THIRUTHANGAL NADAR COLLEGE

(Belongs to the Chennaivazh Thiruthangal Hindu Nadar Uravinmurai Dharma Fund)

Selavayal, Chennai-51.

A Self-Financing Co-educational College of Arts & Science

Affiliated to the University of Madras

Accredited with 'B' Grade by NAAC

An ISO 9001: 2015 Certified Institution

NAME OF THE DEPARMENT: BCA (SHIFT-I_

SUBJECT : PROGRAMMING IN C

TOPIC :UNIT (1-5)

STAFF NAME :Mrs.EDITHMARIADELILAH

Unit I - Overview

- Character set
- Identifier
- Keywords
- Data Types
- Variables
- Constants
- Operators

Character set of C

- Character set is a set of alphabets, letters and some special characters
    - Alphabets : Uppercase : A – Z and Lowercase: a – z
    - Numbers : 0 – 9
    - Special Symbols : {, }, [,],?, +, -, *, /, %, !, &, # and more
- The words formed from the character set are building blocks of C language and are called as tokens
- Different types of token used in C
    - Identifiers
    - Keywords
    - Constants
    - Operators
    - Punctuation Symbols

Identifier

- Identifier refers to name given to entities such as variables, functions, structures etc

- They are created to give unique name to an entity to identify it during the execution of the program.

- **Rules for writing an identifier**

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore
3. They are case sensitive
4. Keywords cannot be used as identifiers

C Keywords

- Keywords are predefined, reserved words used in programming that have special meanings to the compiler.

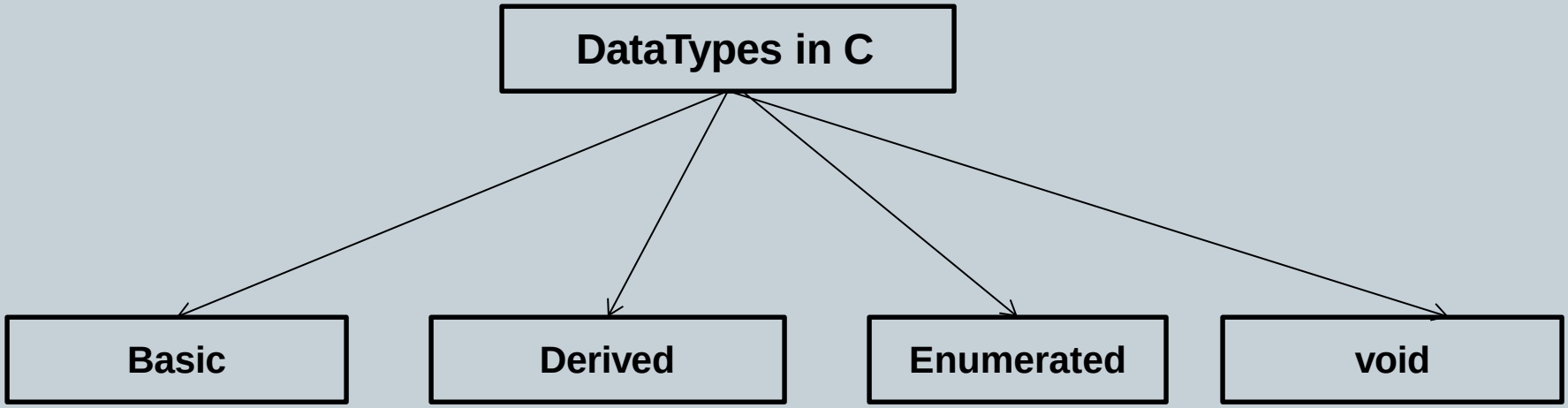
  Example: int age;

- Here, int is a keyword that indicates ‘age' is a variable of type integer.
- As C is a case sensitive language, all keywords must be written in lowercase.
- Some of the keywords in C language are given below

auto	break	case	char	const	continue	default	do
double	else	enum	extern	float	for	goto	if
int	long	register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void	volatile	while

C - Data Types

- A data type specifies the type of data that a variable can store such as integer, floating, character etc.
- The type of a variable determines how much space it occupies in storage.
- The data types in C can be classified as follows:



Data Types - Classification

THIRUTHANGAL NADAR COLLEGE
knowledge is power

S.No	Types & Description
1	**Basic Types** They are arithmetic types and are further classified into: (a) integer types and (b) floating-point types.
2	**Derived types** They include (a) Pointer types, (b) Array types, (c) Structure types, (d) Union types and (e) Function types.
3	**Enumerated types** They are again arithmetic types and they are used to define variables that can only assign certain discrete integer values throughout the program.
4	**The type void** The type specifier void indicates that no value is available.

Basic Data Types

- The basic data types are integer-based and floating-point based.
- C language supports both signed and unsigned literals

Data Types	Memory Size	Range
char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	−32,768 to 32,767
Unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	1.2E-38 to 3.4E+38
double	8 byte	2.3E-308 to 1.7E+308
long double	10 byte	3.4E-4932 to 1.1E+4932

Variables in C

THIRUTHANGAL NADAR COLLEGE
knowledge is power

- A **variable** is a name assigned to a memory location. It is used to
- store data.
- Its value can be changed and reused many times.
- It is a way to represent memory location through a name so that it
- can be easily identified.
- Each variable in C has
    - a specific type, which determines the size and layout of the variable's memory
    - range of values that can be stored within that memory
    - set of operations that can be applied to the variable.

Declaration of Variables

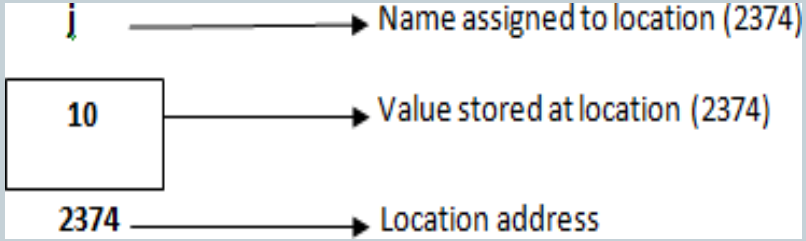
THIRUTHANGAL NADAR COLLEGE
knowledge is power

- A variable must be declared before using it
- A variable declaration specifies a data type and contains a list of one or more variables of that type as follows -

  datatype var_list ;

- Example: int rollno, age; float salary; char grade;
- A variable can also be initialized with a value at the time of declaration

  int j = 10;



Constants in C

- Constants refer to fixed values that the program may not alter during its execution.
- These fixed values are also called **literals**.
- An identifier can be defined as a constant using the const keyword

  Example: const double PI = 3.14;
- C supports the following types of constant -

Constant	Example
Decimal	10, 230, 4595
Real or Floating-point	120.3, 430.267, 40.6
Octal	071, 063, 036
Hexadecimal	0x29, 0x7d, 0xff
Character	'a', 'b', 'x'
String	"C program", "Shasun"

Operators in C

- An operator is simply a symbol that is used to perform operations

- C language is rich in built-in operators and provides the following types of operators −

    - Arithmetic Operators
    - Relational Operators
    - Logical Operators
    - Ternary or Conditional Operators
    - Bitwise Operators
    - Assignment Operator

Arithmetic Operator

- An arithmetic operator performs mathematical operations such as addition, subtraction and multiplication on numerical values (constants and variables)

Operator	Meaning of Operator
+	addition or unary plus
-	subtraction or unary minus
*	multiplication
/	division
%	remainder after division (modulo division)
++	Increments the value by 1 (unary)
--	Decrements the value by 1 (unary)

Relational Operators

- A relational operator checks the relationship between two operands. If the relation is true, it returns 1; if the relation is false, it returns value 0.

