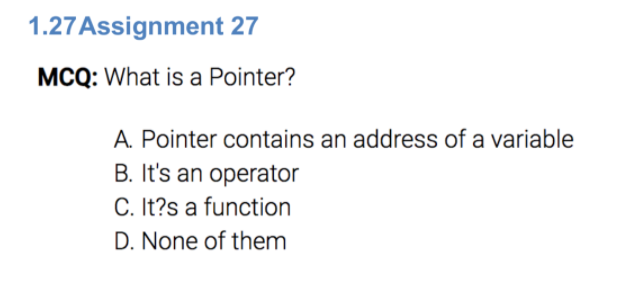
-> chọn B, vì dùng con trỏ để tham chiếu đến một giá trị gọi là Indirection.

-----------------------------------------------------------------------------------------------------------------------------------------------------------



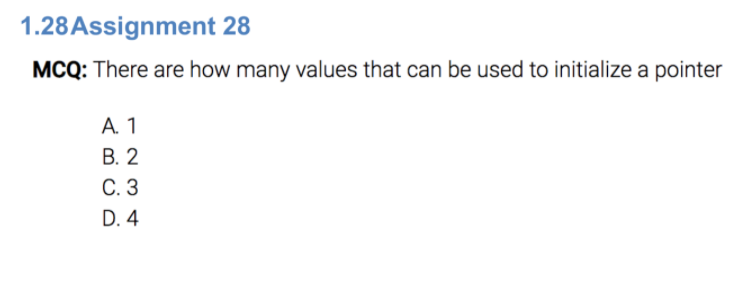
-> chọn a, toán tử sizeof sẽ đưa ra kích thước của tham số truyền vào, khi ta truyền tên mảng vào sẽ đưa ra kích thước tính theo byte của mảng, nếu cần đưa ra kích thước chuẩn ta có: sizeof(arr)/sizeof(arr[0]).

-----------------------------------------------------------------------------------------------------------------------------------------------------------



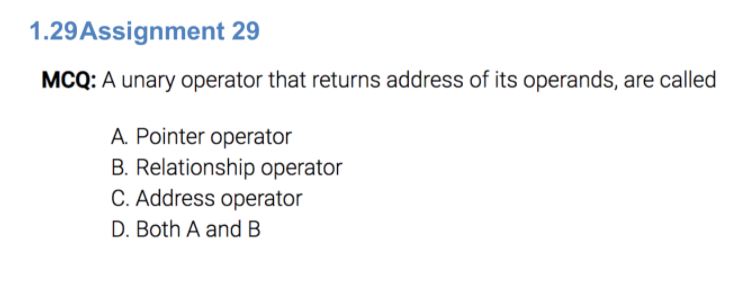
-> chọn A, vì con trỏ lưu trữ địa chỉ của biến.

-----------------------------------------------------------------------------------------------------------------------------------------------------------



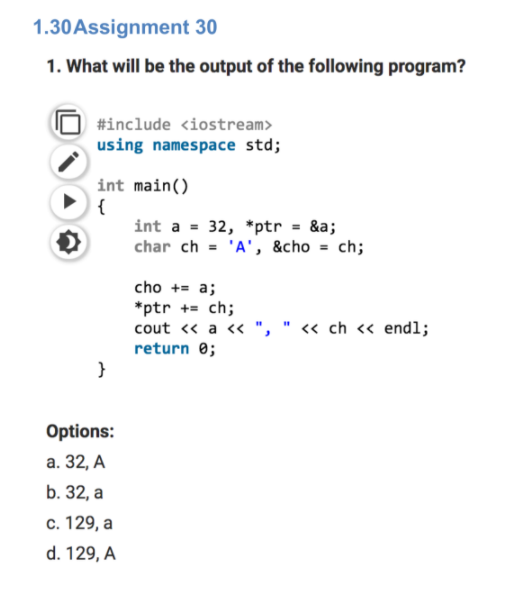
-> chọn A, Khi khởi tạo giá trị cho con trỏ chỉ có thể gán bởi địa chỉ của duy nhất một biến hoặc NULL.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

