CONTENTS

UNIT – I

1.1 Introduction

About C:

C is a programming language developed at AT&T's Bell Laboratories of USA in 1972. It was designed and written by a man named Dennis Ritchie. In the late seventies C began to replace the more familiar languages of that time like PL/I, ALGOL etc.

History of C:

By 1960 a hoard of computer languages had come into existence, almost each for specific purpose. For example, COBOL was being used for Commercial Applications, FORTRAN for Engineering and Scientific Applications and so on. At this stage people started thinking that instead of learning and using so many languages, each for a different purpose, why not use only one language which can program all possible applications. Therefore, an international committee came out with a language called ALGOL 60. However, ALGOL 60 never really became popular because it is too abstract, too general. To reduce this new language called Combined Programming Language (CPL) was developed at Cambridge University. However, CPL turned out to be so big, having so many features, that it was hard to learn and difficult to implement.

Basic Combined Programming Language (BCPL), developed by Martin Richards at Cambridge University aimed to solve this problem by bringing CPL down to its basic good features. Around the same time a language called B was written by Ken Thompson at

AT&T's Bell Labs. Ritchie inherited the features of B and BCPL, added some of his own and developed C. Ritchie's main achievement is the restoration of the lost generality in BCPL and B, and still keeping powerful. C Stand between……

Problem oriented languages or High level languages:

These languages have been designed to give a better programming efficiency, i.e. faster program development. Examples are FORTRAN, BASIC, and PASCAL.

Machine oriented languages or Low level languages:

These languages have been designed to give a better machine efficiency, i.e faster program execution. Examples are Assembly languages and Machine language.

C stands in between these two categories. That's why it is often called as a middle level language, since it was designed to have both a relatively good programming efficiency (as compared to Machine oriented language) and relatively good machine efficiency (as compared to Problem oriented languages).C is a

- general-purpose,
- high-level language

that was originally developed by Dennis M. Ritchie to develop the UNIX operating system at Bell Labs. C was originally first implemented on the DEC PDP-11 computer in 1972.

In 1978, Brian Kernighan and Dennis Ritchie produced the first publicly available description of C, now known as the K&R standard.

The UNIX operating system, the C compiler, and essentially all UNIX application programs have been written in C. C has now become a widely used professional language for various reasons

- Easy to learn
- Structured language
- It produces efficient programs
- It can handle low-level activities
- It can be compiled on a variety of computer platforms

Facts about C

C was invented to write an operating system called UNIX.

C is a successor of B language which was introduced around the early 1970s. The language was formalized in 1988 by the American National Standard Institute (ANSI). The UNIX OS was totally written in C.

Today C is the most widely used and popular System Programming Language.

Why use C?

C was initially used for system development work, particularly the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language. Some examples of the use of C might be –

Operating Systems

Language Compilers

Assemblers

Text Editors

Print Spoolers

Network Drivers
Modern Programs
Databases
Language Interpreters
Utilities
C Programs

1.2 Constants

A C program can vary from 3 lines to millions of lines and it should be written into one or more text files with extension ".c".

For example, hello.c.

we can use "vi", "vim" or any other text editor to write your C program into a file.

Constants/Literals

A constant is a value or an identifier whose value cannot be altered in a program.

Or

Constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.

Eg:

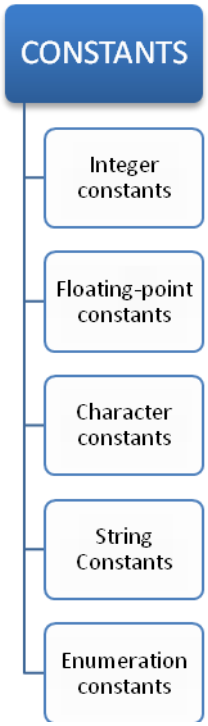
1, 2.5, "C programming is Easy to understand"

As mentioned, an identifier also can be defined as a constant.

Example

```
const double PI = 3.14
```

Here, PI is a constant. Basically what it means is that, PI and 3.14 is same for this program.



- Integer constants
- Character constants
- String constants
- Enumeration constants
- Floating-point constants

1.2.1 Integer constants

A integer constant is a numeric constant (associated with number) without any fractional or exponential part. There are three types of integer constants in C programming:

- decimal constant(base 10)
- octal constant(base 8)
- hexadecimal constant(base 16)

For example:

```
Decimal constants: 0, -9, 22 etc
Octal constants: 021, 077, 033 etc
Hexadecimal constants: 0x7f, 0x2a, 0x521 etc
```

In C programming, octal constant starts with a 0 and hexadecimal constant starts with a 0x.

1.2.2 Floating-point constants

A floating point constant is a numeric constant that has either a fractional form or an exponent form. For example:

```
-2.0
0.0000234
-0.22E-5
```

Note: E-5 = 10^{-5}

1.2.3 Character constants

A character constant is a constant which uses single quotation around characters. For example: 'a', 'l', 'm', 'F'

Escape Sequences

Sometimes, it is necessary to use characters which cannot be typed or has special meaning in C programming. For example: newline(enter), tab, question mark etc. In order to use these characters, escape sequence is used.

For example: \n is used for newline. The backslash (\ ) causes "escape" from the normal way the characters are interpreted by the compiler.

Escape Sequences	
Escape Sequences	Character Description
\b	Backspace
\f	Form feed
\n	Newline
\r	Return
\t	Horizontal tab
\v	Vertical tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark
\?	Question mark
\0	Null character

1.2.4 String constants

String constants are the constants which are enclosed in a pair of double-quote marks. For example:

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

1.2.5 Enumeration constants

Keyword enum is used to define enumeration types. For example:

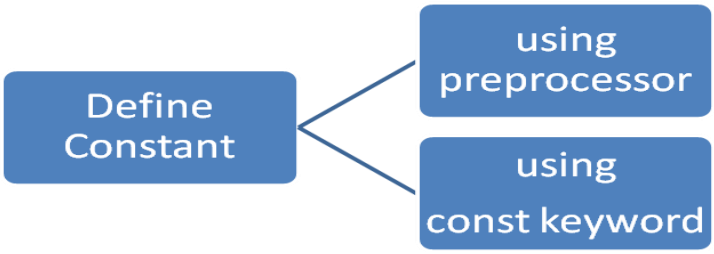
```
enum color {yellow, green, black, white};
```

Here, color is a variable and yellow, green, black and white are the enumeration constants having value 0, 1, 2 and 3 respectively.

An integer literal can also have a suffix that is a combination of U and L, for unsigned and long, respectively. The suffix can be uppercase or lowercase and can be in any order.

Defining Constants

To define a constant by using preprocessor and const keyword. Thatt are shown below.



- Using preprocessor.
- Using const keyword.

The #define Preprocessor

Given below is the form to use #define preprocessor to define a constant

#define identifier value

The following example explains it in detail

```
#include <stdio.h>
#define LENGTH 10
#define WIDTH  5
#define NEWLINE '\n'
int main() {
int area;
area = LENGTH * WIDTH;
printf("value of area : %d", area);
printf("%c", NEWLINE);
return 0;
}
```

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

value of area : 50

The const Keyword

You can use const prefix to declare constants with a specific type as follows const type variable = value;

The following example explains it in detail

```
#include <stdio.h>
int main() {
constint  LENGTH = 10;
constint  WIDTH = 5;
const char NEWLINE = '\n';
int area;
area = LENGTH * WIDTH;
printf("value of area : %d", area);
printf("%c", NEWLINE);
return 0;
}
```

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

value of area : 50

1.3 Variables

A variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

```
int playerScore = 95;
```

Here, playerScore is a variable of integer type. The variable is holding 95 in the above code.

The value of a variable can be changed, hence the name 'variable'.

Rules for writing variable name in C

1. A variable name can have letters (both uppercase and lowercase letters), digits and underscore only.
2. The first letter of a variable should be either a letter or an underscore. However, it is discouraged to start variable name with an underscore. It is because variable name that starts with an underscore can conflict with system name and may cause error.
3. There is no rule on how long a variable can be. However, the first 31 characters of a variable are discriminated by the compiler. So, the first 31 letters of two variables in a program should be different.
4. In C programming, you have to declare a variable before you can use it.
5. set of operations that can be applied to the variable
6. Variables are case sensitive

1.4 Symbolic Constant

A symbolic constant is an "variable" whose value does not change during the entire lifetime of the program.

- The character may represent a numeric constant, a character constant, or a string. When the program is compiled, each occurrence of a symbolic constant is replaced by its corresponding character sequence.
- They are usually defined at the beginning of the program.

- The symbolic constants may then appear later in the program in place of the numeric constants, character constants, etc., that the symbolic constants represent.

Syntax

#define symbolic_name value

For example:

A C program consists of the following symbolic constant definitions.

```
#define PI 3.141593
#define TRUE 1
#define FALSE 0
```

#define PI 3.141593 defines a symbolic constant PI whose value is 3.141593. When the program is preprocessed, all occurrences of the symbolic constant PI are replaced with the replacement text 3.141593.

Note that the preprocessor statements begin with a #symbol, and are not end with a semicolon. By convention, preprocessor constants are written in UPPERCASE.

/* program illustrating use of declaration, assignment of value to variables also explains how to use symbolic constants.

Program to calculate area and circumference of a circle */

```
#include<stdio.h>
#include<conio.h>
#define PI 3.1415    /* NO SEMICOLON HERE */
void main ()
```

```
{
float rad = 5;     /*DECLARATION AND ASSIGNMENT*/
floatarea,circum;     /* DECLARATION  OF VARIABLE*/
area=PI*rad*rad;
circum=2*PI*rad;
printf("AREA OF CIRCLE = %f\n",area);
printf("CIRCUMFERENCE OF CIRCLE =%f\n",circum);
getch();
clrscr();
}
```

OUTPUT :

AREA OF CIRCLE =78.537498

1.5. Character set

Character:- It denotes any alphabet, digit or special symbol used to represent information.

Use: - These characters can be combined to form variables. C uses constants, variables, operators, keywords and expressions as building blocks to form a basic C program.

Character set: - The character set is the fundamental raw material of any language and they are used to represent information. Like natural languages, computer language will also have well defined character set, which is useful to build the programs.

The characters in C are grouped into the following two categories:

Source character set

a. Alphabets

b. Digits

c. Special Characters

d. White Spaces

Execution character set

a. Escape Sequence

Source character set

ALPHABETS

Uppercase letters	A-Z
Lowercase letters	a-z

DIGITS 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

SPECIAL CHARACTERS

~	tilde	%	percent sign
|	vertical bar	@	at symbol
+	plus sign	<	less than
_	underscore	-	minus sign
>	greater than	^	caret
#	number sign	=	equal to
&	ampersand	$	dollar sign
/	slash	(	left parenthesis
*	asterisk	\	back slash
)	right parenthesis	′	apostrophe
:	Colon	[	left bracket
"	Quotation mark	;	semicolon

]	right bracket	!	Exclamation mark
,	comma	{	left flower brace
?	Question mark	.	dot operator
}	right flower brace		

WHITESPACE CHARACTERS

\b	blank space	\t	horizontal tab
\v	vertical tab	\r	carriage return
\f	form feed	\n	new line
\\	Back slash	\’	Single quote
\"	Double quote	\?	Question mark
\0	Null	\a	Alarm (bell)

Execution Character Set

Certain ASCII characters are unprintable, which means they are not displayed on the screen or printer. Those characters perform other functions aside from displaying text. Examples are backspacing, moving to a newline, or ringing a bell. They are used in output statements. Escape sequence usually consists of a backslash and a letter or a combination of digits. An escape sequence is considered as a single character but a valid character constant. These are employed at the time of execution of the program. Execution characters set are always represented by a backslash (\) followed by a character. Note that each one of character constants represents one character, although they consist of two characters. These characters combinations are called as escape sequence.

Backslash character constants

Character Result	ASCII value	Escape Sequence
Null Null	000	\0
Alarm (bell) Beep Sound	007	\a
Back space Moves previous position	008	\b
Horizontal tab Moves next horizontal tab	009	\t
New line Moves next Line	010	\n
Vertical tab Moves next vertical tab	011	\v
Form feed Moves initial position of next page	012	\f
Carriage return Moves beginning of the line	013	\r
Double quote Present Double quotes	034	\"
Single quote Present Apostrophe	039	\'
Question mark Present Question Mark	063	\?
Back slash Present back slash	092	\\
Octal number	\000	
Hexadecimal number	\x	

1.6 Keywords and Identifiers

- Keywords are predefined, reserved words used in programming.
- Keywords are standard identifiers that have standard predefined meaning in C. Keywords are all lowercase,

Points to remember:

1. Keywords can be used only for their intended purpose.
2. Keywords can't be used as programmer defined identifier.
3. The keywords can't be used as names for variables.

Syntax keyword variable; Eg: int money;
Here, int is a keyword that indicates 'money' is a variable of type integer.

Identifiers

Identifiers are the names you can give to entities such as variables, functions, structures etc.
Identifier names must be unique. They are created to give unique name to a C entity to identify it during the execution of a program.

For example:

```
int money;
doubleaccountBalance;
```

Here, money and accountBalance are identifiers.

Also remember, identifier names must be different from keywords. You cannot use int as an identifier because int is a keyword.

Rules for writing an identifier

i. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscore only.
ii. The first letter of an identifier should be either a letter or an underscore. However, it is discouraged to start an identifier name with an underscore. It is because identifier that starts with an underscore can conflict with system names.
iii. In such cases, compiler will complain about it. Some system names that start with underscore are _fileno, _iob, _wfopen etc.
iv. There is no rule on the length of an identifier. However, the first 31 characters of identifiers are discriminated by the compiler. So, the first 31 letters of two identifiers in a program should be different
v. Identifier name must be a sequence of letter and digits, and must begin with a letter.
vi. The underscore character ('_') is considered as letter.
vii. Names shouldn't be a keyword (such as int, float, if, break, for etc)
viii. Both upper-case letter and lower-case letter characters are allowed. However, they're not interchangeable.
ix. No identifier may be keyword.
x. No special characters, such as semicolon, period, blank space, slash or comma are permitted

1.7 Declaration of Variables

- Variables should be declared in the C program before to use.
- Memory space is not allocated for a variable while declaration.
- It happens only on variable definition.

Example :int player Score = 95;

Variable Declaration in C

- A variable declaration provides assurance to the compiler that there exists a variable with the given type and name so that the compiler can proceed for further compilation without requiring the complete detail about the variable. A variable definition has its meaning at the time of compilation only, the compiler needs actual variable definition at the time of linking the program.
- A variable declaration is useful when you are using multiple files and you define your variable in one of the files which will be available at the time of linking of the program. You will use the keyword extern to declare a variable at any place. Though you can declare a variable multiple times in your C program, it can be defined only once in a file, a function, or a block of code.

