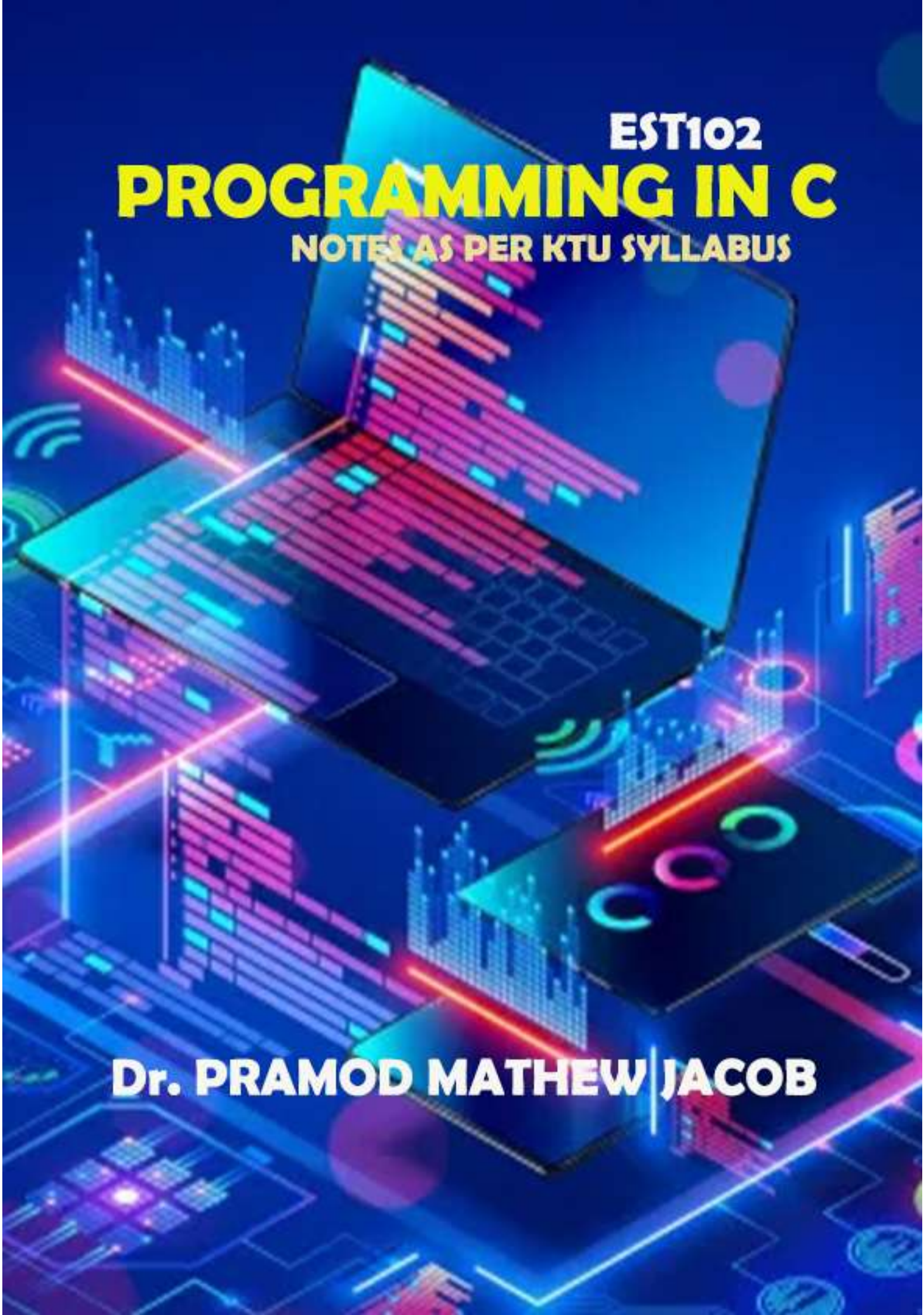

EST102
PROGRAMMING IN C
NOTES AS PER KTU SYLLABUS
Dr. PRAMOD MATHEW JACOB

# PROGRAMMING IN C

Notes as per KTU syllabus


## Dr. PRAMOD MATHEW JACOB

**B.Tech-COMPUTER SCIENCE & ENGINEERING**

**M. Tech-SOFTWARE ENGINEERING**

**PhD-INFORMATION TECHNOLOGY**

# AUTHORS PROFILE

**Pramod Mathew Jacob** has completed his B. Tech in Computer Science & Engineering from Kerala University. He possesses M. Tech in Software Engineering from SRM Institute of Science and Technology Chennai. He achieved his doctorate degree from Vellore Institute of Technology Vellore. Presently he is working as Associate Professor in Providence College of Engineering, Kerala. He has a teaching experience of eight years and Research experience of three years. He has published 18 Scopus indexed papers in various international journals and conferences. He has also published two patents in the area of Internet of Things. He is a professional member of ACM. His areas of interest include Software Engineering, Software Testing, and Internet of Things.

**SYLLABUS** **EST102 PROGRAMMING IN C**

**Module 1** **7 HOURS**

Basics of Computer Architecture: processor, Memory, Input& Output devices Application Software & System software: Compilers, interpreters, High level and low-level languages, Introduction to structured approach to programming, Flow chart Algorithms, Pseudo code (bubble sort, linear search - algorithms and pseudocode

**Module 2** **8 HOURS**

Basic structure of C program: Character set, Tokens, Identifiers in C, Variables and Data Types, Constants, Console IO Operations, printf and scanf Operators and Expressions: Expressions and Arithmetic Operators, Relational and Logical Operators, Conditional operator, size of operator, Assignment operators and Bitwise Operators. Operators Precedence Control Flow Statements: If Statement, Switch Statement, Unconditional Branching using goto statement, While Loop, Do While Loop, For Loop, Break and Continue statements. (Simple programs covering control flow)

**Module 3** **6 HOURS**

Arrays Declaration and Initialization, 1-Dimensional Array, 2-Dimensional Array String processing: In built String handling functions (strlen, strcpy, strcat and strcmp, puts, gets) Linear search program, bubble sort program, simple programs covering arrays and strings

**Module 4** **7 HOURS**

Introduction to modular programming, writing functions, formal parameters, actual parameters Pass by Value, Recursion, Arrays as Function Parameters structure, union, Storage Classes, Scope and life time of variables, simple programs using functions

**Module 5** **7 HOURS**

Basics of Pointer: declaring pointers, accessing data though pointers, NULL pointer, array access using pointers, pass by reference effect File Operations: open, close, read, write, append Sequential access and random access to files: In built file handling functions (rewind() ,fseek(), ftell(), feof(), fread(), fwrite()), simple programs covering pointers and files.

**Text Books**

1. Schaum Series, Gottfried B.S.,Tata McGraw Hill,Programming with C
2. E. Balagurusamy, Mcgraw Hill,Programming in ANSI C
3. Asok N Kamthane, Pearson,Programming in C
4. Anita Goel, Pearson, Computer Fundamentals

**Reference Books**

1. Anita Goel and Ajay Mittal, Pearson, Computer fundamentals and Programming in C
2. Brian W. Kernighan and Dennis M. Ritchie, Pearson, C Programming Language
3. Rajaraman V, PHI, Computer Basics and Programming in C
4. Yashavant P, Kanetkar, BPB Publications, Let us C

# Contents

# MODULE 1

# BASICS OF COMPUTER ARCHITECTURE

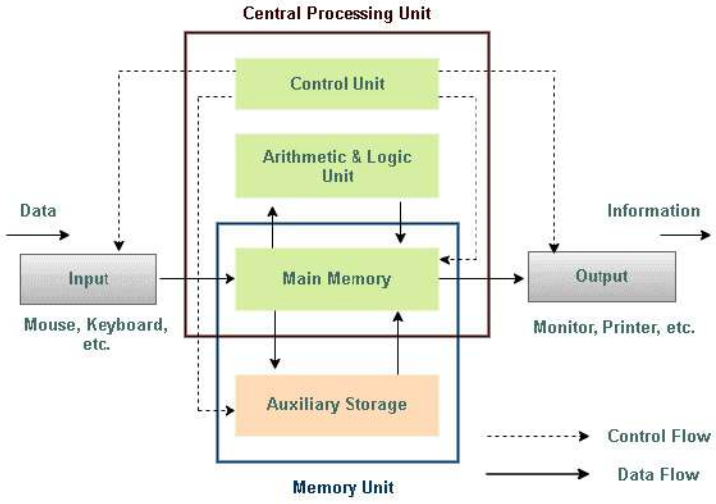
## COMPUTER

Computer is a fast electronic calculating machine that accepts digitized input information, processes it according to a list of internally stored instructions, and produces the resulting output information. The list of instructions is called a computer program, and the internal storage is called computer memory.

COMPUTER ORGANIZATION – Carl Hamacher, Zvonko Vranesic, Satwat Zaky

Computer is an electronic machine, which takes a set of data as input, process it according to the stored instructions and produces suitable output. These stored instructions are called **computer programs**

## BLOCK DIAGRAM OF A DIGITAL COMPUTER / VON-NEUMANN ARCHITECTURE




## COMPONENTS OF DIGITAL COMPUTER

**Input Unit:** The input unit consists of input devices that are attached to the computer. These devices take input and convert it into binary language that the computer understands. Example: keyboard, mouse, microphone, joystick, scanner etc.

**Central Processing Unit (CPU):** The CPU is called the brain of the computer because it is the control center of the computer. It first fetches instructions from memory and then interprets them so as to know what is to be done. Thereafter CPU executes or performs the required computation and then either stores the output or displays on the output device. The CPU has three main components which are responsible for different functions – Arithmetic Logic Unit (ALU), Control Unit (CU) and Memory registers

**Arithmetic and Logic Unit (ALU):** The ALU performs all the arithmetical and logical operations. Arithmetic operations include addition, subtraction, multiplication and division. Logical decisions involve comparison of two data items to see which one is larger or smaller or equal.

**Control Unit:** The Control unit coordinates and controls the data flow in and out of CPU and also controls all the operations of ALU, memory registers and also input/output units.

**Registers:** A register is a temporary unit of memory in the CPU. These are used to store the data which is directly used by the processor.

**Primary Memory:** Memory attached to the CPU is used for storage of data and instructions and is called internal memory The internal memory is also called the Primary memory or Main memory. This memory is also called as RAM, i.e. Random Access Memory. The time of access of data is independent of its location in memory, therefore this memory is also called Random Access memory (RAM). RAM is volatile memory.

**Secondary Memory:** Memory which is used for permanent storage of data. It is non-volatile. Examples: Hard disk drive, SSD, CD, DVD, USB stick etc.

**Output Unit:** The output unit consists of output devices that are attached with the computer. It converts the binary data coming from CPU to human understandable form. The common output devices are monitor, printer, plotter etc.

## WORKING OF DIGITAL COMPUTER

1) The input devices feed data into the computer and is stored in secondary memory.
2) When the data needs to be processed, it is fetched from the secondary memory to the main memory.
3) All these activities are controlled and coordinated by the control unit.
4) The instructions are loaded to the primary memory and ALU performs the processing.
5) The processed results are stored in the secondary storage devices.
6) The results are displayed to the user through output devices.

## PROGRAMMING

**Programming:** The process of developing and implementing various sets of instructions to enable a computer to do a certain task. These instructions are considered computer programs and help the computer to operate smoothly.

**Computer Program:** The set of instructions provided to a computer to perform a particular task. Collection of programs are called software.

**Data vs Information:** Data are raw facts; meaningful data is called information.

## CATEGORIES OF LANGUAGES

### Low Level Language/Machine Language / Binary code / Object code

- Machine language or binary language is the only language that a computer can understand.
- Everything is represented in the form of 0's and 1's (ON / OFF). Simply a collection of binary digits.

### Assembly Language / Mnemonics

- An assembly language is a low-level programming language for microprocessors and other programmable devices.
- An assembly language implements a symbolic representation of the machine code needed to program a given CPU architecture.
- Example: SIC /XE program, 8085 code etc.

**High Level Language (HLL)**

- High-level languages are designed to be used by the human operator or the programmer.
- Every single program written in a high-level language must be interpreted into machine language before being executed by the computer.
- BASIC, C/C++ and Java are popular examples of high-level languages.

## LANGUAGE PROCESSORS

Computer can recognize only the machine language/ Binary language (0s and 1s). But programmers used to write programs in High Level Language (HLL) which is easily understandable to the humans. So, there is a need of a translator which converts the programs written in HLL into an equivalent machine language. The system programs which perform this translation is called **Language Processors**. The different language processors are

**Assembler:** Converts the program written in assembly language into its equivalent machine language. Example: NASM

**Interpreter:** Converts the program written in High Level Language (HLL) into its equivalent machine language by converting and executing each line by line. If an error is there in the HLL code, it reports the error and execution terminates. Execution is resumed only after correcting the error in that line. Example: Python (interpreter), Ruby (interpreter), PHP (interpreter) etc.

**Compiler:** Compiler is similar to Interpreter which translates a High-Level Language into its equivalent machine language. But Compiler compiles a whole program and then it lists out the errors along with its line numbers. If there are no errors in the HLL code, computer generates the object files. The process of translating a HLL into its equivalent machine language is termed as compilation. Example: GCC (GNU Compiler Collection), javac, clang, go compiler etc.

