

UNIT-1

Module – 1: (Introduction to Programming)

Introduction to components of a computer system

- Memory, processor, I/O Devices
- Storage, Operating system, Concept of assembler, Compiler, Interpreter, Loader and Linker.

Idea of Algorithm:

- Representation of Algorithm
- Flowchart, Pseudo code with examples
- From algorithms to programs, Source code.

Programming Basics:

- Structure of C program
- Writing and executing the first C program.
- Syntax and logical errors in compilation, Object and executable code.
- Components of C language. Standard I/O in C.
- Fundamental data types, Variables and Memory locations.
- Storage classes.

Short Questions & Answers

Q1. What are the various functions of an Operating System?

Ans: (i) Process Management (ii) Memory Management (iii) CPU Scheduling
(iv) I/O Management (v) Network Management. (vi) Security management.

Q2. What is the working of Loader System Utility?

Ans: This is an OS utility that copies programs from a storage device to RAM(Main Memory) , where they can be executed. Types of loaders are like Absolute loader, bootstrap loader

Q3. What is the difference between low level language and high level language?

Ans: Low level language is machine language in form of 0 and 1 or Assembly language in Mnemonics (e.g : ADD , SUB ,DIV, STR, LOAD)which can be understand by computer hardware. It is very difficult to understand by the user. Each type of CPU has its own machine language depending on the hardware.

High level language is those which are used by programmers to create computer instructions, coding the programs and to develop the software's. These are closer to human languages and are combination of natural language like English & some mathematical expressions and Language Syntax and rules. These are easier to understand, write and maintain as compared to low level languages. **Example: Ada, Pascal, FORTRAN, C, C++, Java, C #, COBOL.**

Q4.What is Arithmetic Logic Unit in CPU?

Ans: ALU performs two types of operations—Arithmetic & logic. Arithmetic operations are like addition, subtraction, division trigonometrically functions (Sine, Cosine, etc.), square roots.

Logical operations involve Boolean Logic like: AND, OR, NOT.

Q5. What is ann Operating System(OS) in computer terminology ? Mention any five types of Operating System.

Ans: (i) An operating system is a system software which works as an interface between the user and hardware of computer system.

(ii) OS is responsible for managing and coordinating various activities and resource sharing within the given limit of computer. Example : Microsoft windows, Linux, Android, Mac OS, Symbian etc.

Various types of OS are as follows :

- (i) Real time OS (ii) Multi user and single user OS (iii) Multitasking OS (iv) Multi Programming OS
- (v) Distributed OS (vi) Network OS
- (vii) Embedded OS.

Q6. What are language translators? Give examples.

Ans: Translator is a program that converts a code written in one language to another language.

Example : Compiler , Assembler , Interpreter. It also checks and points the errors in a program. Compiler resides in the main memory during compilation of a code. It has a separate editor in which user types the program.

Ques 7. What are the differences b/w compiler and interpreter?

Ans :

S.NO	COMPILER	INTERPRETER
------	----------	-------------

1.	Entire program is checked for errors and statement in a one go, lists all the error statements, converts into machine code.	Errors are checked statement by statement and interpreter converts into machine code and executes it.
2.	Does not stop when an erroneous statements is encountered.	Stops when an erroneous statements is encountered
3.	Separate command is issued to execute Statement.	Executes the statement after converting a non erroneous data of program.
4.	Generally a separate editor is used to enter the program.	A built in editor is available.
5.	Compiler resides in the main memory during the process of compilation only. Hence storage is not wasted.	Interpreter should be raised in main memory throughout the process of checking for errors.

Ques 8. Define Software. Differentiate between system software and application software.

Ans : Software is basically a collection of programs , where as programs are set of instructions which are stored electronically in computer system. Example : Adobe photo shop, railway reservation software, anti virus software, gaming softwares, operating system.

S.No	System Softwares	Application Softwares
1.	These are low level programs that interact with computer at very basic level.	These are high end softwares used to solve specific problems
2.	Mostly written in assembly language, C	These are written in C , C++, Java, Dot Net, Android
3.	Some utility programs like device drivers are also system softwares. These are generally hardware specific.	These are portable to any type of platform/Operating system
4.	Example : Operating system, Device drivers, compilers, linker , loader.	MS Office, Adobe photo shop, Gaming softwares, Android apps, reservation sites.

Ques 9. What are the different data types?

Ans: In computer programming, a **data type** is a classification identifying one of various types of data, such as floating-point, integer, or Boolean, that determines the possible values for that type; the operations that can be done on that type; and the way the values of that type are stored.

(a) **Integer** : In more common parlance, whole number; a number that has no fractional part.

e.g : 23, 678, -18 etc.

(b) **Floating-point** : A number with a decimal point. For example, 3 is an integer, but 3.5 is a floating-point number. Another representation of floating point constant is 3.5×10^{-3} . Here **3.5 is a mantissa and 10^{-3} is exponent.**

(c) **Character (text)**: Readable text . E.g : A, b , Z, c , d.

Ques 10. Mention some Characteristics of computer system.

Ans : (i) **Speed** : Computers can perform millions of operations per second. Speed is given in nanoseconds

and pico seconds.

- (ii) **Accuracy:** A computer is very fast, reliable, and robust electronic device. It always gives accurate results, provided set of instructions are correct data and set of instructions to it.
- (i) **Automation :** These are used to automate any type of mechanical devices, or day to day problems.
- (ii) **Versatile :** Computer systems are flexible towards the changing technology, hardware and softwares. They also can be used as personal computers, for scientific purpose, for weather forecasting, for military operations and business analysis too.
- (iv) **Memory :** Like human beings computers can also store large set of information and data fetched from real world. Memory is used to store information. Primary and secondary memory are designed inside the system.

Long Questions and Answers

Ques 11. What are tokens in C programming language? Describe with examples. Ans: In C programming tokens are defined as the smallest unit of a program which is indivisible. Program is combination of these C tokens. There are following six types of token categories :

- (i) **Keywords:** These are set of fixed or reserved words that can not be used as an identifier or variable. E.g: printf, scanf, gets, stdio, clrscr, int, float, char, signed, break, for, while etc.
- (ii) **Identifiers:** These are used to identify the data and other objects in a program. These are names given to variables, array, functions consisting of alphabets, numerals and underscore. Identifiers never start with a number. E.g : a, ab, akash_cse, ab123.
- (iii) **Constants :** Those data objects which are fixed and values can't be changed.
E.g : Integer constant : 1, 34, 257, -675 etc.
Floating constant : 0.02, 3.145159, 2.14E-3 etc.
Char constant : A single character enclosed in a single quotes. E.g : 'a', '&', 'x' etc. In computer characters are stored using machine's character sets in ASCII form.
- (iv) **String constants :** Sequence of characters stored within double quotes. E.g : "abc", "xy67", "computer" etc.
- (v) **Special characters :** @, \$, &, !, }, % etc.
- (vi) **Operators :** this is a symbol that specifies the mathematical, logical, or relational operation to be performed .e.g : =, -, &&, /, =, == etc.

Ques 12. What is an algorithm? What is its purpose? Mention the characteristics of algorithm. Write an

algorithm to print the first 10 natural numbers. Also draw its flow chart.

Ans : Algorithm is a formally defined procedure for performing calculation. If a procedure is formally defined then it must be represented in a formal way i.e programming language. The algorithm gives step by step solution to reach some goal of a problem or to find a solution of existing problem. Purpose of algorithms is to implement re-usability in software applications. Once we have designed the overview of solution to any problem in algorithmic form it can be coded in any language like C , C++,Java etc.

Pseudo Code : Algorithms can be represented in the form of pseudo code and flow chart both. Pseudo code is the way to write any algorithm using some formal notations of mathematics, programming syntax and English words step by step.

Characteristics of algorithm are as follows:

- (i) Algorithm must be precise and simple.
- (ii) Algorithm must be reusable.
- (iii) Algorithm must be written in finite number of steps.
- (iv) Algorithm must be unambiguous. It should not represent more than one meaning or for same input there should not be different output.
- (v) Algorithm must be effective and should be terminated easily in minimum time .

Control Structures used in algorithms are :

- (i) Sequence : Each step of algorithm is executed in specific order one after another.
- (ii) Decision : When the output of a process depends on certain decision criterion, then Decision statements are used. E.g : if $x = y$, then print " EQUAL". Decision statements may either evaluate to TRUE or FALSE.
- (iii) Repetition: This involves execution of one or more steps for a number of times repeatedly. Programming constructs such as while, do-while and for loops are used.

Algorithm to Print the First N natural numbers

Step 1: [initialize] set $I = 1$, $N = 10$

Step 2 : Repeat steps 3 and 4 while $I \leq N$

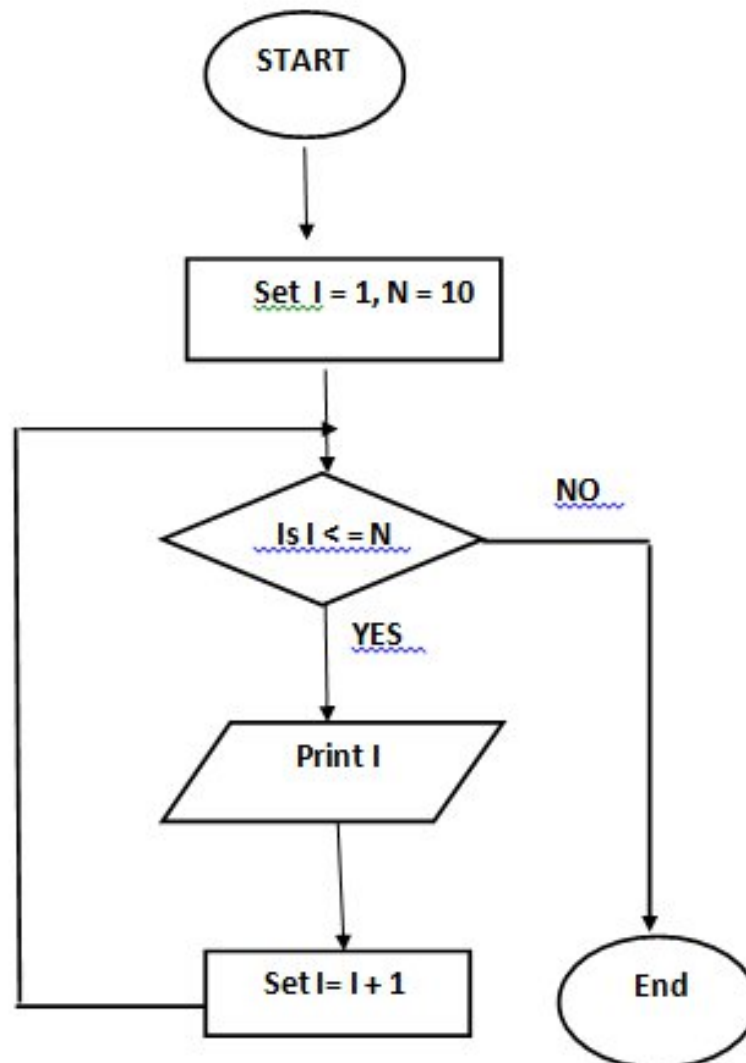
Step 3 : Print I

Step 4 : Set I = I + 1

[End of loop]

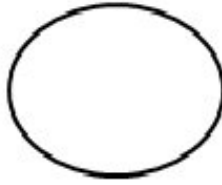
Step 5 : End

Flow Chart



Ques 13. What are the symbols used in flowchart?

Ans: A flow chart consists of various boxes with different shapes connected by flow lines. Flow lines have arrows that tell the direction of command or data flow between the boxes. Symbols used are as follows



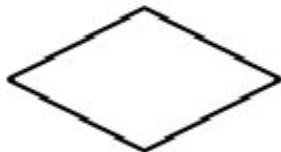
Oval: Represents Start / End point of the



Rectangle: Represents Process of an algorithm



Parallelogram: Represents Input/ Output of Algorithms.

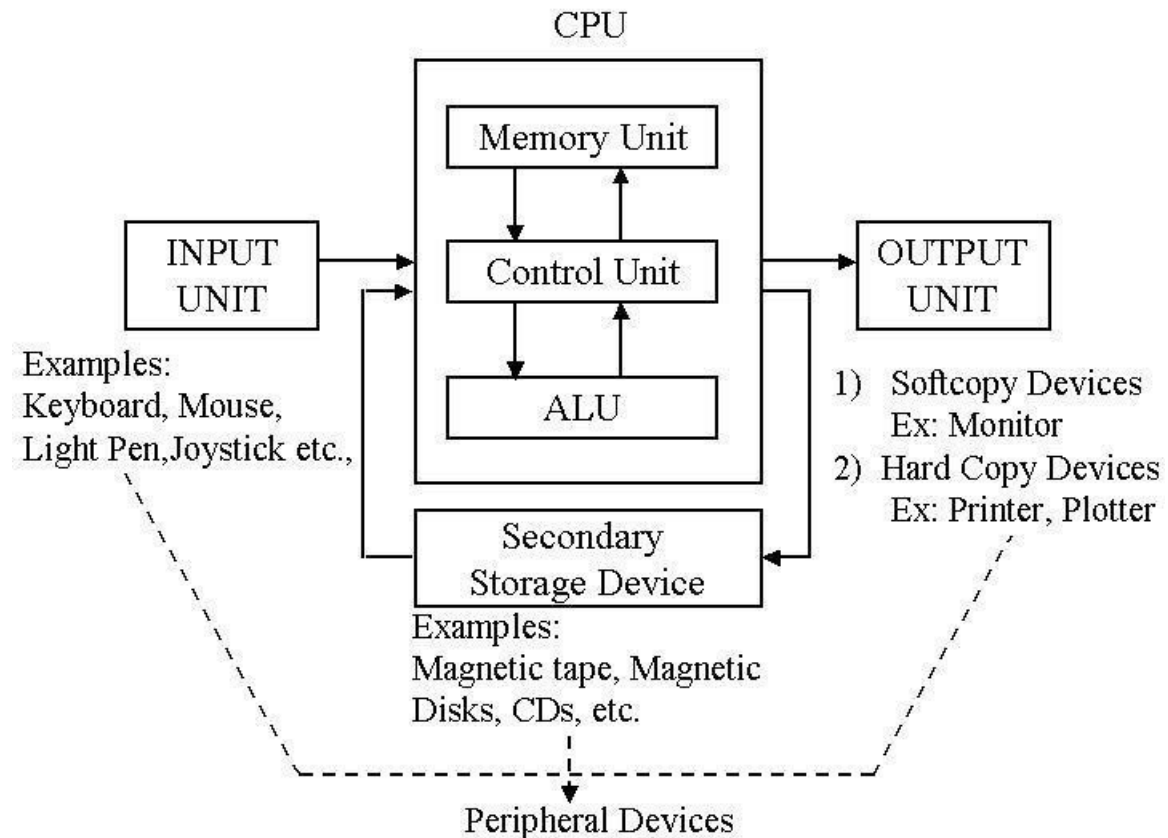


Diamond: Represents Decision Making / Conditions in



Arrow: Connector Between different shapes showing relationships in algorithm

Ques 14. Draw the block diagram of computer systems .What are major functions of a computer. Also explain its components.



Major functions of a computer are as following :

- (a) Accepting the data or(input).
- (b) Storing the data.
- (c) Processing the data.
- (d) Display results (output)
- (e) Controlling and coordinating all the operations inside the computer.

Components of a computer system

- (i) **Input :** This is the process of entering data and instructions /programs into the computer system with the help of input devices (e.g : Keyboard, Mouse, Microphone etc.)Input device converts the input data into binary code (0 or 1) for further processing.
- (ii) **Storage :** Storage is the process of saving and storing the data and information in computer memory. It also stores intermediate results.

(a) **Primary Storage** : This is also called main memory, which is directly accessible by the CPU at very high speeds. It is used to store the data and parts of programs, the intermediate results of processing and current job results.

Primary memory is volatile in nature i.e. as soon as computer is switched off, data is erased/lost from the memory. It is also expensive E.g : RAM (random access Memory)

(b) **Secondary storage** : Also known as auxiliary memory. This is non-volatile memory and data is permanently stored in this. E.g : magnetic Disks, CD, DVD, Hard Disk, Pen Drives, Memory Cards etc.

Central Processing Unit (CPU) : This is also called processor which is a chip inside the cabinet of computer system. It is referred to as the brain of computer. It carries out instructions of a computer program by performing basic arithmetic, logical, control and I/O operations specified by instruction. Speed of CPU is measured in Giga Hertz (Ghz).

