d jyDoes Strncopy add the null character automatically?

(^yes if the destination is smaller than the source)

It doesn’t always, and it doesn’t hurt to add it manually if you do it correctly.

Loops -> string\_fun? [https://drive.google.com/drive/folders/0ByiySreZqN6 HcFBIbjNOX3JHdWs](https://drive.google.com/drive/folders/0ByiySreZqN6HcFBIbjNOX3JHdWs)

# Chapter 5 Loops

1. Be able to convert any type of loop to any other
   1. For to while

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

printf(“I love panda\n”);

}

int i = 0;

while ( i < 3) {

printf(“I love panda\n”);

i++;

}

* 1. While to for

Same as a but reverse…

* 1. Do while to while

int i = 0;

do {

printf("The new alien movie is awesome..\n");

i++;

} while(i < 3);

int i = 0;

while (i < 3) {

printf("The new alien movie is awesome..\n");

i++;

}

// It still should be i < 3. The process is print check(i = 1), print check(i = 2), print check(i = 3), go out of the loop.

//What if int i started out as 4? Then the do while loop would run once but the while loop wouldn’t, right???

//Then the Do … While loop will do the loop body once, and the While loop will not do the loop body. (In case that the i started at 4).

// From 5.4 “A *do-while loop* is a loop construct that first executes the loop body's statements, then checks the loop condition.”

While to do while

Same as c but reverse;

* 1. For to do while

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

printf(“I love pie\n”);

}

int i = 0;

do {

printf("I love pie\n");

i++;

} while(i < 3);

* 1. Do while to for

Same as d but reverse;

1. What is the difference between a for, while, and do while loop?
   1. When should you use each

For-- Use when the number of iterations is computable before entering the loop, as when  
 counting down from X to 0, printing a character N times, etc.

While-- Use when the number of iterations is not computable before entering the loop, as when  
 Iterating until a user enters a particular character.

Checks test condition before iterating.

Do While-- use when you want the loop to iterate at least once

Iterates before evaluating a test condition

Checks test condition after iterating.

Interesting for input validation; usually do while loop is used for this purpose,

bc you are checking at least once

1. Know for each type of loop how the following affect their behavior
   1. Break: break out of the current loop.
   2. Continue: cause an immediate jump to the very top of the loop

//I think it should be ending the current loop and begin the next loop.

For instance for(i = 0; i<100; i++){

if (i == 4) {continue;}

If (i ==10) {break;}

printf (“c ”);

} // it should print ‘c’ for 9 times.

### **Example: continue statement inside for loop**

#include <stdio.h>  
int main()  
{  
 for (int j=0; j<=8; j++)  
 {  
 if (j==4)  
 {  
 /\* The continue statement is encountered when  
 \* the value of j is equal to 4.  
 \*/  
 continue;  
 }  
  
 /\* This print statement would not execute for the  
 \* loop iteration where j ==4 because in that case  
 \* this statement would be skipped.  
 \*/  
 printf("%d ", j);  
 }  
 return 0;  
}

Output:

0 1 2 3 5 6 7 8

Value 4 is missing in the output, why? When the value of variable j is 4, the program encountered a continue statement, which makes the control to jump at the beginning of the for loop for next iteration, skipping the statements for current iteration (that’s the reason printf didn’t execute when j is equal to 4).

* 1. Return: return a variable and exit the function. // Not necessary to return a value psa values and variables are different things

1. Be able to solve problems with loops including nested loops

Can someone give an example of continue in a loop? //

I think it would be something related to 2 or more loops nested together, possibly with an If statement within the loop5

I found this helpful:<https://stackoverflow.com/questions/6414/c-sharp-loop-break-vs-continue>

Question:

/\*

What if the break; is placed inside the second for loop, will it break out of the second loop ONLY, or the first for loop as well? ANSWER: second loop only

Eg.

int main() {

for (i = 0; i < 8; i++) {

for (j = 0; j < 8; j++) {

printf(“Trial. ”);