Example

Try the following example, where variables have been declared at the top, but they have been defined and initialized inside the main function −

```
#include <stdio.h>

// Variable declaration:
externint a, b;
externint c;
extern float f;

int main () {

   /* variable definition: */
int a, b;
int c;
float f;

   /* actual initialization */
   a = 10;
   b = 20;

   c = a + b;
printf("value of c : %d \n", c);

   f = 70.0/3.0;
printf("value of f : %f \n", f);

return 0;
}
```

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

value of c : 30

value of f : 23.333334

The same concept applies on function declaration where you provide a function name at the time of its declaration and its actual definition can be given anywhere else.

For example

```
// function declaration
intfunc();

int main() {

   // function call
int i = func();
}

// function definition
intfunc() {
return 0;
}
```

Assigning values to variables There are two kinds of expressions in C Lvalues and Rvalues in C

- lvalue − Expressions that refer to a memory location are called "lvalue" expressions. An lvalue may appear as either the left-hand or right-hand side of an assignment.

• rvalue – The term rvalue refers to a data value that is stored at some address in memory. An rvalue is an expression that cannot have a value assigned to it which means an rvalue may appear on the right-hand side but not on the left-hand side of an assignment.

Variables are lvalues and so they may appear on the left-hand side of an assignment. Numeric literals are rvalues and so they may not be assigned and cannot appear on the left-hand side. Take a look at the following valid and invalid statements

int g = 20; // valid statement

10 = 20; // invalid statement; would generate compile-time error

1.8 Assigning Values to Variables

Variable initialization means assigning a value to the variable.

Variable declaration

Syntax: data_type variable_name;

Eg: int x, y, z; char flat, ch;

Variable initialization

Syntax: data_type variable_name = value;

Eg: int x = 50, y = 30; char flag = 'x', ch='1';

There are two types of variables in c program they are,

i. Local variable

ii. Global variable

i. Local variable in c:

- The scope of local variables will be within the function only.
- These variables are declared within the function and can't be accessed outside the function.
- In the below example, m and n variables are having scope within the main function only. These are not visible to test function.
- Likewise, a and b variables are having scope within the test function only. These are not visible to main function.

ii. Global Variable In C

- The scope of global variables will be throughout the program.
- These variables can be accessed from anywhere in the program.
- This variable is defined outside the main function.
- This variable is visible to main function and all other sub functions.

1.9 Declaring variable as a constant

Declaring a variable is very easy. First you have to declare the type-name. After the type-name you place the name of the variable. The name of a variable can be anything you like as long it includes only letters, underscores or numbers.

To declare a constant is not much different then declaring a variable. The only difference is that you have the word const in front of it.

The *const* keyword is to declare a constant

Syntax: const type-name variable name

Eg:

```
int main()
    {
        const float PI = 3.14;
        char = 'A';
        return 0;
    }
```

Difference between Local and Global Variable

Local Variable	**Global Variable**
A local variable is a variable that is declared inside a function	A global variable is a variable that is declared outside all functions
A local variable can only be used in the function where it is declared	A global variable can be used in all functions
Local variables must always be defined at the top of a block	Global variable is defined at the top of the programm file and it can be visible
When a local variable is defined it is not initialized by the system, must initialize it	Global variables are initialized automatically by the system

Eg:

```
#include<stdio.h>

	// Global variables
	int A;
	int B;

	int Add()
	{
		return A + B;
	}

	int main()
	{
		int answer; // Local variable
		A = 5;
		B = 7;
		answer = Add();
		printf("%d\n",answer);
		return 0;
	}
```

We will see the Difference between Variable Declaration and Variable Definition by following table.

Variable Declaration	Variable definition
Declaration tells the compiler about data type and size of the variable	Definition allocates memory for the variable
Variable can be declared many times in a program	it can happen only one time for a variable in a programm
The assignment of properties and identification to a variable	Assignments of storage space to a variable

And some examples of variable types are given below.

Type	Description
Char	Typically a single octet(one byte). This is an integer type.
Int	The most natural size of integer for the machine.
Float	A single-precision floating point value.
double	A double-precision floating point value.
Void	Represents the absence of type.

C programming language also allows defining various other types of variables, which we will cover in subsequent chapters like Enumeration, Pointer, Array, Structure, Union, etc. For this chapter, let us study only basic variable types.

1.10 Data Types

Data types simply refer to the type and size of data associated with variables and function. This data types are declared using the keywords char, int, float and double respectively. Typical memory requirements of the basic data types are given below

- Fundamental Data Types
    - Integer types
    - Floating type
    - Character type
- Derived Data Types
    - Arrays
    - Pointers
    - Structures
    - Enumeration

Basic data types are given below

Data Type	Meaning	Size <byte>	Minimal range
Char	Character	1	-128 to 127
Int	Integer	2	-32,768 to 32, 768
Float	Single precision real number	4	3.4E-38 to 3.4E+38
Double	Double precision real number	8	1.7E-308 to
Void	valueless	0	1.7E+308

1.10.1 Integer data types

Integers are whole numbers that can have both positive and negative values, but no decimal values. Example: 0, -5, 10. In C programming, keyword int is used for declaring integer variable. For example:

```
int id;
```

Here, id is a variable of type integer.

1.10.2 Floating types

Floating type variables can hold real numbers such as: 2.34, -9.382, 5.0 etc. You can declare a floating point variable in C by using either float or double keyword. For example:

```
floataccountBalance;
doublebookPrice;
```

Here, both accountBalance and bookPrice are floating type variables.

1.10.3 Character types

Keyword char is used for declaring character type variables. For example: char test = 'h',Here, test is a character variable. The value of test is 'h'.

Questions and Answers

1. Define Constants in C.

- The constants refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.
- Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.
- The constants are treated just like regular variables except that their values cannot be modified after their definition

2. Mention the Constants Types

- Integer constants
- Character constants
- String constants
- Enumeration constants
- Floating-point constants

3. What is meant by Enumerated data type? Enumerated data is a user defined data type in C language.

Enumerated data type variables can only assume values which have been previously declared.

Example :

enum month { jan = 1, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec };

4. What are Keywords?

Keywords are certain reserved words that have standard and pre-defined meaning in „C". These keywords can be used only for their intended purpose.

5. What do you mean by variables in "C"?

A variable is a data name used for storing a data value.

- It can be assigned different values at different times during program execution.
- It can be chosen by programmer in a meaningful way so as to reflect its function in the program.

  Examples: Sum

6. Difference between Local and Global variable in C.

Local

These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program. As such, they are normally implemented using a stack. Local variables cease to exist once the function that created them is completed. They are recreated each time a function is executed or called.

Global

These variables can be accessed by any function comprising the program. They are implemented by associating memory locations with variable names. They do not get recreated if the function is recalled.

7. What is local Variable?

These variables only exist inside the specific function that creates them. They are unknown to other functions and to the main program.

8. What is Global Variable?

The variable that are declared outside all the functions are called Global variable

9. What is the difference between declaring a variable and defining a variable?

- Declaring a variable means describing its type to the compiler but not allocating any space for it.
- Defining a variable means declaring it and also allocating space to hold the variable.

10. What is variable?

A variable is a container (storage area) to hold data.

To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location

11. Write a C program to print Hello world without using semicolon

Solution: 1

```
#include<stdio.h>
void main(){
   if(printf("Hello world")){
   }
}
```

Solution: 2

```
#include<stdio.h>
void main(){
   while(!printf("Hello world")){
   }
}
```

Solution: 3

```
#include<stdio.h>
void main(){
   switch(printf("Hello world"))
}
```

12. Write a C program for area of circle

```
#include <stdio.h>
#define PI 3.141
int main(){
  float r, a;
  printf("Radius: ");
  scanf("%f", &r);
  a = PI * r * r;
  printf("%f\n", a);
  return 0;
}
```

Answer the following questions

1. Explain in Detail about Constants and their types.
2. Explain in detail about Variables and briefly describe about their types.
3. Describe briefly about local variables with examples
4. Describe briefly about global variables with examples
5. Explain in Detail about various Data types and their syntax.

UNIT - II

2.1. Decision Making and Branching

Introduction:

'C' language processes decision making capabilities supports the flowing statements known as control or decision making statements

1. If statement
2. switch statement
3. conditional operator statement
4. Goto statement

If Statement:

The if statement is powerful decision making statement and is used to control the flow of execution of statements The If statement may be complexity of conditions to be tested

(a) Simple if statement
(b) If else statement
(c) Nested If-else statement
(d) Else –If ladder Decision making with if statement

Decision making with IF statement

The syntax is

if (expression)

The expression is evaluated and depending on whether the value of the expression is true (non zero) or false (zero) it transfers the control to a particular statement.

The general form of simple if statement is

```
if (test expression)
{
statement block;
}
statement n;
```

Simple IF statement:

Example:

```
if(code==1)
{
salary=salary+500;
}
printf("%d",salary);
```

The statement block may be a single statement or a group of statements. If the test expression is true, the statement block will be executed otherwise the statement block will be skipped and the execution will jump to the statement n. But if the condition is true, both the statement block and statement n will be executed. Thus if the expression is false only statement n will be executed.

The if... else statement is an extension of the simple if statement

Syntax is:

```
if(testexpression)
{
statementblock;
}
else
{
```

```
    statementblock;
    }
    statement n;
```

Example

```
if (code = = 1)
{
salary = salary + 500;
}
else
{
salary = salary + 250;
}
printf ("%d', salary);
```

If the code is equal to 1, the statement salary = salary + 500 is executed and the control is transferred to statement printf("%d",salary) after skipping the else part.

If the code is not equal to 1, the statement, salary = salary + 500 is skipped and the statement in the else part, salary = salary + 250 is executed before the control reaches the statement printf ("%d", salary);

Nesting of if….else statement

When a series of conditions are to be checked, we may have to use more than one if… else statement in the nested form.

```
if (test condition 1)
{
if (test condition 2)
```

```
{
statement block 1;
}
else
{
statement block 2;
} statement m;
}
else
{
if (test condition 3)
{
statement block 3;
}
else
{
statement block 4
} statement n;
}
statement x;
```

If the test condition 1 is true then, test condition 2 is checked and if it is true, then the statement block 1 will be executed and the control will be transferred to statement m and it will executed and then statement x will be executed.

If the test condition 1 is true but test condition 2 is false, statement block 2 will be executed and the control is transferred to statement m and it will be executed and then statement x will be executed.

If the test condition 1 is false, then test condition 3 is checked and if it is true, statement block 3 will be executed, then control is transferred to statement n and it will be executed and then statement x will be executed.

If the test condition 1 is false and test condition 3 is also false, statement block 4 will be executed, then the control is transferred to statement n and it will be executed and then statement x is executed.

The Else-If ladder

A multi path decision is charm of its in which the statement associated with each else is an If. It takes the following general form.

```
  If (condition1)
St –1;
Else  If (condition2)
St –2;
Else if (condition 3)
St –3;
Else
Default – st;
St –x;
```

This construct is known as the wise-If ladder. The conditions are evaluated from the top of the ladder to down wards. As soon as a true condition is found thestatement associated with it is executed and the control the is transferred to the st-X (i.e.., skipping the rest of the ladder). when all the n-conditions become false then the final else containing the default – st will be executed.

```
Example: If (code = = 1)          Color = “red”;
Else if (code = = 2)    Color = “green”
Else if (code = = 3)     Color = “white”;
Else     Color = “yellow”;
```

If code number is other than 1, 2 and then color is yellow.

When many conditions are to be checked then using nested if...else is very difficult, confusing and cumbersome. So C has another useful built in decision making statement known as switch. This statement can be used as multiway decision statement. The switch statement tests the value of a given variable or expression against a list of case values and when a match is found, a block of statements associated with that case is executed.

GOTO Statement

This statement is used to branch unconditionally from one point to another in the program. This statement goto requires a label to locate the place where the branch is to be made.A label is any valid identifier and must be followed by a colon. The label is placed immediately before the statement where the control is to be transferred.

Different ways of using goto statement are given below:

Syntax: goto label;

Forward jump: In this the position of the label is after the goto statement.

```
goto label;
:
label:
statement n;
```

Example:

```
goto read;
n = 5 * 4;
```

:

:

read:

scanf ("%d", &code);

Backward jump: In this, the position of the label is before the goto statement

label:

statement n;

: goto label;

Example:

. . read:

scanf ("%d", &code);

:

goto read;

n = 5 * 4;

2.2 Decision Making and Looping

The 'C' language provides three loop constructs for performing loop operations they are execution of a statement or set of statement repeatedly is called as looping.

The loop may be executed a specified number of times and this depends on the satisfaction of a test condition.

A program loop is made up of two parts one part is known as body of the loop and the other is known as control condition.

- The while statement
- Do-while statement
- The for statement

While Statement:

This type of loop is also called an entry controlled, is executed and if is true then the body of the loop is executed this process repeated until the Boolean expression becomes false. Ones it becomes false the control is a transferred out the loop. The general form of the while statement is

```
While (boolean expression)
{
body of the loop;
}
```

```
Ex :            i = 1;
While(I<=5)
{   printf("%d",i);
i++; }
```

In the above example the loop with be executed until the condition is false

Do...while Statement:

This type of loop is also called an exist controlled loop statement. The boolean expression evaluated at the bottom of the loop and if it is true then body of the loop executed again and again until the boolean expression becomes false. Ones it becomes false the control is do-while statement is

```
            Do
            {
body of the loop ;
            }
while ( boolean expression)
```

Ex :

```
i = 1;
Do
{
printf("%d",i);
i++;
}
While (i<=5)
```

For Statement:

The for loop is another entry controlled loop that provides a more concise loop control structure the general form of the for loop is

Syntax

```
for (initialisation; test condition; increment)
 {
        body of the loop;
 }
```

Where initialization is used to initialize some parameter that controls the looping action, 'test condition' represents if that condition is true the body of the loop is executed, otherwise the loop is terminated After evaluating information and the new value of the control variable is again tested the loop condition. If the condition is satisfied the body of the loop is again executed it this process continues until the value of the control variable false to satisfy the condition.

Example :

```
for (I=1; I<=5; I++)
{
printf("%d",i);
    }
```

Output : 1 2 3 4 5

Jumps in Loops

Break Statement The break statement can be accomplish by using to exist the loop. When break is encountered inside a loop, the loop is immediately exited and the program continues with the statement which is followed by the loop. If nested loops then the break statement inside one loop transfers the control to the next outer loop.

