

Part-1:

- **Arrays:** Concepts, Using Array in C, Array Application, Two Dimensional Arrays, Multidimensional Arrays, Programming Example

Part-2:

- **Strings:** String Concepts, C String, String Input / Output Functions, Arrays of Strings, String Manipulation Functions, Programming Example
- **Enumerated, Structure, and Union:** The Enumerated Types, Structure, Unions, Type Definition (Type def) and Programming Application.

STRINGS

A. What is a string in C?

A C string is a variable-length array of characters that is delimited by a \0 (null) character.

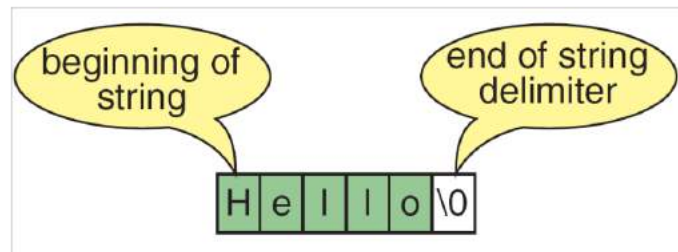
1. A string is a sequence of characters in C,
2. Every string is enclosed within double quotes “ ”,
3. The C compiler automatically adds a '\0' (null) character at the end of a string,
4. Strings are created using a one-dimensional array of 'char' datatype.
5. **Empty String:** A space in “ ” create an **empty string** by simply adding a \0 (null) character to it.
6. **Size:** the **size** of a string = Number of characters in the string + 1

Note: \0 (null) character takes up 1-byte space at the end of a string. Hence, we must consider one extra space while declaring the size of a string.

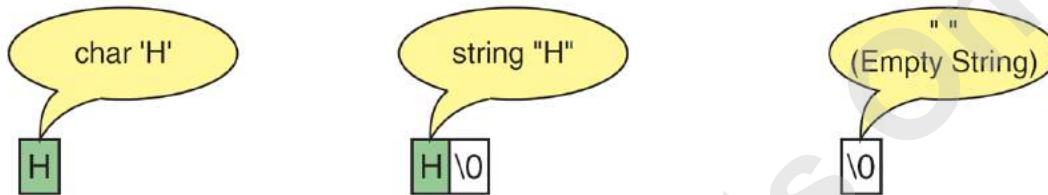
String vs Array of characters:

1. **String:** A **string** must be enclosed in “ ” and the \0 (null) character is automatically added at the end.
2. **Array:** Characters in ‘ ’ create a character array **with NO** null character at the end. That is *NOT a string!*

Storing a string "Hello"



Storing Strings and Characters



String vs Character Array



String Constants or Literals:

A series of characters enclosed in **double quotes** “ ” are called **string constants**. The compiler automatically **adds a \0 (null) character** at the end of each string. The string constants are also called **string literals**.

Ex: `char str[9] = "Software"` **OR**

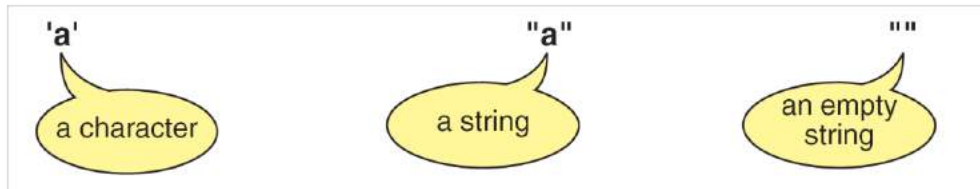
Ex: `char str[9] = 'S','o','f','t','w','a','r','e','\0'`

Array of Character Constants or Literals:

One or a series of characters enclosed in **single quotes** ‘ ’ is called a character constant. It **DOES NOT** add a \0 (null) character at the end. Hence, it is called an array of characters; **not a string**.

