**Original:**

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 (Incorrect syntax)

ptr += 2;

ptr[0] = 10; // set arr[2] to 10 (Does not set arr[2] to 10)

while (ptr >= arr) (Condition is reversed)

{

ptr--; (Should come after the cout)

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

}

}

**Solution:**

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

while (ptr <= arr + 2)

{

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

ptr++;

}

}

1. The findMax function will not work in the way we want it to because the pointer pToMax is currently passed by value, not passed by reference. When it is passed by value, the function only creates a copy of the pointer pToMax. It really should be passed by reference so that the variable can be accessed outside of the findMax function. To fix this problem, we can change the pointer pToMax to be passed by reference such that:

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

1. It may not work because the pointer is uninitialized, meaning there is no variable for the pointer to point to. This can lead to undefined behavior as the pointer isn’t pointing to anything defined in the memory. We can fix this problem by creating an integer variable in which the pointer would point to and then properly assigning a pointer to that variable. Within “int main” we can make the following changes:

int main()

{

int x;

int\* ptr = &x;

computeCube(5, ptr);

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

}

1. The problem with this function is that it is not correctly comparing each individual character between str1 and str2. In the original program, the function is comparing the actual address of str1 and str2. What we want to check is whether the characters str1 and str2 point to are different from one another. We can fix this by implementing pointers into the program whenever we need to check the individual characters. This would allow us to check if the characters str1 and str2 point to are different from one another. Within the function “strequal”, we can make the following changes:

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

{

while (\*str1 != 0 && \*str2 != 0)

{

if (\*str1 != \*str2)

return false;

str1++;

str2++;

}

return \*str1 == \*str2;

}

1. The incorrect part of the program comes in when it tries to access the variable anArray, through a pointer, after the function finishes. When the function, “getPtrToArray” is done, the memory inside the variable anArray is gone and set to random values. Thus, when the function tries to return a pointer to anArray, this function would then produce undefined behavior.
   1. double\* cat;
   2. double mouse[5];
   3. cat = &mouse[4];
   4. \*cat = 25;
   5. \*(mouse + 3) = 54;
   6. cat -= 3;
   7. cat[1] = 17;
   8. cat[0] = 42;
   9. bool d = cat == &mouse[0];
   10. bool b = \*cat == \*(cat + 1);
2. double mean(const double\* scores, int numScores)

{

double tot = 0;

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

{

tot += \*(scores + i);

}

return tot/numScores;

}

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

{

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

{

if (\*str == chr)

{

return str;

}

}

return nullptr;

}

Step 1: An array of 6 ints called “array” is declared and initialized.

Step 2: The pointer ptr is declared and initialized through the function maxwell. The maxwell function returns a pointer to the int that has a greater value between a or b. In this case, a and b are array and array[2] respectively which represent array[0] and array[2]. Array[0] has the larger value so ptr points to and returns &array[0].

Step 3: Set \*ptr, or array[0], to -1.

Step 4: The actual pointer ptr is moved up 2 elements to &array[2].

Step 5: Set the next element in array to 9, meaning set array[3] to 9.

Step 6: \*(array + 1) is the same as array[0+1] so set array[1] to 79.

Step 7: Print &array[5] – &array[2], which is equal to 3, and then create a newline

**OUTPUT:**

**3**

Step 8: swap1 function is called but, it only swaps the local pointers and does not impact the actual ints. Thus, it has no effect on the outcome of the program.

Step 9: The function swap2 is called and swap2 successfully swaps ints a and b using pointers. In this case swap2 swaps array[0] and array[2], which are -1 and 4

Step 10: The array now looks like { 4, 79, -1, 9, 22, 19 }. Iterate through the array and print out each value of the array on a newline

**OUTPUT:**

**3**

**4**

**79**

**-1**

**9**

**22**

**19**

void removeS(char\* initial)

{

char\* final = initial;

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

{

if (\*initial != ‘S’ && \*initial != ‘s’)

{

\*final = \*initial;

final++;

}

}

\*final = ‘\0’;

}