(iii) **Control Unit** : A control unit handles all the processor signals. It directs all the input and output flow, fetches instructions from microprograms and directs it to other units of computer by timing and control signals. Control unit takes care about correct execution of instructions.

(iv) **ALU : Arithmetic and logic unit** is used to perform arithmetic operations and logical operations. Arithmetic operations include addition, subtraction, division, multiplication, trigonometrical operation (like Sin, Cos etc)
Logical operations consist of Boolean operations that predict either TRUE/1 or FALSE/0.
Example : logical AND, NOT, XOR, OR etc.

(v) **Output** : This consists of operation that provides the result regarding certain input and sends data from computer to another device provided in the system.
O/P peripheral devices are printers, speakers, monitors, projectors, headphones etc.

Ques 15. Write Short Notes on the following:

- (i) **Top down programming** (ii) **Structured Programming**
(iii) **Syntax and logical errors** (iv) **Linker & Loader**

Ans : (i) Top down programming : In this approach of programming a system is break down into smaller sub systems , so that we can see more detail concept of modules. In a top down approach an overview of the system is firstly formulated , then specifying but not detailing any first level sub systems.

Each sub system is then refined into more primary and smaller sub systems. This is called modularization. Modularization makes the program to understand , write and debug.

Structured Programming : This implements logical structure on the program to be written. Thus program becomes more efficient, simple and easy. This allows program loaded into memory and to be reused .Modules are coded separately and tested. After that all the modules are combined together to make larger sub system. **Language like C , Pascal support structured programming. Each module has its own local data and performs specific task.**

Syntax and logical errors : When compiler compiles the program then due certain wrong inputs and wrong codes errors occur. Syntax Error occurs when code or program has grammatical mistakes due to mistyping. Like missing parenthesis, not using ; etc.Syntax error leads to no output.

Logical Error are those which give the output which is not required for certain inputs due to wrong interpretation of algorithm and logic in code.

Loader : This is an Operating system utility that copies programs from a storage A storage device to main memory, where they can be executed.

Absolute Loader : Loads the object program from translation time address and transfers control to it.

Bootstrap loader : This loader is responsible for loading the OS when system is turned ON and transferring control to it. It is present in ROM area of main memory.

Linker : Also called link editor / binder. It is a program that combines object modules to form an executable program. This simplifies the designing of large softwares because they can be divided into smaller modules and afterwards can be combined using linker.

Ques 16. What is the structure of C program ? What are header files and their uses. Explain formatted input output functions.

Ans : **Structure of C Language program**

- 1) Comment line
- 2) Preprocessor directive
- 3) Global variable declaration
- 4) main function()

```

{
Local variables;
Statements;
}
User defined function
}
}

```

Comment line

It indicates the purpose of the program. It is represented as

```
/* .....*/
```

Comment line is used for increasing the readability of the program. It is useful in explaining the program and generally used for documentation. It is enclosed with the decimeters. Comment line can be single or multiple line but should not be nested. It can be anywhere in the program except inside string constant & character constant.

Preprocessor Directive:

#include<stdio.h> tells the compiler to include information about the standard input/output library. It is also used in symbolic constant such as #define PI 3.14(value). The stdio.h (standard input output header file) contains definition & declaration of system defined function such as printf(), scanf() etc. Generally printf() function used to display and scanf() function used to read value

Global Declaration:

This is the section where variable are declared globally so that it can be access by all the functions used in the program. And it is generally declared outside the function.

main()

It is the user defined function and every function has one main() function from where actually program is started and it is enclosed within the pair of curly braces. The main() function can be anywhere in the program but in general practice it is placed in the first position.

Syntax :

```
main()
{
.....
.....
.....
}

```

The main() function return value when it declared by data type as

```
int main()  
{ return 0}
```

The main function does not return any value when void (means null/empty) as

```
void main(void ) or void main()  
{  
printf ("C language");  
}
```

Output: C language

Example : **/*First c program with return statement*/**

```
#include <stdio.h>  
int main (void)  
{  
printf ("welcome to c Programming language.\n");  
return 0;  
}
```

Output: welcome to c programming language

Header Files : When we work with large projects, it is desirable to separate out some procedures from main() function of the program. Some procedures need to be used several times in different programs. So one solution is to copy the code of that procedure from one program to another several times , which results in waste of time.

So another solution is to make procedures and store them in a different file called header files. System defined Header files contain the definitions of Library functions like printf(), scanf(), getch() , puts(), pow(), clrscr() etc. ***Conventionally , header files ends with a ‘ . h’ extension and names can use only letters, digits, dashes , and underscores. Header files can be system define in compiler it self or programmers can design their own header files too.***

Examples : **stdio.h** : standard input & output (functions like printf() ,scanf() are defined.)

conio.h : console input & output. (getch(), clrscr () are defined.)

string.h : for string handling functions. ()

math.h : Math functions like pow(), sin(), log() , sqrt() are defined.

stdlib.h : Library functions like exit (),

Foramttd Input & output functions : C language supports two formatting functions

printf () and scanf (). **Printf()** is used to convert text data stored in the program onto text stream for o/p to the monitor. **Scanf** is used to convert text stream coming from keyboard to data values and stores them in program variables. These are called as **formatted Input and output functions** because they have special format to read and write The text from streams and then converting them into binary stream.

Syntax of printf() : printf (“ control string” , var1, var2var n).

Printf (“Hello welcome to C program”)

Printf (“ %d %d %f” a , b , c), where a and b are integer variables and c is float variable.

%d and %f are known as format specifiers.

Syntax of scanf() : scanf (“ Control string” , & var1, & var2,.....,& var n).

Format specifiers : For integer variables ----- %d

For float variables----- %f

For character variables ----- %c

Ques 17. What are storage classes? Explain their types briefly.

Ans : **Storage class in C** decides the part of storage to allocate memory for a variable.

- It also determines the scope of a variable. All variables defined in a C program get some physical location in memory where variable's value is stored.
- Memory and CPU registers are types of memory locations where a variable's value can be stored.
- The storage class of a variable in C determines the **life time** of the variable if this is '**global**' or '**local**'.
- Along with the life time of a variable, **storage class** also determines variable's storage location (memory or registers), the **scope (visibility level) of the variable**, and **the initial value of the variable**.
- **There are four storage classes in C those are automatic, register, static, and external.**

(i) **Automatic Storage Class:**

- This is the default storage class for all the variables declared inside a function or a block.
- Auto variables can be only accessed within the block/function they have been declared and not outside them (which defines their scope).
- They are assigned a garbage value by default whenever they are declared.
- Variables having automatic storage class are local to the block which they are defined in, and get destroyed on exit from the block.

```
#include <stdio.h>
int main()
{
    auto int i = 1;
    {
        auto int i = 2;
    }
    {
        auto int i = 3;
    }
    printf ( "\n%d ", i);
}
```

```
}
printf ( "%d ", i);
}
printf( "%d\n", i);
}
```

OUTPUT

3 2 1

(ii) **Register Storage Class:**

- The **register specifier** declares a variable of register storage class.
- Variables belonging to register storage class are **local to the block in which they are defined**.
- Variables are placed in CPU registers, not in memory.
- Register variables are also given no initial value by the compiler.

(iii) **Static Storage Class:**

- The **static specifier** gives the declared variable static storage class.
- Static variables are **not visible outside their function or file**, but they maintain their values between calls.
- Static variables can be used within function or file.
- Static variables have **default initial value zero** and initialized only once in their lifetime.

<pre>#include <stdio.h> void staticDemo() { static int i; { static int i = 1; printf("%d ", i); i++; } printf("%d\n", i); i++; }</pre>	<pre>int main() { staticDemo(); staticDemo(); } OUTPUT ===== 1 0 2 1</pre>
--	--

(iv) **External Storage Class:**

- The **extern specifier** gives the declared variable external storage class.
- Variable is declared with **external linkage elsewhere** in the program.
- When **extern** specifier is used with a variable declaration then **no storage is allocated to that variable**.
- It is assumed that the variable has already been defined elsewhere in the program.

When we use **extern** specifier the variable cannot be initialized because with **extern** specifier variable is

declared, not defined.

```
#include <stdio.h>
extern int x;
int main()
{
    printf("x: %d\n", x);
}
int x = 10;
```

UNIT-2

Module - 2 : (Arithmetic expressions & Conditional Branching)

- **Arithmetic expressions and precedence**
- Operators and expression using numeric and relational operators
- **Mixed operands, Type conversion**
- Logical operators, Bit operations
- **Assignment operator, Operator precedence and associativity.**

Conditional Branching:

- ~~Applying if and switch statements~~
- Nesting if and else

- Use of break and default with switch.

Short Question & Answers

Ques 1 What are C expressions? Give examples.

Ans : An expression is a combination of variables, constants, operators and function call. It can be arithmetic, logical and relational for example:-

```
int z = x+y
// arithmetic expression
```

```
a>b          // relational
a==b        // logical
func(a, b)   // function call
```

Expressions consisting entirely of constant values are called *constant expressions*. But if i were declared to be an integer variable, the expression $180 + 2 - j$ would not represent a constant expression.

Ques : 2 Define the term C operator. Explain Arithmetic and Logical Operators.

Ans: An operator is a symbol which helps the user to command the computer to do a certain mathematical or logical manipulations. Operators are used in C language program to operate on data and variables.

(iii) Arithmetic Operators

All the basic arithmetic operations can be carried out in C. Example : +, -, %, /, *, &&, etc. Both unary and binary operations are available in C language. Unary operators operate on a single operand, where operand can be a constant, a variable or a valid C expression. Binary operators operate on two operands. Ternary operators operate on three operands.

Unary Operators: +a (unary plus), -a (unary minus), ++a (pre increment), !a (logical NOT).

Binary Operators: a+b (binary addition), a-b (binary subtraction), a*b (multiplication),

a / b (division), a % b (Modulus division), x >= y (relational operator)

Ternary Operator: This is also called conditional operator (? :)

Test expression ? expression 1 : expression 2

Logical Operators: C has the following logical operators; they compare or evaluate logical and relational expressions. The output evaluated is either a true value or false value of an expression. Represented as 0 (false), 1 (True).

A || b is Logical OR,

a && b is Logical AND, !a is Logical NOT

Ques 3. What is the difference between “=” and “==”?

Ans: = (**assignment operator**) in C language is used to assign the value of right-hand side value/variable/expression to the left hand side variable .e.g: a = 4; here constant 4 at R.H.S is called r-value and identifier a at the L.H.S is called l-value

== (**Equality Operator**) in C language is used to check the value of left hand variable/expression with the right hand Variable/expression. Whether the two values are equal or not. It returns true if these are equal else it will return false.

e.g. (2==3) : It means whether 2 is equal to 3 or not which is false so output is 0.

(3==3) : It mean 3 is equal to 3 ,which is true so output is 1

Ques 4. What is the output of arithmetic statement, $S = 1/3 * a/4 - 6/2 + 2/3 * 6/g$?Given

that (int a = 4 , int g = 3 , assume s to be an int type).HINT: apply the concept of type casting.**

Ans : $S = 1/3 * a/4 - 6/2 + 2/3 * 6/g$
 $= 1/3 * 4/4 - 6/2 + 2/3 * 6/3$
 $= 0 * 4/4 - 6/2 + 2/3 * 6/3$
 $= 0/4 - 6/2 + 2/3 * 6/3$
 $= 0 - 3 + 0 * 2$
 $= -3$

Ques 5 . Give the output of following code :

```
void main()
{ float u = 3.5 ;
  int v , w , x , y ;
  v = ( int ) ( u + 0.5 ) ;
  w = ( int ) u + 0.5 ;
  x = ( int ) ( ( int ) u + 0.5 ) ;
  y = ( u + ( int ) 0.5 ) ;
  printf ( “ %d %d %d %d ” , v , w , x , y ) ;
}
```

Ans : Output is : 4 3 3 3

Ques 6. What is the difference between the pre-increment operator and the post-increment Operator? Give example also.

Ans. (i) The pre-increment operator increases the operand's value by 1 first, and then returns the modified Value.

(iv) The post-increment operator stores a copy of the operand value in a temporary Location and then increases the operand value by 1. For example given $x = 1$, the $++x$ expression returns 2, while the $x++$ expression returns 1.

EXAMPLE: What is the output of the following C code?

```
main()
{
    int i=5,;
    printf(“ %d %d %d %d %d ”, i++, i--, ++i, --i, i);
}
```

Ans: 4 5 5 4 5

Ques 7. Find the output of the following program?

```
main()
{
    int x=100;
    printf(“%d”,10 + x++);
    printf(“%d”,10 + ++x);
}
```

Ans. 110 112

Ques 8. What is the output of the following C code?

```
main()
{
    int i = 0 , j = 1 , k =2 , l ;
    l = i && j++ && ++k ;
    printf( “ %d %d %d %d ” , i , j , k , l );
}
```

Ans: Output : 0 1 2 0

Ques 9. What will be the value of x when the following segment is executed?

```
int x=10, y=15;
x=(x<y) ? (y+x) : (y-x);
```

Ans. Output – 25

Ques 10. What is the dangling else problem?

Ans: When a program contains more number of if clauses than else clauses, then there exists a potential ambiguity regarding with which if clause does not match with else clause. This ambiguity is called *Dangling else problem*

Ques 11. What is the precedence and associativity of operators?

Ans : **Precedence :** When an expression has more than one operators then relative priorities of operators with respect to each other is checked to evaluate expression in that order.

E.g : Multiplicative operators (* , / , %) have high priority than additive operators (+ -).

Note : If all the operators in same expression have equal priority then operator which comes first from left is evaluated first and then we move towards right .

Example : In expression $2 / 3 * 6 \% 4$, all are multiplicative operators, hence have same precedence. So first / is evaluated , then * is evaluated and then % division is performed.

Associativity: This defines the direction in which the operator acts on the operands. It can be either left to right or right to left.. First precedence is checked then associativity is applied.

++ (prefix /postfix)	Right to left
*, / , %	Left to right
< , <= , > , >=	Left to right
= assignment	Right to left
&& , 	Left to right.

Long Questions & Answers

Ques 12. Describe the bitwise operators available in C with examples.

Ans: These operators work with bits rather than larger structures such as a byte. For example, each of the eight bits in a byte can be used as an individual flag to signal yes/no, on/off (1 or 0) about some condition. The Boolean operators AND, OR and NOT also deal with individual bits rather than bytes. Bitwise operators are programming commands that work with individual bits. The primary ones are.

Symbol Function

- << Shift left
- >> Shift right
- & bitwise AND
- | bitwise OR
- ^ bitwise XOR (Exclusive OR)
- ~ bitwise NOT (0 to 1; 1 to 0)

Examples : (i) int a = 10 , b = 20 , c = 0;

C = a & b ;

Here compiler will first convert int variables a and b in binary form , then bit wise AND is performed.

(ii) **Shift operator** shift the bits either to left or right. The syntax of shift operation can be given as:

Operand op num , Where the bits in operand are shifted left or right depending on the operator

(left if operator is << and right if operator is >>) = 00011101

X << 1 produces 00111010

Shifting once to the left multiplies the number by 2. Hence multiple shifts of 1 to the left result in multiplying by 2 several times. If x = 0001 1101 then x >>4 produces 0000 0001. Shifting from right by 1 bit divides the number by 2.

Ques 13. What is meant by type conversion and type casting? Why it is necessary? Explain about implicit and explicit type conversion with examples.

Ans : When any C expression or statement involves different data types e.g : multiplication of a float number by integer number , then to get the correct result in terms of values *Type conversion / Type casting is required in programming*. In this we change a variable of one data type to variable of another data type.

** Type conversion is done implicitly by the compiler

** Type casting is performed explicitly by the programmer.

(d) **Type conversion** : It is done when expression has variables of different data types. To evaluate the expression , data type is promoted from lower to higher data type (or from higher to lower).


Type conversion is automatically performed.

e.g : float x ;

int y = 3;

x = y ; // Here x will have the value 3.0 as automatically integer value is converted to its equivalent floating point value.

Conversion hierarchy of data types is as shown below :

long double	Higher level
double	
Float	
Unsigned long int	
Long int	
Unsigned int	
Int	
Short , char	

** When a char type is operated with an int type char is promoted to int.

** When a float type data is operated with an int , then int is promoted to float.

** When any operand is double then another operand is also converted to double

(e) **Type casting** : It is also called as forced conversion. It is done when value of higher data type is to be converted to lower data type. This conversion is under programmer's control , not in compiler's control.