Ex: `'S','o','f','t','w','a','r','e'`

String Literals vs Character Literals



We can access a string literal using its index number as follows:

<pre>//Access String Literal using its index number #include<stdio.h> int main() { printf("%c", "India"[1]); return 0; }</pre>	<table border="1"> <tr><td><code>"India"[0]</code></td><td><code>I</code></td></tr> <tr><td><code>"India"[1]</code></td><td><code>n</code></td></tr> <tr><td><code>"India"[2]</code></td><td><code>d</code></td></tr> <tr><td><code>"India"[3]</code></td><td><code>i</code></td></tr> <tr><td><code>"India"[4]</code></td><td><code>a</code></td></tr> <tr><td><code>"India"[5]</code></td><td><code>\0</code></td></tr> </table>	<code>"India"[0]</code>	<code>I</code>	<code>"India"[1]</code>	<code>n</code>	<code>"India"[2]</code>	<code>d</code>	<code>"India"[3]</code>	<code>i</code>	<code>"India"[4]</code>	<code>a</code>	<code>"India"[5]</code>	<code>\0</code>	<p>Output: n</p>
<code>"India"[0]</code>	<code>I</code>													
<code>"India"[1]</code>	<code>n</code>													
<code>"India"[2]</code>	<code>d</code>													
<code>"India"[3]</code>	<code>i</code>													
<code>"India"[4]</code>	<code>a</code>													
<code>"India"[5]</code>	<code>\0</code>													

B. Declaration of Strings in C:

There are 3 methods to declare and create strings in C.

1. **1-D String array** of character datatype (static memory allocation)
2. **2-D Strings array** character datatype (static memory allocation)
3. **Pointer Array** of character datatype (dynamic memory allocation)

[Note: Method 1 & 2 will be explained here. Method 3 will be explained later in Pointers section]

Declaration of Strings in a 1-D array:

In C, strings are created as a one-dimensional array of character datatype. When we create a string, the size of the array must be 1 more than the actual number of characters to be stored. The 1 extra memory block is used to store the END of the string character `'\0'` (null).

Syntax: Declaration

dataType StringName[size];

dataType is usually 'char',

StringName is any name given to the string variable,

size is the length of the string, i.e the number of characters stored in the string.

Ex: `char str[9];`

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]

String Declaration in C

C. Initializing or Assigning a String value during Declaration of variable

A string value can be initialized or assigned in 5 ways.

1. Assigning a string without size

```
char str1[ ] = "Computer";
```

2. Assigning a string with size

```
char str2[9] = "Computer";
```

3. Assigning character by character without size

```
char str3[ ] = 'C','o','m','p','u','t','e','r','\0';
```

4. Assigning character by character with size

```
char str4[9] = 'C','o','m','p','u','t','e','r','\0';
```

5. Pointer String

```
char *ptr = "Computer";
```

Example

```
//String Declaration & Initialization
//5 ways to Initialize during declaration
#include<stdio.h>
int main()
{
    //--- 5 String Initializations during declaration ---
    //String in double quotes
    //Automatically appends NULL character \0 at the end
    char str1[] = "Computer";
    char str2[9] = "Computer";

    //String in an array of chars in single quotes,
    //we MUST Manually append NULL character \0 at the end
    char str3[] = {'C','o','m','p','u','t','e','r','\0'};
}
```

```
char str4[9] = {'C', 'o', 'm', 'p', 'u', 't', 'e', 'r', '\0'};
printf("\n %s", str4);

//Pointer string
char *ptr = "Computer";
printf("\n %s", ptr);
return 0;
}
```

Ex: `char str[9] = "Computer";`
Str - name of the string
9 - number of characters in the string;
"Computer" - the value of the string
0-8 - are index numbers

str[0]	str[1]	str[2]	str[3]	str[4]	str[5]	str[6]	str[7]	str[8]
C	o	m	p	u	t	e	r	\0

String Initialization in C

6. Assigning or Initialization a string value AFTER Declaration

Arrays and strings do not support the assignment operator after they are declared.

```
char s1[9];
s1 = "Computer"; // ERROR! array type is not assignable.
```

So, we must use the **strcpy()** function in **string.h** to copy a string value into a variable.

```
strcpy(s1, "Computer"); //copies "computer" string into s1 variable
```

Example:

```
//Assign string value by using strcpy(dest, source) function
//You must include string.h header file
#include<stdio.h>
#include<string.h>
int main()
{
    //Declaration
    char s1[9];
    //Assigning or Initialization of string values
    strcpy(s1, "Computer"); //using function from string.h header file
    printf("\n %s", s1);
    return 0;
}
```

Output :

Computer

Here, “**strcpy**” function copies “**Computer**” string into “**s1**” string variable.

Note :

Strings do not need to be printed character by character like in an array.

Strings can be printed using “**printf**” statement using “**%s**” format specifier.