Operator	Meaning of Operator	Example
==	Equal to	3 == 3 returns 1
>	Greater than	9 > 4 returns 1
<	Less than	7 < 3 returns 0
!=	Not equal to	3 != 3 returns 0
>=	Greater than or equal to	8 >= 2 returns 1
<=	Less than or equal to	7 <= 5 returns 0

Logical Operators

- Logical operator is used to evaluate two or more relational expressions. It returns either 0 or 1 depending upon whether expression results true or false.

Operator	Meaning of Operator	Example
&&	Logical AND True only if all operands are true	(3 == 5) && (7 > 5) returns 0
||	Logical OR True only if either one operand is true	(3 == 5) || (7 > 5) returns 1
!	Logical NOT True only if the operand is 0	! (5 == 5) returns 0

Conditional & Bitwise Operators

- **conditional operator** is otherwise called as ternary operator which works on 3 operands.
- Syntax

  conditional_Expression ? expression1 : expression2
- The first expression conditional_Expression is evaluated first. If conditional_Expression is true, expression1 is evaluated. If conditional_Expression is false, expression2 is evaluated.
- Example

  (a>b) ? printf(" a is greater") : printf("b is greater")
- **Bitwise operators** are used to perform bit operations. Decimal values are converted into binary values which are the sequence of bits and bit wise operators work on these bits.

  & (bitwise AND) | (bitwise OR) ~ (bitwise NOT)

  ^ (XOR) << (Left shift) >> (Right shift)

Assignment Operators

- An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

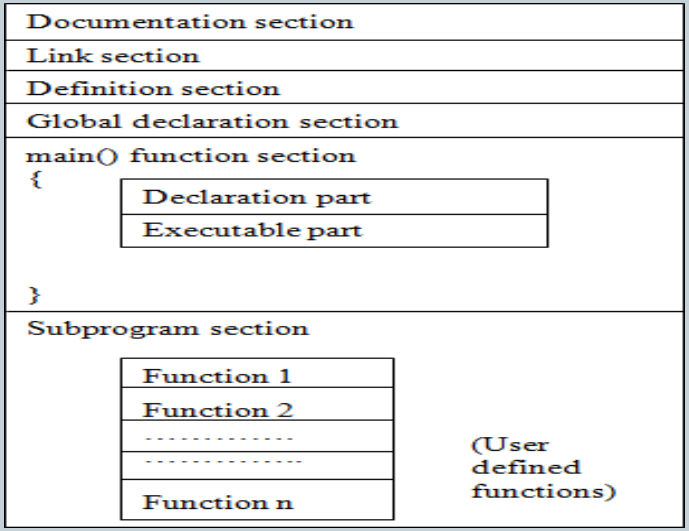
Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Unit II - Overview

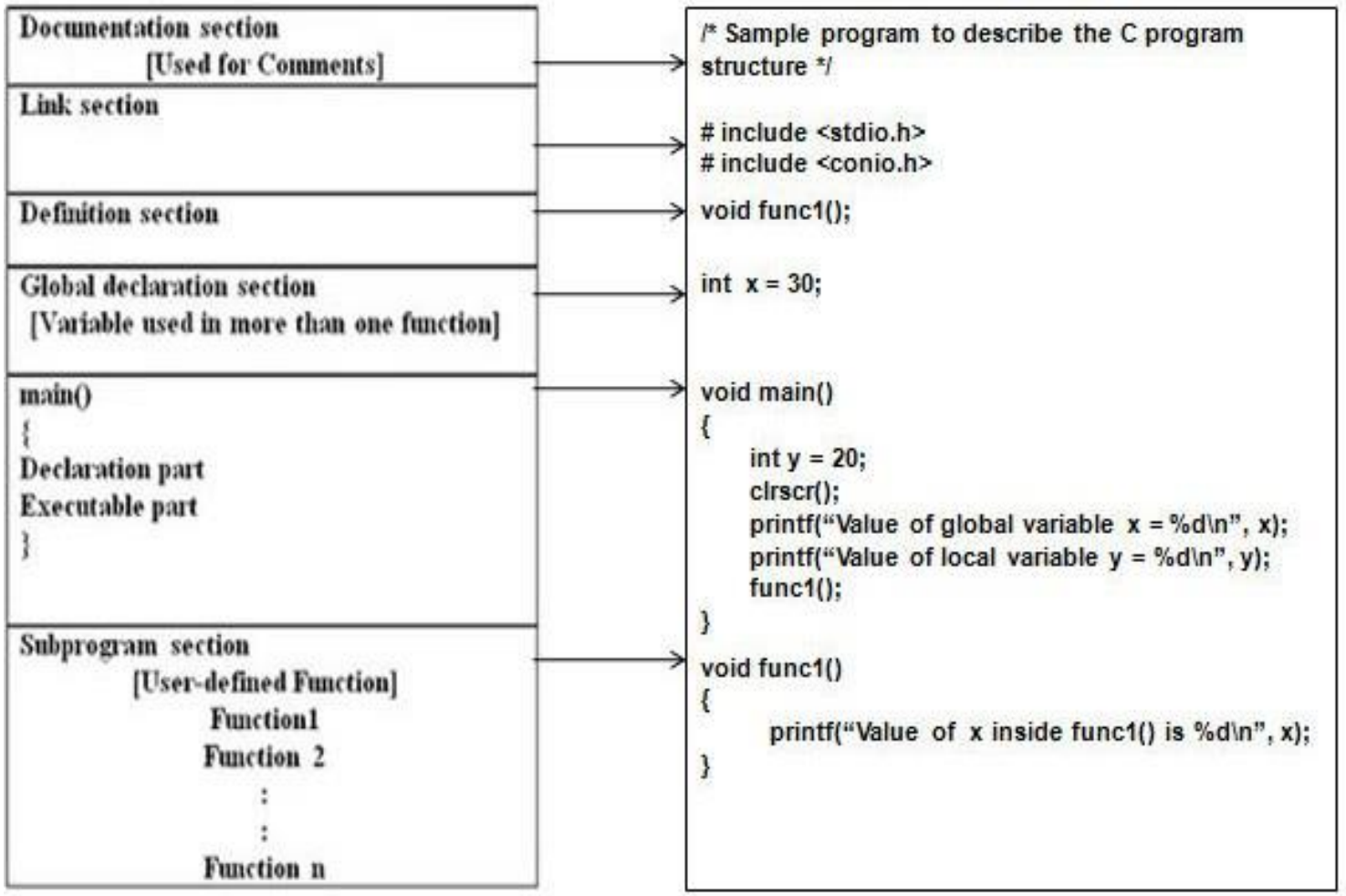
- C program structure
- Data input/output functions
- Decision making statements
- Looping statements

C Program Structure

- Basically a C program involves the following sections:
  - Documentations (Documentation Section)
  - Preprocessor Statements (Link Section)
  - Global Declarations (Definition Section)
  - The main() function
  - Local Declarations
  - Program Statements & Expressions
  - User Defined Functions



C Program Structure - Example



Data Input/Output Functions

- I/O operations are useful for program to interact with users. “stdio.h” is the standard C library for input output operations.
- While dealing with input-output operations in C, there are two important streams that play their role. These are:

Standard File	File Pointer	Description
Standard Input	stdin	Reads input from devices such as the **keyboard** as a data stream
Standard Output	stdout	Prints output to a device such as a **monitor**.

