Bonnie Liu

005300989

CS 31 Discussion 1F

Project 6: What's the Point?

1a) One bug in the original program was the line \*ptr + 1 = 20; This is not allowed because the program is told to access a value from the pointer, add 1 to it, and then set it to 20, but the computer is not allowed to set values to each other. What the programmer intended to do was set the integer variable after the pointer to 20. We can do this by putting in parenthesis around ptr + 1, as shown in the corrected version of the program below. The program also had a logic error in the while loop. The line ptr--; should come after printing the values. The original program (after fixing the syntax error mentioned above) printed 20 30 and some garbage integer value, one per line. To print 30 20 10, one per line, we need to access the elements of the array properly without going out of bounds, and this can be done using the method shown below.

int main()

{

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

int\* ptr = arr;

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

// I made changes here

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

ptr += 2;

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

while (ptr >= arr)

{

// I made changes here

cout << arr[2-(ptr-&arr[0])] << endl; // print values

ptr--;

}

}

1b) This function will not do that because in the function, the parameter pToMax is passed by copy. This means that when the program calls the function findMax, a copy of the pointer ptr is made, which means the function cannot change ptr. To fix this problem, we can pass pToMax by reference. In the function declaration, we can write int\*& pToMax instead of int\* pToMax, as shown below.

// I made changes here

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;

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;

}

1c) This program may not work because the pointer ptr is uninitialized in the main function. This means that when the program tries to access \*ptr when it tries to print out the result of five cubed, there will be an error because the computer doesn’t know what to print.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

// I made changes here

int num = 5;

int\* ptr = &num;

computeCube(num, ptr);

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

}

1d) The problem with the implementation of this function is that the function is not accessing and comparing the elements of the array properly; instead, it is comparing the memory addresses of the two arrays. A solution to this problem is shown below:

// return true if two C strings are equal

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

{

// I made changes here

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?

}

int main()

{

char a[15] = "Chen";

char b[15] = "Cheng";

if (strequal(a, b))

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

}

1e) The problem with this program is that anArray gets destroyed after completing the function, so the pointer tries to access something that doesn’t exist anymore. The pointer points at garbage values.

2a) double\* cat;

2b) double mouse[5];

2c) cat = &mouse[4];

2d) \*cat = 25;

2e) mouse[3] = 54;

2f) cat -= 3;

2g) cat[1] = 27;

2h) cat[0] = 42;

2i) bool b = (\*cat == \*(cat + 1));

2j) bool d = (cat == &mouse[0]);

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);

}

return tot / numScores;

}

3b)

const char\* findTheChar(const char\* ptr, char chr) {

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

if (\*(ptr + i) == chr)

return ptr + i;

}

return nullptr;

}

3c)

const char\* findTheChar(const char\* ptr, char chr) {

while (\*ptr != '\0') {

if (\*ptr == chr)

return ptr;

ptr++;

}

return nullptr;

}

4) The program prints 3 4 79 -1 9 22 19, one per line.

The function maxwell returns the pointer that points to the greater value. If the values are equal, it returns the second parameter by default. Since array[0], which equals 5, is greater than array[2], which equals 4, we return the pointer pointing to array[0]. Consequently, ptr points to array[0].

The line \*ptr = -1; sets the value pointed to by ptr to -1. This means array[0] is now -1. Then we increment ptr by 2, which moves the pointer to the next 2 integers. In this case, ptr now points at array[2]. The line ptr[1] = 9; sets the integer coming immediately after the one ptr is pointing to to 9. \*(array + 1) = 79; changes the value of array[1] to 79 because array is a pointer to array[0]. Therefore, \*(array + 1) = 79; accesses the value of the array that comes immediately after array[0] and sets it to 79.

The program starts off by printing 3 because &array[5] – ptr is the distance between the pointer pointing to the array at position 5 and wherever the pointer ptr is pointing to at the moment. In this case, ptr points to the array at position 2 due to the line ptr += 2;, which was run a couple of lines before. The program then prints all the elements of the array in order. The function swap1 did not do anything to the array because pointers are passed by value rather than passed by reference, so swap1 made changes to the copies of the original pointers. The function swap1 also did not attempt to manipulate the values pointed to by the pointers. Thus, in the main program, both the pointers and the values the pointers were pointing to in the array were unaffected. The function swap2, however, did swap the values of the array at position 0 and 2. Unlike swap1, swap2 accesses the values pointed to by the pointers, so it was able to successfully manipulate the array.

5)

void removeS(char\* ptr) {

while (\*ptr != '\0') {

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

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

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

}

ptr--;

}

ptr++;

}

}