ACE
Engineering College

Ankushapur, Ghatkesar, Telangana 501301 (EAMCET Code: ACEG)
NAAC Accridated with 'A' Grade

Department of Computer Science and Engineering

Programming for Problem Solving
I Year B.Tech. (I - Sem)

Prepared by :

Ms. Shubhangi Mahule

Assistant Professor (M.Tech)

in association with ACE Engineering Academy

Unit 2 Programming for Problem Solving

Content:

1 Array

2 Strings,

3 Structures and union

4 Enumeration data type

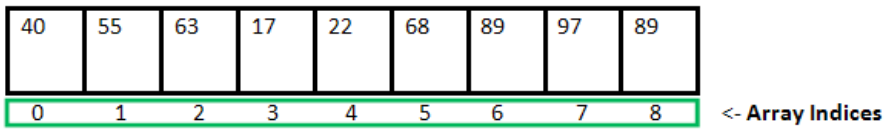
5 Pointers:

For Micro Notes by the Student

Array:

- An array is collection of items or elements stored at continuous memory locations. (or) An array is a group of related data items that share a common name.
- The idea is to declare multiple items of same type together, accessed using a common name

Memory Adddress



Array Length = 9
First Index = 0
Last Index = 8

Elements of an array are accessed by specifying the index (offset) of the desired element within square [ ] brackets after the array name.

The types of Arrays

1 One dimensional array

2 Two dimensional array

3 Multi dimensional array

1 One dimensional Array declaration:

Syntax:

Storageclass dataType arrayname[size];

In C, we can declare an array by specifying its type and size or by initializing it or by both.

Storage class is optional.

For Micro Notes by the Student

Declaration and Initialization :

Storageclass dataType arrayname[size] = {List of Value};

1. int arr[10]; // declaration for one dimensional array
2. int arr[] = {10, 20, 30, 40} // declaration and initialization
   above is same as "int arr[4] = {10, 20, 30, 40}"
3. int arr[6] = {10, 20, 30, 40}
   above is same as "int arr[] = {10, 20, 30, 40, 0, 0}"

Write statement for declaring an array for floating point and character data (value).

Accessing Array Elements:

Array elements are accessed by using an integer index. Array index starts with 0 and goes till size of array minus 1.

Ex1: program for how to assign the array element

```
int main()
{
 int arr[5];
 arr[0] = 5;
 arr[2] = -10;
 arr[3/2] = 2; // this is same as arr[1] = 2
```

For Micro Notes by the Student

```
  arr[3] = arr[0];
   printf("%d %d %d %d", arr[0], arr[1], arr[2], arr[3]);
   return 0;
}
```

Output:

Ex2: Reading the element into an array and print all the element of array

```
#include <stdio.h>
 int main ()
  {
         int n[ 10 ]; /* n is an array of 10 integers */
       int i,j;
        /* initialize elements of array n to 0 */
        for ( i = 0; i < 10; i++ )
           {
              n[ i ] = i + 10; /* set element at location i to i + 100 */
           }
     /* output each array element's value */
       for (j = 0; j < 10; j++ )
        {
           printf("Element[%d] = %d\n", j, n[j] );
        }
   return 0;
}
```

Output:

For Micro Notes by the Student

2 Two – Dimensional Array:

To store tables we need two dimensional arrays. Each table consists of rows and columns. Two dimensional arrays are declare as

Syntax:

Storageclass datatype arrayname [row-size][col-size];

Initializing Two Dimensional Arrays:

- They can be initialized by following their declaration with a list of initial values enclosed in braces.
  Ex:- int table[2][3] = {0,0,0,1,1,1};
- Initializes the elements of first row to zero and second row to one. The initialization is done by row by row. The above statement can be written as
  int table[2][3] = {{0,0,0},{1,1,1}};
- When all elements are to be initialized to zero, following short-cut method may be used.
  int m[3][5] = {{0},{0},{0}};

Ex: Write a program for reading and printing 2-D array.

For Micro Notes by the Student

Output:

3 Multi-Dimensional Array:

- C allows arrays of three or more dimensions. The exact limit is determined by Compiler.

  Syntax:

  Storageclass datatype array-name[s1][s2][s3] - - - - - [sn];

  where s1,s2,...sn is size of dimension.

