This lecture explains the sixth homework assignment.

This assignment needs a program that takes an article stored in a file and a word. The program counts how many times this word appears in the file.

What is a “word”? A word is a string. However, C does not have the data type for string.

In C, a string is an array of characters. One of the array elements must be a special character expressed as back slash zero. If no element is this special character, this is an array but it is not a string.

Since each element is a character, each element can store a value between 0 and 255. The conversion between the value and a character follows the ass kee table.

If the special back slash zero is not the last element in the array, the elements after this special character is not used for the string.

Here are some examples of strings. The first string is E C E followed space, followed by P U R D U E. . Please notice that the last element is the special character back slash zero.

In the second example, the array has more elements than the string’s need. The string has 2

6

4

Space

C.

The elements after back slash zero are not part of the string.

These unused elements are marked D C to indicate that we do not care about them.

In the third example, one of the characters is back slash n. This is the new line character.

You may want to ask why we have an array that is longer than necessary. Why do we have these unused elements in an array? There are many reasons. One of the most common reasons is that we create an array that is long enough to accommodate the longest possible string.

The C language has many functions for strings. S T R L E N. is one example. Let’s read the document of this function. The document says, “S T R L E N. calculates the length of the string pointed to s, excluding the terminating null byte”. This special null character is the back slash zero, mentioned earlier.

Please be careful that you have to add back slash for the null character. Without back slash, it is zero and it has a different meaning.

The null character’s ass kee value is zero.

The character zero, without back slash, has 48 as the ass kee value.

Please notice that the input must have this special null character. If the input has no null character, the behavior of S T R L E N. is undefined.

S T R L E N. takes only one argument and it is a pointer of character. Also, C O N S T is added in front of C H A R . .

It is very important to remember that S T R L E N. does not include the null character, even though this character is needed to terminate a string.

I will explain why this is pointer and why C O N S T is added in a moment.

Let’s consider another string function: S T R C P Y. .

This function copies one string to another string.

The S T R C P Y function takes two arguments, both of them are pointers. The first argument is the destination. The second is the source. The function copies the characters from the source to the destination, including the terminating null character. The document says that these memory space for the source and the destination must not overlap. If the memory space overlap, the behavior of the program is undefined. Also, the destination space must be large enough to accommodate the entire string from the source.

C does not check either conditions. It is the programmers’ responsibility ensuring that these two conditions are met.

Let me explain why a string is always a pointer. Here are two ways to create arrays. The first one uses heap memory. First, we create a pointer called A R R one. The space for 20 characters are allocated in the heap memory.

The pointer A R R 1 stores the address of the first element.

Next, we can use the S T R C P Y. function to copy the string Purdue E C E. to the array. Please notice that using double quotations automatically adds the terminating null character.

What is the difference between double and single quotations? In a C program, double quotation means a string and thus the null character is added. Single quotation means a single character.

Thus, when we express the terminating null character, we use single quotation because this is one character.

The length of Purdue E C E is 6 for Purdue, 1 for space, 3 for E C E . Thus, the length is 10. We have to add one more space for the null character. Thus, the destination must have at least 11 elements. We use malloc to allocate the space for 20 characters. This is large enough.

Another way of creating an array is to put the space on the stack memory. A R R 2 is an array of 20 characters. The symbol A R R 2 means the address of the first element of the array. This is the same as the meaning of A R R 1.

Please notice that we must not free A R R 2 because the space of A R R 2 is on the stack, not heap.

We must free A R R 1 because the space is allocated using malloc.

Malloc and Free always go together. One malloc must be accompanied by one free.

In C programs, strings are arrays of characters. Thus, a string is a pointer.

A pointer may not be an array. In this example, there is a character called C H and its value is the letter ei. Pee is a pointer to a character. The third line assigns the address of C H to pee. There is no array involved, even though pee is a pointer.

Thus, a pointer may not be an array.

Let me explain why the source and the destination must not overlap.

Consider this example. We create an array of 20 characters. A string of E C E space Purdue is stored in this array. This array is large enough to store this string. There is no problem.

Next, two pointers are created, pointing to the addresses of two elements of this array.

Everything is all right so far.

This slide shows the memory of the array and the pointers.

Please notice that S R C and D E S T point to two different elements of the same array.

What will happen if we use S T R C P Y now?

The characters will be copied from the source to the destination. The original string is changed.

In the S T R C P Y function, the word C O N S T is added in front of the source. What does this mean?

