====Assignment 4 - Theodore Hung Nguyen - 704-156-701 - CS31 Summer 2014=====

```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <string>
#include <cstring>
#include <cmath>
using namespace std;
```

1. Define integer variables x = 2, y = 12 and two pointer variables p and q(p pointer
of x, and q pointer of y).Then print the following information :
a.The address of x and the value of x.
b.The value of p and the value of *p.
c.The address of y and the value of y.
d.The value of q and the value of *q.
e.The address of p.
f.The address of q.

```
int main()
{
int x = 2, y = 12;
int* p = &x;
int *q = &y;
//a) address of x, value of x
cout << "a) The address of x is " << &x << " while the value of x is " << x << endl;
//b) the value of p and the value of *p
cout << "b) The value of p is " << p << " while the value of *p is " << *p << endl;
//c) the address of y and the value of y
cout << "c) The address of y is " << &y << " while the value of y is " << y << endl;
//d) the value of q and the value of *q
cout << "d) The value of q is " << q << " while the value of *q is " << *q << endl;
//e) the address of p
cout << "e) The address of p is " << &p << endl;
//f) the address of q
cout << "f) The address of q is " << &q << endl;

}
```

2. Define an array of integers arr[5] and fill this array with values 11, 40, 27, 0, 3.
Define integer i and a pointer variable p that points to i. What is the output of

```
following two printing loops :
for (i = 0; i < 5; i++)
        cout << i << " " << arr + i << " " << arr[i] << endl;
cout << endl;
i = 0;
p = arr;
while (p < (arr + 5))
{
        cout << i << " " << p << " " << *p << endl;
        i++;
        p++;
}



int main()
{
        int arr[5] = { 11, 40, 27, 0, 3 };
        int i;
        int* p = &i;
//sets i = 0, goes through the loop 5 times; each time, i, arr +i, and arr[i] is
//outputted

        for (i = 0; i < 5; i++)
// arr + i outputs the memory address of arr[i]; format: 'i location    value'
                cout << i << " " << arr + i << " " << arr[i] << endl;
        cout << endl;

//sets i = 0, sets p to the memory location at arr[0], condition is that p, the
//memory address location, must be less than the address location at the value after

//the end of the array
        i = 0;
        p = arr;
//outputs the same thing as the prior loop in the format: 'i    location  value'
        while (p < (arr + 5))
        {
                cout << i << " " << p << " " << *p << endl;
                i++;
                p++;
        }

}




will output

        0       memory location0        11
        1       memory location1        40
        2       memory location2        27
        3       memory location3        0
        4       memory location4        3

        0       memory location0        11
        1       memory location1        40
        2       memory location2        27
        3       memory location3        0
        4       memory location4        3
```

where "memory location#" is the location in the memory for the stored variable
- is different for each iteration

3. The subparts to this problem involve errors in the use of pointers.
a.This program is supposed to write 50 60 70, one per line.Find all of the
bugs and show a fixed version of the program :

ORIGINAL - pointing out bugs!

```
#include <iostream>
using namespace std;
int main()
{
        int MyArray[3] = { 10, 20, 30 };
        int *p = MyArray;
        *p = 70; // set MyArray[0] to 70
        *p += 1;        //this is incorrect:
                        //increases value of MyArray[0] by one, making 70 -> 71
        *p = 60; // set MyArray[1] to 60 //changes value of MyArray[0] 71 -> 60
        p += 2;                         //this code line works.
        p[0] = 50; // set MyArray[2] to 50
        do
        {
                p--;

                cout << *p << endl; // print values //actually prints 20, 60,
                                                   //GARBAGE on different lines,
                                                   //b/c he did p-- before cout
        } while (p >= MyArray);
}
```

FIXED version of program

```
int main()
{
        int MyArray[3] = { 10, 20, 30 };
        int *p = MyArray;
        *p = 70; // set MyArray[0] to 70
        p += 1;                         //removed the * operator to increment
                                        //the pointer instead of the value at
                                        //the location
        *p = 60; // set MyArray[1] to 60
        p += 1;                         //increment by 1 instead of 2, because
                                        //we had incremented before
        p[0] = 50; // set MyArray[2] to 50
        do
        {
                cout << *p << endl; //print values //moved p-- to after the cout
                p--;
        } while (p >= MyArray);
}
```

3b. The findmin function is supposed to find the minimum item in an array
and set the pToMin parameter to point to that item so that the caller knows
that item's location. Explain why this function won't do that, and show
how to fix it. Your fix must be to the function only; you must not change
the main routine below in any way, yet as a result of your fixing the
function, the main routine below must work correctly.