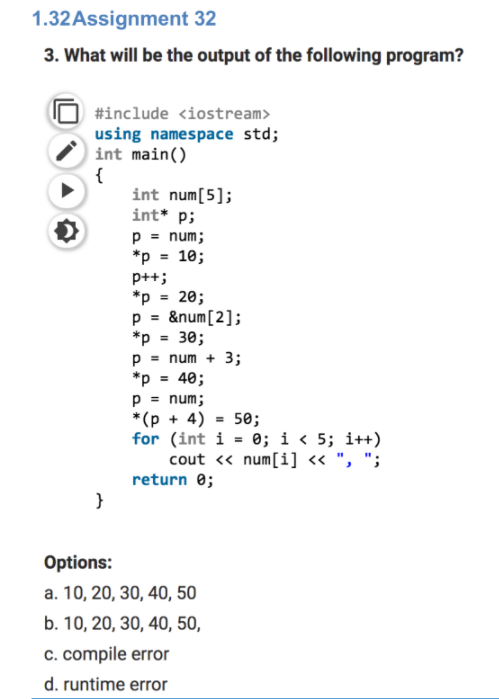


-> chọn C, vì toán tử &(address-of) dùng để lấy địa chỉ.

-----------------------------------------------------------------------------------------------------------------------------------------------------------



-> chọn c, vì: sử dụng biến tham chiếu cho để tham chiếu tới biến ch, vậy dòng lệnh cho += a; tương đương với ch += a; và tương đương với biến ch tăng lên 32 đơn vị. Biến ch chứa kí tự ‘A’ nên lúc tăng lên 32 đơn vị sẽ thành ‘a’. \*ptr trỏ đến a và tại dòng lệnh: \*ptr += ch ta tương đương với a += ch;. Lúc này ta có ch = vị trí của ‘A’ trong bảng mã ASCII + 32 = 65 + 32 = 97. Vậy \*ptr += ch tương đương với \*ptr += 97 và tương đương với a += 97. Vậy a = 127, int(ch) = 97 và ch = ‘a’.  
-----------------------------------------------------------------------------------------------------------------------------------------------------------



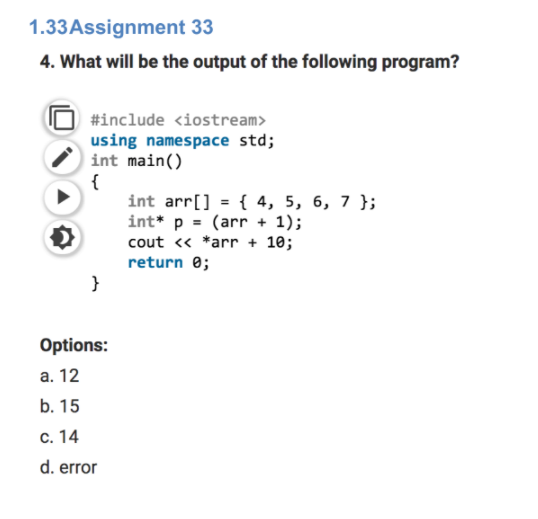
-> chọn b, vì giá trị num[0] được thiết lập tại: p =num; \*p = 10; tương đương với num[0] = 10.

num[1] được thiết lập tại: p++; \*p = 20; tương đương với num[1] = 20 .

num[2] được thiết lập tại: p = &num[2]; \*p = 30; ; tương đương với num[2] = 30.

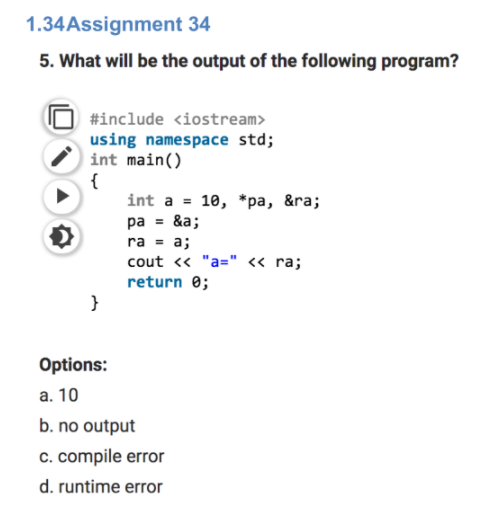
num[3] được thiết lập tại: p = num + 3; \*p = 40; ; tương đương với num[3] = 40.

num[4] được thiết lập tại: p = num; \*(p + 4) = 50; ; tương đương với num[4] = 50.