Demotion : float f = 3.5 ;

int I ;

I = f ;

Statement I = f results in f to be demoted to type int , i.e. the fractional part of f will be lost and I will contain 3 not 3.5 ; this is called down conversion/demotion. But result is not correct so explicit type

casting is required. **Explicit type casting is done by placing the target data type in parantheses followed by the variable name that has to be converted.**

Variable 2 = (data type) variable1

e.g : float salary = 10000.00 ;

int tot_sal =0;

sal = (int) salary

** When floating point numbers are converted to integers , the digits after decimal are truncated.

Ques 14. What is conditional branching statement ? Explain switch case statements in detail , with a proper example.

Ans : The conditional branching statements help to jump from one part of the program to another depending whether a particular condition is true or false. These decision control statements include :

(vi) **if statement (ii) if – else statement (iii) if – else-if statement (iv) switch statement.**

A switch case statement is a multi-way decision statement that is simplified version of if – else block. Staement blocks refer to statements lists that may contain zero or more statements. These statements ar not enclosed within opening and closing braces.

Switch statements are mostly used in following situations :

- **When there is only one variable to evaluate in the expression.**
- **When many conditions are being tested for.**
- **Switch case is preferable over if –else if many conditions are to be tested.**

Syntax of switch – case statement

Switch (int expression/ int variable)

{

Case value1 :

Statement Block 1 ;

Break ;

Case value 2 :

Statement block 2;

Break;

.....

Case value N :


```

Statement block N ;
Break ;
Default :
Statement block D ;
Break ;
}
Statement x ;

```

Switch case compares the value of integer expression/variable with the value of each case statement that comes in that case. If value of switch expression and case number matches then that statement block is executed.

Default Case : When switch expression does not matches with any case value , then statements in default

case are executed. (default case is optional).

Use of break in switch case : The break statement must be used after each case statement in ideal conditions. If it is not used then the case which is matched and all the cases following that case are executed resulting in logical error or incorrect output.

Some Rules of Switch – Case statements

- Control expression inside switch must be integer type variable or integral expression
E.g : switch (int n) , switch (int x + 6) etc.
- Each case label should be followed by a constant or a constant expression.
- Every case label must have unique constant expression value.
- Case labels must end with a colon.
- Default label can be placed anywhere in switch block. There must be only one default in switch program.
- case labels should not be logical or relational expressions.
- Order in which case labels are written are not fixed.

Example 1 of switch case with break	Example 2 of switch case without break
<pre> # include <stdio.h> # include <conio.h> Void main() { char grade = 'C' 'switch (grade) { Case 'O' : Printf ("\n Outstanding "); </pre>	<pre> # include <stdio.h> Int main() { Int option =1; Switch (option) { Case 1 : printf (" I am in case 1\n"); Case 2 : printf (" I am in case 2\n"); </pre>

```

Break ;
Case ' A' :
Printf ( "\n Excellent " );
    Break ;
                Case 'B' :
Printf ( "\n GOOD " );
    Break ;
                Case 'C' :
Printf ( "\n Satisfactory " );
    Break ;
                Case 'F' :
Printf ( "\n Fail " );
    Break ;
                Default :
Printf ( "\n Outstanding " );
    Break ;
    } getch (); } // End of main()
*****

```

Output : Satisfactory

```

Default : printf ( " I am in case default " );
    }

```

```

return 0;
getch();

```

```

}

```

Output : I am in case 1
I am in case 2
I am in case default.

Ques 15. What is the meaning of statement in C language. Briefly explain its classification with example.

Ans : A statement is the smallest logical unit that can independently exist in a C program. No entity smaller than statement (i.e variables, expressions, constants etc) can exist independently in a program.

Example : $a = b + 5$ is an expression , but when it is terminated by a semicolon , like $a = b + 2 ;$ then it becomes a assignment statement. It can be also called arithmetic statement , because operator + is also associated here.

Classification of C statements : (a) Based upon the type of action they perform.

(iv) Based upon the number of constituent statements.

(v) Based upon their role.

Based upon the type of action they perform

5) Non executable statements 2. Executable statements

Non executable statements tell compiler how to build a program, how to allocate memory, interpret and compile other statements. A non-executable statements can affect only statements below them , because compiler starts scanning from top to bottom.

Examples : Prototype declarations, global variables, preprocessor directives , header files.

Executable statements: These represent the instructions that are to be performed when the code is executed. Executable statement appears only inside the function body. *Example : looping statement, branching statements, assignment statements .*

Based upon the number of constituent statements

(i) Simple statement 2, Compound statement

Simple statements are those which have only single statement, terminated by a semicolon.

Example : `int a = 10; // declaration statement`

`a = a + 10 // assignment statement.`

`printf(“ Welcome to the concepts of C ”); // print statement.`

A compound statement contains sequence of more than single statements enclosed within a pair of curly braces {}. Example :

```

{
    int x = 10;
    x = x + 3 ;
    ++x ;
    printf( “ %d” , x ) ;
}

```

A compound statement is also called as block. Compound statement can be empty also, i.e nothing written inside the braces. E.g : { }. A compound statement need not to be terminated by a semicolon after braces.

Based upon their role

➤ Declaration statement and Definition statement.

Example: `int a ;` is a declaration. But if we assign some value to this variable it becomes definition of this variable. E.g : `int a = 30 ;`

➤ Null statement and expression: Null statements are those which contain only semicolon, nothing else. Expression is `2 + 3 / 5`.

➤ Flow control statements: The order in which control flow occurs in the program is called flow of program control. Program control flows sequentially from top to bottom. These are of two type :

- Branching Statement: (a) Selection Statements (b) Jump statements.
- Iteration Statements

Examples : Selection statements : if statement, if-else statement, switch statement.

Jump Statements : break ,continue, return , goto.

Iteration Statements : for loop, while loop, do-while loop.

Important Programs

<p>1. Program to find the simple interest, given principle, rate of interest and time.</p>	<p>2. Ramesh's basic salary is input through the keyboard. His dearness allowance is 40% of basic salary, and house rent allowance is 20% of basic salary. Write a program to calculate his Gross Salary.</p>
<pre>#include <stdio.h> #include <conio.h> void main() { float p, r, si; int t; clrscr(); printf("Enter the values of p,r and t\n"); scanf ("%f%f%d", &p, &r, &t); si = (p * r * t)/ 100.0; printf ("Amount = Rs. %5.2f\n", p); printf ("Rate = Rs. %5.2f%\n", r); printf ("Time = %d years\n", t); printf ("Simple interest = %5.2f\n", si); getch() ; }</pre>	<pre>#include<stdio. #include<conio.h> void main() { float bs , da= 0.0, hra=0.0 ,gross=0.0 ; clrscr() ; printf("Enter the basic salary\n") ; scanf("%f" ,&bs) ; printf("BASIC SALARY OF RAMESH=%f\n ", bs) ; da = (bs*40) / 100 ; hra = (bs *20) /100 ; gross = bs + da + hra ; printf("*****"); printf("GROSS SALRY OF RAMESH = % f", gross) ; getch() ; }</pre>

3. Program to find the area of a circle, given the radius.	4. Program to find the area of a triangle, given three sides a, b and c.
<pre>#include <stdio.h> #include <conio.h> #include <math.h> #define PI 3.142 void main() { float radius, area; clrscr(); printf("Enter the radius of a circle\n"); scanf ("%f", &radius); area = PI * pow (radius,2); printf ("Area of a circle = %5.2f\n", area); getch() ; }</pre>	<pre>#include <stdio.h> #include <conio.h> #include <math.h> void main() { float a, b, c ; float s =0.0 , area =0.0 ; clrscr(); printf("Enter the values of a,b and c\n"); scanf ("%f%f%f ", &a, &b, &c); s = (a + b + c) / 2; /* computes perimeter */ area = sqrt (s * (s-a) * (s-b) * (s-c)); /* compute the area * printf ("Area of a triangle = %f\n", area);</pre>

5. Program to swap the values of two variables without using third variable.	6. Program to multiply a given number with 2 ⁿ , without using multiplication operator.
<pre>#include<stdio.h> #include<conio.h> void main() { int a , b ; clrscr() ; printf("Enter the values of a and b \n") ; scanf(" %d %d", &a , &b) ; printf("Values variables before swapping are a=%d \t b=%d: \n", a , b) ; a = a + b ; b = a - b ; a = a - b ; printf("Values of a and b after swapping are:\n") ; printf("a=%d\t b=%d", a , b) ; getch(); }</pre>	<pre>#include<stdio.h> #include<conio.h> void main() { int num , n , result ; clrscr() ; printf("Enter the number to be multiplied \n") ; scanf(" %d", &num) ; printf("Enter the value of n \t") ; scanf(" %d", &n) ; result = num << n ; printf("Result of multiplication = %d", result); getch() ; }</pre>

<p>7. Program for finding the sum of a five digit no.</p>	<p>8. . Write a C program to check whether a given integer is odd or even.</p>
<pre>#include<stdio.h> #include<conio.h> void main() { long int num , sum =0 , rem ; // Max. five digit no. can be 99999, so long int declared clrscr() ; printf("Enter the five digit number\n"); scanf("% ld" , &num) ; printf("The five digit number=%ld\n", num) ; rem1 = num%10 ; num = num /10 ; rem2 = num%10 ; num = num /10 ; rem3 = num%10 ; num=num/10 ; rem4 = num%10; num=num/10 ; sum = rem1 + rem2 + rem3 + rem4 + num ; printf("Sum of the digits of %ld =% ld", num ,sum) ; getch(); }</pre>	<pre>#include <stdio.h> #include <conio.h> void main() { int x , remainder; clrscr(); printf("Enter an integer :"); scanf ("%d", x); remainder = x % 2; if (remainder == 0) { printf ("%d, is an even integer\n", x); else printf ("%d, is an odd integer\n", x); } getch (); } /* End of main() */</pre>

<p>9. Write a C program to find the largest of three numbers.</p>	<p>10. Program to determine whether a triangle is right angle , scalene, equilateral or isosceles where the Sides a , b and c are entered by the user. Also check whether the triangle is possible or not?</p>
<pre>#include <stdio.h> #include <conio.h> #include <math.h> void main() { int a, b, c; clrscr(); printf("Enter the values of a,b and c\n"); scanf ("%d %d %d", &a, &b, &c); printf ("a = %d\tb = %d\tc = %d\n", a,b,c); if (a > b) { if (a > c) { printf ("A is the greatest among three\n"); } else { printf ("C is the greatest among three\n"); } } else if (b > c) { printf ("B is the greatest among three\n"); } else { printf ("C is the greatest among three\n"); } }</pre>	<pre>#include<stdio.h> #include<conio.h> void main() { float a , b , c ; clrscr() ; printf ("Enter the sides of the triangle : \n") ; scanf(“%f%f%f” , & a , & b , &c) ; if ((a + b < c) (b + c < a) (a + c < b)) printf (“Triangle is NOT POSSIBLE”) ; if ((a * a == b * b + c * c) (b * b == a * a + c * c) (c * c == a * a + c * c)) ; printf(“ Triangle is RIGHT ANGLED”) ; if ((a == b) && (b != c)) ; printf(“Triangle is ISOSCELES”) ; if ((b == c) && (c != a)) ; printf(“Triangle is ISOSCELES”) ; if ((c == a) && (a != b)) ; printf(“Triangle is ISOSCELES”) ; if ((a == b) && (b == c)) ; printf(“ Triangle is EQUILATERAL”) ; if ((a != b) && (b != c) && (c !=a)) ; printf(“ Triangle is SACLENE”) ; getch () ; }</pre>

[END OF 2nd UNIT]

UNIT-3

Iteration and loops:

- **Use of while, do while and for loops**
- Multiple for loop variables
- Use of break and continue statements.

Functions:

- **Introduction, types of functions**
 - Functions with array
 - **Passing parameters to functions**
 - Call by value and Call by reference
 - **Recursive functions**
-

Short Question & Answers (Module1 : Loops and Iterations)

Ques 1: Can we use continue statement within the body of switch statement like break?

Ans: No, a *continue* can appear only in or as a loop body. A switch statement is a branching statement is a branching statement and not a looping statement .

Ques 2. Differentiate between a while loop and do –while loop.

Ans :

S.NO	While loop	do while loop
1.	A while loop is used to execute and repeat a statement block depending on a condition which evaluates at start of the loop.	A do while loop is used to execute and repeat a statement block depending on a condition which evaluates at the end of the loop.
2.	A variable value is initialized at the beginning or before the loop and used in condition.	A variable value is initialized before the the loop or assigned inside the loop and used in condition.
3.	The statement block will not be executed when the value of the condition is false.	The statement block will not be executed when the value of the condition is false. Block is executed at least once irrespective of the value of condition.
4.	Syntax : Statement x ; while (condition) { Block of statements under execution ; } Statements y ;	Statement x ; do { Block of statements under execution ; } while (condition); Statement y;
5.	Entry controlled loop	Exit controlled loop.

Ques 3. Can the while statement end with a semicolon?

Ans: By definition, the while statement does not end with a semicolon. However, it's legal in C to put a Semicolon right after the while statement like this: **while (expression);** which means there is a **null Statement controlled by the while statement**. Remember that the result will be quite different from what you expect if you accidentally put a semicolon at the end of the while statement.

Ques 4. Write a program to generate even number series from 1 to 50 , using while loop.

```
Ans : # include <stdio.h>
      # include <conio.h>
      Void main()
      {
        int i , n ; i= 2, n= 50 ;
        while (i<= n)
          {
            printf (“ %d \n ” , i);
            i = i + 2 ; // i +=2
          } // End of while loop.
          getch();
      } // End of main()
```

Ques 5. For every use of a for loop, we can implement an equivalent while loop. So when should we prefer to use these in different programs?

Ans: A *while loop* is preferred over *a for loop* when the number of iterations to be performed is not known in advance. The termination of the while loop is based on the occurrence of some particular condition, i.e specific sentinel value. Whereas the *for loop* is preferred when no. of iterations to performed are known beforehand .

Ques 6. What will be the output of the following programs?

```
#include<stdio.h>
#include<conio..h>
void main()
{
  int c=5;
  do
  { printf( “Hello”) ;
    c++;
  } while (c<5) ;
  return 0; }
```

Ans : Output “ Hello” will be printed.

Ques 7. Explain continue statement with an example .

Ans : Continue is a keyword . When we want to take control to the beginning of loop, by ignoring the statements without their execution inside the loop , then continue statement is used.

When continue is placed inside any loop, control automatically passes to the start of loop. *Continue is generally associated with an If statement.*

```
Example : void main()
           { int i=1 ;
             while ( i<=10)
             {
               if ( i == 5)
                 continue ;
               printf(“ \t %d ” , );
               ++ i;
             } return 0;
           } // End of main
```

Output : 1 2 3 4 5 6 7 8 9 10

Ques 8.What is the output of following code snippet?

```
void main ()
{
  int i , j ;
  for ( i=1 ; i<3 ; i++)
    for ( j=1 ; j < 4 ; j++)
      { if ( j==2) continue ;
        printf (“%d %d ” , i ,j ) ;
      }
}
```

Ans: OUTPUT: 1 1 , 1 3 , 2 1 , 3 3

Long Question & Answers (Module1 : Loops and Iterations)

Ques 9. What do you mean by iteration? Explain entry and exit controlled loops with example. Give the classification of loops also.

