Programming for Problem Solving using C

Unit I

Introduction to Programming:

Programming is the process of instructing computers to carry out tasks.
A computer program is a sequence of instructions that the computer executes.

Characteristics of the computer:

Computer is the electronic device which receives data and instruction from user, process the data according to the given instruction and produce the result.

The characteristics of the computer are

1. Speed:

   A computer is a very fast device. The computer takes a fraction of seconds to perform any operation. The speed of computer is measured in nano seconds (10^{-})
2. Accuracy:

   The accuracy of computer is very high and for a particular computer, each and every calculation is performed with the same accuracy.
3. Storage Capacity:

   Computers can store data and instruction with a lot of volume.
4. Diligence:

   Unlike human being a computer is free from tiredness, lack of concentration etc. and hence can work for hours together without creating any error.
5. Reliability:

   it gives consistent result for similar set of data
6. Versatility:

   Computers are capable of performing all levels of tasks- simple or complex.

Limitations of a Computer System:

1. Computers can’t Think:
2. Computers can’t Decide:
3. Computers can’t Express their Ideas:
4. Computers can’t Implement:
5. It required power to operate.
6. Problem may occur due to system breakdown.

Anatomy or Structure of Digital Computer

A computer can be defined as an electronic device capable of processing the data and producing the information.

The computer system essentially comprises three important parts

1. input device
2. central processing unit (CPU)
3. output device.

Input:

Sending the data and command to the computer is known as input.

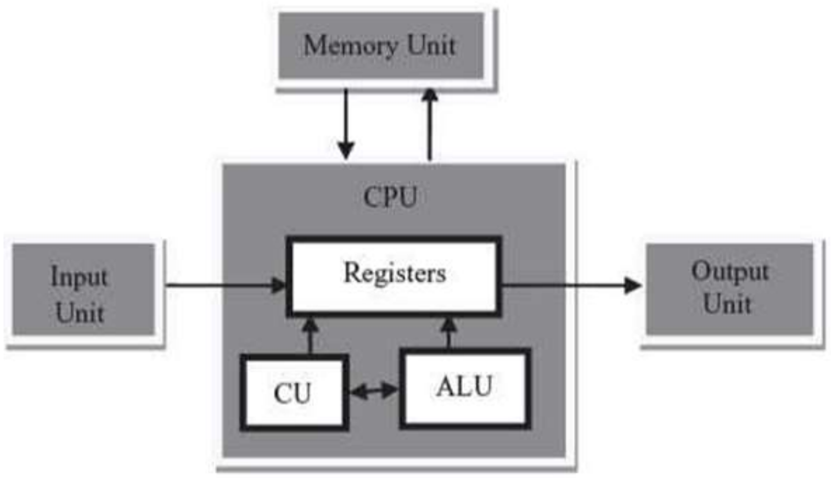
Central Processing Unit (CPU)

CPU controls, coordinates and supervises the operations of the computer. It is responsible for processing of the input data. CPU consists of Arithmetic Logic Unit (ALU) and Control Unit (CU) and Memory unit.

- ALU performs all the arithmetic and logic operations on the input data.
- CU controls the overall operations of the computer i.e. it checks the sequence of execution of instructions, and, controls and coordinates the overall functioning of the units of computer.
- Memory Unit: Memory unit stores the data, instructions, intermediate results and output, temporarily, during the processing of data.

Output:

The Output unit provides the output in a form that is understandable by the user.



Hardware Vs Software:

A computer system is divided into two categories:

1. Hardware
2. Software.

Hardware

It refers to the physical and visible components of the system such as a monitor, CPU, keyboard and mouse.

Types of Computer Hardware

- Input Devices
- Output Devices
- Storage Devices
- Internal Component

1. **Input Devices:**

   Input Devices are those devices through which a user enters data and information into the Computer or simply, User interacts with the Computer. Examples of Input Devices are Keyboard, Mouse, Scanner, etc.

2. **Output Devices:**

   Output Devices are devices that are used to show the result of the task performed by the user. Examples of Output Devices are Monitors, Printers, Speakers, etc.

3. **Storage Devices:**

   Storage Devices are devices that are used for storing data and they are also known as Secondary Storage Data. Examples of Storage Devices are CDs, DVDs, Hard Disk, etc

4. **Internal Component:**

   Internal Components consists of important hardware devices present in the System. Examples of Internal Components are the CPU, Motherboard, etc.

Software

Software is a set of programs (sequence of instructions) that is used to perform a well-defined function or some specified task.

Types of software

Software's are broadly classified into two types,

1. , **System Software**

2. **Application Software**..

System Software

System software basically controls a computer's internal functioning and also controls hardware devices such as monitors, printers, and storage devices, etc.

Types of system software:

1. Operating System:

Operating system is software that acts as an interface between the user and system hardware. It also provides various services to other computer software.

It is the main program of a computer system. When the computer system ON it is the first software that loads into the computer's memory. Basically, it manages all the resources such as memory, CPU, printer, hard disk, etc., and

Examples of operating systems are Linux, Apple macOS, Microsoft Windows, etc.

2. Device Driver:

It is a computer program that operates or controls a particular device attached to a computer or automaton.

Application Software

Application Software Software that performs special functions or provides functions. It is a product or a program that is designed only to fulfill end-users' requirements.

Computer languages:

A computer language is a formal language used to communicate with a computer. Computer languages can be classified into

3. Machine language
4. Assembly language
5. High level language

Machine Language:

Machine language is the language written as string of binary 0's and 1's. It is the only language which a computer understand without using a translation program.

Advantages:

1. Faster execution.
2. Requires less amount of memory space.

Disadvantages:

1. It is machine dependent.
2. It is difficult to program and write.
3. It is difficult to debug.
4. It is difficult to modify.

Assembly Language:

It a low level programming language that allows the user to write program using mnemonic codes.

It requires translator known as assembler to convert program from assembly language to machine language.

Advantages:

1. It is easy to understand and use.
2. It is easy to locate and correct error.
3. It is easy to modify.

Disadvantages:

1. It is machine dependent.

High level Languages:

It allows the user to write the programs in a language that resembles English words. It is machine independent language.

It requires translator known as Compilerr to convert program from assembly language to machine language.

Advantages:

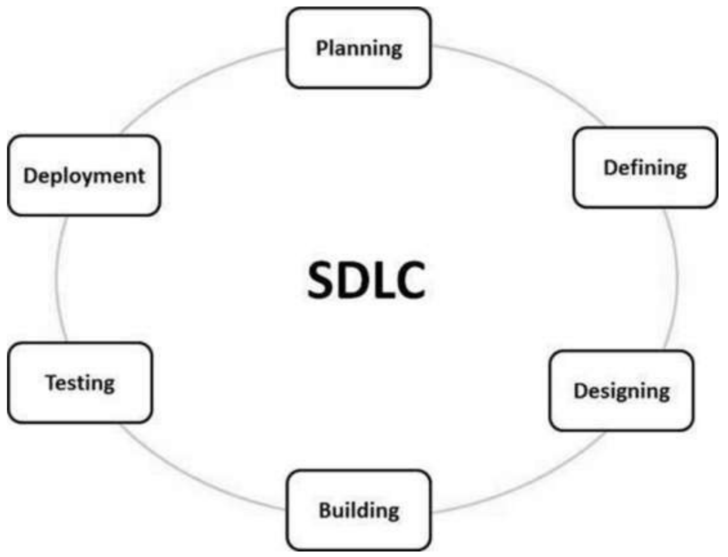
1. It is machine independent.
2. It is easy to learn and use.
3. It is easy to locate and correct error.

Disadvantages:

1. It require more space and time to execute the program.

SDLC:

The Software Development Life Cycle (SDLC) is a structured process that enables the production of high-quality, low-cost software, in the shortest possible production time.



Planning and requirement analysis

During this phase, all the relevant information is collected from the customer to develop a product as per their expectation.

Business analyst and Project Manager set up a meeting with the customer to gather all the information like what the customer wants to build, who will be the end-user, what is the purpose of the product.

Defining Requirements

Once the requirement is clearly understood, the SRS (Software Requirement Specification) document is created. This document should be thoroughly understood by the developers and also should be reviewed by the customer for future reference.

