**Project 6: hw.doc YAMAN YUCEL UID:605704529**

**Part 1.a)** In this part, one can simply change the following labeled lines to get 30 20 10 one per line. However, if I need to set arr[0] = 30, arr[1] = 20, arr[2] = 10. Next solution can be used. Bugs are \*ptr + 1 and order of statements in while loop.

int main()

{

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

int\* ptr = arr;

\*ptr = 10; // set arr[0] to 30

\*(ptr + 1) = 20; // set arr[1] to 20 //BUG // FIX: ADDED A BRACKET ()

ptr += 2;

ptr[0] = 30; // set arr[2] to 10

while (ptr >= arr)

{

cout << \*ptr << endl; // print values //BUG //FIX: SWITCHED TWO STATEMENTS

ptr--; // ptr would be looking at the first element - 1 location

}

}

Or one can use the following if we are not allowed to change what a[0] a[1] a[2]’s values.

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 // ADDED A BRACKET ()

ptr += 2;

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

ptr -= 2;

while (ptr <= arr + 2)

{

cout << \*ptr << endl; // print values // SWITCHED TWO STATEMENTS

ptr++; // ptr would be looking at the last element + 1 location

}

}

**Part 1.b)** In this part we need to use a reference parameter to be able to change the pointer (pass by reference). Otherwise, copy of pointer will be created and we were not able to change the pointer passed to the function.

void findMax(int arr[], int n, int\*& pToMax) // We should add a reference parameter since we want variable to be changed.

{

if (n <= 0)

return; // no items, no maximum!

pToMax = arr; // both point to beginning

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 = &nums[0];

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;

}

**Part 1.c)** Uninitialized pointers are risky, since they can modify any memory location. Therefore, compilers does not allow you to call a function with an uninitialized pointer. There are two solutions to this problem, first one is initializing a new memory location, second one is initializing pointer with the address of another variable, however this will change the variable’s value. First one is open to memory leak, if we lose the address of that memory.

void computeCube(int n, int\* ncubed)

{

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

}

int main()

{

// PROBLEM pointer does not point any memory location and it is uninitialized.

//int n = 5;

//int\* ptr = &n;

int\* ptr = new int; // if you do not want n to be changed else uncomment above.

computeCube(5, ptr);

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

delete ptr;

}

**Part 1.d)** str1 and str2 points to the beginning of the arrays, therefore they are pointer. We need to access to the value when comparing, we should not compare the pointers, if we want to check whether they reached to the zero byte. I have changed 0 to '\0' since it is easier to read code, but it was unnecessary. While condition statement and if statement is changed such that values are compared. Also, at the end if str1 and str2 points to the zero byte, function must return true else false. So the values pointed by str1 and str2 should be compared, not the pointers.

// return true if two C strings are equal

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

{

while (\*str1 != '\0' && \*str2 != '\0') // zero bytes at ends '\0' stylistically better,// Need to compare characters not pointers

{

if (\*str1 != \*str2) // compare corresponding characters // Need to compare characters not pointers

return false;

str1++; // advance to the next character

str2++;

}

return \*str1 == \*str2; // both ended at same time? // Need to compare characters not pointers

}

int main()

{

char a[15] = "Chang";

char b[15] = "Zhang";

if (strequal(a, b))

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

}

**Part 1.e)** When the below code is run, meaningless values are printed to the screen because anArray[100] is declared inside the function getPtrToArray. When the program is done with the that function the array declared is deleted or removed. Therefore, ptr is not accessing any meaningful values. It is same for the junk[100] it is created inside the function and deleted after the execution of the function. Note that, value of m changes since it is passed by reference.

int\* nochange(int\* p)

{

return p;

}

int\* getPtrToArray(int& m)

{

int anArray[100]; // DECLARED INSIDE THE FUNCTION, WHEN YOU ARE DONE WITH FUNCTION ARRAY IS DELETED.

for (int j = 0; j < 100; j++)

anArray[j] = 100 - j;

m = 100;

return nochange(anArray);

}

void f()

{

int junk[100];

for (int k = 0; k < 100; k++)

junk[k] = 123400000 + k;

junk[50]++;

}

int main()

{

int n;

int\* ptr = getPtrToArray(n);

f();

for (int i = 0; i < 3; i++)

cout << ptr[i] << ' ';

for (int i = n - 3; i < n; i++)

cout << ptr[i] << ' ';

cout << endl;

}

**Part 2)**

int main()

{

double\* cat;// a

double mouse[5];// b

cat = mouse + 4;// c //same as &mouse[4];

\*cat = 25;// d

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

cat -= 3;// f

cat[1] = 42;// g //mouse[2] = 42, cat points to second element now

cat[0] = 17;// h

bool d = cat == mouse;// i

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

}

**Part 3.a)** New integer index is used to visit each double in scores array.

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

{

// no [], use an integer variable to visit each double in the array;

//const double\* ptr = scores; // we do not need ptr, since we only use the first index of the array.

double tot = 0;

int index = 0; // new integer that holds index

while (index != numScores)

{

tot += \*(scores + index);

index++;

}

// index is numScores now

return tot / numScores;

