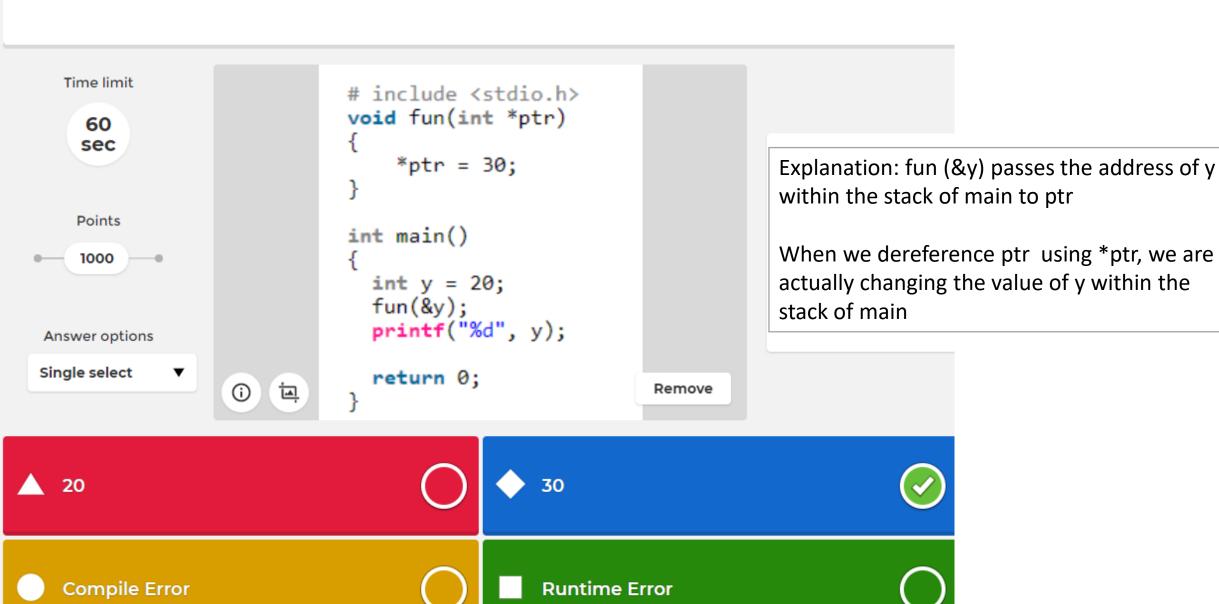
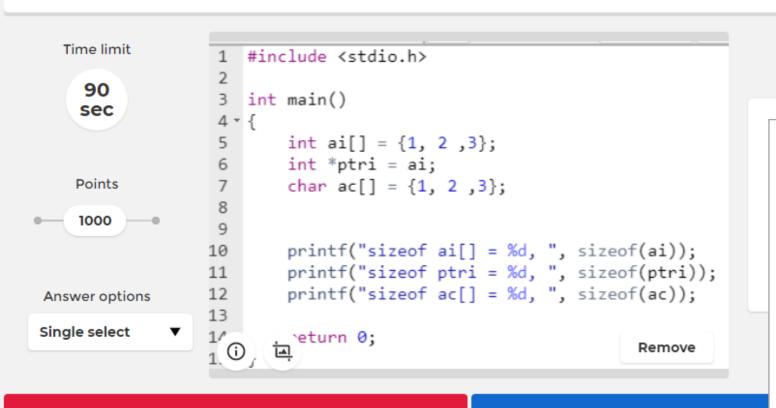
What is the output of the following program?



What is the output? where int takes 4 bytes, char takes 1 byte and pointer takes 4 bytes.



Explanation: size of function returns the size of of whole array in bytes within the function the array is declared.

Once the array is passed into another function array name is reduced to a poiunter.

Since each integer is 4 bytes, 3 integers make 12 bytes

Since each char is 1 byte, 3 chars make 3 bytes

sizeof ai[] = 12, sizeof ptri = 4, sizeof ac[] = 3,



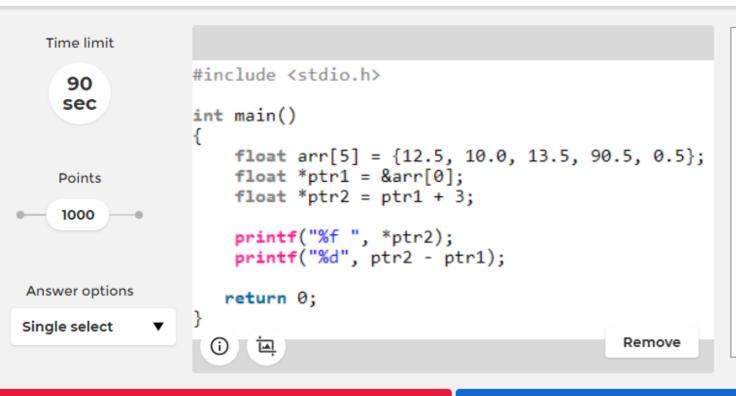
sizeof ai[] = 12, sizeof ptri = ac[] = 12,

sizeof ai[] = 3, sizeof ptri = 4, sizeof ac[] = 3,



sizeof ai[] = 12, sizeof ptri = 12, sizeof ac[] = 3,

Assume that float takes 4 bytes, What is the output of following program.