Design

In this phase, the requirement gathered in the SRS document is used as an input and software architecture that is used for implementing system development is derived.

Implementation or Coding

Once the developer gets the Design document. The Software design is translated into source code. All the components of the software are implemented in this phase.

Testing

Testing starts once the coding is complete and the modules are released for testing. In this phase, the developed software is tested thoroughly and any defects found are assigned to developers to get them fixed.

Deployment

Once the product is tested, it is deployed in the production environment or first UAT (User Acceptance testing) is done depending on the customer expectation.

Maintenance

After the deployment of a product on the production environment, maintenance of the product i.e. if any issue comes up and needs to be fixed or any enhancement is to be done is taken care by the developers.

Structured programming

It is a programming paradigm aimed at improving the clarity, quality, and development time of a computer program by making extensive use of the structured control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

control structures

Mechanisms that allow us to control the flow of execution within a program.

There are three main categories of control structures:

Iteration:

A control structure that allows some lines of code to be executed many times.

Selection:

A control structure where the program chooses between two or more options.

Sequence:

A control structure where the program executes the items in the order listed.

Types of Programming languages:

A computer language is a formal language used to communicate with a computer.

Computer languages can be classified into

6. Machine language
7. Assembly language
8. High level language

Machine Language:

Machine language is the language written as string of binary 0's and 1's. It is the only language which a computer understand without using a translation program.

Advantages:

3. Faster execution.
4. Requires less amount of memory space.

Disadvantages:

5. It is machine dependent.
6. It is difficult to program and write.
7. It is difficult to debug.
8. It is difficult to modify.

Assembly Language:

It a low level programming language that allows the user to write program using mnemonic codes.

It requires translator known as assembler to convert program from assembly language to machine language.

Advantages:

4. It is easy to understand and use.
5. It is easy to locate and correct error.
6. It is easy to modify.

Disadvantages:

2. It is machine dependent.

High level Languages:

It allows the user to write the programs in a language that resembles English words. It is machine independent language.

It requires translator known as Compilerr to convert program from assembly language to machine language.

Advantages:

4. It is machine independent.
5. It is easy to learn and use.
6. It is easy to locate and correct error.

Disadvantages:

1. It require more space and time to execute the program.

Introduction to C

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.

The main reason for its popularity is because it is a fundamental language in the field of computer science.

Need for C Language:

If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar.

The main features of the C language:

- General Purpose and Portable
- Low-level Memory Access
- Fast execution
- Clear Syntax

Developing a C Program

The development of a C program involves primarily three steps:

- Writing the C program
- Compile the program and
- Execute the program

Software components such as an operating system, a text editor, and the C compiler, assembler, and linker are required to apply these steps.

Error:

Errors are the problems or the faults that occur in the program, which makes the behavior of the program abnormal, and experienced

Debugging:

Programming errors are also known as the bugs or faults, and the process of removing these bugs is known as **debugging**.

Types of Error are

- Syntax error
- Run-time error
- Linker error
- Logical error
- Semantic error

Syntax error:

These errors are mainly occurred due to the mistakes while typing or do not follow the syntax of the specified programming language.

If we want to declare the variable of type integer,

int a; // this is the correct form

Int a; // this is an incorrect form.

Commonly occurred syntax errors are:

- If we miss the parenthesis (}) while writing the code.
- Displaying the value of a variable without its declaration.
- If we miss the semicolon (;) at the end of the statement.

Run-time error:

Sometimes the errors exist during the execution-time even after the successful compilation known as run-time errors.

Linker error:

Linker errors are mainly generated when the executable file of the program is not created. This can be happened either due to the wrong function prototyping or usage of the wrong header file.

Logical error:

The logical error is an error that leads to an undesired output.

Semantic error:

Semantic errors are the errors that occurred when the statements are not understandable by the compiler.

The following can be the cases for the semantic error:

int a, b, c;

a+b = c;

Debugging techniques

▪Comment out (or delete) code -tests to determine whether removed code was source of problem

▪Test one function at a time

▪Add print statements -Check if certain code is reachable -check current state of variables

▪test the edges -code often breaks at the beginning or end of the loop, the entry or exit of a function.

Unit II

Tokens:

C tokens are the basic building blocks in C language which are constructed together to write a C program. C Tokens are classified into

1. **Keywords**
2. **Identifiers**
3. **Constants and variables**
4. **Strings**
5. **Special symbols**
6. **Operators**

Keywords

Keywords have fixed meanings, and the meaning cannot be changed. There are a total of 32 keywords in 'C'. Keywords are written in lowercase letters.

Ex

int , struct, long, return, if, while, etc.,

Identifiers

Identifier is the name given to variable, function or array. Identifiers are the user-defined names consisting of 'C' standard character set. Following rules must be followed for identifiers:

1. The first character must always be an alphabet and consequent letters may be alphabet or numeric.
2. A keyword cannot be used as an identifier.
3. It should not contain any special character (except underscore), whitespace character.
4. The name must be meaningful.

Variable

A variable is an identifier which is used to store some value. The value of variables can change during the execution of a program
Ex:
'age' variable can be used to store the age of a person.

Constants

Constants are the fixed values that never change during the execution of a program. Following are the various types of constants:

Integer constants

An integer constant is nothing but a value consisting of digits or numbers with out decimal point.

Ex: 123

Real Constants

A real constant is nothing but a value consisting of digits or numbers with decimal point. The real constants are also called as floating point constants.

Ex: 123.12

Character constants

A character constant contains only a single character enclosed within a single quote (').

Ex: 'a'

String constants

A string constant contains a sequence of characters enclosed within double quotes (").

Ex: "program"

Strings

Strings are always represented as an array of characters having null character '\0' at the end of the string. This null character denotes the end of the string.

Ex: char a[10] = "c Prog";

Special symbols in C

Some special characters are used in C, and they have a special meaning which cannot be used for another purpose.

- **Square brackets [ ]:** The opening and closing brackets represent the single and multidimensional subscripts.
- **Simple brackets ( ):** It is used in function declaration and function calling.

  For example, printf() is a pre-defined function.

- **Curly braces { }:** It is used in the opening and closing of the code. It is used in the opening and closing of the loops.
- **Comma (,):** It is used for separating for more than one statement
- **Hash/pre-processor (#):** It is used for pre-processor directive. It basically denotes that we are using the header file.
- **Asterisk (*):** This symbol is used to represent pointers and also used as an operator for multiplication.
- **Period (.):** It is used to access a member of a structure or a union.

Structure of C program:

The basic structure of C program is as follows

```
Documentation section

Link section

Definition section

Global declaration section

main()

{

Local declaration section

Executable part

}

Subprogram section
```

Function 1

Function 2

:

:

Function n

<u>Documentation section:</u>

This section consist of set of comment lines giving information about the program, author, etc.,

<u>Link section:</u>

This section provide instruction to compiler to link function from system library.

<u>Definition section:</u>

This section consists of set of lines that defines the symbolic constant.

<u>Global declaration section:</u>

In this section we declare some variables that are used in more than one function.

<u>main()</u>

It is the compulsory function. The main() function intimate, this is beginning of

executable part to the compiler.

The two parts of main function are declaration part and executable part, they must appear with in opening and closing braces.

1. declaration part declares all the variables used in executable part.
2. executable part may include at least one programming statement.

<u>Subprogram section:</u>

This section include all the user defined function, that are called from the main function.

Operators:

An operator is a symbol that tells the computer to perform certain mathematical and logical operations. C operators can be classified into

1. Arithmetic operator
2. Relational operator
3. Logical operator
4. Assignment operator
5. increment and decrement operator
6. Conditional operator

1. Arithmetic operator:

C provide all the basic arithmetic operators such as

+ (Addition or Unary plus) Ex: a+b or +a

- (Subtraction or Unary minus) Ex: a-b or -a

* (Multiplication) Ex: a*b

/ (Division) Ex: a/b

% (Modulo division) Ex: a%b

Unary operator:

The operators that operates single operand is called unary operator.

Binary operator:

The operators that operates two operand is called binary operator.

The integer division truncate any fractional part. The modulo operator return the remainder of the integer division.