- The functions used to handle console I/O in C-
  - getchar() & putchar()
  - gets() & puts()
  - scanf() & printf()

getchar() & putchar()

- getchar() function is used to read a single character from the standard input
- Syntax:

  var_name = getchar();

```
Example:
# include<stdio.h>
void main()
 {
     char grade;
     grade = getchar();
 }
```

- putchar() function is used to write a single character on the standard output
- Syntax:

  putchar(var_name);

```
Example:
# include<stdio.h>
void main()
 {
     char grade = 'A';
     putchar(grade);
     putchar('\n');
 }
```

gets() & puts()

- gets(char *str) reads a line from stdin into the string pointed to by str and is terminated when the new line is read or EOF is reached.
- Syntax:

  char *gets(char *str);

- puts(char *str) is used to write a string to stdout but it does not include null characters. A new line character need to be append to the output.
- Syntax:

  int puts(const char *str);

Example:

```
#include<stdio.h>
void main()
{
    char college_ name[30];
    printf("Enter your college name\n");
    gets(college_name);
    printf("your college name is :\n");
    puts(college_name);
}
```

scanf() & printf()

- The **scanf()** function is used for formatted input. It reads the input data from the console.

  scanf("format string",argument_list);

- **printf()** function is used for formatted output. It prints the given statement to the console.

  printf("format string",argument_list);

- The **format string** can be %d (integer), %c (character), %s (string), %f (float) etc.

Example:

```
# include<stdio.h>
void main()
 {
     int rollno;
     char sname[30];
     printf("Enter rollno & student name\n");
     scanf("%d %s", &rollno,&sname);
     printf("Roll no  :%d, Name : %s", rollno,sname);
  }
```

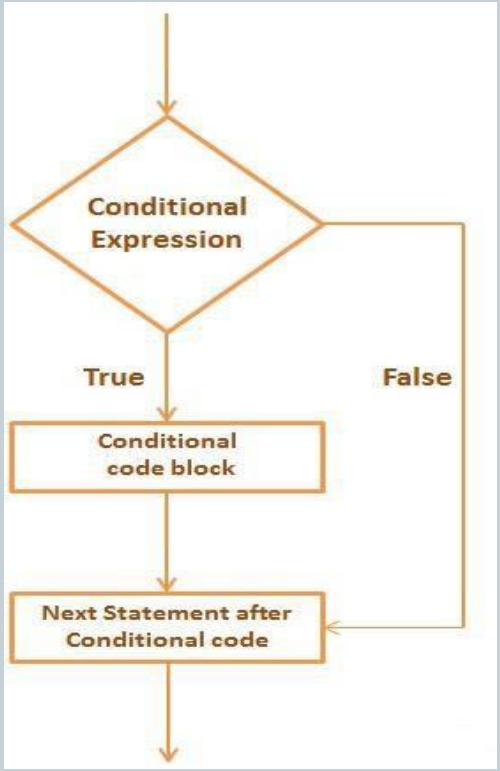
C – Decision Making

THIRUTHANGAL NADAR COLLEGE
knowledge is power

- C conditional statements allows to make decision, based upon the result of a test_expression condition. These statements are called as Decision Making Statements or Conditional Statements.

- They modify the control flow of the program.

- Conditional Statements in C (Two-way branching)
    - simple if statement
    - if-else statement
    - If -else if ladder



if Statement

- Simple If statement is used to execute a block of code if the condition is true.
- if the condition is true, then respective block of code is executed. Otherwise the block is skipped and the next statement gets executed.

Syntax:

```
If (condition)
{

//code to be executed if condition is true

}
next statement
```

Example:

```
# include <stdio.h>
void main()
{
   int n1, n2;
   printf("Enter two numbers\n");
   scanf("%d%d",&n1,&n2);
   if (n1 > n2)
   {
       printf("n1 is greater");
   }
}
```

if - else Statement

- The if-else statement is used to execute the code if condition is true or false.
- If the condition is true, then block-1 statements are executed. If it is false, block1 is skipped and block-2 statements gets executed.

Syntax:

```
If (condition)
{
// block-1
//code to be executed if condition is true
}
else
{
// block-2
//code to be executed if condition is false
}
next statement
```

Example:

```
# include <stdio.h>
void main()
{
   int n1, n2;
   printf("Enter two numbers\n");
   scanf("%d%d",&n1,&n2);
   if (n1 > n2)
   {
       printf("n1 is greater");
   }
  else
  {
     printf("n2 is greater");
  }
}
```

if - else if ladder

THIRUTHANGAL NADAR COLLEGE
knowledge is power

- The if else-if statement is used to execute one block of code from multiple conditions.

Syntax:

```
If (condition1)
{
//code to be executed if condition1 is true
}
else if (condition2)
{
//code to be executed if condition2 is true
}
else if (condition3)
{
//code to be executed if condition3 is true
}
...
else
{
//code to be executed if all the conditions are false
}
```

Example:

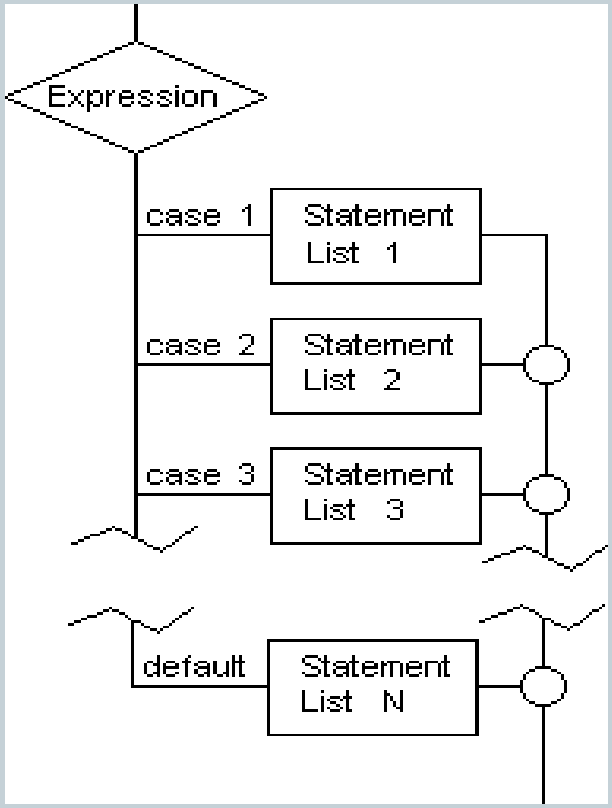
```
# include <stdio.h>
void main()
{
   int n1;
   printf("Enter a number\n");
   scanf("%d",&n1);
   if (n1>0)
       printf("%d is a Positive number");
  else if (n1<0)
      printf ("%d is a Negative number");
  else
      printf("Number is zero");
}
```

Switch-case Statement

- switch-case is a multi-way branching statement. It is used when there are multiple possibilities for the if statement.