Example:

```
 for (I=1; I<5; I++)
 {
if ( I == 4)
break;
printf("%d",i);
}
```

Output : 1 2 3

Continue Statement:

The continue statement which is like break statement. Its work is to skip the present statement and continues with the next iteration of the loop.

Example

```
 for (i=1; i<5; i++)
  {
if ( i== 3)
continue;
printf("%d",i);
}
```

Output : 1 2 4 5

In the above example when i=3 then the continue statement will rise and skip statement in the loop and continues for the next iteration i.e..,i=4.

Questions and Answers:

1. Distinguish between while.. and do..while statement in C.

While	DO..while
Executes the statements block if only the while condition is true	Executes the statements within the while block atleast once.
The condition is checked at the starting of the loop.	The condition is checked at the end of the loop

2. Mention the various Decisions making statement available in C.

Statement	Description
if statement	An if statement consists of a Boolean expression followed by one or more statements.
if...else statement	An if statement can be followed by an optional else statement, which executes when the Boolean expression is false.

nested if statements	You can use one if or else if statement inside another if or else if statement(s).
switch statement	A switch statement allows a variable to be tested for equality against a list of values.
nested switch statements	You can use one switch statement inside another switch statement(s).

3. What are the types of looping statements available in C?

C programming language provides following types of loop to handle looping requirements.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
for loop	Execute a sequence of statements multiple times and abbreviates the code that manages the loop variable.
do...while loop	Like a while statement, except that it tests the condition at the end of the loop body
nested loops	one or more loop inside any another while, for or do..While loop.

4. Write a C program to determine a given number is ‘odd’ or ‘even’

```
 #include<stdio.h>
#include<conio.h>
void main()
{
 int n;
 printf("Enter the number");
scanf("%d",&n);   if( n%2 == 0 )
   {
 printf(“NUMBER IS EVEN”);
}
if( n%2 != 0 )
{
printf(“NUMBER IS ODD”);
}
getch();
 }
```

5. Give output of following program

a)

```
void main ()
{
 int n=6, t=1;
 for(;n<10;n=n+2)
printf(“%d %d\n”,n ,++ t);
 }
```

b)

```
 void main()
{
 int a=3 b=5,c,*p,*q; p=&b; q=&a; c=*p % *q; ++(*p);
 printf (“%d %d”,*p,*q);
```

```
printf (“\n %d %d”, c ,b);
 }
```

6 2 6 3

8 3 26

6. What is the output of following program?

```
main ()
{
int d = 1;
do
printf(“%d\n”, d++);
while (d < = 9);
}
```

Displaying integers from 1 to 9

7. Write a C program to Determine the given number is Prime or not

```
#include<stdio.h>
int main()
{
    int num,i,count=0;
    printf("Enter a number: ");
    scanf("%d",&num);
    for(i=2;i<=num/2;i++){
        if(num%i==0){
         count++;
            break;
        }
    }
   if(count==0 && num!= 1)
        printf("%d is a prime number",num);
   else
```

```
    printf("%d is not a prime number",num);
  return 0;
}
```

Sample output:

Enter a number: 5

5 is a prime number

8. Write a c program to find the largest Number

- This program uses only if statement to find the largest number.

```
#include <stdio.h>
int main()
{
    double n1, n2, n3;

    printf("Enter three numbers: ");
    scanf("%lf %lf %lf", &n1, &n2, &n3);

    if( n1>=n2 && n1>=n3 )
        printf("%.2f is the largest number.", n1);

    if( n2>=n1 && n2>=n3 )
        printf("%.2f is the largest number.", n2);

    if( n3>=n1 && n3>=n2 )
        printf("%.2f is the largest number.", n3);
    return 0;
        }
```

This program uses if...else statement to find the largest number.

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

```
int main()
{
    double n1, n2, n3;

    printf("Enter three numbers: ");
    scanf("%lf %lf %lf", &n1, &n2, &n3);

    if (n1>=n2)
    {
        if(n1>=n3)
            printf("%.2lf is the largest number.", n1);
        else
            printf("%.2lf is the largest number.", n3);
    }
    else
    {
        if(n2>=n3)
            printf("%.2lf is the largest number.", n2);
        else
            printf("%.2lf is the largest number.",n3);
    }

    return 0;
}
```

This program uses nested if...else statement to find the largest number

```
#include <stdio.h>
int main()
{
    double n1, n2, n3;
    printf("Enter three numbers: ");
    scanf("%lf %lf %lf", &n1, &n2, &n3);
    if( n1>=n2 && n1>=n3)
        printf("%.2lf is the largest number.", n1);
    else if (n2>=n1 && n2>=n3)
        printf("%.2lf is the largest number.", n2);
    else
        printf("%.2lf is the largest number.", n3);
    return 0;
}
```

Though, the largest number among three numbers is found using multiple ways, the output of these entire programs will be same.

```
Enter three numbers: -4.5
3.9
5.6
5.60 is the largest number.
```

UNIT-III

3.1 Arrays

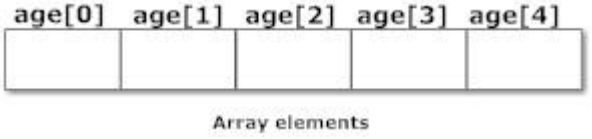
Array is a collection of variables belongings to the same data type. You can store group of data of same data type in an array.or An array is a sequence of data item of homogeneous value (same type). A particular element of the array can referred of its subscript or index in brackets after the array name.

- Array might be belonging to any of the data types
- Array size must be a constant value.
- Always, Contiguous (adjacent) memory locations are used to store array elements in memory.
- It is a best practice to initialize an array to zero or null while declaring, if we don't assign any values to array.

Array elements

Size of array defines the number of elements in an array. Each element of array can be accessed and used by user according to the need of program. For example:

```
int age[5];
```



Array elements

Types of Arrays

1. One dimensional Array
2. Two dimensional Array
3. Multidimensional Array

3.1.1 One-dimensional Arrays

Arrays can be initialized at declaration time in this source code It is not necessary to define the size of arrays during initialization.

Example intage[]={2,4,34,3,4};

In this case, the compiler determines the size of array by calculating the number of elements of an array.

Declaration of one-dimensional array

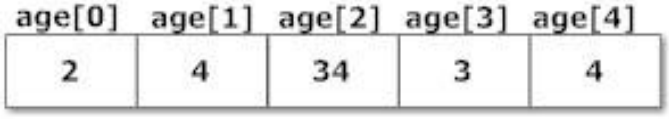
```
data_typearray_name[array_size];

For example:
int age[5];
```

Initialization of one-dimensional array

Type variable name[size];
Example : float height[50];

It is not necessary to define the size of arrays during initialization.
int age[]={2,4,34,3,4};

In this case, the compiler determines the size of array by calculating the number of elements of an array.



Initialization of one-dimensional array

3.1.2 Two Dimensional Arrays

Two dimensional array is nothing but array of array.

syntax :data_typearray_name[num_of_rows][num_of_column]

Two Dimensional Array : Two dimensional array declaration is as follows

Type array-name[row-size][coloumn-size];

Ex :int a[40][90];

3.1.3 Multidimensional Arrays

C programming language allows programmer to create arrays of arrays known as multidimensional arrays.

For example:

Declaration of multi-dimensional array

data_typearray_name[array_size1] [array_size1];

```
float a[2][6];
```

Here, a is an array of two dimension, which is an example of multidimensional array.

For better understanding of multidimensional arrays, array elements of above example can be think of as below:

	Col1	Col 2	Col 3	Col 4	Col 5	Col 6
row 1	a[0] [0]	a[0][1]	a[0] [2]	a[0] [3]	a[0] [4]	a[0] [5]
row 2	a[1] [0]	a[1] [1]	a[1] [2]	a[1] [3]	a[1] [4]	a[1] [5]

Figure Multidimensional Arrays

Initialization of Multidimensional Arrays

In C, multidimensional arrays can be initialized in different number of ways.

```
int c[2][3]={{1,3,0}, {-1,5,9}};
                OR
int c[][3]={{1,3,0}, {-1,5,9}};
                OR
int c[2][3]={1,3,0,-1,5,9};
```

3.1.4 Dynamic Arrays

A dynamic array is simply an array of void ** pointers that is pre-allocated in one shot and that point at the data.

In the linked list you had a full struct that stored the void *value pointer, but in a dynamic array there's just a single array with all of them. This means you don't need any other pointers for next and previous records since you can just index into it directly.

3.2 Handling of Character 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.

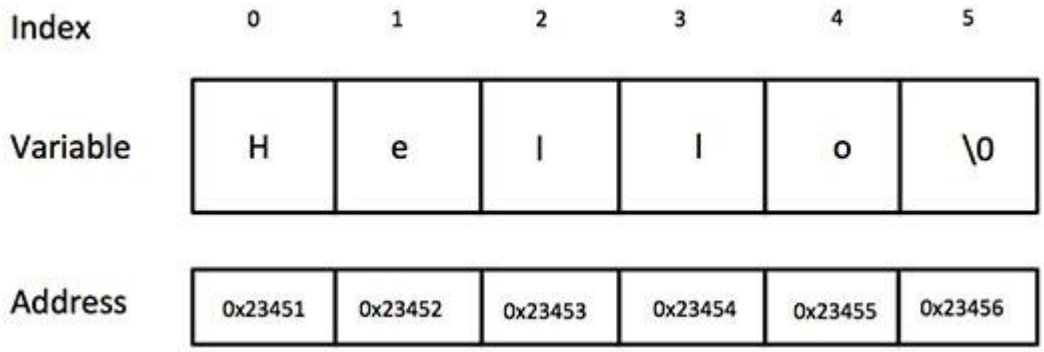
The following declaration and initialization create a string consisting of the word "Hello". To hold the null character at the end of the array, the size of the character array containing the string is one more than the number of characters in the word "Hello."

```
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



3.3 Declaring and Initializing String Variable

String declaration: There is no string data type in c. So string is declared as the array of characters.

Syntax: char variable-name[size];

Eg: char S[10];

String initialization: like the integer array the character array can also initialized at the time of declaration. The two following methods are for string initialization.

char name[13]=”jatinsharma”;

char name[13]={’j’,’a’,’t’,’i’,’n’,’ ’,’s’,’h’,’a’,’r’,’m’,’a’,’\0’};

we have used 13 size for the string because in the last the compiler automatically inserts the ‘\0’ which called null character. And remember ‘ ’ space is also a character.

Run time initialization: we have many functions to initialize the string at run time. These are: scanf(): scanf() function is also used for the other data types and it used '%s' in the function.

Syntax: scanf("%s",variable-name);

Eg: scanf("%s",str);

getchar(): getchar() function is used for read only one character.

Syntax: variable=getchar();

Eg: char a;

a=getchar();

gets(): gets() function is used for input a line of string.

Syntax: char variable-name[size];

gets(variable);

Eg: char str[20];

gets(str);

3.4. Arithmetic Operations on Character

C Programming Allows you to Manipulate on String Whenever the Character is variable is used in the expression then it is automatically Converted into Integer Value called ASCII value

All Characters can be Manipulated with that Integer Value.(Addition,Subtraction)

Examples :

ASCII value of : 'a' is 97

ASCII value of : 'z' is 121

Possible Ways of Manipulation :

- Way 1:

Displays ASCII value[Note that %d in Printf]

```
char x = 'a';
printf("%d",x); // Display Result = 97
```

- Way 2 :

Displays Character value[Note that %c in Printf]

```
char x = 'a';
printf("%c",x); // Display Result = a
```

- Way 3 :

Displays Next ASCII value[Note that %d in Printf]

```
char x = 'a' + 1 ;
printf("%d",x);
// Display Result = 98 ( ascii of 'b' )
```

- Way 4 :

Displays Next Character value[Note that %c in Printf]

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

- Way 5 : Displays Difference between 2 ASCII in Integer[Note %d in Printf]

```
char x = 'z' - 'a';
printf("%d",x);
/* Display Result = 25
   (difference between ASCII of z and a ) */
```

- Way 6 :

Displays Difference between 2 ASCII in Char [Note that %c in Printf]

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

3.5. String Handling Functions

There is lots of string handling functions available in string.h header file. Here are some of the important string manipulation fFunction Purpose
strlen()

strlen() function is used to find length of string. length of string means number of characters in the string.

It accept a string as an argument and returns an integer number which indicates length of string.

Syntax

length = strlen (StringName);

strcpy()

strcpy() function is used to copy one string into another string.

strcpy() function accepts two string (source and destination) as an argument and copy source string into destination string. It does not return any value.

Syntax:

strcpy(DestinationString, SourceString);

strcat()

strcat() function is used to append (join) one string at the end of another string.

strcat function accepts two strings (string1 and string2) as an argument and append string2 at the end of string1 and result is stored in string1. It does not return any value.

Syntax: strcat (String1, String2);

strcmp()

strcmp() funcion is used to compare one string with another string.

strcmp() function accepts two strings (string1 and string2) as an argument and returns one of the following three integer values:

0 if both string are equal.

>0 if string1 is greater then string2.

<0 if string1 is less then string2.

Syntax:

answer = strcmp (String1, String2)

strrev()

strrev() function is used to reverse given string.

strrev() function accepts a string as an ergument and reverse the string.

Syntax:

strrev (String);

strstr()

strstr() function is used to search one string into another string.

strstr() function accepts two strings (OriginalString and SearchString) as an argument. It returns NULL if search string is not found in Original String.

Syntax:

strstr (OriginalString, SearchString);unctions.

Function

A large C program is divided into basic building blocks called C function. C function contains set of instructions enclosed by "{ }" which performs specific operation in a C program. Actually, Collection of these functions creates a C program.

A function is a block of code that performs a specific task.

3.6. User defined Functions

C allows programmer to define functions according to their need. These functions are known as user-defined functions. For example:

Suppose, a program related to graphics needs to create a circle and color it depending upon the radius and color from the user. You can create two functions to solve this problem:

- createCircle() function
- color() function

Example: User-defined function

Here is a example to add two integers. To perform this task, a user-defined function addNumbers() is defined.

