1. Solve errors

1. 30, 20, 10

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 (this won’t work)

ptr += 2;

ptr[0] = 10; // set arr[2] to 10 (ptr isn’t the whole array

//therefore it won’t set a 0th index

while (ptr >= arr) // if you want to print it forwards, it needs to //be less than, and increment upwards.

{

ptr--;

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

}

}

Correct:

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) = 30; //set arr[2] to 10

while (ptr < arr + 3)

{

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

ptr++;

}

}

1. findMax

The findMax function won’t set the parameter inputted in pToMax because the pointer is passed by value, not reference. To fix it, the findMax function should look like this:

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;

}

}

1. computeCube

The ptr, first of all, is passed by reference, so the function doesn’t save anything to memory when it gets passed to the computeCube function. A way to fix this is to make the pointer point to a reference to an int that has been assigned the value 5, and use that as the input parameter for computeCube.

int main()

{

int n = 5;

int \*ptr = &n;

computeCube(5, ptr);

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

}

1. strequal

The function strequal compares the entire array using Boolean comparisons instead of actually comparing each character of the C string. To fix this, you can implement the function using pointers to the C strings instead of directly passing the C strings by reference. This way, you can compare individual characters. Here’s a correct implementation:

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

{

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

{

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

return false;

str1++; // advance to the next character

str2++;

}

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

}

1. 100 99 98 3 2 1

When the function getPtrToArray returns the pointer to anArray, it only keeps the first index of the array in memory (that the pointer is pointing to), and the rest of the array is set to garbage values. So, when it tries to print out 100 99 98 3 2 1, it can only print out 100 correctly and then prints out garbage values.

2. Write a single C++ statement that performs the task

1. double\* cat;
2. double mouse[5];
3. cat = mouse + 4;
4. \*cat = 25;
5. \*(mouse + 3) = 54;
6. cat -= 3;
7. cat[2] = 27;
8. cat[1] = 42;
9. bool b = (\*cat == \*(cat+1));
10. bool d = (cat == &mouse[0]);

3.

a. Rewrite so it returns the same result but doesn’t increment ptr

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

{

const double\* ptr = scores;

double tot = 0;

int iter = 0;

while (iter < numScores)

{

tot += \*(ptr + iter);

iter++;

}

return tot/numScores;

}

b. Rewrite function without square brackets

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;

}

4. What does the program print and why

#include <iostream>

using namespace std;

int\* maxwell(int\* a, int\* b) // returns a pointer to whichever value in the

//references a and b are greater

{

if (\*a > \*b)

return a;

else

return b;

}

void swap1(int\* a, int\* b) // swaps the pointers to a and b but doesn’t

// do anything

{

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int\* b) //swaps the values in a and b

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = maxwell(array, &array[2]); // compares 5 and 4. Since

// 5 is greater than 4, Maxwell returns a

// pointer to position 0 of array

\*ptr = -1; // sets the 0th index of array to be -1

ptr += 2; // ptr now points at the 2nd index

ptr[1] = 9; //sets array[3] to be 9 instead of 17

\*(array+1) = 79; //sets array[1] to be 79 instead of 3

//array is now: {-1, 79, 4, 9, 22, 19}

cout << &array[5] - ptr << endl; //prints 3. This is because

// ptr is equivalent to &array[2] right now

// &array[5] - &array[2] is the same as 5-2=3

swap1(&array[0], &array[1]); // swaps the addresses of the two

// but doesn’t have an effect on the output

swap2(array, &array[2]); //swaps the 0th and 2nd index of array.

//array is now {4, 79, -1, 9, 22, 19}

for (int i = 0; i < 6; i++) // prints out the above commented array

//which is 4 79 -1 9 22 19, each in a new line.

cout << array[i] << endl;

}

Output:

3

4

79

-1

9

22

19

5. Write a function named removeS

void removeS(char\* str) {

char\* newstr = str;

while(\*str != 0) {

if(\*str == ‘S’ || \*str == ‘s’) {

str++;

} else {

\*newstr = \*str;

newstr++;

str++;

}

}

\*newstr = 0;

}