Willis Allstead

4/24/17

CS 326

HW6

* 1. The reason i seems to “remember” its value between calls could be that on my specific system it is being incremented through the same stack pointer each call. But this is undefined behavior and could show completely different values on another computer.
  2. void foo() {

int i;

printf("%d ", i++);

}

void bar() {

int i;

i++;

}

int main() {

int j;

for (j = 1; j <= 10; j++) {

foo();

bar();

}

}

to alter the behavior of this program I have included a similar function called bar() in which I have defined and incremented i. I call this along with foo() and the output (on my machine) is:

0 2 4 6 8 10 12 14 16 18

This shows that my previous hypothesis could be true and that the system will take both of these int i’s are assigned a single stack pointer.

1. I don’t believe it is possible to write a macro in C that “returns” the factorial of an integer argument because at compile time there is no way of knowing the base case.
2. 1. Parameter passing by value in a language would mean that it is impossible to write a general purpose swap subroutine because you wouldn’t be able to affect the arguments being passed in, just the new variables with the same values as those passed in.
   2. Parameter passing by name wouldn’t work in the general case either. If you pass a simple and a subscript int as arguments the temp would be set to the first argument X. X would be set to the evaluated subscript Y, and the subscript Y would be set to temp, which is evaluated as the subscript of Y.
   3. Output (pass by value): 1 10 11 /\* the function call does not affect the values in main in this case. \*/
   4. Output (pass by reference): 2 2 3 /\* the function call affects all values. When y is set to z, i is referenced, which is equivalent to x at that point. When z is incremented, it increments the current value of i, which is 2. \*/
   5. Output (pass by value-result): 2 1 11 /\* the function returns i as the value of i+1: 2. It then returns the value of a[1] as 1, and a[2] wasn’t changed by the function. The question I see with evaluating this is whether or not the index of the array argument is evaluated at the current local scope of the function or the passed-in value. \*/
   6. Output (pass by name): 2 10 3 /\* the function increments i to 2. It then evaluates a[2] as equal to i. Then it sets increments i again. \*/
3. It shouldn’t run any faster. Those optional parameters are just default values which are given in the absence of supplied ones. This should not affect the speed of the program.