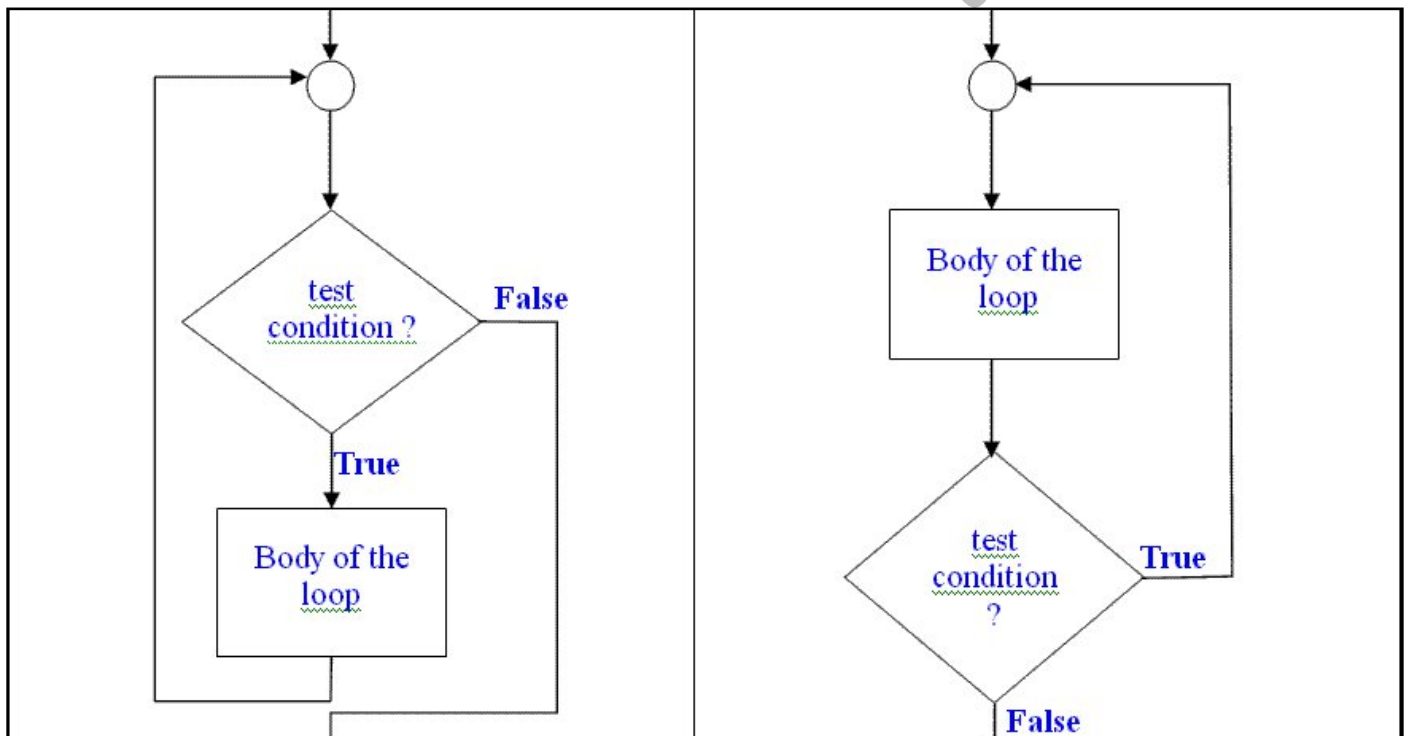
Ans : The versatility of the computer lies in its ability to perform a set of instructions repeatedly. Some specified portion of code is repeated n times , till a particular condition is satisfied. This repetitive operation is done through a loop control instruction.

A program loop is the combination of body of the loop and a control statement. Control statement tests certain conditions and then directs the repeated execution of the statements contained in the body of the loop.

Depending on the position of control statement inside the loop. Two types of control structures are defined

(a) **Entry controlled loop** (b) **Exit controlled loop.**

In Entry controlled loop pre testing is done. In exit controlled loop post testing is done.



Entry controlled loop

Exit controlled loop

Steps of the Looping Process:

- (v) Declaration and initialization of a condition variable.
- (vi) Execution of the statements inside the loop.
- (vii) Test for a specified value of condition variable for loop execution.
- (viii) Updating (increment/decrement) of condition variable.

Types of loop statements in C language are : (i) while loop (ii) do while loop (iii) for loop.

Classification of Loops : Based on the nature of control variable and the kind of value assigned to it for testing the control expression, the loops may be classified as :

- (i) Counter controlled loops
- (B) Sentinel controlled loops.

Counter controlled loops:

- (i) In this the number of iterations to be performed is known in advance.
- (ii) These loops used a counter variable called loop counter to keep track of iterations inside the loop.
- (iii) It starts with the initial value of the loop counter and terminates when the final value of loop counter is reached.
- (iv) Also known as definite repetition loops , because iteration is in fixed number of times.

Sentinel controlled loops:

- (i) In this number of iterations to be performed are not known in advance
- (ii) The execution and termination of loop depends on a sentinel value.
- (iii) If sentinel value is true loop body will be executed otherwise not.
- (iv) Also known as indefinite controlled loop.

Ques10. How is for loop executed? Give its syntax and example. What are additional features of for loop?

Ans : for loop is a entry controlled loop. It provides a most concise structure in program as compared to while loop and do while- loop. It is used when number of iterations are known before hand in the program.

Syntax of for loop : for (initialization ; test condition ; update variable)
 { body of the loop }

The execution is as given below:

- (f) Initialization of the control variables is done first.(loop control variable).
 - (g) The test condition can be any expression with relational operators($x \leq 10$, $m = n$ etc) or any logical expression ($x \parallel y$, $!a \ \&\& \ t$ etc.). This determines when the loop will terminate.
 - (h) If test condition is true body of loop executes, else loop is terminated.
-

- (i) After execution the control is transferred back to the for statement after evaluating the last statement in the loop.
- (j) Now control variable is updated again, and test condition is checked for true/false.
- (k) If test condition is true body of loop is executed again , otherwise exit from the loop occurs.

Additional Features of for loop

- (i) **Multiple Initialization:** In this more than one variable can be initialized at a time in the for statement. E.g : `p = 1;`

`for (n = 0 ; n < 17 ; n++)`, in this loop we can reconstruct as follows:

```
:      for ( p = 1 , n = 0 ; n < 17 ; n++) // variable p is also initialized inside for
      statement.
```

- (ii) **Multiple Increment:** In this the increment section may also have more than one part.

```
for( n=1 , m= 50 ; n <= m ; n ++ , m- -)
{
    p = m/n ;
    printf ( “ %d %d %d \n ” , n , m , p ) ;
}
```

- (iii) **Variation in test condition:** for loop may have any compound relational expression or logical expression for testing the loop condition, rather than depending on a single loop control variables.

```
sum = 0 ;
for( i=1 ; i < 20 && sum < 100 ; ++ i)
{
    sum = sum + i ;
    printf(“ %d ” , sum) ;    // Here i is counter variable
}                            // sum is sentinel variable.
```

Loop will execute as long as both the conditions are true

- Ques 10. (a) Explain break and continue statement with examples.**
(b) What are the properties of nested loops?

Ans : (a) Break statement:

- (vii) In C break statement is used to terminate the execution of the nearest enclosing loop in which it appears.
 - Used with for , while and do-while loops.
 - When compiler encounters the break statement , the control passes to the statement That follows the loop in which break statement appears

- Generally break is associated with if statement.

Syntax of break with while loop	Syntax of break with for loop
<pre> while (test condition) { if(condition) break ; } </pre> <p>Transfers control out of the loop while.</p>	<pre> for(.....) { if(condition) break ; } </pre> <p>Transfers control out of the for loop.</p>
<p>Example :</p> <pre> #include <stdio.h> void main() { int i=1; while(i<=10) { if(i== 5) break; printf(“ \n %d ” , i); ++i; } } </pre>	<p>Example :</p> <pre> #include <stdio.h> void main() { int i=1; for (; ;) { printf (“ \n %d ” , i); if(i= = 5) break; } } </pre>

(vi) **Continue statement :**

- This keyword is used inside the loop body
- When compiler encounters a continue statement, then rest of the statements in loop are skipped and control is unconditionally transferred to the start statement of the loop.
- When it is present inside the nested loop, it only terminates current iteration of the nearest enclosing loop.
- There is no limitation on the number of continue statements that can be present inside the loop

Syntax of continue with while loop	Syntax of continue with for loop
<pre>while (test condition) { if(condition) continue ; }an be udes to control Transfers control out of the loop while.</pre>	<pre>for(.....) { if(condition) continue ; } Transfers control out of the for loop.</pre>
<p>Example :</p> <pre># include<stdio.h> void main() { int i=1; while(i<=10) { if(i%2 == 0) continue ; printf(“ \n %d ” , i); ++i; } }</pre>	<p>Example:</p> <pre># include<stdio.h> void main() { int i; for (i=1; i<=10 ; i++) { if(i%2 == 0) continue; printf(“ \t %d ” , i); } }</pre> <p>Output : 1 3 5 7 9</p>

Nested Loops :. Another outer loop is used to control the number of times that a whole loop is repeated. Loops that can be placed inside other loop. This feature can work with any loop but it is mostly useful in for loop because of

<p>Example : void main()</p> <pre>{ int i,j; for(i=1; i<3; i++) for(i=1; j<4; j++) { if(j= = 2) continue ; printf (“ %d %d \n” , i,j) ; }}</pre>	<p>Output :</p> <pre>1 1 1 3 2 1 2 3</pre>
--	---

simplicity of for loops. A for loop can be used to control the number of times that a particular set of statements will be execute . Indentation of nested loops should be proper. Loops can be nested to any level .

Module 1: Programs on Iterations & Loops

1. Write a C program to find the factorial of a given number.

```

#include<conio.h>
#include <stdio.h>
void main()
    {
long int i,fact=1,num;
clrscr() ;
printf("Enter the number\n");
scanf("%ld",&num);
if( num <= 0)
fact = 1;
else
    {
for(i = 1; i <= num; i++)
    { fact = fact * i;}
}
/* End of else */
printf("Factorial of %ld =%ld\n", num, fact );
}
/* End of main() */

```

2. Program to generate the table of any number using loop

```

#include<stdio.h>
#include<conio.h>
void main()
    {
int t , num , t ;
clrscr() ;
printf ("Enter the number whose table is to be
generated : \n" ) ;
scanf(“%d”, & num ) ;
printf ( “ Table of %d is :” , num) ;
for ( i=1 ; i<= 10 ; i++ )
    {
t = num * i ;
printf( “\n %d x %d = %d ” , num , i , t ) ;
}
getch () ;
}

```

3. Program to check whether a given number is prime or not and output the given number with suitable message.

```
#include <stdio.h>
#include <conio.h>
void main()
    {

    int i, num;
    int flag = 1;
    clrscr();
    printf(" Enter the number");
    scanf(" %d", & num);
    for( i = 2; i< num; i++)
        {
            if((num % i) == 0)
                {
                    prime = 0;
                }
        }

    if (prime == 1)
        printf("%d is prime number.", num);
    else
        printf("%d is not a prime number.", num);
    getch();
    }
```

**4. Program to generate Fibonacci sequence
Fibonacci sequence is 0 1 1 2 3 5 8 13 21.**

```
#include <stdio.h>
#include <stdio.h>
void main()
    {
    int fib1=0, fib2=1, fib3, limit, count=0;
    printf("Enter the limit to generate the
    fibonacci sequence\n");
    scanf("%d", &limit);
    printf("Fibonacci sequence is ...\n");
    printf("%d",fib1);
    printf("%d",fib2);
    count = 2; /* fib1 and fib2 are already used */
    while( count < limit)
        {
        fib3 = fib1 + fib2; count ++ ;
        printf("%d\n",fib3);
        fib1 = fib2;
        fib2 = fib3;
        }
    getch();
    } /* End of main() */
```

5. Program to accept an integer and reverse it.

```
#include<conio.h>
#include <stdio.h>
void main()
    {
    long num, rev = 0, temp, digit;
    printf("Enter the number\n");
    scanf("%ld", &num);
    /*For better programming, choose
    'long int' */
```

**6. Program to calculate the sum of the series
up to first 100 terms: $1^4 + 3^4 + 5^4 + 7^4 + \dots + n$
terms.**

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main ()
{ int n , i , sum = 0 ;
  clrscr();
  printf("ENTER THE NUMBER OF
  TERMS\n");
  scanf("%d", &n) ;
```

<pre> /*reverse of a five digit no. can be out of range of integer*/ temp = num; while(temp > 0) { digit = temp % 10; rev = rev * 10 + digit; temp = temp/10 ; } printf("Given number = %ld\n", num); printf("Its reverse is = %ld\n", rev); getch(); } </pre>	<pre> for (i=1 ; i<=n ; i++) { sum = sum + pow (i, 4) ; } printf("Sum of the series = %d" , sum) ; getch() ; } </pre>
---	--

<p>7 .Program to generate the sum of the following series:- $sum = x + x^2 / 2! + x^4 / 4! + + x^n / n!$</p>	<p>8.Printing the pattern of program</p> <pre> * * * * * * * * * * * * * * * </pre>
<pre> #include<stdio.h> #include<conio.h> #include<math.h> void main () { int , x , n , , j ,fact =0, float sum ; clrscr(); printf("Enter the no. whose series is to be found: ") ; scanf("%d", &x) ; printf("Enter the no. of terms up to which series is to be generated") ; scanf("%d", &n) ; sum = x ; for (i=1 ; i<=n ; i++) { if (i%2 == 0) { fact =1; for (j=1 ; j<=i ; j++) </pre>	<pre> #include<stdio.h> #include<conio.h> void main () { int row, col; clrscr(); for (row=1;row<=5; row++) { for (col=1 ; col <=row; col ++) printf(" * "); printf("\n"); } getch(); } </pre>

```

        { fact = fact * j;
          }

        sum = sum + pow (x ,i) / fact ;
    }      // End of IF BLOCK
}        // End of outer for loop
printf ("Sum of the series = %f", sum) ;
getch() ;
}        // End of main ()

```

9. Program to print the following pattern :

```

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

```

```

#include<stdio.h>
#include<conio.h>

```

```

void main()
{ int i, j;
  clrscr () ;
  for ( i = 5 ; i >= 1 ; i -- )
  { for ( j = 1 ; j <= i ; j --)
  { printf ("%d" , j ) ; }
  printf ("\n" ) ;
}
getch();
}

```

10. Program to print the following pattern (FLOYD'S TRIANGLE) :

```

      1
     2 3
    4 5 6
   7 8 9 10

```

```

#include<stdio.h>
#include<conio.h>

```

```

void main()
{
int n , r , val =1, j ;
clrscr () ;
printf ( "Enter the number of rows in the
Floyd's triangle\t : " )
scanf ("%d" , &n ) ;
for ( r = 1 ; r <= n ; r ++ )
// print n rows
{
for ( j = 1 ; j <=r ; j ++ )
{ printf ("%d" , val ) ;
val ++ ;
}
printf ("\n" ) ;
} getch();
}

```

<p>11. WAP to detect Armstrong Numbers in three digits between 100 to 999.</p> <pre># include<stdio.h> #include<conio.h> void main() { int d=3 , n , x=0 ; int k , i , cube=0; clrscr(); printf("\n Following number are Armstrong numbers "); for (k =100 ; k<=999 ; k++) { n=k ; while (x<=d) { i=n %d ; cube = cube + pow (i, 3) ; n = n/10; x++; } if(cube == k) printf (" \n \t %d " , k) ; getch(); } // End of main() }</pre>	<p>12. Program to find binary equivalent of a decimal number entered by user.</p> <pre># include<stdio.h> #include<conio.h> void main() { int r , n ; clrscr(); printf ("Enter the value of n :"); scanf (" %d " , &n) ; printf (" Binary number"); while (n!=0) { r = n % 2 ; n = n / 2 ; printf (" %d " , r) ; getch(); } }</pre>
<p>13. Program to calculate m to power n.</p> <pre>#include<conio.h> #include<stdio.h> void main() { int m , n , i=1; long int pow =1 ; clrscr(); printf(" Enter two numbers : "); scanf (" %d %d " , &m , &n); while (i<=n) { pow = pow * m ; i++; } printf (" \n %d to power %d = %ld " , m , n, pow); }</pre>	<p>14. Program to add first seven terms of following series using for loop : $1 / 1! + 2 / 2! + 3 / 3! + \dots + 7 / 7!.$</p> <pre>#include<conio.h> #include<stdio.h> void main() { int i=1, j ; float fact , sum = 0.0; clrscr(); while(i<=7) { fact = 1.0 ; for (j=1 ; j <=i ; j++) fact = fact * j; sum = sum + i/ fact; } printf(" Sum of series = %f " , sum); }</pre>

<pre>getch(); }</pre>	<pre>getch(); }</pre>
-----------------------	-----------------------

<p>15. WAP to find the number and their sum between 100 and 200 divisible by 7.</p> <pre>#include<stdio.h> #include<conio.h> void main() { int num , sum = 0 ; printf(“ Numbers divisible bt 7 b/w 100 & 200 are : \n ”); for (num = 101 ; num < 200 ; num++) { if (num % 7 == 0) { printf (“ %d ” , num) ; sum = sum + num ; } } printf(“ Sum of all integers divisible by 7 b/w 100 & 200 is = %d ” , sum); getch() ; }</pre>	<p>16. WAP to find the average of first n natural numbers using while loop.</p> <pre>#include<stdio.h> #include<conio.h> void main() { int n , i=1, sum = 0 ; float avg = 0.0; printf(“ Enter the value of n : ”); scanf(“ %d ” , &n) ; while(i <=n) { sum = sum + i ; i++; } avg= (float) sum / n ; printf(“\n Sum of first %d numbers = %d “ , n , sum) ; printf(“ \n The average of first %d numbers = % f ” , n , avg) ; getch() ; }</pre>
--	---

<p>17. Program to print the sum of odd numbers and even numbers separately from 1 to 50.</p> <pre>void main() { int i; m , sumeven =0 ,sumodd=0; clrscr(); for (i = 1 ; i<=50 ;++i) { if (i % 2 == 0) sumeven = sumeven + i; else sumodd = sumodd + i; } printf(“ Sum of even numbers = %d \n” , sumeven) ; printf(“ Sum of odd numbers = %d” , sumodd); }</pre>	<p>18. WAP to find the sum of following series: $\sqrt{1} + \sqrt{1} + \sqrt{1} + \dots + \sqrt{1}$</p> <pre>void main() { int c , n ; float sum = 0.0 ; printf(“ Enter the value of n : \n”); scanf(“ %d ” , &n) ; for (c=1 ; c < n ; c++) { sum = sum + sqrt(c) ; } printf(“ Sum of square root series = %f” , sum) ; getch(); }</pre>
---	---

getch());

19. WAP in C to check whether an entered number through the keyboard is palindrome or not .

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int i , n , d , rev=0 , num;
    clrscr();
    printf( " Enter the number \n" );
    scanf( " %d " , &num);
    n= num ;
    while(n>0)
    {
        d=num %10;
        rev=rev *10 +d;
        n = num/10;
    }

    printf(" Reverse of %d = %d \n" , num , rev);

    if(rev == num)
        printf( " \n %d is a palindrome" , num);
    else
        printf(" %d is not a palindrome" , num);
    getch();
}
```

