Andrew Sun

006159649

1a.

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

ptr = arr;

while (ptr < arr + 3)

{

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

ptr++;

}

}

1b.

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 issue with the original program was that the original pointer was localized and was a copy of the pointer imputed into the function (similar to how a int or double would be copied). This means that any alterations to the pointer and its address stays within the function and does not carry over into the main function. As a result, the way to fix this problem would be to make the function a reference.

1c.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int\* ptr = new int ;

computeCube(5, ptr);

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

}

The issue with the main program is that the pointer does not actually point to anything within the main function. We can fix this by setting the pointer equal to a new int such that it can actually can be edited by the function and a value will be stored.

1d.

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] = "Chen, G.";

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

if (strequal(a,b))

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

}

The issue with the program was that the function checks the actual pointers and not the char that it is pointing to. As a result, we can solve this issue by making some of the pointers point to the chars such that proper comparisons can be made.

1e.

The issue with the program is that the array that getPtrToArray creates gets destroyed at the end of the function, leading to undefined behaviors when the pointer is once again accessed in the main function. As a result, the program outputs random numbers and does not do what it is intended to do.

2a.

//2a.

double\* cat;

//2b.

double mouse[5];

//2c.

cat = &mouse[4];

//2d.

\*cat = 25;

//2e.

\*(mouse + 3) = 17;

//2f.

cat -= 3;

//2g.

\*(cat + 1) = 42;

//2h.

cat[0] = 54;

//2i.

bool d = (cat == &mouse[0]);

//2j.

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

3a.

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

{

const double\* ptr = scores;

double tot = 0;

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

{

tot += \*(ptr + i);

}

// while (ptr != scores + numScores)

// {

// tot += \*ptr;

// ptr++;

// }

return tot/numScores;

}

3b.

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;

}

3c.

const char\* findTheChar2(const char\* str, char chr)

{

while (\*str != 0)

{

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

4.

#include <iostream>

using namespace std;

int\* maxwell(int\* a, int\* b) //takes in 2 pointers, checks and returns the pointer with a larger value.

{

if (\*a > \*b)

return a;

else

return b;

}

void swap1(int\* a, int\* b) //swaps the two pointers. A temp pointer is created which is then used to swap pointers a and b

{

int\* temp = a;

a = b;

b = temp;

}

void swap2(int\* a, int\* b) //swaps the value that the two pointers have, same position. A temp int is used to swap the two values that a and b holds

{

int temp = \*a;

\*a = \*b;

\*b = temp;

}

int main()

{

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

int\* ptr = maxwell(array, &array[2]); //new pointer pointing at value 5 array[0], this is because maxwell returns the pointer with a larger value, and 5>4 so ptr points to array0

\*ptr = -1; //value of ptr becomes -1

ptr += 2; //ptr position increases by 2, now points to array2 with a value of 4

ptr[1] = 9; //the int that is after ptr is now set to 9, so array3 is now 9

\*(array+1) = 79; //array[1] = 79, since array position starts at array0, array+1 would point to array1

cout << &array[5] - ptr << endl; //subtraction of positions between array and ptr, which would be 3

swap1(&array[0], &array[1]); //swaps the two pointers of array0 and array1, since the pointers created are copies, nothing happens to the actual array

swap2(array, &array[2]); //value of array0 and array2 are swapped

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

cout << array[i] << endl; //couts every int in the array

}

5.

void removeS (char\* ptr)

{

while (\*ptr != 0)

{

char\* temp = ptr;

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

while (\*ptr != 0) {

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

ptr++;

}

ptr = temp - 1;

}

ptr++;

}

}