If (j == 5) {

Break;

}

}

}

}

jUST THE SECOND (f j==5), then it will run first for loop again then run the second for loop again

\*/

1. Be able to create a static array

What code would you write to declare

* + 1. An array of 10 integers

Int num[10];

* + - 1. An array of 4 doubles

double num[4];

* + 1. An array of integers initialized to 4, 5, 10   
       int num[] = {4,5,10};

//int num[3] = {4,5,10}; does this work too ? yes

* + 1. An array of 12 characters

char letter[12];

* + 1. A string that can hold words 5 letters long

char string[6];

When initializing, it’s 6. Make room for null character. Afterward, index starts at 0 and goes to 5 for a total of 6 elements.

* + 1. A string initialized to the word hello

char string[] = “hello”; Same case as above I believe. Initializing. //since you are initializing

to hello, you don’t have to put a number in the brackets: **char string[] = “hello”; //even better**

//do you not add a \0 at the end of “hello” ?

//nevermind the compiler automatically does that for you

1. What is the difference between an array of characters and a C string?

A C string has to end with the null character (? is that the only thing?) //yes

char string[6] = “Hello” char array[5] = {‘H’, ‘e’, ‘l’, ‘l’, was ‘o’};  
   
 A C string has the null character (‘\0’) at the end, while an array of characters might not

Note that printf and scanf use %s to specify a string item/array.

1. **Be able to use all of the string functions covered in class as well as write the C code to implement them**
   1. Strlen //take the string length
   2. Strcpy // copy the string from variable to another.
   3. Strncpy // copy the string for certain number of elements
   4. Strcat //copy (append) the string to the end of another
   5. Strncat // copy the string to the end of another for certain number elements
   6. Strcmp // return 0 if two string are equal & returns a negative number if str1 is less than str2, and a positive number if str1 is greater than str2. Evaluation first ==compares the character pair at element 0, then at element 1, etc., returning as soon as a pair differs.
2. strcpy(destStr, sourceStr) → Copies sourceStr **(up to and including null character**) to destStr.
3. strncpy(destStr, sourceStr, numChars) → Copies up to numChars characters. Add null character manually

Does strncpy add the null character automatically? No

/\*

Question:

strncpy(destStr, sourceStr, 5)

Does this copy the first 5 characters of sourceStr(+ the 6th ‘\0’) into desStr

OR does it copy the first 4 characters of sourceStr (+ the 5th ‘\0’) into desStr

\*/

//The string ends in a null character so it will copy 4 characters plus the null char (a string of length 5 will only have 4 characters).

// no, supposing Sourcestr is 7 characters long… and the 8th char is = ‘\0’

// Then it would copy 5 characters.

// plus the \0 (6th position) right?

// ...answer here...

1. strncpy ( str3, str2, 5 );  
    str3[5] = '\0'; /\* null character manually added \*/
2. strcat(destStr, sourceStr) → Copies sourceStr **(up to and including null character**) to *end* of destStr (**starting at destStr's null character**)(at “null” means after “null” , right? **No the string replaces the first strings null, so there’s only one null**)
3. Be able to create and work with multidimensional arrays
   1. Declare a 5 X 4 array of ints

Int arra[5][4];

* 1. Declare an array of 10 strings whose maximum length is 20 characters

Char arr[10][20];

* 1. Declare an 3 X 6 X 9 X 12 array of doubles

Double arr[3][6][9][12];

1. Know where space for different parts of your programs is stored and how long it lasts
   1. Code : Text, life of program
   2. Global and static variables: Data; life of programa
   3. Local variables: stack, before the function ends
   4. Arguments: stack, before the function ends
   5. Dynamically allocated space (space created by malloc): Heap, until you free space