```
#include <stdio.h>
intaddNumbers(int a, int b);          // function prototype
int main()
{
int n1,n2,sum;
printf("Enters two numbers: ");
scanf("%d %d",&n1,&n2);
sum = addNumbers(n1, n2);         // function call
printf("sum = %d",sum);
return 0;
}
intaddNumbers(inta,int b)          // function definition
{
int result;
result = a+b;
return result;                  // return statement
}
```

Function prototype

A function prototype is simply the declaration of a function that specifies function's name, parameters and return type. It doesn't contain function body.

A function prototype gives information to the compiler that the function may later be used in the program.

Syntax of function prototype

returnTypefunctionName(type1 argument1, type2 argument2,...);

In the above example, intaddNumbers(int a, int b); is the function prototype which provides following information to the compiler:

- name of the function is add()
- return type of the function is int
- two arguments of type int are passed to the function

The function prototype is not needed if the user-defined function is defined before the main() function.

3.7 Function Calls

Function call – This calls the actual function

Syntax : function call function_name (arguments list);

USES OF C FUNCTIONS:

C functions are used to avoid rewriting same logic/code again and again in a program.

There is no limit in calling C functions to make use of same functionality wherever required.

We can call functions any number of times in a program and from any place in a program.

A large C program can easily be tracked when it is divided into functions.

The core concept of C functions are, re-usability, dividing a big task into small pieces to achieve the functionality and to improve understandability of very large C programs.

Defining a function.

Generally a function is an independent program that carries out some specific well defined task. It is written after or before the main

function. A function has two components-definition of the function and body of the function.Generally it looks like

datatype function name(list of arguments with type)

{

statements

return;

}

If the function does not return any value to the calling point (where the function is

accessed) .The syntax looks like

function name(list of arguments with type)

{

statements

return;

}

If a value is returned to the calling point, usually the return statement looks like

return(value).In that case data type of the function is executed.

Note that if a function returns no value the keyword void can be used before the function name

Example:

```
writecaption(char x[] );
{
printf("%s",x);
return;
}
int maximum(int x, int y)
{
int z ;
z=(x>=y)? x : y ;
```

```
return(z);
}
maximum(intx,int y)
{
int z;
z=(x>=y) ?x : y ;
printf("\n maximum =%d",z);
return ;
}
```

Advantages of functions

1. It appeared in the main program several times, such that by making it a function, it can be written just once, and the several places where it used to appear can be replaced with calls to the new function.
2. The main program was getting too big, so it could be made (presumably) smaller and more manageable by lopping part of it off and making it a function.
3. It does just one well-defined task, and does it well.
4. Its interface to the rest of the program is clean and narrow
5. Compilation of the program can be made easier.

Accessing a function

A function is accessed in the program (known as calling program)by specifying its name with optional list of arguments enclosed in parenthesis. If arguments are not required then only with empty parenthesis. The arguments should be of the same data type defined in the function definition.

Example:

```
1) inta,b,y;
y=maximum(a,b);
```

2) char name[50] ;
writecaption(name);
3) arrange();

3.8. Function Declaration

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

S.no	C function aspects	syntax
1	function definition	return_typefunction_name(arguments list) { Body of function; }
2	function call	function_name(arguments list);
3	function declaration	return_typefunction_name(argument list);

3.9 Structures and Unions

Structure : A collection of data items of different data types using a single name known as structure. The general format of the structure is as follows

```
Struct tag-name
        {
datatype member1;
datatype  member2
        }
```

Here the keyword struct declares a structure to hold the details of the fields. these field is called structure elements or members. we can declare structure variables using the tag name anywhere in the program .

Ex : struct book-bank

```
{
char title[20];
char author[5];
int pages;
float price;
}
main()
struct book-bank ook1,book2,book3;
                 |
```

The keyword struct declares a structure to hold the details of four fields, namely title ,author, pages, and price. And declares book1, book2, book3 as structure variables of type struct book-bank.

Comparision of structure variables : Two variables of the same structure type can be compared the same way as ordinary variables. If person1 and person2 belong to the same structure ,then the following operations

Arrays of Structures : In analysing the marks obtained by a class of students

,we may use to describe student name and marks obtained in various subjects and then declare all the students as structure variables. In such cases , we may declare an array of sturcture variable.

Ex : struct class student[100];

It defines an array called student , that consists of 100 elements. Each element is defined to be of the type struct class. Consider the following declaration

struct marks

```
{
int subject1;
int subject2;
int subject3
}
main()
{
staticsturct
marks student[3]={{45,68,81},{75,53,69},{57,36,71}}
```

This declares the student as an array of three elements Student[0], student[1], student[2] and initializes their members as follows

```
student[0].subject1=45;
student[0].subject2=68;
                |
                |
student[2].subject3=71;
```

An array of structures is stored inside the memory in the same way as a multi-dimensional array.

Arrays within structures : 'C' permits the use of arrays as structure members. similarly , we can use single or multi-dimensional arrays of type int or float.

Ex : The following structure declaration is valid

```
struct marks
{
int number;
float subject[3]
}student[2];
```

Here,the member subject contains three elements, subject[0], subject[1], subject[2] . These elements can be accessed using appropriate subscripts . For example, the name student[1].subject[2] would refer to the marks obtained in the third subject by the second student.

Structures within structures : Structure within a structure means nesting of structures. Nesting of structures is permitted in 'C'.

Ex :

```
struct salary
{
char name[20];
chardepartament[10];
int basic-pay;
int dearness-allowance;
int house-rent-allowance;
int city-allowance;
} employee;
```

This structure defines name, department , basic pay and three kinds of allowances. we can group all the items related to allowance together and declare them under a substructure as shown below.

```
struct salary
{
char name[20];
char department[10];
struct
{
int dearness;
int house-rent;
int city;
} allowance;
} employee;
```

The salary sturcture contains a member named allowance which itself is a structure with three members.

Ex : struct salary

{

struct

{

int dearness;

|

|

}allowance;

}employee[100];

A base member can be accessed as follows employee[i].allowance.dearness ;

Structure and Functions : 'C' supports the passing of structure values as arguments to functions. The general format of sending a copy of a structure to the called function is

function name(structure variable name)

The called function takes the following form

data-type function name(st-name)

struct-typest-name;

{

return(expression);

}

There are two methods by which the values of a structure can be transferred from one function to another . The first method involves passing of a copy of the entire structure to the called function. Since the function is working on a copy of the structure any changes to structure members within the function are not reflected in the original structure(in the

calling function). It is necessary for the function to return the entire structure back to the calling function.

Ex :

```
struct stores
{
char name[20];
float price;
int quantity;
}
main( )
{
struct stores item = {"xyz",25.75,12}
update(item);
printf("name=%s\n",item.name);
printf("price=%f\n",item.price);
printf("quantity=%d\n",item.quantity);
}
update(product)
struct stores product;
{
product.quantity+ = 10 ;
return;
                    }
```

The function update receives a copy of the structure variable item of its parameter. Both the function update and the formal parameter product are declared as type struct stores. At the time of processing the copy of the actual parameter taken asformal parameter (product structure variable). These two referstwo different memory locations. ie., After the execution of the program the output will be

name = xyz

price = 25.75

quantity = 12

The second method employs a concept called pointers to pass the structure as an argument . In this case, the address location of the structure is passed to the called function. The function can access indirectly the entire structure and work on it. This is similar to the arrays passed to functions. This method is more efficient to the as compared to the first one.

Ex :

```
struct stores
{
char name[20];
float price;
int quantity;
}product[2],*ptr;
main( )
{
struct stores product[0]={"xyz",25.75,12},*ptr;
update(product);
printf("name=%s\n",product[0].name);
printf("price=%f\n",product[0].price);
printf("quantity=%d\n",product[0].quantity);
}
update(product)
struct stores product;
{
ptr->quantity+=10
  }
```

In this statement product is an array of two elements , each of the type structstroes and ptr as a pointer to data objects of the type struct stores. The assignment ptr = product; would assign the address of the zero'th element of product to ptr . i.e.., the pointer will now point to product[0]. Its remember accessed the members as ptr->name . Both the function update and the formal parameter product are declared as type struct stores. At the time of processing the address of the actual parameter taken as formal parameter(product structure variable). These two refers same memory locations. i.e., After the execution of the program the output will be

```
name = xyz
price = 25.75
quantity  = 22
```

3.10 Unions

Unions : Unions are like structure, but the major distinction between them in terms of stroage. In structures, each member has its own storage location, whereas all the members of union use the same location. ie, A union may contain many members of different types, but it can handle only one member at a time.

The general syntax is

```
union item
{
int m;
float x;
```

char c;

}code;

Here a variable code is of type union item. The union contains three members , each with a different data type. However we can use only one of them at a time. The compiler allocates a piece of stroage that is large enough to hold the largest variable type in the union.

To access a union member, we can use for structure members. i.e..,for example

code.m=379;

code.x=7859.36

printf("%d",code.m);

Would produce erroneous output (which is machine dependent) ,because a union is a stroage location that can be used by any one of its members at a time. When a different member is assigned a new value, the new value supersedes the previous members value.

Distinguish between union and structure:

Structure	**Union**
It is a method of packing data of different types	Members of unions use the same location in memory
A structure is a convenient to handle a group of logically only related data items	Union may contain many members different types, it can handle a single data item at a time
Structures help to organize a complex data in more meaningful	The compiler allocates piece of storage that is large enough way
It is a static allocation	It is dynamic allocation

Questions and Answers

1. What is an array?

An array is a group of similar data types stored under a common name. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type.
Example: int a[10];
Here a[10] is an array with 10 values.

2. What are the main elements of an array declaration?

Array name

Type and Size

3. How to initialize an array?

You can initialize array in C either one by one or using a single statement as follows:
double balance[5] = {1000.0, 2.0, 3.4, 17.0, 50.0};
The number of values between braces { } cannot be larger than the number of elements that we declare for the array between square brackets [ ]. Following is an example to assign a single element of the array:

4. Why is it necessary to give the size of an array in an array declaration?

When an array is declared, the compiler allocates a base address and reserves enough space in the memory for all the elements of the array. The size is required to allocate the required space. Thus, the size must be mentioned.

5. What is the difference between an array and pointer?

Array	**Pointer**
Array allocates space automatically	It is explicitly assigned to poin to an allocated space
It cannot be resized	It can ne resized using realloc()
It cannot be reassigned	Pointers can be reassigned
Size of (array name) gives the number of bytes occupied by the array.	Size of (pointer name returns the number of bytes used to store the pointer variable

6. List the characteristics of Arrays.

All elements of an array share the same name, and they are distinguished form one another with help of an element number. Any particular element of an array can be modified separately without disturbing other elements.

7. What are the types of Arrays?

1. One-Dimensional Array
2. Two-Dimensional Array
3. Multi-Dimensional Array

8. Define Strings.

Strings: The group of characters, digit and symbols enclosed within quotes is called as String (or) character Arrays. Strings are always terminated with „\0" (NULL) character. The compiler automatically adds „\0" at the end of the strings.

Example: char name[]={„C","O","L","L","E","G","E","E","\0"};

9. What are functions in C?

A function is a group of statements that together perform a task. Every C program has at least one function which is main(), and all the most trivial programs can define additional functions.

10. How will define a function in C?

Defining a Function:

The general form of a function definition in C programming language is as follows:

return_type function_name(parameter list) { body of the function }

A function definition in C programming language consists of a function header and a function body.

Return Type

Function Name

Parameters

Function Body

11. What are the steps in writing a function in a program?

a) Function Declaration (Prototype declaration): Every user-defined function has to be declared before the main ().

b) Function Callings: The user-defined functions can be called inside any functions like main (), user-defined function, etc.

c) Function Definition:

12. What is the purpose of the function main ()

The function main () invokes other functions within it. It is the first function to be called when the program starts execution.

13. What is meant by Recursive function? If a function calls itself again and again, then that function is called Recursive function.

Example: void recursion() { recursion(); /* function calls itself */ }

int main() { recursion(); }

14. Define Structure in C.

C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.

If you want to access structure members in C, structure variable should be declared.

Many structure variables can be declared for same structure and memory will be allocated for each separately.

It is a best practice to initialize a structure to null while declaring, if we don‟t assign any values to structure members.

15. What you meant by structure definition?

A structure type is usually defined near to the start of a file using a typedef statement. typedef defines and names a new type, allowing its use throughout the program. typedefs usually occur just after the #define and #include statements in a file.

Here is an example structure definition.

```
typedef struct { char  name[64];  char course[128];  int age;  int year;  }
student;
```

This defines a new type student variables of type student can be declared as follows.

```
student st_rec;
```

16. How to Declare a members in Structure?

A struct in C programming language is a structured (record) type that aggregates a fixed set of labeled objects, possibly of different types, into a single object. The syntax for a struct declaration in C is:

```
struct tag_name  {
type attribute; type attribute2;  /* ... */  };
```

17. What is meant by Union in C.?

A union is a special data type available in C that enables you to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multi-purpose.

18. How to define a union in C.

To define a union, you must use the union statement in very similar was as you did while defining structure. The union statement defines a new data type, with more than one member for your program. The format of the union statement is as follows:

union [union tag]

{

member definition; member definition; ... member definition; } [one or more union variables];

19. How can you access the members of the Union?

To access any member of a union, we use the member access operator (.). The member access operator is coded as a period between the union variable name and the union member that we wish to access. You would use union keyword to define variables of union type.

20. To Calculate the Average Using Arrays

```
#include <stdio.h>
int main()
1.  {
2.      int n, i;
3.      float num[100], sum = 0.0, average;
```

```
4.
5.      printf("Enter the numbers of elements: ");
6.      scanf("%d", &n);
7.
8.      while (n > 100 || n <= 0)
9.      {
10.         printf("Error! number should in range of (1 to 100).\n");
11.         printf("Enter the number again: ");
12.         scanf("%d", &n);
13.     }
14.
15.     for(i = 0; i < n; ++i)
16.     {
17.         printf("%d. Enter number: ", i+1);
18.         scanf("%f", &num[i]);
19.         sum += num[i];
20.     }
21.
22.     average = sum / n;
23.     printf("Average = %.2f", average);
24.
25.     return 0;
26. }
```

27. **Output**

```
28. Enter the numbers of elements: 6
29. 1. Enter number: 45.3
30. 2. Enter number: 67.5
31. 3. Enter number: -45.6
32. 4. Enter number: 20.34
```