  Ex:- int Survey[3][5][2];

Limitations of Array:

- the dimension of an array is determined the moment the array is created, and cannot be changed later on;
- the array occupies an amount of memory that is proportional to its size, independently of the number of elements that are actually of interest;

For Micro Notes by the Student

- if we want to keep the elements of the collection ordered, and insert a new value in its correct position, or remove it, then, for each such operation we may need to move many elements (on the average, half of the elements of the array); this is very inefficient.

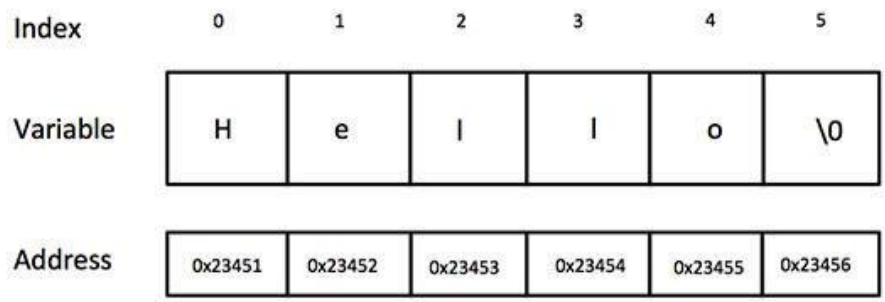
String:

- Strings are declared in a similar manner as arrays. Only difference is that, strings are of **char type.**
- Strings are actually one-dimensional array of characters terminated by a null character (‘**\0’).**

Declaration and initialization:

```
char c[] = “abcd”;
   OR,
char c[50] = “abcd”;
   OR,
char c[] = {‘a’, ‘b’, ‘c’, ‘d’, ‘\0’};
   OR,
char c[5] = {‘a’, ‘b’, ‘c’, ‘d’, ‘\0’};
char greeting[6] = {‘H’, ‘e’, ‘l’, ‘l’, ‘o’, ‘\0’};
   (or)
char greeting[] = “Hello”;
```

The given string is initialized and stored in the form of arrays as below



Note: Actually, you do not place the null character at the end of a **string constant.** The C compiler automatically places the ‘\0’ at the end of the string when it initializes the array.

Example:

```
#include <stdio.h>

int main () {
```

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
printf("Greeting message: %s\n", greeting );
return 0;
}
```

For Micro Notes by the Student

Output:

Greeting message: Hello

Ex. Write a program to read and print the string using all input output function. (getchar(), gets(), scanf() with different format specifiers).

For Micro Notes by the Student

String Library or Manipulation Function:

C supports a wide range of functions that manipulate null-terminated strings

Sr.No.	Function & Purpose
1	strcpy(s1, s2); Copies string s2 into string s1.
2	strcat(s1, s2); Concatenates string s2 onto the end of string s1.
3	strlen(s1); Returns the length of string s1.
4	strcmp(s1, s2); Returns 0 if s1 and s2 are the same; less than 0 if s1<s2;
5	strchr(s1, ch); Returns a pointer to the first occurrence of character ch in string s1.
6	strstr(s1, s2); Returns a pointer to the first occurrence of string s2 in string s1.

The following example uses some of the above-mentioned functions –

For Micro Notes by the Student

```
#include <stdio.h>
#include <string.h>
int main ()
{
      char str1[12] = "Hello";
      char str2[12] = "World";
      char str3[12];
      int  len ;
      /* copy str1 into str3 */
      strcpy(str3, str1);
      printf("strcpy( str3, str1) :  %s\n", str3 );

      /* concatenates str1 and str2 */
      strcat( str1, str2);
      printf("strcat( str1, str2):    %s\n", str1 );

      /* total lenghth of str1 after concatenation */
      len = strlen(str1);
      printf("strlen(str1) :  %d\n", len );

      return 0;
}
```

When the above code is compiled and executed, it produces the following

Output −

