1. There is one problem in the line of code \*ptr + 1 because the \* has higher precedence, which would mean the object the ptr points to plus one = 20 would not make sense. It is necessary to put parenthesis around the ptr + 1.

The second error is in the while loop, which prints out 10 20 30 instead of 30 20 10. To fix that, you can change the address where the 30 20 and 10 are inputted and decrement ptr after the cout statement in the while loop.

Revised code:

int main()

{

int arr[3] = { 5, 10, 15 };

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 30

\*(ptr + 1) = 20; // set arr[1] to 20

ptr += 2;

ptr[0] = 30; // set arr[2] to 10

while (ptr >= arr)

{

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

ptr--;

}

}

1. The findMax function does not work because when ptr is passed to the findMax function, it is uninitialized. The changes made to the pointer in the function are done to its copy, leaving the pointer uninitialized and therefore creating an error. One way to fix that is passing the pointer by reference so the changes made are to the actual pointer, not the copy and the actual pointer is initialized, no only the copy.

Revised code:

void findMax(int arr[], int n, int\*& pToMax) //1b

{

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;

}

1. The main function has a problem because the pointer ptr is not initialized to any address, so the pointer does not point to anything. The pointer needs to be initialized, and to do this you can initialize an int x and set it to 0, then initialize the address of the ptr to the address of x.

Revised code:

void computeCube(int n, int\* ncubed) //1c

{

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

}

int main()

{

int x = 0;

int\* ptr = &x;

computeCube(5, ptr);

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

}

1. The problem with the implementation of the function is that the statements such as (str1 != str2) are comparing the addresses of the two c strings, instead of the characters. One way to fix this is comparing what the str1, and the str2 are pointing to. In the entire function, mostly the addresses are being used instead of the pointees, which is what the function should do.

Revised code:

bool strequal(const char str1[], const char str2[]) //1d

{

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

{

if (\*str1 != \*str2) // compare corresponding char’s

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time?

}

int main()

{

char a[15] = "Zhou";

char b[15] = "Zhu";

if (strequal(a, b))

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

}

1. What the program is doing wrong is that it is initializing the array in the functions, which makes them local variables, so the values in them are gone once the function is completed. This puts a bunch of garbage values in the array.
2. (a) double\* cat;

(b) double mouse[5];

(c) cat = mouse + 4;

(d) \*cat = 25;

(e) \*(mouse + 3) = 42;

(f) cat -= 3;

(g) cat[1] = 54;

(h) cat[0] = 27;

(i) if (\*cat == \*(cat + 1))

bool b = true;

else

bool b = false;

(j) if (cat == mouse)

bool d = true;

else

bool d = false;

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

{

const double\* ptr = scores;

double tot = 0;

for(int k = 0; k < numScores; k\_)

{

ptr = ptr + k;

tot += \*ptr;

}

return tot/numScores;

}

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

{

for (int k = 0; \*str != ‘\0’; k++)

{

str = str + k;

if (\*str == chr)

return &str;

}

return nullptr;

}

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

{

while(\*str != ‘\0’)

{

if (\*str == chr)

return &str;

str++;

}

return nullptr;

}

1. The first output statement will print out a 3, which is the memory address of index 5 in the array subtracted from the memory address of index 2 of the array, since the ptr points to index 2 now. & produces the address of array[5] and the ptr is the address, and since the difference in the two addresses is by 3 positions, the difference in the two addresses is 3.

The second output statement is: 4 -1 79 9 22 19 ,with each integer on a new line. The elements in index 0, 1 and 3 are changed by the pointer. The pointer is initialized to the head of the array and then there is a statement \*ptr = -1; which changes index 0 to the value -1. Then the pointer is incremented by 2, now pointing to index 2. So ptr[1] would be the index 3 of the array, whose value is changed to 9. Array has the address of the head of the array, which is the 0 index, so array + 1 would point to index 1 and its pointee is changed to 79. Then the two swap methods are called. The first swap will not change the values in anything since it is trying to swap the addresses of the elements. This does not change anything, as you can not just swap the values of elements by changing the addresses. The 2nd swap function will swap the 2 values in index 0 and 2, since the function refers to the object the pointer points to.

Output:

3

4

-1

79

9

22

19

void removeS(char c[])

{

char\* temp = c;

cout << "String in removeS is " << c;

while (\*temp != NULL)

{

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

{

for (temp; \*temp != NULL; temp++)

\*temp = \*(temp + sizeof(char));

cout << "Initializing temp = c" << endl;

temp = c;

}

else

temp++;

}

}

int main()

{

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

removeS(msg);

cout << msg << endl;

}