Object lifetime

Exercise 1

Given the following C fragment:

```
int x;

void foo(int y) {

    static int z;

    int * t = malloc(sizeof(int));

    ...

}
```

Choose the WRONG statement?

- ○ x is allocated in static memory
- ○ y is allocated in stack memory
- ○ z is allocated in static memory
- ○ t is allocated in heap memory

Correct

Correct

Correct

Wrong, t is a variable in pointer type. It is allocated in stack memory as it is a local variable of function foo. The object to which p points is allocated in heap memory

Solution

1. Wrong
2. Wrong
3. Wrong
4. Correct Option

Exercise 2

Given the following C fragment:

```
int x;

void foo(int y) {

    static int z;

    int * t = malloc(sizeof(int));

    ...

}
```

Choose the WRONG statement?

- The lifetime of x is the same as the lifetime of the whole program
- The lifetime of y is the same as the lifetime of function foo
- The lifetime of z is the same as the lifetime of function foo
- The lifetime of t is the same as the lifetime of function foo

Correct

Correct, y is the parameter of function foo so it is considered as a local variable which is allocated in stack memory and lives when function foo executes

Wrong, although z is a local variable, its lifetime is the same as the lifetime of the whole program as the variable is allocated in static memory

Correct, t is a local variable of pointer type so it lives just when the function runs. The lifetime of the object t points to however is not the same as the lifetime of function foo.

Solution

1. Wrong
2. Wrong
3. Correct Option
4. Wrong

Exercise 3

Given the following C code:

```
//position 1

int * foo() {
     // position 2

    x[0] = 1;

    return x;

}
```

which declaration of x and at which position can cause a runtime error (dangling reference or garbage)?

- ○ int x[10]; //position 1
- ○ int x[10]; //position 2
- ○ static int x[10]; // position 2
- ○ int* x = malloc(10*sizeof(int)); // position 2

Wrong, an object declared at position 1 (i.e global) will be allocated in static area so its lifetime is the same as the lifetime of the program; the object cannot be garbage. The above code also causes the dangling reference.

Correct, an object declared at position 2(i.e. local) will be allocated in stack area. Its lifetime is the same as the lifetime of the enclosed function (i.e. foo); it is allocated when the function runs and destroyed when the function is terminated. The instruction "return x" gives the reference to the object to some variable outside the functions. The reference to the object still exists after the object is killed when the function is terminated. This scenario causes "dangling reference"

Wrong, an object declared with keyword "static" will be allocated in static area although it is only referred inside the enclosed function.Its lifetime is the same as the lifetime of the program; it always be referred when the function foo is invoked so it cannot be a garbage. It is not destroyed when the program runs so it cannot cause a dancing reference.

Wrong, an object created by malloc will be allocated in heap area. It is not destroyed so it does not cause "dangling reference". It might be a garbage but the pointer is returned to outside. The error of "garbage" might be caused by the outside code, not by the above code.

Solution

1. Wrong
2. Correct Option
3. Wrong
4. Wrong