```
strcpy( str3, str1) :  Hello
strcat( str1, str2)  :  HelloWorld
strlen(str1) :  10
```

Rewrite the above functionalities without using string handling functions

For Micro Notes by the Student

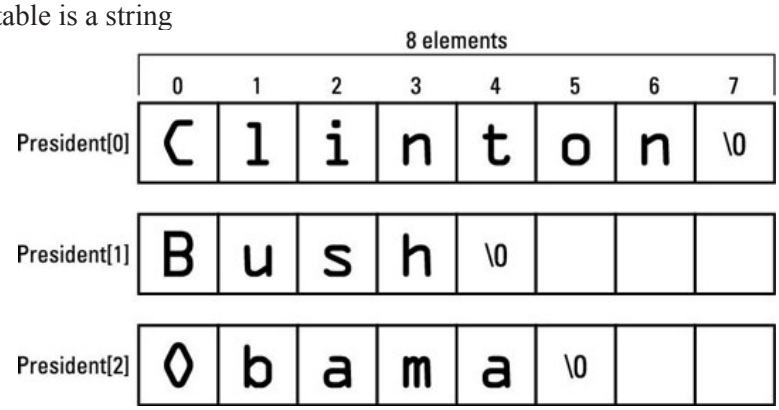
Array of String:

Syntax:

Datatype arrayname[row-size][col-size];

Example: char President[4][8];

- A string is a 1-D array of characters, so an array of strings is a 2-D array of characters or array of arrays of character.
- A NULL character('\0') must terminate each character string in the array.
- We can think of an array of strings as a table of strings, where each row of the table is a string



Write a program to read n number of names and print them.

For Micro Notes by the Student

Structure:

- structure is user defined data type available in C that allows combining data items of different kinds ie int, float, char, array and pointers.
- Structures are used to represent a record. Suppose you want to keep track of your books in a library. You might want to track the following attributes about each book −
  - □ Title
  - □ Author
  - □ Subject
  - □ Book ID
  - □ Publishing date

Defining a Structure

To define a structure, you must use the struct statement. The struct statement defines a new data type, with more than one member. The format of the struct statement is as follows −

Syntax:

```
struct   tag
{
member 1;
member 2;
...
member m;
} variable1, variable2.....  , variable n;
```

- The structure tag is optional and each member definition is a normal variable definition, such as int i; or float f; or any other valid variable definition.
- At the end of the structure's definition, before the final semicolon, you can specify one or more structure variables but it is optional.
- Structure members can't be initialized within structure and there is no memory allocated for the members of the structure.

For Micro Notes by the Student

- Memory for all the member will allocate after creating the **variable of struct type.**
- Here is the way you would declare the Books structure −

```
struct Books {
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
} book;
```

• Structure variable may be defined as a member of another structure. In that case, the declaration of the embedded structure must appear before the declaration of the outer structure.

```
struct date
 {
int    month;
int day;
int year;
} ;
struct Books {
  char  title[50];
  char  author[50];
  char  subject[100];
  int   book_id;
  struct date publishing_date;
} book;
```

The second structure (Books) contains another structure(data)as one of the members.

Accessing Structure Members:

- To access any member of a structure, we use the member **access operator (.) ie dot operator.**
- The member access operator is coded as a period between the structure **variable name** and the **structure member** (book.title, book.subject, ...) that we wish to access. You would use the **keyword struct** to define variables of structure type.

Example1:

```
struct Point
{
  int x, y;
};
```

For Micro Notes by the Student

```
int main()
{
  struct Point p1 = {0, 1};
 // Accesing members of point p1
 p1.x = 20;
printf ("x = %d, y = %d", p1.x, p1.y);
return 0;
}
```

Output:

```
20  1
```

Example 2:

```
#include <stdio.h>
#include <string.h>
 struct Books
      {
       char  title[50];
        char  author[50];
       char  subject[100];
       int   book_id;
      };
 int main( )
 {
        struct  Books  Book1={"C  Programming","Nuha  Ali","C  Programming
       Tutorial",6495407};              /* Declaration and initialization of Book1 of type
       Book */
  struct Books Book2;        /* Declare Book2 of type Book */

  /* book 2 specification */
  strcpy( Book2.title, "Telecom Billing");
  strcpy( Book2.author, "Zara Ali");
  strcpy( Book2.subject, "Telecom Billing Tutorial");
  Book2.book_id = 6495700;
   /* print Book1 info */
  printf( "Book 1 title : %s\n", Book1.title);
  printf( "Book 1 author : %s\n", Book1.author);
  printf( "Book 1 subject : %s\n", Book1.subject);
  printf( "Book 1 book_id : %d\n", Book1.book_id);
  /* print Book2 info */
  printf( "Book 2 title : %s\n", Book2.title);
```

For Micro Notes by the Student

```
  printf( "Book 2 author : %s\n", Book2.author);
  printf( "Book 2 subject : %s\n", Book2.subject);
  printf( "Book 2 book_id : %d\n", Book2.book_id);
  return 0;
}
```

Output:

Book 1 title : C Programming

Book 1 author : Nuha Ali

Book 1 subject : C Programming Tutorial

Book 1 book_id : 6495407

Book 2 title : Telecom Billing

Book 2 author : Zara Ali

Book 2 subject : Telecom Billing Tutorial

Book 2 book_id : 6495700

Array of structures:

Syntax:

structtype variablename[size];

Ex: struct point arr[10];

In above example arr is variable which store maximum 10 values of struct point type unlike normal variable;

Example for an array of structures.

