Programming Languages: Homework #6

Due on April 25, 2019 at 9:00am

Erin Keith Section 101

Michael DesRoches

Problem 1

(24pts) 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)(12pts) Suggest an explanation for this behavior.

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

Solution

(a) The local variable, i, is never initialized. Therefore, the value isn't really known. This will cause undefined behavior depending on the compiler. We can't know the outcome and the outcome may be different on different machines.

(b)

```
static void overWriteMemoryLocation(void)
{
    int j = 10;
}
void foo()
{
    int i;
    printf("%d ", i++);
}
int main()
{
    int j;
    for (j = 1;
    j <= 10; j++)
    foo();
}
```

Calling another function before foo() overwrites the memory location that foo() will use for i.

Problem 2

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

Solution
No, you can't have macro in C w/out calling a soubroutine. Macro is a preprocessor before the code is compiled. Preprocessors include the headers and text replace the macros with their definition. So, any recursive definition of a macro will be replaced only once and not until the base condition is met.

Problem 3

(24pts) 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)(12pts) parameter passing by value.
(b)(12pts) parameter passing by name.

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

Solution

A subroutine is termed as an executable code resided with in the block or in a function. Consider the sub routine for the swap program in the language:

(a) Parameter passing by value:

```
procedure swap(x, y: int):
          var t : int
               t:=x;
               x:=y;
               y:=t;
end;

Main:
     Declare i,j;
     Call swap(i,j)
```

By calling the parameters through pass by values, none of the actual arguments can be changed, so the variables retain the values they are initialized with.

Flow for the function call by its value:

```
temp:=i
```

```
i:=j
j=temp
```

In this scenario the subroutine will not reflect the function. Hence it shows is impossible to write the code for the elements of the array.

(b) Parameter passing by name:

```
procedure swap(x, y: int):
          var t : int
               t:=x;
               x:=y;
               y:=t;
end;

Main:
     Declare i,a[];
     Call swap(i,a)
```

By calling the subroutine by passed by name conflicts are taken care of between actual parameters and the local variables of the function.

Flow for the function call by its name:

```
temp:=i
i:=a
a=temp
```

Call by name cant handle the swap properly by calling its name.

Problem 4

(32pts) 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 potentiallyambiguousissueswhat 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]);
```

Solution
(a) 1, 10, 11
(b) 3, 2, 11
(c) 3, 11, 3
(d) When initially called, i = 1, a[i] = 10. Then the first stetement changes i to 2. Then a[2] is changed to 2, value is set to z which is also the same as x. Then i is changed to 3. Print = 3, 10, 2. Result = 3, 2, 3

Problem 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?

Solution
Performance gain is minimal because the subroutine call but the answer is yes because it takes time to to evaluate the parameters that are being called with the function.