This is from Matthew Butner’s ppt:

|  |  |  |
| --- | --- | --- |
| Section | Purpose | Duration |
| Text | This is where your code is stored | Life of Program |
| Data | This is where globals and static variables are stored | Life of Program |
| Stack | Stores local variables and arguments | Until function completes |
| Heap | Dynamically allocated space (malloc,realloc) | Until you free the space |

1. Be able to work with pointers in C to solve problems
2. If you want to be able to modify a variable of type X your function must accept at least a \_pointer to that type X\_(a.k.a an X\*)\_
3. Be able to use call by pointer to be able to “return” multiple values from a function

void foo(int\* num, double\* num2) {

\*num = 2;

\*num2 = 2.0;

return;

}

int main() {

int num = 0;

double num2 = 0;

foo(&num, &num2);

printf(“output = %d, %.1lf”, num, num2); // output = 2, 2.0

return 0;

}

* 1. A function that “returns” an int, int\*, and int\*\*

Can someone explain this one? I’m trying to simulate this in clion, but the program just doesn’t run anytime I try to do this, granted I actually filled the function out to change the values of int\* num2, int\*\* num3, etc.

//Maybe it’s because num2 and num3 don’t point to anything?

void main (){

int num1;

int\* num2;

int\*\* num3;

function(&num1, &num2, &num3);

}  
  
void function(int\* num1, int\*\* num2, int\*\*\* num3){

//where are the loops?

return;

}

//is there supposed to be a for loop with a j variable in between the i and k loops?

Nope. The third pointer gets taken care of outside of both loops; the second pointer gets taken care of in the i for loop; the first pointer gets taken care of in the k loop; the variable name doesn’t matter, k can be j. The number of for loops is the number of pointers - 1. Since the number of pointers is 3, the number of for loops is 3-1 or 2. Hope that helps!

Ok cool, thanks

1. Be able to allocate space with malloc
   1. Dynamically create an array of 7 integers and assign it to an integer pointer named ar

int\* ar = (int\*)malloc(7 \* sizeof(int));

* 1. //Dynamically create space for a string that will contain 9 characters and assign it to a char\* named str

char\* str = (char\*)malloc(10 \* sizeof(char));

//don’t forget, after you fill str, to add the null character at the end

str[9] = ‘\0’ ? is this correct?

Yes

1. Be able to dynamically create multidimensional arrays
   1. Dynamically create an array of doubles that is 3 X 4

double\*\* num = (double\*\*)malloc(3 \* sizeof(double\*));

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

num[i] = (double\*)malloc(4 \* sizeof(double));

}

* 1. Dynamically create an array of doubles that is 5 X 10 X 12

double \*\*\*num = (double\*\*\*)malloc(5 \* sizeof(double\*\*));

for (int i = 0; i < 5; i++) {

num[i] = (double\*\*)malloc(10 \* sizeof(double\*));

for (int k = 0; k < 10; k++) {

num[i][k] = (double\*)malloc(12 \* sizeof(double));

}

}

1. Be able to reallocate space in an array.
   1. Resize an array of ints called ar to contain 10 elements

ar = (int\*)realloc(ar, 10 \* sizeof(int));   
  
 //If this was inside a function, the argument would be int\*\* ar, right? Same thing for the other questions, would they be char\*\* ar and double\*\*\* num?

//It would be int\* ar, char\* ar and double\*\* num the arrays only contain 10 elements as in ar[10]. num is a 2D array. //Look at the very last problem though. In the function the 2D array is ar\*\*\* as a parameter. //based off the last program you would be right, you would have to pass double and triple pointers through another function. If perhaps we only used a main function, these would be the correct answers. No i'm not sorry. I mean the very last problem. The last one in this google doc.