```
struct Point  // Structure Name
{
  int x, y;
};

int main()
{
  // Create an array of structures
  struct arr[10];
```

For Micro Notes by the Student

```
  // Access array members
  arr[0].x = 10;
  arr[0].y = 20;

  printf("%d %d", arr[0].x, arr[0].y);
  return 0;
}
```

Output:

typedef statement:

The typedef allows us to create an **alias or new name** for an existing type or user defined type. The syntax of typedef is as follows:

Syntax:

typedef data_type new_name;

typedef: It is a keyword.

data_type: It is the name of any existing type or user defined type created using structure/union.

new_name: alias or new name you want to give to any existing type or user defined type.

Ex: typedef int myint;

Myint a=20; //valid statement

```
#include <stdio.h>
#include <string.h>

typedef struct Books {
  char title[50];
  char author[50];
  char subject[100];
  int book_id;
} Book;

int main( ) {
  Book book;
  strcpy( book.title, "C Programming");
```

For Micro Notes by the Student

```
  strcpy( book.author, "Nuha Ali");
  strcpy( book.subject, "C Programming Tutorial");
  book.book_id = 6495407;

  printf( "Book title : %s\n", book.title);
  printf( "Book author : %s\n", book.author);
  printf( "Book subject : %s\n", book.subject);
  printf( "Book book_id : %d\n", book.book_id);
  return 0;
}
```

Output:

Book title : C Programming
Book author : Nuha Ali
Book subject : C Programming Tutorial
Book book_id : 6495407

Union in C:

Like **Structures,** union is a user defined data type. In union, **all members share the same memory location.**

For example in the following C program, both x and y share the same location. If we change x, we can see the changes being reflected in y.

Example

```
#include <stdio.h>

// Declaration of union is same as structures
union test
{
 int x, y;
};

int main()
{
  // A union variable t
  union test t;
 t.x = 2; // t.y also gets value 2
printf ("After making x = 2:\n x = %d, y = %d\n\n",
        t.x, t.y);
```

For Micro Notes by the Student

```
    t.y = 10;  // t.x is also updated to 10
    printf ("After making Y = 'A':\n x = %d, y = %d\n\n",
            t.x, t.y);
    return 0;
}
```

Output:

After making x = 2:
x = 2, y = 2

After making Y = 'A':
x = 10, y = 10

How is the size of union decided by compiler?

Size of a union is taken according the size of largest member in union.

```
#include <stdio.h>

union test1
{
  int x;
  int y;
};
union test2{
 int x;
 char y;
};
union test3
{
 int arr[10];
char y;
};
int main()
{
    printf ("sizeof(test1) = %d, sizeof(test2) = %d,"
            "sizeof(test3) =  %d", sizeof(test1),
             sizeof(test2), sizeof(test3));
     return 0;
}
```

For Micro Notes by the Student

Output:

sizeof(test1) = 4, sizeof(test2) = 4,sizeof(test3) = 40

Enumeration (or enum) in C:

Enumeration (or enum) is a user defined data type in C. It is mainly used to assign names to integral constants, the names make a program easy to read and maintain.

enum State {Working = 1, Failed = 0};

The keyword 'enum' is used to declare new enumeration types in C .

enum declaration.

enum flag{constant1, constant2, constant3, ... };

The name of enumeration is "flag" and the constant are the values of the flag. By default, the values

of the constants are as follows: constant1 = 0, constant2 = 1, constant3 = 2 and so on.

Variables of type enum can also be defined. They can be defined in two ways:

enum week{Mon, Tue, Wed};

enum week day;

Or

enum week{Mon, Tue, Wed}day;

An example program to demonstrate working of enum in C

