What is the output of the following program?

Time limit

60 sec

Points

1000

Answer options

Single select

```
# include <stdio.h>
void fun(int *ptr)
{
    *ptr = 30;
}

int main()
{
  int y = 20;
  fun(&y);
  printf("%d", y);

  return 0;
}
```

Remove

Explanation: fun (&y) passes the address of y within the stack of main to ptr

When we dereference ptr using *ptr, we are actually changing the value of y within the stack of main

- 20
- 30 ✓
- Compile Error
- Runtime Error

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

Time limit

90 sec

Points

1000

Answer options

Single select

```c
#include <stdio.h>

int main()
{
    int ai[] = {1, 2 ,3};
    int *ptri = ai;
    char ac[] = {1, 2 ,3};


    printf("sizeof ai[] = %d, ", sizeof(ai));
    printf("sizeof ptri = %d, ", sizeof(ptri));
    printf("sizeof ac[] = %d, ", sizeof(ac));

    return 0;
}
```

Remove

- ▲ 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,

Explanation: sizeof 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

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

Time limit

90 sec

Points

1000

Answer options

Single select

```c
#include <stdio.h>

int main()
{
    float arr[5] = {12.5, 10.0, 13.5, 90.5, 0.5};
    float *ptr1 = &arr[0];
    float *ptr2 = ptr1 + 3;

    printf("%f ", *ptr2);
    printf("%d", ptr2 - ptr1);

   return 0;
}
```

Remove

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.

- 90.500000 12
- 10.000000 12
- 90.500000 3 ✓
- 0.500000 3

What is the output?

Time limit

30 sec

Points

1000

Answer options

Single select

```c
int main()
{
char *ptr = "CMSC257";
printf("%c", *&*&*ptr);
return 0;
}
```

Remove

- Compiler Error
- Runtime Error
- Address Value
- C

Explanation:

Strings are char arrays in C,

Here ptr contains the address of first character which is C

&& cancels out each other

*ptr is the C character

What is the output?

Time limit

90 sec

Points

1000

Answer options

Single select

```c
#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);
  return 0;
```

Remove

- 10 20 30 40
- 10 20
- Machine Dependent ✓
- Northing

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?

Time limit

30 sec

Points

1000

Answer options

Single select

```
int ( * f) (int * ) ;
```

Remove

Original

3x3

5x5

8x8

Explanation:

* Here is appended to the function name.

A function that takes an integer as argument and returns an integer pointer

A pointer to a function with integer parameter, returns integer

Add answer 3 (optional)

Add answer 4 (optional)

```c
#include <stdio.h>

int (*f) (int i);

int a(int x)
{
    return 2* x;

}

int main()
{
    f = &a;
    printf("%d", f(5));
    return 0;
}
```

Function pointer f points to function a

Function Pointers

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

`returnType (* name)(type1, …, typeN)` nt 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 array

```
#define LEN 4

int negate(int num) {return -num;}
int 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
  }
}

int main(int argc, char** argv) {
  int arr[LEN] = {-1, 0, 1, 2};
  map(arr, LEN, square);
  ...
```

funcptr parameter

funcptr dereference

funcptr assignment

map.c https://onlinegdb.com/S1Miv-GMU

Choose the correct answer

Time limit

60 sec

Points

1000

Answer options

Single select

Consider the following declaration:

```
int a, *b=&a, **c=&b;
```

The following program fragment

```
a=4;
**c=5;
```

Remove

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

- does not change the value of a
- assigns address of c to a
- assigns the value of b to a
- assigns 5 to a ✓

What is the output?

Time limit

90 sec

Points

1000

Answer options

Single select

```c
#include<stdio.h>
void
f (int *p, int *q)
{
  p = q;
  *p = 2;
}

int i = 0, j = 1;
int
main ()
{
  f (&i, &j);
  printf ("%d %d\n", i, j);
}
```

Remove

- 2 2
- 2 1
- 0 1
- 0 2 ✓

Explanation:

P contains the address of i
Q contains the address of j
With p=q p also contains the address of j

With *p=2 , the value of j is changed to 2, i is intact

So program prints 0 2