1. Fix
2. Before modifying:

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 the mistake is that it shouldn’t be \*ptr + 1 to modify the value that the pointer ptr pointing to; instead, it should be \*(ptr +1)

ptr += 2;

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

while (ptr >= arr). //the mistake is that it shouldn’t be ptr >= arr, it should be (ptr <= &arr[2])

{

ptr--;// in that case, it should be ptr ++ and should be later thatn the cout sentence.

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

}

}

**After modifying:**

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++;

}

}

1. Before modifying:

The error is that the function does not pass the pointer address to the function, the ToMax pointer can only be accessed in the findMax function instead of the main function. Therefore we should modify it by passing reference of the pointer to the function to modify the pointer’s original value.

void findMax(int arr[], int n, int\* pToMax). // the function does not pass the pointer address to the function, thus it will not change the maximum value

{

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;

}

**After modifying:**

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;

}

1. Before modifying:

The problem is that the program does not initialize the pointer’s pointing address, therefore leaving a null pointer and result in undefined behavior error when compiling.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int\* ptr;

computeCube(5, ptr);

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

}

**After modifying:**

void computeCube(int n, int \*ncubed)

{

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

}

int main()

{

int a = 5;

int \*ptr = &a;

computeCube(5, ptr);

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

}

1. Before modifying:

The function shouldn’t use str1 != 0 && str2 != 0 to determine whether there are zero bytes at ends or not. The function should compare the item that the pointers are pointing to instead of comparing the pointers themselves. Therefore, it should be \*str1 != '\0' && \*str2 != '\0'.

// return true if two C strings are equal

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

{

while (str1 != 0 && str2 != 0) // zero bytes at ends

{

if (str1 != str2) //if we want to compare corresponding characters, we should get the str that the pointer pointing to intead of comparing pointers. Therefore, it should be (\*str1 != \*str2)

return false;

str1++; // advance to the next character

str2++;

}

return str1 == str2; // check if the strings that the two pointers pointing to are the same. Therefore, it should use \*str1 == \*str2.

}

int main()

{

char a[15] = "Wang, A.";

char b[15] = "Wang, R.";

if (strequal(a,b))

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

}

**After modifying:**

bool strequal(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[15] = "Wang, A.";

char b[15] = "Wang, R.";

if (strequal(a, b))

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

}

1. What is the program doing that is incorrect:

The anArray[100] is only a local variable that can be accessed and changed only in the getPtrToArray function, and will be abandoned outside the function. Therefore, there will generate some random numbers when the system call the ptr pointers since the areas that the pointers point no longer contain the numbers we want any more.

1. Problem:
2. 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 25, using the \* operator.

\*cat = 25;

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 54.

\*(mouse + 3) = 54;

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 17. (You may use cat.)

Cat[1] = 17

1. Without using the \* operator or the name mouse, but using square brackets, set the double pointed to by cat to have the value 42.

cat[0] = 42;

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 cat points to the double at the start of the mouse array, and to false otherwise.

bool d = (cat == mouse);

1. Using the \* operator in the initialization expression, 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.

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

1. Fix:
2. After modifying:

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

{

double tot = 0;

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

{

tot += \*(scores +i);

}

return tot / numScores;

}

1. after modifying:

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

{

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

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

return (str + k);

return nullptr;

}

1. after modifying:

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

{

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

if (\*str == chr)

return str;

return nullptr;

}

1. explain:

#include <iostream>

using namespace std;

int \*maxwell(int \*a, int \*b) //given the int pointer a and int pointer b

{

if (\*a > \*b) //if the int that pointer a is pointing to is greater than the int that pointer b is pointing to

return a; //then return pointer a

else //if the int that pointer a is pointing to is smaller or eaqual than the int that pointer b is pointing to

return b; //then return pointer b

}

//swap1 does not switch the value within a and b's pointers

void swap1(int \*a, int \*b) //given int pointer a and int pointer b

{

int \*temp = a; //let the int pointer temp pointing to the same address as pointer a

a = b; //let a's address is equal to b's address

b = temp; // let b's address is equal to previous a's address

}

void swap2(int \*a, int \*b)//given int pointer a and int pointer b

{

int temp = \*a; //let int temp equal to the int a is pointing to

\*a = \*b; // let the int that pointer a is pointing to equal to the int that pointer b is pointing to

\*b = temp; //let the int that pointer b is pointing to equal to temp(previous int value that pointer a is pointing to)

}

int main()

{

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

int \*ptr = maxwell(array, &array[2]); // to get the address of the larger number between array[0] and array[2]

\*ptr = -1; //let the array[0] be -1

ptr += 2; //move to array[2]

ptr[1] = 9; //change array[3] to 9

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

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

swap1(&array[0], &array[1]); // switch array[0] and array[1]'s address, but their value hasn't been switched

swap2(array, &array[2]); //switch the value of array[0] and array[2]

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

cout << array[i] << endl; // would print 4,79, -1, 9, 22, 19

}

1. fix:

void removeS(char \* ptr)

{

char \* temp = ptr;

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

{

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

{

continue;

}

\*temp = \*ptr;

temp ++;

}

\*temp = '\0';

}

int main()

{

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

removeS(msg);

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

}