Department of Computer Science and Engineering, PES University, Bangalore, India

Lecture Notes
Problem Solving With C
UE24CS151B

Lecture #1

String in C

By,

Prof. Sindhu R Pai,

Theory Anchor, Feb-May, 2025

Assistant Professor

Dept. of CSE, PESU

Many Thanks to

Dr. Shylaja S S (Director, CCBD and CDSAML Research Center, PES University)

Prof. Nitin V Poojari (Dean, Internal Quality Assurance Cell, PES University)

Unit #: 3

Unit Name: Text Processing and User-Defined Types

Topic: String in C

Course objectives: The objective(s) of this course is to make students

- Acquire knowledge on how to solve relevant and logical problems using computing Machine.
- Map algorithmic solutions to relevant features of C programming language constructs.
- Gain knowledge about C constructs and its associated ecosystem.
- Appreciate and gain knowledge about the issues with C Standards and it's respective behaviours.

Course outcomes: At the end of the course, the student will be able to:

- Understand and Apply algorithmic solutions to counting problems using appropriate C Constructs.
- Understand, Analyze and Apply sorting and Searching techniques.
- Understand, Analyze and Apply text processing and string manipulation methods using Arrays, Pointers and functions.
- Understand user defined type creation and implement the same using C structures, unions and other ways by reading and storing the data in secondary systems which are portable.

Sindhu R Pai

Theory Anchor, Feb - May, 2025

Dept. of CSE,

PES University

Let us answer a few questions before starting with Unit 3.

What is Text Processing?

The term text processing refers to the **Analysis, Manipulation, and Generation of text**. Text usually **refers to all the alphanumeric characters specified on the keyboard.** It plays a crucial role in systems where working with natural language or textual information is important — such as in **Search engines, Chat bots, Spell Checkers in Word processors, Text Analytics to analyze reviews/feed backs and AI models like ChatGPT!**

Key tasks of Text Processing Units

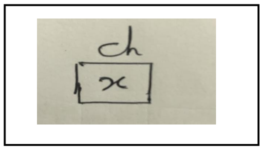
1. **Tokenization -** Breaking text into smaller parts like words, sentences, or characters.
    - Example: "I love pizza." → ["I", "love", "pizza", "."]
2. **Normalization -** Making text uniform by converting to lowercase, removing punctuation, etc.
    - Example: "Hello!" → "hello"
3. **Stop word Removal** - Removing common words like "is", "the", "and" that may not be useful in the application.
4. **Stemming/Lemmatization** - Reducing words to their root forms.
    - Example: "running", "ran" → "run"
5. **Part-of-Speech Tagging (POS) -** Identifying the grammar role of each word (noun, verb, adverb, adjective, etc.).
6. **Named Entity Recognition (NER)** - Detecting names of people, places, dates, etc.
    - Example: "Barack Obama was born in Hawaii." → Recognize "Barack Obama" as a person and "Hawaii" as a location.
7. **Parsing or Syntax Analysis -** Understanding the grammatical structure of sentences.

Text processing in C means working with and manipulating strings. C doesn't have a built-in "string" type like some other languages. So it uses **character arrays and functions** from the standard libraries to handle text.

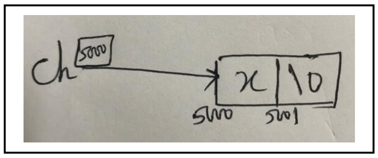
Introduction to Strings in C

String in C is an array of characters terminated with a special character '\0' or **NULL**. **ASCII value of NULL character is 0**. Each character occupies 1 byte of memory. There is **NO data type available to represent a string in C**.

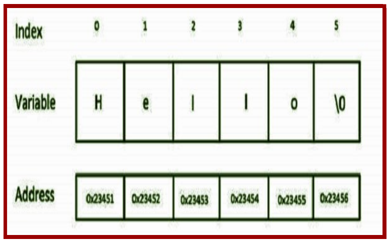
char ch = 'x' ; // character constant in single quote.



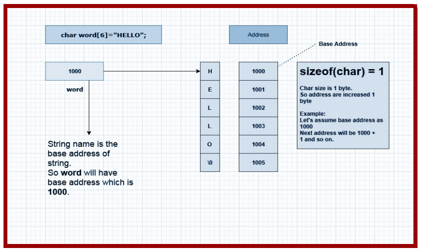
char ch[ ] = "x" ; // String constant. Always in double quotes terminated with '\0'. Always occupies 1 byte more when specified between " and " in initialization.



