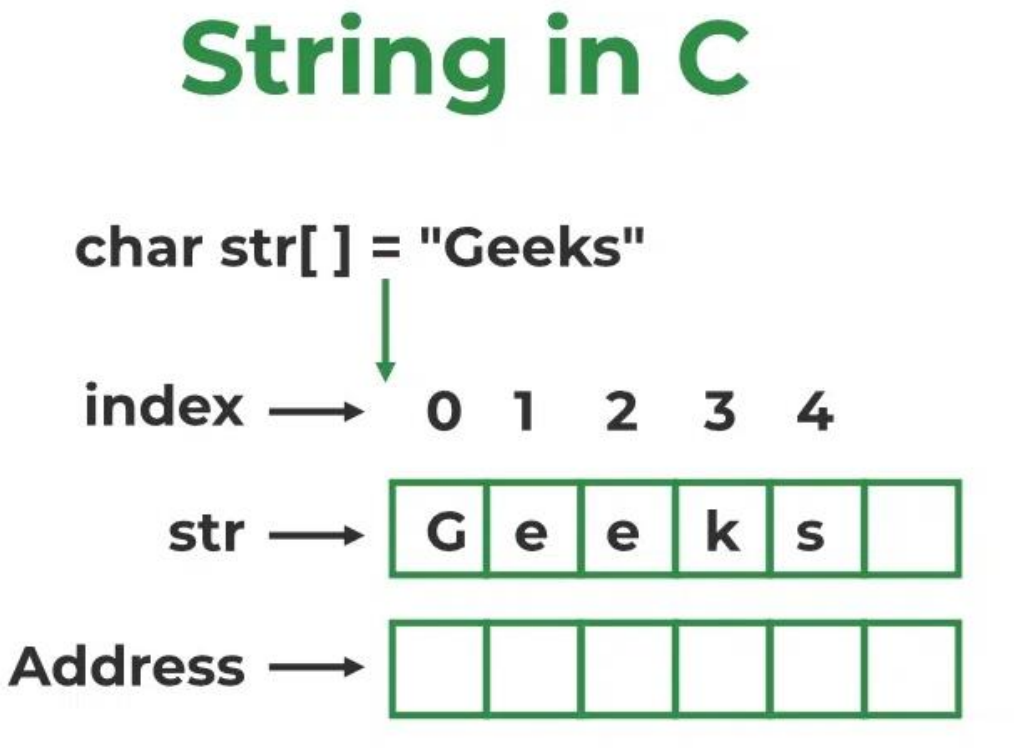
String

A String in C programming is a sequence of characters terminated with a null character ‘\0’. The C String is stored as an array of characters. The difference between a character array and a C string is that the string in C is terminated with a unique character ‘\0’.



C String Declaration Syntax

Declaring a string in C is as simple as declaring a one-dimensional array. Below is the basic syntax for declaring a string.

```
char string_name[size];
```

In the above syntax **string_name** is any name given to the string variable and size is used to define the length of the string, i.e the number of characters strings will store.

There is an extra terminating character which is the ***Null character (‘\0’) used to indicate the termination of a string that differs strings from normal character arrays***.

C String Initialization

A string in C can be initialized in different ways. We will explain this with the help of an example. Below are the examples to declare a string with the name str and initialize it with “GeeksforGeeks”.

We can initialize a C string in 4 different ways which are as follows:

1. Assigning a String Literal without Size

String literals can be assigned without size. Here, the name of the string str acts as a pointer because it is an array.

```
char str[] = "GeeksforGeeks";
```

2. Assigning a String Literal with a Predefined Size

String literals can be assigned with a predefined size. But we should always account for one extra space which will be assigned to the null character. If we want to store a string of size n then we should always declare a string with a size equal to or greater than n+1.

```
char str[50] = "GeeksforGeeks";
```

3. Assigning Character by Character with Size

We can also assign a string character by character. But we should remember to set the end character as ‘\0’ which is a null character.

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

4. Assigning Character by Character without Size

We can assign character by character without size with the NULL character at the end. The size of the string is determined by the compiler automatically.

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

Note: *When a Sequence of characters enclosed in the double quotation marks is encountered by the compiler, a null character ‘\0’ is appended at the end of the string by default.*

Let us now look at a sample program to get a clear understanding of declaring, and initializing a string in C, and also how to print a string with its size.

C String Example

- C

```
// C program to illustrate strings

#include <stdio.h>

#include <string.h>
```

```
int main()

{

    // declare and initialize string

    char str[] = "Geeks";

    // print string

    printf("%s\n", str);

    int length = 0;

    length = strlen(str);

    // displaying the length of string

    printf("Length of string str is %d", length);

    return 0;

}
```

Output

```
Geeks

Length of string str is 5
```

String Format Specifier – %s in C

The %s in C is used to print strings or take strings as input.

