SNS COLLEGE OF ENGINEERING

Kurumbapalayam (Po), Coimbatore – 641 107

An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A' Grade

Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

COURSE NAME : 23ITT101- PROBLEM SOLVING & C PROGRAMMING

I YEAR /I SEMESTER

Unit II – ARRAYS AND STRINGS

Topic : String

Fundamentals of Strings and Characters

- Characters
  - Building blocks of programs
    - Every program is a sequence of meaningfully grouped characters
  - Character constant
    - An **int** value represented as a character in single quotes
    - **'z'** represents the integer value of **z**

Fundamentals of Strings and Characters

- Strings
  - Series of characters treated as a single unit
    - Can include letters, digits and special characters (*****, ***I***, **$**)
  - String literal (string constant) - written in double quotes
    - **"Hello"**
  - Strings are arrays of characters
    - String a pointer to first character
    - Value of string is the address of first character

String and Character Array

String declarations

Declare as a character array or a variable of type **char ***

```
char color[] = "blue";
char *colorPtr = "blue";
```

Remember that strings represented as character arrays end with **'\0'**

color has **5** elements

String and Character Array

Inputting strings

Use **scanf**

scanf("%s", word);

//Copies input into **word[]**

Do not need **&** (because a string is a pointer)

Remember to leave room in the array for **'\0'**

Character Handling Library

Character handling library

- Includes functions to perform useful tests and manipulations of character data
- Each function receives a character (an **int**) or **EOF** as an argument

Character Handling Library <ctype.h>

Prototype	Description
`int isdigit( int c )`	Returns **`true`** if **`c`** is a digit and **`false`** otherwise.
`int isalpha( int c )`	Returns **`true`** if **`c`** is a letter and **`false`** otherwise.
`int isalnum( int c )`	Returns **`true`** if **`c`** is a digit or a letter and **`false`** otherwise.
`int isxdigit( int c )`	Returns **`true`** if **`c`** is a hexadecimal digit character and **`false`** otherwise.
`int islower( int c )`	Returns **`true`** if **`c`** is a lowercase letter and **`false`** otherwise.
`int isupper( int c )`	Returns **`true`** if **`c`** is an uppercase letter; **`false`** otherwise.
`int tolower( int c )`	If **`c`** is an uppercase letter, **`tolower`** returns **`c`** as a lowercase letter. Otherwise, **`tolower`** returns the argument unchanged.
`int toupper( int c )`	If **`c`** is a lowercase letter, **`toupper`** returns **`c`** as an uppercase letter. Otherwise, **`toupper`** returns the argument unchanged.
`int isspace( int c )`	Returns **`true`** if **`c`** is a white-space character—newline (**`'\n'`**), space (**`' '`**), form feed (**`'\f'`**), carriage return (**`'\r'`**), horizontal tab (**`'\t'`**), or vertical tab (**`'\v'`**)—and **`false`** otherwise
`int iscntrl( int c )`	Returns **`true`** if **`c`** is a control character and **`false`** otherwise.
`int ispunct( int c )`	Returns **`true`** if **`c`** is a printing character other than a space, a digit, or a letter and **`false`** otherwise.
`int isprint( int c )`	Returns **`true`** value if **`c`** is a printing character including space (**`' '`**) and **`false`** otherwise.
`int isgraph( int c )`	Returns **`true`** if **`c`** is a printing character other than space (**`' '`**) and **`false`** otherwise.

Example

```
#include<stdio.h>
#include<ctype.h>
void main()
{
 printf("%s", isalpha('A') ? "A is a ": "A is not a","letter");
 printf("%s", isdigit('A') ? "A is a ": "A is not a","digit");
 printf("%s", isalnum('A') ? "A is a ": "A is not a","digit or a letter");

  getch();
}
```

output:

A is a letter
A is not a digit
A is a digit or a letter

Examples

```
#include <stdio.h>
int main( ) {

   int c;

   printf( "Enter a value :");
   c = getchar( );

   printf( "\nYou entered: ");
   putchar( c );

   return 0;
}
```

Output:

```
Enter a value : this is test
You entered: t
```

```
#include <stdio.h>
int main( ) {

   char str[100];

   printf( "Enter a value :");
   gets( str );

   printf( "\nYou entered: ");
   puts( str );

   return 0;
}
```