## INTRODUCTION TO STRUCTURED PROGRAMMING

### STEPS IN PROGRAMMING PROCESS

1. Identify the problem / Problem statement / Problem identification
2. Analyse the problem (Analysis)
3. Design a solution
4. Implement / Coding the solution
5. Test the solution (Execution & Testing)
6. Iteration through the phases to refine / correct the program (Debugging)
7. Documentation

### ALGORITHMS

- An algorithm is a well-defined procedure that allows a computer to solve a problem.
- Step by step procedure to solve a particular problem.
- A particular problem can typically be solved by more than one algorithm.
- Optimization is the process of finding the most efficient algorithm for a given task.
- An algorithm usually has a Start/Begin and a Stop/End.

**Example 1: Write an algorithm to add two integers.**

Step 1: Start
Step 2: Input two integers as int1 and int2
Step 3: Compute sum=int1+int2
Step 4: Display sum
Step 5: Stop

**Example 2: Write an algorithm to find the area and perimeter of a rectangle.**

Step 1: Start
Step 2: Input length and breadth of the rectangle as l and b
Step 3: Compute area=l*b
Step 4: Compute perimeter=2*(l+b)
Step 5: Display area and perimeter
Step 6: Stop

**PSEUDO CODE**

It is a simpler version of a programming code in plain English which uses short phrases to write code for a program before it is implemented in a specific programming language.

**ALGORITHM & PSEUDO CODE FOR LINEAR SEARCH**

**Algorithm for Linear search**

1. Start
2. Define an array to store integers as arr[] and initialize a counter 'flag' as 0
3. Read array size and array elements.
4. Read the element to be searched as 'x'
5. Start from the leftmost element of arr[] and one by one compare 'x' with each element of arr[].
6. If 'x' matches with an element, increment flag by one
7. If flag==0, display 'ELEMENT NOT PRESENT'
8. If flag>0, display 'ELEMENT PRESENT'.
9. Stop.

**Pseudo code for Linear search**

```
FUNCTION linearSearch(array, searchTerm):
INITIALIZE flag=0
    FOR index FROM 0 -> length(array):
      IF array[index] == searchTerm THEN INCREMENT flag BY 1
      ENDIF
      ENDLOOP
      IF flag==0, THEN DISPLAY 'ELEMENT NOT PRESENT'
      ELSE DISPLAY 'ELEMENT PRESENT'.
END FUNCTION
```

**ALGORITHM & PSEUDO CODE FOR BUBBLE SORT**

**Algorithm for Bubble Sort**

1. Start
2. Define an array to store integers as arr[].
3. Read array size as 'n' and then read array elements.
4. Repeat step 5 for all array elements
5. Compare if arr[index]>array[index+1], if true swap array[index] and arr[index+1].
6. Stop

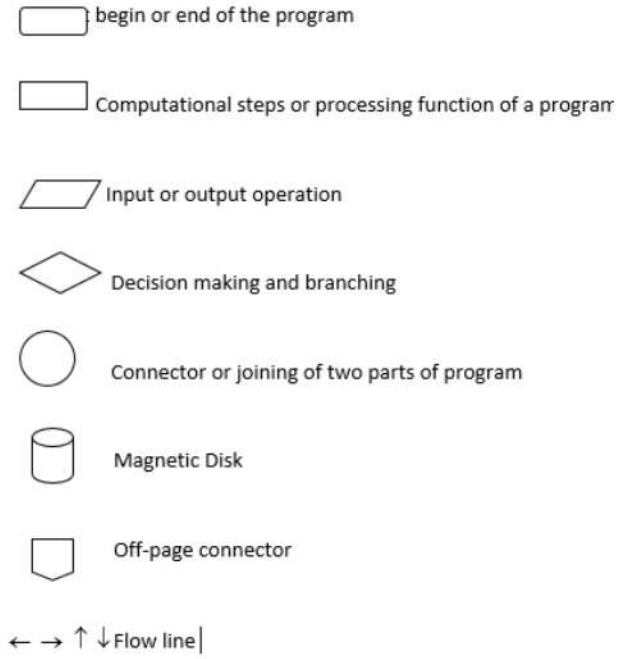
**Pseudo code for Bubble sort**

```
FUNCTION bubblesort(arr[],n)
REPEAT FOR i=0 TO i<n
        REPEAT FOR j=0 TO j<n-i-1
                IF arr[j]>arr[j+1]
                        swap(arr[j],arr[j+1)
                END IF
        END FOR
END FOR
END FUNCTION
```

**FLOW CHART**

- A flowchart is a diagrammatic representation that illustrates the sequence of operations to be performed to get the solution of a problem.
- Pictorial representation of an algorithm

**SYMBOLS USED IN FLOW CHART**

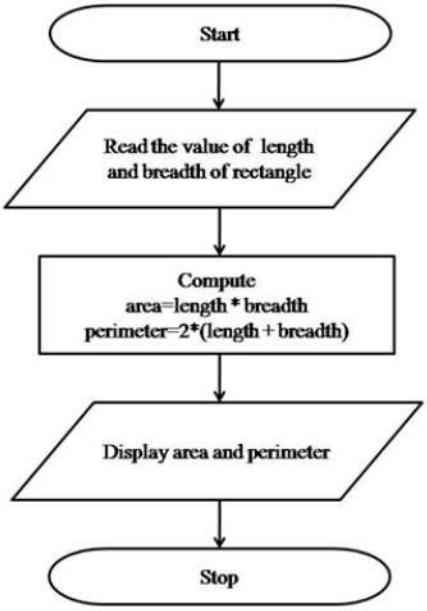



**ADVANTAGES OF USING FLOWCHARTS**

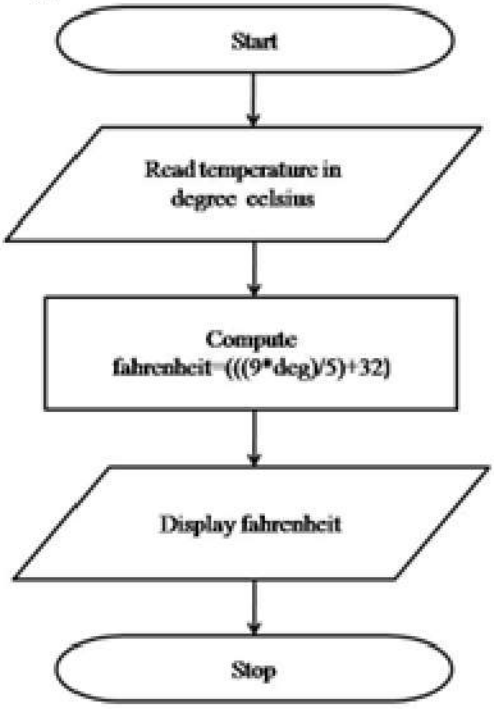
- **Communication:** Flowcharts are better way of communicating the logic of a system to all concerned.
- **Effective analysis:** With the help of flowchart, problem can be analysed in more effective way.
- **Proper documentation:** Program flowcharts serve as a good program documentation, which is needed for various purposes.
- **Efficient Coding:** The flowcharts act as a guide or blueprint during the systems analysis and program development phase.
- **Proper Debugging:** The flowchart helps in debugging process.
- **Efficient Program Maintenance:** The maintenance of operating program becomes easy with the help of flowchart. It helps the programmer to put efforts more efficiently on that part.

