

CSE101-Lec#23

Strings

Outline

- Introduction to string
 - Declaration
 - Initialization
- Reading and writing strings
 - functions of the standard input/output library (stdio.h)
- Processing of strings.
 - String Manipulation Functions from the String Handling Library
 - Comparing strings
 - Determining the length of string
 - All string operations without inbuilt functions
 - Other programs related to strings

Fundamentals of strings

- **Strings**
 - Array of characters treated as a single unit called **string**:
 - Can include **letters, digits and special characters** (*, /, \$)
 - String literal (**string constant**) - written in **double quotes**
 - “Lovely Professional University.”

What is a String??

- String is a collection of characters terminated by null character
- Strings are arrays of characters
 - String is a **pointer** to first character (like array)
 - Value of string is the **address** of first character
- Each element of the string is stored in a **contiguous** memory locations.
- Terminated by a **null character** ('\0') which is automatically inserted by the compiler to indicate the **end of string**.

String Definition

- They are defined as

char array_name[size];

e.g. char carname[30];

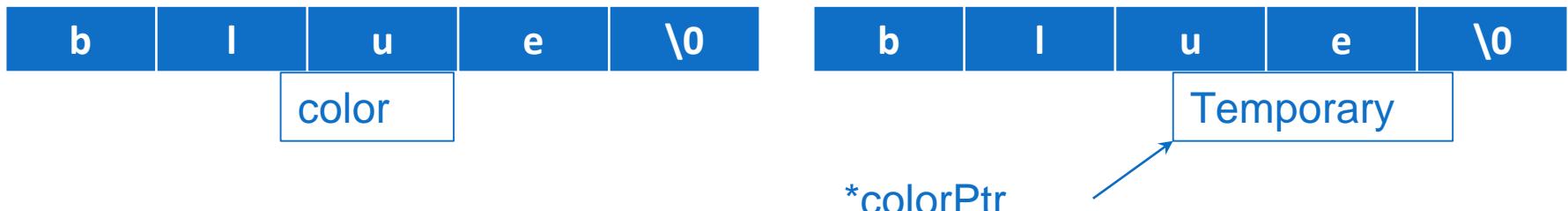
or char *carname;

- It defines an array name and reserves 30 bytes for storing characters and single character consumes 1 bytes each.
- Since the last byte is used for storing null character so total number of character specified by the user cannot exceed **29**.

String Initialization

- String Initialization
 - Two ways:
 - Define as a character array or a variable of type char *

```
char color[] = "blue"; //char array
Or char color[] = { 'b', 'l', 'u', 'e', '\0' };
char *colorPtr = "blue"; //pointer variable
```
 - Remember that strings represented as character arrays end with '\0'



String Input/Output

- Inputting strings
 - Use scanf.

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

 - Copies input into word[]
 - **Do not need & (because a string is a pointer)**
 - Remember to leave last place in the array for '\0'.
 - Because array knows no bounds the string written beyond char array size will **overwrite** the data in memory.
- Displaying strings
 - Use printf.

```
printf("%s", word);
```

Program to read and display string

```
#include <stdio.h>
int main()
{
    char carname[20]; //string char array
    printf("Enter the name of your car: ");
    scanf("%s", carname); // to display the input
    printf("\nName of car is %s", carname);
} //end main
```

```
Enter the name of your car: XUV500
Name of car is XUV500
```

Output



How?

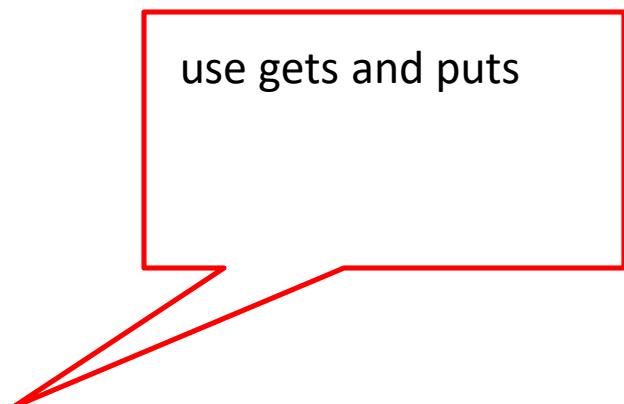
- The last program will print only a single word not the sentences with white spaces?
- That is if input is

Lovely Professional University
- Output will be:

Lovely
- So how to print:

Lovely Professional University

use gets and puts



Standard I/O Library Functions

- List of functions in `#include<stdio.h>`
- Used for string input/output functions.

Function	Description
<code>gets(char *s);</code>	Inputs characters from the standard input into the array s until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
<code>puts(const char *s);</code>	Prints the string s followed by a newline character.

Program to print strings with white spaces using library functions

```
#include <stdio.h>
int main()
{
    char name[100]; //string char array
    puts("\nEnter a string:");
    gets(name); //to input string with space
    printf("\nString is:")
    puts(name); //to output const string
} //end main
```

Enter a string:
Lovely Professional University
String is:
Lovely Professional University

Output

Drawback of gets() :

gets() has been removed from c11. So it might give you a warning when implemented.

We see here that it doesn't bother about the size of the array. So, there is a chance of Buffer_Overflow.

Alternative of gets():

To overcome the above limitation, we can use **fgets** as :

Syntax : char *fgets(char *str, int size, FILE *stream)

Example : fgets(str, 20, stdin); as here, 20 is

MAX_LIMIT according to declaration.

```
#include <stdio.h>
#define MAX_LIMIT 20
int main()
{
    char str[MAX_LIMIT];
    fgets(str, MAX_LIMIT, stdin);
    printf("%s", str);
    return 0;
}
```

Multiple words input using scanf():

1)

```
#include <stdio.h>
int main()
{
char str[20];
scanf("%[^\\n]*c", str);
printf("%s", str);
return 0;
}
```

Here, [] is the scanset character. ^\n tells to take input until newline doesn't get encountered. Then, with this %*c, it reads newline character and here used * indicates that this newline character is discarded.

2)

```
#include <stdio.h>
int main() {
char str[100];
scanf("%[^\\n]s",str);
printf("%s",str);
return 0;
}
```

```
#include <stdio.h>
int main()
{
    char name[]="Hello"; //string char array
    int i=0;
    while(name[i]!='\0') //untill null character
    {
        printf("%c",name[i]);
        i++;
    } //end while
} //end main
```

Program to print strings character by character using loop.