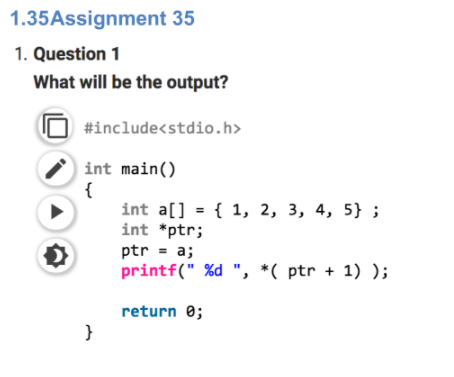


-> chọn c, Vì arr tương ứng với phần tử đầu tiên của mảng nên dòng lệnh: cout << \*arr + 10; sẽ in ra giá trị bằng phần tử đầu tiên của mảng cộng thêm 10 đơn vị.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

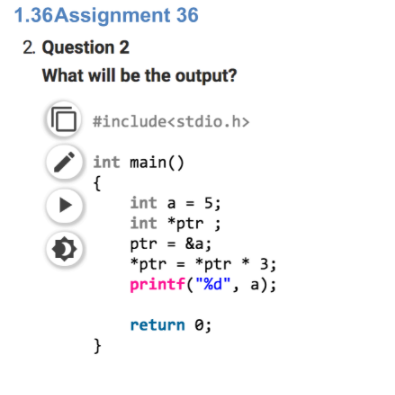


-> Chọn c, vì biến tham chiếu &ra khi khởi tạo phải được gán với một biến nào đó có chung kiểu dữ liệu nếu không sẽ bị lỗi compile error.



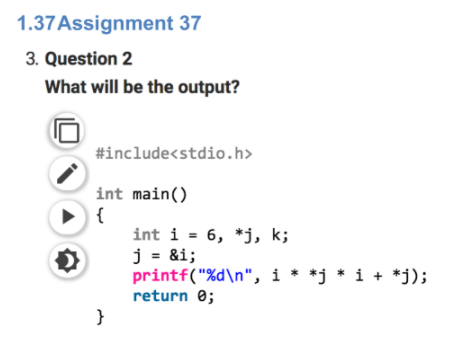
-> Output : “2”. Vì con trỏ ptr trỏ đến mảng a, nên tại dòng lệnh printf(“ %d ”, \*(ptr + 1) ); sẽ in ra phần tử a[1] là 2.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

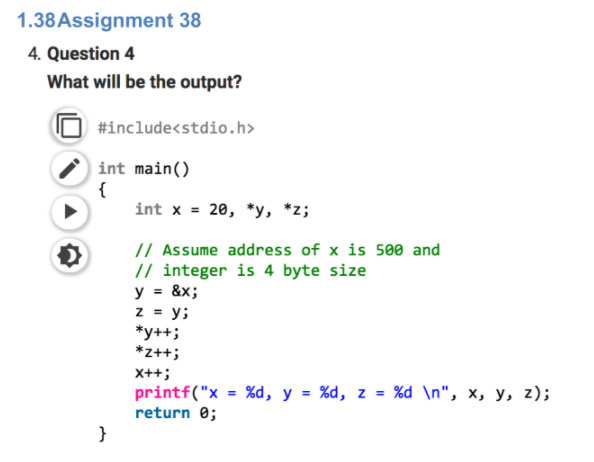


-> Output: “15”. Vì con trỏ ptr trỏ đến biến a, và tại dòng lệnh \*ptr = \*ptr \* 3 đã làm giá trị biến a tăng lên 3 lần tương đương với a= a\* 3.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