Syntax:

```
printf("%s...", ...);
scanf("%s...", ...);
```

Example:

- C

```
// C program to illustrate the use of %s in C

#include <stdio.h>
```

```
int main()
{
    char a[] = "Hi Geeks";
    printf("%s\n", a);
    return 0;
}
```

Output

```
Hi Geeks
```

C Variables, Constants and Literals

Variables

In programming, a variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

```
int playerScore = 95;
```

Here, `playerScore` is a variable of `int` type. Here, the variable is assigned an integer value `95`.

The value of a variable can be changed, hence the name variable.

```
char ch = 'a';
// some code
ch = 'l';
```

Rules for naming a variable

1. A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
2. The first letter of a variable should be either a letter or an underscore.
3. There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.

> **Note:** You should always try to give meaningful names to variables. For example: `firstName` is a better variable name than `fn`.

C is a strongly typed language. This means that the variable type cannot be changed once it is declared. For example:

```
int number = 5;      // integer variable
number = 5.5;        // error
double number;       // error
```

Here, the type of `number` variable is `int`. You cannot assign a floating-point (decimal) value `5.5` to this variable. Also, you cannot redefine the data type of the variable to `double`. By the way, to store the decimal values in C, you need to declare its type to either `double` or `float`.
Visit this page to learn more about different types of data a variable can store.

Literals

Literals are data used for representing fixed values. They can be used directly in the code. For example: `1`, `2.5`, `'c'` etc.
Here, `1`, `2.5` and `'c'` are literals. Why? You cannot assign different values to these terms.

---

1. Integers

An integer is a numeric literal(associated with numbers) without any fractional or exponential part. There are three types of integer literals in C programming:

- decimal (base 10)
- octal (base 8)
- hexadecimal (base 16)

For example:

```
Decimal: 0, -9, 22 etc

Octal: 021, 077, 033 etc

Hexadecimal: 0x7f, 0x2a, 0x521 etc
```

In C programming, octal starts with a `0`, and hexadecimal starts with a `0x`.

---

2. Floating-point Literals

A floating-point literal is a numeric literal that has either a fractional form or an exponent form. For example:

```
-2.0

0.0000234

-0.22E-5
```

> **Note:** `E-5 = 10^-5`

---

3. Characters

A character literal is created by enclosing a single character inside single quotation marks. For example: `'a'`, `'m'`, `'F'`, `'2'`, `'}'` etc.

---

4. Escape Sequences

Sometimes, it is necessary to use characters that cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc.

In order to use these characters, escape sequences are used.

Escape Sequences	
Escape Sequences	Character
`\b`	Backspace
`\f`	Form feed

Escape Sequences	
Escape Sequences	Character
`\n`	Newline
`\r`	Return
`\t`	Horizontal tab
`\v`	Vertical tab
`\\`	Backslash
`\'`	Single quotation mark
`\"`	Double quotation mark
`\?`	Question mark
`\0`	Null character

For example: `\n` is used for a newline. The backslash `\` causes escape from the normal way the characters are handled by the compiler.

---

5. String Literals

A string literal is a sequence of characters enclosed in double-quote marks. For example:

```
"good"                  //string constant

""                      //null string constant

"      "                //string constant of six white space

"x"                     //string constant having a single character.

"Earth is round\n"          //prints string with a newline
```

Constants

If you want to define a variable whose value cannot be changed, you can use the `const` keyword. This will create a constant. For example,

```
const double PI = 3.14;
```

Notice, we have added keyword `const`.

Here, `PI` is a symbolic constant; its value cannot be changed.

```
const double PI = 3.14;
PI = 2.9; //Error
```

Reading Strings and Lines of Text

The `scanf()` function is also capable of reading strings and lines of text. This is done using the **%s** format specifier. For instance, to read a string, you would use:

```
scanf("%s", str);
```

This tells the function to read a string and store it in the variable **str**. However, the `scanf()` function stops reading a string when it encounters a white space. To read a full line of text, including white spaces, you can use the `fgets()` function instead. This function reads a line of text until it encounters a newline character or reaches the end of the specified length.

getchar Function in C

- **C getchar** is a standard library function that takes a single input character from standard input. The major difference between getchar and getc is that getc can take input from any no of input streams but getchar can take input from a single standard input stream.
  - It is defined inside the **<stdio.h> header file.**
  - Just like getchar, there is also a function called putchar that prints only one character to the standard output stream.

Syntax of getchar() in C

```
int getchar(void);
```

getchar() function does not take any parameters.