An expression that involve arithmetic operators is called arithmetic expression. The arithmetic expression can be classified into three types,

1. integer arithmetic expression
2. real arithmetic expression
3. mixed-mode arithmetic expression

1. Integer arithmetic:

If both operands in single arithmetic expression are integer then it is called as integer arithmetic expression. If a and b are integer then the integer arithmetic expression is as follows

a-b

a+b

2. Real arithmetic:

If both operands in single arithmetic expression are real then it is called as integer arithmetic expression. If a and b are real then the following real arithmetic expression is as follows

a*b

a/b

3. Mixed-mode arithmetic:

In an arithmetic expression, if one operand is integer and another operand is real then the expression is called as mixed-mode arithmetic expression. If a is integer and b is real then the following is the mixed-mode expression.

a-b

a+b

2. Relational operators:

These operators used to compare two quantities and take decision depending on their relation. The result of relational expression is either TRUE or FALSE. The operators are

<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to
==	is equal to

!=	is not equal to

An expression containing relational operators is called relational operator. A simple relation expression contains only one relational operator and its general form is

ae-1 relational operator ae-2

where ae-1 and ae-2 are arithmetic expressions, which may simple constant or

variable. If operands of relational expression are arithmetic expression, then the arithmetic expressions are first evaluated and its results are compared by relational operator.

Example:

(a+b) > c

When the above relational expression encountered, first the arithmetic expression (a+b) is evaluated and its result is compared with the variable c.

The expression 4<=5 returns TRUE

The expression 4>=5 returns FALSE

The relational expressions are used in control structures.

<u>3. Logical operator:</u>

The logical operator used to combine two or more relational expression to make decision. The logical operators are

&&	logical AND
||	logical OR
!	logical NOT

An expression that containing logical operators to combine two or more relational expressions is termed as logical expression.

Syntax

re-1 logical operator re-2

Where re-1 and re-2 are relational expression.

Example:

```
mark>=40 && mark<=60
```

the above logical expression return TRUE when both relational expressions returns TRUE, otherwise FALSE.

Truth table for logical AND:

Logical AND returns TRUE only if both inputs are TRUE, otherwise FALSE.

a	b	a && b
F	F	F
F	T	F
T	F	F
T	T	T

Truth table for logical OR

Logical OR returns TRUE if either one of input is TRUE, if return FALSE only if both inputs are FALSE.

a	b	a | | b
F	F	F
F	T	T
T	F	T
T	T	T

Truth table for Logical NOT

Logical NOT returns TRUE if input is FALSE and FALSE if input is TRUE.

a	NOT a
T	F

F T

4. Assignment operator:

Assignment operator used to assign result of expression to a variable.’=’ is the assignment operator. The general form of assignment statement is

var = exp;

where var is variable and exp is an expression. First evaluate the expression on the right side of equal sign and its result assigned to the variable on left hand side of equal sign.

Example:

x= a+b;

Short hand assignment or compound assignment operator.

The general form of short hand assignment operator is

var **op** = exp;

Where var is a valid variable, exp is expression and op is a C binary arithmetic operator. the op= is known as short hand assignment operator. The above expression is equivalent to the following expression

var = var op exp;

The short hand assignment operators used only when the variable on left hand side of assignment operator also present in right hand side of assignment operator.

Ex:

x = x + 10; is equivalent to x + = 10;

when this statement is executed , 10 is added to x and result is assigned to x.

Advantages short hand operator:

1. The left hand side variable need not be repeated.

2. easier to read and write.

5. Increment and Decrement operator:

The increment and decrement operators are

++ (increment operator)

-- (decrement operator)

The increment operator adds 1 to the operand and decrement operator subtract 1 from the operand.

Example: m++ increments the value of variable m by 1

m—decrements the value of variable m by 1

Post-increment/Post-increment

First process the operand then the value of operand will be increment or decrement by 1.

Pre-increment/Pre-decrement

First increment or decrement the value of operand by 1 then it will be processes

Consider the following expression

(a) M=5;

y=++m;

in this case, the value of y and m would be 6

(b) m=5

Y=m++

In this case, the value of y would be 5 and m would be 6.

6. Conditional operator:

Ternary Operator:

The operator that operates three operands are called ternary operator.

The ternary operator ?: is the conditional operator that construct the conditional expression of the form

```
exp1?exp2:exp3
```

where exp1, exp2 and exp3 are the expressions.

First evaluate the exp1. If it is non-zero(true), then the exp2 is evaluated and becomes the value of expression. If exp1 is false, then exp3 is evaluated and its value becomes the value of expression.

Example:

```
a = 10;

b = 15;

x = ( a > b) ? a : b;
```

in this example, x will be assigned the value of b.

Expression

Definition:

An expression is combination of variables, constants and operators arranged as per syntax of the language.

Evaluation of expression

Expressions are evaluated using an assignment statement of the form

```
Variable = expression;
```

Where ‘variable’ is any valid C variable name. When the above statement is encountered, the expression evaluated first and the result is assigned to the variable on the left hand side. All the variables used in the expression must be assigned values.

Example:

```
x=a * b – c;
```

when the above expression is encountered, the expression a * b –c is evaluated first and its value assigned to the variable x.

Precedence of arithmetic operators

An arithmetic expression without parentheses will be evaluated from left to right using the rules of precedence of operators. There are tow distinct priority level, they are

Higher priority	* / %
Lower priority	+ -

The evaluation of expression without parentheses include two left-to-right passes through the expression. During first pass, the higher priority operators are applied as they appeared. During the second pass, the lower priority operators are applied

a=9, b=12 and c=3 then

x = a – b / 3 + c * 2 – 1; is evaluated as follows,

x= 9 – 12 / 3 + 3 * 2 – 1;

First pass

Step1: x= 9 - 4 + 3 * 2 -1

Step2: x= 9 – 4 +6 – 1

Second pass

Step3: x = 5 + - 1

Step4: x = 11 – 1

Step5: x =10

The expression with parentheses include three left-to-right passes. During first pass, an expression within parentheses evaluated as they appeared. During second pass, higher priority operators are evaluated as they encountered, during third pass lower priority operators evaluated as the encountered

Example:

Consider the expression with parentheses as shown below.

x = a – b / (c + 3) * (2 -1) is evaluated as follows,

first pass:

step1: x = 9 – 12 / 6 * (2 -1)

step2: x = 9 – 12 / 6 * 1

second pass:

step3: x=9 – 2 * 1

step4: x=9 – 2

third pass:

step5:x=7

Formatted output function:

The printf function is the formatted output function that can used to control the alignment and spacing of prints-out on the terminals. The general form of the printf statement is

printf(“control string”,arg1,arg2,..,argn);

The control string consist of three types of items

1. Characters that can be printed on the screen as they appear.
2. Format specification that define the output format for display of each item.
3. Escape sequence characters such as \n, \t and \b

The arguments arg1, arg2,….,argn are the variables whose values are formatted and printed according to specification of control string. Control string and arguments are separated by commas.

The argument should match in number , order and type with format specifications.

(i) The format specification for printing an integer number is

% w d

The % symbol indicates that the format specification follows.

Where the integer w specifies minimum field width for the output and d is the data type character that specifies the value to be printed as an integer.

The number is written right justified with in given field width. The alignment can be changed by placing the minus sign before w.

Example:	Output
printf("%d",9876);	9876
printf("%6d",9876);	9876
printf("%-6d",9876);	9876

(ii) The format specification for printing a real number is

% w.p f

The % symbol indicates that the format specification follows.

Where the integer w specifies minimum number of field with for the output and the integer p indicates the number of digits to be displayed after the decimal point. f is the data type character that specifies the value to be printed as floating point.

The number is written right justified with in given field width. The alignment can be changed by placing the minus sign before w.

Example:

float y= 98.7654;	Output
printf("%7.4f",y);	98.7654
printf("%7.2f",y);	98.77
printf("%-7.2f",y)	98.77

(iii) The format specification for printing a single character is

%wc

The % symbol indicates that the format specification follows.

Where the integer w specifies minimum field width for the output and c is the data type character that specifies the value to be printed as character.