Note: When the compiler encounters a sequence of characters enclosed in the double quotes, it appends a null character '\0' at the end by default. The NULL character is crucial. Without it, C won't know where the string ends.



Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456



String name is the base address of string.
So **word** will have base address which is **1000**.

sizeof(char) = 1

Char size is 1 byte.
So address are increased 1 byte

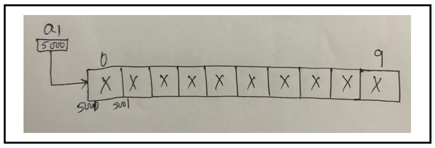
Example:
Let's assume base address as 1000
Next address will be 1000 + 1 and so on.

Definition of a String

char variable_name[size]; // Size of the array must be specified compulsorily.

char a1[10]; // Memory locations are filled with undefined values/garbage value

printf("sizeof a1 is %d",sizeof(a1)); // 10



Initialization of a string

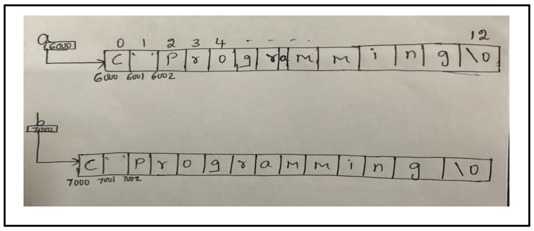
If the size is not specified, the compiler counts the number of elements in the array and allocates those many bytes to an array.

If the size is specified, it allocates those many bytes and unused memory locations are initialized with default value '\0'. This is **partial initialization.**

If the string is hard coded, it is **programmer's responsibility to end the string with '\0'** character.

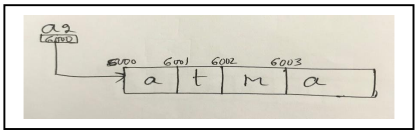
```
char a[ ] = { 'C', ' ', 'P', 'r', 'o', 'g', 'r', 'a', 'm', 'm, 'i', 'n',  'g',  '\0' };
char b[ ] = "C Programming";
```

C standard gives shorthand notation to write initializer's list. **b is a shorthand available only for storing strings in C.**

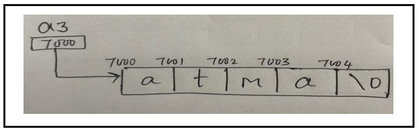

a
6000
0 1 2 3 4 - - - - 12
C ' ' P r o g r a m m i n g \0
6000 6001 6002
b
7000
C ' ' P r o g r a m m i n g \0
7000 7001 7002

Let us consider below code segments.

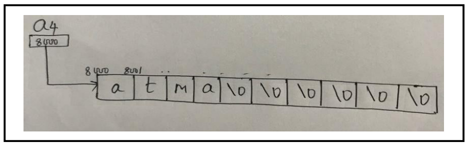
```
char a2[ ] = {'a','t','m','a'};//Initialization
printf("sizeof a2 is %d",sizeof(a2));  // 4  but cannot assure about a2 while printing
```



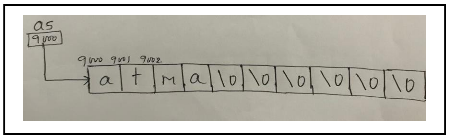
```
char a3[ ] = "atma" ; // Initialization
printf("sizeof a3 is %d",sizeof(a3));  // 5  sure about a3 while printing
```



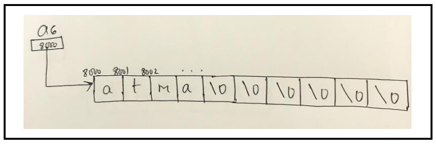
```
char a4[10 ] = {'a','t','m','a'}// Partial Initialization
printf("sizeof a4 is %d",sizeof(a4));// 10      sure about a4 while printing
```