Syntax:

```
switch (expression)
{
   case value1:
   //code to be executed;
   break;
   case value2:
   //code to be executed;
   break;
    ......
   default:
    /*code to be executed if all cases are not
      matched */
}
```



switch-case Rules

1. The switch expression must be of integer or character type.
2. The case value must be integer or character constant.
3. The case value can be used only inside the switch statement.
4. The break statement in switch case is not must. It is optional. If there is no break statement found in switch case, all the cases will be executed after matching the case value.

Example:

```
# include <stdio.h>
void main()
{
   int choice;
   printf("Please enter a no between 1 and 5: ");
   scanf("%d",&choice);
   switch(a)
   {
     case 1: printf("You chose One"); break;
     case 2: printf("You chose Two"); break;
     case 3: printf("You  chose Three"); break;
     case 4: printf("You chose Four"); break;
     case 5: printf("You chose Five."); break;
     default: printf("Invalid Choice");
   }
}
```

C – Loops

THIRUTHANGAL NADAR COLLEGE
knowledge is power

- The *loops in C language* are used to execute a block of code or a part of the program several times.
- There are three types of loops in C language that is given below:

  i) while ii) do while iii) for

i) **while loop**

- It iterates the code until condition is false. Here, condition is given before the code. So code may be executed 0 or more times.

Syntax:

```
while(condition)
{
    //code to be executed
}
```

Example:

```
# include <stdio.h>
void main()
{
      int i =1;
     while (i<=10)
     {
          printf("%d \n", i);
          ++i;
     }
}
```

C – Loops ..continued

THIRUTHANGAL NADAR COLLEGE
knowledge is power

ii) do-while loop

- It iterates the code until condition is false. Here, condition is given after the code. So at least once, code is executed whether condition is true or false.
- It is called as exit-controlled loop

Syntax:

```
do
{

    //code to be executed

}   while(condition);
```

Example:

```
# include <stdio.h>
void main()
{
       int i =10;
      do
      {
            printf("%d \n", i);
            i--;
      } while (i>=1);
}
```

C – Loops ..continued

iii) **for loop**

- Like while, it iterates the code until condition is false. Here, initialization, condition and increment/decrement is given before the code. So code may be executed 0 or more times.
- It is good if number of iteration is known by the user.

Syntax:

```
for (initialization;condition;incr / decr)
{
    //code to be executed
}
```

Example:

```
# include <stdio.h>
void main()
{
      int i ;
      for (i=1; i<=20; i+=2)
     {
          printf("%d \n", i);

     }
}
```

Unit III - Overview

- Functions
- Standard library functions
- User defined functions
- Function parameters and return value
- Parameter passing
- Recursive functions
- Storage classes

Functions in C

- A **function** is a group of statements that together perform a task.
- Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.
- Advantages of functions -
  - C functions are used to avoid rewriting same logic/code again and again in a program.
  - There is no limit in calling C functions to make use of same functionality wherever required.
  - We can call functions any number of times in a program and from any place in a program.
  - The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve readability of very large C programs.

Standard Library Functions

- The standard library functions are built-in functions in C programming to handle tasks such as mathematical computations, I/O processing, string handling etc.

- These functions are defined in the header file. When we include the header file, these functions are available for use.

- For example: The printf() is a standard library function to send formatted output to the screen (display output on the screen). This function is defined in "stdio.h" header file.

- There are other numerous library functions defined under various header files. sample

standard library functions in C programming

User Defined functions

- C allow programmers to define functions. Such functions created by the user are called user-defined functions.

- It reduces complexity of a big program and optimizes the code.

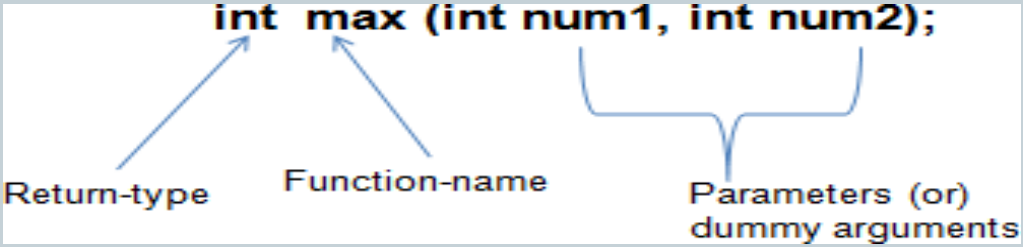
- There are 3 aspects in each C function. They are,
  - **Function declaration or prototype** – This informs compiler about the function name, function parameters and return value's data type.
  - **Function call** – This calls the actual function
  - **Function definition** – This contains all the statements to be executed.

Function Declaration

- A function **declaration** tells the compiler about a function name and how to call the function. The actual body of the function can be defined separately.
- A function declaration has the following syntax−

  return_type function_name(parameter list);

- Example:



- Parameter names are not important in function declaration. Only their type is required, so the following is also a valid declaration −

  int max(int, int);

Function Call & Definition

- While creating a C function, a definition of what the function has to do should be provided.

Syntax:

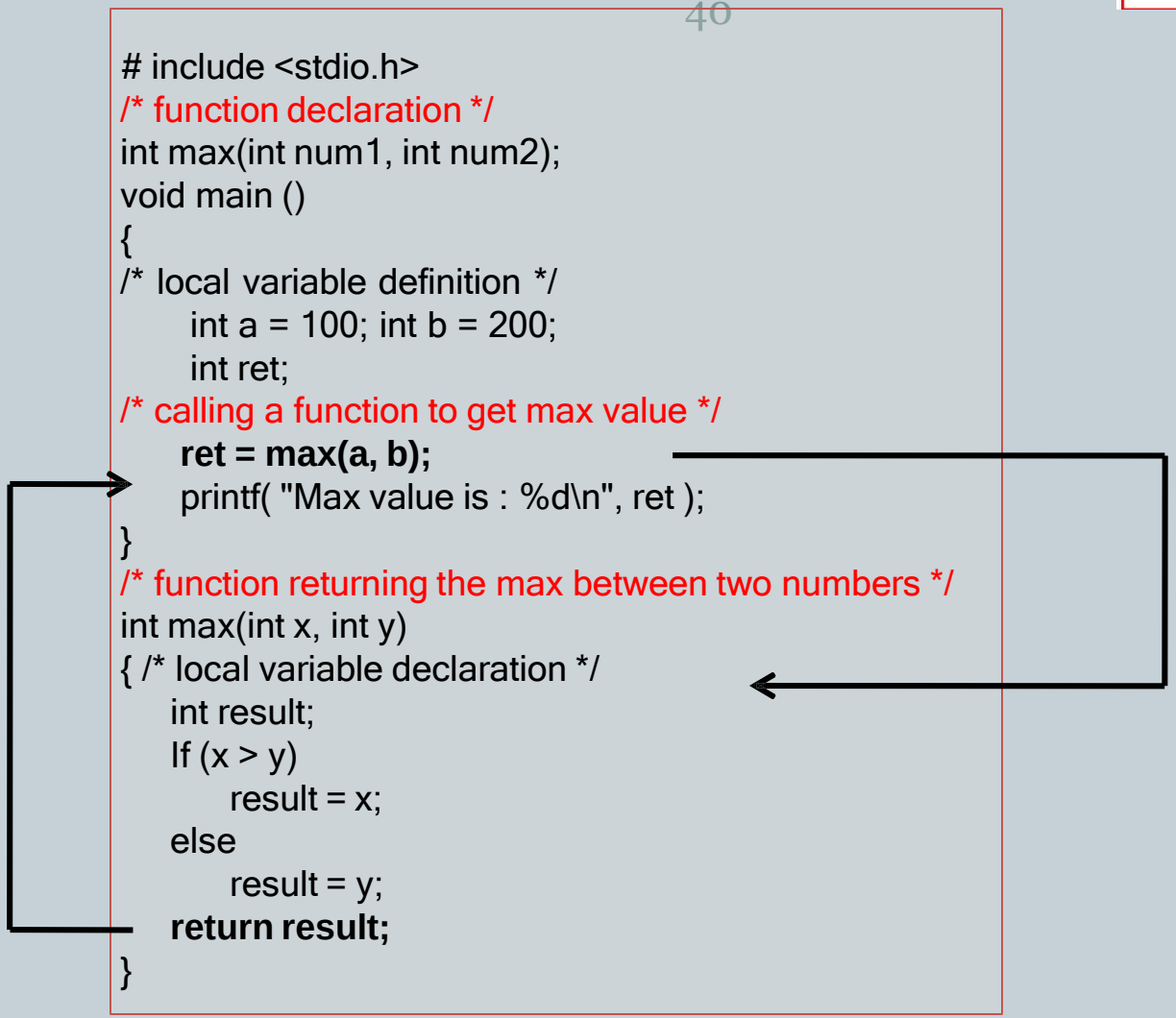
```
return_type  function_name (param_list)
{

    //body of function

}
```