* 1. Resize an array of chars called ar to contain 20 elements

ar = (char\*)realloc(ar, 20 \* sizeof(char));

* 1. Resize a 2 X 5 array of doubles into a 2 X 8 array

for (int i = 0; i < 2; i++) {

num[i] = (double\*)realloc(num**[i]**, 8 \* sizeof(double));

}

1. Be able to delete an array that has been dynamically allocated.
   1. Delete a single dimensional array of ints that contains 10 elements

free(array);

array= NULL;

* 1. Delete a 2D array of doubles that is 3 X 5

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

free(array[i]);

}

free(array);

array = NULL;

I believe we need to add a \* before the array name.. SO:

free(\*array) and also.. \*array= NULL; Any thoughts on this??

Not if this is in the main function - Then you do not need a pointer to array

**void** freeFunc6A(**int** \*\*\*A6){

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

free((\*A6)[i]);

}

free(\*A6);

\*A6 = **NULL**;

}

// this is what it looks like inside of a function

// When someone says “it looks like this inside a function” and adds an extra dereference operator (a \*) to the beginning of everything, is that just because the variable is passed-by-pointer?

//do we really need num = NULL; ?

//Setting unused pointers to NULL is a defensive style, protecting against dangling pointer bugs. If a dangling pointer is most systems, telling you right away what the error is. accessed after it is freed, you may read or overwrite random memory. If a null pointer is accessed, you get an immediate crash on

* 1. Delete an array of 17 strings.

for(int i=0;i<17;++i){

free(ar[i]);

}

free(ar);

ar = NULL;

//why using 2d free up?

//because there are 17 strings, each string is a char\*

**//do we need ar = NULL; ?**

//No, but pointers and strings are often set to NULL after they are freed. This is because if someone mistakenly tries to access the string or pointer after it is freed, they would access a NULL pointer, which is usually easy to debug.

//i think we need add a \* infront of ar because we are setting ar to NULL, refer to piazza @654, correct if im wrong if we are doing this inside the main function where ar is defined, there is no need to put a \* in front of ar. If we were doing this inside of a function, and were pointing to ar, all of our ar’s would need to have a \* in front.

* 1. Delete a 3D array of ints that is 3 X 2 X 2

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

for (int k = 0; k < 2; k++) {

free(theTaTas[i][k]);

}

free(theTaTas[i]);

}

free(theTaTas);

theTaTas = NULL;

//Do we have to set theTaTas to NULL ???

//It’s not *technically* required, but it’s good practice.

# Command Line Arguments

1. Be able to write programs that accept command line arguments

int main(int argc, char\* argv[]) { … return 0;}

1. What is the first command line argument passed to your program? The name of it.

the first command line *parameter* is argc, which is an integer representing the number of elements in the second parameter, argv holds the arguments starting with argv[0] as a string which is the name of your program. Your manually entered argument is argv[1]

However, this argument is not explicitly entered when you run the program. The first argument a user would *type* would technically be the name of the executable -- argc is calculated by the computer and passed into the program automatically

1. If I called a program as follows: ./a.out hello goodbye why sigh what would the value of argc be? 5

Argv[0] == “./a.out”

Argv[1] == “hello”

Argv[2] == “goodbye”

Argv[3] == “why”

Argv[4] == “sigh”

⇒ 5 arguments

//argv[0] is always the file that is called.

//Why is “./” at the beginning of every unix command to run a program? Doesn’t that just return you to the directory you’re already in?

But isnt ./a.out not a value? Isnt it the thing we use to run a program? It is, but it is always counted as the first argument, as argc and argv read everything that is placed on the line. They do not ignore the file call. An example would be our homework assignment is “./connectn.out rows columns pieces” has four arguments, and the first one is the file name.