```
5. Enter number: 33
6. Enter number: 45.6
Average = 27.69
```

This program takes the number of elements in the array and stores in the variable **n**. Then, the for loop gets all the elements from the user and stores the sum of the entered numbers in **sum**. Finally, the average is calculated by dividing **sum** by the number of elements **n**.

21. Explain in detail about Structures

C Structure is a collection of different data types which are grouped together and each element in a C structure is called member.

- If you want to access structure members in C, structure variable should be declared.
- Many structure variables can be declared for same structure and memory will be allocated for each separately.
- It is a best practice to initialize a structure to null while declaring, if we don't assign any values to structure members.

Difference between C variable, C array and C structure:

- A normal C variable can hold only one data of one data type at a time.
- An array can hold group of data of same data type.
- A structure can hold group of data of different data types and Data types can be int, char, float, double and long double etc.

C Structure:

Syntax	struct student
	{
	int a;
	char b[10];
	}

Example	a = 10; b = “Hello”;

C Variable:

int	Syntax: int a; Example: a = 20;
char	Syntax: char b; Example: b=’Z’;

C Array:

int	**Syntax:** int a[3];
	Example:
	a[0] = 10;
	a[1] = 20;
	a[2] = 30;
	a[3] = ‘\0’;

char	**Syntax:** char b[10]; **Example:** b=”Hello”;

UNIT - IV

4.1 Pointers

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

Syntax : type *var-name;

Here, 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.

Understanding Pointers

There are a few important operations, which we will do with the help of pointers very frequently. (a) We define a pointer variable

(b) assign the address of a variable to a pointer and

(c) finally access the value at the address available in the pointer variable.

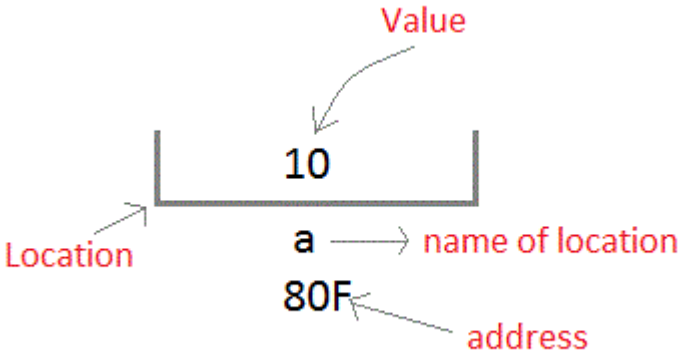
This is done by using unary operator * that returns the value of the variable located at the address specified by its operand

Declaring Pointer Variable

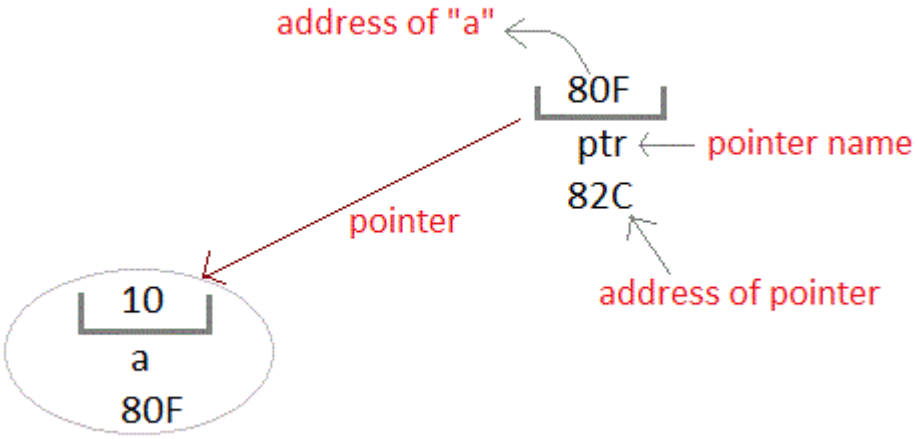
Whenever a variable is declared, system will allocate a location to that variable in the memory, to hold value. This location will have its own address number.

Let us assume that system has allocated memory location 80F for a variable a.

int a = 10 ;



We can access the value 10 by either using the variable name a or the address 80F. Since the memory addresses are simply numbers they can be assigned to some other variable. The variable that holds memory address are called pointer variables. A pointer variable is therefore nothing but a variable that contains an address, which is a location of another variable. Value of pointer variable will be stored in another memory location.



General syntax of pointer declaration is,

data-type *pointer_name;

Data type of pointer must be same as the variable, which the pointer is pointing. void type pointer works with all data types, but isn't used oftenly.

Accessing a variable through its pointer

There are few important operations, which we will do with the help of pointers very frequently.

(a) we define a pointer variable

(b) Assign the address of a variable to a pointer and

(c) Finally access the value at the address available in the pointer variable. This is done by using unary operator * that returns the value of the variable located at the address specified by its operand.The indirection operator (*) is used to access the value of a variable by its ptr

* can be remembered as value at address

int n = *p // int *p = &quantity is done

int n = *&quantity // is = quantity

*5445 where 5445 is a valid location does not yield the content at that address

Syntax:

*pointer variable;

where

* - indirection operator

pointer variable - declared pointer variable

Program:

```
#include<stdio.h>
main()
{
int a;
int *b;
a=100;
b=&a;
printf("The content of the pointer b=%d\n", *b);
}
```

Output:

The content of the pointer b=100

Pointer Expressions

Pointer expression is a linear combination of pointer variables, variables and operators (+, -, ++, __). The pointer expression gives either numerical output or address output.

Pointer Assignment:

The general form of pointer assignment is

variable = pointer expression

Example:

```
int x=5,y;
int *p, *q;
p = &x;
q = &y;
*q = *p +10 => pointer assignment.
```

Example:

```
y = *p1 * *p2;
sum = sum + *p1;
z = 5* - *p2/p1;
*p2 = *p2 + 10;
```

C language allows us to add integers to, subtract integers from pointers as well as to subtract one pointer from the other. We can also use short hand operators with the pointers p1+=; sum+=*p2; etc., we can also compare pointers by using relational operators the expressions such as p1 >p2 , p1==p2 and p1!=p2 are allowed.

4.2 File Management in C

A file represents a sequence of bytes on the disk where a group of related data is stored. File is created for permanent storage of data. It is a readymade structure.

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

Syntax : FILE *fp;

Defining and Opening a File

C provides a number of functions that helps to perform basic file operations. Following are the functions,

Function	Description
fopen()	create a new file or open a existing file
fclose()	closes a file
getc()	reads a character from a file
putc()	writes a character to a file
fscanf()	reads a set of data from a file
fprintf()	writes a set of data to a file
getw()	reads a integer from a file
putw()	writes a integer to a file
fseek()	set the position to desire point
ftell()	gives current position in the file
rewind()	set the position to the begining point

Opening a File or Creating a File

The fopen() function is used to create a new file or to open an existing file.

General Syntax :

*fp = FILE *fopen(const char *filename, const char *mode);

Here filename is the name of the file to be opened and mode specifies the purpose of opening the file. Mode can be of following types,
*fp is the FILE pointer (FILE *fp), which will hold the reference to the opened(or created) file.

Mode	Description
r	opens a text file in reading mode
w	opens or create a text file in writing mode.
a	opens a text file in append mode
r+	opens a text file in both reading and writing mode
w+	opens a text file in both reading and writing mode
a+	opens a text file in both reading and writing mode
rb	opens a binary file in reading mode
wb	opens or create a binary file in writing mode
ab	opens a binary file in append mode
rb+	opens a binary file in both reading and writing mode
wb+	opens a binary file in both reading and writing mode
ab+	opens a binary file in both reading and writing mode

Closing a File

The fclose() function is used to close an already opened file.

General Syntax :

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

Here fclose() function closes the file and returns zero on success, or EOF if there is an error in closing the file. This EOF is a constant defined in the header file stdio.h.

Input / Output operation on Files

In the above table we have discussed about various file I/O functions to perform reading and writing on file. getc() and putc() are simplest functions used to read and write individual characters to a file.

Example

```
#include<stdio.h>
#include<conio.h>
main()
{
 FILE *fp;
charch;
fp = fopen("one.txt", "w");
printf("Enter data");
while( (ch = getchar()) != EOF) {
putc(ch,fp);
 }
fclose(fp);
fp = fopen("one.txt", "r");
while( (ch = getc(fp)! = EOF)
printf("%c",ch);
fclose(fp);
}
```

Random Access to Files

Random access means we can move to any part of a file and read or write data from it without having to read through the entire file. Back thirty years ago, much data was stored on large reels of computer tape. The only way to get to a point on the tape was by reading all the way through the

tape. Then disks came along and now we can read any part of a file directly.we can access the data stored in the file in two ways.

i. Sequentially
ii. Randomly

i. Sequentially

If we want to access the forty fourth record then first forty three records read sequentially to reach forty four records.

ii. Randomly

In random access data can be accessed and processed directly. There is no need to read each record sequentially .if we want to access a particular record random access takes less time than the sequential access. C supports these functions for random access file.

i. fseek( ) Function
ii. ftell ( ) Function

i. fseek() function in C:

This function is used for setting the file position pointer at the specified bytes. fseek is a function belonging to the ANCI C standard Library and included in the file stdio.h. Its purpose is to change the file position indicator for the specified stream.

Syntax :intfseek(FILE*stream_pointer, longoffset, intorigin);
Argument meaning:
stream_pointer is a pointer to the stream FILEstructure of which the position indicator should be changed;
offset is a long integer which specifies the number of bytes from origin where the position indicator should be placed;
origin is an integer which specifies the origin position. It can be:

SEEK_SET: origin is the start of the stream

SEEK_CUR: origin is the current position

SEEK_END: origin is the end of the stream

ii. ftell() Function in C:

This function return the current position of the file position pointer. The value is counted from the beginning of the file.

Syntax : long ftell (file * fptr);

Questions and Answers

1. What is a Pointer? How a variable is declared to the pointer?

Pointer is a variable which holds the address of another variable.

Pointer Declaration

datatype *variable-name;

Example

int *x, c=5;

x=&a;

2. What are the uses of Pointers?

Pointers are used to return more than one value to the function

Pointers are more efficient in handling the data in arrays

Pointers reduce the length and complexity of the program

They increase the execution speed

The pointers save data storage space in memory

3. What are * and & operators means?

'*' operator means 'value at the address'

'&' operator means 'address of'

4. What is meant by Preprocessor?

Preprocessor is the program, that process our source program before the compilation.

5. How can you return more than one value from a function?

A Function returns only one value. By using pointer we can return more than one value.

6. List the header files in 'C' language.

<stdio.h> contains standard I/O functions

<ctype.h> contains character handling functions

<stdlib.h> contains general utility functions

<string.h> contains string manipulation functions

<math.h> contains mathematical functions

<time.h> contains time manipulation functions

7. What is dangling pointer?

In C, a pointer may be used to hold the address of dynamically allocated memory.

After this memory is freed with the free() function, the pointer itself will still contain the address of the released block. This is referred to as a dangling pointer. Using the pointer in this state is a serious programming error. Pointer should be assigned NULL after freeing memory to avoid this bug.

8. What is file?

File is a collection of bytes that is stored on secondary storage devices like disk. There are two kinds of files in a system. They are,

- Text files (ASCII)- Text files contain ASCII codes of digits, alphabetic and symbols
- Binary files- Binary file contains collection of bytes (0's and 1's). Binary files are compiled version of text files.

9. Concatenation of two strings using pointer in c programming language

```
#include<stdio.h>
int main(){
  int i=0,j=0;
  char *str1,*str2,*str3;
```

```
  puts("Enter first string");
  gets(str1);
  puts("Enter second string");
  gets(str2);
  printf("Before concatenation the strings are\n");
  puts(str1);
  puts(str2);
  while(*str1){
      str3[i++]=*str1++;
  }
  while(*str2){
      str3[i++]=*str2++;
  }
  str3[i]='\0';
  printf("After concatenation the strings are\n");
  puts(str3);
  return 0;
}
```

10. Write a C program To Find Largest Element Using Dynamic Memory Allocation - Calloc()

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

int main()
{
    int i, num;
    float *data;

    printf("Enter total number of elements(1 to 100): ");
    scanf("%d", &num);
```

```
    // Allocates the memory for 'num' elements.
    data = (float*) calloc(num, sizeof(float));

    if(data == NULL)
    {
        printf("Error!!! memory not allocated.");
        exit(0);
    }

    printf("\n");

    // Stores the number entered by the user.
    for(i = 0; i < num; ++i)
    {
       printf("Enter Number %d: ", i + 1);
       scanf("%f", data + i);
    }

    // Loop to store largest number at address data
    for(i = 1; i < num; ++i)
    {
       // Change < to > if you want to find the smallest number
       if(*data < *(data + i))
           *data = *(data + i);
    }
    printf("Largest element = %.2f", *data);
    return 0;
}
```

Output

Enter total number of Elements (1 to 100): 10

Enter Number 1: 2.34

Enter Number 2: 3.43

Enter Number 3: 6.78

Enter Number 4: 2.45

Enter Number 5: 7.64

Enter Number 6: 9.05

Enter Number 7: -3.45

Enter Number 8: -9.99

Enter Number 9: 5.67

Enter Number 10: 34.95

Largest element: 34.95

11. Write the Difference between static memory allocation and dynamic memory allocation in C

Static memory allocation	**Dynamic memory allocation**
In static memory allocation, memory is allocated while writing the C program. Actually, user requested memory will be allocated at compile time.	In dynamic memory allocation, memory is allocated while executing the program. That means at run time.
Memory size can't be modified while execution. Example: array	Memory size can be modified while execution. Example: Linked list

12. Write the Difference between malloc() and calloc() functions in C

malloc()	**calloc()**
It allocates only single block of requested memory	It allocates multiple blocks of requested memory
int *ptr;ptr = malloc(20 * sizeof(int));For the above, 20*4 bytes of memory only allocated in one block. Total = 80 bytes	int *ptr;Ptr = calloc(20, 20 * sizeof(int));For the above, 20 blocks of memory will be created and each contains 20*4 bytes of memory. Total = 1600 bytes
malloc () doesn't initializes the allocated memory. It contains garbage values	calloc () initializes the allocated memory to zero
type cast must be done since this function returns void pointer int *ptr;ptr = (int*)malloc(sizeof(int)*20);	Same as malloc () function int *ptr;ptr = (int*)calloc(20, 20 * sizeof(int));

13. Explain in detail about Dynamic memory allocation in C

The process of allocating memory during program execution is called dynamic memory allocation. *Dynamic memory allocation functions in C,*

C language offers 4 dynamic memory allocation functions. They are,

i. malloc()
ii. calloc()
iii. iii. realloc()
iv. iv. free()

Function	Syntax
malloc ()	malloc (number *sizeof(int));
calloc ()	calloc (number, sizeof(int));
realloc ()	realloc (pointer_name, number * sizeof(int));
free ()	free (pointer_name);

i. malloc() function in C:

- malloc () function is used to allocate space in memory during the execution of the program.
- malloc () does not initialize the memory allocated during execution. It carries garbage value.
- malloc () function returns null pointer if it couldn't able to allocate requested amount of memory.

ii. calloc() function in C:

- calloc () function is also like malloc () function. But calloc () initializes the allocated memory to zero. But, malloc() doesn't.

iii. realloc() function in C:

- realloc () function modifies the allocated memory size by malloc () and calloc () functions to new size.
- If enough space doesn't exist in memory of current block to extend, new block is allocated for the full size of reallocation, then copies the existing data to new block and then frees the old block.

iv. free() function in C:

- free () function frees the allocated memory by malloc (), calloc (), realloc () functions and returns the memory to the system.