**FLOWCHART EXAMPLES**

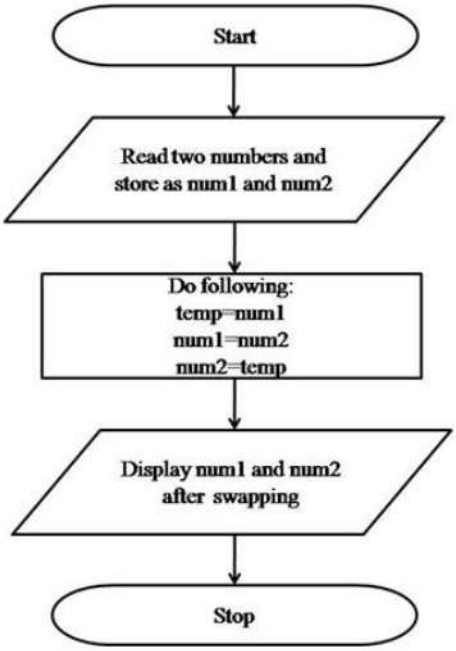
1. **Area and Perimeter of Rectangle**



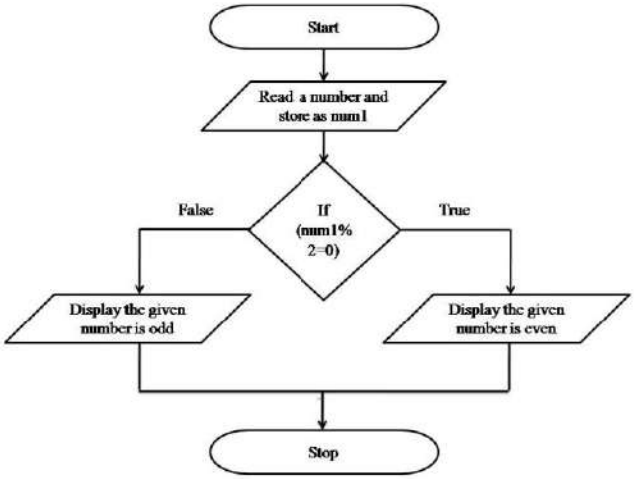

2. **Temperature conversion from Degree to Fahrenheit**



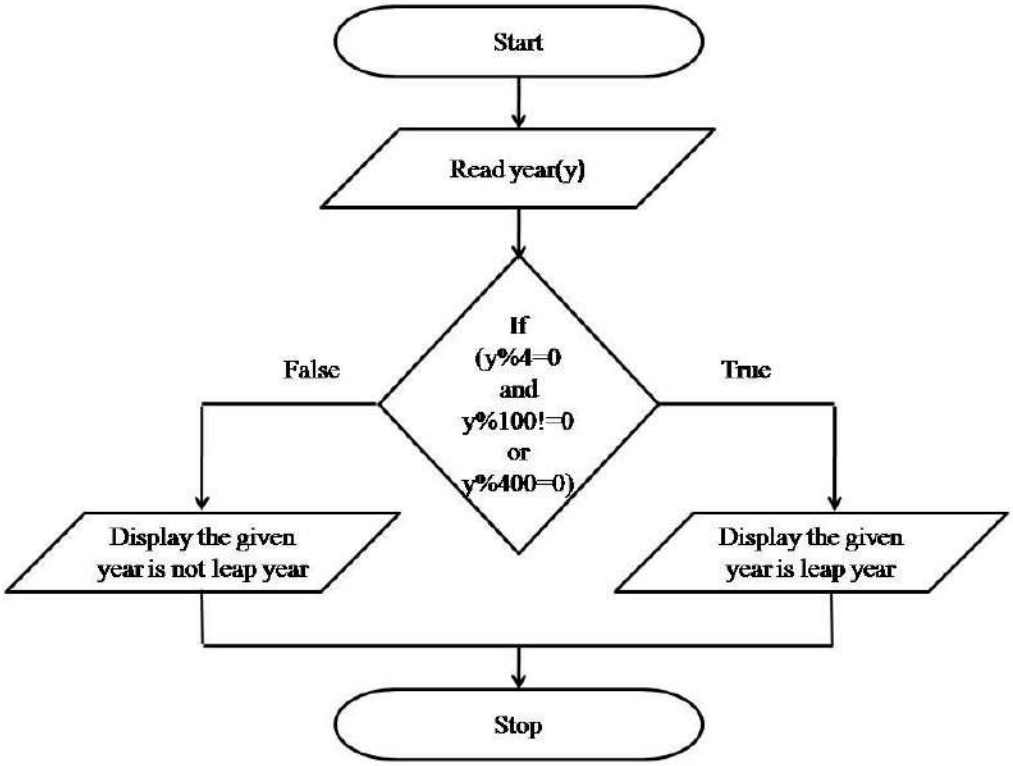

3. **Swap two numbers**



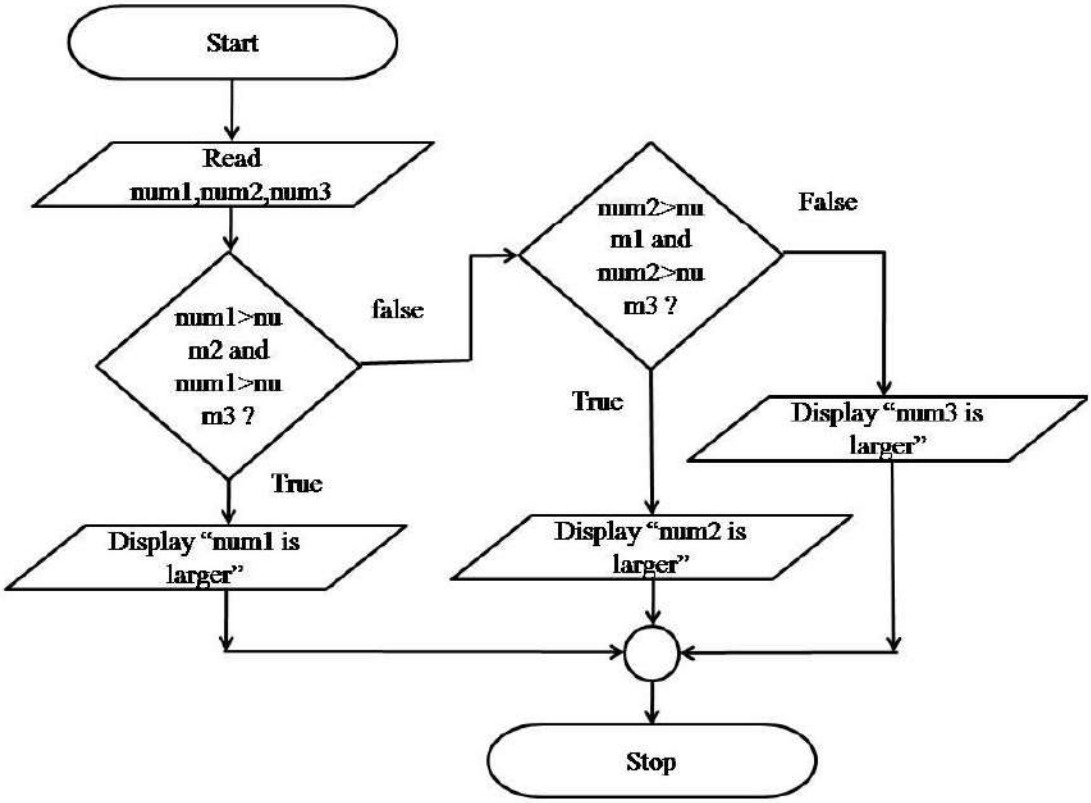

4. **Odd or Even**



**5. Leap year or not**




**6. Largest of three numbers**



# MODULE 2

# BASIC STRUCTURE OF C PROGRAM

## INTRODUCTION TO C PROGRAMMING

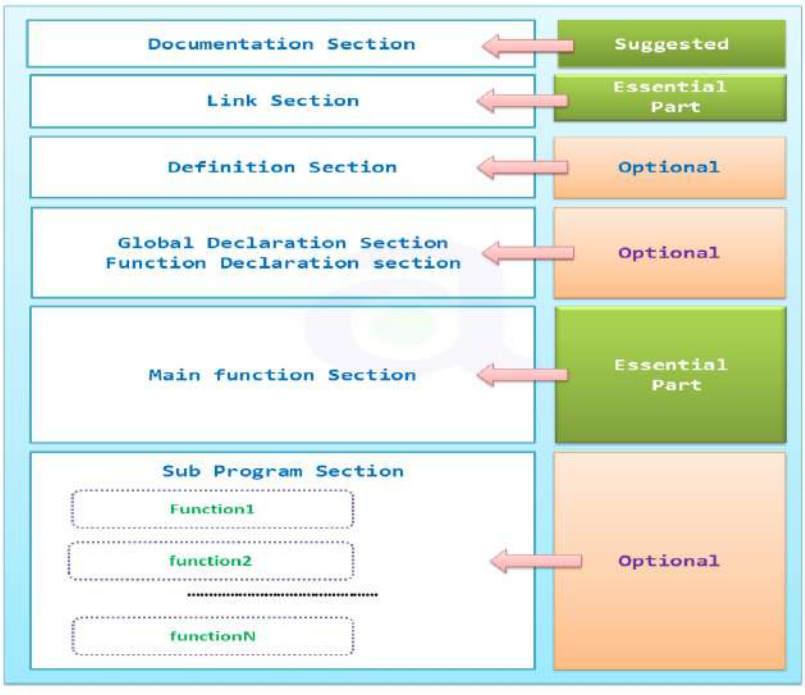
### HISTORY OF C

C is programming language developed by Dennis Ritchie at the Bell Laboratories in 1972. C language is standardized in 1989 by ANSI (American National Standards Institute) known as ANSI C. International standard Organization (ISO) in 1990 which was adopted by ANSI and is known as C89. As part of the normal evolution process the standard was updated in 1995 (C95) and 1999 (C99).

### FEATURES OF C

- It is a robust language with a rich set of built-in functions and operators to write any complex programs.
- Programs written in C are efficient and fast.
- C is highly portable.
- C has the ability to extend itself.
- C language is well suited for structured programming.
- C is case sensitive.

### BASIC STRUCTURE OF C PROGRAM




**Pre-processor directive**: Preprocessor directives are the type of macros and a phase before compilation takes place. It can be said that these are some set of instructions given to compiler to perform actual compilation. Example: #include, #define etc.

**Tokens:** Tokens are the smallest individual units of a program. C has 6 types of tokens.

**Keywords:** Keywords are the reserved words used in programming. Each keyword has fixed meaning and that cannot be changed by user. C language has 32 keywords. (Refer figure below for keywords in C).

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

**Identifiers:** Identifiers are names given to C entities, such as variables, functions, structures etc. Identifier are created to give unique name to C entities to identify it during the execution of program.

Eg: int sum; Here sum is an identifier of type int.

### Rules for naming identifiers

1. First character must be an alphabet (or underscore) followed by letters or digits
2. Must consist of only letters, digits or underscore.
3. Only first 31 characters are significant.
4. Cannot use a keyword.
5. Must not contain white space.

**Variables:** Variables are memory location in computer's memory to store data. To indicate the memory location, each variable should be given a unique name called identifier. Variable names are just the symbolic representation of a memory location. Examples of variable name: sum, car_no, count etc.

**Constants:** Constants are the terms that can't be changed during the execution of a program. For example: 1, 2.5, "Programming is easy." etc. C constants can be classified as

[1] Integer constants.: Integer constants are the numeric constants (constant associated with number) without any fractional part or exponential part. Example: Decimal constants: 0, -9, 22 etc
[2] Octal constants: 021, 077, 033 etc
[3] Hexadecimal constants: 0x7f, 0x2a, 0x521 etc
[4] Floating-point constants: Floating point constants are the numeric constants that has either fractional form or exponent form. For example: -2.0, 0.0000234, -0.22E-5
[5] Character constants: Character constants are the constant which use single quotation around characters. For example: 'a', 'l', 'm', 'F' etc.
[6] String constants: String constants are the constants which are enclosed in a pair of double-quote marks. For example:

```
"good"              //string constant
""                  //null string constant
"      "            //string constant of six white space
"x"                 //string constant having single character.
"Earth is round\n"  //prints string with newline
```

**Escape Sequences:** Sometimes, it is necessary to use newline(enter), tab, quotation mark etc. in the program which either cannot be typed or has special meaning in C programming. In such cases, escape sequence is used. For example: \n is used for newline. The backslash ( \ ) causes "escape" from the normal way the characters are interpreted by the compiler.

| Symbol | Character code (dec / hex / oct) | Function |
|---|---|---|
| \a | 7 / 0x07 / 007 | Produces a beep sound |
| \b | 8 / 0x08 / 010 | Move the cursor one character to the left |
| \t | 9 / 0x09 / 011 | Moves the cursor to the next TAB position |
| \n | 10 / 0x0A / 012 | Moves the cursor to the beginning of the next line |
| \v | 11 / 0x0B / 013 | Moves the cursor to the top left corner of the screen |
| \f | 12 / 0x0C / 014 | Clears the screen |
| \r | 13 / 0x0D / 015 | Moves the cursor to the beginning of the current line |

**Enumeration constants:** Keyword enum is used to declare enumeration types. For example: enum color {yellow, green, black, white}; Here, the variable name is color and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively by default.

## DATATYPES

In C, variable(data) should be declared before it can be used in program. Data types are the keywords, which are used for assigning a type to a variable. Data types in C are

1. **Fundamental / Primitive Data Types**
   - Integer types
   - Floating Type
   - Character types
   - Double
   - void
2. **Derived Data Types**
   - Arrays
   - Pointers
   - Function
3. **User defined data types**
   - Enumeration
   - Structure
   - Union

| Variable Type | Keyword | Bytes Required | Range | Format |
|---|---|---|---|---|
| Character (signed) | Char | 1 | -128 to +127 | %c |
| Integer (signed) | Int | 2 | -32768 to +32767 | %d |
| Float (signed) | Float | 4 | -3.4e38 to +3.4e38 | %f |
| Double | Double | 8 | -1.7e308 to + 1.7e308 | %lf |
| Long integer (signed) | Long | 4 | 2,147,483,648 to 2,147,438,647 | %ld |
| Character (unsigned) | Unsigned char | 1 | 0 to 255 | %c |
| Integer (unsigned) | Unsigned int | 2 | 0 to 65535 | %u |
| Unsigned long integer | unsigned long | 4 | 0 to 4,294,967,295 | %lu |
| Long double | Long double | 10 | -1.7e932 to +1.7e932 | %Lf |

## OPERATORS IN C

C has a rich set of operators including Arithmetic operators, Logical operators, Relational operators, Increment/Decrement operators, Bitwise operators, Assignment operators, Conditional operator and Special operator.

### Arithmetic operators and Increment/Decrement operators

The following table shows all the arithmetic operators supported by the C language. Assume variable **A** holds 10 and variable **B** holds 20 then see the example section.

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands. | A + B = 30 |
| − | Subtracts second operand from the first. | A − B = -10 |
| * | Multiplies both operands. | A * B = 200 |
| / | Divides numerator by de-numerator. | B / A = 2 |
| % | Modulus Operator and remainder of after an integer division. | B % A = 0 |
| ++ | Increment operator increases the integer value by one. | A++ = 11 |
| -- | Decrement operator decreases the integer value by one. | A-- = 9 |

### Logical operators

Following table shows all the logical operators supported by C language. Assume variable **A** holds 1 and variable **B** holds 0, then see example section.

| Operator | Description | Example |
|---|---|---|
| && | Called Logical AND operator. If both the operands are non-zero, then the condition becomes true. | (A && B) is false. |
| \|\| | Called Logical OR Operator. If any of the two operands is non-zero, then the condition becomes true. | (A \|\| B) is true. |
| ! | Called Logical NOT Operator. It is used to reverse the logical state of its operand. If a condition is true, then Logical NOT operator will make it false. | !(A && B) is true. |

**Relational operators**

The following table shows all the relational operators supported by C. Assume variable **A** holds 10 and variable **B** holds 20 then see the example section.

| Operator | Description | Example |
|---|---|---|
| == | Checks if the values of two operands are equal or not. If yes, then the condition becomes true. | (A == B) is not true. |
| != | Checks if the values of two operands are equal or not. If the values are not equal, then the condition becomes true. | (A != B) is true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. | (A > B) is not true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. | (A < B) is true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. | (A >= B) is not true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. | (A <= B) is true. |

**Conditional operator and Special operators**

| Operator | Description | Example |
|---|---|---|
| sizeof() | Returns the size of a variable. | sizeof(a), where a is integer, will return 4. |
| & | Returns the address of a variable. | &a; returns the actual address of the variable. |
| * | Pointer to a variable. | *a; |
| ? : | Conditional Expression. | If Condition is true ? then value X : otherwise value Y |

**Bitwise operators**

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ is as follows –

| p | q | p & q | p \| q | p ^ q |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

Assume A = 60 and B = 13 in binary format, they will be as follows –

A = 0011 1100
B = 0000 1101
---
A&B = 0000 1100
A|B = 0011 1101
A^B = 0011 0001
~A = 1100 0011

The following table lists the bitwise operators supported by C. Assume variable 'A' holds 60 and variable 'B' holds 13, then see the example section.

| Operator | Description | Example |
|---|---|---|
| & | Binary AND Operator copies a bit to the result if it exists in both operands. | (A & B) = 12, i.e., 0000 1100 |
| \| | Binary OR Operator copies a bit if it exists in either operand. | (A \| B) = 61, i.e., 0011 1101 |
| ^ | Binary XOR Operator copies the bit if it is set in one operand but not both. | (A ^ B) = 49, i.e., 0011 0001 |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. | (~A ) = -60, i.e,. 1100 0100 in 2's complement form. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. | A << 2 = 240 i.e., 1111 0000 |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. | A >> 2 = 15 i.e., 0000 1111 |

**Assignment operators**

The following table lists the assignment operators supported by the C language –

| Operator | Description | Example |
|---|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand | C = A + B will assign the value of A + B to C |
| += | Add AND assignment operator. It adds the right operand to the left operand and assign the result to the left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator. It subtracts the right operand from the left operand and assigns the result to the left operand. | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator. It multiplies the right operand with the left operand and assigns the result to the left operand. | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator. It divides the left operand with the right operand and assigns the result to the left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator. It takes modulus using two operands and assigns the result to the left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | Bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| \|= | Bitwise inclusive OR and assignment operator. | C \|= 2 is same as C = C \| 2 |

**OPERATORS ASSOCIATIVITY AND PRECEDENCE IN C**

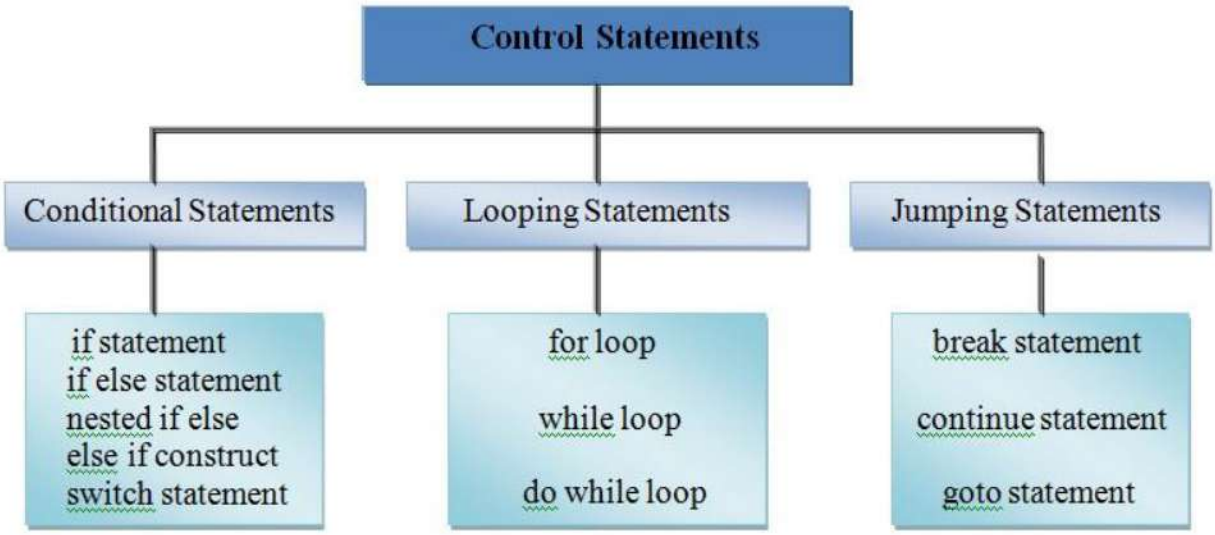
- Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.
- For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.
- Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

| Category | Operator | Associativity |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |

**INPUT & OUTPUT FUNCTIONS IN C**

- **Input** means to provide the program with some data to be used in the program and **Output** means to display data on screen or write the data to a printer or a file.
- C programming language provides many built-in functions to read any given input and to display data on screen when there is a need to output the result.
- **printf() and scanf() functions** - The standard input-output header file, named stdio.h contains the definition of the functions printf() and scanf(), which are used to display output on screen and to take input from user respectively.
- **getchar() and putchar() functions** - The getchar() function reads a character from the terminal and returns it as an integer. This function reads only single character at a time. You can use this method in a loop in case you want to read more than one character. The putchar() function displays the character passed to it on the screen and returns the same character. This function too displays only a single character at a time. In case you want to display more than one characters, use putchar() method in a loop.
- **gets() and puts() functions** - The gets() function reads a line from **stdin**(standard input) into the buffer pointed to by str pointer, until either a terminating newline or EOF (end of file) occurs. The puts() function writes the string str and a trailing newline to **stdout**.
- **Difference between scanf() and gets() :** The main difference between these two functions is that scanf() stops reading characters when it encounters a space, but gets() reads space as character too.

## CONTROL STATEMENTS IN C




## CONDITIONAL STATEMENTS / SELECTION STATEMENTS

**Syntax for If**

```
if(condition)
{
Action1;
}
Action 2;
Action 3;
```

**Syntax for If – else**

```
if(condition)
{
True statement;
}
else
{
False statement;
}
```

**Syntax for if – else if - else**

```
if (condition)
{
Action 1;
}
else if(condition)
{
Action 2;
}
else
{
Action 3
}
```

## LOOPING STATEMENTS / ITERATONS

### ITERATIONS

- Statements used for repeatedly performing some actions.
- There are 3 iterative statements in C.
    1. while loop (entry controlled)
    2. do-while loop (exit controlled)
    3. for loop (entry controlled)

**Syntax for while loop**

```
while(condition)
{
Actions to be repeated;    // if condition is true
}
```

**Syntax for do –while loop**

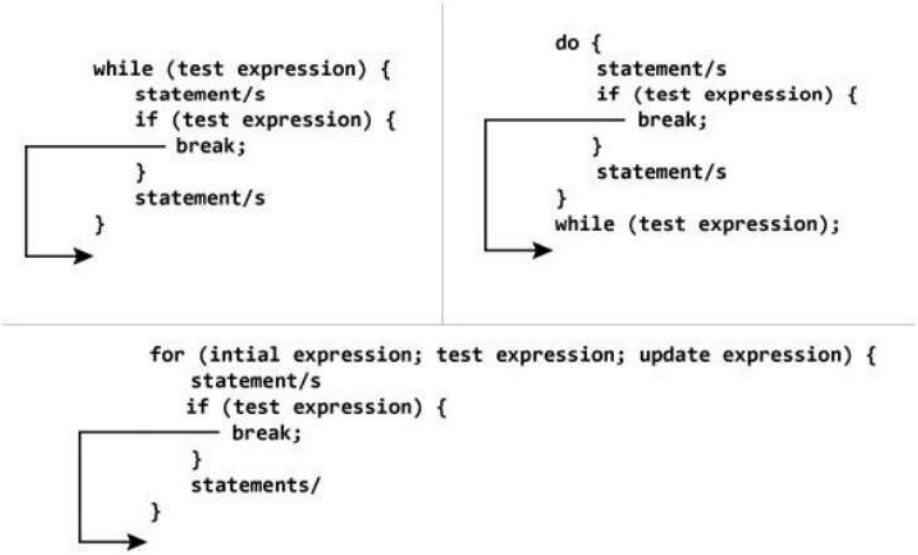
```
do
{
Actions to be repeated;
}
while(condition);
```

**Syntax for for loop**

```
for(initial value;test condition;increment/decrement)
{
Actions to be repeated;
}
```

**break statement**

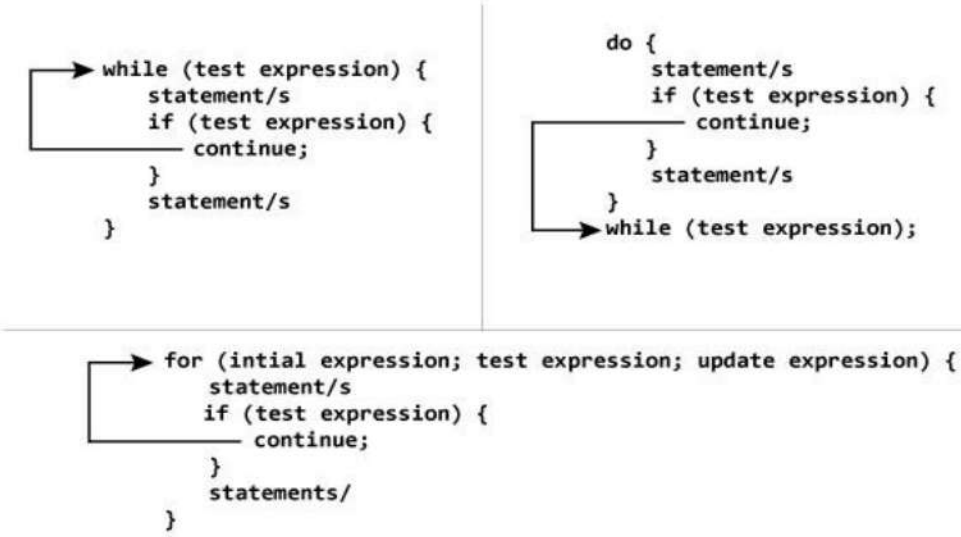
The break; statement terminates a loop (for, while and do..while loop) and a switch statement immediately when it appears.




NOTE: The break statment may also be used inside body of else statement.

**continue statement**

It is sometimes necessary to skip a certain test condition within a loop. In such case, continue; statement is used in C++ programming.




NOTE: The continue statment may also be used inside body of else statement.

**WHILE AND DO- WHILE LOOP COMPARISONS**

| While | Do- while |
|---|---|
| Entry controlled loop | Exit controlled loop |
| Minimum number of executions is zero | Minimum number of executions is one |
| Syntax:<br>while(condition)<br>{<br>Actions to be repeated<br>} | Syntax:<br>do<br>{<br>Actions to be repeated<br>}<br>while(condition); |

**GOTO AND LABEL**

- A **goto** statement in C programming provides an unconditional jump from the 'goto' to a labeled statement in the same function.
- **NOTE** – Use of **goto** statement is highly discouraged in any programming language because it makes difficult to trace the control flow of a program, making the program hard to understand and hard to modify. Any program that uses a goto can be rewritten to avoid them.
- The syntax for a **goto** statement in C is as follows –

  goto label;

  ..

  .

  label: statement;

- Here **label** can be any plain text except C keyword and it can be set anywhere in the C program above or below to **goto** statement.

**PROGRAMMING EXAMPLES**

**1. C program to add two integers.**

```
#include<stdio.h>
int main()
{
int num1,num2,sum;
printf("Enter two numbers to add:\n");
scanf("%d%d",&num1,&num2);
sum=num1+num2;
printf("Sum=%d",sum);
return 0;
}
```

2. **C program to compute the area and perimeter of rectangle.**

```
#include<stdio.h>
int main()
{
float length,breadth,area, perimeter;
printf("Enter length and breadth of a rectangle:\n");
scanf("%f%f",&length,&breadth);
area=length*breadth;
perimeter=2*(length+breadth);
printf("Area=%f\nPerimeter=%f\n",area,perimeter);
return 0;
}
```

**3. C program to compute the area and circumference of circle.**

```
#include<stdio.h>
int main()
{
float radius,area,circumference;
printf("Enter the radius of the circle:\n");
scanf("%f",&radius);
area=3.14*radius*radius;
circumference=2*3.14*radius;
printf("Area=%f\nCircumference=%f\n",area,circumference);
return 0;
}
```

**4. C program to compute the quotient and reminder of two integers.**

```
#include<stdio.h>
int main()
{
int num1,num2,quotient,     reminder;
printf("Enter two numbers:\n");
scanf("%d%d",&num1,&num2);
quotient=num1 / num2;
reminder= num1 % num2;
printf("Quotient=%d\nReminder=%d\n",quotient,reminder);
return 0;
}
```

**5. C program to swap two numbers using temporary variable.**

```
#include<stdio.h>
int main()
{
int num1,num2,temp;
printf("Enter two numbers:\n");
scanf("%d%d",&num1,&num2);
printf("Numbers before swapping:\n");
printf("num1=%d\nnum2=%d\n",num1,num2);
temp=num1;
num1=num2;
num2=temp;
printf("Numbers after  swapping:\n");
printf("num1=%d\nnum2=%d\n",num1,num2);
return 0;
}
```

**6. C program to swap two numbers without using third variable.**

```
#include<stdio.h>
int main()
{
int num1,num2,temp;
printf("Enter two numbers:\n");
scanf("%d%d",&num1,&num2);
printf("Numbers before swapping:\n");
printf("num1=%d\nnum2=%d\n",num1,num2);
num1=num1+num2;
num2=num1-num2;
num1=num1-num2;
printf("Numbers after  swapping:\n");
printf("num1=%d\nnum2=%d\n",num1,num2);
return 0;
}
```

7. **C program to check whether an inputted number is odd or even.**

```
#include<stdio.h>
int main()
{
int num;
printf("Enter a number:\n");
scanf("%d",&num);
if(num%2==0)
{
printf("Given number %d is even\n ",num);
}
else
{
printf("Given number %d is odd\n ",num);
}
return 0;
}
```

**8. C program to find the largest of two numbers.**

```
#include<stdio.h>
int main()
{
int num1,num2;
printf("Enter two numbers:\n");
scanf("%d%d",&num1,&num2);
if(num1>num2)
{
printf("Largest element=%d\n ",num1);
}
else
{
printf("Largest element= %d \n ",num2);
}
return 0;
}
```

**9. C program to find the biggest of three integers.**

```
#include<stdio.h>
int main()
{
int num1,num2,num3;
printf("Enter three numbers:\n");
scanf("%d%d%d",&num1,&num2,&num3);
if((num1>num2)&&(num1>num3))
{
printf("Largest element=%d\n ",num1);
}
else if((num2>num1)&&(num2>num3))
{
printf("Largest element=%d\n ",num2);
}
else if((num3>num1)&&(num3>num2))
{
printf("Largest element=%d\n ",num3);
}
else
        {
        printf("Three numbers are equal");
        }
return 0;
}
```

10. **Write a C program to print first 100 natural numbers.**

```
#include<stdio.h>
int main()
{
int i=1;
while(i<=100)
{
printf("%d\n",i);
i=i+1;
}
return 0;
}
```

**11. C program to find the sum of first N numbers using do -while.**

```
#include<stdio.h>
int main()
{
int n,i=1,sum=0;
printf("Enter the value of N");
scanf("%d",&n);
do
{
sum=sum+I;
i=i+1;
}while(i<=n);
printf("Sum=%d",sum);
return 0;
}
```

**12. C program to find the factorial of a number.**

```
#include<stdio.h>
int main()
{
int n,i=1,fact=1;
printf("Enter the number");
scanf("%d",&n);
for(i=1;i<=n;i++)
{
fact=fact*i;
}
printf("Factorial of %d = %d", n,fact);
return 0;
}
```

**13. C program to print the days of a week.**

```
#include<stdio.h>
Int main()
{
int n;
printf("Enter the number between 1 and 7");
scanf("%d",&n);
switch(n)
{
case 1: printf("Sunday"); break;
case 2: printf("Monday"); break;
case 3: printf("Tuesday"); break;
case 4: printf("Wednesday"); break;
case 5: printf("Thursday"); break;
case 6: printf("Friday"); break;
case 7: printf("Saturday"); break;
default: printf("Invalid Input");
}
return 0;
}
```

**14. C program to check leap year or not**

```
#include<stdio.h>
int main()
{
int year;
printf("Enter the year:\n");
scanf("%d",&year);
if((year%400==0)
{
printf("Given year %d is leap year\n ",year);
}
else if(year%100==0)
{
printf("Given year %d is not leap year\n ",year);
}
else if(year%4==0)
{
printf("Given year %d is leap year\n ",year);
}
else
{
printf("Given year %d is not leap year\n ",year);
}
return 0;
}
```