D. Accessing string value (Formatted & Unformatted Input/Output functions in strings)

We can read or print strings in either formatted or unformatted methods.

[formatted input/output]

1. **scanf()** - reads single word using **%s** format; **printf()** - prints a string using **%s** format

[unformatted input/output]

2. **gets()** - reads a line of text from **stdin** until new line, ex: <enter> key
3. **fgets()** - reads a line of text from **stdin** until new line, ex: <enter> key, Or reads a line of text from a file until EOF (end of file)
4. **puts()** - prints a line of text or string to output stream **stdout** without the null character and appends new line (**\n**) character.

1. scanf()

- Using **scanf()** method we can read **only one word** of the string.
- We use **%s** format specifier to represent a string in “**scanf()**” and “**printf()**” methods.
- **No &** is required before string variable in “**scanf()**”, Ex:
 - **myName[]** is a string array of characters;
 - “**myName**” without ‘**[**’ and ‘**]**’ will gives the base address of the string variable;
 - No need to use **&** before **myName** while inputting string values.

Example: Input string value using scanf()

```
#include<stdio.h>
int main() {
    char myName[30];
    printf("\n Enter your name : ");
    scanf ("%s", myName);
```

```
printf("Hi! %s, Welcome to Software!", myName);  
return 0;    }
```

2. gets()

- We use a **gets()** to read multiple words or a line of text,
- **Enter** character terminates the **stdin** (input of text from keyboard),
- No need to use **'&'** before string variable name.

Syntax: `gets(varName);`

varName is the name of the string variable where the string will be saved.

Example: Input string using gets() & Output string using puts()

```
#include<stdio.h>  
int main() {  
    char myName[50];  
    printf("\n Enter your name : ");  
    gets(myName);  
    printf("Hi! ");  
    puts(myName);  
  
    return 0;  
}
```

3. fgets()

- We use a **fgets()** to read a line of string from **stdin** or a **file**.
- **Enter** key terminates the **stdin** (input of text from keyboard),
- **EOF** terminates reading from a file,
- No need to use **'&'** before string variable name.

Syntax:

```
char *fgets(char *str, int n, stdin or FILE *stream);
```

str is the name of the string variable where the string will be saved.

n - maximum number of characters

stdin - a source from the keyboard (Or)

FILE *stream - a source from file

Example: Read string using **fgets()** & Output string using **puts()**

```
//Input a line of text using fgets()
#include <stdio.h>
#define MAX 100
int main()
{
    char str[MAX];
    //Input string from Keyboard (stdin) using "fgets"
    fgets(str, MAX, stdin);

    printf("Your string is: \n");
    // Output string to console using "puts"
    puts(str);

    return 0;
}
```

Note:

- **gets() function is removed from the C standard** because it allows you to input any length of characters. Hence, there might be a buffer overflow. So, **use fgets()** function.
- **fgets()** is the preferred method compared to gets() to input string values. fgets() function allows specifying buffer size and input more than the buffer size.

4. puts()

Syntax: int puts(const char str[]);

- **str** is the string variable's name.
- Writes a string to **stdout** without the \0 (null) character.
- A newline character (\n) is appended to the output.
- On success a positive value is returned.
- On error EOF is returned.

2-D Array of Strings

1. **Description:** Storing 2 or more strings in an array is called a 2-D Array of Strings.

2. **Declaration:**

```
char arrayName[n][m];
```

char: Type of values stored in the array

arrayName: Name of the 2-D array of strings

n: Max number of strings in the array

m: Max number of characters in each string

3. **Initialization:**

```
char depts[4][12] = { "Computers", "Electronics", "Electrical", "Civil" };
```

- 4 strings are saved in the "depts" string array
- 10 characters can be saved in each of the 4 strings

Memory	0	1	2	3	4	5	6	7	8	9	10	11
95710	C	o	m	p	u	t	e	r	s	\0		
95720	E	l	e	c	t	r	o	n	i	c	s	\0
95730	E	l	e	c	t	r	i	c	a	l	\0	
95740	C	i	v	i	l	\0						

4. **Access 2-D Array of Strings**

<p>Input for(i=0 ; i<4 ; i++) scanf("%s",&name[i][0]);</p>	<p>The second subscript is [0] because,</p> <ul style="list-style-type: none"> • before entering any string the length of the string is 0, and • name[i][0] points to starting address of each string.
<p>Output for(i=0 ; i<4 ; i++) printf("%s \n", name[i]);</p>	<p>Note: This type of code prints the strings and eliminates any garbage values after '\0'.</p>