}

**Part 3.b)** all a[k] replaced with \*(a+k) and I return the pointer using only summation.

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;

}

**Part 3.c)** only str pointer and chr is used. Since str pointer is copied we can use it to traverse the array.

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

{

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

if (\*str == chr)

return str;

return nullptr;

}

**Part 4)** Although I have explained each line and wrote a summary at the end, following problem can be explained as follows. Maxwell function checks the values which are pointed a and b. Returns copy of a if a’s value is greater than b’s value else returns copy of b. swap1 does not do anything to the variables in the main function since a and b are copies and deleted after the execution of swap1. However, swap2 swaps the values that are pointed by a and b since it reaches the memory location. Copy of pointers are deleted but values remain and pointed by the pointers declared in main function. \*ptr = -1 sets first element of array to -1 since maxwell returned the address of the first position of the array. ptr+= 2 makes ptr point to 3rd element of array. ptr[1] = 9 sets 4rd element of the array to 9. \*(array + 1) = 79 sets 2nd element of the array to 79. cout << &array[5] - ptr << endl prints 3. ptr was pointing at the 3rd element so 6 - 3 = 3 since &array[5] returns the pointer that is pointing at the 6th element. Swap1 does not do anything, swap2 makes array[0] = 4, array[2] = -1. Array is now array = [ 4 79 -1 9 22 19].

Overall output is: 3 4 79 -1 9 22 19 one per line.

int\* maxwell(int\* a, int\* b)

{

if (\*a > \*b)

return a; // returns a if value that is pointed by a is greater than the value that is pointed by b.

else

return b; // returns b otherwise

}

void swap1(int\* a, int\* b)

{

int\* temp = a; // local copy a, b, temp.

a = b;

b = temp; // swaps copy a, copy b pointer // then a b is deleted.

}

void swap2(int\* a, int\* b)

{

int temp = \*a; // This function actually does the swap operation on values.

\*a = \*b;

\*b = temp; // value pointed by a is changed by the value pointed by b. value pointed by b is changed by the value pointed by a.

}

int main()

{

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

int\* ptr = maxwell(array, &array[2]); // IN THIS CASE PTR IS POINTING TO THE START [0]// returns the pointer that is pointing at the start of the array if the value pointed is greater than the value pointed by the 3 element of the array. Else returns the pointer that is pointing to the 3rd element of the array.

\*ptr = -1; //array[0] = -1;

ptr += 2; // pointer is pointing to the 3rd element.

ptr[1] = 9; // 4th element is 9. array[3] = 9;

\*(array + 1) = 79; //array[1] = 79

cout << &array[5] - ptr << endl; // 3 // ptr was pointing at the 3rd element so 6 - 3 = 3 since &array[5] returns the pointer that is pointing at the 6th element.

swap1(&array[0], &array[1]); // Does not do anything since copy of pointers are created. swaps copies and those copies are deleted afterwards

swap2(array, &array[2]); // this actually swaps the values at index 0 and index 2. so array[0] = 4, array[2] = -1;

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

cout << array[i] << endl; // array = [ 4 79 -1 9 22 19];

//5 was replaced by -1 with \*ptr = -1,

//3 was replaced by 79 with \*(array +1) = 79,

//4 was swapped with -1 after 5 was replaced by -1.

//17 was replaced by 9 with ptr[1] = 9.

//22 and 19 is unchanged.

}

**Part 5) Only msg and ch pointers are used to perform the task. The algorithm traverses through all C-string and when it finds a s or S, stores it’s position to msg and shifts left all characters from that s’s or S’s position to end. Continues to algorithm from the removed s position until pointer reaches to end.**

// This function removes all s and S from the msg array

void removeS(char\* msg)

{

char\* ch = msg; // additional variable of a pointer type. char\* can be only used,since we have only char values.

while (\*ch != 0) // Traverse through msg

{

if (\*ch == 's' || \*ch == 'S') // if the ch is s or S, shift elements that have greater index value one left including null byte

{

msg = ch; // save where you left to traverse to msg // msg points s to be removed. When it is removed you can treat remaining as a new msg which consists s and S.

while (\*ch != 0) //shift all elements to left by one until we reach to end.

{

\*ch = \*(ch + 1); // shifting

ch++; //preparing to shift next index

}

ch = msg; // restore where you left to proceed removing s's from remaining msg.

}

else

{

ch++; //just pass and point to next element since there is nothing interesting to do, leave char as it is.

}

}

}

int main()

{

//char msg[50] = "She'll shave a massless princess.slsdssldsoSSless";

char msg[50] = "She'll shave a massless princess.";

removeS(msg);

cout << msg; // prints he'll have a male prince.

}

**Version without comments:**

// This function removes all s and S from the msg array

void removeS(char\* msg)

{

char\* ch = msg;

while (\*ch != 0)

{

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

{

msg = ch;

while (\*ch != 0)

{

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

ch++;

}

ch = msg;

}

else

{

ch++;

}

}

}

int main()

{

char msg[50] = "She'll shave a massless princess.";

removeS(msg);

cout << msg; // prints he'll have a male prince.

}