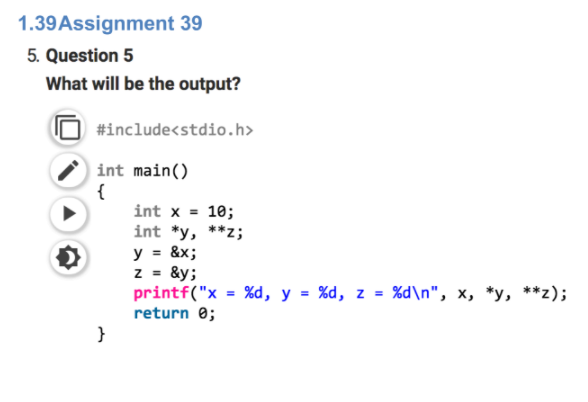
-> Output : “222”. Vì con trỏ j được trỏ đến biến a nên \*j tương đương với a, và dòng lệnh printf( “%d\n”, i \* \*j \* i + \*j); tương đương với printf( “%d\n”, i \* i \* i + i); là: 6\*6\*6 + 6 = 222.

-----------------------------------------------------------------------------------------------------------------------------------------------------------

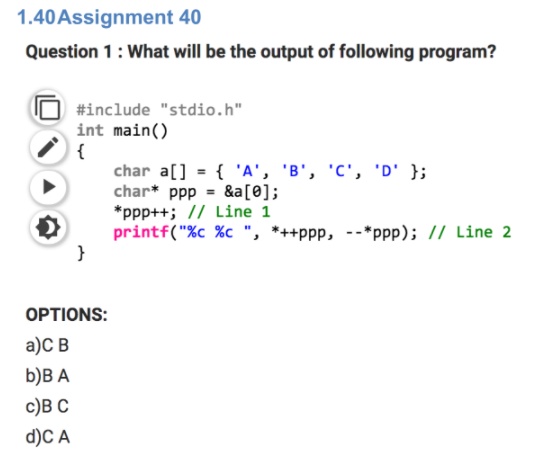


-> Output: “x = 21, y = 504, z = 504 ”. Vì y và z cùng trỏ đến a nên đều có giá trị là 500. Tại dòng lệnh \*y++ sẽ tham chiếu đến ô nhớ có địa chỉ 500 và sau đó tăng giá trị của y lên 1 đơn vị ô nhớ là 4 byte (= 504). . Tại dòng lệnh \*z++ sẽ tham chiếu đến ô nhớ có địa chỉ 500 và sau đó tăng giá trị của z lên 1 đơn vị ô nhớ là 4 byte (= 504). Giá trị của x tăng lên 1 đơn vị ở dòng lệnh x++ ;(=10).

-----------------------------------------------------------------------------------------------------------------------------------------------------------

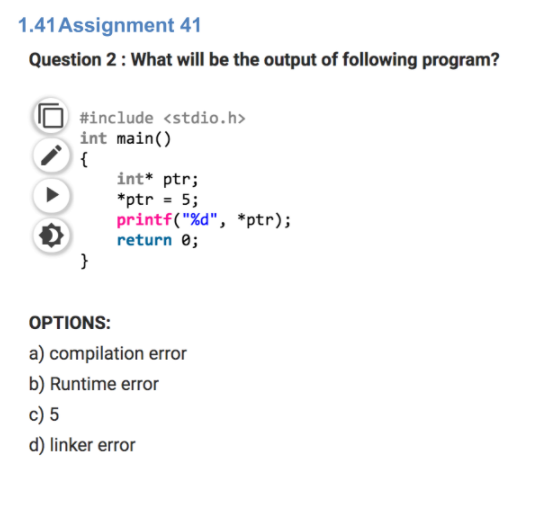


-> Output: “x = 10, y = 10, z = 10 ”. Vì con trỏ cấp 1 y trỏ đến biến x, và con trỏ cấp 2 z trỏ đến con trỏ cấp 1 y nên khi in ra \*y và \*\*z tương đương với in ra biến x.  
-----------------------------------------------------------------------------------------------------------------------------------------------------------

