**1.**

1. **This program is supposed to write  30 20 10, but it doesn't. 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 = 30; // set arr[2] to 30

while (ptr >= arr)

{

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

ptr--;

}

cout << endl;

}

1. **The findDisorder function is supposed to find the first item in an array that is less than the element preceding it, and set the p parameter to point to that item, so the caller can know the location of that item. 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 the main routine below in any way, yet as a result of your fixing the function, the main routine below must work correctly.**

The main function does not initialize the pointer, so we don’t know where the pointer is pointing to. This means that the pointer needs to be initialized by the end of the findDisorder function, but since the pointer p is not passed by reference, it won’t be modified outside of the findDisorder function and will remain uninitialized in the main function.

void findDisorder(int arr[], int n, int\* &p)

{

for (int k = 1; k < n; k++)

{

if (arr[k] < arr[k-1])

{

p = arr + k;

return;

}

}

p = nullptr;

}

int main()

{

int nums[6] = { 10, 20, 20, 40, 30, 50 };

int\* ptr;

findDisorder(nums, 6, ptr);

if (ptr == nullptr)

cout << "The array is ordered" << endl;

else

{

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

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

cout << "The item's value is " << \*ptr << endl;

}

}

1. **The hypotenuse 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 the hypotenuse function in any way.**

The pointer is not initialized in the main function, and since the pointer parameter in the hypotenuse function holds a local copy of pointer p from the main function, any changes made to the pointer does not carry over to the main function.

#include <iostream>

#include <cmath>

using namespace std;

void hypotenuse(double leg1, double leg2, double\* resultPtr)

{

\*resultPtr = sqrt(leg1 \* leg1 + leg2 \* leg2);

}

int main()

{

double k;

double\* p = &k;

hypotenuse(1.5, 2.0, p);

cout << "The hypotenuse is " << \*p << endl;

}

1. **The match 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 function match takes in an array which is taken in the same way a pointer is. Keeping this in mind, we see that instead of comparing the actual value of the array elements we are comparing the location of the elements, which is not what we want.

// return true if two C strings are equal

bool match(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[10] = "pointy";

char b[10] = "pointless";

if (match(a,b))

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

}

1. **This program is supposed to write 1 4 9 16 25 36 49 64 81 100 , 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 method computeSquares is trying to return the address of a local variable to outside of the function, resulting in undefined behavior.

**2.**

1. **Declare a pointer variable named fp that can point to a variable of type string.**

string\* fp;

1. **Declare fish to be a 5-element array of strings.**

string fish[5];

1. **Make the fp variable point to the last element of fish.**

fp = &fish[4];

1. **Make the string pointed to by fp equal to "yellowtail", using the \* operator.**

\*fp = “yellowtail”;

1. **Without using the fp pointer, and without using square brackets, set the fourth element (i.e., the one at position 3) of the fish array to have the value "salmon".**

\*(fish+3) = “salmon”;

1. **Move the fp pointer back by three strings.**

fp -= 3;

1. **Using square brackets, but without using the name fish, set the third element (i.e., the one at position 2) of the fish array to have the value "basa". (You may use fp.)**

fp[1] = “basa”;

1. **Without using the \* operator or the name fish, but using square brackets, set the string pointed to by fp to have the value "sole".**

fp[0] = “sole”;

1. **Using the == operator in the initialization expression, declare a bool variable named d and initialize it with an expression that evaluates to true if fp points to the string at the start of the fish array, and to false otherwise.**

bool d = (fp == fish);

1. **Using the \* operator in the initialization expression, but no square brackets, declare a bool variable named b and initialize it to true if the string pointed to by fp is equal to the string immediately following the string pointed to by fp, and false otherwise.**

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

**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 computeAverage(const double\* scores, int nScores)

{

const double\* ptr = scores;

double tot = 0;

int counter = 0;

while (counter < nScores)

{

tot += \*(ptr+counter);

counter++;

}

return tot / nScores;

}

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)

{

while (\*str != 0)

{

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

**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.**

When following the path of the variable ptr, it is set equal to minimart(array, &array[2]). This function returns the smaller of the two values, and since it is comparing the value at index 0 (5), to the value at index 2, (4), ptr is set equal to &array[2]. The next change to ptr is ptr+=2. This moves the pointer two elements to the right so it is at &array[4]. The difference between &array[4] and &array[5] is one, so the line cout << "diff=" << &array[5] - ptr << endl prints diff=1.

The next lines print the updated array. The first change to it sets ptr[1] = 9, which is equivalent to &array[3] because ptr[0] is equal to &array[2]. Here, the array is { 5, 3, 4, 9, 22, 19 }.

The variable ptr then increments two elements (ptr+=2), landing at &array[4], and sets the value to -1. The next line (\*(array+1) = 79) takes the first element of the array, adds 1, giving us &array[1], and then sets the value at that element to 79. Here, the array is now { 5, 79, 4, 9, -1, 19 }.

The first swap function is called, where the addresses of the first two elements are swapped (&array[0] and &array[1]). The values remain the same. The second swap switches the values of the first element of the array (array) and the third element of the array (&array[2]). So we end up with { 4, 79, 5, 9, -1, 19 }.

The following for loop goes through the array, and prints each element on a new line.

The final result is:

diff=1

4

79

5

9

-1

19

**5.**

**Write a function named deleteG 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 'g' 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. Do not use any of the <cstring> functions such as strlen, strcpy, etc.**

void deleteG(char\* msg)

{

char\* temp = msg;

while (\*msg != 0)

{

if (\*msg != 'g' && \*msg != 'G')

{

\*temp = \*msg;

temp++;

}

msg++;

}

\*temp = 0;

}

int main()

{

char msg[100] = "I recall the glass gate next to Gus in Lagos, near the

gold bridge.";

deleteG(msg);

cout << msg; // prints I recall the lass ate next to us in Laos, near

the old bride.

}