5. Example: Program to search a string in 2-D array of String array

```

//Search a string in 2-D array of strings
#include <stdio.h>
#include <string.h>
int main() {
    char names[3][10]; //2-D string array
    char item[10];     //1-D string array to search
    int i, res, status = 0;
    /* Input 3 names */
    printf("Enter 3 names:\n");
    for (i = 0; i < 3; i++)
        scanf("%s", &names[i][0]);
    /*Enter a name to search in the string array*/
    printf("Enter the name to be searched:\n");
    scanf("%s", &item);

    /*Finding the item in the string array*/
    for (i = 0; i < 3; i++)
    { res = strcmp(&names[i][0], item);
      // compares the string in the array with the item and
      // if match is found returns 0 and stores it in variable res
      if (res == 0)
          status = i;
    }
    if (status == 0) //match is not found
        printf("Given name does not match any name in the list");
    else ///match is found
        printf("Found. Name in the array exists at index : %d", status);
    return 0;
}

```

Output:

Enter 3 names:

Santosh

Vamsi

Abdulla

Enter the name to be searched:

Vamsi

Found. Name in the array exists at index - 1

String Library Functions

String Library Functions are predefined functions in C. These functions are used to manipulate string values. They are defined in “string.h” header file. We must include **string.h** to use any string handling function.

The following table provides the most commonly used string handling function and their use.

Function	Syntax	Description and Example
strlen()	strlen(str);	Returns length of str. char str[] = "Logic"; printf("Length of string is: %d", strlen(str));
strcpy()	strcpy(dest, src);	The string in src will be copied to dest. //strcpy() copy a string #include<stdio.h> #include<string.h> int main() { char src[] = "Engineer"; char dest1[10]; char dest2[10]; strcpy(dest1, src); strcpy(dest2, "Guntur"); printf("%s", dest1); //Engineer printf("%s", dest2); // Guntur return 0; } Output: Engineer Guntur
strncpy()	strncpy(dest, src, 5)	Copies first 5 characters of src into dest
strcat()	strcat(dest, src)	Appends src string to dest. //strcat() string concatenation #include<stdio.h> #include<string.h> int main() { char dest[50] = "Our"; char src[50] = " Plan"; strcat(dest, src); printf("%s", dest) ; // Our Plan return 0; } Output: Our Plan
strncat()	strncpy(dest, src, 4)	Appends first 4 characters of src string to dest

PTC UNIT - III, Part-2 (Strings; Struct-union-enum-typedef)

<p>strcmp()</p>	<p>strcmp(leftStr, rightStr);</p>	<p>strcmp() compares 2 strings. Checks ASCII value character by character. Returns: 0 if all chars are equal, 1 if a char in 1st string is greater, -1 if a char in 1st string is lesser.</p> <pre>#include<stdio.h> #include<string.h> int main() { char str1[] = "abz"; char str2[] = "abc"; int res = strcmp(str1, str2); if (res==0) printf("Strings are equal"); else if (res > 0) printf("str1 is greater than str2"); else printf("str1 is less than str2"); printf("\nValue returned by strcmp() is: %d" , res); return 0; }</pre> <p>Output: str1 is greater than str2 Value returned by strcmp() is: 1</p>
<p>strlwr()</p>	<p>strlwr(string1)</p>	<p>Converts all the characters of string1 to lower case.</p>
<p>strupr()</p>	<p>strupr(string1)</p>	<p>Converts all the characters of string1 to upper case.</p>
<p>strrev()</p>	<p>strrev(string1)</p>	<p>It reverses the value of string1.</p>
		<pre>#include<stdio.h> #include<string.h> int main() { char str1[] = "Best Engineer"; //converts string into uppercase. printf("%s\n", strupr(str1)); //converts string into uppercase. printf("%s\n", strlwr(str1)); //converts string into uppercase. printf("%s\n", strrev(str1)); return 0; }</pre> <p>Output: BEST ENGINEER best engineer reenigne tseb</p>

PTC UNIT - III, Part-2 (Strings; Struct-union-enum-typedef)

