Nathan Tung
Com Sci 31 (Lec 2)
November 21, 2011

CS31 Project 6 Homework

1. The subparts to this problem involve errors in the use of pointers.
   a. This program is supposed to write `10 20 30`, one per line. Find all of the bugs and show a fixed version of the program:

```
int main()
{
    int arr[3] = { 5, 10, 15 };
    int* ptr = arr;

    *ptr = 10;          // set arr[0] to 10
    *ptr + 1 = 20;      // set arr[1] to 20
    ptr += 2;
    ptr[0] = 30;        // set arr[2] to 30

    while (ptr >= arr)
    {
        ptr--;
        cout << *ptr << endl;    // print values
    }
}
```

→The first red section does not compile, since it is dereferencing the pointer first. It should increment the pointer by one then dereference it to change the value to 20. The second red section outputs 30, 20, and 10 on different lines, respectively. To make it work as intended (to print starting from 10), we could use a simple for-loop and output using the original array.

Rewritten version:

```
int main()
{
       int arr[3] = { 5, 10, 15 };
       int* ptr = arr;

       *ptr = 10;
       *(ptr + 1) = 20;    //Find the pointer which points to the element AFTER ptr, then
//dereference and change its value

    ptr += 2;
       ptr[0]=30;

       for(int i=0; i<3; i++)//The example prints out values backwards (incorrect); use a
//for-loop to print elements in order instead
              cout << arr[i] << endl;
}
```

b. The `findMax` function is supposed to find the maximum item in an array and set the `pToMax` parameter to point to that item so that the caller knows its 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.

→The findMax function doesn't return anything; its purpose is to change the pointer passed-in through the parameter. However, the original function did not allow any pass-by-references, so ptr in the main function was never updated with the address of pToMax. To change this, we can simply add the & sign in the last parameter of findMax, making it int* &pToMax.

```
void findMax(int arr[], int n, int* &pToMax) //Must pass pointer by reference!
{
    if (n <= 0)
        return;      // no items, no maximum!
    pToMax = arr;
    for (int i = 1; i < n; i++)
    {
        if (arr[i] > *pToMax)
                pToMax = arr + i;
    }
}

int main()
{
    int nums[4] = { 5, 3, 15, 6 };
    int* ptr;
    findMax(nums, 4, ptr);
    cout << "The maximum is at address " << ptr << endl;
    cout << "It's at index " << ptr - nums << endl;
    cout << "Its value is " << *ptr << endl;
}
```

c. 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 `computeCube` in any way.

→The main function never initializes the value of ptr. Thus when computeCube is called, the pointer that is passed in is dereferenced as a NULL pointer, which causes a program crash. To prevent this, we can initialize an arbitrary int z to 0 then temporarily set ptr to equal its pointer.

```
void computeCube(int n, int* ncubed)
{
    *ncubed = n * n * n;
}

int main()
{
        int z=0;
        int* ptr = &z;
    computeCube(5, ptr);
    cout << "Five cubed is " << *ptr << endl;
}
```

d. 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?

→str1 and str2 refer to the pointer itself, so their pointer addresses are always different. Yet we are comparing their addresses as though they show whether the actual char elements are equal. By dereferencing them as indicated by the red asterisk, we will test for the actual char elements held at those pointers. At the end, we check if both last chars are zero-bytes before declaring the C strings equal.

```
bool strequal(const char str1[], const char str2[])
{
    while (*str1 != 0  &&  *str2 != 0) //Keep looping until a zero byte is found
    {
        if (*str1 != *str2)  //compare the characters, not the pointer address
            return false;
        str1++;
        str2++;
    }
    return *str1 == *str2;   //Compare last characters to see if equal (zero bytes)
}

int main()
{
    char a[10] = "Bryan";
    char b[10] = "Bryan";
    if (strequal(a,b))
        cout << "They're the same guy!\n"; //this will be true and thus is printed
}
```

e. 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.)

→ptr is only holding the pointer address to the first element in the array (that is, 5). However, the actual array itself (anArray) is considered unimportant after the getPtrToArray function has finished execution. It is then considered unimportant, and the system might possibly overwrite it and/or write over it with the values of junk during execution of function f. Thus the original array values may very well be lost.

```
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;
}
```

Nathan Tung
Com Sci 31 (Lec 2)
November 21, 2011

2. 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 `cat` that can point to a variable of type double.

      ```
      double* cat;
      ```

   b. Declare `mouse` to be a 5-element array of doubles.

      ```
      double mouse[5];
      ```

   c. Make the `cat` variable point to the last element of `mouse`.

      ```
      cat = mouse+4;
      ```

   d. Make the double pointed to by `cat` equal to 17, using the * operator.

      ```
      *cat = 17;
      ```

   e. Without using the `cat` pointer, and without using square brackets, set the fourth element (i.e., the one at index 3) of the `mouse` array to have the value 42.

      ```
      *(mouse+3) = 42;
      ```

   f. Move the `cat` pointer back by three doubles.

      ```
      cat -= 3;
      ```

   g. Using square brackets, but without using the name `mouse`, set the third element (i.e., the one at index 2) of the `mouse` array to have the value 33.

      ```
      cat[1] = 33;
      ```

   h. Without using the `*` operator, but using square brackets, set the double pointed to by `cat` to have the value 25.