You should probably also know how to use sscanf to parse strings into other data types because command line arguments are always passed as strings:

Argv[0] == “./a.out”

Argv[1] == “c”

Argv[2] == “16”

Argv[3] == “1.45”

sscanf(Argv[1], “%c”, &userChar);

sscanf(Argv[2], “%d”, &userNum);

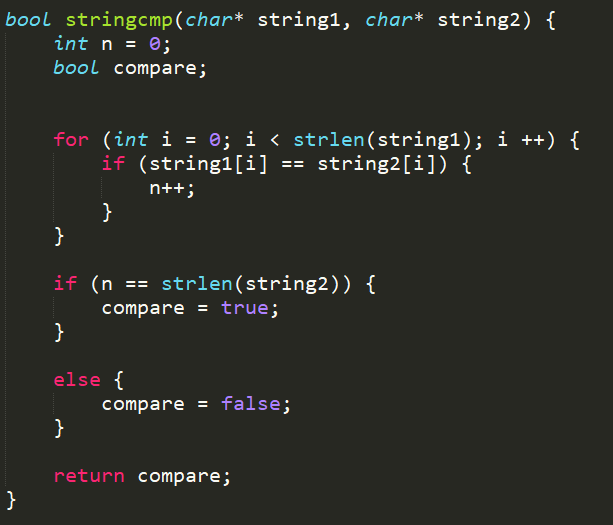
sscanf(Argv[3], “%lf”, &userDouble);

# Debugging // [NOT on midterm 2](https://piazza.com/class/j7jydgyoyta28r?cid=710)

# Practice Programs

1. Write a function that returns true if a string 1 begins with string 2.

WHICH ONE IS CORRECT?



//Is it necessary to pass pointers? Cuz we are not modifying the string.

//string is an array of characters.

//Got it, thanks.

// if you pass in (char string1[], char string2[]) ⇒ no need for pointer, but this will result in an C-string, not array string → the code may need to be changed as well.

//no difference actually, <https://www.cs.bu.edu/teaching/cpp/string/array-vs-ptr/>

// I think the for loop should be (i=0; i < strlen(string2)). 1

//Yes, because if String 1 is “hellohello” and if String 2 is “hello”,actually your n here will be 10 but not 5, which is actually a wrong output.

//the problem is that if string1 is longer than string2, the index will begin to search outside of sting2, this is a safer version:

It will not check indexes that are outside of the bounds of either char array.

//we can also just use function strcmp() which returns 0 if two strings are the same or 1 if they arent.

Shorter version, using string.h:

bool startsWith(char \*a, char \*b) {

return strstr(a, b) == a;

}

Note strstr returns a pointer to the first instance of b found in a - so if it’s found right at the beginning, it will point to the same memory location as a (hence the == a check).

Compiles & runs successfully: <https://onlinegdb.com/rJ-OErJgez>

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*alternate\*\*\*\*\*\*\*

bool strComp(){

char string1[] = "magnetic"; //modify to get user input

char string2[] = "magnet";

char stringtemp[strlen(string2)];

strncpy(stringtemp, string1, strlen(string2));

if (strcmp(stringtemp,string2)==0){

return true;

}else{

return false;

}6

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

//also, you could just do:

return strcmp(stringtemp,string2)==0;

There’s no need for the if/else statement

And what if string2 is longer than string1?

If (strlen(string1) < strlen(string2)) {return false;} //at beginning

K//

**//Is there any reason not to do the below?**

**bool** cmp(**char** \*str1, **char** \*str2){

**for**(**int** i =0; i < strlen(str2);i++){

**if**(str1[i] != str2[i]){

**return false**;

}

}

**return true**;

}

1. Write a void function that will append an element to the end of a list.

//Executed and works. Make sure to include parentheses around pointer in function (when reallocating and assigning to array) and don’t forget ‘&’ when passing list.

Why do we need parentheses around the pointer in the function? It works without it

//Can someone explain what this function is doing?

void Append(int\*\* list, int list\_size){ → can someone explain why int\*\* list ? //we are modifying the an array, so we need to pass a pointer to it

//You could also just not pass the pointer of the array and just pass the array and then you’d only need int \*list, and then you don’t need \*list = etc you can just pust list = etc. I tested and ran it and it works perfectly ←- correct. Idk why tho

// i don’t really understand why we need to pass a pointer if we’re modifying it, i’ve tried multiple examples and it works fine without

You do not need to pass a pointer to the array to modify it. Think of the array as a pointer in and of itself (it points to a list of integers on the heap). If you modify it and add an integer to the end, and then access the array later, the integer you added will still be in that location on the heap.

