Calvin Liu

Project 6

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[0] = 10; // set arr[2] to 10

ptr = arr;

while (ptr <= (arr + 2))

{

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

ptr++;

}

}

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 how 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.

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;

}

This would not do the job because you need to change pToMax outside the function and thus, need to pass it by reference by adding an &.

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);

}

}

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

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int arr[1];

int\* ptr = arr;

computeCube(5, ptr);

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

}

It won’t work because you need to initialize a pointer to something.

The strequal function is supposed to return true if and only if its two C string arguments have exactly same text. What are the problems with the implementation of the function, and how can they be fixed?

// return true if two C strings are equal

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

{

const char\* ptr1 = str1;

const char\* ptr2 = str2;

while (\*ptr1 != '\0' && \*ptr2 != '\0')

{

if (\*ptr1 != \*ptr2) // compare corresponding characters

return false;

ptr1++; // advance to the next character

ptr2++;

}

return \*ptr1 == \*ptr2; // both ended at same time?

}

int main()

{

char a[15] = "Chen, B.";

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

if (strequal(a,b))

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

}

The problem with this function is that you are comparing the reference of str1 and str2. If you want to compare the c string then you must use pointers to point to each element of the array. Another problem was that it was originally trying to increase the array with ++, but you cannot increase the spot when talking about the array, but you can increase the spot a pointer is pointing to. You can then compare what they are pointing to with \*.

This program is supposed to write 5 4 3 2 1, but it probably does not. What is the problem with this program? (We're not asking you to propose a fix to the problem.)

int\* getPtrToArray(int& m)

{

int anArray[5] = { 5, 4, 3, 2, 1 };

m = 5;

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 < n; i++)

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

cout << endl;

}

The problem with this program is that the array inside the function is lost after the function passes n through it.

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;

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 17, using the \* operator.

\*cat = 17;

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 33.

cat[1] = 33;

1. Without using the \* operator, but using square brackets, set the double pointed to by cat to have the value 25.

cat[0] = 25;

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

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 = (cat == mouse);

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 k = 0;

while (ptr != scores + numScores)

{

tot += \*(ptr + k);

k++;

}

return tot/numScores;

}

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.

// 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 NULL (not found).

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

{

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

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

return (ptr + k);

return NULL;

}

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.

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

{

for (; \*ptr != ‘\0’; ptr++)

if (\*ptr == chr)

return ptr;

return NULL;

}

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.

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

{

if (\*a > \*b)

return a; returns a pointer

else

return b;

}

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

{

int\* temp = a; //pass by value, swaps the address numbers

a = b;

b = temp;

}

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

{

int temp = \*a; //pass by ref (array), swaps the elements

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = maxwell(array, &array[2]); //returns &array[0]

\*ptr = -1; //[0] now becomes -1

ptr += 2; //now pointing at [2]

ptr[1] = 9; //17 is now 9

\*(array+1) = 79; //3 is now 79

cout << &array[5] - ptr << endl; //prints 3

swap1(&array[0], &array[1]); // swaps [0] and [1]’s copy’s addresses but the function ends and nothing really happens because it isn’t pass by reference

swap2(array, &array[2]); // swaps element [0] with element [2]

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

cout << array[i] << endl; // print the whole array which is now (3, 4, 79, -1, 9, 22, 19)

}

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 the strcpy library function.

void removeS(char\* cstr)

{

char\* ptr;

for(ptr = cstr; \*ptr != '\0'; ptr++)

{

if(\*ptr == 's' || \*ptr == 'S')

{

for(; \*ptr != '\0'; ptr++)

{

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

}

ptr = cstr;

}

}

}

int main()

{

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

removeS(msg);

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

}

* 1. First, because of the operators' order of precedence, the expression \*ptr + 1 = 20 means (\*ptr) + 1 = 20. The expression (\*ptr) + 1 evaluates to the int 31, not an int variable that can be assigned to. When corrected to \*(ptr+1) = 20, the expression means \*(&arr[1]) = 20, which means arr[1] = 20.

