LAB #8 – More Pointers/Arrays

Pre-Lab:
Since many of you didn’t finish the command line arguments part of lab #7, finish this part of the lab and make sure you understand it!!!

Practice Pointers:
1. Remember that you have a stack and heap for your variables. Statically allocated variables and functions live on the stack, while dynamically allocated variables and objects are on the heap. Write a program that declares primitive variables, pointers, dynamically allocated variables, arrays, and a string object, and print their addresses to determine if they are on the stack or heap. On Linux, the heap is located at smaller memory addresses and the stack is at higher addresses. Here are the variables, arrays, pointers, etc. you must declare to determine their addresses:

   ```
   char p;               //Primitive character type
   char *ptr;           //Pointer to dynamically create a new char
   char a[3];           //Array of 3 characters
   char *aptr;          //Pointer to dynamically create an array, new char[3]
   char a2[3][3];       //2-D array of 9 characters
   char *a2ptr[3];      //Array of 3 pointers to dynamically create 3 new char[3]
   char **a2dptr;       //Double pointer to dynamically create a [3][3] array
   string str;         //String object
   ```

   For each of the pointers, you must create the pointee!!! You must print all addresses, including the pointees’ addresses. Provide a message explaining what you are printing to determine where each variable, pointer, array, and object lives in memory. To ensure you get the address of everything, you can type cast to an (unsigned long), since our machine uses 64-bit addresses. For example:

   ```
   #include <iostream>
   #include <string>

   using std::cout;
   using std::endl;
   using std::hex;
   using std::string;

   int main(int argc, char *argv[]) {
     char *a2ptr[3];
     string str = "hello";

     for(int i=0; i<3; i++)
       a2ptr[i]=new char[3];

     cout << hex;
     cout << "a2ptr address: " << (unsigned long) &a2ptr << endl;
     cout << "a2ptr[] address: " << (unsigned long) &a2ptr[0] << endl;
     cout << "a2ptr[][] address: " << (unsigned long) &a2ptr[0][0] << endl;
   }
   ```
cout << "str address: " << (unsigned long) &str << endl;
cout << "str address: " << (unsigned long) &str[0] << endl;

return 0;
}

Now, you finish create the memory space and printing addresses for all the other variables and pointers, and determine when something is on the stack versus the heap by the address that is printed.

2. In this part of the lab, we will create a pointer to a pointer to a pointer to a pointer that points to an integer, i.e. int ****p;. Now, create an integer, i, and set it to the value 100, i.e. int i=100;. Make p point to i, and print the contents of i using the pointer, p. ***NOTE: Draw a picture to make sure you understand how to set this up!!

Next, change the value of i to 200 using the pointer, p. Next print the contents of i using i, i.e. cout << i << endl;, to make sure its value changed to 200.

Print the addresses of each memory location along the chain that connect ****p to i.

3. Change your reverse string, so that it uses pointers to swap the contents of the strings. The “head” pointer should be set to the address of the first character in the string and the “tail” pointer should be set to the address of the last character in the string (i.e. the character before the terminating null). The program should swap the characters referenced by these pointers, increment “head” to point to the next character, decrement “tail” to point to the second-to-last character, and so on, until all characters have been swapped and the entire string reversed.

Show your programs to a lab TA for full lab credit.

Extended Learning:
Tweak #2, so that we make a linked list of integers using structs that contain an integer and a pointer to another struct. This is what we call a linked list in computer science, and we use this data structure often. The struct that you will create is called a node:

```c
struct node {
    int num;
    node *next;
};
```

Now, create a list of integers that are chained together using the struct node, but you will need to make a “head” that points to the beginning location of the first node in your list of integers, i.e. node *head = new node; Set the num value to 5, and then create 4 more nodes with integers set to 4, 3, 2, and 1 (NOTE: use a tmp to save your pointer value!). Now print the integers in order from 1 to 5. Refer to Chapter 17 for help😊