ORIGINAL - error finding

```
void findmin(int MyArray[], int n, int* pToMin)         //you gotta pass the
                                                        //pointer by reference...
{
        if (n <= 0)
                return; // no items, no maximum!        //because of this, pToMin
                                                //will not have a value after
                                                //the function call ends if the
                                                //array size is zero
        pToMin = MyArray;
        for (int i = 1; i < n; i++)
        {
                if (MyArray[i] < *pToMin)
                        pToMin = MyArray + i;
        }
}
int main()
{
        int nums[4] = { 45, 13, 5, 66 };
        int* p;
        findmin(nums, 4, p);
        cout << "The minimum is at address " << p << endl;
        cout << "It's at position " << p - nums << endl;
        cout << "Its value is " << *p << endl;
}
```

FIXED version

```
void findmin(int MyArray[], int n, int* &pToMin)
{
        if (n <= 0)
        {
                pToMin = NULL;
                return; // no items, no maximum!
        }
        pToMin = MyArray;
        for (int i = 1; i < n; i++)
        {
                if (MyArray[i] < *pToMin)
                        pToMin = MyArray + i;
        }
```

```
}
```

3c. The computeCube function is correct, but the main function has a problem. Explain why it may not work and show how to fix it.
Your fix must be to the main function only; you must not change computeCubein anyway.

```
void computeCube(int n, int* ncubed)    //the pointer does not point to anything,so
                                        //computeCube cannot modify its value
{
        *ncubed = n * n * n;
}
```

ORIGINAL main

```
int main()
{
        int* ptr;
        computeCube(5, ptr);
        cout << "Five cubed is " << *ptr << endl;
}
```

FIXED main

```
int main()
{
        int i;
        int* ptr = &i;
        computeCube(5, ptr);
        cout << "Five cubed is " << *ptr << endl;
}
```

3d.The strequal function is supposed to return true if and only if its two C string arguments have exactly same text.
What are the problems with the implementation of the function, and how can they be fixed ?

```
//#include <iostream>
//using namespace std;
// return true if two C strings are equal
```

ORIGINAL function - error finding

```
bool strequal(const char str1[], const char str2[]);
{
        while (str1 != 0 && str2 != 0)  //should be comparing the values in the
index,
                                        //not the location
```

```
        {
                if (str1 != str2)  // compare corresponding characters
                                        //again, should be comparing values,
                                        //not locations
                        return false;
                str1++;             // advance to the next character
                str2++;
        }
        return *str1 == *str2;   // both ended at same time?
}
int main()
{
        char a[15] = "Chen, B.";
        char b[15] = "Chen, B";

        if (strequal(a, b))
                cout << "They're the same person!\n";
}



FIXED function
bool strequal(const char str1[], const char str2[])
{
        while (*str1 != 0 && *str2 != 0)

        {
                if (*str1 != *str2)  //compare corresponding chars //fixed by adding

                                                        //* to compare the actual
                                                        //values, not the locations

                        return false;
                str1++;             //advance to the next character
                str2++;
        }
        return *str1 == *str2;   //both ended at same time?
}
*/
```

3e.This program is supposed to write 5 4 3 2 1, but it probably does not.
What is the problem with this program ? (We're not asking you to propose a fix to the problem.)

```
int* getPtrToArray(int& m)
{
        int anArray[5] = { 5, 4, 3, 2, 1 };
        m = 5;
        return anArray;
}

void f()
{
        int junk[100];
        for (int k = 0; k < 100; k++)
                junk[k] = 123400000 + k;
}

int main()
```

```
{
        int n;
        int* ptr = getPtrToArray(n);
        f();
        for (int i = 0; i < n; i++)
                cout << ptr[i] << ' ';
        cout << endl;
}
```

What is wrong with the program:

This program doesn't work because it tries to make ptr in the main function point to an array that is LOCAL to getPtrToArray;
afterwards, it tries to output the values inside the locally created array, which does not exist outside its scope - this is why it returns
garbage values, different for each time you run the program, onto the output. The f() function call in the main function does absolutely
nothing to help hte situation either - its literally junk.

4. For each of the following parts, write a single C++ statement that performs the indicated task.
For each part, assume that all previous statements have been executed(e.g., when doing part e, assume
the statements you wrote for parts a through d have been executed).

a. Declare a pointer variable named student that can point to a variable of type double.
b. Declare grades to be a 5-element array of doubles.
c. Make the student variable point to the last element of grades.
d. Make the double pointed to by student equal to 17, using the * operator.
e. Without using the student pointer, and without using square brackets, set the fourth element
(i.e., the one at position 3) of the grades array to have the value 72.
f. Move the student pointer back by three doubles.
g. Using square brackets, but without using the name grades, set the third element (i.e., the one at position 2)
of the grades array to have the value93.
h. Without using the * operator, but using square brackets, set the double pointed to by student to have the value 85.

Solution:

```
int main()
{
        //a
        double* student;                                    //nothing yet
        //b
        double grades[5] = {};  // 0 0 0 0 0
        //c
        student = &grades[4];   // 0 0 0 0 0
```

```
        //d
        *student = 17;                  // 0 0 0 0 17
        //e
        *(grades + 3) = 72;             // 0 0 0 72 17
        //f
        student -= 3;                   // 0 0 0 72 17
        //g
        student[0+1] = 93;              // 0 0 93 72 17
        //h
        student[0] = 85;                // 0 85 93 72 17
}
```

5a. Rewrite the following function so that it returns the same result, but does not increment
the variable ptr.Your new program must not use any square brackets, but must use an integer variable
to visit each double in the array.You may eliminate any unneeded variable.