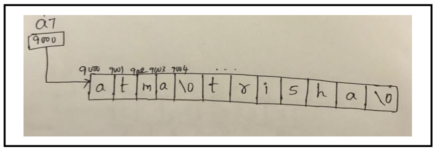
```
char a5[10 ] = "atma";
printf("sizeof a5 is %d",sizeof(a5));// 10 sure about a5 while printing
```



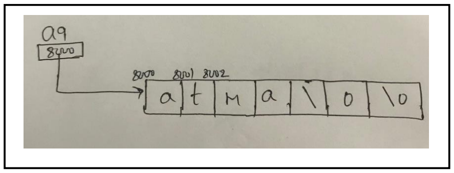
```
char a6[10 ] = {'a','t','m','a', '\0'};
printf("sizeof a6 is %d",sizeof(a6));// 10 sure about a6 while printing
```



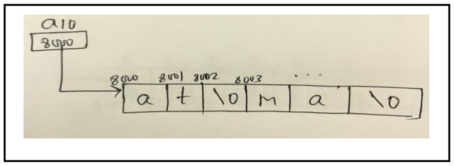
```
char a7[ ] = {'a', 't', 'm', 'a', '\0', 't', 'r', 'i', 's', 'h', 'a', '\0' };
printf("sizeof a7 is %d",sizeof(a7));  // 12 a7 will be printed only till first '\0'
```



```
char a9[ ] = "atma\\0" ;
printf("sizeof a9 is %d",sizeof(a9));// 7 sure about a9 while printing
```



```
char a10[ ] = "at\0ma" ;
printf("sizeof a10 is %d",sizeof(a10));//6 a10 will be printed only till first '\0'
```



Read and Display Strings in C

As usual, the formatted functions, **scanf and printf are used to read and display** a string. **%s is the format specifier for string.**

Consider

```
char str1[] = {'a', 't', 'm', 'a', 't', 'r', 'i', 's', 'h', 'a', '\0' };
```

One way of printing all the characters is using putchar/printf in a loop.

```
int i;
for (i = 0; i<sizeof(str1); i++)
        printf("%c",str1[i]);    //each element of string is a character. So %c.
```

Think about using while(str1[i] != '\0') rather than sizeof

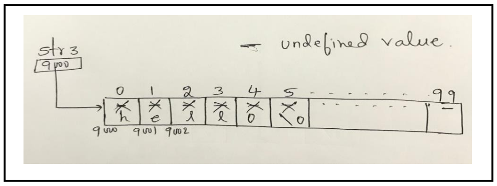
```
while(str1[i] != '\0')
{
        printf("%c",str1[i]);
        i++;
}
```

Also taking each character from the user using getchar() looks like a tedious task and also it doesn't serve the purpose if you want to store all characters in a string variable and play with it. You can only store one character at a time in a char variable.

So, use %s format specifier with scanf. **scanf with %s will introduce '\0' character at the end of the string. printf with %s requires the address and will display the characters until '\0' character is encountered.**

Coding Example_1: Input a string from the user and print that string.

```
char str3[100] ;
printf("Enter the string");
scanf(("%s", str3);          // User entered Hello Strings and pressed enter key
printf("%s\n",str3);         // Hello    . Reason: scanf terminates when white space is
encountered in the user input
```



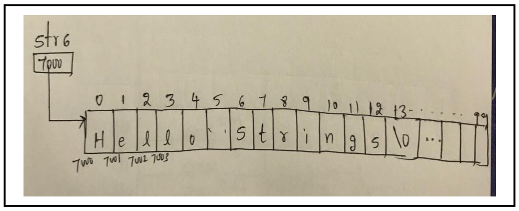
Coding Example_2: Read two strings from the user separated by a white space.

```
char str4[100] ;
char str5[100] ;
printf("Enter the string");
scanf(("%s %s", str4, str5);              // User entered Hello Strings and pressed enter key
printf("%s     %s",str4, str5);           // Hello     Strings
```

If you want to store the entire input from the user until user presses new line in one character array variable, use [^\n] with %s. Till \n encountered, everything has to be stored in one variable.