Example:

```
int max (int x, int y)
{ /* x and y are formal arguments */

      if (x > y)
            return x;
       else
            return y;
}
```

- To call a function, we simply need to pass the required parameters along with the function name, and if the function returns a value, then the returned value can be stored.
- Syntax: function_name (arg1,arg2,..);
- Example: mval = max (a, b); // a & b – actual arguments

How function works?



```
# include <stdio.h>
/* function declaration */
int max(int num1, int num2);
void main ()
{
/* local variable definition */
      int a = 100; int b = 200;
      int ret;
/* calling a function to get max value */
     ret = max(a, b);
     printf( "Max value is : %d\n", ret );
}
/* function returning the max between two numbers */
int max(int x, int y)
{ /* local variable declaration */
     int result;
     If (x > y)
          result = x;
     else
          result = y;
     return result;
}
```

Return value

- A C function may or may not return a value from the function. If a function does not return any value then use void for the return type.

```
/* function without return value */
void write()
{
    printf("No return value");
 }
```

- If a function returns a value then we need to use any data type such as int, long, char etc. The return type depends on the value to be returned from the function.

```
/* function returning int value */
int get()
{
   return 10;
}
```

```
/* function returning float value */
float get_salary()
{
   return 12355.50;
}
```

Function parameters in C

- A ‘C’ function may have 0 or more parameters. The type of parameter in a function can be such as int, float, char etc. Parameters are otherwise called as arguments.
- If a function is to use arguments, it must declare variables that accept the values of the arguments. These variables are called the **formal parameters** of the function.
- Formal parameters behave like other local variables inside the function and are created upon entry into the function and destroyed upon exit.

```
/* function with one argument*/
int  square(int n)
{
   return n*n;
}
```

```
/* function without argument */
void get_count()
{
    count++;
    printf(“count = %d”,count);
}
```

User defined function - Types

- For better understanding of arguments and return value, user-defined functions can be categorized as:
  - Function with no arguments and no return value
  - Function with no arguments and a return value
  - Function with arguments and no return value
  - Function with arguments and a return value.

- Function types - examples

Passing arguments

- While calling a function, there are two ways in which arguments can be passed to a function

Call Type	Description
Call by value	This method copies the actual value of an argument into the formal parameter of the function. In this case, changes made to the parameter inside the function have no effect on the argument.
Call by reference	This method copies the address of an argument into the formal parameter. Inside the function, the address is used to access the actual argument used in the call. This means that changes made to the parameter affects the argument.

Call by value

- By default, C programming uses *call by value* to pass arguments.

```
# include <stdio.h>
void swap(int x, int y); // function declaration
int main ()
{
   int a = 100; int b = 200;
   printf("Before swap, value of a : %d\n", a );
   printf("Before swap, value of b : %d\n", b );
   swap(a, b); // function call to swap the values
   printf("After swap, value of a : %d\n", a );
   printf("After swap, value of b : %d\n", b );
}
/* function definition to swap the values */
void swap (int x, int y)
{
    int temp;
    temp = x;   //save the value of x
    x = y;        // copy y into x
    y = temp; // put temp into y
    return;
 }
```

Now, if we call the function **swap()** by passing actual values, the program produces the following result –

Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 100
After swap, value of b : 200

It shows that there are no changes in the values, though they had been changed inside the function.

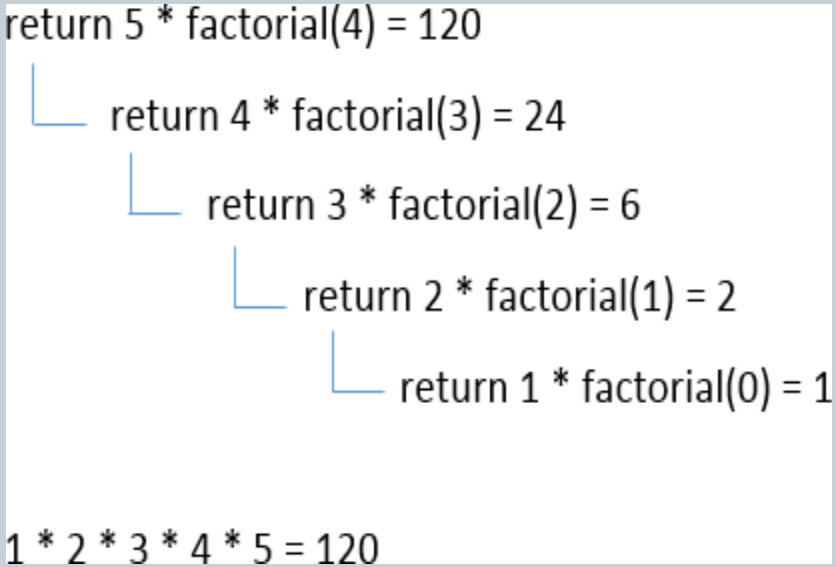
Recursive function

THIRUTHANGAL NADAR COLLEGE
knowledge is power

- A function that calls itself is known as a recursive function.
- To prevent infinite recursion, if...else statement (or similar approach) can be used where one branch makes the recursive call and other doesn't.