20. Program to compute the sum of following series : $x - x^3/(3!) + x^5/5! - x^7/7! + \dots +$ upto n terms.

```
void main()
{
    int p =1, n ;
    float sum=0.0, term;
    printf(" Enter the value of x \n");
    scanf( " %d" , &x) ;
    printf("Enter the power of nth term\n ");
    scanf(" %d " , &n);
    while( p<=n)
    {
        sum = sum + term;
```

```
term=(term *x *x -1) / (( i + 1) * (i +2));
        i = i +2 ;
    }

    printf( " Sum of Series = %f " , sum) ;

    getch();
}
```

--	--

21. WAP to check whether a given number is perfect or not ?

```

void main()
{
int num , sum =0 , i;
clrscr();
printf(" Enter the number \n");
scanf(" %d ", &num);
for( i=1 ; i<num ; i++)
{
if(num %i == 0)
sum = sum + i;
}

if( num == sum)
printf(" %d is a perfect number", num);
else
printf(" %d is not a perfect number",
num);
getch();
}

```

22. WAP to generate the following pattern:

```

      A
     A B
    A B C
   A B C D

```

```

#include<stdio.h>
#include<conio.h>
void main()
{
char i, j ;
for ( i=65 ; i<=68 ; i++)
{
for( j=65;j<=i; j++)
{
printf( " %c" , j);
}
printf("\n");
}
getch();
}

```


Short Question & Answers (Module 2 : Functions)

Ques 1. What is the purpose of main () function?

Ans: In our c program, every program starts with main () because it tells the C compiler where the program starts its execution. It always returns an integer value. It is a well-defined user defined function, because In different programs the body of main() contains different code written by the programmer.

Ques 2. What are the local variables with respect to a function?

Ans: The local variables are defined within the body of the function or the block. The variable defined is local to that function or block only. Other functions can access these variables. The compiler shows errors in case other functions try to access the variables.

Ques 3. What are library functions and user defined functions?

Ans : Library functions are those which are pre defined in the library of C compiler. The definitions are given inside the header files . E.g : printf(),scanf(), getch(), sqrt(), clrscr() etc.

User defined functions are those in which definition is written by the programmer inside the function body.E.g A function named add(int a , int b) to add two integer numbers a and b .

Ques 4. What is the meaning of keyword void before the name of a function?

Ans: Keyword void means that the function should not return any value.

Ques5. What is an argument? Differentiate between formal arguments and actual arguments?

Ans: An argument is an entity used to pass the data from calling function to the called function. Formal arguments are the arguments available in the function definition. They are preceded by their own data types. Actual arguments are available in the function call.

Ques 6. What is the purpose of return keyword while programming with functions?

Ans : When a function calculates some value , then after that control returns from a function, values returned are collected in calling function using the keyword 'return'. A function can return only one value at a time. Example : Following return statements are incorrect : return (a , b) , return (x , 12),

Because both are trying to return two values. ***In case if we want to use more than one values in return keyword syntax is : return value 1, value 2 , value 3.....value n. Here left to right evaluation is done And the right most value is returned to calling function.***

So correct statement are : return (a) , return (x).

But a function body can contain more than one return statements.

```
E.g : if ( x >= 5)
        return (x) ;
        else return ( 0) ;
```

Ques 7. Define exit() function with an example.

Ans : C provides a way to leave the program before it is actually terminated using exit() function.

(i) Syntax : exit(status) , where status is an int variable / constant.

(viii) This function is defined in **stdlib.h**

header file Example :

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

```
#include <conio.h>
```

```
# include <stdlib.h>
```

```
void main()
```

```
{
```

```
    printf( " C programming is logical and interesting " );
```

```
    exit (0);
```

```
    printf( " C can be easily learned" );
```

```
    printf( " C is a powerful language invented by Denis Ritchie in AT & T labs Bell");
```

```
}
```

Long Questions & Answers (Module 2 : Functions)

Ques 8. What are functions and their types ? What are the elements of user defined functions , explain?

Ans : Functions break large computing tasks into smaller ones, and make the task of coders easy.

A function is a self-contained block of statements that perform a coherent task of some kind.

In C language we use many small functions rather than few big functions. Functions support modular programming , in which a software is divided into small modules and each module consists of several functions/sub modules.

In a C program, execution starts from the function main(). A function can activate some another function. It can also call itself recursively , such functions are called recursive functions.

Types of functions: (a) Library functions (b) User defined functions.

Library Functions: Library functions are need not to be written by the programmers. Their definitions are already given by the compiler itself in the header files. Whenever we require to use any library function , the respective header file is included before the function main() at top of the program. *E.g : In `stdio.h` : `printf()`, `scanf()` functions , In `conio.h` : `getch()`, `clrscr()`;*

User Defined functions: Definition of these functions are written by the programmers itself .These are not already defined in the library of C compiler. Advantages of user defined functions are as follows :

- 6) It facilitates top down approach of modular programming.
- 7) Provides more structured and easy way to read and understand the large & complex codes.
- 8) It is easy to find faulty functions and their testing and debugging.
- 9) User defined functions may be used by many other programs (with the help of user defined Header Files). So repetition of same code again and again can be avoided, which makes less execution time and reduced code size.

Elements of user defined functions:

(i) Function declaration(eclaration)

(ii) Function call

(iii)Function definition.

Function declaration: Like variables in C , all the functions in a C program must be declared before they are invoked . A function declaration consists of four parts :

- Function return type. E.g : int , float , char, void etc.
- Function name (Follows the rules of an identifier.)
- Parameter or argument list. A function can contain one or more parameters with data type of arguments.
- Termination semicolon.
- Prototype definition and declaration of library functions are inside the header files.

Syntax : **function-type function name (parameter list) ;**

Parameter list must be separated by commas. Names need not to be same in the prototype declaration & function definition.

Use of parameter names is optional in prototype declaration, but their data type is compulsory.

If a function has no formal parameters , the list is written as (void)

Parameters : These are the communication medium to pass information between calling and called functions. *These are used in three places: prototype, function call , function definition.*

Formal Parameters : The parameters used in prototypes and function definitions.

Actual Parameters : The parameters used in function calls.

Function Call: A function can be called by simply using the function name followed by a list of actual parameters if any required.

- When the compiler encounters a function call, the control is transferred to the function definition.
- This function is then executed line by line and a value is returned when return statement is executed.

Function definition: It is the actual implementation of function, where actual logic of program is written. A function can only be defined once but can be declared many times.

A function can be declared within the body of some other functions but cannot be defined within the body of other function.

Elements of Function definition: (i) Function name (ii) Function name (iii) List of parameters.
(iv) Local variable declarations (v) function statements (vi) return statements.

```
Example : #include <stdio.h>
#include<conio.h>
int cal_sum(int , int , int);
void main()
{
    int a , b , c , sum=0 ;
    printf(" Enter the three values :");
    scanf( " %d %d %d", &a , &b , &c);
    sum=cal_sum(a,b,c) ;
    printf( "\n Sum = %d ",sum) ;
}
int cal_sum (int x , int y , int z) ;
{
    int d ;
    d= x + y + z ;
    return (d) ;
}
```

Ques 9: Explain Call by Value and Call by Reference methods of parameter passing. In support give the example programs also.

Ans : Call by Value : In this method values of variables are passed by the calling function to the called function. New variables are created by the called function to store values of arguments passed to it. So called function uses a copy of the actual arguments .Changes are reflected only in the called function , not the calling function.

Advantage: Arguments passed can be variables (e.g x) , literals (e.g : 6) , or expressions (e.g x + 1).

Disadvantage: Copying data consumes additional storage. It can take lot of time to copy , so performance of function may decrease if it is called several times .

Call by Reference : When a function wants to modify the value of the argument , then we pass arguments using address of variables. Function parameters are declared as a reference/address. When this is done then any changes made by the function to the arguments it receives, are visible in the calling function.

For reference an asterisk(*) is placed after the data type in parameter list.

Advantage : Since arguments passed are not copied into new variables , it provides greater time and space efficiency. We can return multiple values using this method.

Disadvantage : Side effect of this method is whrn an argument is passed using call by address, it becomes difficult to tell whether that argument is meant for input , output or both.

<p align="center">Example of Call by Value :</p> <p align="center">/* Swapping the values of two variables using call by value technique*/.</p>	<p align="center">Example of Call by Reference :</p> <p align="center">/* Swapping the values of two variables using call by reference method*/.</p>
<pre>#include<stdio.h> #include<conio.h> void swap (int x , int y) ; void main() { int a , b ; clrscr(); printf (“ Enter the values of two numbers \n”) ; scanf(“% d %d “ , &a, &b); printf(“Values of a and b before swapping : \n”); printf (“ a= %d\t b=%d” , a ,b); swap (a , b) ; printf(“Values of a and b after swapping : \n”); printf (“ a= %d\t b=%d” , a ,b); getch(); } void swap(int x , int y) { int t ; t = x; x = y; y = t ; printf(“ In swap function values are : \n”); printf(\n x = %d \t y = %d, x ,y); } Output : Enter the values of two numbers: a=10 , b=20 Values of a and b before swapping :a=10, b=20 In swap function values are : x= 20, y=10 Values of a and b after swapping : a =10, b=20</pre>	<pre>#include<stdio.h> #include<conio.h> void swap (int *x , int *y) ; void main() { int a , b ; clrscr(); printf (“ Enter the values of two numbers \n”) ; scanf(“% d %d “ , &a, &b); printf(“Values of a and b before swapping : \n”); printf (“ a= %d\t b=%d” , a ,b); swap (a , b) ; printf(“Values of a and b after swapping : \n”); printf (“ a= %d\t b=%d” , a ,b); getch(); } void swap(int *x , int *y) { int t ; t = *x; *x = *y; *y = t ; printf(“ In swap function values are : \n”); printf(\n x = %d \t y = %d, x ,y); } Output : Enter the values of two numbers: a=10 , b=20 Values of a and b before swapping :a=10, b=20 In swap function values are : x= 20, y=10 Values of a and b after swapping : a =20, b=10</pre>

Ques 10. (i) Explain recursion and recursive function.

(ii)WAP using recursive function to compute the sum of following series upto n terms :

$$1 + 2 + 3 + \dots + (n-1) + n .$$

Ans : A recursive function is defined as a function that calls itself again and again to solve a smaller version of its task until a final call is made. Every recursive solution has two major cases :

- **Base Case :** In this the problem is simple enough to be solved directly without making any further calls to the same function.
- **Recursive Case :** In this first the given problem is divided into simpler sub parts. Then secondly , function calls itself but to handle sub parts of first step.

Basic steps of recursion are analyzed as follows :

- Initially Specify the base case which will terminate the function call itself.
- Check whether the current value processed matches with the value of the base case. If yes, then process and return the value.
- Divide the problem into smaller sub parts and call the function from sub part.
- Combine the results of sub parts.
- Return the result of the entire problem.

Recursive function for adding 1 + 2 + 3 ...n terms.

<pre>#include<stdio.h> #include<conio.h> int sum (int) ; // function prototype. void main() { int n , s ; clrscr(); printf ("Enter the value of n terms :"); scanf ("%d " , &n) ; s = sum(n) ; // function call in main() printf (" Sum of %d natural numbers = %d" , n,s) ;</pre>	<pre>getch(); } int sum (int x) // function definition { int a = 0 ; if (x == 0) return 0 ; else a = x + sum(x-1) ; // recursive case. return(a); }</pre>
--	--

Ques 11. Differentiate between the following :

- **Call by value and call by reference method of parameter passing.**
- **Iteration and recursion.**

Ans :

Call by Value method	Call by Reference method
1. This is a usual method to call a function in which only the value of variable is passed as an argument.	In this method the address of variable is passed as an argument.
2. A copy of actual arguments is passed in formal arguments in function definition.	Formal parameters are pointers to actual arguments.
3. Any changes in the value of formal arguments does not affect in value of actual arguments.	Changes made in formal arguments are reflected back to the values of actual arguments.
4. By this method a function can return only a single value.	By this method a function can return more than one values at a time.
5. Memory locations occupied by formal and actual arguments are different.	Memory locations occupied by formal and actual arguments are same , so less memory is occupied as compared to Call by Value.
6. This is a slow process.	This is more efficient and fast method.
7. Function prototype contains data type as argument.	Function prototype is a pointer prefixed to arguments inside it.

Iteration	Recursion
1. This uses a set of statements which are repeated.	This involves selection of a particular structure in a function.
2. Repetition is applied to the loop constructs.	Function calls itself again and again.
3. Iteration terminates when loop test condition becomes false.	Recursion stops when base case is encountered in function call , within its definition.
4. Iteration requires a counter variable or a sentinel value.	In this base case and recursive case is required.
5. Less memory consumed	More memory is consumed , because each time a function calls itself , a copy of variables is created.