//I think you **do** need a double pointer, because you’re reallocating. So you might actually have to change where the memory is pointing if there’s not enough space. I think the code is wrong, but you’ll need a double pointer in case realloc has to move the data around to find a big enough space

void fool(int\*\* arr);

int main(){

int\*\* arr = (int\*\*)malloc(10\*sizeof(int\*));

for(int i = 0; i < 10; i++){

arr[i] = (int\*)malloc(10\*sizeof(int));

}

fool(arr);

}

void fool(int\*\* arr){

arr = (int\*\*)realloc(arr, 20\*sizeof(int\*));

for (int i = 0; i < 20; i++){

arr[i] = (int\*)realloc(arr[i], 20\*sizeof(int));

}

free(arr);

arr = NULL;

}

Yes we don’t need a triple pointer here, because we are not modifying the previous double pointer variable.

void Append(int\*\* list, int list\_size){ → can someone explain why int\*\* list ? //we are modifying the an array, so we need to pass a pointer to it //you do not need to pass a pointer in order to modify the array contents, the array itself is a pointer to the memory location of the elements

//but we are modifying the array - we’re making it one element longer

//why just use int \*list?<- we’re passing \*list rather than list[] to the function, and need a pointer to that array, thus int\*\* list. I do not know if int\*list[] is a form you can use in your programming, but I guess in theory it’s the same thing?

int element = 9;

(\*list) = (int\*)realloc((\*list), (list\_size + 1) \* sizeof(int));

(\*list)[list\_size] = element;

//Right? If the list size = 8, then 7 is the last index. Resize for an added element and now 8 is the final index. //yes //but how can you be sure that the elements in the list will stay the same? What if realloc makes the list point to a completely different section of memory, and you end up losing the contents of the list?

}

int main(){

int\* list = NULL;

int list\_size = 8;

int i = 0;

list = (int\*)malloc(list\_size \* sizeof(int));

for(i = 0; i < list\_size; ++i){

list[i] = i;

}

Append(&list, list\_size);

for(i = 0; i < list\_size; ++i){ //wouldn’t you get a seg fault error here when i = list\_size?

printf("%d\n", list[i]);

}

return 0;

}

void append(int \*\*list, int \*listLength, int valToAppend){

\*list = (int\*)realloc(list, (\*listLength + 1) \* sizeof(int));

\*list[\*listLength] = valToAppend;

\*listLength++;

return;

}

// Note that we need an int \*\* because realloc might return a new memory

// address to use for the list if it runs out of space in the current place.

// So, we want to modify an int \*, so we need at least an int \*\*

void append(int \*\*list, int \*numItems, int item) {

(\*numItems)++; // We’re adding another item

(\*list) = (int \*)realloc((\*list), (\*numItems) \* sizeof(int)); // We need to make space for it

(\*list)[(\*numItems) - 1] = item; // Actually set the item in the new, larger memory block

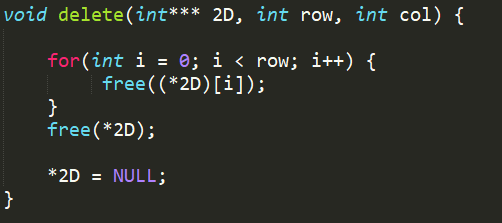
}

See here, this append function compiles & seems to work: <https://onlinegdb.com/SyVgeCJxf>

1. **Write a function that will delete a 2D array of ints and set the pointer to NULL.**

**Do we need “col” here?**

**Why int\*\*\*?**

 **No I think.**

**Col is not needed; depends on row for the array.**

**// here is how it works, you go through each row and free ALL COLUMNS in the row. Then, you free all the row //should this be like this? No because it’s only a two dimensional array.**