```
#include<stdio.h>

enum week{Mon, Tue, Wed, Thur, Fri, Sat, Sun};

int main()
{
   enum week day;
   day = Wed;
   printf("%d",day);
   return 0;
}
```

Output: 2

In the above example, we declared "day" as the variable and the value of "Wed" is allocated to day, which is 2. So as a result, 2 is printed.

For Micro Notes by the Student

- Two enum names can have same value. For example, in the following C program both 'Failed' and 'Freezed' have same value 0.
  enum State {Working = 1, Failed = 0, Freezed = 0};

- If we do not explicitly assign values to enum names, the compiler by default assigns values starting from 0. For example, in the following C program, sunday gets value 0, monday gets 1, and so on.
  enum day {sunday, monday, tuesday, wednesday, thursday, friday, saturday};
- We can assign values to some name in any order. All unassigned names get value as value of previous name plus one.
  enum day {sunday = 1, monday, tuesday = 5,
  wednesday, thursday = 10, friday, saturday};
- The value assigned to enum names must be some integeral constant, i.e., the value must be in range from minimum possible integer value to maximum possible integer value.

Pointer:

- Pointers in C language is a variable that stores/points the address of another variable. A Pointer in C is used to allocate memory dynamically i.e. at run time.
- The pointer variable might be belonging to any of the data type such as int, float, char, double, short etc.
- Pointer is a derived data type.

Pointer Declaration:

Syntax:

Datatype *var_name;

Example : int *p; char *p;
Where, * is used to denote that "p" is pointer variable and not a normal variable.

Points To Remember About Pointers In C:

- Normal variable stores the value whereas pointer variable stores the address of the variable.
- The content of the C pointer always be a whole number i.e. address.
- Always C pointer is initialized to null, i.e. int *p = null.
- The value of null pointer is 0.
- & symbol is used to get the address of the variable.
- * symbol is used to get the value of the variable that the pointer is pointing to.
- If a pointer in C is assigned to NULL, it means it is pointing to nothing.

For Micro Notes by the Student

- Two pointers can be subtracted to know how many elements are available between these two pointers.
- But, Pointer addition, multiplication, division are not allowed.
- The size of any pointer is 2 byte (for 16 bit compiler).

Advantages:

- Pointers are more efficient in handling arrays and data tables.
- Pointers can be used to return multiple values from a function via function arguments.
- Pointers allow passing a function as argument to other functions.
- The use of pointer arrays to character strings results in saving of data storage space in memory.
- Pointers allow C to support dynamic memory management.
- Pointers provide an efficient way for manipulating dynamic data structures such as structures, linked lists, queues, stacks and trees.
- Pointers increase the execution speed and thus reduce the program execution time.

Example program for pointers in C.

```
#include <stdio.h>
int main()
{
  int *ptr, q;
  q = 50;
  /* address of q is assigned to ptr */
  ptr = &q;
  /* display q's value using ptr variable */
  printf("%d  %d  ",q, *ptr);
  return 0;
}
```

Output: 50 50

Explanation:

For Micro Notes by the Student

Write a program to illustrate the use of pointer

Pointer to Array:

Syntax:

data_type (*var_name)[size_of_array];

int (*ptr)[10];

- Here ptr is pointer that can point to an array of 10 integers. Since subscript([]) have higher precedence than indirection(*), it is necessary to enclose the indirection operator(*) and pointer name inside parentheses.
- Here the type of ptr is 'pointer to an array of 10 integers'.

Note : The pointer that points to the 0th element of array and the pointer that points to the whole array are totally different. The following program shows this:

/* C program to understand difference between pointer to an integer and pointer to an array of integers.

```
#include<stdio.h>
 int main()
{
   // Pointer to an integer
```

For Micro Notes by the Student

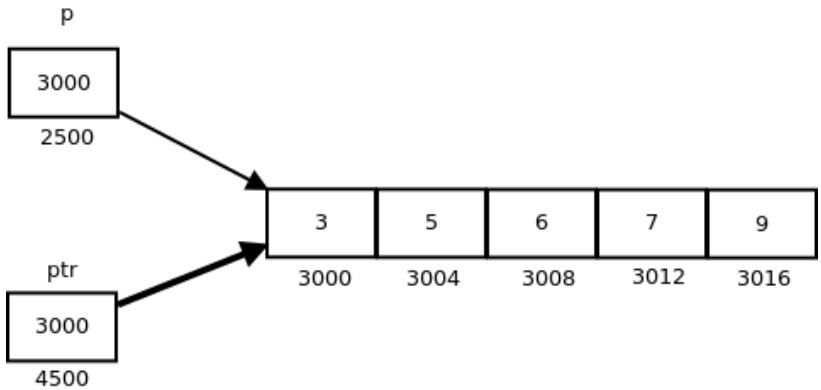
```
    int *p;

