1a.

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 (error, illegal syntax)

ptr += 2;

**ptr[0] = 30;**  // set arr[2] to 30 (error, does not set arr[2] to 30)

while (ptr >= arr)

{

ptr--;

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

}

cout << endl;

}

*//Solution:*

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*

ptr ++;

**while (ptr > arr)**

{

ptr--;

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

}

cout << endl;

}

b. The problem with the *findDisorder* function is that the pointer variable *ptr* is passed by value. Thus, when the program returns to *main*, the value of *ptr* is unchanged and equals the null pointer. The fix is to pass *ptr* by reference so that when *findDisorder* returns to *main*, the changes made to *p* in *findDisorder* apply to *ptr* in *main*.

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

c. This program does not work because the pointer variable *p* defined in *main* is declared but not initialized, so it does not point to anything. Thus, when *p* is passed into *hypotenuse* and *resultptr*, a copy of *p*, is dereferenced to get a *double* value, there is no place for this *double* value to be stored. The fix to this issue is to initialize *p* in *main* so that it points to a *double* instead of *nullptr*.

#include <iostream>

#include <cmath>

**using** **namespace** std;

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

{

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

}

**int** main()

{

**double** result = 0;

**double**\* p = &result;

hypotenuse(1.5, 2.0, p);

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

}

d. This function isn’t actually comparing each individual character, but is instead comparing the addresses of the array in memory. Thus, it will always return false. The way to fix this is to pass the function pointers to the arrays. The program can then dereference the pointers to check for the zero byte and dereference the pointers to compare individual characters. Finally, the program dereferences the pointers for a final comparison and a return value.

*// 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] = "pointy";

**if** (match(a, b))

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

}

e. The problem with the program is that the *computeSquares* function is returning the address of the stack of memory associated with the integer array *arr*. The values in the stack of memory are mutable -- in fact, the integer array *junk* created in function *f* may occupy this stack of memory. Even though the pointer variable *ptr* gets the address of the first element in *arr*, the values stored in the memory allocated for *arr* end up changing. The program fails because it relies on undefined behavior and prints out random values.

2.

string\* fp;

string fish [5];

fp = &fish[4];

\*fp = "yellowtail";

\*(fish + 3) = "salmon";

fp -= 3;

fp[1] = "sole";

fp [0] = "eel";

**bool** d = (fp == &fish[0]);

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

3a.

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

{

**const** **double**\* ptr = scores;

**double** tot = 0;

**for** (**int** i = 0; i < nScores; i++)

tot += \*(ptr + i);

**return** tot/nScores;

}

b.

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

{

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

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

**return** (str + i);

**return** **nullptr**;

}

c.

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

{

**for** (;\*str != 0; str++)

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

**return** str;

**return** **nullptr**;

}

4.

*/\*The program prints:*

***diff=1***

***4***

***79***

***5***

***9***

***-1***

***19\*/***

*//Program:*

#include <iostream>

**using** **namespace** std;

**int**\* minimart(**int**\* a, **int**\* b) *//returns lesser dereference value between the pointer to array[0] and the pointer to array[2], which is &array[2].*

{

**if** (\*a < \*b)

**return** a;

**else**

**return** b;

}

**void** swap1(**int**\* a, **int** \*b) *//swaps addresses of array[0] and array[1], but has no effect on output of program.*

{

**int**\* temp = a;

a = b;

b = temp;

}

**void** swap2(**int**\* a, **int** \*b) *//swaps values at array[0] and array[2].*

{

**int** temp = \*a;

\*a = \*b;

\*b = temp;

}

**int** main()

{

**int** array[6] = { 5, 3, 4, 17, 22, 19 }; *//initializes an integer array of 6 elements.*

**int**\* ptr = minimart(array, &array[2]); *//initializes a pointer to an integer as &array[2].*

ptr[1] = 9; *//ptr[1] = array[3] = 9*

ptr += 2; *//ptr = &array[4]*

\*ptr = -1; *//\*ptr = array[4] = -1*

\*(array+1) = 79; *//array[1] = 79*

cout << "diff=" << &array[5] - ptr << endl; *//prints the difference between the address of the last element in array (array[5]) and the current address of the variable that ptr points to, or array[4]. Thus, prints "diff = 1".*

swap1(&array[0], &array[1]); *//swaps addresses of array[0] and array[1].*

swap2(array, &array[2]); *//swaps values at array[0] and array[2].*

**for** (**int** i = 0; i < 6; i++) *//prints values of array, separated by a new line: 4, 79, 5, 9, -1, 19.*

cout << array[i] << endl;

}

5.

**void** deleteG(**char**\* msg)

{

**char**\* result = msg; *//char pointer result points to first char, 'I'.*

**for** (;\*msg != 0; msg++) *//increment char pointer msg to each element in array msg until null byte.*

{

**if** (\*msg == 'G' || \*msg == 'g') *//if the value that msg points to is a G/g, keep incrementing through char array*

**continue**;

**else** *//if not a G/g, the value that result points to gets the value that msg points to and result now points to the next element in msg.*

{

\*result = \*msg;

result++;

}

}

\*result = 0; *//terminate the c-string result with a null byte.*

}