Example:

```
# include <stdio.h>
int factorial (int n)
{
   if (n == 0)
      return 1; //Terminating condition
   return (n * factorial (n -1));  // Recursive call
}
void main()
{
   int fact=0;
   fact=factorial(5);  // function call
   printf("\n factorial of 5 is %d", fact);
}
```



Storage Classes

- Storage class specifiers in C language tells the compiler
  - where to store a variable
  - how to store the variable
  - what is the initial value of the variable
  - life time of the variable
- **Syntax:**

  storage_specifier data_type variable _name;
- There are four storage classes in C programming.
  - auto
  - extern
  - static
  - register

Storage Classes..continued

Storage Class	Storage Place	Default Value	Scope	Life-time
auto	RAM	Garbage Value	Local	Within function
extern	RAM	Zero	Global	Till the end of main program. Declared outside a function in the program
static	RAM	Zero	Local	Till the end of main program, Retains value between multiple functions call
register	Register	Garbage Value	Local	Within function

Storage Class modifiers

- The **auto** keyword is applied to all local variables automatically. It is the default storage class.

  auto **int** b=10;

- The **extern** variable is visible to all the programs. It is used if two or more files are sharing same variable or function.

  extern int counter=0;

- The **static** variable is initialized only once and exists till the end of the program. It retains its value between multiple functions call.

  static int i=5;

- The **register** variable has a faster access than other variables. It is recommended to use register variable only for quick access such as in counter.

  register int counter=0;

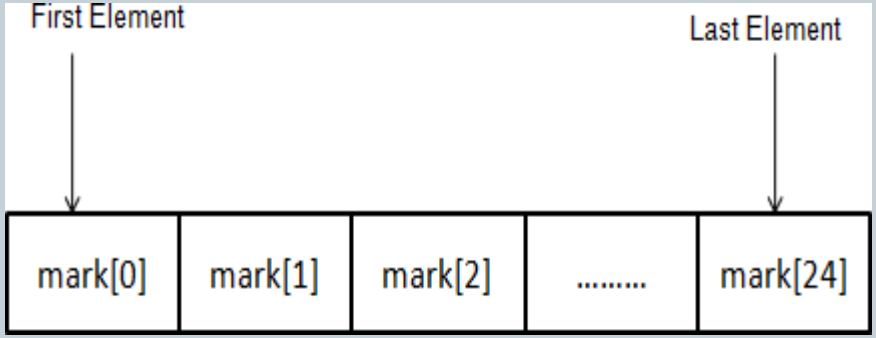
Storage class modifiers - Example

Unit IV - Overview

- Arrays
- Single dimension array
- Multi dimension array
- Character array - strings
- Structure
- Passing structure to functions
- Self-referential structure
- Union

Arrays in C

- An array is a collection of variables that are of similar data types and are referred by a common name.
- A specific element in an array is accessed by a particular index of that array.
- For example, if we need to store 25 subject marks of a student; declaring individual variables, such as mark1, mark2, ..., and mark25 is a tedious way.
- Instead, we can declare one array variable such as mark and use mark[0], mark[1], and ..., mark[24] to represent individual variables.
- The lowest address corresponds to the first element and the highest address to the last element.



Characteristics of Array

- Array may be used to hold the values of any data type. But all the elements of c array are *homogeneous* (similar).
- Array size must be a constant value.
- Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
- An array index always starts with 0.
- We can access any element randomly using the array index.
- By using loops, we can retrieve the elements of an array easily.

Array Declaration

- To declare an array in C, the type of the elements and the number of elements required by an array must be specified as follows −

  datatype arr_name [arraysize];

- This is called a ***single-dimensional*** **array**.

- The **arraysize** must be an integer constant greater than zero and **datatype** can be any valid C data type.

- Example: int mark[10]; declares a 10-element array called mark of type int

- The array index always starts with zero.

Array Initialization

- Arrays can be initialized using a single statement using the following syntax-

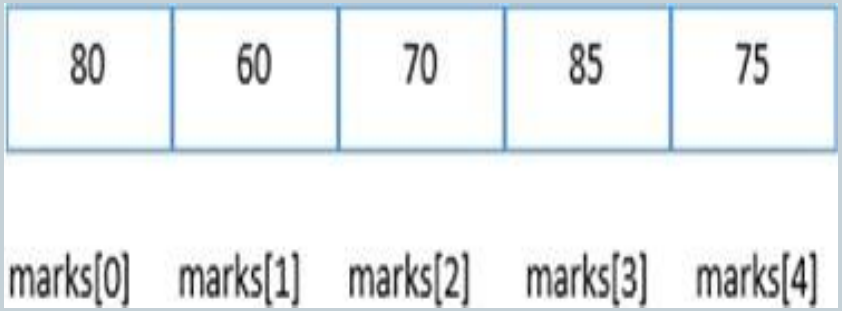
  datatype arr_name [arraysize]={value1, value2, value3,…};

- Example

  int marks[5]={80,60,70,85,75};

- Individual elements of an array can be accessed using the array name and index

```
marks[0]=80; //initialization of array
marks[1]=60;
marks[2]=70;
marks[3]=85;
marks[4]=75;
```



Array Traversal

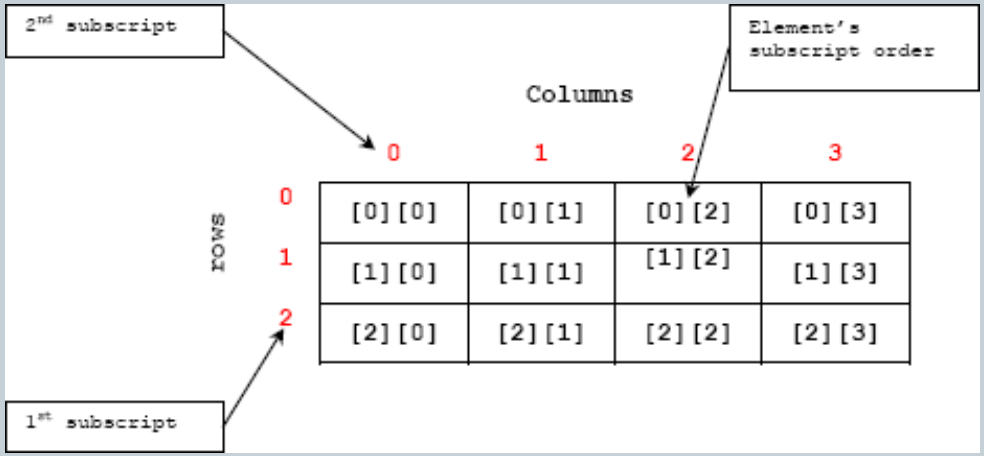
- The elements of an array can be easily accessed using any of the looping structure in C language.
- Using loop structures to traverse arrays helps in code optimization

```
# include <stdio.h>
void main()
{
    int i=0;
    int marks[5]; //Array declaration

    marks[0]=80; //Array initialization
    marks[1]=60;
    marks[2]=70;
    marks[3]=85;
    marks[4]=75;

   //Traversal of array
   for (i=0;i<5;i++)
   {
         printf("Mark %d: %d \n", i+1, marks[i]);
    }

}
```

Multi Dimensional Array

- C supports multidimensional arrays.
- The general form of a multidimensional array declaration-
  datatype arr_name[size1][size2]...[sizeN];
- The simplest form of the multidimensional array is the two-dimensional array.
- A two-dimensional array can be considered as a table which will have x number of rows and y number of columns.
- A two-dimensional array **X**, which contains 3 rows and 4 columns can declared as follows −

  int X[3][4];



Accessing 2-D Array

- An element in a two-dimensional array is accessed by using the subscripts, i.e., row index and column index of the array.
- A two dimensional array can be initialized at the time of declaration-

int matrix[3][3]={{1,2,3},{4,5,6},{7,8,9}};

```
# include <stdio.h>
void main()
{
    int i=0, j=0;
    int matrix[3][3]={{1,2,3},{4,5,6},{7,8,9}};

    //Traversing 2D array
    for(i=0;i<3;i++)
    {
        for(j=0;j<3;j++)
            printf("%d ", matrix[i][j]);
        printf("\n");
    }
}
```