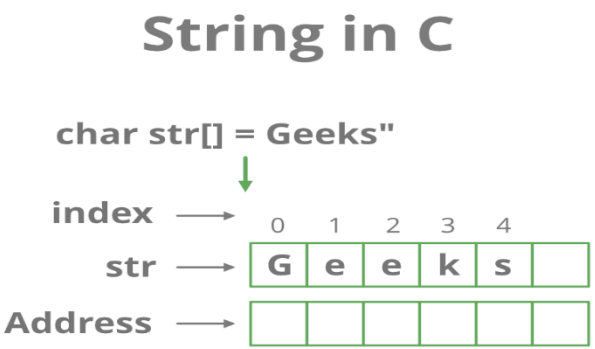
Output:

```
Enter a value : this is test
You entered: this is test
```

- Strings are defined as an array of characters.

For example

- The string "hello world" contains 12 characters including '\0' character.
- Which is automatically added by the compiler at the end of the string.



Declaring and Initializing a string variables

- There are different ways to initialize a character array variable.
- Declaring a string is as simple as declaring a one dimensional array.

```
char str_name[size];
```

```
char name[13] = "StudyTonight";      // valid character array

char name[10] = {'L','e','s','s','o','n','s','\0'};      // valid initialization
```

- Remember that when you initialize a character array by listing all of its characters separately then you must supply the '\0' character explicitly.

- Some examples of illegal initialization of character array are,

```
char ch[3] = "hell";    // Illegal
char str[4];
str = "hell";   // Illegal
```

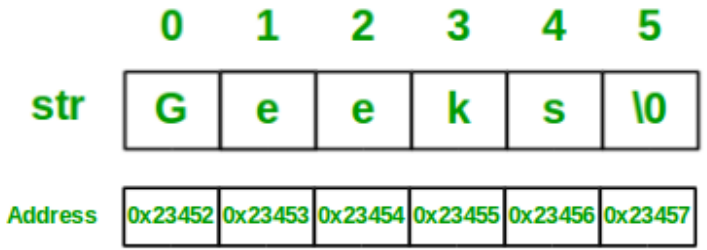
A string can be initialized in different ways.

1. char str[] = "GeeksforGeeks";

2. char str[50] = "GeeksforGeeks";

3. char str[] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};

4. char str[14] = {'G','e','e','k','s','f','o','r','G','e','e','k','s','\0'};



C program to read strings

```
#include<stdio.h>
  int main()
{
    // declare and initialize string
    char str[] = "Geeks";
          // print string
    printf("%s",str);
          return 0;
}
```

Output:

Geeks

```
#include<stdio.h>
int main()
{        // declaring string
    char str[50];
        // reading string
    scanf("%s",str);
        // print string
    printf("%s",str);
    return 0;
}
```

Output:

Geeks

String Input and Output

- Input function scanf() can be used with %s format specifier to read a string input from the terminal.
- But there is one problem with scanf() function, it terminates its input on the first white space it encounters.
- To read an input string "Hello World" using scanf() function, it will only read Hello and terminate after encountering white spaces.

Read string in C using gets()

- The gets() function is defined inside “stdio.h” library.
- The gets() function takes the start address of an area of memory suitable to hold the input as a single parameter.
- The gets() function read a line (terminated by a newline character \n) from the input stream and make a null-terminates string out of it.
- It reads data until it finds a newline or end-of-file. the gets() function declaration is,

```
char* gets(char* strptr);
```

To read character string with white spaces

```
//using scanf()

#include<stdio.h>
#include<string.h>
void main()
{
    char str[20];
    printf("Enter a string");
  //scanning the whole string, including the white spaces
scanf("%[^\n]", &str);
 printf(" The Entered string  : %s", str);
}
```

```
//using the gets() function.

#include<stdio.h>
#include<string.h>
void main()
{
 char str[20];
 printf("Enter a string");
 gets(str);
 printf("");
 printf(" The Entered string  : %s", str);
}
```

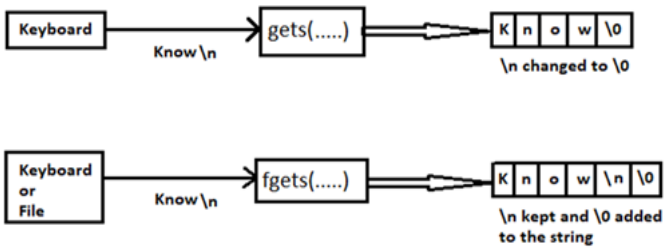
Output:

