Drew Smith
COP 4020
April 21, 2016

Homework 5 Due April 22, 2016

1. (20 points) Explain the difference between unstructured and structured control flow. For at least two common structured control flow mechanisms (such as while loops, if- statements, etc...), briefly explain how they replace the use of goto statements in a programming language.

Unstructred control flow utilizes goto statements and statement labels to implement control flow while structured control flow uses statement sequencing. Two structured control flow mechanisms that were able to replace goto statements in a programming language consist of "mid-loop exit and continuation" and "early returns from subroutines". Mid-loop exit and continuation, replaced with "one-and-a-half" loop constructs, uses keywords like "break" and "continue" to implement control flow. Early return from subroutines allows a value to be returned early within a function.

2. (16 points) Explain the difference between enumeration-controlled loops and logically controlled loops. For a language like C in which enumeration-controlled loops are not available, explain (with a sentence or two) and write a small code example showing how they may be emulated.

Enumeration-controlled loops repeat a collection of statements a number of times, where in each iteration, a loop index variable takes the next value of a set of values specified at the beginning of the loop. Logically-controlled loops repeat a collection of statements until some Boolean condition changes value in the loop. "For loops" in C can be structured to mimic enumeration loops but it is the programmers responsibility to do so. An example:

```
for(int I = 0; I <= n; i++)

        i++;
```

As we can see it is the programmers responsibility to increment the value of 'i'.

3. (16 points) What is tail-recursion and why is it desirable over naive recursion? Is it possible to write a fully tail-recursive version of the classic Quicksort algorithm? Hint: if you're going to look this one up online, think carefully about the answer you submit.

Drew Smith
COP 4020
April 21, 2016

Tail-recursive functions are functions in which no operations follow the recursive calls. A tail-recursive call could reuse the subroutine's frame on the run-time stack, since the current subroutine state is no longer needed. It is desirable over naïve recusion functions because tail-recursion can be implemented more efficiently than general recursion. When we make a normal recursive call, we have to push the return address onto the call stack then jump to the called function. This means that we need a call stack whose size is linear in the depth of the recursive calls. It is not possible to write a full-tail-recursive version of the classic Quicksort algorithm, but the Quicksort algorithm can be edited to mimic a tail-recursive function.

4. (16 points) Consider the following C-style program below. Diagram the changes in the stack that occur when some subroutine bar is called from the current subroutine foo. In particular, show the subroutine frames on the stack before the call of bar, while bar is executing, and after it has returned. Show the positions of the stack pointer (sp) and the frame pointer (fp) at each stage. Also, using the typical subroutine frame layout discussed in class, indicate the stack locations of the arguments to bar as well as the return value.

```
int bar(int a, int b){     int ret;

ret = a+b;

return ret; }

void foo(){     int x;
    x = bar(1,2); }
```

5. (16 points) Assume we have the following program for some language with C-like syntax.

```
int x, y, z;

void foo(a, b, c) {
      a = a*2;
      b = b*2;
      x++;
      c = c+b;

}

x = 1; y = 2; z = 3;
foo(x, y, z);
```

Drew Smith
COP 4020
April 21, 2016

What are the values of x, y, and z after the call to foo for each of the following cases?
a) The formal parameters a, b, and c are all call-by-value?

x = 1; y = 2; z =3;

b) The formal parameters a, b, and c are all call-by-reference?

x = 2; y = 4; z = 6;

c) The formal parameters a, b, and c are all call-by-value/result (i.e. in/out mode in Ada)?

x = 2; y = 4; z = 6;

d) The formal parameters a, b, and c are all call-by-name?

x = 3, y = 4; z = 6;

6. (16 points) Using the same definition for foo that we used in question 5, assume we make the following call.

```
x = 1;
foo(x, x, x+1);
```

What is the value of x after the call to foo for each of the following cases?
a) The formal parameters a, b, and c are all call-by-value?
x = 1

b) The formal parameters a, b, and c are all call-by-reference?
x = 4

c) The formal parameters a, b, and c are all call-by-value/result (i.e. in/out mode in Ada)?

x = 2

Drew Smith
COP 4020
April 21, 2016

d) The formal parameters a, b, and c are all call-by-name?

x = 6