Return Value

- The input from the standard input is read as an unsigned char and then it is typecast and returned as an integer value(int).
- EOF is returned in two cases:
  - When the file end is reached
  - When there is an error during the execution

Examples of C getchar Function

The following C programs demonstrate the use of getchar() function

Example 1: Read a single character using getchar() function.

Below is the C program to implement getchar() function to read a single character:

- C

```
// C program to implement getchar()
```

```
// function to read single character

#include <stdio.h>



// Driver code

int main()

{

    int character;

    character = getchar();



    printf("The entered character is : %c", character);

    return 0;

}
```

Input

```
f
```

Output

```
The entered character is : f
```

gets()

Reads characters from the standard input (stdin) and stores them as a C string into str until a newline character or the end-of-file is reached.

- It is not safe to use because it does not check the array bound.
- It is used to read strings from the user until a newline character is not encountered.

Syntax

```
char *gets( char *str );
```

Parameters

- **str:** Pointer to a block of memory (array of char) where the string read is copied as a C string.

Return Value

- The function returns a pointer to the string where input is stored.

Example of gets()

Suppose we have a character array of 15 characters and the input is greater than 15 characters, gets() will read all these characters and store them into a variable. Since, gets() does not check the maximum limit of input characters, at any time compiler may return buffer overflow error.

- C++

```
// C program to illustrate

// gets()

#include <stdio.h>

#define MAX 15



int main()

{

    // defining buffer

    char buf[MAX];



    printf("Enter a string: ");
```

```
    // using gets to take string from stdin

    gets(buf);

    printf("string is: %s\n", buf);



    return 0;

}
```

putchar() function in C

- The **putchar(int ch)** method in C is used to write a character, of unsigned char type, to stdout. This character is passed as the parameter to this method.
Syntax:

```
int putchar(int ch)
```

Parameters: This method accepts a mandatory parameter **ch** which is the character to be written to stdout.
Return Value: This function returns the character written on the stdout as an unsigned char. It also returns EOF when some error occurs.

The below examples illustrate the use of putchar() method:

Example 1:

- C

```
// C program to demonstrate putchar() method


#include <stdio.h>;


int main()

{
```

```
    // Get the character to be written
    char ch = 'G';


    // Write the Character to stdout
    putchar(ch);


    return (0);
}
```

Output

```
G
```

puts() in C

- In C programming language, **puts()** is a function defined in header **<stdio.h>** that prints strings character by character until the NULL character is encountered. The puts() function prints the newline character at the end of the output string.

Syntax

```
int puts(char* str);
```

Parameters

- **str:** string to be printed.

Return Value

The return value of the puts function depends on the success/failure of its execution.

- On success, the puts() function returns a non-negative value.
- Otherwise, an End-Of-File (EOF) error is returned.

Example

- C

```
// C program to illutrate the use of puts() function
```

```c
#include <stdio.h>

int main()

{

    // using puts to print hello world

    char* str1 = "Hello Geeks";

    puts(str1);



    puts("Welcome Geeks");



    return 0;

}
```

Output

```
Hello Geeks

Welcome Geeks
```

Null character in C

The Null character is used to end character strings in the C coding language. In other terms, in C, the Null character represents the conclusion of a string, the end of an array, or other concepts. '0' or '\0' or simply NULL represents the conclusion of the character string or NULL byte. Because there is no specified mark associated with the NULL character, it is not needed. That's the primary purpose for which it acts as a string terminator.

Remember: The memory space for each character NULL holds is 1 byte.

Null Character in C

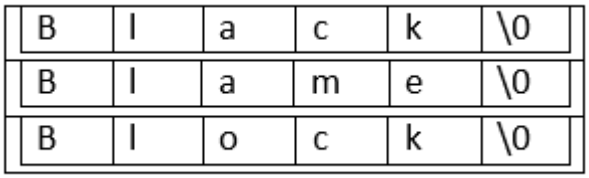
A NULL byte terminates multiple ideas in the C computer language, not just strings or arrays. A NULL byte is utilized to denote the end of a string in concepts such as arrays, string literals, and character strings. This is best demonstrated with an array illustration.

Assume we have a 10-dimensional array, and we require to keep the string "computer" within it. This is easily accomplished with the following code;

Using Two-dimensional Arrays:

Creating a String Array is one of the applications of two-dimensional Arrays. To get a picture of the arrangement, observe the below representation:

For suppose we want to create an Array of 3 Strings of size 5:



Every String in a String Array must terminate with a null Character. It is the property of a String in C.