Other String Functions		
strncmp()	strncmp(string1, string2, 4)	Compares first 4 characters of both string1 and string2
strcmpi()	strcmpi(string1, string2)	Compares two strings, string1 and string2 by ignoring case (upper or lower)
stricmp()	stricmp(string1, string2)	Compares two strings, string1 and string2 by ignoring case (similar to strcmpi())
strdup()	string1 = strdup(string2)	Duplicated value of string2 is assigned to string1
strchr()	strchr(string1, 'b')	Returns a pointer to the first occurrence of character 'b' in string1
strrchr()	strrchr(string1, 'b')	Returns a pointer to the last occurrence of character 'b' in string1
strstr()	strstr(string1, string2)	Returns a pointer to the first occurrence of string2 in string1
strset()	strset(string1, 'B')	Sets all the characters of string1 to given character 'B'.
strnset()	strnset(string1, 'B', 5)	Sets first 5 characters of string1 to given character 'B'.
atoi()	int atoi(const char *string)	<p>Converts a string to an integer Returns the integer value, if successfully.</p> <ol style="list-style-type: none"> Returns 0, if the string starts with an alphanumeric character or only contains alphanumeric characters. Converted to Integer, if string starts with a numeric character & followed by an alphabet. It converts the number to an integer until the occurrence of the first alphabet. <pre> #include<stdio.h> #include <stdlib.h> int main() { char str1[10] = "127"; char str2[10] = "Namaskar!"; char str3[10] = "77Hi!"; char str4[6] = "10.97"; int x1 = atoi(str1); int x2 = atoi(str2); </pre>

PTC UNIT - III, Part-2 (Strings; Struct-union-enum-typedef)

		<pre>int x3 = atoi(str3); int x4 = atoi(str4); printf("Convert'Namaskar!': %d\n", x1); printf("Converting '127': %d\n", x2); printf("Converting '77Hi!': %d\n", x3); printf("Converting '10.97': %d\n", x4); return 0; }</pre>
--	--	--

Example: Program to demonstrate common String Library Functions from string.h

```
//gets(), strlen(), strcpy(), strcat(), strcmp(),strupr(), strlwr()
#include <stdio.h>
#include <string.h>
int main() {
// declaring string variables
char name1[20], name2[30], myname[50];
// Input 2 strings
printf("Enter 1st name: ");
gets(name1); // input 1st string
printf("Enter 2nd name: ");
gets(name2); // input 2nd string
// prints the length of the name1[] string
printf("Length of 1st name: %d\n",strlen(name1));

// concatenates the two strings and stores the result in name1[]
printf("Both names are: %s\n", strcat(name1, name2));

// copying the string in name1[] to myname[]
strcpy(myname, name1);
printf("Copied string to myname: %s\n", myname);

// compare the two strings
printf("Compare name1 & name2: %d\n", strcmp(name1, name2));
// convert the string to lowercase
printf("Lower case name1: %s\n",strlwr(name1));
// convert the string to uppercase
printf("Upper case name1: %s\n",strupr(name1));
return 0;
}
```

Output:

```
Enter 1st name: The Program
Enter 2nd name: Logic
Length of 1st name: 11
Both names are: The Program Logic
Copied string to myname: The Program Logic
Compare name1 & name2: 1
Lower case name1: the program logic
Upper case name1: THE PROGRAM LOGIC
```

```

Example: /* Program to Reverse a string */
#include <stdio.h>
#include <string.h>
int main()
{
char ptc[100]; // to store input string
char ptcrev[100]; // to store reversed string
int len; //to save length of the string

printf("\nEnter a string : ");
fgets(ptc, sizeof ptc, stdin);
len=strlen(ptc);
int j=0;
//len-2 will point to last character
//Ex: Hello len=6 but index is 0 1 2 3 4
for(int i=len-2;i>=0;i--)
{
    ptcrev[j] = ptc[i];
    //printf("ptcrev[%d]=%c", j,ptcrev[j]);
    //printf(" ptc[%d]=%c\n", i,ptc[i]);
    j++;
}
ptcrev[j]='\0';
puts("Reversed String: ");
puts(ptcrev);

return 0;
}

/* Output:
Enter a string : Computer
Reversed String:
retupmoC
*/

```

Enumerated, Structure, and Union: The Enumerated Types, Structure, Unions, Type Definition (Type def), and Programming Application.

What are User-Defined Data Types in C?