Hello

Output

String Handling Library

- Functions defined in `#include<string.h>`
- String handling library provides **many** useful functions:
 - Manipulate string data(copy and concatenate)
 - Comparing strings
 - Determine string length

String Manipulation Functions(or Functions in string library)



L
P
U

Function prototype

Function description

char *strcpy(char *s1, const char *s2);

Copies the string s2 into the character array s1. The value of s1 is returned.

char *strncpy(char *s1, const char *s2, size_t n);

Copies at most n characters of the string s2 into the character array s1. The value of s1 is returned.

char *strcat(char *s1, const char *s2);

Appends the string s2 to s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.

char *strncat(char *s1, const char *s2, size_t n);

Appends at most n characters of string s2 to string s1. The first character of s2 overwrites the terminating null character of s1. The value of s1 is returned.

int strcmp(const char *s1, const char *s2);

Compares the string s1 with the string s2. The function returns a value of zero, less than zero or greater than zero if s1 is equal to, less than or greater than s2, respectively.

int strncmp(const char *s1, const char *s2, size_t n);

Compares up to n characters of the string s1 with the string s2. The function returns zero, less than zero or greater than zero if the n-character portion of s1 is equal to, less than or greater than the corresponding n-character portion of s2, respectively.

More functions in string library



- `strlen()`-It is used to find the length of string without counting the null character
- `strrev()`-It is used to display the reverse of a string
- `strlwr()`-Converting a string from upper to lower case
- `strupr()`-Converting a string from lower to upper case

strcpy() and strncpy()

- **strcpy()** copies the entire **second** argument string into **first** argument.

strcpy(s1, s2);

- **strncpy()** copies the **first n** characters of **second** string argument into **first** string argument.

strncpy(s1, s2, 4);

- A null character ('\0') is appended **explicitly** to first argument, because the call to strncpy in the program **does not** write a terminating null character.
- The third argument is less than the string length of the second argument.

Examples



L
P
U

```
//strcpy() function
#include<stdio.h>
#include<string.h>
int main()
{
    //strcpy function
    char ori[20],dup[20];
    char *z;
    printf("\n Enter your name:");
    gets(ori);
    z=strcpy(dup,ori);
    printf("Original String
is:%s",ori);
    printf("\nDuplicate String
is:%s",dup);
    printf("\n Value of z is:%s",z);
    return 0;
}
```

```
//strncpy()
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[15],str2[15];
    int n;
    printf("\nEnter Source String:");
    gets(str1);
    printf("\nEnter Destination String:");
    gets(str2);
    printf("Enter number of characters to
copy in destination string:");
    scanf("%d",&n);
    strncpy(str2,str1,n);
    printf("Source string is:%s",str1);
    printf("\nDestination String
is:%s",str2);
    return 0;
}
```

- Function **strcat** appends its second argument string to its first argument string.

strcat(s1, s2);

- The array used to store the first string should be large enough to store
 - the first string
 - the second string and
 - the terminating null character copied from the second string.

strncat()



- Function strncat appends a specified number of characters from the second string to the first string.

strncat(s1, s2, 6)

- A terminating null character is **automatically** appended to the result.

Examples



```
//strcat
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[20],str2[10];
    printf("\n Enter first string:");
    gets(str1);
    printf("\n Enter second string:");
    gets(str2);
    strcat(str1,str2);
    printf("\n String after
concatenation:%s",str1);
    return 0;
}
```

```
//strncat
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[20],str2[10];
    int n;
    printf("\n Enter first string:");
    gets(str1);
    printf("\n Enter second string:");
    gets(str2);
    printf("\n Enter number of
characters you want to combine:");
    scanf("%d",&n);
    strncat(str1,str2,n);
    printf("\n String after
concatenation:%s",str1);
    return 0;
}
```

Comparison Functions of the String Handling Library

- Comparing strings
 - Computer compares numeric ASCII codes of characters in string
 - **strcmp()** Compares its first string argument with its second string argument, character by character.
 - Function **strncmp()** does not compare characters following a null character in a string.

strcmp()

int strcmp(const char *s1, const char *s2);

- Compares string s1 to s2
- Returns
 - a negative number if $s1 < s2$,
 - zero if $s1 == s2$
 - a positive number if $s1 > s2$

strcmp()

int strcmp(const char *s1, const char *s2, int n);

– Compares up to n characters of string $s1$ to $s2$

- a negative number if $s1 < s2$,
- zero if $s1 == s2$
- a positive number if $s1 > s2$

Examples



L
P
U

```
//strcmp
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[20],str2[10];
    int x;
    printf("\n Enter first string:");
    gets(str1);
    printf("\n Enter second string:");
    gets(str2);
    x=strcmp(str1,str2);
    if(x==0)
    {
        printf("\n Strings are equal");
    }
    else if(x>0)
    {
        printf("\n First string is greater
than second string(strings are not equal)");
    }
    else
    {
        printf("\n First string is less than
second string(strings are not equal)");
    }
    return 0;
}
```

```
// strncmp
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[20],str2[10];
    int x,n;
    printf("\n Enter first string:");
    gets(str1);
    printf("\n Enter second string:");
    gets(str2);
    printf("\n Enter no. of characters to compare:");
    scanf("%d",&n);
    x=strncmp(str1,str2,n);
    if(x==0)
    {
        printf("\n Strings are equal");
    }
    else if(x>0)
    {
        printf("\n First string is greater than
second string(strings are not equal)");
    }
    else
    {
        printf("\n First string is less than
second string(strings are not equal)");
    }
    return 0;
}
```

strcmp()[Ignore case], strcmp will ignore the case



L
P
U

```
#include<stdio.h>
int main()
{
    char str1[20],str2[10];
    int x;
    printf("\n Enter first string:");
    gets(str1);
    printf("\n Enter second string:");
    gets(str2);
    x=stricmp(str1,str2);
    if(x==0)
    {
        printf("\n Strings are equal");
    }
    else if(x>0)
    {
        printf("\n First string is greater than second string(strings are not equal)");
    }
    else
    {
        printf("\n First string is less than second string(strings are not equal)");
    }
    return 0;
}
//consider str1(HELLO) and str2(hello) and if we apply strcmp on these strings, then 0 will be returned, as
strings are equal
```

Determining the length of string

strlen()

- **Function** strlen in #include<string.h>
- Function **strlen()** takes a **string** as an argument and returns the **number** of characters in the string
 - the terminating null character is not included in the length

Example

```
#include<stdio.h>
#include<string.h>
int main()
{
char str[]="Hello";
printf("\n Length of the given string is:%d",strlen(str));
return 0;
}
```

strrev()-Example

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s[100]="Hello";
    printf("%s",strrev(s));
    return 0;
}
```

strlwr(),strupr()-Examples

```
#include<stdio.h>
#include<string.h>
int main()
{
    char s[]="hello";
    strupr(s);
    puts(s);
    strlwr(s);
    puts(s);
    return 0;
}
```

All string operations without inbuilt functions

- Copying one string to another
- Finding length of a string
- Concatenation(or Combining) of two strings
- Comparing two strings
- Displaying reverse of a number
- Checking whether a given string is palindrome or not
- Converting all characters of a given string from lowercase to uppercase
- Converting all characters of a given string from uppercase to lowercase

WAP to copy one string to another without using strcpy() or inbuilt function