Explanation:

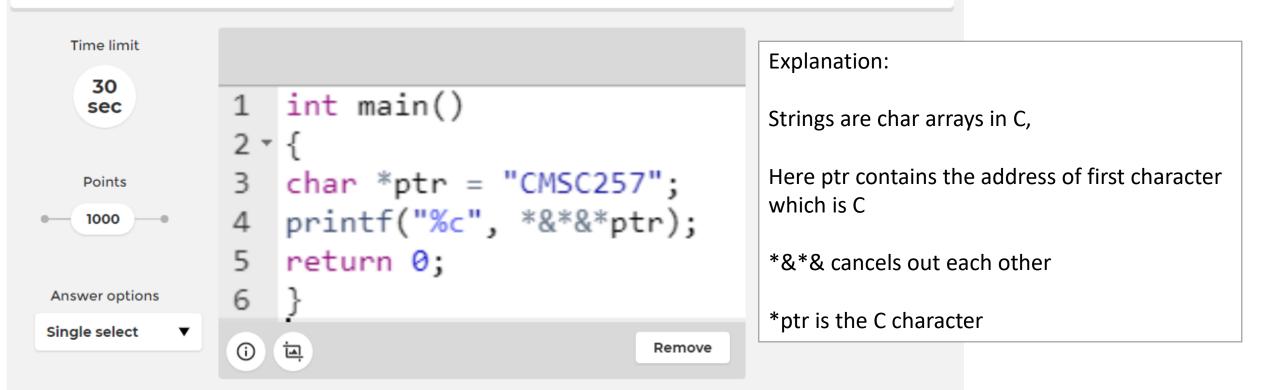
ptr1 holds the address of arr[0] ptr2 holds the address of arr[3], this is pointer arithmetic, refer to the lectures

When we print dereferenced ptr2, arr[3] will; be printed

Ptr2-ptr1 is again pointer arithmetic, it is not size in bytes, refer to the lectures.



What is the output?





What is the output?

Time limit 90 sec Points 1000 **Answer options** Single select

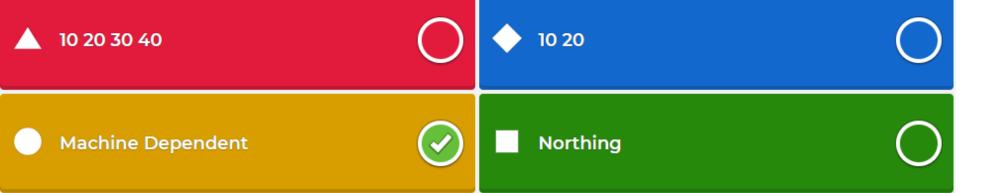
```
#include<stdio.h>
void fun(int arr[])
{
   int i;
   int arr_size = sizeof(arr)/sizeof(arr[0]);
   for (i = 0; i < arr_size; i++)
        printf("%d ", arr[i]);
}

int main()
{
   int i;
   int arr[4] = {10, 20, 30, 40};
   fun(arr);
   reform 0;
   Remove</pre>
```