14. Explain in Inbuilt file handling functions in C language C programming language offers many inbuilt functions for handling files. They are given below.

File handling functions	Description
fopen ()	fopen () function creates a new file or opens an existing file.
fclose ()	fclose () function closes an opened file.
getw ()	getw () function reads an integer from file.
putw ()	putw () functions writes an integer to file.
getc (), fgetc ()	getc () and fgetc () functions read a character from file.
putc (), fputc ()	putc () and fputc () functions write a character to file.
gets ()	gets () function reads line from keyboard.
puts ()	puts () function writes line to o/p screen.
fgets ()	fgets () function reads string from a file, one line at a time.
fputs ()	fputs () function writes string to a file.
feof ()	feof () function finds end of file.
fgetchar ()	fgetchar () function reads a character from keyboard.
fgetc ()	fgetc () function reads a character from file.
fprintf ()	fprintf () function writes formatted data to a file.
fscanf ()	fscanf () function reads formatted data from a file.
fgetchar ()	fgetchar () function reads a character from keyboard.
fputchar ()	fputchar () function writes a character from keyboard.
fseek ()	fseek () function moves file pointer position to given location.

SEEK_SET	SEEK_SET moves file pointer position to the beginning of the file.
SEEK_CUR	SEEK_CUR moves file pointer position to given location.
SEEK_END	SEEK_END moves file pointer position to the end of file.
ftell ()	ftell () function gives current position of file pointer.
rewind ()	rewind () function moves file pointer position to the beginning of the file.
getc ()	getc () function reads character from file.
getch ()	getch () function reads character from keyboard.
getche ()	It reads character from keyboard and echoes to o/p screen.
getchar ()	getchar () function reads character from keyboard.
putc ()	putc () function writes a character to file.
putchar ()	putchar () function writes a character to screen.
printf ()	printf () function writes formatted data to screen.
sprinf ()	sprinf () function writes formatted output to string.
scanf ()	scanf () function reads formatted data from keyboard.
sscanf ()	sscanf () function Reads formatted input from a string.
remove ()	remove () function deletes a file.
fflush ()	fflush () function flushes a file.

15. Write down Basic file operations in C programming

There are 4 basic operations that can be performed on any files in C programming language. They are,

1. Opening a file
2. Closing a file
3. Reading a file
4. Writing in a file

Let us see the syntax for each of the above operations in a table

File operation/ Syntax	**Description**
file open FILE *fp; fp=fopen (“filename”, ”‘mode”);	fopen function is used to open a file. Where, fp is file pointer to the data type “FILE”.
file close: fclose(fp);	fclose function closes the file that is being pointed by file pointer fp.
file read: fgets (buffer, size, fp);	fgets is used to read a file line by line. where, buffer – buffer to put the data in. size – size of the buffer fp – file pointer
file write: fprintf (fp, “some data”); fprintf (fp, “text %d”, variable_name);	fprintf writes the data into a file pointed by fp.

UNIT - V

5.1 Introduction

R is a programming language and software environment for statistical analysis on data.

- R is used for graphics representation and reporting.
- R is registered under GNU (General Public License).
- R is an open source software package to use by data scientist statisticians and others who need to make glean key insights from data using mechanisms, such as regression, clustering, classification, and text analysis.
- R provides a wide variety of statistical, machine learning (linear and nonlinear modeling, classic statistical tests, time-series analysis, classification, clustering) and graphical techniques, and is highly extensible.
- R has various built-in as well as extended functions for statistical, machine learning, and visualization tasks such as:
- Data extraction
- Data cleaning
- Data loading
- Data transformation
- Statistical analysis
- Predictive modeling
- Data visualization

5.2 History of R programming

R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, which is currently developed by the R Development Core Team. R made its first appearance in 1993. This programming language was named R, based on the first letter of first name of the two R authors (Robert Gentleman and Ross Ihaka), and partly a play on the name of the Bell Labs Language S.

- A large group of individuals has contributed to R by sending code and bug reports.
- Since mid-1997 there has been a core group (the "R Core Team") who can modify the R source code archive.

5.3 R Commands

help()	Obtain documentation for a given R command
c(), scan()	Enter data manually to a vector in R
seq()	Make arithmetic progression vector
rep()	Make vector of repeated values
data()	Load (often into a data.frame) built-in dataset
View()	View dataset in a spreadsheet-type format
str()	Display internal structure of an R object read.csv(),
read.table()	Load into a data.frame an existing data file
library(), require()	Make available an R add-on package
dim()	See dimensions (# of rows/cols) of data.frame
length()	Give length of a vector
ls()	Lists memory contents
rm()	Removes an item from memory
names()	Lists names of variables in a data.frame
hist()	Command for producing a histogram

histogram()	Lattice command for producing a histogram
stem()	Make a stem plot
table()	List all values of a variable with frequencies
xtabs()	Cross-tabulation tables using formulas
mosaicplot()	Make a mosaic plot
cut()	Groups values of a variable into larger bins
mean(), median()	Identify “center” of distribution
by()	apply function to a column split by factors
summary()	Display 5-number summary and mean
var(), sd()	Find variance, sd of values in vector
sum()	Add up all values in a vector
quantile()	Find the position of a quantile in a dataset
plot()	Produces a scatterplot
barplot()	Produces a bar graph
barchart()	Lattice command for producing bar graphs
boxplot()	Produces a boxplot
bwplot()	Lattice command for producing boxplots
xyplot()	Lattice command for producing a scatterplot
lm()	Determine the least-squares regression line
anova()	Analysis of variance (can use on results of
predict()	Obtain predicted values from linear model
nls()	estimate parameters of a nonlinear model
residuals()	gives (observed - predicted) for a model fit to data
sample()	take a sample from a vector of data
replicate()	repeat some process a set number of times
cumsum()	produce running total of values for input vector
ecdf()	builds empirical cumulative distribution function
dbinom(), etc.	tools for binomial distributions
dpois(), etc.	tools for Poisson distributions
pnorm(), etc.	tools for normal distributions
qt(), etc.	tools for student t distributions

pchisq(), etc.	tools for chi-square distributions
binom.test()	hypothesis test and confidence interval for 1 proportion
prop.test()	inference for 1 proportion using normal approx.
chisq.test()	carries out a chi-square test
fisher.test()	Fisher test for contingency table
t.test()	t test for inference on population mean
qqnorm(), qqline()	tools for checking normality
addmargins()	adds marginal sums to an existing table
prop.table()	compute proportions from a contingency table
par()	query and edit graphical settings
power.t.test()	power calculations for 1- and 2-sample t
anova()	compute analysis of variance table for fitted model

5.4 Random Numbers Generation

A sequence of random numbers R1, R2, …, must have two important statistical properties:

- Uniformity
- Independence.

Random Number, Ri, must be independently drawn from a uniform distribution

As we know, random numbers are described by a distribution. That is, some function which specifies the probability that a random number is in some range.

For example P(a < X ≤ b). Often this is given by a probability density (in the continuous case) or by a function P(X=k) = f(k) in the discrete case. R will give numbers drawn from lots of different distributions. In order to use them, you only need familiarize yourselves with the parameters that are given to the functions such as a mean, or a rate. Here are examples of the most common ones. For each, a histogram is given for a random sample of size 100, and density (using the ``d'' functions) is superimposed as appropriate.

Uniform

Uniform numbers are ones that are "equally likely" to be in the specified range. Often these numbers are in [0,1] for computers, but in practice can be between [a,b] where a,b depend upon the problem. An example might be the time you wait at a traffic light. This might be uniform on [0,2].

```
>runif(1,0,2)                # time at light
[1] 1.490857                   # also runif(1,min=0,max=2)
>runif(5,0,2)                # time at 5 lights
[1] 0.07076444 0.01870595 0.50100158 0.61309213 0.77972391
>runif(5)                    # 5 random numbers in [0,1]
[1] 0.1705696 0.8001335 0.9218580 0.1200221 0.1836119
```

The general form is runif(n,min=0,max=1) which allows you to decide how many uniform random numbers you want (n), and the range they are chosen from ([min,max])

To see the distribution with min=0 and max=1 (the default) we have

```
> x=runif(100)                 # get the random numbers
>hist(x,probability=TRUE,col=gray(.9),main="uniform on [0,1]")
>curve(dunif(x,0,1),add=T)
```

Normal

Normal numbers are the backbone of classical statistical theory due to the central limit theorem The normal distribution has two parameters a mean μ and a standard deviation s. These are the location and spread parameters. For example, IQs may be normally distributed with mean 100 and standard deviation 16, Human gestation may be normal with mean 280 and standard deviation about 10 (approximately). The family of normals can be standardized to normal with mean 0 (centered) and variance 1. This is achieved by "standardizing" the numbers, i.e. $Z=(X-\mu)/s$.

Here are some examples

```
>rnorm(1,100,16)              # an IQ score
[1] 94.1719
>rnorm(1,mean=280,sd=10)
[1] 270.4325                  # how long for a baby (10 days early)
```

Here the function is called as rnorm(n,mean=0,sd=1) where one specifies the mean and the standard deviation.

To see the shape for the defaults (mean 0, standard deviation 1) we have (figure 26)

```
> x=rnorm(100)
>hist(x,probability=TRUE,col=gray(.9),main="normal mu=0,sigma=1")
>curve(dnorm(x),add=T)
## also for IQs using rnorm(100,mean=100,sd=16)
```

Binomial

The binomial random numbers are discrete random numbers. They have the distribution of the number of successes in n independent Bernoulli trials where a Bernoulli trial results in success or failure, success with probability p.

A single Bernoulli trial is given with n=1 in the binomial

```
> n=1, p=.5                  # set the probability
>rbinom(1,n,p)               # different each time
[1] 1
>rbinom(10,n,p)              # 10 different such numbers
 [1] 0 1 1 0 1 0 1 0 1 0
```

A binomially distributed number is the same as the number of 1's in n such Bernoulli numbers. For the last example, this would be 5. There are then two parameters n (the number of Bernoulli trials) and p (the success probability).

To generate binomial numbers, we simply change the value of n from 1 to the desired number of trials. For example, with 10 trials:

```
> n = 10; p=.5
>rbinom(1,n,p)               # 6 successes in 10 trials
[1] 6
>rbinom(5,n,p)               # 5 binomial number
[1] 6 6 4 5 4
```

The number of successes is of course discrete, but as n gets large, the number starts to look quite normal. This is a case of the central limit theorem which states in general that (X-- -)/s is

normal in the limit (note this is standardized as above) and in our specific case that

The graphs show 100 binomially distributed random numbers for 3 values of n and for p=.25. Notice in the graph, as n increases the shape becomes more and more bell-shaped. These graphs were made with the commands

```
> n=5;p=.25                    # change as appropriate
> x=rbinom(100,n,p)            # 100 random numbers
>hist(x,probability=TRUE,)
## use points, not curve as dbinom wants integers only for x
>xvals=0:n;points(xvals,dbinom(xvals,n,p),type="h",lwd=3)
>points(xvals,dbinom(xvals,n,p),type="p",lwd=3)
...    repeat with n=15, n=50
```

Exponential

The exponential distribution is important for theoretical work. It is used to describe lifetimes of electrical components (to first order). For example, if the mean life of a light bulb is 2500 hours one may think its lifetime is random with exponential distribution having mean 2500. The one parameter is the rate = 1/mean. We specify it as follows rexp(n,rate=1). Here is an example with the rate being 1/2500.

```
> x=rexp(100,1/2500)
>hist(x,probability=TRUE,col=gray(.9),main="exponential
mean=2500")
>curve(dexp(x,1/2500),add=T)
```

5.5 Data Types

In contrast to other programming languages like C and java in R, the variables are not declared as some data type. The variables are assigned with R-Objects and the data type of the R-object becomes the data type of the variable. There are many types of R-objects. The frequently used ones are

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frames

Vectors

When you want to create vector with more than one element, you should use c() function which means to combine the elements into a vector.

```
# Create a vector.
apple<- c('red','green',"yellow")
print(apple)

# Get the class of the vector.
print(class(apple))
```

Lists

A list is an R-object which can contain many different types of elements inside it like vectors, functions and even another list inside it.

```
# Create a list.
list1 <- list(c(2,5,3),21.3,sin)
# Print the list.
print(list1)
```

Matrices

A matrix is a two-dimensional rectangular data set. It can be created using a vector input to the matrix function.

```
# Create a matrix.
M = matrix( c('a','a','b','c','b','a'), nrow = 2, ncol = 3, byrow = TRUE)
print(M)
```

Arrays

While matrices are confined to two dimensions, arrays can be of any number of dimensions. The array function takes a dim attribute which creates the required number of dimension. In the below example we create an array with two elements which are 3x3 matrices each.

```
# Create an array.
a <- array(c('green','yellow'),dim = c(3,3,2))
print(a)
```

Factors

Factors are the r-objects which are created using a vector. It stores the vector along with the distinct values of the elements in the vector as labels. The labels are always character irrespective of whether it is numeric or character or Boolean etc. in the input vector. They are useful in statistical modeling.

Factors are created using the factor() function.Thenlevels functions gives the count of levels.

```
# Create a vector.
apple_colors<- c('green','green','yellow','red','red','red','green')
# Create a factor object.
factor_apple<- factor(apple_colors)
# Print the factor.
print(factor_apple)
print(nlevels(factor_apple))
```

Data Frames:

Data frames are tabular data objects. Unlike a matrix in data frame each column can contain different modes of data. The first column can be numeric while the second column can be character and third column can be logical. It is a list of vectors of equal length.

Data Frames are created using the data.frame() function.

```
# Create the data frame.
BMI <-        data.frame(
gender = c("Male", "Male","Female"),
height = c(152, 171.5, 165),
weight = c(81,93, 78),
   Age = c(42,38,26)
)
print(BMI)
```

5.6 Objects

Objects are assigned values using <- , an arrow formed out of < and -. (An equal sign, =, can also be used.) For example, the following command assigns the value 5 to the object x.

```
x <- 5
```

After this assignment, the object x 'contains' the value 5. Another assignment to the same object will change the content.

```
x <- 107
```

we can check the content of an object by simply entering the name of the object on an interactive command line. Try that throughout these examples to see what the results are of the different operations and functions illustrated.