The character will be displayed right-justified in field of w columns. The alignment can be changed by placing the minus sign before w

Example:

```
char c='x';                 Output
printf("%c", c);         x
printf("%2c",c);          x
printf("%-2c",c);               x
```

(iv) The format specification for printing string of character is

%w.p s

Where w specifies minimum field width for display the output and p instruct that only the first p characters of string are to be displayed.

s is the data type character that specifies the value to be printed as string. The string will be displayed right-justified in field of w columns. The alignment can be changed by placing the minus sign before w

Example:

```
char name[10] = "new delhi";          Output
printf("%s",name);                    new delhi
printf("%12s",name);                    new delhi
printf("%12.3s",name);             new
printf("%-12.3s",name);       new
```

Formatted input function

Formatted input refers to an input data that has been arranged in a particular format. Scanf statement is used for reading the formatted data. The general form of scanf function is scanf(“control string”,arg1,arg2,…,argn);

The control string specifies the field format in which the data is to be entered and arguments arg1,agr2,…,argn specify the address of location where the data is stored. Control string and arguments are separated by comma.

(i) The field specification for reading an integer number is

% w d

The percentage sign specifies that a conversion specification follows. w is an integer that specifies field width of number to be read and d is known as data type character indicate that the number to be read is in integer mode.

Example:

scanf(“%2d%5d”,&num1,&num2);

data line

50 31426

The value 50 is assigned to num1 and 31426 to num2. suppose the input data is as follow

31426 50

The value 31 assigned to num1 and 426 is assigned to num3. the value 50 will be unread.

(ii) The format specification for reading an real number is

%f

f is known as data type character indicate that the number to be read is in real mode

Example:

scanf(“%f%f%f”,&x,&y,&z);

data line

475.89 43.2 678

Will assign 475.89 to x, 43.2 to y and 678.0 to z

(iii) The format specification for reading a character is

%c

c is known as data type character indicate that the data to be read is in character mode

Example:

```
scanf("%c%c",&a,&b);
```

data line

```
xy
```

will assign the character 'x' to a and 'y' to b.

(iv) The format specification for reading a string is

```
%s
```

s is known as data type character indicate that the data to be read is in string mode

Example:

```
scanf("%s%s", &name,&addr);
```

data line

```
ram gym
```

will assign the string "ram" to name and "gym" to addr.

Control statements

Program is set of statements which are normally executed sequentially in order in which they are appears. The control statements can be classified into

1. Branching statement
2. Looping statement

1. Branching statement

These statement can be used to alter the program execution sequence based on certain condition or unconditionally. Branching statement can be classified into

(a) conditional branching
(b) unconditional branching

(a) conditional branching statement:

These statements alters the programming execution sequence based on condition. The following are the conditional branching statemenmts

1. simple if statement
2. if …. else
3. if … else if ladder
4. nested if
5. switch case

<u>Simple if statement</u>

The general form of simple if statement is

```
if(test condition)

{

Statement-block;

}

Statement-x;
```

The 'statement-block' may be a single statement or a group of statements. If the test-condition is true, the statement block will be executed; otherwise the statement-block will be skipped.

Example:

```
if(category == sports)

{

marks = marks + 5;

}

printf("%f",marks);
```

The above program tests the category of student. If the student belongs to sports category, then additional 5 marks are added.

if …else statement

The if…else statement is an extension of simple if statement. The general form is

```
if(test expression)
{
True-block statements;
}
else
{
False-block statements;
}
Statement-x;
```

If the test expression is true, then the true-block statement(s), immediately following the if statement are executed; otherwise, the false-block statement(s) are executed. In either case, either true-block or false-block will be executed, not both.

Example:

```
if(category == sports)
{
marks = marks + 5;
}
else
{
Marks = marks + 2;
}
printf("%f", marks);
```

In the above program test the category of student. If the student belongs to sports category, then bonus marks 5 will be added; otherwise bonus marks 2 will be added.

The if…else if ladder

This is the multipath decision making statement. A miltipath decision is a chain of ifs in which the statement associated with each else is an if. Its general form is

```
            if(condition 1)

            statement -1;

            else if(condition 2)

            statement -2;

            else if (condition 3)

            statement -3;

            -----------

            ----------

            else if(condition n)

            statement – n;

            else

            default-statement;

      statement –x;
```

The condition are evaluated from top. As soon as a true condition is found, the statement associated with it is executed and control is transferred to statement – x. When all n conditions become false, then the final else containing default-statement will be executed.

Example:

```
            If(marks>=60)

            printf("First class");

            else if(marks>=50)

            printf("Second class");

            else if(marks>=40)

            printf("Third class");
```

```
else
printf("Fail");
```

The above program test the marks of student. If marks is greater than or equal to 60, then 'first class' will be printed. If not, test whether the marks is greater than or equal to 50, if yes, then 'second class' will be printed. If not, test whether the marks is greater than or equal to 40, if yes, then 'third class' will be printed. If not, then the final else statement will be executed and 'fail' will be printed.

Nested if

if statement placed within another if statement is called nested if statement. The general form of nested if is

```
        if(test condition1)
        {
        If(test condition2)
        {
        Statement -1;
        }
        else
        {
        Statement -2;
        }
        }
        else
        {
        Statement – 3;
        }
    Statement –x;
```

If condition -1 is false the statement-3 will be executed; otherwise it continues to perform the second test. If condition-2 is true, the statement-2 will be evaluated; otherwise statement-2 will be evaluated and then the control is transferred to statement-x.

Example:

```
if(gender='f')

{

if(balance>5000)

bonus= balance * 0.05;

else

bonus=balance * 0.02;

}

else

{

bonus=balance * 0.02;

}

balance = bonus + balance;
```

In the above program test the gender of person. If gender if female then balance of person is checked. If balance is greater than 5000 then bonus will be 5 percent; otherwise 2 percent bonus will be added.

Unconditional branching statement:

goto statement used to branch unconditionally from one point to another point in the program.

The goto statement requires a label in order to identify the place where the branch is to be made. The general form of goto and label is as follows.

```
goto label;

----------
```

```
----------

-----------

Label:

Statement;
```

The label: can be anywhere in the program either before or after the goto label;

If label is placed before goto statement then some statement will be executed repeatedly.

If label is placed after goto statements then some statements are skipped.

Example:

```
goto begin;
```

when the above statement is executed, the flow of control jump to statement immediately following the begin;. This happens unconditionally.

```
main()
{
Double x,y;
read:
scanf("%f",&x);
if(x<0)
goto read;
y=sqrt(x);
printf("%f",y);
goto read;
}
```

The switch statement

Designing program using 'if' statement is increase complexity when number of alternatives increases for that switch statement can be used.

The general form of switch statement is

```
switch(expression or variable)
{
case value-1: statement 1;
        break
case value-2: statement 2;
        break;
-------------------
------------------------
case value-n: statement n;
        break;
default: statement:
}
```

The switch statement test the value of expression or variable against the case values and when a match is found, a block of statement associated with that case is executed. When there is no match found the default statement is executed.

Example:

```
scanf("%c",&c);
switch(c)
{
case 'a':
case 'e':
case 'i' :
case 'o':
case 'u': printf("the given character is vowel"); break;
default: printf("the given character is not a vowel");

}
```

If we give a or e or i or o or u for the variable c, then the result is 'the given character is vowel' ; otherwise the result is 'the given character is not a vowel' .

Iterative control statement(loop control statements)

The iterative control statements are used to execute the block of repeatedly until certain condition satisfied or a specific number of times. The c has three loop control statements, they are

1. while loop
2. do .. while loop
3. for loop

While loop

The general form is

```
while(condition)

{

Body of loop;

}
```

It is the entry control loop. While loop first evaluate the condition, if the condition is true then the it execute the body of loop. The test condition once again evaluated and if condition is true, the body of loop executed once again. This process continued until the test condition finally becomes false.

Example:

```
sum=0;

n=1;

while(n<=10)

{

sum = sum + n * n;
n = n+1;
}
printf("sum = %d", sum);
```

in the above example the body of loop is executed 10 times for n=1,2,….,10 each time adding the square of value of n, which is incremented inside the loop.

Do ..while statement:

The general form is

```
do
{
Body of loop;
} while(condition);
```

It is the exit control loop. In do while statement the body of loop evaluated first. At the end of the loop, the condition is evaluated. If the condition is true, the program continue to evaluate body of loop once again. this process continues as long as condition is true.

Example:

```
int n, sum=0;
do
{
Scanf("%d",&n);
sum=sum+n;
}while(n>0);
```

The above code read a number from keyboard until a zero or negative number is keyed in.