```
#include<stdio.h>
int main() {
    char s1[100], s2[100];
    int i;
    printf("\nEnter the string :");
    gets(s1);//Hello
    i = 0;
    while (s1[i] != '\0') {
        s2[i] = s1[i];
        i++;
    }
    s2[i] = '\0';
    printf("\nCopied String is %s ", s2);
    return (0);
}
```

WAP to find the length of a string without using strlen() or inbuilt function



```
#include<stdio.h>
int main()
{
    char x[100];
    int i=0;
    printf("\n Enter String:");
    gets(x);
    while(x[i]!='\0')
    {
        i++;
    }
    printf("\n Length of the string is:%d",i);
    return 0;
}
```

WAP to concatenate(or combine) two strings without using strcat/ or inbuilt function



```
#include<stdio.h>
int main()
{
    char
str1[100],str2[100],str3[200];
    int i=0,j=0;
    printf("\n Enter the first
string:");
    gets(str1);
    printf("\n Enter the second
string:");
    gets(str2);
    while(str1[i]!='\0')
    {
        str3[j]=str1[i];
        i++;
        j++;
    }
    i=0;
    while(str2[i]!='\0')
    {
        str3[j]=str2[i];
        i++;
        j++;
    }
    str3[j]='\0';
    printf("\n The concatenated
string is:");
    puts(str3);
    return 0;
}
```



L
P
U

WAP to compare two strings without using strcmp() or inbuilt function

```
#include <stdio.h>
#include<string.h>
int main ()
{
    // declare variables
    char str1 [30], str2 [30];
    int i = 0, flag=0 ,length1, length2, length;
    // take two string input
    printf ("Enter string1:");
    gets (str1);
    printf ("\nEnter string2:");
    gets (str2);
    //length of both string
    length1 = strlen (str1);
    length2 = strlen (str2);
    if(length1>length2)
        length=length1;
    else
        length=length2;
```

```
while (i<length)
{
    if( str1 [i] == str2 [i])
    {
        i++;
        continue;
    }
    if( str1 [i] < str2 [i])
    {
        flag = -1;
        break;
    }
    if( str1 [i] > str2 [i])
    {
        flag = 1;
        break;
    }
}
if (flag == 0)
    printf ("\nBoth strings are equal ");
if(flag == -1)
    printf ("\nstring1 is less than string2 ");
if( flag == 1)
    printf ("\nstring1 is greater than string2 ");
return 0;
}
```

Dry running

Comparing two strings

```

str1 → first string | flag=0;
str2 → second string i=0
length1 = strlen(str1);
length2 = strlen(str2);
if (length1 > length2)
    length = length1;
else
    length = length2;
while (i < length)
    if (str1[i] == str2[i])
        i++;
    continue;
    if (str1[i] < str2[i])
        flag = -1;
        break;
    if (str1[i] > str2[i])
        flag = 1;
        break;
    if (flag == 0)
        printf("equal");
    if (flag == -1)
        printf("%s1 is less than %s2");
    if (flag == 1)
        printf("%s1 is > than %s2");

```

consider

str1 → Have
 str2 → Has
 length1 = 4
 length2 = 3
 $4 > 3$

length = 4.

i=0, $0 < 4$ (True)
 H == H (True)
 i=1
 continue (Next Iteration)
 $1 < 4$ (True)
 a == a (True)
 i=2
 continue (Next Iteration)
 $2 < 4$ (True)
 v == s (False)
 v < s (False)
 v > s (True)
 flag = 1;
 break; [Loop terminates]
 i == 1 (flag == 1) True
 S1 is > (greater than) S2

WAP to display the reverse of a given string without strrev() or inbuilt function

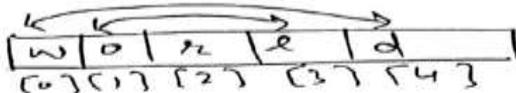
```
#include<stdio.h>
#include<string.h>
int main() {
    char str[100], temp;
    int i, j;
    printf("\nEnter the string :");
    gets(str);
    i = 0;
    j = strlen(str) - 1;
    while (i < j) {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
    printf("\nReverse string is :%s", str);
    return (0);
}
```

Dry running

Reverse of a given string
Without using string class

```
i = 0; [ str → string ]
j = strlen(str) - 1;
while (i < j)
{
    temp = str[i];
    str[i] = str[j];
    str[j] = temp;
    i++;
    j--;
}
printf("The reverse string is: %s", str);
```

str = "world"



i = 0;
j = 5 - 1 = 4

0 < 4 (True)

temp = w
w = d
d = w } swapping

i = 1
j = 3

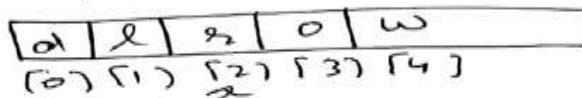
1 < 3 (True)

temp = o
o = l
l = o } swapping

i = 2
j = 2

2 < 2 (False)
Loop stops.

So, str:



Reverse

WAP to check whether the given string is palindrome or not(without using strrev())



```
#include<stdio.h>
#include<string.h>
int main() {
    char str[100], temp;
    char str1[100];
    int i, j;
    printf("\nEnter the string :");
    gets(str);
    i = 0;
    j = strlen(str) - 1;
    strcpy(str1,str);
    while (i < j)
    {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
    if(strcmp(str1,str)==0)
    {
        printf("\n Given String is Palindrome");
    }
    else
    {
        printf("\n Not a Palindrome");
    }
    return (0);
}
```

WAP to convert all characters of a given string into uppercase without using strupr() or inbuilt function

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[10];
    int i,len;
    printf("Enter any string \t");
    gets(str1);
    len=strlen(str1);
    for(i=0;i<len;i++)
    {
        if(str1[i]>='a' && str1[i]<='z')
            str1[i]=str1[i]-32;
    }
    puts("string in upper is");
    puts(str1);
    return 0;
}
```

WAP to convert all characters of a given string into lowercase without using strlwr() or inbuilt function

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[10];
    int i,len;
    printf("Enter any string \t");
    gets(str1);
    len=strlen(str1);
    for(i=0;i<len;i++)
    {
        if(str1[i]>='A' && str1[i]<='Z')
            str1[i]=str1[i]+32;
    }
    puts("string in lower is");
    puts(str1);
    return 0;
}
```

More programs on strings

WAP to sort the characters of a given string into ascending order