Programmes (Module 2 : Functions)

1. WAP in C for finding the sum of three no. and their average using user defined functions sum() and avg().

```
#include<stdio.h>
#include<conio.h>
int sum (int , int , int ) ;
float avg (int , int , int ) ;
// Prototype or declaration of function
void main()
{
int a , b , c, sum=0 ;
float av =0 ;
s= sum (a , b , c); // function call
av = avg (a , b , c) ;
scanf (“%d%d%d”, &a, &b, &c) ;
// actual parameters
printf (“Sum= %d\t Average = %f” , s , av) ;
getch() ;
} // End of main ()

// Definition of user defined function sum ()
int sum (int x , int y , int z )
// x , y and z are formal parameters
{
return (x + y + z) ;
}

/ Definition of user defined function avg ()
float avg (int x , int y , int z )
{
float m ;
m = (float) (x + y + z) / 3;
return m ;
}
```

2. Program to compute factorial of a number using recursive function.

```
#include<stdio.h>
#include<conio.h>
long int rec_fact(long int) ;
void main ()
{
int num , fact ;
clrscr() ;
printf(“Enter the number”);
scanf (“%ld”, &num) ;
fact = rec_fact (num) ;
printf(“Factorial of %ld = %ld”, num , fact) ;
getch() ;
}
// recursive definition of factorial

long int rec_fact(long int x) ;
{ int f;
if(x == 1 )
return 1; // base case
else
{
f = x * rec_fact(x-1) ; // Recursive case
return (f) ;
}
}
```

<p>3. Program to generate the Fibonacci series using recursive function</p>	<p>4. WAP to find largest number among the given two numbers using a user defined function large().</p>
<pre> #include<stdio.h> #include<conio.h> int fibo(int) ; void main () { int n , f ; clrscr() ; printf("Enter the number upto which series is to be generated \n") ; scanf ("%d", &n) ; printf("Fibonacci series up to %d terms is : \n \n", n) ; for (int i=1 ; i<= n ; i++) ; { f= fibo (i) ; } int fibo (int n) // Recursive definition { if (n == 1) return 0 ; // Base case if (n == 2) return 1 ; return (fibo(n - 1) + fibo(n - 2)) ; // At a time two recursive function called so binary } </pre>	<pre> #include<stdio.h> #include<conio.h> float large (float m , float n) ; void main() { float x , y , max ; float large (float m , float n) ; printf(" Enter the two numbers \n"); scanf(" %f%f" , &x , &y) ; printf(" x = %f and y = %f \n" , x , y) ; max = large(x , y) ; printf(" The largest number = %f \n" , max); } float large (float m , float n) { if (m > n) return m ; else return n ; } </pre>

<p>5. WAP that accepts the length and width of a rectangle and print the area. The area of a rectangle is calculated by a user defined function and returns it value to main program where it is printed. The values length and width are accepted from the keyboard as an integer value. Also draw the flow chart of program.</p>	<p>6. WAP that accepts the three sides of a triangle and print the area. The area of a triangle is calculated by a user defined function and returns it value to main program where it is printed. The values of three sides are accepted from the keyboard as an integer value</p>
<pre># include<stdio.h> #include<conio.h> int rect_area(int , int); void main() { int length , width ,area=0 ; clrscr() ; printf(“ Enter the length and width \n ”); scanf(“ %d %d ”, length , width); printf(“ Length =%d width= %d \n”, &length, & width) ; area= rect_area (length , width) ; printf(“ Area of rectangle = %d ”, area); getch(); } int rect_area(int l , int w) { return (l *w) }</pre>	<pre># include<stdio.h> #include<conio.h #include <math.h> float tri_area(float , float, float); void main() { float s1 ,s2 ,s3, area=0.0 ; clrscr() ; printf(“ Enter the three sides of triangle \n ”); scanf(“ %f%f%f”, s1 , s2, s3) ; printf(“ side1 =%f side2= %f side3=%f\n”, &s1, & s2, &s3) ; area= tri_area (s1 , s2 ,s3) ; printf(“ Area of triangle = %f”, area); getch(); } float tri_area (float a , float b , float c) { perim=0.0 , a=0.0 ; perim = (a + b + c) /2 ; a= sqrt(perim * (perim-a) *(perim-b) *(perim-c)) ; return (a); }</pre>

[END of 3rd UNIT]

UNIT 4

Module – 4: (Arrays & Basic Algorithms)

- **Arrays: Array notation and representation**
- Manipulating array elements
- **Using Multi dimensional arrays.**
- Character arrays and strings
- **Structure, union, enumerated data types**
- Array of structures
- Passing arrays to functions.

Basic Algorithms:

- **Searching & Basic Sorting Algorithms (Bubble, Insertion and Selection)**
- Finding roots of equations.

Short Questions & Answers

Ques 1. How one dimensional array are initialized? Explain.

Ans : An array elements can be initialized like an ordinary variables when declared. The general syntax is as following : `data_type array_name [size] = { list of values separated by comma};`

Where [] is known as subscript operator.

Example: `int arr[5] = { 10, 34, 21,76, 20};`

`float f[4] = { 0.0 , 1,24, 5.63 ,3.6};`

`int a [] = { 10 , 30 ,15, 8 , 40 , 60, 50};`

`char ch[6] = { 'A' , '8' , 'B' , ' &' , 'd' , '\0 '};`

- (i) Array elements contain garbage values if no values are given.
- (ii) If number of elements are not known at compile time, then C does not allows to declare that array.
- (iii) When all the elements are listed when declaring an array, size is optional.
- (iv) Total elements are from 0 to size-1. Size should be a compile time constant expression of integer type.

Ques 2. What are the different ways to store the values in arrays?

Ans : (a) Initialization of array during declaration (b) Input values for the elements from the keyboard.
(c) Assign values to individual elements.

Examples: (a) **Initialization of array during declaration**

`int marks [5] { 90, 65, 78, 80 , 47}; // initialization at the time of declaration.`

90	65	78	80	47
[0]	[1]	[2]	[3]	[4]

When values are initialized in a 1-D array they are stored consecutively in memory, starting from index 0 to size-1 i.e. in above example from 0 to 4 , because size of array is 5. **If we declare like this :**

`int marks [] = { 2 , 4, 10};` We have not mentioned the size inside subscript operator, So compiler will allocate space automatically equal to number of elements inside { } braces.

(b) **Input values for the elements from the keyboard.**

`int i , marks [15];`

`for (i=0 , i< 10 ; i++)`

`scanf (“ %d” , & marks [i]);`

(c) **Assigning Values to individual Elements :** This can be done using assignment operator. Any value that matches to the data type of an array can be assigned to the individual array element.

Example : `marks [3] = 45 ; // 45 is assigned to 4th index of 1-D array.`

Ques 3. Mention some invalid declarations in an array.

Ans : (a) ~~int arr [j] ; // j is a variable not a constant.~~
 (b) int arr [4.7] ; // Size can not be a float value.
 (c) int arr [-1] ;
 int arr [0] // Size must be >= 1

Ques 4. Define the pointer. How it is declared and initialized ?

Ans : Pointer : A **pointer** is a variable whose value is the address of another variable, i.e., direct address of the memory location. Like any variable or constant, you must declare a pointer before using it to store any variable address. The general form of a pointer variable declaration is –

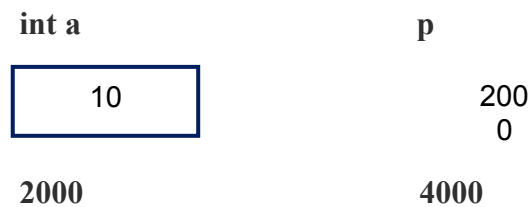
Data type *var-name;

Here, *data type is the pointer's base type*; it must be a valid C data type and **var-name** is the name of the pointer variable. The asterisk * used to declare a pointer is the same asterisk used for multiplication. However, in this statement the asterisk is being used to designate a variable as a pointer. Valid pointer declarations –

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

Initialization: `data type * ptr_var = & var name 2 ;// & is known as address of operator`
 Example : `// * is called value at address or indirection operator`

```
int a = 10;
int *p = &a ; // pointer variable p holds the value of variable a.
```



- (i) 2000 is the address of variable a , and pointer p holds this address as a value. Pointer variable p has its own address 4000. 10 is the value stored at location a.

Ques 5. WAP in C to print the values and address with the help of pointers.

Ans :

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

```

Void main()
{
    int x ;
    int *p;
    x = 10 ; p = &x ;
    printf( "Value of x = %d \n" , x );
    printf( " Variable x has address = %u \n" , &x ) ;
}

```

Ques 6. How are character arrays declared and initialized ?

Ans : Character arrays are also known as 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/C++ –

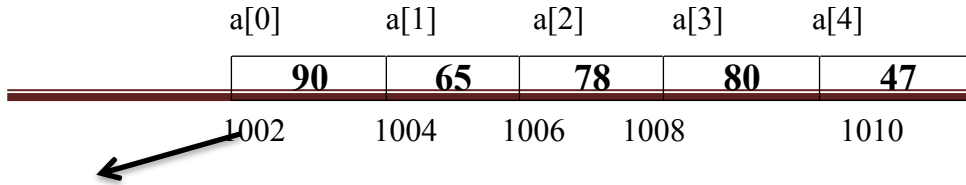
Index	0	1	2	3	4	5
Variable	H	e	l	l	o	\0
Address	0x23451	0x23452	0x23453	0x23454	0x23455	0x23456

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.

Ques 7. What is the base address of an array?

Ans : Base address of an array refers to the 1st element of a 1-D array at 0th Index position. If we only write the name of an array then also it refers to the base address.

E.g : int a [5] = { 1 , 4 , 7 , 9 , 10 } ;



Base address

Here 1002 is the base address of array a. Since it is an integer array, so addresses of each elements will be at the difference of 2 bytes. While displaying the elements of array, next adjacent address is automatically called when its previous element is displayed. Using pointers we can access the individual elements.

Increment / decrement of pointers lead to increment /decrement of addresses based on the data type of pointers.

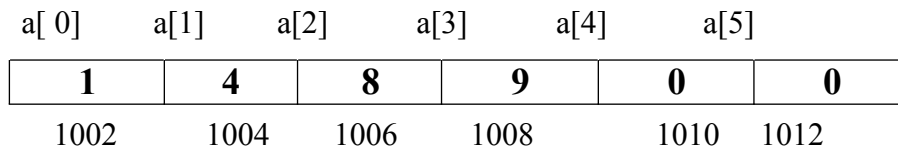
Ques 8. What is the output of following code :

```

void main()
{
    int a[6] = { 1, 4, 8, 9 };
    int j;
    printf( " Elements of array are : \n " );
    for (j=0 ; j < 6 ; j ++ )
    {
        printf ( " %d ", a [j] );
    }
    getch();
}

```

Ans : Output of above code is : 1 4 8 9 0 0.



If number of elements stored in an array are less than the maximum size of array , then remaining elements will take 0 value for remaining index. Since array a is initialized with only four elements, and max size is 6 Therefore elements at a[4] & a[5] will take values 0 , 0 by default.

Ques 9. Explain the following string manipulation functions :

- (i) gets()
- (ii) puts()
- (iii) sprintf()
- (iv) strcmp()

Ans : (i) gets() : This is a library function . Definition is stored in string.h header file. This function reads the string as **gets(str)** , where str is the name of character array str.It takes the starting address of an string which

will hold the input. This function does not terminate if a blank space occurs between two strings as in `scanf()`.

(ii) **puts()** : The string can be displayed by writing puts(str). This function overcomes the ~~drawback of printf. The puts() function writes a line of output on the screen. It terminates the line by '\n'.~~

(iii) **sprintf()** : This is formatted output is written to memory area rather than directly on the output screen. Syntax is as following :

*sprintf(char * buffer , const char* format string [arg 1,arg2arg n]) ;*

E.g : void main()

```
{
char buf [100] ;
int num = 10 ;
printf( buf , “ num = %d ” , num ) ;
}
```

(iv) **strcmp()** : This function compares the string pointed to by str1 with string pointed by str2.

*Syntax strcmp(const char *str1 , const char str2) ;*

Function returns zero if the strings are equal. Otherwise it returns a value less than zero or greater than zero if str1 is less than or greater than str2 respectively.

Ques 9. How are 2-Dimensional arrays declared and initialized.

Ans : The 2-D arrays are also known as matrix which are created in row and column form.

Syntax to declare: data_type array_name [i] [j] ; Where two subscripts i and j are rows and columns of a 2-D array. Subscripts i and j must be set at compile time.

E.g : int a [2] [3] ; // 2-D array with 2 rows and 3 columns.

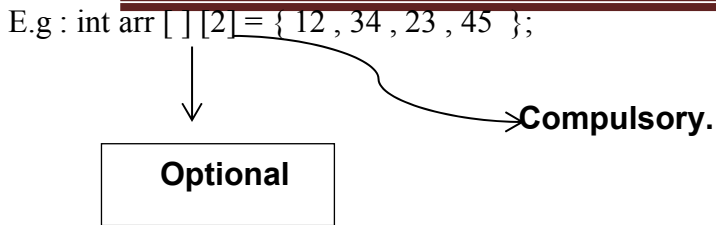
Initialization of 2-D array : int a [2] [3] = { 0, 2, 3, 10, 30, 20 } ;

Matrix can be arranged in Row major and column major forms. In Row major form firstly elements of 1st row , then 2nd row ...so on are stored . In column major form 1st column , then 2nd column...son on are stored.

Row major form : 0 2 3
 10 20 30 OR int a[2] [3] = { { 0,2,3} , { 10, 20, 30} } ;

Column Major form : 0 3 30
 2 10 20

While initializing a 2-D array , it is necessary to mention 2nd dimension(number of columns), whereas 1stdimension is optional.



Long Questions & Answers

Ques 10. *What is the relationship between arrays and pointers? Explain with an example. Give the advantages and limitations of array.*

Ans : The concept of array is very much dependent on pointers. Elements of array occupy contiguous memory locations depending on the data type of elements stored. Array location is a form of a pointer notation. The name of the array is the starting/base address of an array.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
1	4	8	9	0	0
1002	1004	1006	1008	1010	1012

`int *ptr ;`

`ptr = & a [0] ; // ptr is a pointer variable of integer type , that holds the address of 1st element a[0]`

if array stores elements with float data type , then addresses will change an pointer to 1st element will be of float type . E.g : `float * fptr ;`

`fptr = & b [0] ;`

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]
1.82	4.67	8.00	10.23	3.67	92.78
1002	1006	1010	1014	1016	1020

Now each element is stored with the difference of four bytes because of floating data type. Similarly in character array difference b/w each element will be of 1 byte.

Use of pointers : We can access the values of array elements using pointers as mentioned below:

- (i) Expression $a[i]$ is equivalent to write $*(a + i)$. If a is the name of array, then compiler implicitly takes $a = \&a[0]$. To print the value of third element of the array, we can use the expression $*(a + 2)$. Because $a[i] = *(a + i)$.

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]
11	10	80	49	12	34
1002	1004	1006	1008	1010	1012

```
Example : #include <stdio.h>
#include <conio.h>
void main ()
{
    int a [6] = {11, 10, 80, 49, 12, 34};
    printf( " First element is at address %u ", a );
    printf( " Third element is at address %u ", a + 2 );
    printf( " Last element is at address %u ", a + 5 );
    getch();
}
```

Advantages of Arrays

- (a) The direct indexing support is biggest advantage. Time required to read any element in an array of any dimension is almost the same irrespective of memory location.

Limitations of arrays

- (a) The memory in array is allocated at compile time.
 (b) Arrays are static in nature. The size of an array can not be changed (expanded/squeezed) at run time.
 (c) Size of an array has to be kept big enough to occupy worst case. Memory usage is inefficient.

Ques 12. (i) What is referencing and dereferencing in pointers. Give an example. (ii) What are the rules for pointers and Pointer arithmetic?

Ans : Referencing in pointers : In this a pointer variable is used to refer the address of an object. Use of reference operator ($\&$, *ampersand*), also called *address of operator*. This is a unary operator on left side of operands. The operand should be a variable of arithmetic type or pointer type.

E.g : `float f = 12.5 ;`

`float * fptr ;`

`fptr = &f ;`

Dereferencing a pointer : When we want to access the value stored at a particular address in a variable , then we use dereferencing with the help of pointers. This operator is also known as *value at address operator* , (*** , *asterisk*) . Operand of *** operator must be a pointer type variable.. This is a unary operator.

E.g : If we write **(p)* or **(&a)* = value at address of variable a.

// Referencing & Dereferencing of pointers

```
void main()
{
    int a = 12;
    int *i = &a;
    printf ( " Value of a is = %d \n " , a ) ;
    printf ( " Value by dereferencing a = %d \n " , *i ) ;
    printf ( " Value of a = %d \n" , *(&a))
    printf ( " Address of a = %u " , & a ) ;.
}
```

Rules of pointers

(a) A pointer can be assigned /initialized with the address of a variable. A pointer can't hold a non address value , so it can only be initialized with addresses.

(b) A pointer to a specific data type variable /object can't point to an object of another data type.
E.g : Following is an invalid assignment :

```
void main()
{
    int val = 10 ;
    float * ptr = & val ;
    printf ( " value of variable = %d \n" , val ) ;
    printf ( " Pointer ptr holds the address = %u \n " , ptr ) ;
}
```

(c) Increment and decrement operator can be applied to pointer variables which will change the addresses Accordingly.

E.g : float * fptr ;
fptr = ptr ++ // post increment and initially , ptr = 2000 (address).
So address will be incremented by 4 bytes because of float data type .So fptr = 2004.But ptr will remain 2000 because it is post increment. Similarly , If we perform ptr - - , then new address will be 1996.

(d) Subtraction: Subtraction between two pointer variables gives the difference in number of bytes , (subscripts of two array elements)depending on data type of variables. E.g : Consider following float array b

b[0]	b[1]	b[2]	b[3]	b[4]	b[5]
1.82	4.67	8.00	10.23	3.67	92.78

1002 1006 1010 1014 1016 1020

Let pointer $p1 = 1002$ and pointer $p2 = 1010$. So $p1 - p2 = 2$ i.e. difference of two subscripts where each subscript element is of 4 bytes (Array is float type). So, $4 \times 2 = 8$ bytes.