    // Pointer to an array of 5 integers
    int (*ptr)[5];
    int arr[5];

    // Points to 0th element of the arr.
    p = arr;

    // Points to the whole array arr.
    ptr = &arr;

    printf("p = %p, ptr = %p\n", p, ptr);
    p++;
    ptr++;
    printf("p = %p, ptr = %p\n", p, ptr);

    return 0;
}
```

Output:

p: is pointer to 0th element of the array arr, while **ptr: is a pointer** that points to the whole array arr.

- The base type of p is int while base type of ptr is 'an array of 5 integers'.
- We know that the pointer arithmetic is performed relative to the base size, so if we write ptr++, then the pointer ptr will be shifted forward by 20 bytes.

The following figure shows the pointer p and ptr. Darker arrow denotes pointer to an array.



For Micro Notes by the Student

On dereferencing a pointer expression we get a value pointed to by that pointer expression. Pointer to an array points to an array, so on dereferencing it, we should get the array, and the name of array denotes the base address. So whenever a pointer to an array is dereferenced, we get the base address of the array to which it points.

```
// C program to illustrate sizes of pointer of array
#include<stdio.h>

int main()
{
   int arr[] = { 3, 5, 6, 7, 9 };
   int *p = arr;
   int (*ptr)[5] = &arr;

   printf("p = %p, ptr = %p\n", p, ptr);
   printf("*p = %d, *ptr = %p\n", *p, *ptr);

   printf("sizeof(p) = %lu, sizeof(*p) = %lu\n",
                    sizeof(p), sizeof(*p));
   printf("sizeof(ptr) = %lu, sizeof(*ptr) = %lu\n",
                  sizeof(ptr), sizeof(*ptr));
   return 0;
}
```

Output:

Pointer to Structure:

can be created and accessed using pointers. A pointer variable of a structure can be created as below:

```
struct name
 {
   member1;
   member2;
   .
   .
 };
```

For Micro Notes by the Student

```
int main()
{
   struct name *ptr;
}
```

Here, the pointer variable of type struct name is created.

Accessing structure's member through pointer

A structure's member can be accesssed through pointer in two ways:

1. Referencing pointer to another address to access memory
2. Using dynamic memory allocation

Referencing pointer to another address to access the memory

Consider an example to access structure's member through pointer.

```
#include <stdio.h>
typedef struct person
{
  int age;
  float weight;
};

int main()
{
   struct person *personPtr, person1;
   personPtr = &person1; // Referencing pointer to memory address of person1

   printf("Enter integer: ");
   scanf("%d",&(*personPtr).age);

   printf("Enter number: ");
   scanf("%f",&(*personPtr).weight);

   printf("Displaying: ");
   printf("%d%f",(*personPtr).age,(*personPtr).weight);

   return 0;
}
```

In this example, the pointer variable of type struct person is referenced to the address of person1. Then, only the structure member through pointer can can accessed.

For Micro Notes by the Student

(*personPtr).age refer to the age of a person.
(*personPtr).weight refer to the weight of a person.

Using -> operator to access structure pointer member

Structure pointer member can also be accessed using -> operator.
(*personPtr).age is same as personPtr->age
(*personPtr).weight is same as personPtr->weight

Self Referential Structures:

Self Referential structures are those structures that have one or more pointers which point to the same type of structure, as their member.

In other words, structures pointing to the same type of structures are self-referential in nature.

Example:

```
struct node {
   int data1;
   char data2;
   struct node* link;
};

int main()
{
   struct node ob;
   return 0;
}
```

In the above example 'link' is a pointer to a structure of type 'node'. Hence, the structure 'node' is a self-referential structure with 'link' as the referencing pointer.

An important point to consider is that the pointer should be initialized properly before accessing, as by default it contains garbage value.

Self Referential Structure with Single Link:

These structures can have only one self-pointer as their member. The following example will show us how to connect the objects of a self-referential structure with the single link and access the corresponding data members. The connection formed is shown in the following figure.

For Micro Notes by the Student

