Reema Kshetramade

SID: 205328402

Project 6 Homework

1.

a. 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 -= 2;

while (ptr < arr + 3)

{

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

ptr++;

}

}

b. This function does not work because the pointer is passed by value, so the changes made to the parameter ptr are only in findMax and not in main so in main, ptr remains uninitialized and in all the cout statements ptr does not have the value of pToMax determined in findMax. This is fixed by passing the pointer by reference.

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;

}

c. The pointer ptr in main is not pointing to anything, and computeCube is trying to change the value of the object ptr is pointing to which is undefined behavior.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

int x = 0;

int\* ptr = &x;

computeCube(5, ptr);

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

}

d. All the statements that are meant to be comparing characters in the strings are actually comparing the values of the pointers, not the values of the objects that they point to, so the function is not actually comparing the C strings.

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?

}

int main()

{

char a[15] = "Chen";

char b[15] = "Cheng";

if (strequal(a,b))

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

}

e. The array created by getPtrToArray only exists in that function, so although the function returns a pointer to where that array would be, when that pointer is used to print values from the array, there is no longer that array stored in memory starting at the address in the pointer.

2.

a. double\* cat;

b. double mouse[5];

c. cat = mouse + 4;

d. \*cat = 25;

e. \*(mouse + 3) = 54;

f. cat -= 3;

g. cat[1] = 27;

h. cat[0] = 42;

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

f. bool d = (cat == mouse);

3.   
 a.

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

{

double tot = 0;

int i = 0;

while (scores + i != scores + numScores)

{

tot += \*(scores + i);

i++;

}

return tot/numScores;

}

b.

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;

}

c.

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

{

while (\*str != 0)

{

if (\*str == chr)

return str;

str++;

}

return nullptr;

}

4.

3

4

79

-1

9

22

19

The first line of output is 3 because ptr is first set to the address of the max value between the value array points to (the first value in array which is 5) and the value at the address of array[2] (the value of array[2] which is 4). The max value between these two is 5 so ptr points to the first value in array. 2 is later added to ptr so it points to array[2]. For the first line of output, ptr is equal to address of the element at index 2 in array so &array[5]- ptr is equal to the distance between the elements at index 5 and 2 of array which is equal to 3.

While ptr is still pointing to the first value in array, the value that ptr points to is then set to -1 so array holds [-1, 3, 4, 17, 22, 19]. Then 2 is added to ptr so that it now points to array[2]. Then element at index 1 of the array starting at the address of ptr, meaning the element of array[3], is set to 9 so array now holds [-1, 3, 4, 9, 22, 19]. Then the value that (array + 1) points to, meaning array[1], is set to 79 so array now holds [-1, 79, 4, 9, 22, 19]. The first line of output does not change array or ptr. The call to swap1 switches the addresses of the elements at index 0 and 1 of array, however, these changes are only present within the function and have no effect on array in main. The call to swap2 exchanges the values at the addresses of the elements at index 0 and 2 in array so array now holds [4, 79, -1, 9, 22, 19]. The for loop after this loops through every item in array and prints them out on a new line.

5.

void removeS(char\* str)

{

char\* ptr;

while (\*str != 0)

{

ptr = str;

if (\*str == ‘s’ || \*str == ‘S’)

{

while (\*ptr != 0)

{

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

ptr++;

}

}

else

str++;

}

}