**When you are deleting a 2D array, you should have a triple pointer since you are passing the address of the 2D array. Also, you need to do “ \*2D = NULL” since you are referencing an outside array. If you were doing it in main, you could just do it without the pointer. Why we need a triple pointer? We can just use a double pointer parameter to store the 2D array’s first address. And it seems we don’t need to alter the original variable’s value which has stored the first address of the 2D array. Yeah I don’t see any reason to use a triple pointer Remember that when you pass a variable as a function argument, the function makes a copy of that variable instead of modifying the original. That holds true for pointers too. Usually this doesn’t matter because you’re not modifying the pointer itself, but wherever that pointer points to, which (unless something has gone very wrong) should be the same as what you passed in. For this function however, you’re specifically modifying the pointer itself by freeing it, so you have to use a triple pointer in order to modify the array. If you don’t, then you’re not freeing the array itself, but its copy, and then you have a memory leak.**

**This looks like it’s not in main function, should use triple pointer if setting the said pointer to NULL imo. But array itself is pointer, why do we always pointer pointer in functions? Because you want to change where the array pointer *is pointing to* (now it should be NULL)**

4. Write a function that will sort an array of strings by their length. J

Below is a **FUNCTIONAL** sort algorithm .

**#include <stdlib.h>**

**#include<memory.h>**

**void sort(char\*\*,int);**

**int main(){**

**int row=?;**

**char \*\* array=malloc(row\*sizeof (char \*));**

**for (int i = 0;i < row; i++){**

**char k=(char)(rand()%26+97)**

**char afa[]={k};**

**array[i]=malloc(k\*sizeof(char));**

**for (;k>0;k--){**

**strcat(array[i],afa);**

**}**

**}**

**sort (array,row);**

**}**

**void sort (char\*\* array,int row){**

**int minlength, index = 0, count = 0;**

**char\* temp;**

**while(count<row-1){**

**minlength=(int)strlen(array[count]);**

**index=count;**

**for(int i=count;i<row;i++){**

**int length=(int)strlen(array[i]);**

**if (length<minlength){**

**minlength=length;**

**index=i;**

**}**

**}**

**temp=array[index];**

**array[index]=array[count];**

**array[count]=temp;**

**count++;**

**}**

**}**

**This sort function really works!!!!**

***This one works as well but it is a bit sloppy because it leaves chars after the NULL character alive***

int main() {

chad

strcpy(names[2], "a");

strcpy(names[3], "bp");

strcpy(names[4], "bobby");

int numberOfStrings = 5;

sortNames(names, numberOfStrings);

for (int i = 0; i < 5; i++) {

printf("%s", names[i]);

printf("\n");

}

}

void sortNames(char names[5][6], int numberOfStrings) {

char temp[256];

for (int i = 0; i <= (numberOfStrings - 2); i++) {

if (strlen(names[i]) < strlen(names[i + 1])) {

strcpy(temp, names[i + 1]);

strcpy(names[i + 1], names[i]);

strcpy(names[i], temp);

strcpy(temp, " ");

i = -1;

}

}

}

// i is set to -1 so that the loop is not exited until the entire list is sorted

//If it iterates through the entire list without switching 2 strings then it is considered sorted and the for loop is exited

void sort (char name[4][21], int first) {

char temp[21];

for(int b = first - 1; b > 0; b--){

for (int a = 0; a < b; a++) {

if (strlen(name[a]) > strlen(name[a+1])) {

strcpy(temp,name[a]);

strcpy(name[a],name[a+1]);

strcpy(name[a+1], temp);

}

}

}

}

int main(){

char name[4][21]={"is where","China","Wuxi","I come from."};

sort(name, 4);

for(int i=0;i<4;i++) {

printf("%s\n",name[i]);

}

}

// This one works too

Another solution, which compiles online: <https://onlinegdb.com/HJM3wkxxz>