Syntax to create a 2D Array:

1. Data_type name[rows][columns] = {{values in row 1}, {values in row 2}...};

Syntax to create a String Array:

1. **char** Array[rows][columns] = {"String1", "String2"...};

Now, let us create an example String Array:

- Observe that when we assign the number of rows and columns, we need to consider the Null Character to the length.

```
1. #include<stdio.h>
2. int main()
3. {
4.     int i;
5.     char Array[3][6] = {"Black", "Blame", "Block"};
6.     printf("String Array: \n");
7.     for(i = 0; i < 3; i++)
8.     {
```

```
9.          printf("%s\n", Array[i]);
10.     }
11.     return 0;
12. }
```

Output:

```
String Array:
Black
Blame
Block
```

- char Array[3][6] = {"Black", "Blame", "Black"} -> {{'B', 'l', 'a', 'c', 'k', '\0'}, {'B', 'l', 'a', 'm', 'e', '\0'}, {'B', 'l', 'a', 'c', 'k', '\0'}}
- We cannot directly manipulate the Strings in the Array as a String is an immutable data type. The compiler raises an error:

```
1. char Array[0] = "Hello";
```

Output:

```
[Error] assignment to expression with Array type
```

- We can use the strcpy() function to copy the value by importing the String header file:

```
1. char Array[3][6] = {"Black", "Blame", "Block"};
2. strcpy(Array[0], "Hello");
3. for(i = 0; i < 3; i++)
4. {
5.     printf("%s\n", Array[i]);
6. }
```

Output:

```
String Array:
Hello
Blame
Block
```

The Disadvantage of using 2D Arrays:

Suppose we want to store 4 Strings in an Array: {"Java", "T", "point", "JavaTpoint"}. We will store the Strings like this:

J	a	v	a	\0						
T	\0									
p	o	i	n	t	\0					
J	a	v	a	T	p	o	i	n	t	\0

- The number of rows will be equal to the number of Strings, but the number of columns will equal the length of the longest String.
- The memory allocated to all the Strings will be the size of the longest String, causing "**Memory wastage**".
- The orange part in the above representation is the memory wasted.

Command Line Arguments in C

- The most important function of C is the main() function. It is mostly defined with a return type of int and without parameters.

```
int main() {

    ...

}
```

We can also give command-line arguments in C. Command-line arguments are the values given after the name of the program in the command-line shell of Operating Systems. Command-line arguments are handled by the main() function of a C program.

To pass command-line arguments, we typically define main() with two arguments: the first argument is the **number of command-line arguments** and the second is a **list of command-line arguments.**

Syntax

```
int main(int argc, char *argv[]) { /* ... */ }

            or

int main(int argc, char **argv) { /* ... */ }
```

Here,

- **argc (ARGument Count)** is an integer variable that stores the number of command-line arguments passed by the user including the name of the program. So if we pass a value to a program, the value of argc would be 2 (one for argument and one for program name)
- The value of argc should be non-negative.
- **argv (ARGument Vector)** is an array of character pointers listing all the arguments.

- If argc is greater than zero, the array elements from argv[0] to argv[argc-1] will contain pointers to strings.
- argv[0] is the name of the program , After that till argv[argc-1] every element is command -line arguments.

For better understanding run this code on your Linux machine.

Example

The below example illustrates the printing of command line arguments.

- C

```
// C program named mainreturn.c to demonstrate the working

// of command line arguement

#include <stdio.h>



// defining main with arguments

int main(int argc, char* argv[])

{

    printf("You have entered %d arguments:\n", argc);



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

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

    }

    return 0;

}
```