Coding Example_3: to store the entire input from the user until user presses new line in one string variable

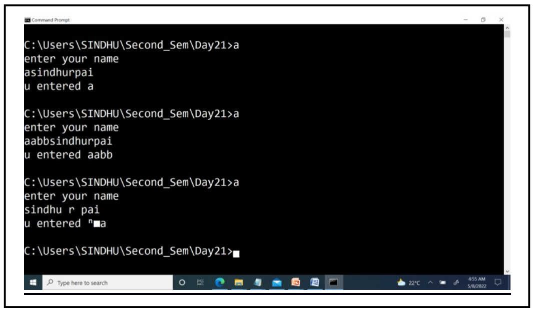
```
char str6[100] ;
printf("Enter the string");
scanf(("%[^\n]s", str6); // User entered Hello Strings and pressed enter key
printf("%s", str6);                    // Hello Strings
```



If you **want few restricted characters only to be allowed in user input**, how do you handle it?

Coding Example_4:

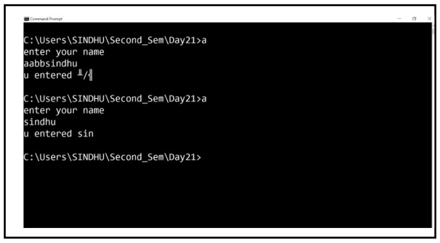
```
char str6[100] ;
printf("enter your name\n");
scanf("%[abcd]s",str6); // [ ] character class, starting with either a or b or c or d.
//When it encounters other characters, scanf terminates
printf("u entered %s\n",str6);
```



```
C:\Users\SINDHU\Second_Sem\Day21>a
enter your name
asindhurpai
u entered a

C:\Users\SINDHU\Second_Sem\Day21>a
enter your name
aabbsindhurpai
u entered aabb

C:\Users\SINDHU\Second_Sem\Day21>a
enter your name
sindhu r pai
u entered "▪a

C:\Users\SINDHU\Second_Sem\Day21>
```

Coding Example_5: How to negate the above character class

```
char str7[100] ;
printf("enter your name\n");
scanf("%[^abcd]s",str6); // neither a nor b nor c nor d
//When it encounters any of these characters, scanf terminates
printf("u entered %s\n",str6);
```



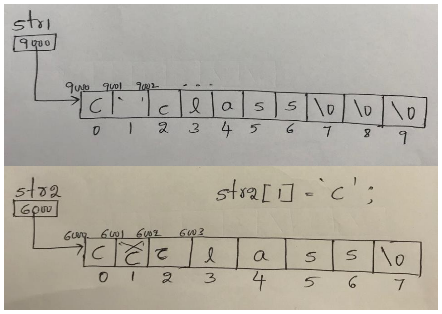
```
C:\Users\SINDHU\Second_Sem\Day21>a
enter your name
aabbsindhu
u entered ╨/╣

C:\Users\SINDHU\Second_Sem\Day21>a
enter your name
sindhu
u entered sin

C:\Users\SINDHU\Second_Sem\Day21>
```

Pointers v/s String

As array and pointers are related, strings and pointers go hand in hand. Let us consider a few examples.

Coding Example_6: Modifying the string pointed by character array.

```
char str1[10] = "C class" ;
printf("%s", str1);  // C class
char str2[ ] = "C class" ;
printf("%s\n", str2);// C class
str2[1] = 'C' ;
printf("%s\n", str2);  // CCclass
```



Coding Example_7: Modifying the string pointed by character pointer

```
char *p1 = "pesu"; //p1 is stored at stack.  "pesu" is stored at code segment of memory. It is read only. // This statement assigns to p1 variable a pointer to the character array
printf("size is %d\n", sizeof(p1));                    // size of pointer
printf("p1 is %s", p1) ;          //pesu          // p1 is an address. %s prints until \0 is encountered
p++;    // Pointer may be modified to point elsewhere.
printf("p1 is %s", p1) ;          //esu
p1[1] = 'S' ;      // No compile time Error
printf("p1 is %s", p1) ; //Behaviour is undefined if you try to modify the read only location
```

Coding Example_8: Array is a constant pointer.

```
char p2[] = "pesu";                           // Stored in the Stack segment of memory
printf("size is %d\n", sizeof(p2));      // 5 = number of elements in array+1 for '\0'
printf("p2 is %s", p2) ;          //pesu
p2[1] = 'E';     //Individual characters within the array may be modified.
printf("p2 is %s", p2) ;          //pEsu
p2++;  // Compile time Error // Array always refers to the same storage.
```

Happy Coding using Strings!!