Lena Kemmelmeier

CS 326 - Programming Languages, Concepts and Implementation

Homework 6
Submitted: April 22nd, 2024

CS326 – Programming Languages, Concepts, and Implementation

Homework 6

(Due April 23)

1. (24 pts) Consider the following C program:

```
void foo()
 {
    int i;
    printf("%d ", i++);
 }

void main()
 {
    int j;
    for (j = 1; j <= 10; j++)
          foo();
 }
```

Local variable `i` in subroutine `foo` is never initialized. On many systems, however, variable `i` appears to "remember" its value between the calls to `foo`, and the program will print 0 1 2 3 4 5 6 7 8 9.

(a) (12 pts) Suggest an explanation for this behavior.

Each time foo is called, an activation frame is pushed onto the stack. Because of the way stack allocation works (foo is pushed onto the stack, then popped off repeatedly here), the activation frames for different calls of foo might belong to the same part of memory on stack. Taken together, because i is not initialized, it will contain whatever value happens to be at its memory location when foo is called. This means i will inherit whatever value is there, like from a previous instance of foo being called, appearing to "remember." On many systems, stack memory is initialized to zero when something new is made, then it makes sense why i (uninitialized) starts at 0 and increments with this "remembering."

(b) (12 pts) Change the code above (without modifying function `foo`) to alter this behavior.

If we initalize i before foo is called each time, then we get rid of this remembering behavior:

```
void foo()
```

```
{
   int i;
   printf("%d ", i++);
}

void main()
{
   int j;
   for (j = 1; j <= 10; j++)
         int i = 0; // new code here!
         foo();
}
```

2. (20 pts) Can you write a macro in C that "returns" the factorial of an integer argument (without calling a subroutine)? Why or why not?

No, you cannot write a macro in C that returns the factorial of an integral argument (without calling a subroutine) because to accomplish that goal, you would have to use a loop (for multiplying over and over), which you aren't able to use in the macro expression. After all, macros are essentially just text replacements, and they can't return the value of the expression without a loop.

3. (24 pts) Consider a subroutine `swap` that takes two parameters and simply swaps their values. For example, after calling `swap(X,Y)`, `X` should have the original value of `Y` and `Y` the original value of `X`. Assume that variables to be swapped can be simple or subscripted (elements of an array), and they have the same type (integer). Show that it is ***impossible*** to write such a general-purpose `swap` subroutine in a language with:

(a) (12 pts) parameter passing by value.

With passing by value, changes from inside the subroutine don't change the original values of X and Y: the values of the parameters are *copied* into the body of the subroutine. After we run swap, X and Y are still the same values they were before.

```
// general purpose swap subroutine

void swap(int first, int second){

      int placeHolder = first;

      first =  second;
```

```
        second = first;
}
int X = 1;
int Y = 2;
swap(X, Y);


print(X); // output with the example above would be 1
print(Y); // output with the example above would be 2
```

(b) (12 pts) parameter passing by name.

With pass-by-name, the parameters are substituted directly into swap without being copied - this basically gives rise to unintended/unexpected behavior. X and Y are dependent on each other in the swap here, but the pass-by name treats them as being independent, which produces incorrect results.

```
// general purpose swap subroutine
void swap(int first, int second){
        int placeHolder = first; // substitute first for X
        first =  second;  // substitute first for X and substitute second for Y
        second = first;// substitute second for Y
}
int X = 1;
int Y = 2;
swap(X, Y);


print(X); // output with the example above would be 1
```

print(Y); // output with the example above would be 2

Hint: for the case of passing by name, consider mutually dependent parameters.

4. (32 pts) Consider the following program, written in no particular language. Show what the program prints in the case of parameter passing by (a) value, (b) reference, (c) value-result, and (d) name. Justify your answer. When analyzing the case of passing by value-result, you may have noticed that there are two potentially ambiguous issues – what are they?

```
procedure f (x, y, z)
   x := x + 1
   y := z
   z := z + 1

// main
i := 1;
a[1] := 10;
a[2] := 11
f (i, a[i], i);
print (i);
print (a[1]);
print (a[2]);
```

(a) The program will print 1 10 11. Here, the values of the parameters are passed into the subroutine, but no changes are reflected in the originals.

(b) The program will print 3 2 11. Here, we pass in the memory addresses of the parameters, meaning changes are reflected in the originals.

The variable i is 3 because x is modified twice, a[1] becomes 2, (y is modified), and a[2] doesn't change, it's still 11.

(c) The program could print 2 1 11 (see my case below) - see reasons why its ambiguous below, too. In passing by value-result, the value of what's passed in is copied, and then it is copied back.

After the call, i is 2, a[1] is now 1 (x is modified), and a[2] is still 11. <u>However, these results change depending on the order of how we evaluate parameters (what value do we copy back?), or depending on how many times an expression is evaluated (i.e. could lead to the program incrementing a value again).</u>

(d) The program will print 3 10 2. Here, the argument is substituted into the function.

Here, i becomes 3 (x is modified twice), a[1] becomes 10 (y is replaced by a[i]), and a[2] is 2 because the value of y changed to 2 and i was 2 at that point.

5. (Extra Credit - 10 pts) Does a program run faster when the programmer does not specify values for the optional parameters in a subroutine call?

No, because these optional parameters still have a default value (whether you specify a value is resolved at compile-time) so runtime costs will be very similar.