Subtraction of two pointers is useful only if both the pointers hold the address of variables in same array.

Illegal pointer operations

1. Two pointer variables can't be added.
2. Only integer constants can be added to a pointer variable, but float /double constants can't be added.
E.g: `int * a ;`
 `int * p ;`
 `p = a + 3 ;` // this is valid , but `p = a + 3.67` is invalid.
3. Multiplication and division can't be performed on pointer variables.
4. Bit wise operations are invalid.

Programs on 1-D Arrays

Ques 13. WAP in C to read and display n numbers Using an array.	Ques 14. WAP in C Find the smallest number of a 1-D array .
<pre>#include<stdio.h> #include<conio.h> void main() { int i=0 , n , arr [20] ; clrscr(); printf (“ Enter the number of elements : \n”) ;</pre>	<pre>#include <stdio.h> #include <conio.h> void main() { int arr [100] , min , i , n , s , pos ; float avg_marks = 0.0 ; clrscr() ;</pre>

<pre> scanf(“ %d ” , &n) ; printf(“ \n Enter the elements \n”); for (i=0 ; i<n ; i++) { printf(“ \n arr [%d] = ” , i); scanf(“ %d” , &arr[i]); } printf(“ \n The array elements are \n ”); For (i=0 ; I <n ; I ++) printf(“ arr[%d] = %d\t ”, i , arr[i]); getch(); </pre>	<pre> printf(“ Number of elements in array\n”); scanf(“ %d ” , &n) ; printf(“ Enter elements of array\n”); for(i = 0; i < n ; i++) { scanf(“ %d” , &a[i]); } Small = a[0] ; Pos =0 ; for (i=1 ; i<n ; i++) { If (small > a[i]) { Small = a[I] ; Pos =I ; } } printf(“ Smallest number of array is %d at position %d \n “ , small , pos + i); getch () ; } </pre>
---	--

<p>Ques 15.WAP in C to enter number of digits. Form a number using these digits.</p>	<p>Ques 16. WAP in C to calculate the sum and average of marks , secured by students using array named marks[] .Number of students must be read through keyboard.</p>
<pre> #include<stdio.h> #include<conio.h> #include<math.h> void main { int num =0, digit [10], n , i ; clrscr() ; </pre>	<pre> #include <stdio.h> #include <conio.h> void main() { int marks[200] , i , sum = 0, studs ; float avg_marks = 0.0 ; clrscr() ; </pre>

<pre> printf("Enter the number of digits for number \n"); scanf ("%d" , &n); for(i=0 ; i < n; i ++) { printf ("\n Enter the %d digit : " , i); scanf("%d" , &digit [i]); } i=0; while (i < n) { num = num + digit [i] + pow (10,i) ; i++; } printf("\n The number = %d " , num) ; getch(); } </pre>	<pre> printf (" Enter the number of students \n"); scanf ("%d" , & studs) ; printf (" Enter the marks of all students \n"); for(i = 0; i < studs ; i++) { printf ("Enter the marks of students %d \n\t",marks[i]); scanf ("%d" , & marks [i]) ; i) } for (i=0 ; i < studs ; i++) { sum = sum + marks [i] ; } avg_marks = (float) sum / studs ; printf (" Sum of marks = %d" , sum); printf("\n Average marks of %d students = %f" , studs , avg_marks) ; getch () ; } </pre>
--	---

<p>Ques 17. WAP in C to print the elements of an array using pointers.</p>	<p>Ques 18.WAP in C to check whether an array of integers contain a duplicate number or not.</p>
<pre> #include<stdio.h> #include<conio.h> { int b [3] = { 10 , 20 , 30 } ; printf("Elements are %d %d %d \n", b[0], b[1] , b[2]) ; printf(“ Elements are %d %d %d \n” , *(b +0), *(b+1) , *(b + 2)) ; getch(); } </pre>	<pre> #include<stdio.h> #include<conio.h> void main() { int i , n , j , arr [20] , flag=0 ; clrscr(); printf (" Enter the number of elements : \n") ; scanf("%d" , &n) ; printf("\n Enter the elements \n") ; for (i=0 ; i<n ; i++) { printf(" \n arr [%d] = " , i); scanf("%d" , &arr[i]) ; } for (i=0 ; i < n ; i ++) { </pre>

	<pre> for(j= i +1 ; j < n ; j++) { if (arr [i] == arr [j] && i != j) { flag =1; printf (“ Duplicate numbers found at location %d and %d “, i , j) ; } } if (flag=0) Printf(“ \n No duplicate numbers found ”) getch(); } </pre>
--	---

Ques 19. *What operations can be performed on 1-D arrays? Explain linear searching. WAP to demonstrate linear searching in an array.*

Ans : **Operations on arrays :**

- (a) Searching an element in an array.
- (b) Sorting an array.
- (c) Finding smallest and largest number in the given array.
- (d) Reading and printing the elements of an array.
- (e) Finding Kth Largest element in an array.
- (f) Comparison of two arrays.
- (g) Deletion of an element at specific position in an array.

SEARCHING: The technique for finding the desired data element that has been stored in an array is known as searching .After searching is completed , there may be two cases w.r.t data element : Case 1: Search is successful , if it locates an element at required position
Case 2 : Search is unsuccessful , if it fails to locate an element .

Types of Searching: (A) *Linear /Sequential search*

(B) *Binary Search.*

LINEAR SEARCH : In this method , array is searched for a particular data item by comparing every element of the array one by one until a match is found .An element to be searched is generally called KEY element is used to compare with another values. ~~Linear search is generally applied to unordered list (unsorted) of elements.~~

Example :

```
#include<stdio.h>
#include<conio.h>
void main()
{ int arr[ 10 ] = { 15, 20 , 30 , 9 , 14 , 56 , 40 , 100 , 67, 7 };
  int i, num ;
  printf( “ Enter the key element to be searched in array\n”);
  scanf ( “ %d” , & num);
  for(I = 0 ; i <10 ; i ++ )
    { if (num == arr [ i ])
      { printf ( “ %d number is found \n ”, num) ;
        break ;
      }
    else
      {
        printf( “ %d number not found ”, num ) ;
      }
    }
  getch () ; } // End of main()
```

Ques 20 : What is sorting ? WAP in C to sort the elements of an array using bubble sort.

Ans : Term sorting means arranging the elements of an array in some relevant order which may be either ascending / descending. If A is an array then , the elements of an array arranged in ascending order will be such that : $A[0] < A[1] < A[2] < \dots A[N-1]$. Example : if we have elements of array as :

$A[] = \{ 21 , 34 , 11, 9 , 1 , 0 , 22 \}$; Then the sorted array in ascending order will be as given below :
 $A[] = \{ 0 , 1 , 9 , 11 , 21 , 22 , 34 \}$. Efficient sorting algorithms are used to optimize the use of other Algorithms like search and merge algorithms which require sorted lists to work properly.

Two types of Sorting exist : (i) Internal Sorting (ii) External Sorting.

Internal sorting deals with sorting the data stored in computer’s memory. **External sorting** deals with the data stored in files . It is used when large data can not be stored in memory of computer.

Bubble Sort: In this method of sorting array elements is done by repeatedly moving the largest elements to the highest index position of the array(if using ascending order). In this consecutive adjacent pairs of elements are compared with each other. If element at lower index is greater than the element at higher index, the two elements interchange their positions. This process continues till the list if unsorted elements are finished.

Program of Bubble Sort Algorithm in C.

<pre>#include <stdio.h> #include <conio.h> void main () { int i,j, a, n, number[30]; clrscr(); printf ("Enter the value of N\n"); scanf ("%d", &n); printf ("Enter the numbers \n"); for (i=0; i<n; ++i) scanf ("%d",& number[i]); for (i=0; i<n ; ++i) { for (j=i+1; j<n; ++j) if (number[i] > number[j]) { a= number[i]; number[i] = number[j]; number[j] = a; } } }</pre>	<pre>printf ("The numbers arranged in ascending order are given below\n"); for (i=0; i<n; ++i) printf ("%d",number[i]); getch(); } /* End of main() */ Output Enter the value of N 5 Enter the numbers 80 20 67 10 45 The numbers arranged in ascending order are given below 10 20 45 67 80</pre>
---	---

<p>Ques 21. Write a C program to accept a matrix of order M x N and find the sum of each row and each column</p>	<p>Ques 22. Write a C program to accept a matrix of order m x n and find its transpose .</p>
<pre>#include <stdio.h> #include<conio.h> void main () { int m1[10][10]; int i, j, m, n, sum=0; clrscr() ; printf ("Enter the order of the matrix\n"); scanf ("%d %d", &m, &n); printf ("Enter the co-efficients of the matrix\n"); for (i=0; i<m ; ++i) { for (j=0 ; j<n ; ++j) { scanf ("%d", &m1[i][j]) ; } } for (i=0;i<m; ++i) { for (j=0;j<n; ++j) { sum = sum + m1[i][j] ; } } printf ("Sum of the %d row is =</pre>	<pre>#include <stdio.h> #include <conio.h> void main () { int mat [10][10]; int i, j, m, n ; printf ("Enter the order of the matrix \n"); scanf ("%d %d", &m ,&n); printf ("Enter the elements of the matrix\n"); for (i=0; i<m; ++i) { for (j=0;j<n; ++j) { scanf ("%d",& mat[i][j]); } } printf ("The given matrix is \n"); for (i=0;i<m; ++i) // Here outer loop is now for counter variable i {</pre>

<pre> %d\n", i, sum) ; sum = 0; } for (j=0; j<n ;++j) { for (i=0 ; i< m ; ++i) { sum = sum+m1[i][j] ; } printf ("Sum of the %d column is = %d\n", j, sum); } getch () ; } /*End of main() */ /* </pre> <hr/> <p style="text-align: center;">Output</p> <p>Enter the order of the matrix 3 3</p> <p>Enter the co-efficients of the matrix 1 2 3 4 5 6 7 8 9</p> <p>Sum of the 0 row is = 6 Sum of the 1 row is = 15 Sum of the 2 row is = 24 Sum of the 0 column is = 12 Sum of the 1 column is = 15 Sum of the 2 column is = 18</p>	<pre> for (j=0;j<n;++j) { printf (" %d", mat[i][j]); } printf ("\n"); } printf ("Transpose of matrix is \n"); for (j=0; j<n; ++j) // Here outer loop is now for counter variable j { for (i=0 ;i<m ;++i) { printf (" %d",mat[i][j]); } printf ("\n"); } /* End of main() */ </pre> <hr/> <p style="text-align: center;">Output</p> <p>Enter the order of the matrix 2 2</p> <p>Enter the coefficients of the matrix 3 -1 6 0</p> <p>The given matrix is 3 -1 6 0</p> <p>Transpose of matrix is 3 6 -1 0</p>
--	--

Ques 23. Program to Multiply Two 2-D arrays

<pre> #include<stdio.h> #include<conio.h> void main() { int a[10][10],b[10][10],c[10][10],i,j,k,r1,c1,r2,c2; int sum=0; clrscr(); printf("Enter number of rows and columns of first matrix \n"); scanf("%d%d",&r1,&c1); printf("Enter number of rows and columns of sec ond matrix \n"); scanf("%d%d",&r2,&c2); </pre>	<pre> printf("The First Matrix Is: \n"); //print the first matrix for(i=0; i<r1; i++) { for(j=0; j<c1; j++) printf(" %d ",a[i][j]); printf("\n"); } printf("The Second Matrix Is:\n"); // print the second matrix for(i=0; i<r2; i++) { for(j=0; j<c2; j++) </pre>
--	--

<pre> if(r2==c1) { printf("\n Enter First Matrix:"); for(i=0; i<r1; i++) { for(j=0; j<c1; j++) scanf("%d",&a[i][j]); } printf("\n Enter Second Matrix: "); for(i=0; i<r2; i++) { for(j=0; j<c2; j++) scanf("%d",&b[i][j]); } </pre>	<pre> printf(" %d ",b[i][j]); printf("\n"); } printf("Multiplication of the Matrices:\n"); for(i=0; i<r1; i++) { for(j=0; j<c2; j++) { c[i][j]=0; for(k=0; k<r1; k++) c[i][j]+=a[i][k]*b[k][j]; printf("%d ",c[i][j]); } printf("\n"); } } </pre>
<p style="text-align: center;">OUTPUT:</p>	<pre> else { printf("Matrix Multiplication is Not Possibl e"); } getch(); } </pre>
<pre> Enter number of rows and columns of first matrix (MAX 10) 3 3 Enter number of rows and columns of second matri x MAX 10) 3 3 Enter First Matrix:2 2 2 2 2 2 2 2 2 Enter Second Matrix: 3 3 3 3 3 3 3 3 3 The First Matrix Is: 2 2 2 2 2 2 2 2 2 The Second Matrix Is: 3 3 3 3 3 3 3 3 3 Multiplication of the Matrices: 18 18 18 18 18 18 18 18 18 </pre>	

Ques 24.WAP to find the length of a string without using library function.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char string[50];
    int i , length = 0;
    printf ("Enter a string\n");
    gets (string);
    for (i=0; string[i] != '\0'; i++)
        { length++ ;
        }
```

```
printf("The length of a string is the number of
characters in it\n");
printf("So, the length of %s =%d\n", string,
length);
}
```

/*-----

Output

Enter a string
hello
The length of a string is the number of
characters in it
So, the length of hello = 5

Ques 25. Program to compare two strings using user defined function. If strings are identical display “The Two Strings are Identical” otherwise the strings are different.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int count1=0,count2=0,flag=0,i;
    char str1[10],str2[10];
    clrscr();
    puts("Enter a string:");
    gets(str1);
    puts("Enter another string:");
    gets(str2);
    /*Count the number of characters in str1*/
    while (str1[count1]!='\0')
        count1++;
    /*Count the number of characters in str2*/
    while (str2[count2]!='\0')
        count2++;
    i=0 ;
    while ( ( i < count1) && ( i < count2))
        { if(str1[i] == str2[i])
            { i++;
            continue ;
            }
            if (str1[i]<str2[i])
            { flag = -1;
            break;
            }
```

```
if (flag==0)
printf("Both strings are equal\n");
if (flag==1)
printf("String1 is greater than string2\n",
str1, str2);
if (flag == -1)
printf("String1 is less than string2\n", str1,
str2);
getch(); }
```

/*-----

Output

Enter a string:
happy
Enter another string:
HAPPY
String1 is greater than string2

RUN2

Enter a string:
Hello
Enter another string:
Hello
Both strings are equal

RUN3

Enter a string:
gold

<pre> } printf("\n\n Reversed string is:%s", rev); return 0; getch(); } </pre>	<pre> s[i] = s[j] ; s[j] = temp ; j++ , i-- ; } return s ; } </pre>

Ques 28. Write short notes on the following concepts:

- (i) Passing arrays to functions.
- (ii) Enumerated data types.
- (iii) Union.

Ans : (i) Passing Arrays to functions : When we need to pass an entire array to a function, we can pass the name of the array. Entire array is always passed by reference to the function. Following are the rules to pass 1-D array in a function :

Rule 1. The actual argument in the function call should only be the name of the array without any subscript, because name of an array represents base address.

Rule 2. Formal parameters in the function definition must be of array type or pointer type (i.e pointer to first element of the array.) If formal parameter is of array type , it will be implicitly converted to pointer type.

Rule 3. Parameter type in the function declaration should be of array type or pointer type.

<p>Example :</p> <pre> void main() // Calling function. { int arr [5] = { 1 , 4 , 10 , 45 , 65 }; func(arr) ; } void func (int arr [5]) // Called function { int i ; </pre>	<p>Note : When an entire array is to be sent to the called function, the calling function just needs to pass the name of array.</p> <p>In cases where called function does not makes any changes to the array, the array must be received as a constant array by the called function. It prevents any type of unnecessary modifications by called function to the array</p>
--	---

<pre>for (i=0 ; i<5 ; i++) printf(“ %d ” , arr [i]) ; }</pre>	<p>elements.</p>
--	------------------

(ii) **Enumerated Data Types** : This is a user defined data type based on the basic integer type. Enumeration has a set of named integer constants. Each integer value is assigned an identifier, also known as enumeration constant, which can be used as a symbolic name to make program easy to be read.

- Keyword enum is used for this data type.
- *Syntax* : **enum enumeration_name { identifier1 , identifier 2,, identifier n}.**

Enumeration name is optional.