5.7 Basic data and Computations

R is case sensitive programming, it treats data as completely different objects. Statistics is the study of data. After learning how to start R, the first thing we need to be able to do is learn how to enter data into R and how to manipulate the data once there.

Example

help() #give help regarding a command, e.g. help(hist)

c() #concatenate objects,e.g.x = c(3,5,8,9)ory= c(”Jack”,”Queen”,”King”)

1:19 #create a sequence of integers from 1 to 19

(…) #give arguments to a function, e.g. sum(x), or help(hist)

[…] #select elements from a vector or list, e.g. x[2] gives 5, x[c(2,4)] gives 5 9 for x as above

matrix() #fill in (by row) the values from y in a matrix of 4 rows and 3 columns by giving #m = matrix(y,4,3,byrow=T)

dim() #gives the number of rows and the number of columns of a matrix, or a data frame

head() #gives the first 6 rows of a large matrix, or data frame

tail() #gives the last 6 rows of a large matrix, or data frame

m[,3] #gives the 3rd column of the matrix m

m[2,] #gives the 2nd row of the matrix m

= or <- #assign something to a variable, e.g. x = c(”a”,”b”,”b”,”e”)

== #ask whether two things are equal, e.g. x = c(3,5,6,3) and then x == 3

< #ask whether x is smaller than y,

> #ask whether x is larger than y

& #logical „and“

| #logical „or“

sum() #get the sum of the values in x by sum(x)

mean() #get the mean of the values in x by mean(x)

median() #get the median of the values in x by median(x)

```
sd()          #get the standard deviation of the values in x
var()         #get the variance of the values in x
IQR()         #get the IQR of the values in x
summary()     #get the summary statistics of a single variable, or of all
               variables in a data frame
round()       #round values in x to 3 decimal places by round(x,3)
sort()        #sort the values in x by giving sort(x)
unique()      #get the non-duplicate values from a list,
               e.g. x = c(3,5,7,2,3,5,9,3) and then
unique(x)     #gives 3 5 7 2 9
length(x)     #gives the length of the vector x, which is 8
hist()        #create a histogram of the values in x by hist(x)
stem()        #create a stem and leaf plot of the values in x by stem(x)
boxplot()     #create a boxplot of the values in x by boxplot(x)
plot()        #scatterplot of x vs. y by plot(x,y); for more parameters see
               help(plot.default)
cor()         #gives the linear correlation coefficient
lm()          #fit a least squares regression of y (response) on x (predictor) by
              fit = lm(y~x)
names()       #get or set the names of elements in a R object. E.g. names(fit)
               will give the names of the R #object named "fit", or #get or set
               the names of variables in a data frame.
fit$coef      #gives the least squares coefficients from the fit above,
               i.e. intercept and slope
fit$fitted          #gives the fitted values for the regression fitted above
fit$residuals       #gives the residuals for the regression fitted above
lines()             #add a (regression) line to a plot by lines(x,fit$fitted)
abline()            #add a straight line to a scatterplot
points()          #add additional points (different plotting character) to a plot
scan()              #read data for one variable from a text file,
                     e.g. y = scan("ping.dat")
```

read.table() #read spreadsheet data (i.e. more than one variable) from a text file

table() #frequency counts of entries, ideally the entries are factors

write() #write the values of a variable y in a file data.txt by write(y,file="data.txt")

log() #natural logarithm (i.e. base e)

log10() #logarithm to base 10

seq() #create a sequence of integers from 2 to 11 by increment 3 with seq(2,11,by=3)

rep() #repeat n times the value x, e.g. rep(2,5) gives 2 2 2 2 2

getwd() #get the current working directory.

setwd() #change the directory to. E.g. setwd("c:/RESEARCH/GENE.project/Chunks/")

dir() #list files in the current working directory

search() #searching through reachable datasets and packages

library() #link to a downloaded R package to the current R session. E.g. library(Biostrings) link to the #R package #called "Biostrings" which you had downloaded earlier onto your laptop

Input and Display

load("c:/RData/pennstate1.RData") #load a R data frame

read.csv(filename="c:/stat251/ui.csv",header=T) #read .csv file with labels in first row

x=c(1,2,4,8,16) #create a data vector with specified elements

y=c(1:10) #create a data vector with elements 1-10

vect=c(x,y) #combine them into one vector of length 2n

mat=cbind(x,y) #combine them into a n x 2 matrix

mat[4,2] #display the 4th row and the 2nd column

mat[3,] #display the 3rd row

mat[,2] #display the 2nd column

Data Manipulation Examples

```
x.df=data.frame(x1,x2,x3 …)
                              #combine different kinds of data into a data frame

scale()                        #converts a data frame to standardized scores
round(x,n)                     #rounds the values of x to n decimal places
ceiling(x)                     #vector x of smallest integers > x
floor(x)                       #vector x of largest integer< x
as.integer(x)                  #truncates real x to integers (compare to round(x,0)

as.integer(x <cutpoint)
                #vector x of 0 if less than cutpoint, 1 if greater than cutpoint)

factor(ifelse(a <cutpoint, "Neg", "Pos"))
        #is another way to dichotomize and to make a factor for analysis

transform(data.df,variable names = some operation)
                #can be part of a set up for a data set
```

Statistical Tests

```
binom.test()
prop.test()          #perform test with proportion(s)
t.test()                     #perform t test
chisq.test()         #perform Chi-square test
pairwise.t.test()
power.anova.test()
power.t.test()
aov()
anova()
TukeyHSD()
kruskal.test()
```

Distributions

sample(x, size, replace = FALSE, prob = NULL) # take a simple random sample of size n from the

population x with or without replacement

rbinom(n,size,p)

pbinom()

qbinom()

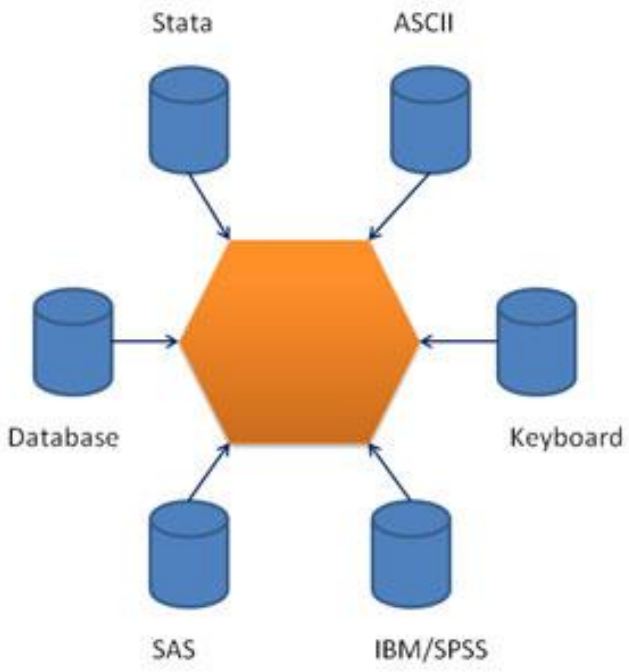
dbinom()

rnorm(n,mean,sd) #randomly generate n numbers from a Normal distribution with the specific mean and sd

pnorm() #find probability (area under curve) of a Normal(10,3^2) distribution to the left

qnorm() #find quantity or value x such that area under Normal(10,3^2)

5.8 Data Input



Unlike SAS, which has DATA and PROC steps, R has data structures (vectors, matrices, arrays, data frames) that you can operate on through

functions that perform statistical analyses and create graphs. This section describes how to enter or import data into R, and how to prepare it for use in statistical analyses. Topics include R data structures, importing data (from Excel, SPSS, SAS, Stata, and ASCII Text Files), entering data from the keyboard, creating an interface with a database management system, exporting data (to Excel, SPSS, SAS, Stata, and Tab Delimited Text Files), annotating data (with variable labels and value labels), and listing data. In addition, methods for handling missing values and date values are presented.

5.9 Data Frames

A data frame is used for storing data tables. It is a list of vectors of equal length. For example, the following variable df is a data frame containing three vectors n, s, b.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
>df = data.frame(n, s, b)       # df is a data frame
```

Build-in Data Frame

We use built-in data frames in R for our tutorials. For example, here is a built-in data frame in R, called mtcars.

```
>mtcars
mpgcyldisphp drat   wt ...
Mazda RX4     21.0   6  160 110 3.90 2.62 ...
Mazda RX4 Wag 21.0   6  160 110 3.90 2.88 ...
Datsun 710    22.8   4  108  93 3.85 2.32 ...
```

The top line of the table, called the header, contains the column names. Each horizontal line afterward denotes a data row, which begins with the name of the row, and then followed by the actual data. Each data member of a row is called a cell.

To retrieve data in a cell, we would enter its row and column coordinates in the single square bracket "[]" operator. The two coordinates are separated by a comma. In other words, the coordinates begins with row position, then followed by a comma, and ends with the column position. The order is important.

Here is the cell value from the first row, second column of mtcars.

```
>mtcars[1, 2]
[1] 6
```

Moreover, we can use the row and column names instead of the numeric coordinates.

```
>mtcars["Mazda RX4", "cyl"]
[1] 6
```

Lastly, the number of data rows in the data frame is given by the nrow function.

```
>nrow(mtcars)    # number of data rows
[1] 32
```

And the number of columns of a data frame is given by the ncol function.

```
>ncol(mtcars)    # number of columns
[1] 11
```

Further details of the mtcars data set is available in the R documentation.

```
>help(mtcars)
```

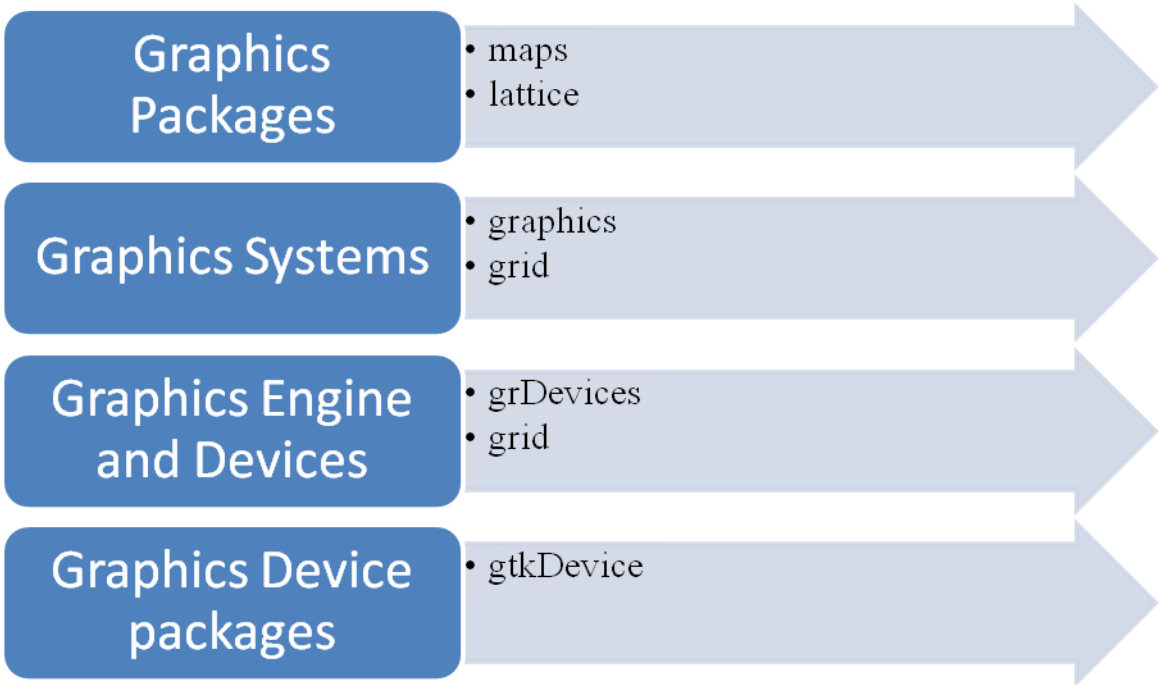
Instead of printing out the entire data frame, it is often desirable to preview it with the head function beforehand.

```
>head(mtcars)
mpgcyldisphp drat   wt ...
Mazda RX4     21.0   6  160 110 3.90 2.62 ...
```

5.10 Graphics

This provides the most basic information to get started producing plots in R. This section provides an introduction to R graphics by way of a series of charts, graphs and visualization. R has also been used to produce figures that help to visualize important concepts or teaching points.

The organization of R graphics this section briefly describes how R's graphics functions are organized so that the user knows where to start looking for a particular function. The R graphics system can be broken into four distinct levels: graphics packages; graphics systems; a graphics engine, including standard graphics devices; and graphics device packages



To visualize data:

- ggplot2 - R's famous package for making beautiful graphics.ggplot2 lets you use thegrammar of graphics to build layered, customizable plots.
- ggvis - Interactive, web based graphics built with the grammar of graphics.
- rgl - Interactive 3D visualizations with R
- Colors : The package colorspace provides a set of functions for transforming between color spaces and mixcolor() for mixing colors within a color space.
- htmlwidgets - A fast way to build interactive (javascript based) visualizations with R. Packages that implement htmlwidgets include:
- leaflet (maps)
- dygraphs (time series)
- DT (tables)
- diagrammeR (diagrams)
- network3D (network graphs)
- threeJS (3D scatterplots and globes).

Graphics formats that R supports and the functions that open an appropriate R Programming language has numerous libraries to create charts and graphs.R provides the usual range of standard statistical plots, including scatterplots, boxplots, and histograms, bar plots, pie charts, and basic3Dplots

Types of charts

- scatterplots,
- boxplots
- histograms
- bar plots
- pie charts
- basic3Dplots

5.11 Table

A table is an arrangement of information in rows and columns that make comparing and contrasting information easier. As you can see in the following example, the data are much easier to read than they would be in a list containing thread.table() #read spreadsheet data (i.e. more than one variable) from a text file table() #frequency counts of entries, ideally the entries are factors(although#it works with integers or even reals)at same data.

Example