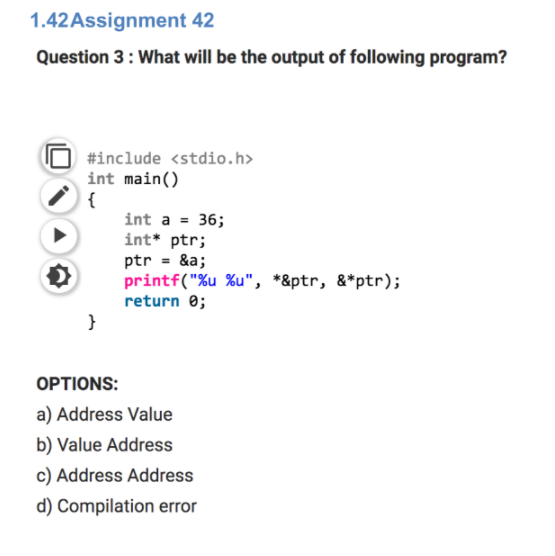


-> Không có đáp án, Output: “C C” . Vì con trỏ ppp trỏ đến mảng a. và tại Line 1 \*ppp++; đã làm tăng con trỏ ppp lên 1 đơn vị ô nhớ tương đương a[1]. Tại Line 1 ta có \*++ppp là tăng con trỏ ppp lên 1 đơn vị ô nhớ sau đó tham chiếu đến địa chỉ mà nó nắm giữ và --\*ppp là tham chiếu đến địa chỉ mà nó nắm giữ và sau đó giảm giá trị tại địa chỉ đó đi 1 đơn vị. Vì độ ưu tiên của hai toán tử \* và ++, -- là khác nhau nên khi in ra ta sẽ có kết quả là “C C”.

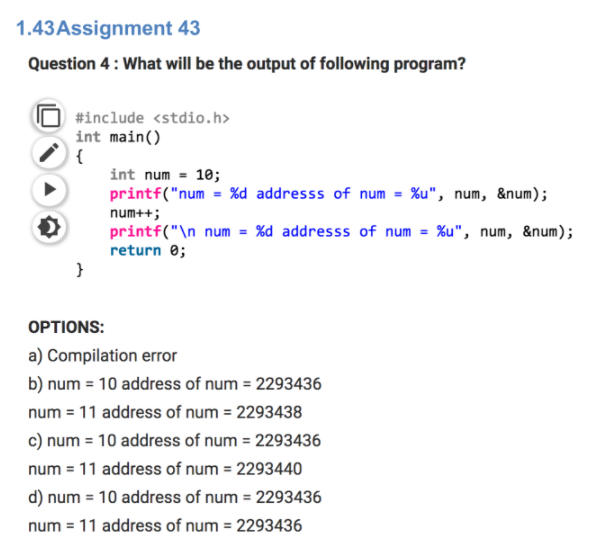
-----------------------------------------------------------------------------------------------------------------------------------------------------------



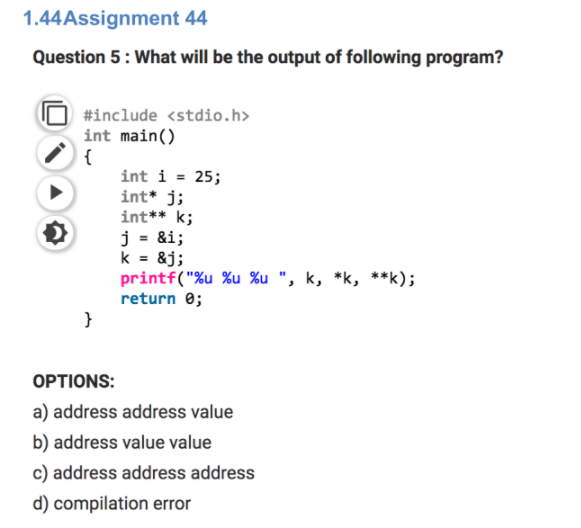
-> Chọn b, vì con trỏ ptr chưa được khởi tạo giá trị nên giá trị bên trong nó đang là địa chỉ rác mà sử dụng toán tử dereference để thay đổi giá trị bên trong địa chỉ mà nó nắm giữ tại dòng lệnh \*ptr = 5; nên sẽ gây ra lỗi Runtime.  
-----------------------------------------------------------------------------------------------------------------------------------------------------------