**15. C program to find area of triangle when three sides are given**

```
#include<stdio.h>
#include<math.h>
int main()
{
int side1,side2,side3,s,area;
printf("Enter three sides of a triangle:\n");
scanf("%d%d%d",&side1,&side2,&side3);
s=((side1+side2+side3)/2) ;
area=sqrt(s*(s-a)*(s-b)*(s-c));
printf("Area of a triangle=%d\n",area);
return 0;
}
```

**16. C program to check whether a given character is vowel or not.**

```
#include<stdio.h>
int main()
{
char c;
printf("Enter a character:\n ");
scanf("%c",&c);
if(c=='a'||c=='A'||c=='e'||c=='E'||c=='i'||c=='I'||c=='o'||c=='O'||c=='u'||c=='U')
printf("%c is vowel",c);
else
printf("%c is not  vowel",c);
return 0;
}
```

**17. C program to print factors of a number**

```
#include<stdio.h>
int main()
{
int num,i,count=0;
printf("\nEnter number");
scanf("%d",&num);
printf("\nFactors are");
for(i=1;i<=num;i++)
{
        if(num%i==0)
        {
                printf("%d\t",i);
                count++;
        }
}
printf("\nThe number of factors are %d",count);
return 0;
}
```

**18. C program to print sum of digits of a number**

```
#include<stdio.h>
int main()
{
int num,digit,sum=0;
printf("\nEnter number");
scanf("%d",&num);
while(num>0)
{
digit=num%10;
sum=sum+digit;
num=num/10;
}
printf("\nSum of digits =%d",sum);
return 0;
}
```

**19. C program to print reverse of a number and check whether the number is palindrome or not**

```
#include<stdio.h>
int main()
{
int num,digit,rev=0,temp;
printf("\nEnter number");
scanf("%d",&num);
temp=num;
while(num>0)
{
digit=num%10;
rev=(rev*10)+digit;
num=num/10;
}
printf("\nReverse =%d",rev);
if(rev==temp)printf("\nPalindrome number");
else printf("\nNot palindrome number");
return 0;
}
```

**20. C program to check whether a three digit number is Armstrong or not.**

```
#include<stdio.h>
#include<math.h>
int main()
{
int num,digit,sum=0,temp;
printf("\nEnter number");
scanf("%d",&num);
temp=num;
while(num>0)
{
digit=num%10;
sum=sum+pow(digit,3);
num=num/10;
}
if(sum==temp)printf("\nArmstrong number");
else printf("\nNot Armstrong number");
return 0;
}
```

**21. C program to check whether a given number is prime or not.**

```
#include<stdio.h>
#include<math.h>
int main()
{
int num,i,flag=0;
printf("\nEnter number");
scanf("%d",&num);
for(i=2;i<=num/2;i++)
{
        if(num%i==0)
        {
        flag++;
        break;
        }
}
if(flag==0)printf("\nPrime number");
else printf("\nNot prime number");
return 0;
}
```

**22. C program to display roots of quadratic equation**

```
#include<stdio.h>
#include<math.h>
int main ()
{
  float a,b,c,r1,r2,d;
  printf("Enter the values of a b c”);
  scanf ("%f %f %f", &a, &b, &c);
  d= b*b – 4*a*c;
  if (d>0)
  {

    r1 = -b+sqrt (d)/(2*a);
    r2 = -b-sqrt (d)/(2*a);
```

```
    printf("The real roots = %f %f", r1, r2);
  }
  else if (d==0)
  {
    r1 = -b/(2*a);
    r2 = -b/(2*a);
    printf("roots are equal =%f %f", r1, r2);
  }
  else
    printf("Roots are imaginary");
  return 0;
}
```

**23. C program to print Fibonacci series**

```
#include<stdio.h>
#include<math.h>
int main ()
{
int first=0,second=1,next;
int term,i;
printf("\nEnter number of terms to be printed");
scanf("%d",&term);
printf("\nThe fibonacci series is 0\t1\t");
for(i=0;i<term-2;i++)
{
        next=first+second;
        printf("%d\t",next);
        first=second;
        second=next;
}
  return 0;
}
```

**24. C program to print pyramid of numbers as follows:**

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

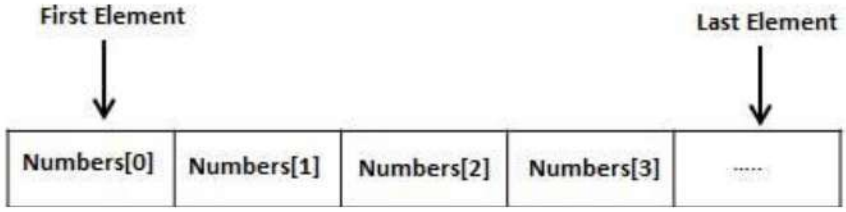
```
#include <stdio.h>
int main()
{
  int i, j, rows;
  printf("Enter the number of rows: ");
  scanf("%d", &rows);
  for (i = 1; i <= rows;i++)
  {
    for (j = 1; j <= i;j++)
    {
      printf("%d ", j);
     }
    printf("\n");
  }
  return 0;
}
```

# MODULE 3

# ARRAYS & STRINGS

## ARRAYS

- Arrays a kind of data structure that can store a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
- Instead of declaring individual variables, such as number0, number1, ..., and number99, you declare one array variable such as numbers and use numbers[0], numbers[1], and ..., numbers[99] to represent individual variables. A specific element in an array is accessed by an index.
- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.




## DECLARING AN ARRAY

- To declare an array in C, a programmer specifies the type of the elements and the number of elements required by an array as follows –

  datatype arrayName [ arraySize ];

- This is called a *single-dimensional* array. The **arraySize** must be an integer constant greater than zero and **type** can be any valid C data type. For example, to declare a 10-element array called **balance** of type int, use this statement –

  int balance[10];

  Here, *balance* is a variable array which is sufficient to hold up to 10 integer numbers.

## INITIALIZING AN ARRAY

- An array can be initialized in C either one by one or using a single statement as follows:

  int balance[5] = {10,20,30,40,50};

- The number of values between braces {} cannot be larger than the number of elements that we declare for the array between square brackets [ ].
- If you omit the size of the array, an array just big enough to hold the initialization is created. Therefore, if you write as shown below, it will create an array of seven elements and the following values will be stored in it.

  int balance[] = {10,20,30,40,50,60,70};

**MULTIDIMENSIONAL ARRAYS**

- C programming language allows multidimensional arrays. Here is the general form of a multidimensional array declaration –

  datatype name[size1][size2]...[sizeN];

- For example, the following declaration creates a three dimensional integer array –

  int a[3][4][5];

**Two-dimensional Arrays**

- The simplest form of multidimensional array is the two-dimensional array. A two-dimensional array is, in essence, a list of one-dimensional arrays. To declare a two-dimensional integer array of size [x][y], you would write something as follows –

  datatype arrayName [ x ][ y ];

- Where **type** can be any valid C data type and **arrayName** will be a valid C identifier. 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 **a**, which contains three rows and four columns can be shown as follows –

| | Column 0 | Column 1 | Column 2 | Column 3 |
|---|---|---|---|---|
| Row 0 | a[ 0 ][ 0 ] | a[ 0 ][ 1 ] | a[ 0 ][ 2 ] | a[ 0 ][ 3 ] |
| Row 1 | a[ 1 ][ 0 ] | a[ 1 ][ 1 ] | a[ 1 ][ 2 ] | a[ 1 ][ 3 ] |
| Row 2 | a[ 2 ][ 0 ] | a[ 2 ][ 1 ] | a[ 2 ][ 2 ] | a[ 2 ][ 3 ] |

- Every element in the array **a** is identified by an element name of the form **a[ i ][ j ]**, where 'a' is the name of the array, and 'i' and 'j' are the subscripts that uniquely identify each element in 'a'.

**Initializing Two-Dimensional Arrays**

- Multidimensional arrays may be initialized by specifying bracketed values for each row. Following is an array with 3 rows and each row has 4 columns.

```
int a[3][4] = {
{0, 1, 2, 3} ,   /* initializers for row indexed by 0 */
{4, 5, 6, 7} ,   /* initializers for row indexed by 1 */
{8, 9, 10, 11}   /* initializers for row indexed by 2 */
};
```

- The nested braces, which indicate the intended row, are optional. The following initialization is equivalent to the previous example –

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

## STRINGS

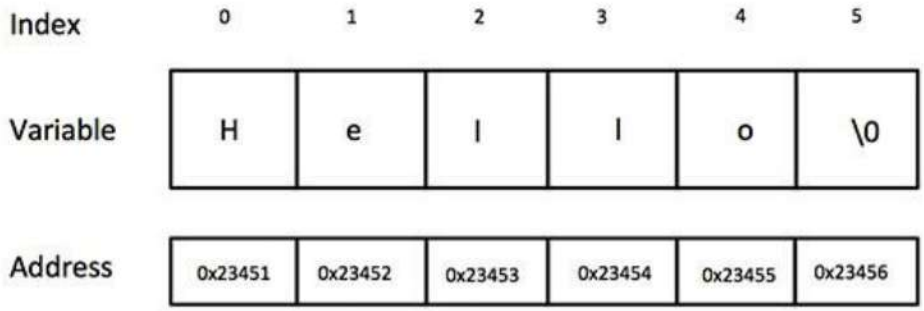
- Strings are actually one-dimensional array of characters terminated by a **null** character '\0'. Thus, a null-terminated string contains the characters that comprise the string followed by a **NULL**.
- The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

  char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

- If you follow the rule of array initialization then you can write the above statement as follows −

  char greeting[] = "Hello";

- Following is the memory presentation of the above defined string in C




- 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.

### STRING HANDLING FUNCTIONS

| Sl.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);**<br>Returns 0 if s1 and s2 are the same; less than 0 if s1<s2; greater than 0 if s1>s2. |
| 5 | **strchr(s1, ch);**<br>Returns a pointer to the first occurrence of character ch in string s1. |
| 6 | **strstr(s1, s2);**<br>Returns a pointer to the first occurrence of string s2 in string s1. |

## APPLICATIONS OF ARRAYS

- For Processing multiple elements of similar datatype
- Implementation of Stacks and Queues
- Matrix operations implementation
- 3D geometry

## SAMPLE PROGRAMS

### 25. C program to read and display elements of an array

```
#include<stdio.h>
int main()
{
	int a[100],i,j,n;
	printf("Enter the size of array");
	scanf("%d",&n);
	printf("\nEnter the elements of the array");
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	printf("\nThe elements of the array are\n");
	for(i=0;i<n;i++)
	{
		printf("%d\t",a[i]);
	}
return 0;
}
```

### 26. C program to perform linear search

```
#include<stdio.h>
int main()
{
	int a[100],i,j,n, flag=0,searchelement;
	printf("Enter the size of array");
	scanf("%d",&n);
	printf("\nEnter the elements of the array");
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	printf("\nEnter the element to be searched");
	scanf("%d",&searchelement);

	for(i=0;i<n;i++)
	{
		if(a[i]==searchelement)
		{
			flag++;
		}
	}
	if(flag==0)
	{
		printf("\nElement is not present");
	}
	else
	{
		printf("\nElement is present");
	}
return 0;
}
```

**27. C program to sort an array of n integers using Bubble sort**

```
#include<stdio.h>
int main()
{
	int a[100],i,j,n,temp;
	printf("Enter the size of array");
	scanf("%d",&n);
	printf("\nEnter the elements of the array");
	for(i=0;i<n;i++)
	{
		scanf("%d",&a[i]);
	}
	printf("\n The elements in the array before sorting\n");
	for(i=0;i<n;i++)
	{
		printf("\t%d",a[i]);
	}

	for(i=0;i<n;i++)
	{
		for(j=0;j<n-i-1;j++)
		{
			if(a[j]>a[j+1])
			{
				temp=a[j];
				a[j]=a[j+1];
				a[j+1]=temp;
			}
		}
	}
		printf("\n The elements in the array after sorting\n");
	for(i=0;i<n;i++)
	{
		printf("\t%d",a[i]);
	}
return 0;
}
```

**28. C program to find the transpose of a matrix**

```
#include<stdio.h>
int main()
{
int a[10][10],b[10][10],s[10][10],i,j,r,c;
printf("\nEnter the order of matrices");
scanf("%d%d",&r,&c);
printf("\nEnter the matrix");
for(i=0;i<r;i++)
{
	for(j=0;j<c;j++)
	{
		printf("\nEnter the element at %d %d location",i,j);
		scanf("%d",&a[i][j]);
	}
}
```

```
printf("\nThe transpose of matrix is\n");

        for(i=0;i<r;i++)
        {
        for(j=0;j<c;j++)
        {
                printf("\t%d",a[j][i]);
        }
        printf("\n");
}
return 0;
}
```

**29. C program to multiply two matrices.**

```
#include<stdio.h>
int main()
{
int a[10][10],b[10][10],c[10][10],i,j,k,r1,c1,r2,c2;
printf("\nEnter the order of first matrix");
scanf("%d%d",&r1,&c1);
printf("\nEnter the order of second matrix");
scanf("%d%d",&r2,&c2);
if(r1==c2)
{
printf("\nEnter the first matrix");
for(i=0;i<r1;i++)
{
        for(j=0;j<c1;j++)
        {
                printf("\nEnter the element at %d %d location",i,j);
                scanf("%d",&a[i][j]);
        }
}

printf("\nEnter the second matrix");
for(i=0;i<r2;i++)
{
        for(j=0;j<c2;j++)
        {
                printf("\nEnter the element at %d %d location",i,j);
                scanf("%d",&b[i][j]);
        }
}
for(i=0;i<r1;i++)
{
        for(j=0;j<c2;j++)
        {
                sum=0;
                for(k=0;k<c1;k++)
                {
                        sum=sum+a[i][k]*b[k][j];
                        c[i][j]=sum;
                }
        }
```

```
}
printf("\nThe resultant matrix is\n");
        for(i=0;i<r1;i++)
        {
        for(j=0;j<c2;j++)
        {
                printf("\t%d",c[i][j]);
        }
        printf("\n");
}
}
else
{
        printf("\nMultiplication is not possible for the given order");
}
return 0;
}
```