```
#include<stdio.h>
#include<string.h>
int main()
{
char s[10],t;
int n,i,j;
printf("\n Enter String:");
gets(s);
n=strlen(s);
for(i=0;i<n-1;i++)
{
    for(j=0;j<n-i-1;j++)
    {
        if(s[j]>s[j+1])
        {
            t=s[j];
            s[j]=s[j+1];
            s[j+1]=t;
        }
    }
}
printf("%s",s);
}
```

WAP to count vowels in a given string



```
#include<stdio.h>
int main()
{
    char x[100];
    int i=0,count=0;
    printf("\n Enter the string:");
    gets(x);
    while(x[i]!='\0')
    {
        if(x[i]=='a' | |x[i]=='e' | |x[i]=='i' | |x[i]=='o' | |x[i]=='u' | |x[i]=='A' | |x[i]=='E' | |x[i]=='I' | |x[i]=='O' | |x[i]=='U')
        {
            count++;
        }
        i++;
    }
    printf("\n Number of vowels in the string are:%d",count);
    return 0;
}
```

```
#include<stdio.h>
int main()
{
    char *g="C Programming";
    int length=0,i=0;
    while(*g!='\0')
    {
        printf("%c",*g);//Value at address
        g++;//Pointer is incremented by 1 after each iteration
        length++;//Variable for counting length
    }
    printf("\nLength of the string is:%d",length);
    return 0;
}
```

WAP to count total no. of characters and words in a given string



```
#include<stdio.h>
int main()
{
    char x[100];
    int i=0,length=0,c=0,w=1;
    printf("\n Enter String:");
    gets(x);
    while(x[i]!='\0')
    {
        if(x[i]==' ' && x[i+1]!=' ')
        {
            w++;
        }
        c++;
        i++;
    }
    printf("\n Total number of characters are:%d, and no. of words are:%d",c,w);
    return 0;
}
```

WAP to demonstrate array of strings in C



```
#include<stdio.h>
int main()
{
    char names[5][10];
    int i,n;
    printf("\n Enter the number of students:");
    scanf("%d",&n);
    fflush(stdin);
    for(i=0;i<n;i++)
    {
        printf("\n Enter the name of student %d: ",i+1);
        gets(names[i]);
    }
    printf("\n Names of the students are:\n");
    for(i=0;i<n;i++)
        puts(names[i]);
    return 0;
}
```

WAP to traverse a string character by character



```
#include <stdio.h>
int main()
{
    char name[]="Hello World"; //string char array
    int i=0;
    while(name[i]!='\0') //untill null character
    {
        printf("%c\n", name[i]);
        i++;
    } //end while
} //
```

WAP to replace all spaces in a given string with '\$'[Example for character replacement]



```
#include<stdio.h>
int main()
{
    char x[100];
    int i=0;
    printf("\n Enter the string:");
    gets(x);
    while(x[i]!='\0')
    {
        if(x[i]==' ')
        {
            x[i]='$';//Character replacement
        }
        i++;
    }
    printf("\n String after character replacement is:%s",x);
    return 0;
}
```

Output-1??



```
#include<stdio.h>
int main()
{
char str[]="Practice MCQ";
printf("\n%d",sizeof(str));
return 0;
}
```

- A. 13
- B. 12
- C. 11
- D. 1

Output-2??

```
#include<stdio.h>
int main()
{
    char str[]="Program";
    printf("%c",str[7]);
    return 0;
}
```

- A. m
- B. Program
- C. Compile time error
- D. Nothing will be visible

Output-3??

```
#include<stdio.h>
int main()
{
char str1[]="Good";
char str2[5];
str2=str1;
printf("%s",str2);
return 0;
}
```

A. Good
B. Garbage value
C. Compile time error
D. Nothing will be visible

Output-4??

```
#include<stdio.h>
int main()
{
char str1[]="Good";
char *str2;
str2=str1;
puts(str2);
return 0;
}
```

A. Good
B. Garbage value
C. Compile time error
D. Nothing will be visible

```
#include<stdio.h>
#include<string.h>
int main()
{
char str[20]="Example";
printf("%d %d",sizeof(str),strlen(str));
return 0;
}
```

- A. 7 7
- B. 20 20
- C. 20 7
- D. 20 8

Output-6??

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[20]="Example";
    char str2[30]="Exam";
    if(strncmp(str1,str2,4))
        printf("\nHello");
    else
        printf("\nWorld");
    return 0;
}
```

- A. Hello
- B. World
- C. Nothing will be printed
- D. Compile time error

Output-7??

What will be the output of the program ?

```
#include<stdio.h>
#include<string.h>
int main()
{
    char str1[20] = "Hello", str2[20] = " World";
    printf("%s\n", strcpy(str2, strcat(str1, str2)));
    return 0;
}
```

- A. Hello
- B. World
- C. Hello World
- D. WorldHello

Output-8??

In below program, what would you put in place of "?" to print "Quiz"?

```
#include <stdio.h>
int main()
{
    char arr[] = "HelloQuiz";
    printf("%s", ?);
    return 0;
}
```

- A. arr
- B. (arr+5)
- C. (arr+4)
- D. Not possible

Question-9

Which of the following C code snippet is not valid?

- (A) `char* p = "string1"; printf("%c", *++p);`
- (B) `char q[] = "string1"; printf("%c", *++q);`
- (C) `char* r = "string1"; printf("%c", r[1]);`
- (D) None of the above

Output-10 ??

```
#include<stdio.h>

int main()
{
    printf("C Programming\n");
}
```

- A. mming
- B. ming
- C. amming
- D. gramming

Output-11 ??

```
//Assume unsigned integer takes 4 bytes
#include <stdio.h>
int main()
{
    char *str1 = "Hello";
    char str2[] = "Hello";
    printf("sizeof(str1) = %d, sizeof(str2) = %d", sizeof(str1), sizeof(str2));
    return 0;
}
```

- A. sizeof(str1) = 6, sizeof(str2) = 6
- B. sizeof(str1) = 4, sizeof(str2) = 6
- C. sizeof(str1) = 4, sizeof(str2) = 4
- D. sizeof(str1) = 6, sizeof(str2) = 4

Output-12 ??