The for statement

The general form is

```
for(initialization;test-condition;incrememt/decrement)
```

```
{
Body of loop;
}
```

It is the entry control loop. It involve three statements are as follows

Initialization of control variable done first, using assignment statement such as i=0and j=1. the i and j are called loop control variable/

The value of control variable is tested using the test-condition. The test-condition is the relational expression such as i<10. if the test-condition is true then the body of loop will be executed; otherwise the loop is terminated.

The value of control variable is increment/ decrement in third statement. the updated value of control variable is again tested to see whether it satisfies the test condition. If the condition is satisfied the body of loop is again executed.

Example:

```
for(x=0;x<10;x++)
{
printf("%d", x);
}
```

In the above program the body of loop executed 10 times and prints digits from 0 to 9.

Break statement:

When the break statement is encountered inside the loop, the loop is immediately exited and program continues with the statement immediately following the loop. The general form of break statement is

```
break;
```

Example:

```
for(i=0;i<1000;i++)
{
Scanf("%d",&n);
If(n<0)

break;

sum=sum+n;

}
```

In the above example the for loop written to read 1000 values. If we want to add set of values less than 1000 values, then we enter negative value after the last value to make the end of input.

Continue statement:

When continue statement encountered inside the loop, skip the statement after 'continue' statement and the loop continued with next iteration. The general form is

```
continue;
```

Example:

```
for(i=0;i<100;i++)

{

scanf("%d",&n);

if(n<0)

continue;

sum=sum+n;

}
```

In the above example for loop used to read 100 values. If the read value is negative then it will not be added; otherwise added with the sum.

Unit III

Array:

Array is a group of related data item of same type that shares common name. The individual elements of array are referred by index number. Index number starts from 0 to array size minus 1. There are three types of array.

1. one dimensional array

2. two dimensional array

3. multi dimensional array

One dimensional array:

List of items can be given one variable name using only one subscript value is called one dimensional array.

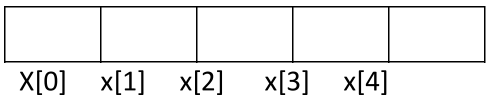
Syntax:

```
type array_name[size];
```

the type specifies the type of element that will be contained in the array such as int, float or char and size indicates the maximum number of element that can be stored inside the array. For example

```
int x[5];
```

declares the x to be an array containing 5 integer element and computer reserves five storage locations as shown below



initialization of array

the general form is

```
type array_namr[size] = {list of values};
```

one dimensional array variable can be initialize like any other variable at the time of declaration by placing list of values with in curly braces, the values in the list are separated by comma. For example,

```
int number [3] = {0,0,0};
```

will declare the variable number as an array of size 3 and will assign zero to each element. If the number of values in the list is less than the number of elements, then only that many element will be initialized. The remaining element set to zero automatically.

The size may be omitted. In such case the compiler allocate enough space for all initialized elements. For example

```
Int number[] = {1,1,1,1};
```

Will declare number array to contain four elements with initial vales 1.

Example program

```
int a[5],sum=0;

for(int i=0;i<5;i++)

{

Scanf("%d",&a[i]);

}

for(i=0;i<5;i++)

{

Sum=sum+a[i];

}

printf("%d'",sum);
```

in the above program a is the one dimensional array variable that can hold 5 integer values and for loop used to receive 5 integer values and to calculate sum.

Two dimensional array:

List of data item can be given one variable name using two subscript value is called two dimensional array. The general form of two dimensional array is

Syntax:

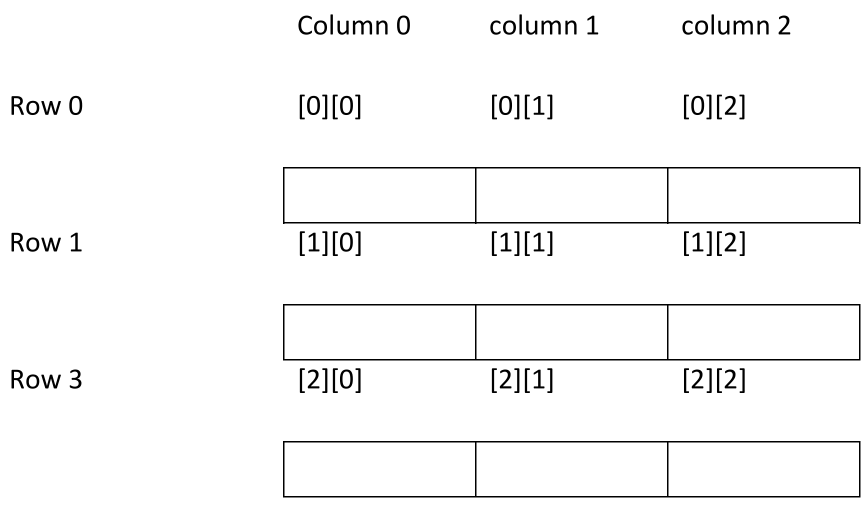
```
type array_name[row-size][column-size2];
```

The type specifies the type of element that will be contained in the array such as int, float or char and row-size specifies maximum number of row and column-size specifies maximum number of column within row.

Example:

```
int number[3][3];
```

The array variable number stored in memory as shown below



Example Program

```
int a[5][5],sum=0,i,j;

for(int i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

Scanf("%d",&a[i][j]);

}

}

for(i=0;i<5;i++)

{

for(j=0;j<5;j++)

{

Sum=sum+a[i][j];

}

}

printf("%d'",sum);
```

in the above program a is the two dimensional array variable that can hold 25 integer values and for loop used to receive 25 integer values and to calculate sum.

Two dimensional array initialization:

The general form is

data_type array_name[row_size][col_size] = {{row1 values}, {row2 values},…..{row N values}};

in two dimensional array initialization is done row by row. Each row value placed within braces and each value in a row separated by comma.

Ex

```
int a[2][3]={{0,0,0},{1,1,1}};
```

initialize the elements of fist row to zero and second row to one

Multidimensional array:

List of data items can be given one name using three or more subscript value is called multidimensional array. The general form is

```
type array_name[s1][s2]…[s3];
```

where si is the size of ith dimension.

Example:

```
Int number[3][3][3];
```

number is the three dimensional array declared to contain 27 integer type elements.

Example program

```
int a[5][5][5],sum=0,i,j.k;

for(int i=0;i<5;i++)

{

for(j=0;j<5;j++)
```

```
{
for(k=0;k<5;k++)
{
Scanf("%d",&a[i][j][k]);
}
}
for(i=0;i<5;i++)
{
for(j=0;j<5;j++)
{
for(k=0;k<5;k++)
{
Sum=sum+a[i][j][k];
}
}
printf("%d'",sum);
```

In the above program a is the three dimensional array variable that can hold 125 integer values and for loop used to receive 125 integer values and to calculate sum.

Searching:

Searching is the process of finding whether a particular element is present in the ;list or not.

Two popular search methods are

Linear Search

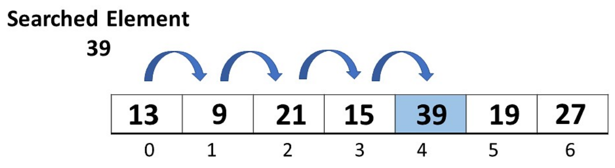
Binary Search

Linear search:

Linear search, often known as sequential search, is the most basic search technique. In this type of search, you go through the entire list and try to fetch a match for a single element. If you find a match, then the address of the matching target element is returned.

On the other hand, if the element is not found, then it returns a NULL value.

Following is a step-by-step approach employed to perform Linear Search Algorithm.



The procedures for implementing linear search are as follows:

Step 1: First, read the search element (Target element) in the array.

Step 2: In the second step compare the search element with the first element in the array.

Step 3: If both are matched, display "Target element is found" and terminate the Linear Search function.

Step 4: If both are not matched, compare the search element with the next element in the array.

Step 5: In this step, repeat steps 3 and 4 until the search (Target) element is compared with the last element of the array.

Step 6 - If the last element in the list does not match, the Linear Search Function will be terminated, and the message "Element is not found" will be displayed.