**30. C program to add two matrices**

```
#include<stdio.h>
int main()
{
        int a[3][3],b[3][3],c[3][3],i,j;
        printf("\nEnter the elements of first matrix");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        printf("\nEnter element at Row %d  Column %d ", i,j);
                        scanf("%d",&a[i][j]);
                }
        }
        printf("\nEnter the elements of second matrix");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        printf("\nEnter element at Row %d  Column %d ", i,j);
                        scanf("%d",&b[i][j]);
                }
        }
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        c[i][j]=a[i][j]+b[i][j];
                }
        }
                printf("\nSum of matrices are\n");
        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        printf("%d\t",c[i][j]);
                }
```

```
                printf("\n");
        }
        return 0;
}
```

**31. C program to read and display a string**

```
#include<stdio.h>
int main()
{
        char s[20];
        printf("\nEnter a string");
        scanf("%s",s);
        printf("\nThe entered string is %s", s);
return 0;
}
```

**32. C program to find the length of a string**

```
#include<stdio.h>
int main()
{
        char s[20];
        int i,len=0;
        printf("\nEnter a string");
        scanf("%s",s);
        printf("\nThe entered string is %s", s);
        for(i=0;s[i]!='\0';i++)
        {
                len++;
        }
        printf("\nThe length of the string %s is %d",s,len);
return 0;
}
```

**33. C program to find the reverse of a string**

```
#include<stdio.h>
int main()
{
        char str[20],rev[20];
        int i,j,len=0;
        printf("\nEnter a string");
        scanf("%s",str);
        printf("\nThe entered string is %s", str);
        for(i=0;str[i]!='\0';i++)
        {
                len++;
        }
        for(i=len-1,j=0;i>=0;i--,j++)
        {
                rev[j]=str[i];
        }
        printf("\nThe reverse of string %s is %s",str,rev);
return 0;
}
```

**34. C program to concatenate two strings**

```
#include<stdio.h>
#include<string.h>
int main()
{
        char s1[20],s2[20],s3[40];
        int len1,len2,i,j;
        printf("Enter first string");
        scanf("%s",s1);
        printf("\nEnter second string");
        scanf("%s",s2);
        len1=strlen(s1);
        len2=strlen(s2);
        for(i=0;i<len1;i++)
        {
                s3[i]=s1[i];      //Copying string 1 to string 3
        }
        //copying string 2 to string 3
        for(i=len1,j=0;j<len2;i++,j++)
        {
                s3[i]=s2[j];
        }
        printf("\nThe concatenated string is %s",s3);
return 0;
}
```

**35. C program to read a character from user and check whether that character is present in the string or not.**

```
#include<stdio.h>
#include<string.h>
int main()
{
        char s[10],ch;
        int i,len,flag=0;
        printf("\nEnter character to be searched");
        scanf("%c",&ch);
        printf("\nEnter string");
        scanf("%s",s);
        len=strlen(s);
        for(i=0;s[i]!='\0';i++)
        {
                if(s[i]==ch)
                {
                flag=1;
                break;
                }
        }
        if(flag==0)printf("Character not present");
        else printf("\nCharacter present");
        return 0;
}
```

36. **C program to check whether a string is palindrome or not**

```
#include<stdio.h>
#include<string.h>
int main()
{
char a[100], b[100];
printf("Enter a string to check if it is a palindrome\n");
gets(a);
strcpy(b, a);  // Copying input string
strrev(b);  // Reversing the string
if (strcmp(a, b) == 0) printf("The string is a palindrome.\n");
else printf("The string isn't a palindrome.\n");
return 0;
}
```

# MODULE 4

# FUNCTIONS

## FUNCTIONS IN C

- In C, we can divide a large program into the basic building blocks known as function. The function contains the set of programming statements enclosed by {}.
- A function can be called multiple times to provide reusability and modularity to the C program.
- In other words, we can say that the collection of functions creates a program.
- The function is also known as procedure or subroutine in other programming languages.

## ADVANTAGES OF FUNCTIONS

- By using functions, we can avoid rewriting same logic/code again and again in a program.
- We can call C functions any number of times in a program and from any place in a program.
- We can track a large C program easily when it is divided into multiple functions.
- Reusability is the main achievement of C functions.

## FUNCTION DECLARATION

- The function can be declared using the following syntax.

  return_type function_name(argument list);

- 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 declaration is required when you define a function in one source file and you call that function in another file. In such case, you should declare the function at the top of the file calling the function.

## FUNCTION DEFINITION

- **Syntax:**

```
return_type function_name(argument list )
{
   body of the function
}
```

- A function definition in C programming consists of a function header and a function body. Here are all the parts of a function –

  **Return Type** – A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

  **Function Name** – This is the actual name of the function. The function name and the parameter list together constitute the function signature.

  **Parameters** – A parameter is like a placeholder. When a function is invoked, you pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

  **Function Body** – The function body contains a collection of statements that define what the function does.

**FUNCTION CALLING**

- The function is invoked only if it is called.
- Syntax: function_name(argument list);
- When a program calls a function, the program control is transferred to the called function. A called function performs a defined task and when its return statement is executed or when its function-ending closing brace is reached, it returns the program control back to the main program.
- To call a function, you have to pass the required parameters along with the function name, and if the function returns a value, then you can store the returned value.

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

  **1. 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.

  **2 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 affect the argument.

- By default, C uses call by value to pass arguments. In general, it means the code within a function cannot alter the arguments used to call the function.

**RECURSION / RECURSIVE FUNCTION**

- Function calling itself is termed as recursion.
- Stack is used for the implementation of recursion.

## STORAGE CLASSES IN C

- Storage Classes are used to describe about the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.
- C language uses 4 storage classes, namely auto,register,static,extern.

**AUTO**

- This is the default storage class for all the variables declared inside a function or a block. Hence, the keyword auto is rarely used while writing programs in C language.
- Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope). Of course, these can be accessed within nested blocks within the parent block/function in which the auto variable was declared. However, they can be accessed outside their scope as well using the concept of pointers given here by pointing to the very exact memory location where the variables reside.
- They are assigned a garbage value by default whenever they are declared.

**EXTERN**

- Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used.
- Basically, the value is assigned to it in a different block and this can be overwritten/changed in a different block as well.
- So, an extern variable is nothing but a global variable initialized with a legal value where it is declared in order to be used elsewhere. It can be accessed within any function/block. Also, a normal global variable can be made extern as well by placing the 'extern' keyword before its declaration/definition in any function/block.
- The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program. For more information on how extern variables work, have a look at this link.

**STATIC**

- This storage class is used to declare static variables which are popularly used while writing programs in C language.
- Static variables have a property of preserving their value even after they are out of their scope. Hence, static variables preserve the value of their last use in their scope.
- So, they are initialized only once and exist till the termination of the program. Thus, no new memory is allocated because they are not re-declared.
- Their scope is local to the function to which they were defined. Global static variables can be accessed anywhere in the program. By default, they are assigned the value 0 by the compiler.

**REGISTER**

- This storage class declares register variables which have the same functionality as that of the auto variables. The only difference is that the compiler tries to store these variables in the register of the microprocessor if a free register is available.
- This makes the use of register variables to be much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only.
- Usually, few variables which are to be accessed very frequently in a program are declared with the register keyword which improves the running time of the program.
- An important and interesting point to be noted here is that we cannot obtain the address of a register variable using pointers.

| Storage Specifier | Storage | Initial value | Scope | Life |
|---|---|---|---|---|
| auto | Stack | Garbage | Within block | End of block |
| extern | Data segment | Zero | global Multiple files | Till end of program |
| static | Data segment | Zero | Within block | Till end of program |
| register | CPU Register | Garbage | Within block | End of block |

## PARAMETER PASSING MECHANISMS

| Call by Value | Call by Reference |
|---|---|
| The actual arguments can be variable or constant. | The actual arguments can only be variable. |
| The values of actual argument are sent to formal argument which are normal variables. | The reference of actual argument is sent to formal argument which are pointer variables. |
| Any changes made by formal arguments will not reflect to actual arguments. | Any changes made by formal arguments will reflect to actual arguments. |

## SAMPLE PROGRAMS

### 37. Factorial of a number using recursion

```
#include<stdio.h>
int factorial(int n)
{
        if(n>1)return n*factorial(n-1);
        else return 1;
}
int main()
{
        int number,fact;
        printf("Enter number");
        scanf("%d",&number);
        fact=factorial(number);
        printf("Factorial of %d is %d",number,fact);
return 0;
}
```

### 38. Fibonacci Series using recursion

```
#include<stdio.h>
int Fibonacci(int n)
{
  if ( n == 0 )
    return 0;
  else if ( n == 1 )
    return 1;
  else
    return ( Fibonacci(n-1) + Fibonacci(n-2) );
}

int main()
{
  int n, i = 0, c;
     printf("Enter range");
  scanf("%d",&n);
  printf("Fibonacci series\n");
  for ( c = 1 ; c <= n ; c++ )
  {
    printf("%d\n", Fibonacci(i));
    i++;
  }
return 0;
}
```

**39. C program to print area and perimeter of rectangle using functions.**

```
#include<stdio.h
int calcarea(int len, int brd)
{
        int area;
        area=len*brd;
        return area;
}
int calcperi(int len,int brd)
{
        int peri;
        peri=2*(len+brd);
        return peri;
}
int main()
{
        int l,b,a=0,p=0;
        printf("Enter length and breath:");
        scanf("%d %d",&l,&b);
        a=calcarea(l,b);
        p=calcperi(l,b);
        printf("Area=%d",a);
        printf("\nPerimeter=%d",p);
return 0;
}
```

**40. C program to display Fibonacci series using function.**

```
#include<stdio.h>
void fibonacci()
{
        int previous=0,current=1,range,i,term;
        printf("Enter range");
        scanf("%d",&range);
        printf("Fibonacci series is \n 0 1");
        for(i=0;i<range-2;i++)
        {
                term=current+previous;
                printf(" %d", term);
                previous=current;
                current=term;
        }
}
int main()
{
        fibonacci();
return 0;
}
```

**41. Menu driven program to find the area of rectangle, circle, square using functions.**

```
#include<stdio.h>
void rectangle()
{
 int l,b,area;
 printf("enter the length and breadth of a rectangle");
```

```
 scanf("%d%d",&l,&b);
 area=l*b;
 printf("area=%d",area);
}
void circle()
{
        int r;
        float area;
        printf("enter the radius");
        scanf("%d",&r);
        area=3.14*r*r;
        printf("area=%f",area);
}
void square()
{
int a,area;
printf("enter side of a square");
scanf("%d",&a);
area=a*a;
printf("area=%d",area);
}
int main()
{
        int choice;
        printf("MENU");
        printf("\n 1:  area of rectangle,  \n2: area of circle,  \n3: area of square");
printf("enter your choice");
scanf("%d",&choice);
switch(choice)
{
        case 1: rectangle();break;
        case 2: circle();break;
        case 3: square();break;
        default: printf("\nInvalid choice");
}
return 0;
}
```

**42. Menu driven program to perform arithmetic operations using functions.**

```
#include<stdio.h>
void add()
{
        int n1,n2,res;
        printf("\n Enter two numbers");
        scanf("%d %d",&n1,&n2);
        res=n1+n2;
        printf("\n Sum=%d",res);
}
void sub()
{
        int n1,n2,res;
        printf("\n Enter two numbers");
        scanf("%d %d",&n1,&n2);
        res=n1-n2;
```

```
        printf("\n Difference=%d",res);
}
void mul()
{
int n1,n2,res;
        printf("\n Enter two numbers");
        scanf("%d %d",&n1,&n2);
        res=n1*n2;
        printf("\n Product=%d",res);
}
void div()
{
        int n1,n2,res,rem;
        printf("\n Enter two numbers");
        scanf("%d %d",&n1,&n2);
        res=n1/n2;
        rem=n1%n2;
        printf("\n Quotient=%d     Reminder =%d",res,rem);
}
int main()
{
        int ch,n;
        do
        {
        printf("MENU");
        printf("\n1. Addition \n 2. Subtraction  \n 3. Multiplication \n 4. Division \n Enter
your choice");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: add(); break;
                case 2: sub(); break;
                case 3: mul(); break;
                case 4: div(); break;
                default: printf("\nInvalid option is chosen");
        }
        printf("\n\nDo you want to run again-----Press 1 to continue");
        scanf("%d",&n);
}while(n==1);
return 0;
}
```