Enter a string: Know Program C language
The Entered string : Know Program C language

Read string in C using fgets()

- The fgets() function is also defined inside “stdio.h” library.
- It also takes a line (terminated by a newline) from the input stream and makes a null-terminates string out of it. The fgets() function can get input from a file or standard input.
- The fgets() function declaration is,

```
char* fgets(char* strptr, int length, FILE * stream);
```



Comparison between gets() and fgets()

Aspect	gets()	fgets()
Safety	Unsafe, no bounds checking, can lead to buffer overflow.	Safe, reads up to the specified limit, preventing overflow.
Newline Character	Does not store the newline character (\n).	Stores the newline character if there is space in the buffer.
Buffer Overflow	Can cause buffer overflow if input exceeds buffer size.	Does not cause buffer overflow as it limits the number of characters read.
Termination	Stops reading at the first newline (\n) or EOF.	Stops reading at the first newline (\n), EOF, or the specified limit.
End of Input	Can cause undefined behavior if the input exceeds buffer size.	Returns NULL on error or if EOF is encountered before reading any characters.
Return Value	Returns the string on success or NULL on failure.	Returns the string on success or NULL on failure.

Display string in C

puts() or fputs() or printf() with %s format specifier.

- puts() - It terminates the line with a new line, '\n'.
- The puts() function change '\0' to a new line but fputs() function doesn't change it.

```
int puts(const char* strptr);
int fputs(const char* strptr, FILE* stream);
```

/*Display string in C using printf() with %s format code*/

```
#include<stdio.h>
int main()
{
   char str[100];
   printf("Enter a string: ");
   fgets(str, sizeof(str), stdin);
   printf("%s",str);
   return 0;
}
```

Output:-

```
Enter a string: Know Program
Know Program
```

//Demonstrating fscanf() and fprintf() function

```
#include<stdio.h>
int main()
{
   char str[10];
   printf("Enter a string: ");
   fscanf(stdin, "%s", str);
   fprintf(stdout, "String = %s", str);
   return 0;
}
```

Output:-

```
Enter a string: Know Program C language
String = Know
```

- String Input-Output using fscanf() and fprintf() functions
- Each C program has three input-output streams:- stdin, stdout, and stderr.
- The input stream is called standard-input (stdin), the output stream is called standard-output (stdout), and the side stream of output characters from errors is called the standard error (stderr).
- Internally they occupy file descriptors 0, 1, and 2 respectively. The fprintf() sends formatted output to a stream and fscanf() scans and formats input from a stream.

Aspect	scanf()	gets()
Input Type	Used for reading formatted input (e.g., strings, integers, floats).	Used for reading a whole line of text, including spaces.
Whitespace Handling	Stops reading at the first whitespace (space, tab, newline) for strings.	Reads entire line, including spaces.
Safety	Can cause issues if used incorrectly, particularly with %s (it can lead to buffer overflow if the buffer is not large enough).	Unsafe because it does not check buffer size, leading to buffer overflows.
Newline Character	Does not store newline characters (\n) in the buffer.	Does not store the newline character (\n).
Common Use	Commonly used for formatted input, like reading specific data types (e.g., integers, strings).	Used for reading a complete line of text, but should be avoided due to safety issues.
Return Value	Returns the number of successfully scanned items. Returns EOF on error or end of input.	Returns the string on success, NULL on error.
Buffer Overflow	Can lead to buffer overflow if the input exceeds the size of the variable when using %s.	Can lead to buffer overflow because it doesn't check the buffer size.
Deprecation	Not deprecated, but must be used carefully, especially with %s.	Deprecated in modern C standards (C11) due to security risks.

Aspect	gets()	puts()
Purpose	Reads a line of input from the user.	Prints a string to the output.
Input/Output	Input: Reads a string from the user.	Output: Prints a string to the screen.
Buffer Overflow	Unsafe: Can cause buffer overflow if input exceeds buffer size.	No risk of buffer overflow (because it only prints, does not read).
Newline Handling	Does not store the newline character (\n) in the buffer.	Automatically adds a newline after printing the string.
Return Value	Returns the string (str) on success, or NULL on error.	Returns a non-negative integer on success, or EOF on error.
Deprecation	Deprecated in C11 because of security risks.	Not deprecated and is safe to use.
Typical Use	Used for reading input from the user, typically without any format restrictions.	Used for printing output, often to display strings with a newline.


Thank
you