Predict the output of the following program:

```
#include <stdio.h>

int main()
{
    char str[] = "%d %c", arr[] = "HelloWorld";
    printf(str, 0[arr], 2[arr + 3]);
    return 0;
}
```

- A. 72 W
- B. H W
- C. W H
- D. Compile-time error

Output-13??

```
#include <stdio.h>

int main()
{
    char *str="WORLD";
    while(*++str)
    {
        printf("%c",*str);
    }
    return 0;
}
```

A. ORLD
B. WORLD
C. RLD
D. Compile time error

CSE101-Lec#24

- Some other functions from string handling library

Outline

- String Conversion Functions
- Character arithmetic

String Conversion Functions

- String Conversion functions
 - In `<stdlib.h>` (general utilities library)
- Convert strings of digits to **integer** and **floating-point values and vice versa**

Function prototype	Function description
<code>double atof(const char *nPtr);</code>	Converts the string nPtr to double.
<code>int atoi(const char *nPtr);</code>	Converts the string nPtr to int.
<code>long atol(const char *nPtr);</code>	Converts the string nPtr to long int.
<code>char * itoa(int value, char *str,int base);</code>	Converts the integer value to string

atof()

Function atof

- Function **atof converts its argument—a string that represents a floating-point number—to a double value.**
- The function returns the double value.
- If the converted value cannot be represented—for example, if the first character of the string is a letter—the behavior of function **atof** is undefined.

atof()-Program example

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    double d;
    d=atof("99.23");
    printf("%.2lf",d);
    return 0;
}
```

atoi()

Function atoi

- Function **atoi converts its argument—a string of digits that represents an integer— to an int value.**
- The function returns the int value.
- If the converted value cannot be represented, the behavior of function atoi is undefined.

atoi()-Program example

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    char x[]="99";
    int i;
    i=atoi(x);
    printf("%d",i);
    return 0;
}
```

atol()

Function atol

- Function **atol** converts its argument—a string of digits representing a long integer—to a long value.
- The function returns the long value.
- If the converted value cannot be represented, the behavior of function **atol** is undefined.
- If **int** and **long** are both stored in 4 bytes, function **atoi** and function **atol** work identically.

atol()-Program example

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    long int i;
    char x[]={ "10000000"};
    i=atol(x);
    printf("%ld",i);
    return 0;
}
```

itoa()

- `itoa ()` function in C language converts int data type to string data type. Syntax for this function is given below.
- `char * itoa (int value, char * str, int base);`
- Base values could be: 2, 8, 10,16(Depending upon the number system)

itoa()-Program example

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
    int a=123;
    char str[100];
    itoa(a,str,2);
    printf("\n Binary value:%s",str);
    itoa(a,str,10);
    printf("\n Decimal value:%s",str);
    itoa(a,str,16);
    printf("\n Hexadecimal value:%s",str);
    itoa(a,str,8);
    printf("\n Octal value:%s",str);
    return 0;
}
```

Converting from float to string



- There is no inbuilt function for this type of conversion, so we need to write the explicit code for the same,
- In the following ftoa1() is a user defined function which is converting floating type value to string
- Example:

```
#include<stdio.h>
void ftoa1(float f1,char s[])
{
sprintf(s,"%f",f1);// s is array, %f format specifier and f1 is float variable
}
int main()
{
char str[20];
float f=12.340000;
ftoa1(f,str);//it means convert float value f to string and store it in str// function
call//
printf("\n%s",str);
return 0;
}
```

Character arithmetic

- To perform increment , decrement, addition subtraction operations on the characters.
- These operations work on the **ASCII** value of characters.
- Starting from ASCII value of 'a' = 97 to the ASCII value of 'z' = 122

Increment

- To display next char value

```
int main()
{
char x = 'a' + 1;
printf("%c", x); // Display Result = 'b'
printf("%c", ++x); // Display Result = 'c'

}
```

Decrement

- To display previous char value

```
int main()
{
char x = 'b' - 1;
printf("%c", x); // Display Result = 'a'
}
```

Addition

- Adding two ASCII values

```
int main()
{
    char x = 'a' + 'c';
    printf("%c", x); /* Display Result = - ( addition of
    ASCII of a and c is 196) */
}
```

Subtraction

- Subtracting two ASCII values

```
int main()
{
    char x = 'z' - 'a';
    printf("%c",x); /* Display Result = ↓ (difference
    between ASCII of z and a ) */
}
```

CSE101-Lec#25

- User defined data types(Structures)

Outline

- Declaration of a structure
- Definition and initialization of structures.
- Accessing structures.

Introduction

- **Structures**
 - Structure is a group of data items of different data types held together in a single unit.
 - Collections of related variables under **one name**
 - Can contain variables of different data types
 - Commonly used to define records to be stored in files.

Why Use Structures?

- Quite often we deal with entities that are collection of dissimilar data types.
- For example, suppose you want to store data about a car. You might want to store its name (a string), its price (a float) and number of seats in it (an int).
- If data about say 3 such cars is to be stored, then we can follow two approaches:
 - Construct individual arrays, one for storing names, another for storing prices and still another for storing number of seats.
 - Use a structure variable.



Structure

- There are three aspects of working with structures:
 - Defining a structure type
 - Declaring variables and constants of newly created type
 - Using and performing operations on the objects of structure type

Structure Definition

Syntax

```
struct sname {  
    type var1;  
    type var2;  
    type var3;  
    .  
    .  
    type varN;  
};
```

struct is a keyword to define a structure.

sname is the name given to the structure/structure tag.

type is a built-in data type.

var1, var2, var3,....., varN are elements of structure being defined.

; semicolon at the end.

Structure Definitions

- Example:

```
struct car{  
    char *name;  
    int seats;  
    float price  
};
```

- **struct keyword** introduces the definition for structure **car**
 - **car** is the structure name or tag and is used to declare variables of the structure type
 - **car** contains three members of type **char**, **float**, **int**
 - These members are **name**, **price** and **seats**.
- No variable has been associated with this structure
- No memory is set aside for this structure.

Structure Definitions

- **struct information**
 - A structure definition does not reserve space in memory .
 - Instead creates a new data type used to define structure variables
- **Defining variables of structure type**
 - Defined like other variables:

```
Car myCar, cars[5], *cPtr;
```
 - Can use a comma separated list along with structure definition:

```
struct car{  
    char *name;  
    int seats;  
    float price;  
} myCar, cars[5], *cPtr;
```

At this point, the **memory is set aside** for the structure variable **myCar**.