- The User-Defined Data Type is derived from any existing data type in C.
- We can use them for extending the pre-defined data types that are already available in C.
- We can also create various customized data types of your own.

Why do we Need User-Defined Data Types in C?

- The Pre-defined data types (int, char, cfloat etc) and Derived data types (arrays) in C may not offer a wide variety of functions.
- The User-Defined Data Types in C help us define custom data types of our own based on our needs. These data types offer various functions on the basis of how one define them. Hence, these are termed as “User-Defined”.

The user-defined data types in C are

1. **enum** - Enumerated data type
2. **struct** - Structure data type
3. **union** - Union data type
4. **typedef** - Type Definition

1. Enumerated Types (**enum**) in C

Enumeration is the process of creating user defined datatype by assigning names to integral constants

- **enum** is used to create user-defined enumeration datatypes in C.
- The enum data types allow a user to create symbolic names of their own for a list of all integer constants that are related to each other.
- The enumeration helps to set names to integer constants.
- A program becomes more readable by using these names for integer numbers.

Syntax:

enum identifier {name1, name2, name3, ... }

Here, integer 0 will be assigned to name1, integer 1 will be assigned to name2 and so on.

We can also assign our own integral constants as follows.

enum identifier {name1 = 10, name2 = 30, name3 = 15, ... }

PTC UNIT - III, Part-2 (Strings; Struct-union-enum-typedef)

Now, the integer 10 will be assigned to name1, the integer 30 will be assigned to name2 and so on.

Example: Program for weekdays using ENUM

```
#include<stdio.h>
#include<conio.h>
enum day { Mon, Tue, Wed, Thu, Fri, Sat, Sun} ;

void main() {
    enum day today;
    today = Tue ;
    printf("\nThis day is %d ", today);
}
```

Output:

This day is 1

Explanation:

day is a user defined datatype with 7 values as below:

Mon = 0, Tues= 1, Wed= 2, Thu= 3, Fri= 4, Sat= 5, Sun= 6

So when we display **Tue**, the respective integral constant '1' will be displayed.

We can also change the order of integral constants, consider the following example program.

Example: Program for weekdays using ENUM with changed integral constant values

```
//Changing order of the integral constants
#include<stdio.h>
#include<conio.h>
enum day { Mon=1, Tue, Wed, Thu, Fri, Sat, Sun} ;

void main() {
    enum day today;
    today = Tue ;
    printf("\nThis day is %d ", today);
}
```

Output:

This day is 2

Explanation: In this program, the integral constant value starts with '1' instead of '0'. Here, Tuesday's value is displayed as '2'.

We may also define different integral values as we wish.

Example: Program for weekdays using ENUM with different integral constant values

```
//enum with defferent integral constant values
#include<stdio.h>
#include<conio.h>
enum grades {pass = 40, second = 60, first = 80, top = 100} ;
void main() {
    enum grades status;
    status = pass ;
    printf("\nYour are at %d marks. You must study well!", status) ;
}
```

Example: enum declaration

```
enum Boolean { true=1, false=0 };
```

- **enum** is the keyword used to define user defined data type
- **enum** members are basically integers
- Can use expressions like integers
- Makes code easier to read
- Cannot get string equivalent
- Ex: day++ always increments its value by 1
- more than one name can have same integral constant

2. Structures (**struct**) in C

- A structure is a collection of variables of different data types (non-homogenous) that are related to each other. For example, information of a person, an account, or a part, etc.
- Every data item present in a structure is called as a member. These members are also called fields.
- We use the **struct** keyword for creating a structure.

Advantages of Structures:

- Easy to access its members,
- Allocation of all the members is in a continuous memory,
- Faster to access its members

Structure is a collection of different type of elements under a single name that acts as user defined data type in C.

Syntax Declaration:

```
struct <structure_name>
{
    data_type1 member1;
    data_type2 member2, member3;
    -
    data_type_n member_n;
};
```

- **struct** is a keyword to declare a structure in C.
- **structure_name** is an identifier to use structure
- All members variables must be enclosed in curly braces
- Every structure must be terminated by a semicolon ;

Example Structure in C:

```
struct Student
{
    char stud_name[30];
    int reg_number;
    float average;
};
```

How to create & use a structure variable?

We create a structure variable in two ways.

1. while defining the structure and
2. in main() after terminating structure

How do we access a member of a structure?