Output

```
You have entered 4 arguments:

./main
```

```
geeks
for
geeks
```

Creating a pointer for the string

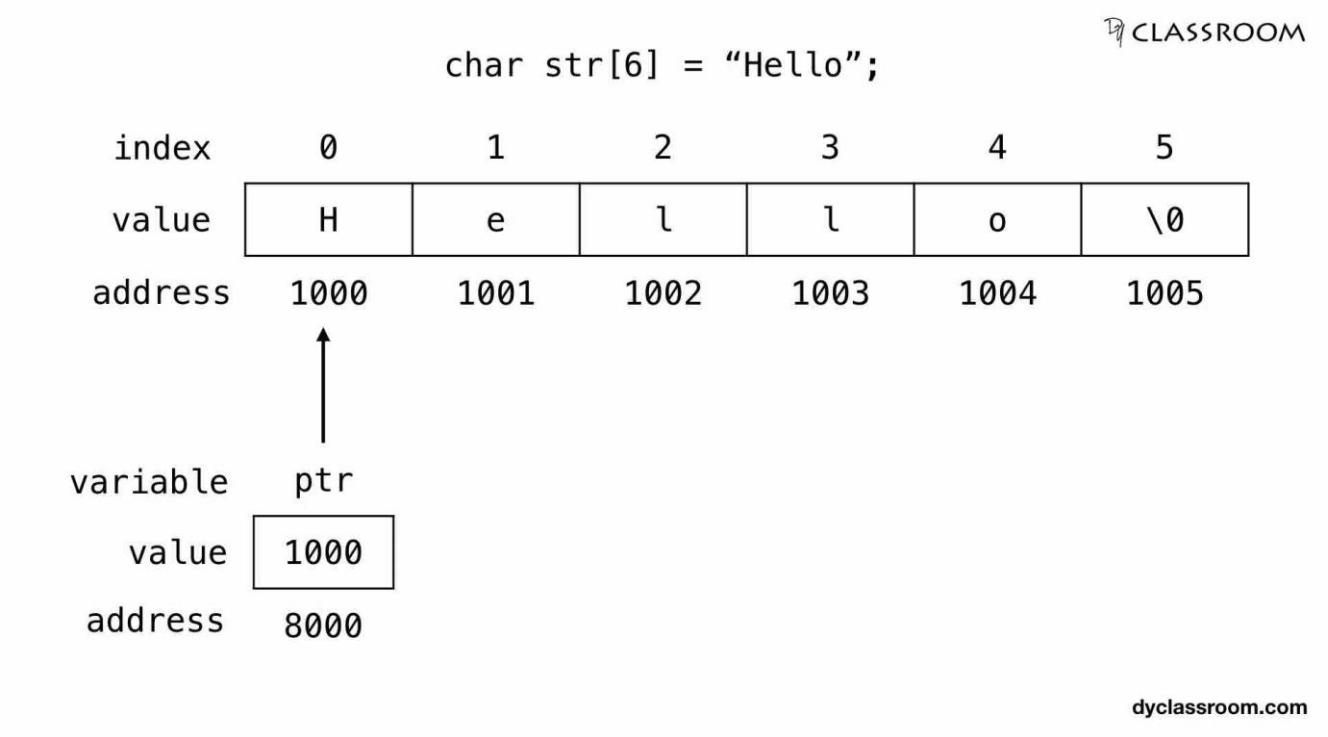
The variable name of the string str holds the address of the first element of the array i.e., it points at the starting memory address.

So, we can create a character pointer ptr and store the address of the string str variable in it. This way, ptr will point at the string str.

In the following code we are assigning the address of the string str to the pointer ptr.

```
char *ptr = str;
```

We can represent the character pointer variable ptr as follows.



The pointer variable ptr is allocated memory address 8000 and it holds the address of the string variable str i.e., 1000.

Accessing string via pointer

To access and print the elements of the string we can use a loop and check for the `\0` null character.

In the following example we are using `while` loop to print the characters of the string variable `str`.

```
#include <stdio.h>

int main(void) {

  // string variable
  char str[6] = "Hello";

  // pointer variable
  char *ptr = str;

  // print the string
  while(*ptr != '\0') {
    printf("%c", *ptr);

    // move the ptr pointer to the next memory location
    ptr++;
  }

  return 0;
}
```

Using pointer to store string

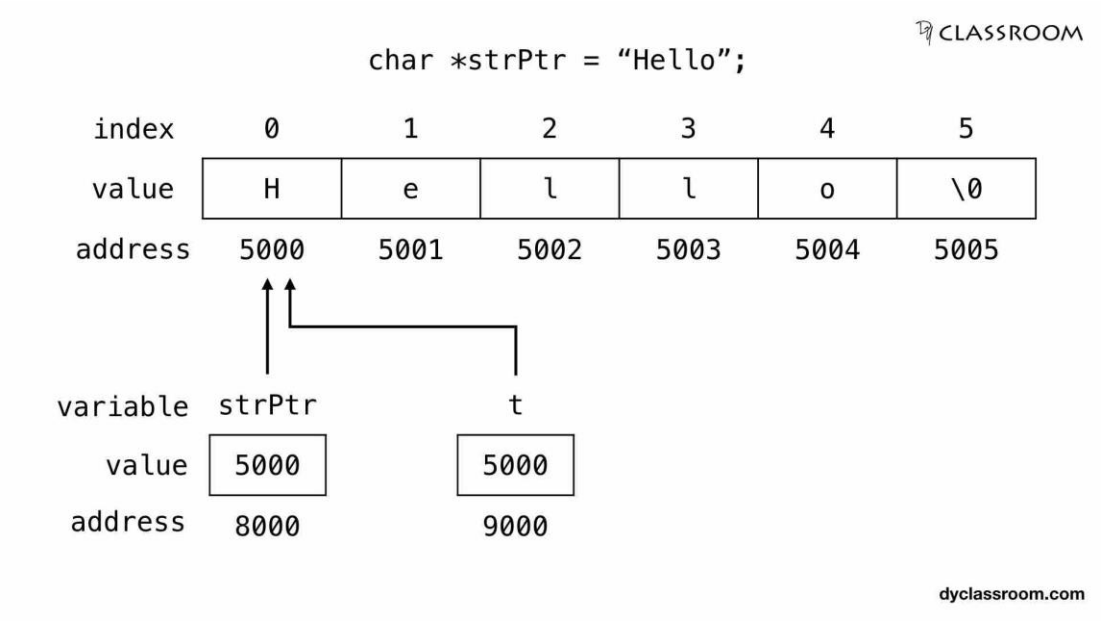
We can achieve the same result by creating a character pointer that points at a string value stored at some memory location.