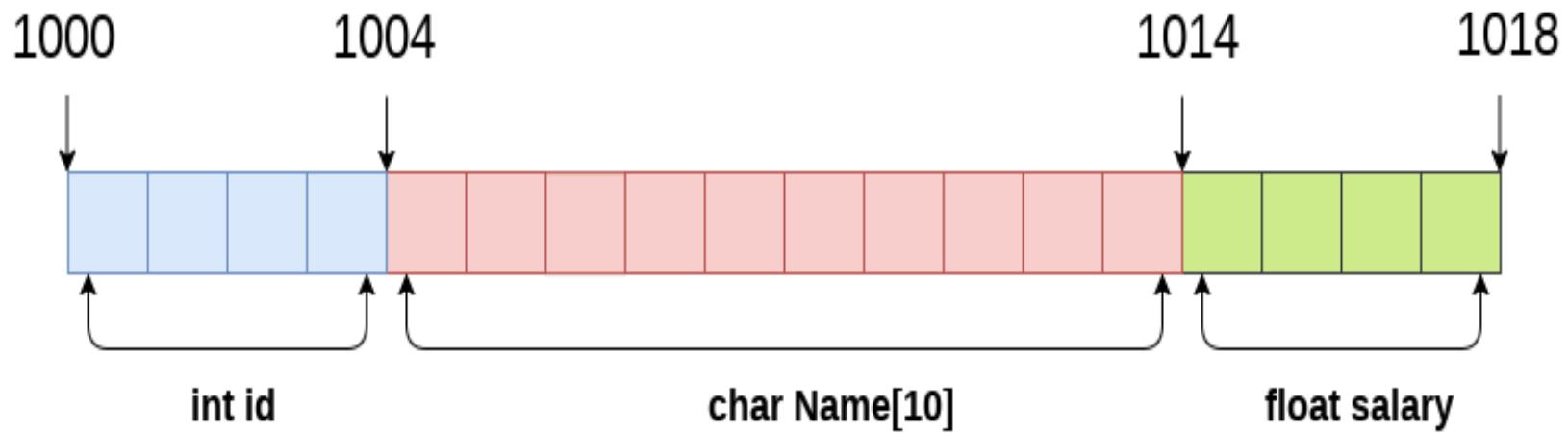
How the members are stored in memory

Consider the declarations to understand how the members of the **structure variables are stored in memory**

```
struct employee{  
    int id;  
    char name[20];  
    float salary;  
}e1;
```

Note: all members are stored in contiguous memory location in order in which they are declared.

Memory allocation



Structure Definitions

- Operations that can be performed on structures
 - Assigning a structure to a structure of the same type
 - Taking the address (&) of a structure
 - Accessing the members of a structure
 - Using the `sizeof` operator to determine the size of a structure

Initializing Structures

- **Initializer list**
 - To initialize a structure similarly like arrays
 - Example:

```
struct car myCar = { "Renault", 500000, 2 };
```
 - Could also define and initialize myCar as follows:

```
struct Car myCar;  
strcpy(name,"Renault");//if name is array of characters  
myCar.price = 500000;  
myCar.seats = 2;
```

Accessing Members of Structures

- Two operators are used to access members of Structures:

- Dot operator (.) used with structure variables

```
struct car myCar;  
printf("%d", myCar.seats);
```

- Arrow operator (->) used with pointers to structure variables

```
struct car *myCarPtr = &myCar;  
printf("%d", myCarPtr->seats);
```

- myCarPtr->name is equivalent to
(*myCarPtr).seats

dot Operator

- Members are accessed using dot operator.
- It provides a powerful and clear way to refer to an individual element.
- **Syntax:** **sname.vname**
- **sname** is structure variable name.
- **vname** is name of the element of the structure.
- **Eg:** the members of the structure variable **car** can be accessed as

`myCar.name, myCar.seats, myCar.price`

Use of Assignment Statement for Structures

- The main advantage of structure is that it can be treated as single entity.
- The only legal operations that can be performed on structure are copying to it as a single unit using the assignment operator.
- Value of one structure variable can be assigned to another variable of the **same type** using simple assignment statement.
- If `myCar` and `newCar` are structure variable of type `car`, then

```
newCar = myCar;
```

Use of Assignment Statement for Structures

- When we assigns value of structure variable *myCar* to *newCar*, all values of members of one structure get copied into corresponding members of another structure.
- Or we can copy one member at a time:
 - `newCar.name = myCar.name;`
- *Simple assignment cannot be used this way for arrays.*
- This is really a big advantage over arrays where in order to copy one array into another of same type, we have copied the contents element by element either using loop or individually.

Program to
show how to
access
structure.

```
#include <stdio.h>

struct car{
    char *name;
    int seats;
    float price;
}; //end structure car

int main()
{
    struct car myCar; //define struct variable
    myCar.name = "Renault";
    myCar.price = 500000;
    myCar.seats = 2;
    printf("%s %f %d \n", myCar.name,
    myCar.price, myCar.seats);
} //end main
```

Renault 500000 2

Program to enter data into structure.

```
#include <stdio.h>
struct car{
    char name[50];
    int seats;
    float price;
};

main()
{
    struct car myCar;
    printf("Enter name of car:\n");
    gets(myCar.name);
    printf("Enter number of seats in car:\n");
    scanf("%d", &myCar.seats);
    printf("Enter price of car:\n");
    scanf("%f", &myCar.price);
    printf("\n\nParticulars of car are:\n");
    printf("Car name:%s",myCar.name);
    printf("\nNumber of seats:%d",
myCar.seats);
    printf("\nPrice:%f", myCar.price);
} //end main
```



L
P
U

Enter name of car: Micra

Enter number of seats in car: 4

Enter price of car: 600000

Particulars of car are:

Car name: Micra

Number of seats: 4

Price: 600000

Array & Structure

Array	Structure
1. It is a collection of data items of same data type.	1. It is a collection of data items of different data types.
2. It has declaration only.	2. It has declaration & definition.
3. There is no keyword.	3. <code>struct</code> is the keyword.
4. An array name represents the address of the starting element.	4. A structure name is called tag. It is a short hand notation of the declaration.

Q1

What will be the output of following code?

```
#include<stdio.h>
struct employee
{
    char name[20];
    int roll_no;
    float salary;
};
int main()
{
    struct employee e1={"Smith"};
    printf("%f",e1.salary);
    return 0;
}
```

- A. Compile time error
- B. 0.000000
- C. 1
- D. Garbage value

Q2

What will be the output of following code?

```
#include<stdio.h>
struct book
{
    char b_name[50];
    int no_of_pages;
    float price;
};
int main()
{
    struct book b1={"Programming"};
    printf("%d",b1.no_of_pages);
    return 0;
}
```

A. 1
B. 0
C. Compile time error
D. Garbage value

Q3

What will be the output of following code?

```
#include<stdio.h>
struct student
{
    char name[20];
    int roll_no;
    float fees
}v1={"Goerge",23,4500.45},v2={"Tom",45,3400.56};
int main()
{
printf("%d",v2.roll_no);
return 0;
}
```

A. 45
B. 23
C. Garbage value
D. Compile time error

Q4

What will be the output of following code?

```
#include<stdio.h>
struct student
{
    char name[20];
    int roll_no;
    float fees
};v1={"Goerge",4500.45};
```

```
int main()
{
printf("%d",v1.roll_no);
return 0;
}
```

- A. Garbage value
- B. 4500
- C. 4500.45
- D. 0

Q5

What will be the output of following code?

```
#include<stdio.h>
struct student
{
    int a=10;
}v1;
int main()
{
printf("%d",v1.a);
return 0;
}
```

A. 10
B. 0
C. Compile time error
D. Garbage value

CSE101-Lec#26

- Reading and displaying one record using structure
- Copying and Comparing structures
- Array of structures
- Pointer to structures