**43. Menu driven C program to perform string operations**

```
#include<stdio.h>
#include<string.h>
void stringlen()
{
        char s[100];
        int len;
        printf("\nEnter a string");
        scanf("%s",s);
        len=strlen(s);
        printf("\n Length of %s is %d",s,len);
}
```

```
void stringrev()
{
        char s[100],rev[100];
        printf("\nEnter a string");
        scanf("%s",s);
        printf("\nReverse is %s",strrev(s));
}
void stringconcat()
{
        char s1[100],s2[100];
        printf("\nEnter first string");
        scanf("%s",s1);
        printf("\nEnter second string");
        scanf("%s",s2);
        printf("\n Concatenated string is %s",strcat(s1,s2));
}
int main()
{
        int ch,n;
        do
        {
        printf("\n MENU\n 1: Calculating String length \n 2: Reverse of  a string \n 3: Concatenate two strings");
        printf("\n Enter your choice");
        scanf("%d",&ch);
        switch(ch)
        {
                case 1: stringlen(); break;
                case 2: stringrev(); break;
                case 3: stringconcat(); break;
                default: printf("\n Invalid choice");
        }
        printf("\nDo you want to continue..Press 1 to continue");
        scanf("%d",&n);
}while(n==1);
return 0;
}
```

**44. Swap to numbers using Call by reference**

```
#include<stdio.h>
void swap(int *n1,int *n2)
{
        printf("\nIn function program before swapping number1=%d and number2 =%d",n1,n2);
        int temp;
        temp=*n1;
        *n1=*n2;
        *n2=temp;
        printf("\nIn function program after swapping number1=%d and number2 =%d",n1,n2);
}
int main()
{
        int n1,n2;
        printf("Enter two numbers");
```

```
        scanf("%d %d",&n1,&n2);
        printf("\nIn main program before swapping number1=%d and number2 =%d",n1,n2);
        swap(&n1,&n2);
        printf("\nIn main program after swapping number1=%d and number2 =%d",n1,n2);
return 0;
}
```

## STRUCTURE

- Structure is a heterogeneous collection data items.
- struct keyword is used to create a structure.
- **Syntax of structure:**

```
struct structure name
   {
     member definition;
     member definition;
     ...
     member definition;
   };
```

## UNION

- Union is also heterogeneous collection of data items. But union allocates the memory size of the largest element and that memory is shared by other elements.
- union keyword is used to create a union.
- **Syntax of Union:**

```
union [union name]
   {
     member definition;
     member definition;
     ...
     member definition;
   };
```

## DIFFERENCE BETWEEN STRUCTURE & UNION

| | STRUCTURE | UNION |
|---|---|---|
| **Keyword** | The keyword **struct** is used to define a structure | The keyword **union** 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 the first member of a union can be initialized. |

## SAMPLE PROGRAMS – STRUCTURE

**45. C program to display student record. Read name, roll number, marks of three subjects from user. Compute total and average and print rank list based on total marks.**

```
#include<stdio.h>
struct student
{
        char name[20];
        int rollno,m1,m2,m3,total,average;
};
int main()
{
        student s[5],temp ;
        int i=0,j;
        for(i=0;i<5;i++)
        {
                printf("\nEnter name");
                scanf("%s",s[i].name);
                printf("\nEnter roll no");
                scanf("%d",&s[i].rollno);
                printf("\nEnter marks of 3 subjects");
                scanf("%d%d%d",&s[i].m1,&s[i].m2,&s[i].m3);
                s[i].total=s[i].m1+s[i].m2+s[i].m3;
                s[i].average=s[i].total/3;
        }
        for(i=0;i<5;i++)
        {
                for(j=0;j<5-1-i;j++)
                {
                        if(s[j].total>s[j+1].total)
                        {
                                temp=s[j];
                                s[j]=s[j+1];
                                s[j+1]=temp;
                        }

                }
        }
        printf("\nNAME\tROLL NO\tMARK1\tMARK2\tMARK3\tTOTAL\tAVERAGE");
        for(i=0;i<5;i++)
        printf("\n%s\t%d\t%d\t%d\t%d\t%d",s[i].name,s[i].rollno,s[i].m1,s[i].m2,s[i].m
3,s[i].total,s[i].average);
        return 0;
}
```

46. **C program to read name, roll number, marks of three subjects. Calculate the equivalent grades and display the grade sheet.**

```
#include<stdio.h>
#include<string.h>
struct student
{
        char name[20];
        int rollno,m1,m2,m3;
        char g1[2],g2[2],g3[2];
};
void main()
{
        struct student s[5];
        int i;
        for(i=0;i<5;i++)
        {

        printf("\nEnter the name roll number Mark of first subject Mark of second subject, MArk of third subject");
        scanf("%s %d %d %d %d",s[i].name,&s[i].rollno,&s[i].m1,&s[i].m2,&s[i].m3);
        //Grade of first subject
        if(s[i].m1>=90) strcpy(s[i].g1,"A+");
        else if((s[i].m1>=80)&&(s[i].m1<90)) strcpy(s[i].g1,"A");
        else if((s[i].m1>=70)&&(s[i].m1<80)) strcpy(s[i].g1,"B+");
        else if((s[i].m1>=60)&&(s[i].m1<70)) strcpy(s[i].g1,"B");
        else if((s[i].m1>=50)&&(s[i].m1<60)) strcpy(s[i].g1,"C+");
        else strcpy(s[i].g1,"F");

        //Grade of second subject
        if(s[i].m2>=90) strcpy(s[i].g2,"A+");
        else if((s[i].m2>=80)&&(s[i].m2<90)) strcpy(s[i].g2,"A");
        else if((s[i].m2>=70)&&(s[i].m2<80)) strcpy(s[i].g2,"B+");
        else if((s[i].m2>=60)&&(s[i].m2<70)) strcpy(s[i].g2,"B");
        else if((s[i].m2>=50)&&(s[i].m2<60)) strcpy(s[i].g2,"C+");
        else strcpy(s[i].g2,"F");

        //Grade of Third subject
        if(s[i].m3>=90) strcpy(s[i].g3,"A+");
        else if((s[i].m3>=80)&&(s[i].m3<90)) strcpy(s[i].g3,"A");
        else if((s[i].m3>=70)&&(s[i].m3<80)) strcpy(s[i].g3,"B+");
        else if((s[i].m3>=60)&&(s[i].m3<70)) strcpy(s[i].g3,"B");
        else if((s[i].m3>=50)&&(s[i].m3<60)) strcpy(s[i].g3,"C+");
        else strcpy(s[i].g3,"F");
        }
        printf("\nName\t\tRoll No\t Mark 1\t Grade 1\t Mark 2\t Grade 2\t Mark 3\t Grade 3");
        for(i=0;i<5;i++)
        {
        printf("\n\n%s\t\t%d\t%d\t%s\t%d\t%s\t%d\t%s",s[i].name,s[i].rollno,s[i].m1,s[i].g1,s[i].m2,s[i].g2,s[i].m3,s[i].g3);

        }
}
```

# MODULE 5

# POINTERS & FILES

## POINTERS

- A Pointer is a variable which holds the address of another variable of same data type.
- Pointers are used to access memory and manipulate the address.

## ADDRESS IN C

- Whenever a variable is defined in C language, a memory location is assigned for it, in which its value will be stored. We can check this memory address, using the & symbol.
- If var is the name of the variable, then &var will give it's address.
- Consider the following program:

```
#include<stdio.h>
int main()
{
   int var = 7;
   printf("Value of the variable var is: %d\n", var);
   printf("Memory address of the variable var is: %x\n", &var);
return 0;
}
```

- In above program Value of the variable var is: 7 Memory address of the variable var is: bcc7a00
- You must have also seen in the function scanf(), we mention &var to take user input for any variable var.

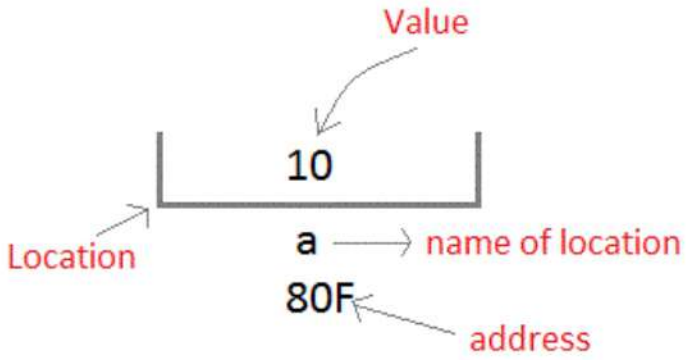
```
scanf("%d", &var);
```

- This is used to store the user inputted value to the address of the variable var.

## POINTER CONCEPT

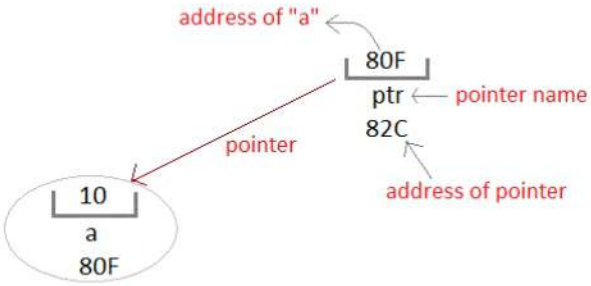
- Whenever a **variable** is declared in a program, system allocates a location i.e an address to that variable in the memory, to hold the assigned value. This location has its own address number.
- Assume that system has allocated memory location 80F for a variable a.

```
int a = 10;
```




- We can access the value 10 either by using the variable name a or by using its address 80F.

- The question is how we can access a variable using it's address? Since the memory addresses are also just numbers, they can also be assigned to some other variable. The variables which are used to hold memory addresses are called **Pointer variables**.
- A **pointer** variable is therefore nothing but a variable which holds an address of some other variable. And the value of a **pointer variable** gets stored in another memory location.




---

**BENEFITS OF USING POINTERS**

- Pointers are more efficient in handling Arrays and Structures.
- Pointers allow references to function and thereby helps in passing of function as arguments to other functions.
- It reduces length of the program and its execution time as well.
- It allows C language to support Dynamic Memory management.

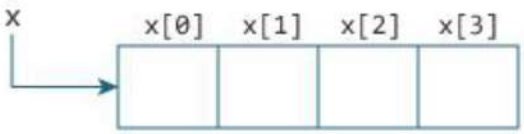
**NULL POINTER**

- A Null Pointer is a pointer that does not point to any memory location. It stores the base address of the segment. The null pointer basically stores the Null value while void is the type of the pointer.
- A null pointer is a special reserved value which is defined in a stddef header file. Here, Null means that the pointer is referring to the 0th memory location.

## ARRAY ACCESS USING POINTERS

- It is possible to access the array elements using pointers.
- Consider an array:

```
int x[4];
```




- From the above example, it's clear that *x* and &x[0] both contains the same address. Hence, &x[0] is equivalent to *x*. And, x[0] is equivalent to *x.
- Similarly,
    - &x[1] is equivalent to x+1 and x[1] is equivalent to *(x+1).
    - &x[2] is equivalent to x+2 and x[2] is equivalent to *(x+2).

- ...
- Basically, &x[i] is equivalent to x+i and x[i] is equivalent to *(x+i).
-

- Example program to display array elements using pointer variable.

```
#include <stdio.h>
int main() {
   int a[5],i;
   printf("Enter elements: ");
   for (i = 0;i < 5;i++)
        {
        scanf("%d", &a[i]);
        }
   printf("The array elements are: \n");
   for (i = 0; i < 5;i++)
        {
        printf("%d\n", *(a+ i));   //a represents the base address of array a[5].
}
   return 0;
}
```

## FILES

- File is a place on your physical disk where information is stored.

### NEED OF FILES

- When a program is terminated, the entire data is lost. Storing in a file will preserve your data even if the program terminates.
- If you have to enter a large number of data, it will take a lot of time to enter them all. However, if you have a file containing all the data, you can easily access the contents of the file using few commands in C.
- You can easily move your data from one computer to another without any changes.

## TYPES OF FILES

### 1. TEXT FILES

- Text files are the normal .txt files that you can easily create using Notepad or any simple text editors.
- When you open those files, you'll see all the contents within the file as plain text. You can easily edit or delete the contents.
- They take minimum effort to maintain, are easily readable, and provide least security and takes bigger storage space.

### 2. BINARY FILES

- Binary files are mostly the .bin files in your computer.
- Instead of storing data in plain text, they store it in the binary form (0's and 1's).
- They can hold higher amount of data, are not readable easily and provides a better security than text files.

## FILE OPERATIONS

1. Creating a new file
2. Opening an existing file
3. Closing a file
4. Reading from and writing information to a file

**Working with files**

When working with files, you need to declare a pointer of type file. This declaration is needed for communication between the file and program.

```
FILE *fptr;
```

**Opening a file - for creation and edit**

- Opening a file is performed using the library in the "**stdio.h**" header file: fopen().
- The syntax for opening a file in standard I/O is:

```
ptr = fopen("fileopen","mode")
```

For Example: fp1=fopen("data.txt",'w');

- Suppose if the file data.txt doesn't exist in the system. The first function creates a new file named data.txt and opens it for writing as per the mode 'w'. The writing mode allows you to create and edit (overwrite) the contents of the file.

| Opening Modes in Standard I/O | | |
|---|---|---|
| **File Mode** | **Meaning of Mode** | **During Inexistence of file** |
| R | Open for reading. | If the file does not exist, fopen() returns NULL. |
| Rb | Open for reading in binary mode. | If the file does not exist, fopen() returns NULL. |
| W | Open for writing. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| Wb | Open for writing in binary mode. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| A | Open for append. i.e, Data is added to end of file. | If the file does not exists, it will be created. |
| Ab | Open for append in binary mode. i.e, Data is added to end of file. | If the file does not exists, it will be created. |
| r+ | Open for both reading and writing. | If the file does not exist, fopen() returns NULL. |
| rb+ | Open for both reading and writing in binary mode. | If the file does not exist, fopen() returns NULL. |
| w+ | Open for both reading and writing. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| wb+ | Open for both reading and writing in binary mode. | If the file exists, its contents are overwritten. If the file does not exist, it will be created. |
| a+ | Open for both reading and appending. | If the file does not exists, it will be created. |
| ab+ | Open for both reading and appending in binary mode. | If the file does not exists, it will be created. |

**Closing a File**

- The file (both text and binary) should be closed after reading/writing.
- Closing a file is performed using library function fclose().
- fclose(fptr); //fptr is the file pointer associated with file to be closed.