Second, the while loop doesn't access arr[2], and tries to access arr[-1]. One possible fix is

for (ptr = arr; ptr < arr + 3; ptr++) cout << \*ptr << endl;

* 1. findMax puts the correct value in pToMax, but pToMax is a *copy* of the caller's variable ptr, so findMax has no effect on ptr. The parameter pToMaxmust be passed by reference, not by value:

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

* 1. The declaration int\* ptr; declares ptr to be a pointer to int, but leaves it uninitialized — it does not point to any particular int. That uninitialized pointer is copied into the parameter ncubed. In the expression \*ncubed = n\*n\*n, the attempt to dereference the uninitialized ncubed pointer leads to undefined behavior. A fix would be to make sure computeCube is passed a valid pointer; one possibility is, in the main routine, to say:

**int k;** int\* ptr **= &k**;

* 1. The test str1 != 0 is asking if the str1 pointer itself has a value different from the null pointer. (The integer constant 0 used in a context where a pointer is required means the null pointer.) The test we want, though, is to see if the character *pointed to* by str1 is different from the zero byte that marks the end of a C string. (The same applies to str2.)

Similarly, the test str1 != str2 is asking whether those two pointers have different values (i.e., they point to different places). But what should be tested is whether the characters they *point to* have different values. (The same applies to str1 == str2.)

The corrected function body is thus

while (**\***str1 != '\0' && **\***str2 != '\0') // 0 instead of '\0' is also OK { if (**\***str1 != **\***str2) return false; str1++; str2++; } return **\***str1 == **\***str2;

* 1. The storage for the local variable anArray goes away when the function getPtrToArray returns. But that function returns a pointer to that storage. Attempting to follow that pointer in the main routine (implied by ptr[i]) yields undefined behavior.
  2. double\* cat;
  3. double mouse[5];
  4. cat = &mouse[4];   or   cat = mouse + 4;
  5. \*cat = 17;
  6. \*(mouse + 3) = 42;
  7. cat -= 3;
  8. cat[1] = 33;
  9. cat[0] = 25;
  10. bool b = \*cat == \*(cat+1);
  11. bool d = cat == mouse;   or   bool d = cat == &mouse[0];
  12. double mean(const double\* scores, int numScores) { **int k = 0;** double tot = 0; while (**k != numScores**) { tot += **\*(scores + k)**; **k++;** } return tot/numScores; }
  13. const char\* findTheChar(const char**\* str**, char chr) { for (int k = 0; **\*(str+k)** != 0; k++) if (**\*(str+k)** == chr) return **str + k**; return NULL; }
  14. const char\* findTheChar(const char\* str, char chr) { for ( ; **\*str** != 0; **str++**) if (**\*str** == chr) return **str**; return NULL; }

1. 3 *(see note 4 below)* 4 *(see notes 1, 5, and 6 below)* 79 *(see notes 3 and 5 below)* -1 *(see note 6 below)* 9 *(see note 2 below)* 22 19
   1. maxwell is called with pointers to array[0] and array[2]. It returns a pointer to whichever of the ints pointed to has a larger value. Since array[0] has the larger value, the function returns &array[0]. The expression \*ptr = -1 sets array[0] to -1.
   2. ptr[i] = 9; sets array[3] to 9.
   3. \*(array+1) = 79; sets array[1] to 79.
   4. &array[5] - &array[2] is 3.
   5. The swap1 function swaps its *copies* of the pointers passed in to it, with no effect on the ints pointed to.
   6. The swap2 function swaps the ints pointed to.
2. void removeS(char\* source) { char\* destination = source; for ( ; \*source != '\0'; source++) { if (\*source != 's' && \*source != 'S') { \*destination = \*source; destination++; } } \*destination = '\0'; // Don't forget the zero byte at the end }

or

void removeS(char\* source) { char\* destination = source; while (\*source != '\0') { if (\*source != 's' && \*source != 'S') { \*destination = \*source; destination++; } source++; } \*destination = '\0'; // Don't forget the zero byte at the end }