```
smoke <-matrix(c(51,43,22,92,28,21,68,22,9),ncol=3,byrow=TRUE)
colnames(o) <-c("High","Low","Middle")
rownames(o) <-c("current","former","never")
smoke<-as.table(smoke)
smoke
        High Low Middle
current   51  43     22
former    92  28     21
never     68  22      9
```

5.12 Computation measures of Central Values

A measure of central tendency (also referred to as measures of centre or central location) is a summary measure that attempts to describe a whole set of data with a single value that represents the middle or centre of its distribution.

There are three main measures of central tendency:

- The mode
- The median
- The mean.

Each of these measures describes a different indication of the typical or central value in the distribution.

Mode

The mode is the most commonly occurring value in a distribution.

Consider this dataset showing the retirement age of 11 people, in whole years.

54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

This table shows a simple frequency distribution of the retirement age data.

Age	Frequency
54	3
55	1
56	1
57	2
58	2
60	2

The most commonly occurring value is 54, therefore the mode of this distribution is 54 years.

Advantage of the mode:

The mode has an advantage over the median and the mean as it can be found for both numerical and categorical (non-numerical) data.

Limitations of the mode:

The are some limitations to using the mode. In some distributions, the mode may not reflect the centre of the distribution very well. When the distribution of retirement age is ordered from lowest to highest value, it is easy to see that the centre of the distribution is 57 years, but the mode is lower, at 54 years.

54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

It is also possible for there to be more than one mode for the same distribution of data, (bi-modal, or multi-modal). The presence of more than one mode can limit the ability of the mode in describing the centre or

typical value of the distribution because a single value to describe the centre cannot be identified.

In some cases, particularly where the data are continuous, the distribution may have no mode at all (i.e. if all values are different).

In cases such as these, it may be better to consider using the median or mean, or group the data in to appropriate intervals, and find the modal class.

Median

The median is the middle value in distribution when the values are arranged in ascending or descending order.

The median divides the distribution in half (there are 50% of observations on either side of the median value). In a distribution with an odd number of observations, the median value is the middle value. Looking at the retirement age distribution (which has 11 observations), the median is the middle value, which is 57 years:

54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

When the distribution has an even number of observations, the median value is the mean of the two middle values. In the following distribution, the two middle values are 56 and 57, therefore the median equals 56.5 years:

52, 54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

Advantage of the median:

The median is less affected by outliers and skewed data than the mean, and is usually the preferred measure of central tendency when the distribution is not symmetrical.

Limitation of the median:

The median cannot be identified for categorical nominal data, as it cannot be logically ordered.

Mean:

The mean is the sum of the value of each observation in a dataset divided by the number of observations. This is also known as the arithmetic average. Looking at the retirement age distribution again:

54, 54, 54, 55, 56, 57, 57, 58, 58, 60, 60

The mean is calculated by adding together all the values (54+54+54+55+56+57+57+58+58+60+60 = 623) and dividing by the number of observations (11) which equals 56.6 years.

Advantage of the mean:

The mean can be used for both continuous and discrete numeric data.

Limitations of the mean:

The mean cannot be calculated for categorical data, as the values cannot be summed. As the mean includes every value in the distribution the mean is influenced by outliers and skewed distributions. Most common statistics of central tendency can be calculated with functions in the native stats package. The psych and Desc Tools packages add functions for the geometric mean and the harmonic mean. The describe function in the psych package includes the mean, median, and trimmed mean along with other common statistics. In the native stats package, summary is a quick way to see the mean, median, and quantiles for numeric variables in a data frame. The mode is not commonly calculated, but can be found in DescTools.

```
Input =(
"Stream                    Fish
 Mill_Creek_1               76
 Mill_Creek_2              102
 North_Branch_Rock_Creek_1  12
 North_Branch_Rock_Creek_2  39
 Rock_Creek_1               55
```

```
 Rock_Creek_2              93
 Rock_Creek_3              98
 Rock_Creek_4              53
Turkey_Branch             102
")
Data = read.table(textConnection(Input),header=TRUE)
  Arithmetic mean
mean(Data$ Fish, na.rm=TRUE)
 [1] 70
 Geometric mean
library(psych)
geometric.mean(Data$ Fish)
 [1] 59.83515
library(DescTools)
Gmean(Data$ Fish)
[1] 59.83515
 Harmonic mean
library(psych)
harmonic.mean(Data$ Fish)
 [1] 45.05709
>library(DescTools)
Hmean(Data$ Fish)
[1] 45.05709


 Median
median(Data$ Fish, na.rm=TRUE)
 [1] 76
 Mode
library(DescTools)
Mode(Data$ Fish)
```

```
[1] 102
```

Summary and describe functions for means, medians, and other statistics
The interquartile range (IQR) is 3rd Qu. minus 1st Qu.

```
summary(Data$ Fish)          # Also works on whole data frames
                             # Will also report count of NA's

Min. 1st Qu.Median    Mean 3rd Qu.    Max.
    12      53      76      70      98      10

library(psych)
describe(Data$ Fish,          # Also works on whole data frames
type=2)        # Type of skew and kurtosis
vars n mean    sd median trimmed  mad min max range  skew kurtosis   se
1    1 9   70 32.09     76      70 34.1  12 102    90 -0.65    -0.69 10.7
```

5.13 Measures of Dispersion

Such as range, variance, standard deviation, and coefficient of variation—can be calculated with standard functions in the native stats package. In addition, a function, here called summary.list, can be defined to output whichever statistics are of interest.

Range

```
range(Data$ Fish, na.rm=TRUE)

[1]  12 102     # Min and max
max(Data$ Fish, na.rm=TRUE) - min(Data$ Fish, na.rm=TRUE)

[1] 90
```

Sample variance

```
var(Data$ Fish, na.rm=TRUE)
 [1] 1029.5
```

Standard deviation

```
sd(Data$ Fish, na.rm=TRUE)
[1] 32.08582
```

Coefficient of variation, as percent

```
sd(Data$ Fish, na.rm=TRUE)/
mean(Data$ Fish, na.rm=TRUE)*100
[1] 45.83689
```

5.14 Fitting of Distributions

Fitting distributions consists in finding a mathematical function which represents in a good way a statistical variable. A statistician often is facing with this problem: he has some observations of a quantitative character.

Distribution fitting is the procedure of selecting a statistical distribution that best fits to a data set generated by some random process. In other words, if you have some random data available, and would like to know what particular distribution can be used to describe your data, then distribution fitting is what you are looking for.

x_1, x_2,… x_n and he wishes to test if those observations, being a sample of an unknown population, belong from a population with a pdf (probability density function) $f(x,\theta)$, where θ is a vector of parameters to estimate with available data.

We can identify 4 steps in fitting distributions:

1) Model/function choice: hypothesize families of distributions;
2) Estimate parameters;
3) Evaluate quality of fit;
4) Goodness of fit statistical tests.

To face fitting distributions dealing shortly with theoretical issues and practical ones using the statistical environment and language R

- R is a language and an environment for statistical computing and graphics flexible and powerful. And going to use some R statements concerning graphical techniques (§ 2.0), model/function choice (§ 3.0),
- parameters estimate (§ 4.0), measures of goodness of fit (§ 5.0) and most common goodness of fit tests (§6.0).
- To understand this work a basic knowledge of R is needed. We suggest a reading of "An introduction to R"
- R statements, if not specified, are included in stats package.

Goodness of fit tests

Goodness of fit tests indicate whether or not it is reasonable to assume that a random sample comes from a specific distribution. They are a form of hypothesis testing where the null and alternative hypotheses are:

- H0: Sample data come from the stated distribution
- HA: Sample data do not come from the stated distribution

These tests are sometimes called as omnibus test and they are distribution free, we shall point out our attention to normality tests.

- The chi-square test

It is the oldest goodness of fit test dating back to Karl Pearson (1900). The test may be thought of as a formal comparison of a histogram with the fitted density.

An attractive feature of the chi-square (x goodness of fit test is that it can be applied to any univariate distribution for which you can calculate the cumulative distribution function. The chisquare goodness of fit test is applied to binned data (i.e., data put into classes). This is actually not a restriction since for non-binned data you can simply calculate a histogram or frequency table before generating the chi-square test. However, the value of the chi-square test statistic is dependent on how the data is binned. Another disadvantage of this test is that it requires a sufficient sample size in order for the chi square approximation to be valid. The chisquare goodness of fit test can be applied either to discrete distributions or continuous ones.

- Kolmogorov-Smirnov and Anderson-Darling tests are restricted to continuous distributions. Estimating the model parameters with sample is allowed with this test. The chi-square test is defined for the hypothesis

H0: the data follow a specified distribution

HA: the data do not follow the specified distribution

List of R statements useful in fitting distributions.

The package including statement is written in parenthesis.

ad.test(): Anderson-Darling test for normality (nortest)

chisq.test(): chi-squared test (stats)

cut: divides the range of data vector into intervals

cvm.test(): Cramer-von Mises test for normality (nortest)

ecdf(): computes an empirical cumulative distribution function (stats)

fitdistr(): Maximum-likelihood fitting of univariate distributions (MASS)

goodfit(): fits a discrete (count data) distribution for goodness-of-fit tests (vcd)

hist(): computes a histogram of the given data values (stats)

jarque.bera.test(): Jarque-Bera test for normality (tseries)

ks.test(): Kolmogorov-Sminorv test (stats)

kurtosis(): returns value of kurtosis (fBasics)

lillie.test(): Lilliefors test for normality (nortest)

mle(): estimate parameters by the method of maximum likelihood (stats4)

pearson.test(): Pearson chi-square test for normality (nortest)

plot(): generic function for plotting of R objects (stats)

qqnorm(): produces a normal QQ plot (stats)

qqline(), qqplot(): produce a QQ plot of two datasets (stats)

sf.test(): test di Shapiro-Francia per la normalità (nortest)

shapiro.test():Shapiro-Francia test for normalità (stats)

skewness(): returns value of skewness (fBasics)

table(): builds a contingency table (stats)

5.15 Correlation Coefficient and fitting of regression lines using R :

The quantity r, called the linear correlation coefficient, measures the strength and the direction of a linear relationship between two variables. The linear correlation coefficient is sometimes referred to as the Pearson product moment correlation coefficient in honor of its developer Karl Pearson.

The mathematical formula for computing r is:

$$r = \frac{n\sum xy - \left(\sum x\right)\left(\sum y\right)}{\sqrt{n\left(\sum x^2\right) - \left(\sum x\right)^2}\sqrt{n\left(\sum y^2\right) - \left(\sum y\right)^2}}$$

where n is the number of pairs of data.

The value of r is such that -1 < r < +1. The + and – signs are used for positive linear correlations and negative linear correlations, respectively.

Positive correlation:

If x and y have a strong positive linear correlation, r is close to +1. An r value of exactly +1 indicates a perfect positive fit. Positive values indicate a relationship between x and y variables such that as values for x increase, values for y also increase.

Negative correlation:

If x and y have a strong negative linear correlation, r is close to -1. An r value of exactly -1 indicates a perfect negative fit. Negative values indicate a relationship between x and y such that as values for x increase, values for y decrease.

No correlation:

If there is no linear correlation or a weak linear correlation, r is close to 0. A value near zero means that there is a random, nonlinear relationship between the two variables

Note that r is a dimensionless quantity; that is, it does not depend on the units employed.

A perfect correlation of ± 1 occurs only when the data points all lie exactly on a straight line. If r = +1, the slope of this line is positive. If r=-1, the slope of this line is negative.

A correlation greater than 0.8 is generally described as strong, whereas a correlation less than 0.5 is generally described as weak. These values can vary based upon the "type" of data being examined. A study utilizing scientific data may require a stronger correlation than a study using social science data.

Coefficient of Determination, r2 or R2:

The coefficient of determination, r2, is useful because it gives the proportion of the variance (fluctuation) of one variable that is predictable from the other variable.

It is a measure that allows us to determine how certain one can be in making predictions from a certain model/graph.

The coefficient of determination is the ratio of the explained variation to the total variation.

The coefficient of determination is such that $0 < r2 < 1$, and denotes the strength of the linear association between x and y.

The coefficient of determination represents the percent of the data that is the closest to the line of best fit. For example, if $r = 0.922$, then $r2=0.850$, which means that 85% of the total variation in y can be explained by the linear relationship between x and y (as described by the regression equation). The other 15% of the total variation in y remains unexplained.

The coefficient of determination is a measure of how well the regression line represents the data. If the regression line passes exactly through every point on the scatter plot, it would be able to explain all of the variation. The further the line is away from the points, the less it is able to explain.

Question and Answers

1. What is R?

   R is a programming language which is used for developing statistical software and data analysis.

2. How R commands are written?

   By using # at the starting of the line of code like #division commands are written.

3. What is t-tests() in R?

   It is used to determine that the means of two groups are equal or not by using t.test() function.

4. What are the advantages of R?

   • It is used for managing and manipulating of data.
   • No license restrictions
   • Free and open source software.
   • Graphical capabilities of R are good.
   • Runs on many Operating system and different hardware and also run on 32 & 64 bit processors etc.

5. What are the disadvantages of R Programming?

   The disadvantages are
   • Lack of standard GUI
   • Not good for big data.
   • Does not provide spreadsheet view of data.

6. How to create new variable in R programming?

   For creating new variable assignment operator '<-' is used For e.g. mydata$sum <- mydata$x1 + mydata$x2 21.

7. What are R packages?

Packages are the collections of data, R functions and compiled code in a well-defined format and these packages are stored in library.

8. What is the workspace in R?

Workspace is the current R working environment which includes any user defined objects like vector, lists etc.

9. How can you merge two data frames in R language?

Data frames in R language can be merged manually using cbind () functions or by using the merge () function on common rows or columns.

10. What is the process to create a table in R language without using external files?

MyTable= data.frame ()

edit (MyTable)

The above code will open an Excel Spreadsheet for entering data into MyTable.

11. Explain about the significance of transpose in R language

Transpose t () is the easiest method for reshaping the data before analysis.

12. What are with () and BY () functions used for?

With () function is used to apply an expression for a given dataset and BY () function is used for applying a function each level of factors.

Answer the following questions:

1. Explain in detail about Advantages and Disadvantages of R.
2. Explain about various Data Types support by R programming.
3. Explain about Data Manipulation in R
4. Explain in detail about Distribution.
5. Explain briefly about Computation measures of Central Values.

Books for Study:

1. Balagurusamy, E. (2010) Programming in ANSI C (5th Edition), Tata McGraw-Hill Education, New Delhi.
2. Ashok, M. Kamthane (2006) programming with ANSI and Turbo C, Dorling Kindersley (India) Pvt. Ltd., New Delhi.
3. Purohit S. G., Gore S.D. and Deshmukh S.K. (2010) Statistics using R, Narosa Narosa Publishing House Pvt. Ltd., New Delhi.
4. Ugarte, M. D., A.F. Militino, A.T. Arnholt (2008) Probability and Statistics with R,
   CRC Press, Taylor & Francis Group, London.
5. Peter Dalgaard (2008) Introductory Statistics with R,
   Springer India Private Limited, New Delhi.