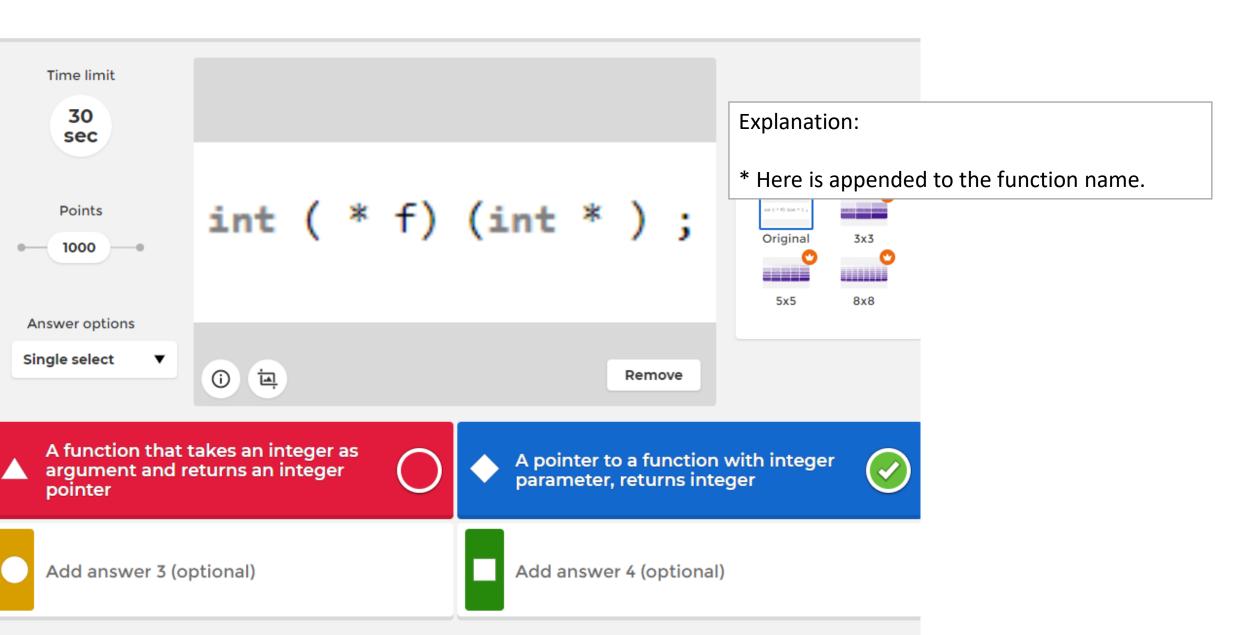
Explanation:

sizeof(arr) is dependent on the machine. For 32 bit machines it is 4 bytes for 64 bit machines it is 8 bytes.

arr reduces to an address within the fun function so it's size is no more same as the size of 4 integers as in main.



What does the following C-statement declare?



```
#include <stdio.h>
int (*f) (int i);
int a(int x)
    return 2* x;
                                   Function pointer
                                   f points to
                                   function a
int main()
    f = &a;
    printf("%d", f(5));
    return 0;
```

Function Pointers

- Can use pointers that store addresses of functions!
- Generic format:

```
returnType ((* name) (type1, ..., typeN) ht of name
```

- Why are parentheses around (* name) needed?
- Using the function:
 - Calls the pointed-to function (*name)(arg1, ..., argN) rn the return value

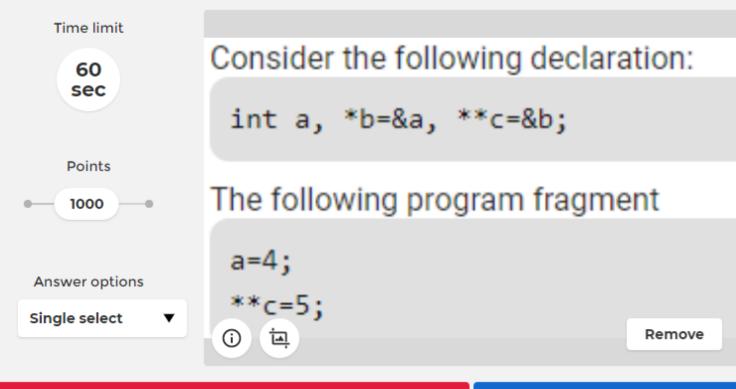


Function Pointer Example

• map () performs operation on each element of an

```
#define LEN 4
                                             funcptr parameter
int negate(int num) {return -num;}
   square(int num) {return num*num;}
// perform operation pointed to on each array element
void map(int a[], int len, int (* op)(int n))
  for (int i = 0; i < len; i++)
    a[i] = (*op)(a[i]); // dereference function pointer
                  funcptr dereference
int main(int argc, char** argv) {
\rightarrow nt arr[L\piN] = {-1, 0, 1, 2};
  map(arr, LEN, square);
                                       funcptr
                                       assignment
```

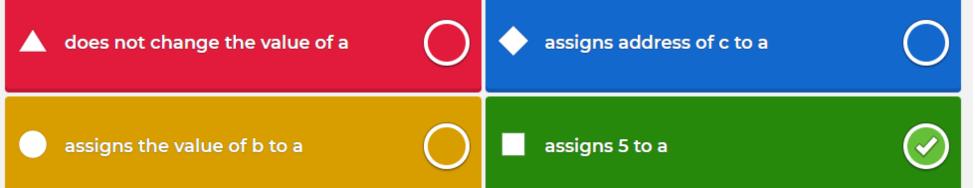
Choose the correct answer



Explanation:

b contains the address of a a contains the address of b

a is initialized to 4, then by double dereferencing the value of a is changed to 5



What is the output?