WAP to read and display information of one student[or one record] using structure



```
#include <stdio.h>
struct student
{
    char name[50];
    int roll;
    float marks;
};
int main()
{
    struct student s;
    printf("Enter information for first
student:\n");
    printf("Enter name: ");
    gets(s.name);
    printf("Enter roll number: ");
    scanf("%d", &s.roll);
    printf("Enter marks: ");
    scanf("%f", &s.marks);
    printf("Displaying Information:\n");
    printf("Name: ");
    puts(s.name);
    printf("Roll number: %d\n", s.roll);
    printf("Marks: %.1f\n", s.marks);
    return 0;
}
```

Copying data from one structure variable to another



```
#include <stdio.h>
struct student
{
    char name[50];
    int roll;
    float marks;
};
int main()
{
    struct student s,s1;
    printf("Enter information:\n");
    printf("Enter name: ");
    scanf("%s", s.name);
    printf("Enter roll number: ");
    scanf("%d", &s.roll);
    printf("Enter marks: ");
    scanf("%f", &s.marks);
    s1=s;//Copying structure variable to another
    printf("Displaying Information:\n");
    printf("Name: ");
    puts(s1.name);
    printf("Roll number: %d\n",s1.roll);
    printf("Marks: %.1f\n", s1.marks);
    return 0;
}
```

Comparing two structure variables



```
#include <stdio.h>
struct student
{
    char name[50];
    int roll;
    float marks;
};
int main()
{
    struct student s,s1;
    printf("Enter information of first
student:\n");
    printf("Enter name: ");
    scanf("%s", s.name);
    printf("Enter roll number: ");
    scanf("%d", &s.roll);
;
    printf("Enter marks: ");
    scanf("%f", &s.marks);
    printf("Enter information of second student:\n");
    printf("Enter name: ");
    scanf("%s", s1.name);
    printf("Enter roll number: ");
    scanf("%d", &s1.roll);
    printf("Enter marks: ");
    scanf("%f", &s1.marks);
    if(s.marks>s1.marks)
    {
        printf("\n Marks of first student are more..");
    }
    else
    {
        printf("\n Marks of second student are more..");
    }
    return 0
}
```

Array of Structures

- to store data of 100 cars we would be required to use 100 different structure variables from **car1** to **car100**, which is definitely not very convenient. A better approach would be to use an array of structures.

```
struct car mycar[100];
```

- This provides space in memory for 100 structures of the type **struct car**.

Program to print array of structures.

```
#include <stdio.h>
struct car{
    char name[50];
    int seats;
    float price;
};
int main()
{
    int i;
    struct car myCar[100];

for(i=0; i<100; i++){
    printf("\n\nEnter data for car[%d]:\n", i);
    scanf("%s %d %f", &myCar[i].name,
&myCar[i].seats, &myCar[i].price);
}

for(i=0; i<100; i++){
    printf("\nData about your car[%d] is: %s %d
%f", i, myCar[i].name, myCar[i].seats,
myCar[i].price);
}
}
```

Enter data for car0: Racer 1 1200000

Data about your car0 is Racer 1 1200000

Enter data for car1: Micra 4 500000

Data about your car1 is Micra 4 500000

Enter data for car2: RacerGt 1 800000

Data about your car2 is RacerGt 1 800000

.

.

.

.

.

Enter data for car99: RacerEf 1 2000000

Data about your car99 is RacerEf 1 2000000

Array of structures-Example 2

WAP to read and display information of n number of books in a library



```
#include <stdio.h>
struct Bookinfo
{
    char bname[20];
    int pages;
    float price;
};
int main()
{
    struct Bookinfo book[100];
    int i,n;
    printf("\nEnter number of records you want to
enter(less than or equal to:100)");
    scanf("%d",&n);
    fflush(stdin);
    for(i=0;i<n;i++)
    {
        printf("\nEnter the Name of Book :");
        gets(book[i].bname);
        fflush(stdin); //It is used to clear the input
buffer(stdin-Input taken from keyboard)
        printf("\nEnter the Number of Pages :");
        scanf("%d",&book[i].pages);
        fflush(stdin);
        printf("\nEnter the Price of Book :");
        scanf("%f",&book[i].price);
        fflush(stdin);
    }
    printf("\n----- Book Details -----");
    for(i=0;i<n;i++)
    {
        printf("\nName of Book : %s",book[i].bname);
        printf("\nNumber of Pages : %d",book[i].pages);
        printf("\nPrice of Book : %.2f",book[i].price);
    }
    return 0;
}
```

Pointers to Structure

```
struct car myCar, *ptr;
```

It declares a structures variable *myCar* and a pointer variable *ptr* to structure of type *car*.

ptr can be initialized with the following assignment statement

```
ptr = &myCar;
```

HOW WE CAN ACCESS THE ELEMENTS OF STRUCTURE?

```
*ptr.name, *ptr.seats, *ptr.age
```

But this approach ***will not work*** because dot has higher priority

Correctly way to write is:

```
(*ptr).name, (*ptr).seats, (*ptr).price)
```

or

```
ptr->name, ptr->seats, ptr->price
```

Accessing Members of Structures

- Arrow operator (`->`) used with pointers to structure variables

```
car *myCarPtr = &myCar; //initializing pointer  
printf("%s", myCarPtr->name);
```

- `myCarPtr->name` is equivalent to
`(*myCarPtr).name`

Program for pointer to a structure

```
#include <stdio.h>
struct car{
    char *name;
    int seats;
    float price;
};
int main()
{
    struct car myCar = {"Renault", 2, 500000};
    struct car *myCarPtr; //define a pointer to car
    myCarPtr = &myCar; /*assign address of myCar
    to myCarPtr */
    printf("%s %f %d \n%s %f %d\n",
    myCar.name, myCar.price, myCar.seats,
    myCarPtr->name, myCarPtr->price,
    myCarPtr->seats,
    (*myCarPtr).name, (*myCarPtr).price,
    (*myCarPtr).seats);
} //end main
```

```
Renault 500000 2
Renault 500000 2
Renault 500000 2
```

CSE101-Lec#27

- Nested Structure
- Union

Outline

- Nested Structure
- Union

Nested Structure

- Nested structures are structures as member of another structure.
- We can also take objects of one structure as member in another structure.
- Thus, a structure within a structure can be used to create complex data application.
- Dot operator is used twice because we are accessing first structure through the object of second structure.

