CSE 2341 – Fall 2019

Homework Assignment 1

Due: Wednesday, September 18, 2019 at 11:00 pm upload to Canvas

Submission Directions: Complete the questions below. For questions that require diagrams, create the diagrams on your computer (no hand-drawn/scanned diagrams) using something like PowerPoint, Google Draw, or similar. Copy-and-paste the diagrams into the correct place in the document you submit. For any code or pseudocode, use a fixed-point font (courier for example). Please submit a ***PDF*** file of your solutions.

1. A stack overflow error means that more "data" has been placed on the stack than it can hold based on its size. Implement a simple recursive function that does not contain a base case. How many times can this function call itself before you reach a stack overflow? To answer this question, provide the count as well as the code for the recursive function. Copy and paste the code for the recursive function in your solutions document. [10]

Answer:
count reached 261932 before segfault.

```
#include <iostream>

void overflow(int& count){
        std::cout << count << std::endl;
        overflow(++count);
}

int main(){
        int count = 0;
        overflow(count);
}
```

2. Given:

```
char data[6][10]   = {"Sam", "Robert", "Mark", "Jason", "Alex", "Karen"};
```

Determine the output of the following statements. Note that some may print memory addresses. If so, indicate the memory address by referring to the letter or array that it points to. For example:

```
cout << &data[1][1];
```

a correct answer would be "address of o of Robert". If the statement contains an error, state such.

[2 points each]

a) `cout << data[3];`
   Jason

b) `cout << *data[2];`
   M

c) `cout << *(*(data+1)+1);`
   o

d) `cout << ***data;`
   Error, cannot dereference a char.

e) `cout << **(data+ 4);`
   A

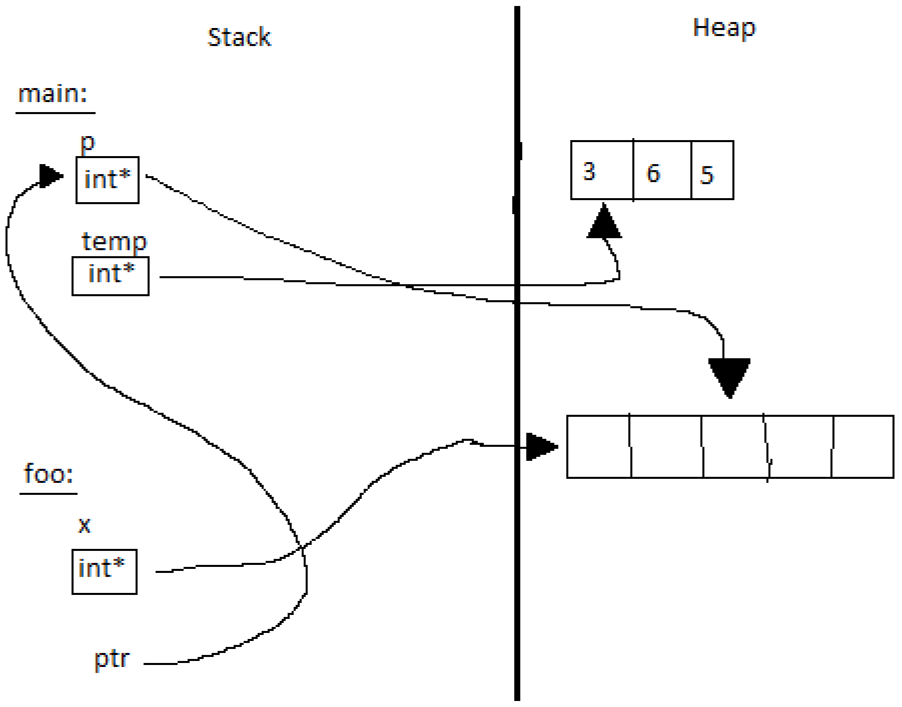
f) `cout << *(data + 3);`
   Jason

g) `cout << data;`
   The address of Sam

3. Draw the state of memory (a memory diagram) for the following code at the point indicated. [10]

```
void foo(int *& ptr)
{
        int * x = new int[5];
        ptr = x + 2;
        //Draw memory diagram
        //based on state of memory
        //here
}
```

```
int main (){
        int *p = new int[3];
        p[0] = p[1] = p[2] = 5;
        *p = 3;
        *(p+1) = 6;
        int * temp = p;
        foo(p);
        return 0;
}
```