**Reading and writing to a text file**

- For reading and writing to a text file, we use the functions fprintf() and fscanf().
- They are just the file versions of printf() and scanf(). The only difference is that, fprint and fscanf expects a pointer to the structure FILE.

## SAMPLE PROGRAMS - FILES

### 47. Program to copy the contents of one file to another

```
#include <stdio.h>
main()
{
   FILE *fp1, *fp2;
   char ch;
   fp1 = fopen("abc.txt", "r");
   fp2 = fopen("xyz.txt", "w");
   while((ch = getc(fp1)) != EOF)
      putc(ch, fp2);
   fclose(fp1);
   fclose(fp2);
   getch();
}
```

### 48. Program to write text to a file a print the number of vowels in that file.

```
#include<stdio.h>
void main()
{
        FILE *fp1,*fp2;
        char character;
        int count;
        fp1=fopen("file1.txt","w");
        printf("\nEnter characters,Press escape to stop");
        while(1)
        {
        character=getche();
        if(character==27) break;
        putc(character,fp1);
        }
        fclose(fp1);
        printf("\n Contents of file\n");
        fp1=fopen("file1.txt","r");
        if(fp1==NULL)
        {
                printf("\nCannot open file");
        }
        while((character=getc(fp1))!=EOF)
        {
```

```
        printf("%c",character);
        }
        fclose(fp1);
        fp1=fopen("file1.txt","r");
        fp2=fopen("Vowels.txt","w");
        while((character=getc(fp1))!=EOF)
        {

        if((character=='a')||(character=='e')||(character=='i')||(character=='o')||(character==
'u')
||(character=='A')||(character=='E')||(character=='I')||(character=='O')||(character=='U'))
                {
                        count++;
                        putc(character,fp2);
                }
        }
fclose(fp1);
fclose(fp2);
fp2=fopen("Vowels.txt","r");
printf("\nVowels in the text are ");
while((character=getc(fp2))!=EOF)
{
        printf("%c ",character);
}
printf("\nNumber of vowels in the text are %d",count);
fclose(fp2);
}
```

**49. Program to read n integers to a file. Write the odd and even numbers to separate files.**

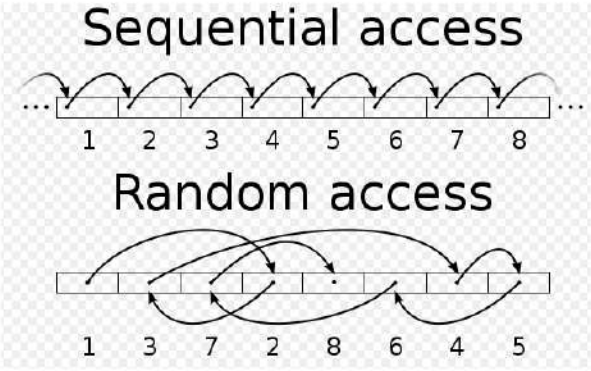
```
#include<stdio.h>
void main()
{
        int number,i,n;
        FILE *fp1,*fp2,*fp3;
        printf("\nEnter the number of integers");
        scanf("%d",&n);
        fp1=fopen("DATA","w");
        printf("\n\nEnter numbers to the data file");
        for(i=1;i<=n;i++)
        {
                printf("Enter number %d",i);
                scanf("%d",&number);
                putw(number,fp1);
        }
        fclose(fp1);
        fp1=fopen("DATA","r");
        fp2=fopen("ODD","w");
        fp3=fopen("EVEN","w");

        while((number=getw(fp1))!=EOF)
        {
                if(number%2==0) putw(number,fp3);
                else putw(number,fp2);
```

```
        }
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
        fp1=fopen("DATA","r");
        fp2=fopen("ODD","r");
        fp3=fopen("EVEN","r");

        printf("\n Contents of Data file is");
        while((number=getw(fp1))!=EOF)
        {
                printf("%d ",number);
        }
        printf("\n Contents of ODD file is");
        while((number=getw(fp2))!=EOF)
        {
                printf("%d ",number);
        }
        printf("\n Contents of Even file is");
        while((number=getw(fp3))!=EOF)
        {
                printf("%d ",number);
        }
        fclose(fp1);
        fclose(fp2);
        fclose(fp3);
}
```

**SEQUENTIAL & RANDOM ACCESS IN FILES**




- In computer programming, the two main types of file handling are:
  - Sequential;
  - Random access.

**Sequential files** are generally used in cases where the program processes the data in a sequential fashion – i.e. counting words in a text file – although in some cases, random access can be feigned by moving backwards and forwards over a sequential file.

**True random access** file handling, however, only accesses the file at the point at which the data should be read or written, rather than having to process it sequentially. A hybrid approach is also possible whereby a part of the file is used for sequential access to locate something in the random access portion of the file, in much the same way that a File Allocation Table (FAT) works.

| File operation | Declaration & Description |
|---|---|
| fscanf() | Declaration: int fscanf(FILE *fp, const char *format, ...)<br>fscanf() function is used to read formatted data from a file. In a C program, we use fscanf() as below.<br><br>fscanf (fp, "%d", &age);<br>Where, fp is file pointer to the data type "FILE".<br>Age – Integer variable<br>This is for example only. You can use any specifiers with any data type as we use in normal scanf() function. |
| fprintf() | Declaration: int fprintf(FILE *fp, const char *format, ...)<br>fprintf() function is used to write formatted data into a file. In a C program, we use fprinf() as below.<br>fprintf (fp, "%s %d", "var1", var2);<br><br>Where, fp is file pointer to the data type "FILE".<br>var1 – string variable<br>var2 – Integer variable<br>This is for example only. You can use any specifiers with any data type as we use in normal printf() function. |
| ftell() | Declaration: long int ftell(FILE *fp)<br>ftell function is used to get current position of the file pointer. In a C program, we use ftell() as below.<br>**ftell**(fp); |
| rewind() | Declaration: void rewind(FILE *fp)<br>rewind function is used to move file pointer position to the beginning of the file. In a C program, we use rewind() asbelow.<br>rewind(fp); |
| fseek() | int fseek(FILE *pointer, long int offset, int position)<br>pointer: pointer to a FILE object that identifies the stream.<br>offset: number of bytes to offset from position<br>position: position from where offset is added.<br><br>returns:<br>zero if successful, or else it returns a non-zero value<br>position defines the point with respect to which the file pointer needs to be moved. It has three values:<br>SEEK_END : It denotes end of the file.<br>SEEK_SET : It denotes starting of the file.<br>SEEK_CUR : It denotes file pointer's current position. |
| feof() | The C library function int feof(FILE *stream) tests the end-of-file indicator for the given stream. |
| fread() | The fread() function reads the entire record (of a structure) at a time. |
| fwrite() | The fwrite() function writes an entire record at a time. |

Reg No.:_____ Name:_____

**APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY**

Second Semester B.Tech Degree Examination July 2021 (2019 scheme)

**Course Code: EST102**

**Course Name: PROGRAMMING IN C (Common to all programs)**

**(FN Session)**

Max. Marks: 100 Duration: 3 Hours

**PART A**

***Answer all Questions. Each question carries 3 Marks*** Marks

1 Differentiate between system software and application software. (3)

2 Differentiate between complier and interpreter. (3)

3 What is the importance of precedence and associativity? Write the table for operator precedence. (3)

4 Differentiate between 'break' and 'continue' statements. (3)

5 Explain any 3 string handling functions using examples. (3)

6 Write a C program to find the occurrence of each element in an array. (3)

7 Define formal parameters and actual parameters. Illustrate with an example. (3)

8 With examples show how: (3)

(i) an array is passed as argument of a function.

(ii) individual elements of an array is passed as argument of a function.

9 Write any three file handling functions in C. (3)

10 Differentiate between address operator(&) and indirection(*) operator. (3)

**PART B**

***Answer any one Question from each module. Each question carries 14 Marks***

11 a) Explain different types of memory used in a computer. (7)

b) Write an algorithm to find sum of digits of a number. (7)

**OR**

12 Explain bubble sort with an example. Draw a flowchart and write pseudo code to perform bubble sort on an array of numbers. (14)

13 a) Explain different data types supported by C language with their memory requirements. (7)

b) Write a C program to check if a number is present in a given list of numbers. If present, give location of the number otherwise insert the number in the list at the end. (7)

**OR**

14 a) Write a C program to find the sum of first and last digit of a number. (7)

b) What is type casting? Name the inbuilt typecasting functions available in C language. What is the difference between type casting and type conversion? (7)

15 a) Write a C program to perform linear search on an array of numbers. (7)

b) Write a C program to reverse a string without using string handling functions. (7)

**OR**

16 a) Write a C program to find the transpose of a matrix. (7)

b) Write a C program to print number of vowels and consonants in a string. (7)

17 a) What is the purpose of function declaration and function definition and function call? With examples illustrate their syntax. (7)

b) Write a C program to : (7)

(i) Create a structure containing containing the fields: Name, Price, Quantity, Total Amount.

(ii) Use separate functions to read and print the data

**OR**

18 a) What are different storage classes in C? Give examples for each. (7)

b) Write a C program to find sum and average of an array of integers using user defined functions. (7)

19 a) Explain the different modes of operations performed on a file in C language. (7)

b) Write a program in C to copy the contents of one file into another. (7)

**OR**

20 a) Explain how pointers can be passed to functions in C. (7)

b) Explain any 5 file handling functions in C? (7)

****

# 01EST102052002 B

**F** **Pages: 2**

Reg No.:_____ Name:_____

**APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY**

Second Semester B.Tech Degree Examination July 2021 (2019 scheme)

**Course Code: EST102**
**Course Name: PROGRAMMING IN C**
**(AN Session)**

Max. Marks: 100 Duration: 3 Hours

**PART A**

***Answer all Questions. Each question carries 3 Marks*** Marks

| | | | |
|---|---|---|---|
| 1 | | Differentiate between system software and application software. | (3) |
| 2 | | Write an algorithm to find the largest of three numbers | (3) |
| 3 | | What is the difference between assignment and equality operators? | (3) |
| 4 | | What is a static variable? When should it be used? | (3) |
| 5 | | Write a C program to find length of a string without using string handling functions. | (3) |
| 6 | | What is an array? Illustrate using an example, how a single dimensional array is initialised. | (3) |
| 7 | | Differentiate between structure and union using an example. | (3) |
| 8 | | Illustrate the purpose of return statement using an example. | (3) |
| 9 | | Differentiate between char name[] and char *name in C. | (3) |
| 10 | | Explain the use of fseek() function. | (3) |

**PART B**

***Answer any one Question from each module. Each question carries 14 Marks***

| | | | |
|---|---|---|---|
| 11 | a) | Draw a flowchart to find the factorial of a number. | (6) |
| | b) | With the help of a neat diagram explain the functional units of a computer. | (8) |

**OR**

| | | | |
|---|---|---|---|
| 12 | a) | List five important registers in CPU. Also state the purpose of each register. | (6) |
| | b) | Write algorithm and draw flowchart to perform swapping of two numbers. | (8) |
| 13 | a) | Explain arithmetic, logical and bitwise operators with examples. | (7) |
| | b) | Write a C Program to check if a given number is a strong number or not. A strong number is a number in which the *sum of the factorial of the digits is equal to the number itself.*<br>Eg:- 145=1!+4!+5!=1+24+120=145 | (7) |

**OR**

| | | | |
|---|---|---|---|
| 14 | a) | Write C program to convert the given decimal number into binary number. | (7) |

| | | | |
|---|---|---|---|
| | b) | What do you mean by Formatted Input? Explain in detail the prototype of 'scanf()' function in C including its argument list and return type. | (7) |
| 15 | a) | Explain any 4 string handling functions in C programming. | (7) |
| | b) | Write a C program to perform linear search on an array of numbers | (7) |
| | | **OR** | |
| 16 | a) | Write a C program to find second largest element in an array. | (7) |
| | b) | Write a C program to check whether a string is palindrome or not without using string handling functions. | (7) |
| 17 | a) | What are different storage classes in C? Give examples for each. | (7) |
| | b) | Write a C program to:<br>(i) Create a structure with fields: Name, Address, Date of birth.<br>(ii) Read the above details for five students from user and display the details | (7) |
| | | **OR** | |
| 18 | a) | What is recursion? Write a C program to display Fibonacci series using recursive function. | (7) |
| | b) | Write a C program to sort N numbers using functions. | (7) |
| 19 | a) | Explain any 5 file handling functions in C. | (7) |
| | b) | Write a C program to reverse a string using pointers. | (7) |
| | | **OR** | |
| 20 | a) | Differentiate between array of pointers and pointer to an array. | (7) |
| | b) | Write a C program to count number of lines in a text file. | (7) |

****

## TOPICS COVERED

BASICS OF COMPUTER ARCHITECTURE
INTRODUCTION TO C PROGRAMMING
ARRAYS & STRINGS
FUNCTIONS & STRUCTURES
POINTERS & FILES

# EST102 PROGRAMMING IN C

DR. PRAMOD MATHEW JACOB