Character array

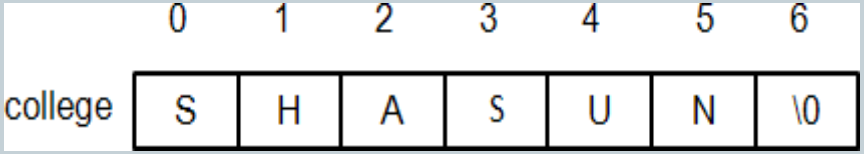
- Strings are actually one-dimensional array of characters terminated by a **null** character '\0' and can be declared as -

  char college[7] = {'S', 'H', 'A', 'S', 'U', 'N','\0'};

- Following the rule of array initialization the above statement can be written as −

  char college[] = "SHASUN";

- The memory presentation of the above defined string is-



- The *null* character need not be placed 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

String functions - <string.h>

Passing arrays to functions

- we can create functions that receives array as argument.
- To pass array in function, we need to write the array name only in the function call.

  function_name(arr_name); //passing array
- The array name refers to the starting address of memory area and is passed as argument.
- There are two ways to declare function that receives array as argument.
  - return_type function_name (type arr_name[])
  - return_type function_name (type arr_name[arrsize) (Optionally, we can define size in subscript notation [])

Passing arrays - Example

An array containing age of 6 persons is passed to a function. This function finds the average age and returns to the main function.

```
# include <stdio.h>
float average(float age[]);
void main()
{
    float avg, age[] = { 32.5, 56, 22.3, 12, 40.5, 24 };
    avg = average(age); /* Only name of array is passed as argument. */
    printf("Average age=%.2f", avg);
}
float average(float age[])
{
    int i;
    float avg, sum = 0;
    for (i = 0; i < 6; ++i)
        sum += age[i];
     avg = (sum / 6);
    return avg;
}
```

Structure

- **structure** is a user defined data type available in C that allows to combine data items of different data types.
- Structures are used to represent a record
- Each element of a structure is called a member.

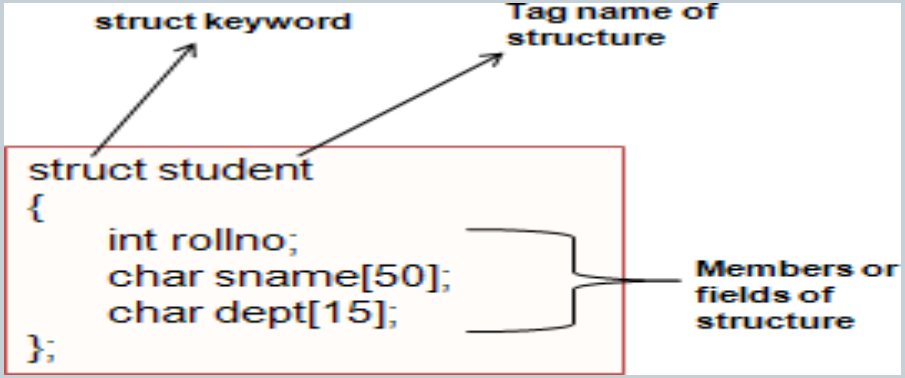
Defining structure

The **struct** keyword is used to define structure.

```
struct  structure_name
{
        data_type member1;
        data_type member2;
           … …
       data_type memeberN;
};
```

Structure..continued

- Example to define structure-



Declaring structure variable

- We can declare variable for the structure, so that we can access the member of structure easily.
- The struct variable for the above structure can be defined as –

  struct student s1, s2;
- To access a member of a structure dot operator is used.

  s1.rollno s1.sname s1.dept

Structure - examples

Passing structure to functions

- A structure can be passed as a function argument in the same way as any other variable.

```
// structure definition
struct student
{
      int rollno;
      char sname[50];
      char dept[15];
};

// Declaration of structure variable
struct student s1;

/*Declaring function using structure as argument */
void print_studdata( struct student t);

/* function definition */
void print_studdata(struct student t)
{
     printf(" Rollno : %d\n", t.rollno);
     printf(" Name : %d\n", t.sname);
     printf(" Dept: %d\n", t.dept);
}
```

Self-referential Structure

- A **self-referential structure** is one in which a member(s) is a pointer to a **structure**(s)
- It is used to create data structures like linked lists, stacks, etc. of the same type
- The general form of self-referential structure is -

```
struct struct_name
    {
        datatype datatypename;
        struct_name * pointer_name;
    };
```

- Example

```
    struct listnode {
        void *data;
        struct listnode *next;
    } linked_list;
```

Union

- Like structure, **Union** is *a* user defined datatype that is used to hold different type of elements. But it doesn't occupy sum of all members size. It occupies the memory of largest member only and shares the memory.
- The memory occupied by a union will be large enough to hold the largest member of the union
- only one member can contain a value at any given time.

Structure	Union

struct Employee{
char x; // size 1 byte
int y; //size 2 byte
float z; //size 4 byte
}e1; //size of e1 = 7 byte
	union Employee{
	char x; // size 1 byte
	int y; //size 2 byte
	float z; //size 4 byte
	}e1; //size of e1 = 4 byte

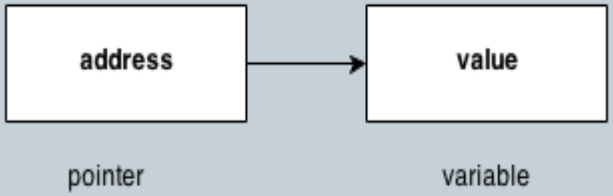
size of e1= 1 + 2 + 4 = 7	**size of e1= 4 (maximum size of 1 element)**

Unit V - Overview

- Pointers
- Pointer Declaration
- Pointer Operations
- Passing pointers to function
- Pointer and arrays
- Array of pointers
- Structure and pointers
- Creation of File
- File Operations

Pointers

- 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.



- Like any variable or constant, a pointer must be declared before using it to store any variable address. The general form of a pointer variable declaration is –

  datatype *var-name;

- Here, **datatype** is the pointer's base type; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * is used to declare a pointer

Pointers..continued

- Some valid pointer declarations are –

  int *ip; /* pointer to an integer */ double *dp; /* pointer to a double */ char *ch /* pointer to a character */

- The **address of** operator '&' returns the address of a variable. By the help of * (**indirection operator**), we can print the value of pointer variable p.

```
#include  <stdio.h>
#include <conio.h>
void main()
{
      int number=100;
      int *p;     //pointer declaration
      p=&number; //stores the address of number variable
      printf("Address of number variable is %x \n",&number);
      printf("Address of p variable is %x \n",p);
      printf("Value of p variable is %d \n",*p);
}
```

Pointer - Features

- 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.
- The size of any pointer is 2 byte (for 16 bit compiler).

