**CS31 Project 6**

**Xuanhe Zhang (706078980)**

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

Bugs:

1. We should add 1 to the pointer and then dereference it to access the value we want to change.
2. In the while loop, we should print out the last value first and then minus 1.
3. The order of the array is wrong, it should be the reversed order of the output.

Fixed Code:

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

// add 1 to the pointer first then dereference it to set right value

ptr += 2;

ptr[0] = 30; // set arr[2] to 30

while (ptr >= arr)

{

cout << \*ptr << endl; // print values

ptr--; //the order of abstraction was wrong

}

}

1. **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 can know the location of that item. Explain why this function won't do that, and show a way 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.**

If we want to change the array out of the method, we should pass them by reference, not pass by value.

Since pass by value will give us a copy of that array.

Fixed Code:

void findMax(int arr[], int n, int\*& pToMax)

{

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 = &nums[0];

findMax(nums, 4, ptr);

cout << "The maximum is at address " << ptr << endl;

cout << "It's at position " << ptr - nums << endl;

cout << "Its value is " << \*ptr << endl;

}

1. **The computeCube function is correct, but the main function has a problem. Explain why it may not work and show a way to fix it. Your fix must be to the main function only; you must not change computeCube in any way.**

We must allocate a memory address to the pointer to initialize it first.

Fixed Code:

void computeCube(int n, int\* ncubed)

{

\*ncubed = n \* n \* n;

}

int main()

{

int i; //we need to assign a memory address to the pointer

int\* ptr = &i;

computeCube(5, ptr);

cout << "Five cubed is " << \*ptr << endl;

}

1. **The strequal function is supposed to return true if and only if its two C string arguments have exactly same text. Explain what the problems with the implementation of the function are, and show a way to fix them.**

We should compare the value (dereference) of the pointer, not the pointer itself.

// return true if two C strings are equal

bool strequal(const char str1[], const char str2[])

{

while (\*str1 != 0 && \*str2 != 0) // zero bytes at ends

{

if (\*str1 != \*str2) // compare corresponding characters

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time?

}

int main()

{

char a[15] = "Chang";

char b[15] = "Zhang";

if (strequal(a,b))

cout << "They're the same person!\n";

}

1. **This program is supposed to write 100 99 98 3 2 1, but it probably does not. What is the program doing that is incorrect? (We're not asking you explain why the incorrect action leads to the particular outcome it does, and we're not asking you to propose a fix to the problem.)**

The anArray defined in getPtrToArray() is a local variable. Its scope is limited in the getPtrToArray(). After the program executes the getPtrToArray(), it will free up this part of memory. If we call the f(), this part will be overwritten.

1. **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). For each part, do not use any variable names not mentioned in that part (e.g., if the part doesn't mention cat, do not use cat in your answer).**

double\* cat;

//Declare a pointer variable named cat that can point to a variable of type double.

double mouse[5];

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

cat = &mouse[4];

//Make the cat variable point to the last element of mouse.

\*cat = 25;

//Make the double pointed to by cat equal to 25, using the \* operator.

\*(mouse + 3) = 54;

//Without using the cat pointer, and without using square brackets, set the fourth element

//(i.e., the one at position 3) of the mouse array to have the value 54.

cat -= 3;

//Move the cat pointer back by three doubles.

cat[1] = 42;

//Using square brackets, but without using the name mouse, set the third element

//(i.e., the one at position 2) of the mouse array to have the value 42. (You may use cat.)

cat[0] = 17;

//Without using the\* operator or the name mouse, but using square brackets, set the double pointed to by cat to have the value 17.

bool d = (cat == mouse);

//Using the == operator in the initialization expression,

//declare a bool variable named d and initialize it with an expression that evaluates to true

//if cat points to the double at the start of the mouse array, and to false otherwise.

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

//Using the \* operator in the initialization expression,

//declare a bool variable named b and initialize it with an expression that evaluates to true

//if the double pointed to by cat is equal to the double immediately following the double pointed to by cat, and to false otherwise.

//Do not use the name mouse.

**3.**

1. **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 mean(const double\* scores, int numScores)

{

const double\* ptr = scores;

double tot = 0;

int i = 0;

while (i < numScores)

{

tot += \*(ptr + i);

i++;

}

return tot / numScores;

}

1. **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. Do not use any of the <cstring> functions such as strlen, strcpy, etc.**

// 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 nullptr (not found).

const char\* findTheChar2(const char\* str, char chr)

{

for (int k = 0; \*str != 0; k++) {

if (\*(str + k) == chr)

return str + k;

}

return nullptr;

}

1. **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. Do not use any of the <cstring> functions such as strlen, strcpy, etc.**

const char\* findTheChar3(const char\* str, char chr)

{

while (\*str != 0) {

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

1. **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.**

Output:

3

4

79

-1

9

22

19

int\* maxwell(int\* a, int\* b)

{

if (\*a > \*b)

return a;

else

return b;

// return the pointer with larger value

}

void swap1(int\* a, int\* b)

{

int\* temp = a;

a = b;

b = temp;

// do nothing, the swap is limited in function’s scope

// a and b just copy the value of address of the pointer

}

void swap2(int\* a, int\* b)

{

int temp = \*a;

\*a = \*b;

\*b = temp;

// swap two values of the pointers

}

int main()

{

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

// create array

int\* ptr = maxwell(array, &array[2]);

// call and get the pointer from maxwell();

// compare array[0] and array[2] and return the larger one's pointer

// now ptr points to array[0]

\*ptr = -1;

// set -1 to the value that ptr points to

// now array[0] == -1;

ptr += 2;

// move pointer ptr to next 2 positions

// now ptr points to array[2];

ptr[1] = 9;

// assign 9 to the next position of current pointer

// now array[3] == 9;

\*(array + 1) = 79;

// set 79 to the next position of array[0]

// now array[1] == 79

cout << &array[5] - ptr << endl;

// &array[5] is the memory address of array[5]

// now ptr points to array[2]

// print out the difference of two positions’ index, 5 - 2 = 3

swap1(&array[0], &array[1]);

// the function is passed by value

// do nothing, the swap is limited in function’s scope

swap2(array, &array[2]);

// swap two values of array[0] and array[2]

// now array[2] == -1;

// now array[0] == 4;

for (int i = 0; i < 6; i++)

cout << array[i] << endl;

// now array contains elements: {4, 79, -1, 9, 22, 19}

// print out each value of array

}

1. **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 any of the <cstring> functions such as strlen, strcpy, etc.**

int main()

{

char msg[50] = "She'll shave a massless princess.";

removeS(msg);

cout << msg; // prints he'll have a male prince.

}

void removeS(char\* ptr)

{

char\* str = ptr; //let str to point the same position with ptr

while (\*ptr != 0) { //while loop until the value of ptr is 0

while (\*ptr == 'S' || \*ptr == 's') {

//while the value that ptr points to is 'S' or 's'

ptr++; //move pointer to point next position

}

\*str = \*ptr; //assign value of ptr to the position that str points to

str++; //move str to point next position

ptr++; //move ptr to point next position

}

\*str = 0; //let the value that str points equal to 0 to end the C string

}