In the following example we are using character pointer variable `strPtr` to store string value.

```
#include <stdio.h>

int main(void) {

  // pointer variable to store string
```

```
  char *strPtr = "Hello";

  // temporary pointer variable
  char *t = strPtr;

  // print the string
  while(*t != '\0') {
    printf("%c", *t);

    // move the t pointer to the next memory location
    t++;
  }

  return 0;
}
```

Note! In the above code we are using another character pointer `t` to print the characters of the string as because we don't want to lose the starting address of the string "Hello" which is saved in pointer variable `strPtr`.



In the above image the string "Hello" is saved in the memory location 5000 to 5005.

The pointer variable `strPtr` is at memory location 8000 and is pointing at the string address 5000.

The temporary variable is also assigned the address of the string so, it too holds the value 5000 and points at the starting memory location of the string "Hello".

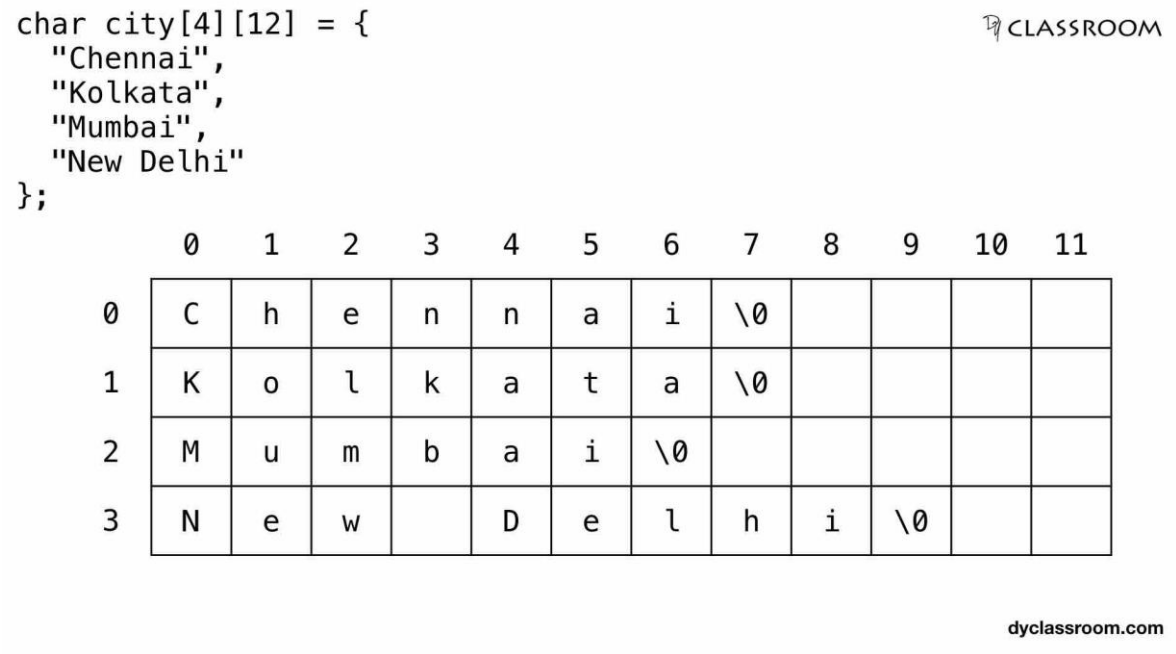
Array of strings

We can create a two dimensional array and save multiple strings in it.

For example, in the given code we are storing 4 cities name in a string array city.

```
char city[4][12] = {
  "Chennai",
  "Kolkata",
  "Mumbai",
  "New Delhi"
};
```

We can represent the city array as follows.