Example : *enum COLORS { RED , BLUE , BLACK , GREEN , YELLOW ,PURPLE} ;*

Now COLORS is a new data type. COLORS is the name given to the set of constants. In case we do not assign any value to a constant , default value for the first one in list , RED has value of 0. Next constants will have sequential values after 0 i.e , 1, 2, 3...so on.

RED = 0 ,BLUE = 1, Black =2 , GREEN = 3 , YELLOW= 4, PURPLE= 5.

We can also initialize symbolic constants explicitly by specific integer values , which may not be in sequence.

Example : *enum COLORS { RED = 3 , BLUE = 7 , BLACK = 0 , GREEN = 5 ,
YELLOW = 2, PURPLE = 10} ;*

(iii) **Union**: This is a user defined data type. It is a collection of variables of different data types.

In Unions we can only store information in one field at any one time. It is like a chunk of memory used to store variables of different data types. When a new value is assigned to a field, the existing data is replaced with the new data.

Syntax of declaration: **union union-name**

{

```
data_type var-name1;
```

```
data_type var-name2 ;  
}
```

Ques 29. What are structures? What is the advantage of structures over array? Differentiate between Structures and Unions.

Ans : Structures are the user defined data types that is of heterogeneous nature , that means it can store information of different data types. Keyword struct is used to declare a structure.

Declaration of structure: `struct structure _name`
`{`
`Data_type var-nmae1;Data_type`
`var-nmae1`
`};`

Example: if we define a structure for a student , then the Related information can be : rollno , name , course , fees. This structure can be declared as :

```
struct student  
{  
    int rollno ;  
    char name [ 20];  
    char course [ 25] ;  
    float fees ;  
};
```

Example : declaration of a structure named date.

```
struct date  
{  
    int day ;  
    int month ;  
    int year ;  
}
```

Each variable name declared inside structure is called member of structure. Structure declaration does not consume any storage space. Example: , we can define a variable by writing :

struct student stud1 ;

E.g : `struct student`
`{`
`int rollno ;`
`char name [20];`
`char course [25] ;`
`float fees ;`
`} stud1,stud2 ;`

Here stud1 and stude2 are two variables of structure name student. Variables are separated by commas.

A separate memory is allocated to variables while declaring them.

Initialization of structures: A structure can be initialized in the same way as other data types

are initialized. Initializing a structure means assigning some constants to the members of the structure. If explicitly, then C automatically does that. For int and float members, values are initialized to zero and character and string members are initialized to '\0' by default. Initializers are enclosed in curly braces separated by commas.

<p>Example: struct student { int rollno ; char name [20]; char course [25]; float fees ; } stud1 = { 001 , Vishal , B.Tech , 76000 } ;</p> <p>OR</p> <p>struct student stud1 = { 001 , Vishal , B.Tech , 76000 } ;</p>	<p>Accessing Members of a Structure.</p> <p>Dot operator is used to access the values of members with the help of variables in structure.</p> <p>Syntax : struct_var. member_name ;</p> <p>Example : Stud1.rollno = 001 Stud1.name= Vishal Stud1.course = B.Tech Stud1.fees = 76000</p>
--	--

Advantage of Structure over Arrays : Structure is advantageous than arrays in the sense that it can store information of variables of different data types where as arrays have information of only same data types.

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.

<p>Ques 30.WAP in C to create a structure named account to hold information of account number , account name , balance amount. Display the values of each variable declared . Values must be entered through the keyboard.</p>	<p>Ques 31.WAP in C to create a structure named student to hold information of three students .Program must accept the roll number , name , marks obtained in three tests. Display the information of each student also and average marks obtained.</p>
<pre>#include<stdio.h> #include<conio.h> void main() { struct account { int acc_no ; char acc_name[15] ; float bal ; } ; struct account a1, a2,a3 ;</pre>	<pre>#include<stdio.h> #include<conio.h> void main() { struct student { int rollno ; char name[20] ; int m1, m2, m3 ; float avg ; } ; int i,total ; struct student std[3] ; // Array of structures printf(“ Enter the information of three students \n ”) ; for(i=0 ; i< 3; i++)</pre>

```

printf(“ Enter the account no , account name and
      balance information \n”);

      // Reading the values from keyboard
scanf( “ %d %s %f” , &a1.acc_no ,&a1.acc_name
      , & a1.bal ) ;
scanf( “ %d %s %f” , &a2.acc_no ,&a2.acc_name
      , & a2.bal ) ;
scanf( “ %d %s %f” , &a3.acc_no ,&a3.acc_name
      , & a3.bal ) ;

      // Displaying the values
printf( “ \n %d %s %f” , a1.acc_no , a1.acc_name
      , a1.bal ) ;
printf( “ \n %d %s %f” , a2.acc_no ,a2. acc_name
      , a2.bal ) ;
printf( “ \n %d %s %f” , a3.acc_no , a3.acc_name
      , a3.bal ) ;

      getch();
}      // End of main

```

```

      {
printf( “Enter the roll no. of student %d
      = ” , i+1);

scanf(“ %d ” , &std[i].rollno ) ;

printf( “Enter the name of student %s
      = ” , i+1 ) ;
scanf(“%s ” , &std[i].name ) ;

printf( “Enter the marks1= “);
scanf(“%d ” , &std[i].m1 ) ;

printf( “Enter the marks2 = “);
scanf( “%d ” , &std[i].m2);

printf( “Enter the marks3 = “);
scanf( “%d ” , &std[i].m3);
      }

      for(i=0 ; i<3 ; i++)
      {

total= std[i].m1 + std[i].m2
      + std[i].m3 ;

      std[i].avg = total/3.0 ;
printf( “ %d %s %d %d %d % f\n ” ,

      std[i].rollno,std[i].name,std[i].m1,
      std[i].m2,std[i].m3,std[i].avg) ;
      } // End of for loop

      getch() ;
}      // End of main()

```

Ques 32 : How an array of structure is created ? Explain with the help of a program.

Ans : It is possible to create an array whose elements are of structure type. Such an array is called Array of Structures. If we consider a book example where we have to display information of book name , author name , number of pages and its price then it can be stored in a variable of type struct

book. We can create two variable book1 , book2 and read and display the information about two books..

But in case if we want to access the information of suppose 100 books , then we will have to create 100 variables like book1 , book2, book3 ,.....book100 . Also we will have to read and display the information separately for each book. This will be a very long , difficult process ,and time consuming also. So array of structures will provides an easy way to store information of 100 books.

General Syntax :

```

    struct struct_name
    {
        Data_type1 member_name1 ;
        Data_type2 member_name2 ;
        Data_type3 member_name3 ;
        .....
        Data_type n member_name n ;
    }; struct struct_name struct_var [ index ];
gram of Array of Structure for Book information */

```

```

#include<stdio.h>
#include<conio.h>
void main
{
    struct book
    {
        char Name[30];
        char author[20] ;
        int pages ;
        float price ;
    };
    struct book b[ 100 ];    //array of structures.
    int i;
    for(i=0;i<100;i++)
    {
        printf("\n Enter details of %d Book ",i+1);

        printf("\n\t Enter Book Name : ");
        scanf("%s",&b[i].name);

        printf("\n\t Enter Author Name : ");
        scanf("%s",&b[i].author);
    }
}

```

```
printf("\n\t Enter No. of pages :  
");scanf("%d",&b[i].pages);  
  
printf("\n\t Enter Book Price :  
");scanf("%f",&b[i].price);  
} // End of loop  
  
printf("\n Details of all the Books ");for(i=0;i<  
100 ; i++)  
printf("\n%d\t %s\t%s\t%d \t %d ",b[i].name ,b[i].author,b[i].pages, b[i].price );getch();  
// End of main()  
}
```

[END of 4th UNIT]

UNIT 5

Module – 5: (Pointers& File Handling)

- Pointers
- Introduction to Macros, Preprocessors

- Introduction to Dynamic memory allocation
 - malloc(),calloc() and free() functions
 - Notion to Linked lists
 - Introduction to file handling
 - How to read,open,close files.
 - Basic Program/s on file handling
-

Short Questions & Answers

Ques 1. What are Macros? Give examples.

Ans. Macros are the identifiers that represent statements or expressions. To associate meaningful identifiers with constants, keywords, and statements or expressions, **#define** directive is used.

If we define a macro i. e **SQUARE(x) x*x**. Here the macro determines the square of the given number.

Ques 2. What is Macro template?

Ans. Macro template is defining the macro at the beginning of a program using **'#' directive**. For example

```
#define CUBE (X) ( X * X* X ) .
```

Ques 3. What are the types of Macro?

Ans. (a) Object type Macro (b) Function types Macro.

Ques 4. What is undef directive?

Ans. This cause the specified identifier to be no longer defined as a macro name. For example:

```
.# undef PI
```

Ques 5. What is the difference between #include<stdio.h> and #include"stdio."h.

Ans. These are source file inclusion directives, the first way of inclusion searches the prespecified list of directories for the source file within <>. The second way the source file is first searched in the current working directory, if the search fails then prespecified list of directories are searched. If still the search fails, then compiler gives an error unable to include file.

Long Questions & Answers

Ques 6. What is dynamic memory allocation? What are malloc() and calloc() and free() functions?

Ans. Dynamic memory allocation deals with allocation & de-allocation of memory space at the time of execution of a program. This technique is used where number of data and information is not known beforehand and it may increase or decrease in future depending on certain situations. For Example

Let a company launches new model of a car and starts the booking. The number of customers interested in booking the car is not known apriori. So here structures, arrays can be allocated memory at run time rather than at compile time.

- The memory allocated by system to a program is divided into three portions (i) Memory for functions (ii) Memory for data (iii) Free memory called *heap* which may be required during execution of the program.
- Some *library functions* used in dynamic memory allocation have their *prototype definition* in header file `<stdlib.h>`. These functions are as follows:-
- **malloc()**: This stands for *memory* allocation. This function allocates a block of memory of the size of its argument in bytes. It returns a void pointer to the first byte of the memory block

Syntax: void * malloc (size) ;

e.g: int *ptri;

```
ptri= (int*) malloc (n *sizeof(int)) ;
```

The function *malloc* allocates $n * \text{sizeof(int)}$ bytes of memory and returns the value of pointer ptri.

Return value of malloc is the value of ptri which is the *address of the first byte* of the memory block. Let $n=2$, then total bytes occupied are $2 \times 2=4$ bytes. This is how an array can be allocated memory at run time. In case allocation is not possible due to lack of sufficient memory, the return value will be NULL pointer. For example Code can be:

```
if (ptri= = NULL)
```

```
printf ("Error in memory allocation") ;
```

```
exit(1) ;
```

Similarly for *character array*: char * ptrch;

```
ptrch= (char*) malloc (n *sizeof(char)) ;
```

In the same way we can create *n structures dynamically* as follows:

```
struct book
```

```
{
```

```
cahr name[15] ;int
```

```
pages ;
```

```
}
```

```
struct book*ptrb ;
```

```
ptrb = (struct book*) malloc(n*sizeof(struct book)) ;
```

(ii) calloc(): This stands for *calculated* allocation. This function is used to allocate an array of memory blocks and initializes each block member to 0. It returns a void pointer to the first byte of the first block of the memory. As compared to malloc function *calloc has two arguments.*

Syntax: void* calloc (n , size_mem) ; // *Here n is no.of blocks; size_m is size of each block*

(iii) realloc(): This function reallocates a memory block of size smaller or bigger than the size of the block earlier allocated by malloc().

Syntax: void* realloc (void *ptr , new_size) ;

e.g: float *ptrf = (float *) malloc (5 * sizeof(float)) ; // allocates 5 x 4= 20 bytes of an array
ptrf = (float *) realloc (ptrf, 3 * sizeof(float)) ; // resized array of 3x 4=12 bytes.

(iv) free() : This function is used to de-allocate the memory allocated by the functions malloc() , calloc() and realloc(). The whole allocated memory block is released.

Syntax: void free (void *ptr) ;

Ques 7. What is File in C? What are different types of file used in file handling?

Ans. A **file** is a collection of bytes stored on a secondary storage device, which is generally a disk of some kind. There are two kinds of files that programmers deal with text files and binary files.

A **text file** can be a stream of characters that a computer can process sequentially. It is not only processed sequentially but only in forward direction.

For this reason a text file is usually opened for only one kind of operation (reading, writing, or appending) at any given time.

A **binary file** is no different to a text file. It is a collection of bytes. In C Programming Language a byte and a character are equivalent.

Ques 8. What is linked List? Explain.

Ans : The major application of dynamic memory allocation lies in creation of link lists. The objects in the link lists are linked together not like as arrays sequentially one after another, rather objects can be stored randomly in memory , but each element in the list keeps the address of the next element in the list , or address of both the neighbours, i.e : the previous as well as the next in the list. Each location is called a **node of link list** .Each node has two fields “data and address”. If the node contains only one memory address, it is called a **singly linked list**. But if it contains **address of just previous & next node** then it is known as a **doubly linked list**.

Each node of a link list is created using structures where data member of a structure node is pointer to that structure.

Creation of a Singly Link List:

Struct List

```
{ char name
    [20];int
    number;
    struct List *nextnode ;
```

```
}; struct List *First ;           // First is a pointer of the type struct List.
```

Now the structure object pointed to by First must be allocated memory so as to store the data items. This is Implemented using malloc()as given below:

```
First = (struct List*) malloc (sizeof
(struct List)) ;typedef struct List
Node ;
```

```
First = (Node*) malloc (sizeof(Node)) ;
```

The data items in this node may be initialized, as well as, a memory block may be created for the next node.

```
name = "Dennis Ritchie";
First -> number = 80 ;
First -> Next = (Node* malloc) (sizeof (Node)) ;
```

Ques 9. Write statement in C language to open and close a file.**Ans : Opening a file:**

The general format of the function used for opening a file is

```
FILE *fp;
fp=fopen("filename", "mode");
```

The first statement declares the variable fp as a pointer to the data type FILE. As stated earlier, File is a structure that is defined in the I/O Library. The second statement opens the file named filename and assigns an identifier to the FILE type pointer fp. fopen() contain the file name and mode (the purpose of opening the file).

r is used to open the file for read only.

w is used to open the file for writing only.

a is used to open the file for appending data to it.

Closing a File

A file must be closed as soon as all operations on it have been completed. This would close the file associated with the file pointer. The input output library supports the function to close a file.

Syntax to close file
fclose(file pointer);

Example:

```
#include<stdio.h>
#include<conio.h>
void main(void)
{
    FILE *myfile;
    char c;
    myfile = fopen("firstfile.txt", "r");

    if (myfile == NULL)

        printf("File doesn't exist\n");
    else {
        do {
            c = getc(myfile);
            putchar(c); } while (c != EOF); }
        fclose(myfile);
    }
```

Ques 10.What are various File operations functions in C:

Ans : The various file operations are:

Function Name	Operation
fopen()	Creates a new file. Opens an existing file.
fclose	Closes a file which has been opened for use
getc()	Reads a character from a file
putc()	Writes a character to a file
fprintf()	Writes a set of data values to a file
fscanf()	Reads a set of data values from a file
getw()	Reads a integer from a file
putw()	Writes an integer to the file
fseek()	Sets the position to a desired point in the file

ftell() Gives the current position in the file

rewind() Sets the position to the beginning of the file

Ques 11. What types of errors may occur in file handling? Ans. (a) Omission of File Pointer.

- (b) Opening a File which does not exist.
- (c) Reading a File which is in write mode.
- (d) Writing into a file which is in read mode.

Ques 12. Write a C program to create a file called emp.rec and store information about a person, in terms of his name, age and salary.

Ans.

```
#include <stdio.h>
#include <conio.h>
void main()
{
    FILE *fptr;
    char name[20];
    int age;
    float salary;

    fptr = fopen ("emp.rec", "w"); /*open for writing*/
    if (fptr == NULL)
    {
        printf ("File does not exists\n");
        return ;
    }

    printf("Enter the name\n");
    scanf("%s", name);
    fprintf(fptr, "Name = %s\n", name);

    printf("Enter the age\n");
    scanf("%d", &age);
    fprintf(fptr, "Age = %d\n", age);

    printf ("Enter the salary\n");
    scanf("%f", &salary);
    fprintf( fptr , "Salary = %.2f\n", salary) ;
    fclose(fptr);
}
```

/*-----

Output

Enter

t
h
e
n
a
m
e
P
r
a
b
h
u

Enter the age25

E

n
t
e
r
t
h
e
s
a
l
a
r
y
2
5
0
0
0

Please note that you have to open the file called emp.rec in the directory

[END of 5th UNIT]