      ```
      cat[0] = 25;
      ```

   i. Using the `*` operator in the initialization expression, declare a bool variable named `b` and initialize it to true if the double pointed to by `cat` is equal to the double immediately following the double pointed to by `cat`, and false otherwise.

      ```
      bool b = (*cat==*(cat+1));
      ```

   j. Using the `==` operator in the initialization expression, declare a bool variable named `d` and initialize it to true if `cat` points to the double at the start of the `mouse` array, and false otherwise.

      ```
      bool d = (cat==&mouse[0]);
      ```

Nathan Tung
Com Sci 31 (Lec 2)
November 21, 2011

3.

a. 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.

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

Rewritten version:

```
double computeMean(const double* scores, int numScores)
{
	const double* ptr = scores; //alternative: we can use scores itself instead of ptr
	double total=0;
	int index=0;
	while(index<numScores)
	{
		total += *(ptr+index);
		index++;
	}
	return total/numScores;
}
```

b. 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`.

```
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;
}
```

Rewritten version:

```
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;
}
```

c. 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.

Rewritten version:

```
const char* findTheChar(const char* str, char chr)
{
	while(*str!='\0')
	{
		if(*str==chr)
			return str;
		str++;
	}
	return NULL;
}
```

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

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

int* maxwell(int* a, int* b) //Compares the int values held at pointer a and b
{
    if (*a > *b)
        return a; //Returns the pointer of the greater int
    else
        return b;
}

void swap1(int* a, int* b) //Swaps the address of pointer a with b; essentially does nothing
{                          //since they aren't passed by reference
    int* temp = a;
    a = b;
    b = temp;
}

void swap2(int* a, int* b) //Swaps the int at pointer a with that of pointer b
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Nathan Tung
Com Sci 31 (Lec 2)
November 21, 2011

```
int main()
{
    int array[6] = { 5, 3, 4, 17, 22, 19 };

    int* ptr = maxwell(array, &array[2]); //ptr is now array (or &array[0]), the bigger one
    *ptr = -1; //the value at ptr or array[0] is now -1
    ptr += 2; //ptr now points to the third element and equal to &array[2]
    ptr[1] = 9; //the value of the next element in the array (array[3]) is now 9
    *(array+1) = 79; //the pointer of &array[1] is dereferenced, and array[1] is now 79
    cout << &array[5] - ptr << endl; //pointer subtracted from pointer gives an integer
                                     //this is basically 5-2=3, so 3 is printed out in a line
    swap1(&array[0], &array[1]); //attempts to swap addresses of the first two array indexes
                                     //but nothing happens since they aren't passed by reference
    swap2(array, &array[2]); //swaps the values at array[0] and array[2]

    for (int i = 0; i < 6; i++)
        cout << array[i] << endl; //all values of the array are then printed out in lines
}
```

→The final array looks something like {4, 79, -1, 9, 22, 19}, and thus each value is printed out in its own line, according to the for-loop and cout code.

```
int* ptr = maxwell(array, &array[2]);
```
→{5, 3, 4, 17, 22, 19} ptr points to index 0

```
*ptr = -1;
```
→{-1, 3, 4, 17, 22, 19} ptr points to index 0

```
ptr += 2;
```
→{-1, 3, 4, 17, 22, 19} ptr points to index 2

```
ptr[1] = 9;
```
→{-1, 3, 4, 9, 22, 19} ptr points to index 2

```
*(array+1) = 79;
```
→{-1, 79, 4, 9, 22, 19} ptr points to index 2

```
cout << &array[5] - ptr << endl;
```
→{-1, 79, 4, 9, 22, 19} ptr points to index 2

```
swap1(&array[0], &array[1]);
```
→{-1, 79, 4, 9, 22, 19} ptr points to index 2; 5-2=3 (3 is printed out)

```
swap2(array, &array[2]);
```
→{4, 79, -1, 9, 22, 19} ptr points to index 2

```
for (int i = 0; i < 6; i++)
    cout << array[i] << endl;
```
→{4, 79, -1, 9, 22, 19} ptr points to index 2; elements printed out in lines

Final output:

3
4
79
-1
9
22
19

Nathan Tung
Com Sci 31 (Lec 2)
November 21, 2011

5. Write a function named `removeS` that accepts one character pointer as a parameter and returns no value. The parameter is a C string. This function must remove all of the upper and lower case 's' letters from the string. The resulting string must be a valid C string.

   Your function must declare no more than one local variable in addition to the parameter; that additional variable must be of a pointer type. Your function must not use any square brackets and must not use the `strcpy` library function.

```
int main()
{
    char msg[50] = "She'll be a massless princess.";
    removeS(msg);
    cout << msg;  // prints   he'll be a male prince.
}



void removeS(char str[]) //take in a C string named str
{
       char* c = str; //initialize a pointer to the first element of str
       for(; *str!='\0'; str++) //as long as the string hasn't ended, loop and increment str
              if(toupper(*str)!='S') //if dereferenced str (the element) is 'S' or 's'
                     *c++=*str; //concatenate it to the pointer c, then increment the pointer
       *c='\0'; //once all the non-S or non-s chars have been seen, end string with a zero byte
}
```