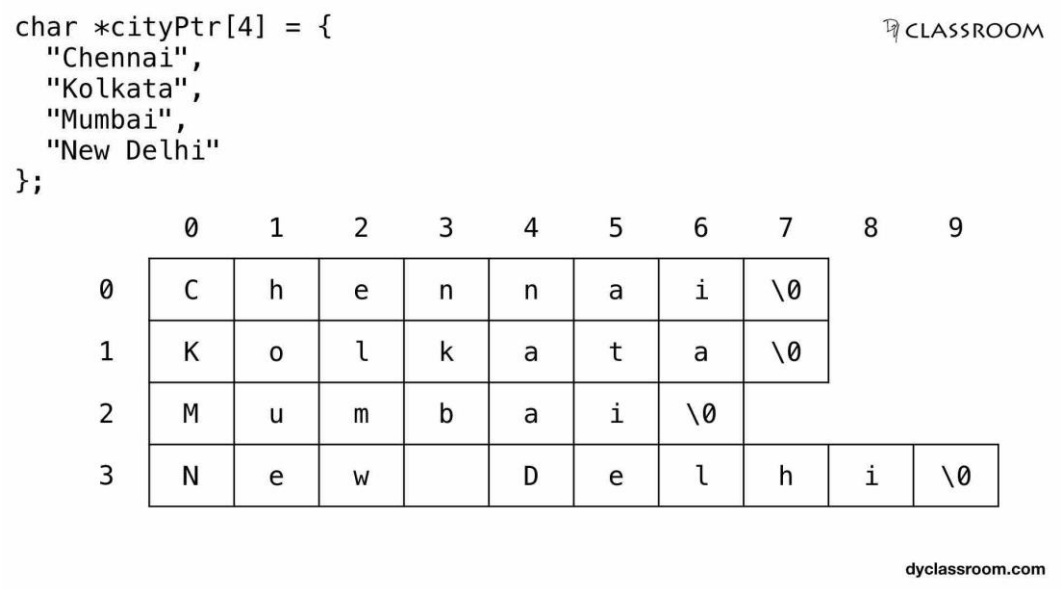
The problem with this approach is that we are allocating 4x12 = 48 bytes memory to the city array and we are only using 33 bytes.

We can save those unused memory spaces by using pointers as shown below.

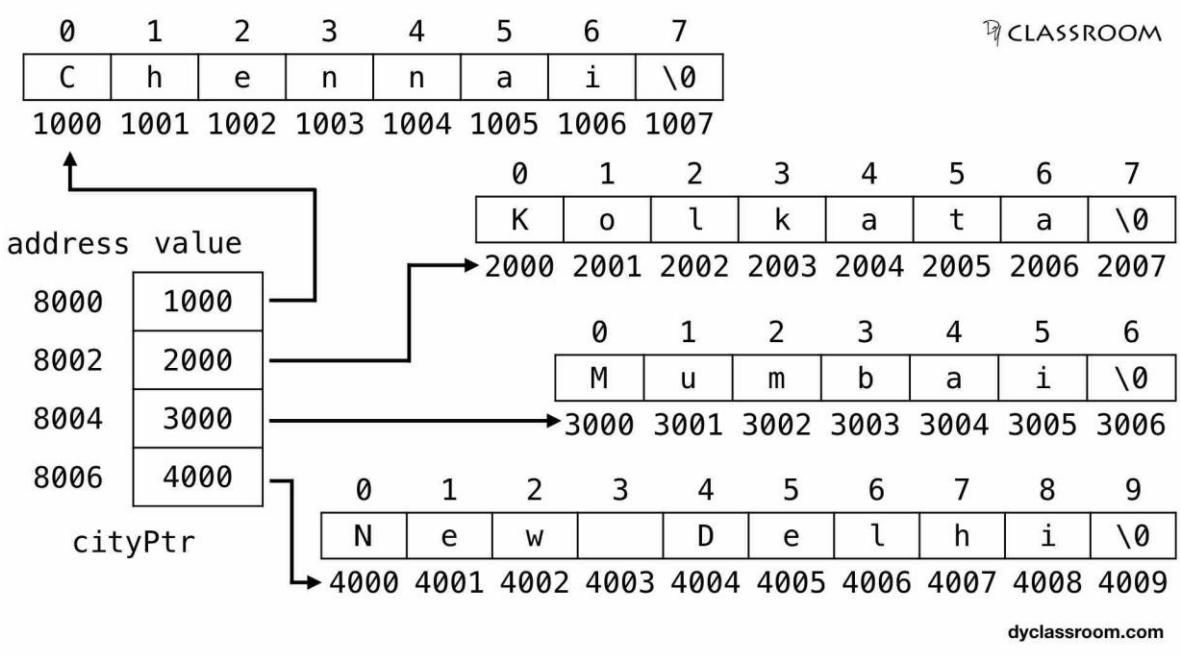
```
char *cityPtr[4] = {
  "Chennai",
  "Kolkata",
  "Mumbai",
  "New Delhi"
};
```

In the above code we are creating an array of character pointer cityPtr of size 4 to store the name of the four cities.