The c Program to implement linear search is as follows

```
main()
{
 int array[100], search, c, n;
 printf("Enter number of elements in array\n");
 scanf("%d", &n);
 printf("Enter %d integer(s)\n", n);
 for (c = 0; c < n; c++)
  scanf("%d", &array[c]);
 printf("Enter a number to search\n");
 scanf("%d", &search);
 for (c = 0; c < n; c++)
 {
  if (array[c] == search)    /* If required element is found */
  {
    printf("%d is present at location %d.\n", search, c+1);
    break;
  }
 }
 if (c == n)
  printf("%d isn't present in the array.\n", search);
 return 0;
}
```

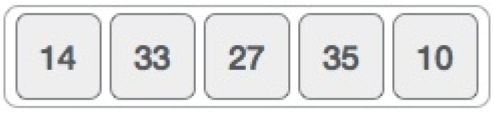
Bubble Sort Algorithm:

In this algorithm,

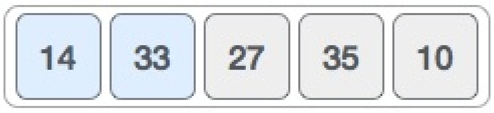
- traverse from left and compare adjacent elements and the higher one is placed at right side.
- In this way, the largest element is moved to the rightmost end at first.
- This process is then continued to find the second largest and place it and so on until the data is sorted.

  Working of Bubble Sort is as follows

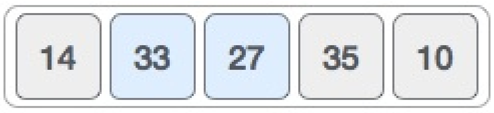
  We take an unsorted array for our example. Bubble sort takes O(n^2) time so we're keeping it short and precise.



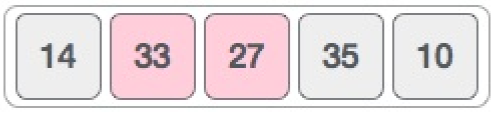
Bubble sort starts with very first two elements, comparing them to check which one is greater.



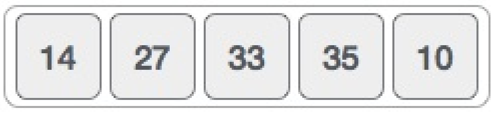
In this case, value 33 is greater than 14, so it is already in sorted locations. Next, we compare 33 with 27.



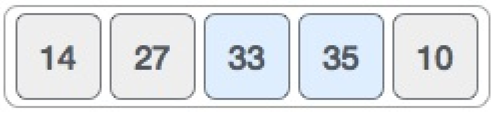
We find that 27 is smaller than 33 and these two values must be swapped.



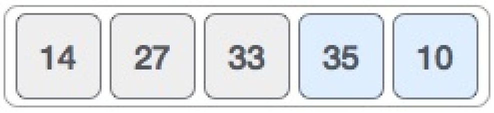
The new array should look like this –



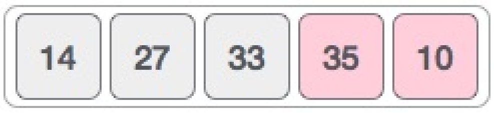
Next we compare 33 and 35. We find that both are in already sorted positions.



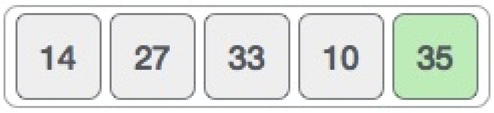
Then we move to the next two values, 35 and 10.



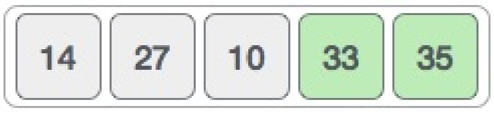
We know then that 10 is smaller 35. Hence they are not sorted.



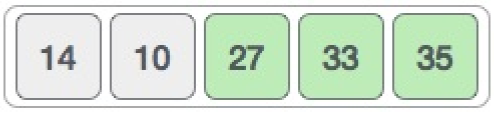
We swap these values. We find that we have reached the end of the array. After one iteration, the array should look like this –



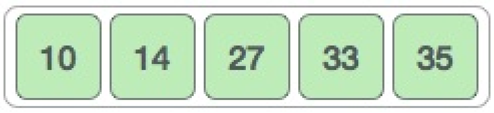
To be precise, we are now showing how an array should look like after each iteration. After the second iteration, it should look like this –



Notice that after each iteration, at least one value moves at the end.



And when there's no swap required, bubble sorts learns that an array is completely sorted.



Now we should look into some practical aspects of bubble sort.

The c Program to implement bubble sort is as follows

```
#include<stdio.h>
void main ()
{
    int i, j,temp;
    int a[5] = { 10, 35, 32, 13, 26};
   for(i = 0; i < n; i++)
    {
      for(j = i+1; j < n; j++)
        {
           if(a[j] < a[i])
           {
               temp = a[i];
               a[i] = a[j];
               a[j] = temp;
           }
        }
    }
for(i = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
 }
```

Strings

The string can be defined as the one-dimensional array of characters terminated by a null ('\0').

Each character in the array occupies one byte of memory, and the last character must always be \0.

The termination character ('\0') is important in a string since it is the only way to identify where the string ends. When we define a string as char s[10], the character s[10] is implicitly initialized with the null in the memory.

There are two ways to declare a string in c language.

1. By char array
2. By string literal

Let's see the example of declaring **string by char array** in C language.

char ch[10]={‘k', 'm', 'g", '\0'};

As we know, array index starts from 0, so it will be represented as in the figure given below

0	1	2	3	4	5	6	7	8	9
k	m	g	\0						

While declaring string, size is not mandatory. So we can write the above code as given below:

char ch[]={‘k', 'm', 'g", '\0'};

We can also define the **string by the string literal** in C language. For example:

char ch[]="kmg";

In such case, '\0' will be appended at the end of the string by the compiler.

Difference between char array and string literal

The main differences between char array and literal.

We need to add the null character '\0' at the end of the array by ourself whereas, it is appended internally by the compiler in the case of the character array.

String Example in C

Let's see a simple example where a string is declared and being printed. The '%s' is used as a format specifier for the string in c language.

```
#include<stdio.h>
#include <string.h>
main()
{
  char ch[10]={‘k', 'm', 'g", '\0'};
   char ch2[10]="kmg";
   printf("Char Array Value is: %s\n", ch);
   printf("String Literal Value is: %s\n", ch2);
}
```

String Handling Functions:

String handling functions are defined in string.h header file. These functions are used to manipulate the string. The most commonly used string handling function in c are as follows.

strlen()

strlen( ) function in C gives the length of the given string. Syntax for strlen( ) function is given below

```
size_t = strlen ( str );
```

strlen( ) function counts the number of characters in a given string and returns the integer value.

It stops counting the character when null character is found. Because, null character indicates the end of the string in C.

strcpy()

strcpy( ) function copies contents of one string into another string. Syntax for strcpy function is given below.

```
char * strcpy ( char * destination, const char * source );
```

Example:

strcpy (str1, str2) – It copies contents of str2 into str1.

If destination string length is less than source string, entire source string value won’t be copied into destination string.

For example, consider destination string length is 20 and source string length is 30. Then, only 20 characters from source string will be copied into destination string and remaining 10 characters won’t be copied and will be truncated.

strcat()

strcat( ) function in C language concatenates two given strings. It concatenates source string at the end of destination string. Syntax for strcat( ) function is given below.

char * strcat (char * destination, const char * source);

Example:

strcat (str1, str2); – str2 is concatenated at the end of str1.

As you know, each string in C is ended up with null character (‘\0’).

In strcat( ) operation, null character of destination string is overwritten by source string’s first character and null character is added at the end of new destination string which is created after strcat( ) operation.

strcmp()

strcmp( ) function in C compares two given strings and returns zero if they are same.

If length of string1 < string2, it returns < 0 value. If length of string1 > string2, it returns > 0 value. Syntax for strcmp( ) function is given below.