```
#include <stdio.h>

struct node {
   int data1;
   char data2;
   struct node* link;
};

int main()
{
   struct node ob1; // Node1

   // Intialization
   ob1.link = NULL;
   ob1.data1 = 10;
   ob1.data2 = 20;

   struct node ob2; // Node2

   // Initialization
   ob2.link = NULL;
   ob2.data1 = 30;
   ob2.data2 = 40;

   // Linking ob1 and ob2
   ob1.link = &ob2;

   // Accessing data members of  ob2 using ob1
   printf("%d", ob1.link->data1);
   printf("\n%d", ob1.link->data2);
   return 0;
}
```

Output:

30
40

UNIT-III – STRINGS & POINTERS

1.How string are declared & initialized in c ?
2M/4M-june-2012

2.Define array of string -3M

3.Explain string Manipulation function &string library function are string handling function –May-16-10M,june-13,Dec-15-5M

4.Explain strspn-2M Dec-15

5.Differentiate strcpy &stcmp-3M June-14

6.what is the purpose of gets &puts function. 3M

7.What is null character ?and differentiate between NULL,'\0' &0 ? - 3M Dec-10

8.Explain about string to data conversion ? 3M –JUNE-12

9.Write a C program to reverse the string passed as an argument that cannot be altered –JUNE-12-7M,DEC-10-5M

10.Write a program using function to find the length of a string passed as an argument-DEC-10-3M

11.Write a program to find position of substring into main string if substring is not present then program display -5M

12.Write C function void insert(char a[],char c, int *n ,int i)that insert character c at index I in the array by shifting all elements above that position by 1 & incrementing n. 10M-JUNE-10

13.Find the numbers of words ,character &lines in the given text. 5M –JUNE-14,JUNE-15

14.Convert the string passed as an argument to its uppercase equivalent 5M –DEC-10

15.Write a program which will read a string &rewrite it in the alphabetical order [example –the word STRING should be written as GINRST] 5M-DEC-15

16.Write a program to replace substring with a new string 5M-DEC-15

17.Explain string I/O function .3M

18Differentiate between getchar() & scanf() for reading string .3M

19.Write a function to delete n character from a given position in a given string -5M

POINTER

1.Explain pointer arithmetic operators that are permitted on pointer with a sample c program 10M/5M/3M-MAY-16,DEC-15,JUNE-12.

2.Differentiate between pointer &variable. 2M-DEC-10

3.How pointer declared ,initialized ? what is pointer to another pointer 5M-DEC-10.

4.Dynamic Memory allocation . 2M/5M-DEC-15,MAY-16,JUNE-14

5.Write a program to swap /exchange two number using call by pointer method -5M JUNE-15

6.Discuss various application of pointer -5M-JUNE-14

7.What is mean by array of pointer ? when it will be used 3M-MAY-16

8.Write &explain a program for pointer to multidimensional array 5M-JUNE-15

For Micro Notes by the Student

UNIT-4 STRUCTURES & UNION

For Micro Notes by the Student

1.Write a program to find length of the string n. JUNE-17-5M

2.Write a brief notes on unions 2M-JUNE-14/2M-DEC-16

3.Explain about declaration ,initialization & accessing structures and also discuss about complex structure 10M-JUNE-14

4.Give the difference between structure & union -2M JUNE-15

5.What is self referential structure with program -3M –JAN-15,JUNE-17/5M

6.With the help of an example explain union of structures 5M-JAN-15

7.Define structure -2M-DEC-15

8.Write about enumerated data type-5M-DEC-15/2M/DEC-16-3M

9.Define structure ,How can we access the members of a structure using pointer.explain it with an example -7M-JUNE-15

10.What is meant by structure ? Discuss with a c program about operator on structure -10M-JAN-15

11.What is a structure ? Define a structure called salary & another structure called allowance structure .use the structure variable allowance in salary structure & write program to read data into the structure variable.5M-JUNE-15

12.Write a recursive function that would scan the created structure array & print the compute data,day,month,&year information if the date is an even number – 5M-JUNE-15

13.Explain the following

1) Nested structure 5M-JUNE-17
2)array of structure
3)union
4)pointer of structures
5)self referential structure
6)typedef
7)enumerated types

14.Write a program to explain use of array of structure. [class note program]