We can represent the array of pointers as follows.



The above array of pointers can be represented in memory as follows.



The cityPtr pointer variable is allocated the memory address 8000 to 8007. Assuming integer address value takes 2 bytes space. So, each pointer gets 2 bytes.

Name of the cities are saved in locations 1000, 2000, 3000 and 4000.

Accessing values pointed by array of pointers

To access and print the values pointed by the array of pointers we take help of loop as shown in the following example.

```
#include <stdio.h>

int main(void) {

  // array of pointers
  char *cityPtr[4] = {
    "Chennai",
    "Kolkata",
    "Mumbai",
    "New Delhi"
  };

  // temporary variable
```

```
  int r, c;

  // print cities
  for (r = 0; r < 4; r++) {
    c = 0;
    while(*(cityPtr[r] + c) != '\0') {
      printf("%c", *(cityPtr[r] + c));
      c++;
    }
    printf("\n");
  }

  return 0;
}
```

Output:

```
Chennai
Kolkata
Mumbai
New Delhi
```

In the above code we are using the `r` variable to access each row of the pointer. And we are using the `c` variable to access each character in a selected row.

Finding total number of characters in a string

To find the total number of characters in a string we use the `strlen()` function which is from the `string.h` header file.

Syntax of finding the length of a string is given below.

```
int len = strlen(str);
```

Where, `str` is the name of the variable holding the string.

The following code will print 5 as there are five characters in the string "Hello".

```
#include <stdio.h>
#include <string.h>
int main(void)
{
  //variable
  char str[] = "Hello";

  printf("%ld", strlen(str));    //this will give us 5
  return 0;
}
```

Comparing two strings

To compare two strings in C we use the `strcmp()` method from the `string.h` header file.

The `strcmp()` function compares two strings lexicographically and will return an integer value.

- If returned value is negative then string1 < string2 i.e., string1 is lexically above string2.
- If returned value is 0 then string1 and string2 are identical.
- If returned value is positive then string1 > string2 i.e., string1 is lexically below string2.

We can't use (str1 == str2) or ("Hello" == "Hi") to compare strings.

In the following example we will take two strings as input from the user and check if they are equal or not.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
  //variable
  char str1[100], str2[100];

  //input
```

```
  printf("Enter string 1: ");
  gets(str1);

  printf("Enter string 2: ");
  gets(str2);

  if (strcmp(str1, str2) == 0) {
    printf("Both are same.\n");
  }
  else {
    printf("Both are different.\n");
  }

  printf("End of code\n");
  return 0;
}
```

Output: Same string

```
Enter string 1: Apple
Enter string 2: Apple
Both are same.
End of code
```

Output: Different string

```
Enter string 1: Apple
Enter string 2: Mango
Both are different.
End of code
```

Concatenate two strings

We use the `concat()` function from the `string.h` header file to concatenate two strings.

In the following example we will concatenate "Hello" and "World".

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

```
{
  //variable
  char
    str1[] = "Hello",
    str2[] = "World",
    str3[100] = "";

  //concat
  strcat(str3, str1);  //concat "Hello" to str3 so, str3 = "Hello"
  strcat(str3, " ");   //concat " " to str3     so, str3 = "Hello "
  strcat(str3, str2);  //concat "World" to str3 so, str3 = "Hello World"

  printf("Concatenated string: %s\n", str3);

  printf("End of code\n");
  return 0;
}
```

Output

```
Concatenated string: Hello World
End of code
```

Reverse string

To reverse a string all we have to do is swap the last character with the first character, the second last character with the second character and so on till we reach the middle of the string.

So, if we have a string "Hello" then its reverse is "olleH".

In the following example we will take a string having less than 100 characters from user and will reverse it.

```
#include <stdio.h>
#include <string.h>
int main(void)
{
  //variable
  char str[100], tmp;
  int i, len, mid;
```

```
  //input
  printf("Enter a string: ");
  gets(str);

  //find number of characters
  len = strlen(str);
  mid = len/2;

  //reverse
  for (i = 0; i < mid; i++) {
    tmp = str[len - 1 - i];
    str[len - 1 - i] = str[i];
    str[i] = tmp;
  }

  //output
  printf("Reversed string: %s\n", str);

  printf("End of code\n");
  return 0;
}
```

Output

```
Enter a string: Hello World
Reversed string: dlroW olleH
End of code
```

Assignment 3

1.List any three string handling function with their usage.

2.What is command line argument? Give advantages of command line argument.

3.What is purpose of strtok() function?

4.Write a C program to check whether string is pailnderome or not

5.Write a C program to accept string and character as command line argument and replace each occurrence of character in string by given character

6.Explain any five string handling functions with their usage