Nested Structure

Two ways of declaring structure within structure or Nested structure:

- Declare two separate structures
- Embedded structures

Example-1-Declare two separate structures(standalone structures)



```
struct Date
{
    int dd;
    int mm;
    int yy;
};

struct Student
{
    char name[20];
    int rollno;
    int marks;
    struct Date dob;
};
```

Here structure Student is nesting structure and structure date is nested structure

Program example-1 Nested structure-Declare two separate structures(standalone structures)



```
#include<stdio.h>
    struct Address
{
    char Housename[25];
    char City[25];
    char Streetname[25];
};

struct Employee
{
    int Id;
    char Name[25];
    float Salary;

    struct Address Add;
};

int main()
{
    struct Employee E;
    printf("\n\tEnter Employee Id : ");
    scanf("%d",&E.Id);
    printf("\n\tEnter Employee Name : ");
    scanf("%s",E.Name);
    printf("\n\tEnter Employee Salary : ");
    scanf("%f",&E.Salary);
    printf("\n\tEnter Employee House Name : ");
    scanf("%s",E.Add.Housename);
    printf("\n\tEnter Employee City : ");
}
```

```
scanf("%s",E.Add.City);
    printf("\n\tEnter Employee street
name : ");
    scanf("%s",E.Add.Streetname);
    printf("\nDetails of Employees");
    printf("\n\tEmployee Id :
%d",E.Id);
    printf("\n\tEmployee Name :
%s",E.Name);
    printf("\n\tEmployee Salary :
%f",E.Salary);
    printf("\n\tEmployee House No :
%s",E.Add.Housename);
    printf("\n\tEmployee City :
%s",E.Add.City);
    printf("\n\tEmployee street
name: %s",E.Add.Streetname);

}
```

Example-2: embedded structures(Nested structure)



```
struct Student
{
    char name[20];
    int rollno;
    struct date
    {
        int dd;
        int mm;
        int yy;
    } dob;
};
```

Program example-2 Nested structure-Embedded structure



```
#include <stdio.h>
struct Employee
{
    char ename[20];
    int ssn;
    float salary;
    struct dob
    {
        int date;
        int month;
        int year;
    }db1;
}emp = {"Aniket",1000,1000.50,{22,6,1990}};

int main()
{
printf("\nEmployee Name : %s",emp.ename);
printf("\nEmployee SSN : %d",emp.ssn);
printf("\nEmployee Salary : %.2f",emp.salary);
printf("\nEmployee DOB :
%d/%d/%d",emp.db1.date,emp.db1.month,emp.db1.year);

return 0;
}
```

Unions

- **union**
 - Memory that contains a variety of objects over time
 - Only contains one data member at a time
 - Members of a union share space
 - Conserves storage
 - Only the last data member defined can be accessed
- **union definitions**
 - Same as struct

```
union Number {  
    int x;  
    float y;  
};  
union Number value;
```

Union

- Union is similar as structure. The major distinction between them is in terms of storage.
- In structure each member has its own storage location whereas all the members of union uses the same location.
- The union may contain many members of different data type but it can handle only one member at a time union can be declared using the keyword union.

Example

- A class is a very good example of structure and union in this example students are sitting in contiguous memory allocation as they are treated as a structure individually. And if we are taking the place of teacher then in a class only one teacher can teach. After leaving the first teacher then another teacher can enter.



Union Declaration

```
union item  
{  
    int m;  
    float x;  
    char c;  
} code;
```

This declare a variable code of type union item

Initializing and accessing union members



```
#include <stdio.h>
#include <string.h>
union Data {
    int i;
    float f;
    char str[20];
};
int main( )
{
    union Data data;
    data.i = 10;
    data.f = 220.5;
    strcpy( data.str, "C Programming");
    printf( "data.i : %d\n", data.i);
    printf( "data.f : %f\n", data.f);
    printf( "data.str : %s\n", data.str);
    return 0;
}
```

Program using union.

```
#include <stdio.h>
union job{
    char name[32];
    float salary;
    int worker_no;
}u;
main()
{
printf("Enter name:\n");
scanf("%s",&u.name);
printf("Enter salary: \n");
scanf("%f",&u.salary);
printf("Displaying\nName :%s\n",u.name);
printf("Salary: %.1f",u.salary);
}
```



L
P
U

```
#include<stdio.h>
union employee
{
    char name[30];
    int id;
    float salary;
}u;
int main()
{
    //union employee u;
    printf("\n Enter name:");
    gets(u.name);//Initialization
    printf("\n Entered name is:%s",u.name);//Accessing
    printf("\n Enter id:");
    scanf("%d",&u.id);//Initialization
    printf("\n Entered id is:%d",u.id);//Accessing
    printf("\n Enter salary:");
    scanf("%f",&u.salary);//Initialization
    printf("\n Entered salary is:%.2f",u.salary);//Accessing
    return 0;
}
```

WAP to read and display n number of records using Array of Unions



```
#include<stdio.h>
union employee
{
    char name[30];
    int id;
    float salary;
}u[100];
int main()
{
    //union employee u[100];
    int n,i;
    printf("\n Enter value of n:");
    scanf("%d",&n);
    fflush(stdin);
    for(i=0;i<n;i++)
    {
        printf("\n Enter name:");
        fflush(stdin);
        gets(u[i].name);
        printf("\n Entered name
is:%s",u[i].name);
        printf("\n Enter id:");
        fflush(stdin);
        scanf("%d",&u[i].id);
        printf("\n Entered id
is:%d",u[i].id);
        printf("\n Enter salary:");
        fflush(stdin);
        scanf("%f",&u[i].salary);
        printf("\n Entered salary
is:.2f",u[i].salary);
    }
    return 0;
}
```

Difference between structure and union



L
P
U

	STRUCTURE	UNION
Keyword	The keyword <code>struct</code> is used to define a structure.	The keyword <code>union</code> is used to define a union.
Size	When a variable is associated with a structure, the compiler allocates the memory for each member. The size of structure is greater than or equal to the sum of sizes of its members.	when a variable is associated with a union, the compiler allocates the memory by considering the size of the largest memory. So, size of union is equal to the size of largest member.
Memory	Each member within a structure is assigned unique storage area of location.	Memory allocated is shared by individual members of union.
Value Altering	Altering the value of a member will not affect other members of the structure.	Altering the value of any of the member will alter other member values.
Accessing members	Individual member can be accessed at a time.	Only one member can be accessed at a time.
Initialization of Members	Several members of a structure can initialize at once.	Only one member can be initialized at one time, hence all members cannot be initialized. Only the first member of a union can be initialized.

Similarities between structure and union



- Both are user-defined data types used to store data of different types as a single unit.
- Their members can be objects of any type, including other structures and unions or arrays. A member can also consist of a bit field.
- Both structures and unions support only assignment = and sizeof operators. The two structures or unions in the assignment must have the same members and member types.
- A structure or a union can be passed by value to functions and returned by value by functions. The argument must have the same type as the function parameter. A structure or union is passed by value just like a scalar variable as a corresponding parameter.
- '.' operator is used for accessing members.