**1a.** This program is supposed to write 30 20 10, one per line, 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 = 30;**

**\*ptr + 1 = 20;**

**//doesn’t assign value like this because what this is saying is the (value of ptr) + 1 = 20 which is not assignable since it would be something like**

**30 + 1 = 20;, originally it was intended to set the next element to 20**

**ptr += 2;**

**ptr[0] = 10;**

**while (ptr >= arr)**

**{**

**ptr--;**

**//changing position of the pointer before printing it which causes the wrong numbers to be printed out**

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

**}**

**}**

**//FIXED VERSION**

**//to make minimal changes, just swapped the order of 30 and 10**

**//changed the way to assign value in arr[1]**

**//switched the order of when the ptr decrements in the while loop**

**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) {**

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

**ptr--;**

**}**

**}**

**1b.** 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.

* This function won’t do what it is meant to do because pToMax is a copy and local variable in the findMax() function so the value doesn’t get changed in the main function

**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;**

**}**

**//FIXED VERSION**

**//**Now pToMax is a reference of int pointer and is able to change the value of the pointer in the main function

**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);**

**//pToMax is a copy not a reference, doesn’t change it actually**

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

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

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

**}**

**1c.** 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.

* int\* ptr; is not initialized so the pointer doesn’t know what memory location to point to

**void computeCube(int n, int\* ncubed)**

**{**

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

**}**

**int main()**

**{**

**int\* ptr;**

**computeCube(5, ptr);**

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

**}**

**//FIXED VERSION**

**void computeCube(int n, int\* ncubed)**

**{**

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

**}**

**int main()**

**{**

**int n = 0; //creates a memory location to assign pointer to**

**int\* ptr = &n; //assigned ptr to memory location of n**

**computeCube(5, ptr);**

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

**}**

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

**// return true if two C strings are equal**

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

**{**

**while (str1 != 0 && str2 != 0) // doesn’t check value of str1 and str2**

**{**

**if (str1 != str2) // doesn’t check value of str1 and str2**

**return false;**

**str1++; // advance to the next character**

**str2++;**

**}**

**return str1 == str2; // doesn’t check value of str1 and str2**

**}**

**int main()**

**{**

**char a[15] = "Chen, G.";**

**char b[15] = "Chen, Y.";**

**if (strequal(a,b))**

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

**}**

**//FIXED VERSION**

**// return true if two C strings are equal**

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

**{**

**while (\*str1 != 0 && \*str2 != 0) // checks the values with the \***

**{**

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

**return false;**

**str1++; // advance to the next character**

**str2++;**

**}**

**return \*str1 == \*str2;**

**// if one is null and other is a character returns false, otherwise both null returns true so we know if they both ended at the same time**

**}**

**int main()**

**{**

**char a[15] = "Chen, G.";**

**char b[15] = "Chen, Y.";**

**if (strequal(a,b))**

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

**}**

**1e.** 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 to 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 is caused by the getPtrToArray function. anArray is a local variable in the getPtrToArray function. Since it is a local variable and is used as an argument (nochange(anArray)), the pointer ptr, ends up pointing to an unexpected address and ends up outputting values from random memory addresses.

**#include <iostream>**

**using namespace std;**

**int\* nochange(int\* p)**

**{**

**return p;**

**}**

**int\* getPtrToArray(int& m)**

**{**

**int anArray[100];**

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

**anArray[j] = 100-j;**

**m = 100;**

**return nochange(anArray);**

**}**

**void f()**

**{**

**int junk[100];**

**for (int k = 0; k < 100; k++)**

**junk[k] = 123400000 + k;**

**junk[50]++;**

**}**

**int main()**

**{**

**int n;**

**int\* ptr = getPtrToArray(n);**

**f();**

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

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

**for (int i = 0; i < 3; i++)**

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

**cout << endl;**

**}**

**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). 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).

* 1. Declare a pointer variable named cat that can point to a variable of type double.
     1. double\* cat;
  2. Declare mouse to be a 5-element array of doubles.
     1. double mouse[5];
  3. Make the cat variable point to the last element of mouse.
     1. cat = &mouse[4];
  4. Make the double pointed to by cat equal to 25, using the \* operator.
     1. \*cat = 25;
  5. 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 17.
     1. \*(mouse + 3) = 17;
  6. Move the cat pointer back by three doubles.
     1. cat -= 3;
  7. 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.)
     1. cat[1] = 42;
  8. Without using the \* operator or the name mouse, but using square brackets, set the double pointed to by cat to have the value 54.
     1. cat[0] = 54;
  9. 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
     1. bool d = (cat == &mouse[0]);
     2. AND/OR
     3. bool d = (cat == mouse);
  10. Using the \* operator in the initialization expression, but no square brackets, 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.
      1. bool b = (\*cat == \*(cat +1));

**3a.** 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;**

**while (ptr != scores + numScores)**

**{**

**tot += \*ptr;**

**ptr++;**

**}**

**return tot/numScores;**

**}**

**//FIXED VERSION**

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

**{**

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

**double tot = 0;**

**int visit = 0; //int variable to visit each double in the array**

**while (visit < numScores)**

**{**

**tot += \*(ptr + visit);**

**visit++; //doesn’t increment ptr**

**}**

**return tot/numScores;**

**}**

**3b.** 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;**

**}**

**//FIXED 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 nullptr;**

**}**

**3c.** 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.

**//FIXED VERSION**

**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.

**#include <iostream>**

**using namespace std;**

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

**{**

**if (\*a > \*b)**

**return a;**

**else**

**return b;**

**}**

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

**{**

**int\* temp = a;**

**a = b;**

**b = temp;**

**}**

**//a and b are copies, you are swapping addresses of the copies, doesn’t change anything in the main function**

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

**{**

**int temp = \*a;**

**\*a = \*b;**

**\*b = temp;**

**}**

**//changing the value of a and b and this gets reflected in the main, even though it is still a copy, it holds memory address and changes value**

**int main()**

**{**

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

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

**//passes in memory address of first element of array (5) and third element (4)**

**//ptr’s memory location is now set to first element of the array after the function is ran through**

**\*ptr = -1; //ptr is setting value of array[0] to -1**

**ptr += 2; //ptr points to the third element of array now**

**ptr[1] = 9; //ptr sets fourth element of array to 9**

**\*(array+1) = 79; //the value of the second element of array is now 79**

**//{-1, 79, 4, 9, 22, 19}**

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

**//&array[5] gives you that location and ptr points to &array[2] so when you subtract the two memory addresses, 5-2 = 3**

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

**//doesn’t switch the 1st and 2nd element’s value**

**//stays: {-1, 79, 4, 9, 22, 19}**

**swap2(array, &array[2]);**

**//switches 1st and 3rd element’s values**

**//{4, 79, -1, 9, 22, 19}**

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

**cout << array[i] << endl;**

**//prints out all the elements of the array {4, 79, -1, 9, 22, 19}**

**The program prints out the following:**

**3**

**4**

**79**

**-1**

**9**

**22**

**19**

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

**void removeS(char\* ch){**

**char\* point = ch;**

**while(\*point != '\0'){**

**if(\*point == 'S' || \*point == 's'){**

**while(\*point != '\0'){**

**\*point = \*(point + 1);**

**point++;**

**}**

**point = ch;**

**} else {**

**point++;**

**}**

**}**

**}**

**int main()**

**{**

**char msg[50] = "She'll blossom like a massless princess.";**

**removeS(msg);**

**cout << msg; // prints he'll bloom like a male prince.**

**}**