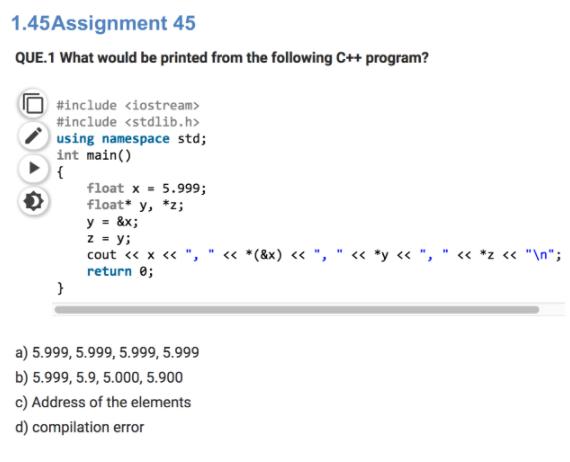
-> Chọn c, vì \*&ptr sẽ tham chiếu đến giá trị của địa chỉ con trỏ ptr, tương đương với giá trị của con trỏ ptr là địa chỉ biến a. &\*ptr tương đương với địa chỉ của giá trị bên trong địa chỉ con trỏ ptr nắm giữ (là biến a) và tương đương với địa chỉ biến a.  
-----------------------------------------------------------------------------------------------------------------------------------------------------------



-> Chọn d, vì khi dùng toán tử &num sẽ in ra địa chỉ của biến num, và dòng lệnh: num++; sẽ tăng giá trị của biến num lên 1 đơn vị (=11) mà không tác động đến địa chỉ của nó.  
-----------------------------------------------------------------------------------------------------------------------------------------------------------

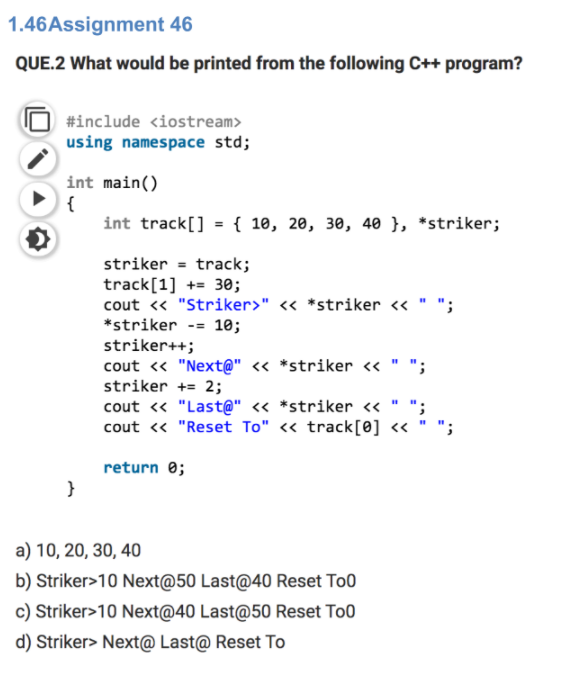


-> Chọn a, vì giá trị của k là địa chỉ của con trỏ cấp 1 j, \*k là giá trị bên trong địa chỉ mà con trỏ cấp 2 k nắm giữ tương đương địa chỉ của biến mà con trỏ cấp 1 j đang giữ địa chỉ (= &i = j), và \*\*k là giá trị bên địa chỉ (\*k) tương đương giá trị trong địa chỉ mà con trỏ cấp 1 j đang giữ (=25).  
-----------------------------------------------------------------------------------------------------------------------------------------------------------



-> chọn a, vì : \*(&x) tham chiếu đến giá trị bên trong địa chỉ của x tương đương giá trị của x.

Vì z = y nên \*z = \*y = x = 5.999.  
-----------------------------------------------------------------------------------------------------------------------------------------------------------.