Pointer Operations

Operation	Formula	Example

Addition	new_address= current_address + (number * size_of(data type))	int number=77;
		int *p;
		p=&number;
		p=p+4;

Subtraction	new_address = current_address - (number * size_of(data type))	int number=77;
		int *p;
		p=&number;
		p=p-2;

Increment	new_address= current_address + i * size_of(data type)	int number=77;
		int *p;
		p=&number;
		p++;

Decrement	new_address= current_address - i * size_of(data type)	int number=77;
		int *p;
		p=&number;
		p--;

Passing pointers to function

C language allows passing a pointer to a function.

```
# include <stdio.h>
void swap(int *x, int *y); // function declaration
int main ()
{
    int a = 100; int b = 200;
    printf("Before swap, value of a : %d\n", a );
    printf("Before swap, value of b : %d\n", b );
    swap(&a, &b); // fun call to swap the values
    printf("After swap, value of a : %d\n", a );
    printf("After swap, value of b : %d\n", b );
}
/* function definition to swap the values */
void swap (int *x, int *y)
{
     int temp;
     temp = *x;    //save the value of x
     *x = *y;         // copy y into x
     *y = temp; // put temp into y
     return;
 }
```

Now, if we call the function **swap()** by passing actual values, the program produces the following result –

Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100

Passing an argument by reference enable the passed argument to be changed in the calling function by the called function

Pointer and Arrays

- An array name is a constant pointer to the first element of the array.
- Therefore, in the following declaration −

  double salary[10];

- **salary** is a pointer to &salary[0], which is the address of the first element of the array.
- Thus, the following program fragment assigns **p** as the address of the first element of **salary** −

```
double *p;
double salary[10];
p = salary;
```

- It is legal to use array names as constant pointers, and vice versa. Therefore, *(salary + 4) is a legitimate way of accessing the data at salary[4].
- Once we store the address of the first element in 'p', we can access the array elements using *p, *(p+1), *(p+2) and so on.

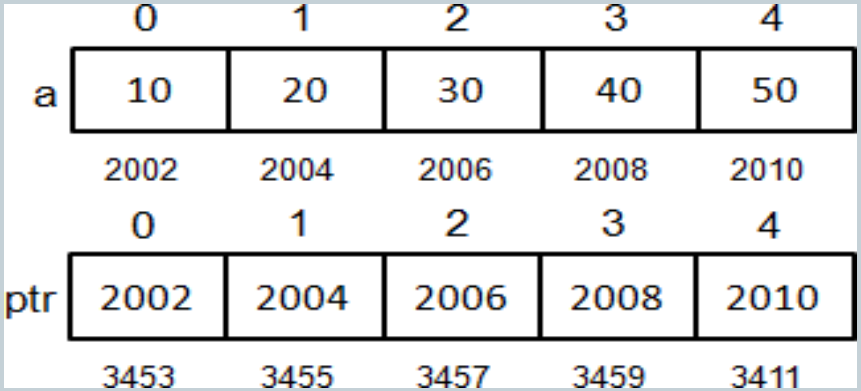
Array of pointers

- Arrays can be defined to hold a number of pointers.
- There may be a situation when we want to maintain an array, which can store pointers to an int or char or any other data type available.
- Following is the declaration of an array of pointers to an integer −

  int *ptr[size];

- It declares **ptr** as an array of integer pointers with specified size. Thus, each element in ptr, holds a pointer to an int value.
- Example:

```
int a[5]={10,20,30,40,50};
int *ptr[5];
ptr[0]=&a[0];
ptr[1]=&a[1];
Ptr[2]=&a[2]
Ptr[3]=&a[3]
```



Structure and arrays

- C structure can be accessed using pointer variable. Arrow (->) operator is used to access the data using pointer variable.

```
# include <stdio.h>
/* structure definition*/
struct student
{
     int id;
     char name[30];
     float percentage;
};
 void main()
{
     int i;
     struct student record1 = {2125, “Roja", 97.5};
     struct student *ptr; //pointer of struct type
      ptr = &record1; //assigning struct address to ptr

/*accessing struct members using ptr */
       printf("Records of STUDENT1: \n");
       printf(" Id is: %d \n", ptr->id);
       printf(" Name is: %s \n", ptr->name);
       printf("  Percentage is: %f \n\n", ptr->percentage);
}
```

Files

- File is a collection of bytes that is stored on secondary storage devices like disk.
- It is used for permanent storage.
- There are two kinds of files in a system. They are,
  - Text files (ASCII)
  - Binary files
- Text files contain ASCII codes of digits, alphabetic and symbols.
- Binary file contains collection of bytes (0's and 1's). Binary files are compiled version of text files.

File handling functions

- In C language, we use a structure pointer of file type to declare a file.

  FILE ***fp**;

- There are 4 basic operations that can be performed on any files-

  Opening/Creating a file

  Closing a file

  Reading a file

  Writing in a file

fopen() & fclose()

- fopen() function is used to open a file to perform operations such as reading, writing etc

  FILE ***fopen*** (const char *filename, const char *mode)

- fopen() function creates a new file if the mentioned file name does not exist.

  FILE *fp;
  fp=**fopen** (“filename”, ”‘mode”);

- Where, fp – file pointer to the data type “FILE”; filename – the actual file name with full path of the file; mode – refers to the operation that will be performed on the file.
- fclose() function closes the file that is being pointed by file pointer fp.

  int **fclose**(FILE *fp);

- In a C program, we close a file as fclose(fp);

File open modes

- A file can be opened in any of the following modes in the fopen() function.

Mode	Description
r	opens a text file in read mode
w	opens a text file in write mode
a	opens a text file in append mode
r+	opens a text file in read and write mode
w+	opens a text file in read and write mode
a+	opens a text file in read and write mode
rb	opens a binary file in read mode
wb	opens a binary file in write mode
ab	opens a binary file in append mode
rb+	opens a binary file in read and write mode
wb+	opens a binary file in read and write mode
ab+	opens a binary file in read and write mode

File i/o functions

fputc() & fgetc() functions

- The fputc() function is used to write a single character into file. It outputs a character to a stream.

```
int fputc(int c, FILE *stream);
```

- The fgetc() function returns a single character from the file. It gets a character from the stream. It returns EOF at the end of file.

```
int fgetc(FILE *stream)
```

fputs() and fgets() functions

- The fputs() function writes a line of characters into file. It outputs string to a stream.

```
int fputs(const char *s, FILE *stream)
```

- The fgets() function reads a line of characters from file. It gets string from a stream.

```
char* fgets(char *s, int n, FILE *stream)
```

File i/o..continued

fprintf() and fscanf() functions

- The fprintf() function is used to write set of characters into file. It sends formatted output to a stream.

  Syntax:

  int fprintf(FILE *stream, const char *format [, argument, ...])

- The fscanf() function is used to read set of characters from file. It reads a word from the file and returns EOF at the end of file.

  Syntax:

  int fscanf(FILE *stream, const char *format [, argument, ...])

File i/o - Example

A file i/o handling example to write and read a sequence of characters to/from the file called “sample.txt”

```
#include <stdio.h>
main()
{
   FILE *fp;
   char buff[255]; //creating char array to store data of file
   fp = fopen(“sample.txt", "w"); //opening file in write mode

   //writing data into file
   fprintf(fp, “Good day. Welcome to Shasun\n");
   fclose(fp); //closing file

   fp = fopen(“sample.txt", "r"); //opening file in read mode

  //reading data from file
   while(fscanf(fp, "%s", buff)!=EOF)
   {
        printf("%s ", buff );
   }
   fclose(fp); //closing file
}
```