ORIGINAL FUNCTION

```
double mean(const double* scores, int numScores)

{
        const double* ptr = scores;
        double tot = 0;
        while (ptr != scores + numScores)
        {
                tot += *ptr;
                ptr++;
        }
        return tot / numScores;
}
```

FIXED - REWRITTEN - function

```
 double mean(const double* scores, int numScores)
 {
         double total = 0;
         for (int i = 0; (scores + i) != scores + numScores; i++)
         {
                 total += *(scores + i);
         }
         return total / numScores;
 }
```

5b. Rewrite the following function so that it does not use any square brackets (not even in the parameter declarations)
but does use the integer variable k.*/

```
    // This function searches through str for the character chr.
    // If the chr is found, it returns a pointer into str where
    // the character was first found, otherwise NULL (not found).
```

ORIGINAL FUNCTION

```
    const char* findTheChar(const char str[], char chr)
```

```
    {
        for (int k = 0; str[k] != 0; k++)
            if (str[k] == chr)
                return &str[k];

        return NULL;
    }
```

FIXED - REWRITTEN FUNCTION

```
        const char* findTheChar1(const char* ptrtocstr, char chr)
        {
                for (int k = 0; *(ptrtocstr + k) != 0; k++)
                if (*(ptrtocstr + k) == chr)
                        return (ptrtocstr + k);

                return NULL;
        }
```

Now rewrite the function shown in part b so that it uses neither square brackets nor any integer variables.
Your new function must not use any local variables other than the parameters.
//5c

REWRITTEN FUNCTION

```
const char* findTheChar2(const char* ptrtocstr, char chr)
{
        while (*(ptrtocstr) != 0)
        {
                if ((*ptrtocstr) == chr)
                        return ptrtocstr;
                ptrtocstr++;
        }
        return NULL;
}
```

6.What does the following program print and why ? Be sure to explain why each line of output prints the why it does to get full credit.

```
int* maxwell(int* a, int* b)    //function returns a pointer to an integer and
                                //takes in two pointers to integers
{
        if (*a > *b)    //if the value of the integer that the first pointer points
                        //to is greater than the second, then return the first
pointer

                return a;       //otherwise, return the value of the second
                                //(aka, this fcn returns the pointer that
                                //points to the larger integer)

        else
                return b;       //however, note that if both integers have the
```

```
                                //same value, the second pointer will be returned
                                //regardless
}

void swap1(int* a, int* b)      //function that does not return anything, takes in
                                // two pointers to integers
{
        int* temp = a;                  //creates a pointer to an integer and
                                        //points it to the same integer that the
                                        //first pointer points to

        a = b;                          //has the first pointer point to the
                                        //integer that the second pointer points to

        b = temp;               //has the second pointer point to what the first
                                //pointer initially pointed to

}

//PROBLEM IS, this is NOT PASSED BY REFERENCE, so none of the pointers point to
//the new assigned memeory location, so this function doesnt really do anything

void swap2(int* a, int* b)      //function that does not return anything and takes
                                // in two pointers to integers
{
        int temp = *a;          //creates an integer and sets it equal to the
                                //integer value that the first pointer points to

        *a = *b;                //sets the integer value that the first pointer
                                //points to equal to the integer value that the
                                //second points to

        *b = temp;              //sets the integer value that the second pointers
                                //points to equal to the integer value that the
                                //first pointed to initially
}

//essentially, the function swapped the values!

int main()
{
        int array[6] = { 5, 3, 4, 17, 22, 19 };         //makes an integer array
                                //with 6 elements, and fills in those elements


        int* ptr = maxwell(array, &array[2]);           //creates a pointer to an
                //integer and sets it equal to the output of function maxwell

//when inputting the address of the 1st and then 3rd element of the array
//the 1st element has a greater value than the 3rd, so ptr points to the first
//element of the array

        *ptr = -1;      //changes the value of array[0] to -1;

        ptr += 2;       //ptr now points to array[2] after addition of 2

        ptr[1] = 9;     //changes array[3] from 17 -> 9

        *(array + 1) = 79;      //changes array[1] from 3 -> 79

        cout << &array[5] - ptr << endl;        //ptr currently points to array[2],
                                        //so &array[5]-&array[2] = 3; 3 is cout'd
```

```
        swap1(&array[0], &array[1]);    //does absolutely nothing! - you can't
                                        //change where arrays point anyway, so
                                        //even if swap1 passed by reference to
                                        //make it work, it would not compile

        swap2(array, &array[2]);        //swaps the value of array[0] with array[2];

                                        //now: array[0] = 4, array[2] = -1

        for (int i = 0; i < 6; i++)     //outputs all the final elements of the
                                                //array: 4 79 -1 9 22 19
                cout << array[i] << endl;
}
```