1. The subparts to this problem involve errors in the use of pointers.
   1. 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:

int main()

{

int arr[3] = { 5, 10, 15 };

int\* ptr = arr;

\*ptr = 30; // set arr[0] to 30

\*(ptr + 1) = 20; // set arr[1] to 20

ptr += 2;

\*ptr = 10; // set arr[2] to 10

ptr -= 2;

while (ptr <= arr+2)

{

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

ptr++;

}

}

* 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 knows that item’s location. 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.
     + This function will not work properly because exiting the function, the pointer disappears.

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;

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.
     + The main function pointer does not work due to the pointer not being initialized.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int s = 7;

int\* ptr = &s;

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.
     + The strings cannot simply be compared; one must compare every single character using string compare (strcmp) or by going through every single character. Additionally, the c strings are constants and thus cannot be edited (adding one to the string). Instead you add one to the k and compare the string at that character space from then. If the two strings are equal then you would return 0 or false. So we want to change this to an if statement or a simple true. Lastly, in the main function, in the if statement, the strings being equal have to result in something with == to proceed to the cout.

// return true if two C strings are equal

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

{

int k = 0;

while (str1[k] != '\0' && str2[k] != '\0')

{

if (str1[k] != str2[k])// compare corresponding characters

return false;

k++; // advance to the next character

}

return true; // both ended at same time?

}

int main()

{

char a[15] = "Zhao";

char b[15] = "Zhou";

if (strequal(a,b) == true)

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 problem with this function is that the array was created locally in the function and when it exits the function, the array also disappears alongside it.

#include <iostream>

using namespace std;

int\* getPtrToArray(int& m)

{

int anArray[100];

for (int j = 0; j < 100; j++)

anArray[j] = 100-j;

m = 100;

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 < 3; i++)

cout << ptr[i] << ' ';

for (int i = n-3; i < n; i++)

cout << ptr[i] << ' ';

cout << endl;

}

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

double\* cat = nullptr;

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

double mouse[5];

* 1. Make the cat variable point to the last element of mouse.

cat = mouse + 4;

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

\*cat = 25;

* 1. 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 42.

\*(mouse + 3) = 42;

* 1. Move the cat pointer back by three doubles.

cat -= 3;

* 1. 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 54.

cat[1] = 54;

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

cat[0] = 17;

* 1. 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(double\* b)

{

if (\*b == \*(b + 1))

return true;

else

return false;

}

* 1. 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(double\* d)

{

if (d == mouse)

return true;

else

return false;

}

* 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)

{

double tot = 0;

for (int k = 0; k < numScores; k++)

tot += \*(scores + k);

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

{

for (int k = 0; \*(str + k) != '\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\* findTheChar(const char str[], char chr)

{

const char \* pointer = str;

while ( \*pointer != '\0' )

{

if (\*pointer == chr)

return pointer;

else

pointer++;

}

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.

#include <iostream>

using namespace std;

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

{

if (\*a > \*b) //If the value that is being pointed at at position a is larger than b, return the pointer to a. Else return the pointer to b.

return a;

else

return b;

}

void swap1(int\* a, int\* b) //This switches the pointers of a and b and where they point but this only works within the function.

{

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int\* b) //This switches the integer values that pointers a and b point to.

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = maxwell(array, &array[2]); // This compares the array at position 0 and the array at position 2 via pointers. Comparing 5 and 4, it determines 5 is larger and points to position zero (five).

\*ptr = -1; //This sets the array at position zero to -1. Result: {-1, 3, 4, 17, 22, 19}.

ptr += 2; // This increments the pointer two forward so that it points at the array at position 2 <- array[2].

ptr[1] = 9; // This sets the array at position 3 to 9. This is because the pointer points at array[2]=4 and thus adding one to that gives us array[3]=17. Result: {-1, 3, 4, 9, 22, 19}.

\*(array+1) = 79; //This sets the array+1 position or array[1]=3 to 79.

Result: {-1, 79, 4, 9, 22, 19}.

cout << &array[5] - ptr << endl; //This prints out the array[5] – array[2] = 3 on a separate line because it subtracts two pointers, giving the integer value that represents how many int spaces these two arrays are.

swap1(&array[0], &array[1]); //This swaps pointers within the function a and b to point at one another respectively. However this disappears when it exits the function.

swap2(array, &array[2]); //This switches the values that a and b point to. Thus the array at position 0 and position 2 get switched. Result: {4, 79, -1, 9, 22, 19}.

for (int i = 0; i < 6; i++) //This prints out every position of the array on a separate line.

cout << array[i] << endl;

}

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.

char removeS(char change[])

{

int whereZero = 0;

char\* changeptr = change;

for (int k = 0; \*(changeptr + k) != '\0'; k++)

{

whereZero++;

}

for (int k = 0; \*(changeptr + k) != '\0'; k++)

{

if (\*(changeptr + k) == 'S' || \*(changeptr + k) == 's')

{

for (int j = k + 1; \*(changeptr + j) != '\0'; j++)

\*(changeptr + j - 1) = \*(changeptr + j);

whereZero--;

k--;

}

}

\*(changeptr + whereZero) = '\0';

return;

}

int main()

{

char msg[50] = "She’ll be a massless princess.";

removeS(msg);

cout << msg; // prints he’ll be a male prince.

}