First, let’s consider this example. A function has two pointers, ei and bee. You probably can recognize this is the swap function.

What will happen if we put C O N S T in front of bee? G C C. will issue an error message. It says that we wanted to use bee to change the value of the address pointed by bee.

What does this mean? It means that we must not use the left hand side rule of the pointer.

Please notice that the right hand side rule is ok.

Please notice the bottom two lines in the function. The bottom line is not allowed. However, the second last line is allowed. Why? What is the difference?

This slide shows the stack memory after tee’s value has been assigned. This line uses the right hand side rule of \* ei.

The next line uses the right hand side rule of bee and the left hand side rule of ei. As a result, the value of x has been changed to -456.

The next line assigns ei’s value to bee’s value. Please notice that there is no \* involved. Thus, we are not using the left hand side rule nor the right hand side rule.

The next line is problematic because it needs the left hand side rule of bee. This is not allowed because of the word C O N S T in front of bee in the argument.

This slide shows that we are not allowed to circumvent the rule by assigning bee’s value to another pointer that does not have C O N S T in front.

It is also possible adding C O N S T in front of a local variable.

In this case, the left hand side rule is still not allowed.

What is the purpose of C O N S T? Why is it useful? C O N S T ensures that it is not possible to modify the value of a memory address through a pointer. Thus, we are comfortable that this function cannot modify the memory in another frame of the stack or the heap.

If a function can only read the memory but not modify, adding C O N S T allows G C C. to check and prevent accidentally modifying the memory.

Next, let’s read two other string functions.

The first function is S T R D U P.

.

This function duplicates a string. The input argument has C O N S T.

Thus, the input string will not be changed.

The second function is S T R S T R.

.

This function takes two arguments. Both of them have C O N S T, meaning that S T R S T R will not change either string.

The S T R S T R function finds the first occurrence of the second string within the first string.

Please notice that S T R D U P will allocate memory. Thus, the allocated memory should be freed when it is no longer needed.

S T R S T R does not compare the null character. Why? What does that mean?

S T R S T R wants to find whether the second argument is part of the first argument. If the null character were used in comparing the two strings, the second argument would have to appear only at the end of the first string because the null character can appear only at the end of the string.

Thus, the null character is not used for comparing the two strings.

This special null character is a frequent source of confusion and mistakes. Let’s consider this example about how to implement the S T R D U P function.

The function needs to allocate memory for the string. However, the S T R L E N function does not include the null character. Thus, we must add one for that null character. Without plus one, the allocated memory is not large enough for the string and the program’s behavior is undefined.

Let’s fully understand how S T R S T R works.

Consider this example. We create a string called tee. The string has P C E space E

C

E

C

E

C

E

C

E. . .

How many E C E does tee have?

One important question is whether E C E C E should count as one E C E or two E C E?

.

We will explain how to distinguish these two cases.

First, let’s see the stack memory. As explained earlier, tee is the address of the first element of the array. Suppose the array starts at address 200. Tee’s value is address 200.

Pee is a pointer and it has not been initialized. Thus, its value is u, meaning unknown.

What will happen to pee if it is the result of S T R S T R of tee and E C E?

S T R S T R will find the first occurrence of E C E in tee. This occurs at addresses 204 to 206.

Thus, pee will store the address 204.

If we want to find the next E C E, what should we do?

First, we know that we should not call S T R S T R tee and E C E again. If we do this, pee is still 204.

Also, we should not call S T R S T R pee and E C E. . If we do this, pee is also 204.

If we want to find the next occurrence of E C E, , we must move forward at least one character.

This slide shows how to do that. Pee plus plus increments pee’s value from 204 to 205.

We can assign this value to another pointer Q. .

Q’s value is address 205.

If we call S T R S T R Q E C E, S T R S T R will return address 206.

If we want the next occurrence, we can do pee plus plus again.

When we do pee plus plus, E C E C E will be treated as two E C E. .

What should we do if we want to count E C E C E as only one E C E? .

We need to move pee far enough so that it moves beyond the entire E C E in the first occurrence. To do so, we will use pee plus equal S T R L E N .

E C E. .

This will add three to pee and the next E C E will be searched from address 207, not 205.

If we search from address 207, then the E C E starting at address 206 will not be counted.

This slide summarizes the two different methods. If we want E C E C E to be counted as two E C E, then pee’s value increments by only one.

If we want E C E C E to be counted as only one E C E, then pee’s value increments by the length of E C E. .