To access members of a structure using structure variable, we should use dot (.) operator.

Example: Program to Create and Use structure variables in C

```
#include<stdio.h>
struct Student
{
    char stud_name[30];
    int reg_number;
    float average;
    char grade; //F for <40, B for 40-74, A for 75-100
} stud1;
int main(){
    struct Student stud2; // using struct keyword
    printf("Enter details of stud1 : \n");
    printf("Name : ");
    scanf("%s", stud1.stud_name);
    printf("Roll Number : ");
    scanf("%d", &stud1.reg_number);
    printf("average : ");
    scanf("%f", &stud1.average);

    //Find & set grade
    if(stud1.average<40)
        stud1.grade='F';
    else if(stud1.average>=40 && stud1.average<75)
        stud1.grade='B';
    else
        stud1.grade='A';
    printf("***** Student 1 Details *****\n");
    printf("Student Name      : %s\n", stud1.stud_name);
    printf("Student Reg. Number : %i\n", stud1.reg_number);
    printf("Student Average     : %f\n", stud1.average);
    printf("Student Grade      : %c\n", stud1.grade);
    return 0;    }
```

Output:

```
Enter details of stud1 :  
Name : Lokesh  
Roll Number : 20450  
average : 65
```

******* Student 1 Details *******

```
Student Name      : Lokesh  
Student Reg. Number : 20450  
Student Average   : 65.000000  
Student Grade     : B
```

Explanation:

- The stucture variable "stud1" is created while defining the structure
- The structure variable "stud2" is created in main() using struct keyword.
- To access the members of a structure, we use the dot (.) operator.

How the memory is allocated for a structure?

Memory does not get allocated while declaring a structure.

Memory does get allocated when we create the variable of a structure.

Size of memory allocated is equal = Sum of memory required for each member of the structure.

In the above example program, the variables stud_1 and stud_2 are allocated with 36 bytes of memory each.

struct Student

```
{  
    char stud_name[30];      30 bytes  
    int reg_number;         4 bytes  
    float average;         4 bytes  
    char grade;            1 byte  
}                          sum = 39 bytes
```

- All the members of a structure can be used simultaneously.
- Until variable of a structure is created no memory is allocated.
- Total memory of a structure variable = Sum of all the memory required by all members of that structure.

3. Unions in C

A union is a collection of elements of different data types (non-homogenous) that are not similar to each other. The union allows the storage of the unrelated elements in the very same memory location. It is also an user-defined data type.

- Unions and Structures are pretty similar to each other
- Difference between Union & Structure is that we can access just a single member of the Union at any given time.
- It is because Union creates memory only for one member that has the biggest size (or the highest number of bytes).
- Elements that are defined in a union are called members of union.
- We use (.) operator to access members of union.

How to create a union?

We declare the union using the “**union**” keyword, and we can access all the members of a Union using the (.) dot operator.

```
union union_name  
{ data_type variable_name1;  
  data_type variable_name2;  
  .  
  .  
  data_type variable_nameN;  
};
```

How to create & use a union variable?

We create a union variable in two ways.

1. while defining the union and
2. in main() after terminating union

How do we access a member of a union?

To access members of a union using union variable, we should use dot (.) operator.

Example: Program to Create and Use union variables in C

```
#include <stdio.h>
#include <string.h>
union Employee {
    char name[32];
    int age;
    float salary;
};
int main(){
    union Employee employee;
    /* Using one member of a union at a time */
    strcpy(employee.name, "Swathi");
    printf("Name = %s Address = %p\n", employee.name,
        &employee.name);

    employee.age = 20;
    printf("Age = %d Address = %p\n", employee.age,
        &employee.age);
    employee.salary = 1234.5;
    printf("Salary = %f Address = %p\n", employee.salary,
        &employee.salary);
    /* Printing all member variable of Union, Only last updated
        member will hold it's value remaining will contain garbage */
    printf("\nName = %s\n", employee.name);
    printf("Age = %d\n", employee.age);
    printf("Salary = %f\n", employee.salary);
    printf("\nSize of Union: %d", sizeof employee) ;
    return 0;
}
```

Output:

```
Name = Swathi Address = 0061FF00
Age = 20 Address = 0061FF00
Salary = 1234.500000 Address = 0061FF00
```

```
Name =
Age = 1150963712
Salary = 1234.500000
```

```
Size of Union: 32
```

How the memory is allocated for a union?

