Madeleine Murphy

004991129

November 23, 2018

HW/Project 6

* 1. 30 20 10 problem

int main(){

    int arr[3] = { 5, 10, 15 };

    int\* ptr;

    ptr = arr; //need to assign it this way

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

    ptr++;

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

    ptr++; //increment by 1 here again

    \*ptr = 10;          // set arr[2] to 10

    ptr = ptr - 2;      //go back to ptr[0] or arr[0]

    while (ptr <= &arr[2]) //print values arr[0] – arr[2]

    {

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

        ptr++;

    }

}

void findMax(int arr[], int n, int\*& pToMax) //add an &

{

    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;

}

When you do it the original way, it does not change ptr’s value outside of the function, thus you need to pass ptr/pToMax by reference to be able to change value and have new value appear.

The given function has a pointer, but the pointer can’t just exist it has to actually point to something 🡪 Essentially need to initialize the pointer to something

void computeCube(int n, int\* ncubed)

{

    \*ncubed = n \* n \* n;

}

int main()

{

    int arr[1];

    int\* ptr = arr;   //it's not pointing to anything originally

    computeCube(5, ptr);

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

}

* 1. The problem with this function is that they’re attempting to compare character by character but are implementing it the wrong way. They were trying to increment it by str1++ and str2++ which makes no sense because you’re not incrementing by the whole array. They also said “while (str1 != 0 and str2 != 0)”, which also makes no sense, that would be saying the address of the string does not equal 0, which it never will. Here’s one way you could fix this:

// return true if two C strings are equal

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

{

    int i = 0;

    while (str1[i] != '\0'  &&  str2[i] != '\0')

    {

        if (str1[i] != str2[i])  // compare corresponding characters

            return false;

        i++;                    // advance to the next character

    }

    return str1[i] == str2[i];   // both ended at same time?

}

int main()

{

    char a[15] = "Zhou";

    char b[15] = "Zou";

    if (strequal(a,b))

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

}

Just increment through character by character, until you reach the nullbyte. If you wanted to do this problem and instead increment by using pointers, you could do it like this:

// return true if two C strings are equal

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

{

    while (\*str1 != '\0' &&  \*str2 != '\0')

    {

        if (\*str1 != \*str2)  // compare corresponding characters

            return false;

        str1++;

        str2++;         // advance to the next characters by incrementing pointers by 1

    }

    return \*str1 == \*str2;   // both ended at same time?

}

int main()

{

    char a[15] = "Zhou";

    char b[15] = "Zhou";

    if (strequal(a,b))

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

}

This is essentially the same thing, but instead of incrementing by the value of the character array str1[0] ... str1[n], you are pointing to each place in both character arrays and comparing the values that the pointers point to.

* 1. The problem with this function is that anArray is only known to the function, not main, and so ptr is not actually initialized correctly. getPtrToArray returns the address for anArray, but this is not known outside of the function and thus ptr is not pointing to anArray. Additionally, f is doing something to the problem that causes it to print out strange values, for instance the first value returned is 123400000. It seems to be a pointless function but is still changing output values.

1. 1. double\* cat;
   2. double mouse[5];
   3. cat = &mouse[4];
   4. \*cat = 25;
   5. \*(mouse + 3) = 42;
   6. cat = cat – 3;
   7. cat[1] = 27;
   8. cat[0] = 54;
   9. bool b = (\*cat == \*(cat + 1));
   10. bool d = (cat == mouse);

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;

}



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;

}



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

{

    for (; \*str != '\0'; str++){

        if (\*(str) == chr)

            return str;

        }

    return nullptr;

}

1. The first thing that this code does is send 5 and 4 (array and &array[2]) to Maxwell. This returns 5 because 5 > 4, so \*ptr is set to 5. Then \*ptr (first position in array) is set to -1. We move to the 3rd position in the array, which holds 4. We move one over from there (ptr[1]) and set it to 9. So where it was 17, now it is 9. Then we set array+1, which is different from ptr+1, so we’re in the second position of the array (value 3) to 79. Ptr is currently at the third position in the array and &array[5] is the 6th position in the array. So when we take &array[5] – ptr, we get 3, since they are 3 positions apart.

The first swap swaps the positions of the arrays for the first two positions but does not actually change the numbers held in those positions outside of the function because it’s not pass by reference. The second swap, however, does successfully swap the numbers held in the first position[0] of the array (-1) and the third position[2] (4). So array[0] is now 4 and array[2] is now -1.

Thus, when we take this all into consideration, we get the output:

4 (from the swap)

79 (was never properly swapped, 79 from the line \*(array+1) = 79)

-1 (from the swap and the line that set \*ptr to -1)

9 (from when we set 17 to 9)

22 (we never changed this)

19 (we never changed this)



void removeS(char\* ptr){

    char\* newString= ptr;

    for (; \*ptr != '\0'; ptr++){

        if (\*(ptr) != 's' && \*(ptr) != 'S')

        {

            \*newString = \*ptr;  //add all the values that aren't S or s

            newString++;

        }

    }

    \*newString = '\0';

}