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

Example

strcmp(str1,str2) – return 0 if str1 and str 2 are same.

strcmp( ) function is case sensitive. i.e, “A” and “a” are treated as different characters.

strrev()

strrev( ) function reverses a given string in C language. Syntax for strrev( ) function is given below.

char *strrev(char *string);

Example

strrev(str1) – return reverse of str1.

strrev( ) function is non standard function which may not available in standard library in C.

strlwr( )

strlwr( ) function converts a given string into lowercase. Syntax for strlwr( ) function is given below.

char *strlwr(char *string);

strlwr( ) function is non standard function which may not available in standard library in C.

strupr()

strupr( ) function converts a given string into uppercase. Syntax for strupr( ) function is given below.

char *strupr(char *string);

strupr( ) function is non standard function which may not available in standard library in C.

**Note: Refer the example program from class work note.

Storage Classes:

A storage class represents the visibility and a location of a variable. It tells from what part of code we can access a variable. A storage class is used to describe the following things:

- The variable scope.
- The location where the variable will be stored.
- The initialized value of a variable.
- A lifetime of a variable.

We have four different storage classes in a C program

- auto
- register
- static
- extern

Auto

The variables defined using auto storage class are called as local variables. Auto stands for automatic storage class. A variable is in auto storage class by default if it is not explicitly specified.

The scope of an auto variable is limited with the particular block only. Once the control goes out of the block, the access is destroyed. This means only the block in which the auto variable is declared can access it.

A keyword auto is used to define an auto storage class. By default, an auto variable contains a garbage value.

```
#include<stdio.h>
void main()
{
auto int i;
{
int i=2
printf("i in inner block is%d",i);2
}
printf("i in outer block is%d",i); 1
}
```

register

The keyword register is used to declare a register storage class. The variables declared using register storage class has lifespan throughout the program.

It is similar to the auto storage class. The variable is limited to the particular block. The only difference is that the variables declared using register storage class are stored inside CPU registers instead of a memory. Register has faster access than that of the main memory.

The variables declared using register storage class has no default value. These variables are often declared at the beginning of a program.

```
#include<stdio.h>
main()
{
register int r;
int *p=&r;
}
```

static

The static variables are used within function/ file as local static variables. They can also be used as a global variable

- Static local variable is a local variable that retains and stores its value between function calls or block and remains visible only to the function or block in which it is defined.
- Static global variables are global variables visible only to the file in which it is declared.
- The static variable has a default initial value zero and is initialized only once in its lifetime.
- The lifespan of a static variable is in the entire program code.

```
#include<stdio.h>
void next();
static int count = 1;
void main()
{
while(count<10)
{
next();
count++;
}
}
void next()
{
static int i;
i++;
printf("iteration = %d and count=%d",i,count);
}
```

Extern

Extern stands for external storage class. Extern storage class is used when we have global functions or variables which are shared between two or more files.

Keyword extern is used to declaring a global variable or function in another file to provide the reference of variable or function which have been already defined in the original file.

The variables defined using an extern keyword are called as global variables. These variables are accessible throughout the program. Notice that the extern variable cannot be initialized it has already been defined in the original file

```
ext.h
extern in q=10;
extex1
#include<stdio.h>
#include "ext.h"
void main()
{
printf("%d",q);
}
```

Storage Class	Declaration	Storage	Default Initial Value	Scope	Lifetime
auto	Inside a function/block	Memory	Unpredictable	Within the function/block	Within the function/block
register	Inside a function/block	CPU Registers	Garbage	Within the function/block	Within the function/block
extern	Outside all functions	Memory	Zero	Entire the file and other files where the variable is declared as extern	program runtime
Static (local)	Inside a function/block	Memory	Zero	Within the function/block	program runtime
Static (global)	Outside all functions	Memory	Zero	Global	program runtime

Unit - IV

Function:

The function is the self contained block of code for a specific task.

Library function:

printf, sqrt, etc are the library function for that user not required to write code.

User-defined functions:

It has to be developed by the user at the time of program writing.

Need for user-defined functions:

If the program contains only main function then the program may become large and complex so the process of debugging, testing and maintaining the program is difficult.

There fore the large problem can be divided in to functional parts and each part coded independently and later combined into single unit results easier to debug and test.

There are times when some types of operation is repeated at many points throughout the program. In such situation, we may repeat the program statement whenever they are needed. Another way is to design a function that can be called and used whenever required.

The advantages of using sub-functions are

1. Length of source program can be reduced.
2. We ca easily identify the error.
3. It facilitates the modular programming concept.

The multifunction program:

Function is self contained block of code that perform a particular task. Once a function is designed and packed then it can be treated as black box, that take some data from calling function and return a value to calling function. the internal operation of function are invisible to rest of the program.

Consider the following C function

```
printline()
{
int i;
for(i=1;i<=20;i++)
printf("*");
}
```

The above function print a line of 20 character length. This function can be used in the following program.

```
main()
{
printline()
printf("\nWelcome to C function");
printline();
}
```

The output will be

```
********************
Welcome to C function
********************
```

In this example main and printline are the user defined function. program execution begins at main function and when printline function is called, the execution control transferred to the function printline(). After executing the printline function the control transferred back to main function.

After executing printf statement control transferred to the printline function. after completing printline function execution control resumed to main function and program ends.

Any function can call any other function.

A function can call itself is called recursion.

A called function may call another function

The form of C function:

The general form of the function is

```
function_name(argument list)
argument declaration;
{
local variable declaration;
executable statement1;
executable statement1;
---------------------
--------------------
return (expression);
}
```

Function name:

The function name must follow the same rule of identifiers. The function name must not be duplicating the library functions.

Argument list:

The argument list contain valid variable names separated by comma. The list must surrounded by parentheses. No semicolon follows the close parentheses. The argument variables receive value from calling function. All argument variable must be declared for their types.

Local variable declaration:

The variable that are used only with in function declared in this part.

Executable part:

The function block may contain any number of executable statement.

Return :

The return statement is the mechanism for returning value to the calling function.

Example:

```
main()
{
int x,y,z;
scanf(“%d%d”,&x,&y)
z=mul(x,y);
printf(“%d”,z);
}
```

```
mul(x,y)

int x,y;

{

int m;

m= x*y;

return(m);

}
```

Return values and their types:

The function may or may not return value to calling function. Returning values to calling function can be achieved by return statement. The return statement must return at most only one value to calling function.

The general form of return statement is

```
return

        or

 return(exp)
```

the plain return statement does not return any value it just return the execution control to the calling function.

The return statement with one parameter (exp) returns the value of expression to the calling function.

The exp may contain an arithmetic expression or a simple variable or constant.

Example:

```
mul(x,y)
int x,y
{
int p;
p=x*y;
return(p);
}
```

The above function return a variable p to the calling function.

The default return type of function is integer. We can force the function to return some other type by specifying return type in function header.

Ex:

```
double mul(x,y)
float mul(x,y)
```

the first example return double type value and second one return float type value to calling function.

Calling a function.

A function can be called by simply using the function name in statement. Example:

```
main()

{

int p;

p=mul(2,3);

printf("%d", p);

}
```

When complier encounters a function call, the control transferred to the function mul(x.y). this function is executed line by line as described and value is returned when a return statement is encountered.

Category of functions:

Depending on whether the argument are present or not and whether the value returned or not, the function can be classified into four types.

1. function with no argument and no return type
2. function with no argument and return type.
3. function with argument and no return type
4. function with argument and return type.

Function with no argument and no return type

In this type, there is no data transfer between calling function and called function. That is, called function does not receive any value from calling function and called function does not send any value to calling function.

Example:

```
main()

{

mul();

}




mul()

{

int m,x=10,y=20;

m= x*y;

printf("%d",m);

}
```

Function with no argument and return type.

In this type, called function does not receive any value from calling function and called function send value to calling function.

Example:

```
main()

{

int res;

res = mul();

printf(“%d”,res);

}




mul()

{

int x=10,y=5;

int m;

m= x*y;

return(m)

}
```

Function with argument and no return type:

In this type, values passed from calling function to called function and calling function does not receive any value from called function.

Example:

```
main()

{

int x=10,y=5;

mul(x,y);

}




mul(x,y)

int x,y;

{

int m;

m= x*y;

printf("%d",m);

}
```

Function with argument and return type:

In this type, values passed from calling function to called function and also called function to called function.

Example:

```
main()
```

```
{

int x=10,y=5,res;

res=mul(x,y);

}




mul(x,y)

int x,y;

{

int m;

m= x*y;

return(m);

}
```

Passing Parameters:

we can pass parameters to function in two ways

1. pass by value
2. pass by reference

Pass by value:

The process of passing actual value of variable is known as pass by value.