Memory does not get allocated while declaring a structure.

Memory does get allocated when we create the variable of a structure.

Size of memory allocated is equal = Size of the largest member of the union.

In the above example program, the variables stud_1 and stud_2 are allocated with 36 bytes of memory each.

```
union Employee {  
    char name [32];           32 bytes  
    int age;                  04 bytes  
    float salary;            04 bytes  
}emp;                         sum = 32 bytes
```

Here, the emp union variable has been allocated 32 bytes and all members share this memory because same memory exists for all members.

Difference Between Structure and Union in C

Key	Structure	Union
Definition	Structure is the collection of multiple variables of different data types that are related to each other.	Union is also the collection of multiple variables of different data types that are related to each other.
Memory Allocation	In a struct, memory is allocated for all members. All members are accessible in structure.	In the union, the memory is allocated only for its largest member. This single memory is shared by all members. Only one member is accessible in the union at any given time.
Syntax	Declaration of structure in C: struct struct_name{ type element1; type element2; . . } variable1, variable2, ...;	Declaration of a union in C: : union u_name{ type element1; type element2; . . } variable1, variable2, ...;
Size	Size of Structure = greater or sum of size of all the data members.	Size of union = Size of largest member among all data members.
Value storage	Each member is stored in a separate memory location. Hence, a structure can store separate values for different members.	Union has only one memory allocation of its largest member. All other members share this memory.. So at any given time, the union stores a single value of one of the members.
Initialization	In Structure multiple members can be can be initialized at same time.	However in Union, only the first member can get initialize at a time.

typedef in C

typedef is a keyword used to create alias name for the existing datatypes. Using typedef keyword we can create a temporary name to the primitive data types int, float, char and double.

Syntax:

```
typedef existing-datatype alias-name
```

typedef with int data type:

In the following example, Number is defined as alias name for integer datatype. So, we can use Number to declare integer variables.

```
#include<stdio.h>
typedef int Number;
int main(){
    Number a,b,c;    // Here a,b,&c are integer type of variables.
    printf("Enter two integer numbers: ") ;
    scanf("%d%d", &a,&b) ;
    c = a + b;
    printf("Sum = %d", c) ;
}
```

Output:

```
Enter two integer numbers: 2 5
Sum = 7
```

typedef with structure or union

```
#include<stdio.h>
typedef struct student
{
    char stud_name[50];
    int stud_rollNo;
}stud;

int main(){
    stud s1;
    printf("Enter the student name: ") ;
    scanf("%s", s1.stud_name);
    printf("Enter the student Roll Number: ");
    scanf("%d", &s1.stud_rollNo);
```

PTC UNIT - III, Part-2 (Strings; Struct-union-enum-typedef)

```
printf("\nStudent Information\n");
printf("Name - %s\nHallticket Number - %d", s1.stud_name,
s1.stud_rollNo);
}
```

Output:

```
Enter the student name: Gopi
Enter the student Roll Number: 553377
```

```
Student Information
Name - Gopi
Hallticket Number - 553377
```

Comments: In the above example program, stud is the alias name of student structure. We can use stud as datatype to create variables of student structure. Here, s1 is a variable of student structure datatype.

typedef with Arrays

In C programming language, typedef is also used with arrays. Consider the following example program to understand how typedef is used with arrays.

Example Program to illustrate typedef with arrays in C.

```
#include<stdio.h>
void main(){
    typedef int Items[50]; //Items acts like an integer array type
of size 50
    Items list = {10,20,30,40,50}; //Items is an array of integer
type with size 5.
    int i;
    printf("list elements are : \n") ;
    for(i=0; i<5; i++)
        printf("%d\t", list[i]) ;
}
```

Output:

```
List elements are :
10  20  30  40  50
```

Comments: In this program, Items is the alias name of integer array type of size 50. Here, list is an integer array of size 5.

typedef with Pointers

We can give a name to a pointer data type using typedef. See the following example.

```
#include<stdio.h>
#include<conio.h>
void main() {
    typedef int* intPointer;

    intPointer ptr; //ptr is a pointer variable of integer datatype.
    int qty = 100;

    ptr = &qty;

    printf("Address of a = %u ",ptr) ;
    printf("\nValue of a = %d ",*ptr);
}
```