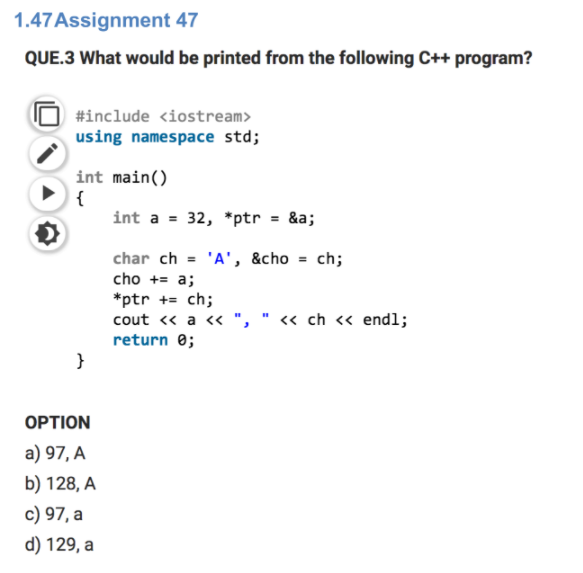


-> Chọn b, vì con trỏ striker trỏ đến mảng track nên \*striker tương đương track[0]. \*striker -= 10 tương đương tracker[0] -= 10, lúc này tracker[0] = 0.

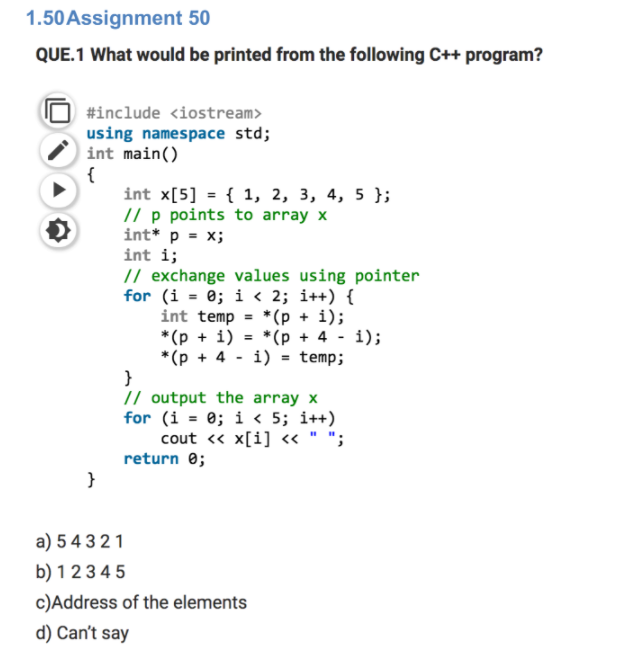
Dòng lệnh striker++ sẽ tăng con trỏ striker lên 1 ô nhớ tương đương với track[1].

Dòng lệnh striker += 2 sẽ tăng con trỏ striker lên 2 ô nhớ tương đương với track[3].

-----------------------------------------------------------------------------------------------------------------------------------------------------------



-> chọn c, vì: sử dụng biến tham chiếu cho để tham chiếu tới biến ch, vậy dòng lệnh cho += a; tương đương với ch += a; và tương đương với biến ch tăng lên 32 đơn vị. Biến ch chứa kí tự ‘A’ nên lúc tăng lên 32 đơn vị sẽ thành ‘a’. \*ptr trỏ đến a và tại dòng lệnh: \*ptr += ch ta tương đương với a += ch;. Lúc này ta có ch = vị trí của ‘A’ trong bảng mã ASCII + 32 = 65 + 32 = 97. Vậy \*ptr += ch tương đương với \*ptr += 97 và tương đương với a += 97. Vậy a = 127, int(ch) = 97 và ch = ‘a’.  
  
-----------------------------------------------------------------------------------------------------------------------------------------------------------



-> Chọn a, vì con trỏ p trỏ đến mảng x và tại các dòng lệnh trong vòng lặp for thứ nhất đã làm thay đổi thứ tự của hai phần tử cuối cùng trong mảng x ra đầu tiên và hai phần tử đầu tiên ra cuối cùng. Do ban đầu mảng x sắp xếp theo chiều tăng dần và số lượng phần tử của mảng x là lẻ nên phần tử ở giữa mảng không bị tác động khi sắp xếp. Sau khi thoát khỏi vòng for đầu tiên ta được mảng sắp xếp theo thứ tự giảm dần.