Memory Diagram:

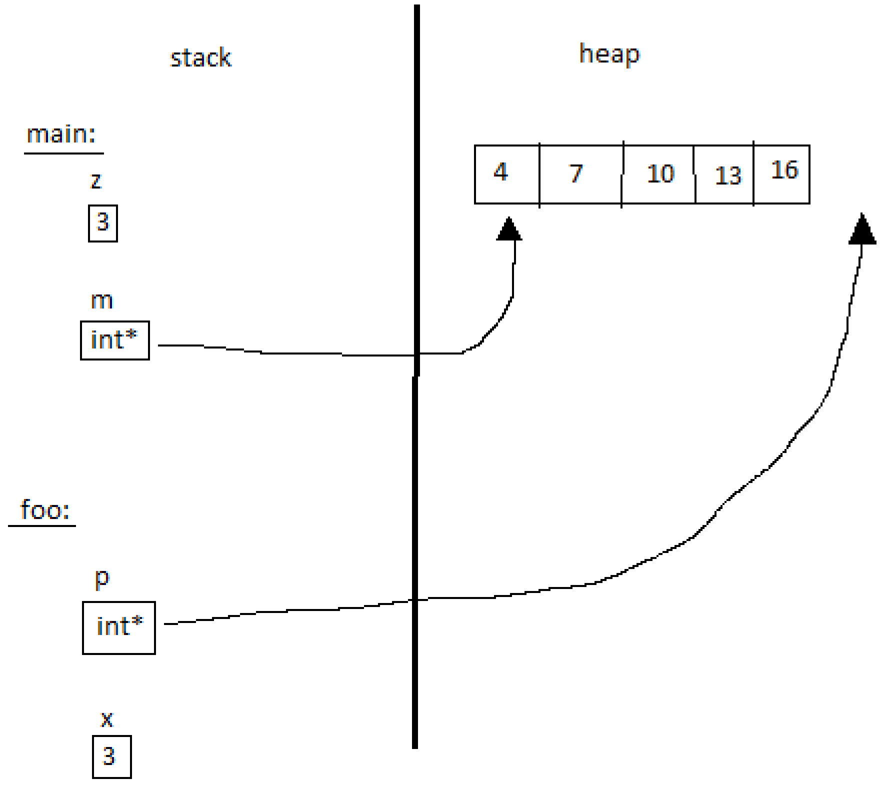
4. Draw the state of memory (a memory diagram) for the following code at the point indicated. [10]

```
int foo(int* p, int x)
{
            for (int i = 1; i <= 4; i++)
            {
            *(p) = *(p - 1) + x;
            p++;
            }
            /*Draw memory diagram at this point*/
}

int main()
{

            int z = 5;
            int* m;
            m = new int[z];
            m[0] = 4;
            z = 3;
            foo(m+1, z);
            return 0;
}
```

Memory Diagram:



5. What’s wrong with the code below? Feel free to compile and run the program to investigate. [6]

```
#include <iostream>
using namespace std;

class HW1
{
public:
  char *word;

  HW1() { }
  ~HW1()
  {
        delete [] word;
  }
};
```

```
void foo(HW1 a)
{
        //some operation here
}

int main()
{
        HW1 a;
        a.word = new char[80];
        strcpy(a.word, "abc");
        foo(a);
        return 0;
}
```

Answer:

Assuming the problem isn’t a missing #include <cstring> and that’s meant to be in the code, the problem is that there isn’t a copy constructor defined for HW1. When a is passed by value to foo, a shallow copy of it is made. That shallow copy then has the destructor called on it upon exiting the scope of foo, which calls delete[] on a copy of a.word. When the program exits main, the destructor is called on a, which calls delete[] on a.word. However, the memory that a.word points to has already been freed by the earlier delete[], causing a double free error.