Frank She

Professor Smallberg

CS 31

26 November 2012

Project 6 Report

1.

a. Fixed Version:

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

ptr=arr;

while (ptr <= &arr[2])

{

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

ptr++;

}

}

b. The findMax function has a parameter, int\* pToMax, that accepts a pointer to an int, however the main function does not initialize the pointer, ptr, before calling findMax. So, to fix the problem, we have to pass ptr by reference rather than by value in order to properly initialize the pointer.

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;

}

}

c. The main function does not work because it does not initialize ptr to anything. The program then tries to use the uninitialized pointer, which is undefined behavior.

Fixed Version:

int main()

{

int n;

int\* ptr=&n;

computeCube(5,ptr);

cout<< \*ptr<<endl;

}

d. This implementation does not work because: the while condition compares the address of the strings to 0, the if statement compares the addresses not the characters at those addresses, and the return statement at the end will not check if one string is longer than the other. To fix this, we need to compare the values stored at the addresses str1 and str2 to make sure they are not the null character, also we need to check those values against each other to check if they are equal or not. To solve the problem of ending at the same time, we change the while condition to || so that we check if the strings end at the same time. Finally, we return true if the strings ended at the same time and were equal. The zeroes in the while condition are considered null characters, because they are in a place where a char is expected.

Fixed Version:

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 true; // both ended at same time?

}

e. The problem with this program is that it creates an array within a function that is not the main function, so once the function is finished being called, the memory that the array elements once occupied can be reused and their values can be overwritten.

2.

a. double \*cat;

b. double mouse[5];

c. cat=&mouse[4];

d. \*cat=17;

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

f. cat -= 3;

g. cat[1]=33;

h. cat[0] = 25;

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

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

3.

a. Rewritten:

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

{

int k =0;

double tot = 0;

while (k<numScores)

{

tot += \*(scores+k);

k++;

}

return tot/numScores;

}

b. Rewritten:

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

{

for (int k = 0; \*(str+k) != '\0'; k++)

if (\*(str+k) == chr)

return (str+k);

return NULL;

}

c. Rewritten:

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

{

for (;\*str != '\0';str++)

if (\*str == chr)

return str;

return NULL;

}

4. This program prints out the numbers 3,4,79,-1,9,22,19.

Each integer was followed by a new line.

The first number printed is 3. This is because the main function calls the function maxwell, passing &array[0] and &array[2], and maxwell returns an int pointer, setting ptr to point to the value at &array[0] because the value at &array[0] was bigger than the value at &array[2]. Then, the statement ptr+=2; moved ptr to point to array[2]. Finally, 3 was printed as a result of the calculation &ærray[5]-ptr.

The following ints are all printed through a for loop, printing all elements of the array.

The next number printed is 4, the element at position 0. First, \*ptr=-1 set array[0] to -1. Also, array[2] was already initialized to 4. Then the program called swap2, which used a temp int to store an old value and then swapped the values of array[0] and array[2]. So array[0] became 4 and array[2] became -1.

The next number printed is 79, the element at position 1. The statement \*(array+1) = 79; set the value pointed to by &array[0+1] to 79.

The next number printed is -1, the element at position 2. This is occurred during the call to swap2, explained above.

The next number printed is 9, the element at position 3. This element is 9 because ptr first pointed to &array[0], then was added 2, making it point to &array[2]. Then, the statement ptr[1]=9; set array[3] to 9 because ptr looked at the position one in front of its current position. Since its current position was array[2], it set array[3] to 9.

The next number printed is 22, which was initialized during the array declaration.

The last number printed is 19, which was initialized during the array declaration.

It is important to notice that the call to function swap1 did not change the elements of the array.

5.

void removeS(char\* param)

{

for(;\*param!= '\0';){

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

for(char\* ptr= param; \*ptr!='\0';ptr++)

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

else

param++;

}

}