Pass b reference:

The process of passing the address of variable is known as pass by reference.

pass by value:

In this type, photocopy of variable passed to function, so any changes within function are not reflected in calling function.

Example:

```
main()

{

int a=10;

change(a);

printf("%d",a);

}


change(a)

int a;

{

if(a>5)

{

a=a+10;

printf("%d",a);

}
```

The value of ‘a’ passed to the function by value, any changes in value of ‘a’ in change function does not reflect in main function. that is value of a displayed by main function is 10 and value of a displayed by sub function is 20.

pass by reference:

In this type, address of variable passed to the function, any changes within function are reflected in calling function. When we pass address to function, the parameter receiving the addresses should be the pointer.

Example:

```
main()

{

int a=10;

change(&a);

printf("%d",a);

}


change(a)

int *p;

{

if(*p>5)

{

*p=*p+10;

printf("%d",*p);

}
```

In the above example, the address of ‘a’ passed to sub function changes, any changes in passed value reflected on main function. that is value of ‘a’ displayed by main function is 20 because the variable ‘a’ and *p both refers same memory location.

Function Prototype:

Function prototype is a declaration of a function that omits the function body but does specify the function's name, number and type of arguments and return type. A function definition specifies what a function does.

```
int fac(int n);
```

This prototype specifies that in this program, there is a function named "fac" which takes an integer argument "n" and returns an integer

Recursion:

Recursion is the process of function calls itself. For example consider the function to evaluate factorial of n is as follows.

```
factorial(n)

int n;

{

int fact;

if(n==1)

return(1);

else
return(fact);

}
```

Assume n=3, since the value of n is not 1, the statement

```
fact = n*factorial(n-1);
```

will be executed with n=3, that is

```
fact = 3 * factorial(2);
```

will be evaluated. The expression on the right hand side include a call to factorial with n=2. this call return the following value.

```
2 * factorial(1);
```

Again factorial is called with n-1, this time function return 1. the sequence of operation summarized as follows

```
fact = 3 * factorial(2)

     = 3 * 2 * factorial(1)

     = 3 * 2 * 1

     = 6
```

The recursion function can be used where the solution is expressed in terms f successively applying the same solution to the subset of problem.

Passing array to function:

To pass an array to called function, it is sufficient to list name of array, with out any subscript, and the size of array as argument. For example,

```
largest(a,n);
```

will pass the elements contained in array a of size n. the largest function must look like

```
largest(a,s)

float a[];

int s;
```

the function largest is defined to take two arguments, the array name and size of array to specify the number of elements in array.

Example:

```
main()

{

float largest();

int value[4] = {2,1,4,3};

prinf("%d", largest(value,4));

}


largest(a,n)

int a[];

int n;

{

int i,max;

max = a[0];

for(i=1;i<n;i++)

if(max<a[i])

max=a[i];
return(max);

}
```

Unit V

Structure:

Structure represents collection of data items of different type using a single name.

Syntax:

```
struct tag_name

{

data_type member1;

data_type member2;

-----------------------

----------------------

data_type member n;

};
```

The struct is the keyword to define structure. tag_name is the user defined identifier.

The structure themselves not a variable, they do not occupy memory until they are associated with the memory variable.

The template is terminated with semicolon.

Each member declared independently for its name and type in a separate statement and member declaration ended with semicolon.

We can declare structure variable anywhere in the program using tag name, syntax is

```
struct tag_name var1,var2…,var n;
```

Example:

```
struct book_bank

{

char title[10];

char author[10];

int pages;

float price;

} book1,book2;
```

Book1,book2 are structure variable.

Giving values to members:

We can assign values to members of structure using the '.'(dot) operator.

For example:

the members itself does not have any meaning. They should be linked with the structure variable in order to give meaning using (.) dot operator.

Example:

```
struct book_bank

{

char title[10];

char author[10];

int pages;

float price;
} book1,book2;
```

book1,book2 are structure variable.

book1.price is the variable representing price of book1. can be treated like any other ordinary variable.

We can also use scanf statement to give values to member through keyboard.

```
scanf("%s",book1.price);
```

Structure initialization:

Like any other variable we can initialize structure variable at the time of declaration by placing list of values within curly braces.

There is one to one correspondence between structure member and list of values in braces.

For example

```
sturct student
{
int weight;
float eight;
} stu1={60,175.3};
```

This assign the value 60 to stu1.weight and 175.3 to stu1.height.

Array of structure:

We can also use array to declare structure variable, each element of array represent a structure variable.

For example:

```
struct mark

{

int m1;

int m2;

int m3;

} stu[3];
```

Defines array called stu, that consist of 3 elements. Each element is defined to be of type struct mark. The three elements are stu[0], stu[1], stu[2].

Comparison of Structure variable:

The structure variable of same type can be compared the same way as ordinary variables.

If p1 and p2 belongs to same structure type, then following comparison are valid.

The operation p1=p2 assigns p2 to p1.

The operation p1 == p2 return 1, if all the members of p1 and p2 are same.

The operation p1 != p2 return 1, if all members of p1 and p2 are not same.

Pointer:

The pointer is a variable that can store address of another variable.

Understanding pointers:

Memory is collection of storage cells. Each cell has a unique number called address and each cell can store a byte of data. the addresses are numbered consecutively, starting from 0 and last address depends on memory size.

Whenever we declare a variable, system allocate memory location to hold the value of variable. this locations have its own address. We can access the value on that location by name of variable or address of that variable.

Example:

int qty = 10;

the above statement instruct the system to allocate memory location for the integer variable 'qty' and put the value 10 on that location. this may represented as follows

qty ← variable

179 ← value

5000 ← address

We can access the value 179 by name qty or address 5000. accessing value on memory location using address is simple.

Accessing address of the variable:

&(address operator) used to access the address of the variable. the operator & immediately preceding a variable return the address of variable. For example,

```
int a=10, *p;

p=&a;
```

the p is the pointer variable and the above statement assign address of ‘a’ to the pointer variable p. this can represented as follows

variable	value	address
a	10	5000
p	5000	5048

Declaring and initializing pointer:

The general form of declaration of pointer is

```
data_type *pt_name;
```

this tells the compiler three things about the variable pt_name

1. The * tells that the variable pt_name is pointer variable.
2. pt_name needs a memory location
3. pt_name points to a variable of type data type.

Example

```
int *p;
```

declares the variable p is pointer variable that points to an integer data type. Remember that the type integer refers to the data type of variable being pointed to by p not the type of value of pointer.

```
float *x;
```

declares as pointer to floating point variable.

once a pointer variable has bee declared, it can be made to point to a variable using assignment statement such as

```
p= &quantity;
```

now p contains address of quantity. This is known as pointer initialization.

Accessing a variable through its pointer:

This is done by the unary operator * (asterisk), usually known as indirection operator.

Example:

```
int quantity,*p,n;

quantity=179;

p=&quantity;

n=*p;
```

first statement declares quantity and n as integer and p as pointer pointing an integer.

Second line assign 179 to quantity.

Third line assign the address of quantity to the pointer variable p.

Fourth statement contain indirection operator * is placed before pointer variable in expression, the pointer returns the value of variable of which the pointer value is the address. The * can remembered as ‘ value at address’, thus the value n would be 179.

Pointer expression or pointer arithmetic:

Like other variables, pointer variable can be used in expression. If p1 and p2 are properly declared and initialized pointers, then the following statements are valid.

```
y = *p1 * *p2;

sum = sum + *p1;

*p2 = *p2 +10;
```

C allows us to add integers to or subtract integer from pointer, as well as subtract one pointer from another pointer. We may use short-hand operator with the pointer

Example,

```
p1++;

--p1;

Sum += *p1;
```

Pointers and Arrays:

When an array is declared, the compiler allocates a base address and sufficient amount of storage to contain all the elements of array in contiguous memory locations. The base address is the location of first element of array.

Example:

```
int a[5] = {1,2,3,4,5};
```

suppose the base address of a is 1000 and assume that each integer requires 2 byte, then 5 elements requires 10 bytes.

If we declare pointer variable p as integer pointer, then we make p to point to the array x by the following statement

```
*p=x;    or        *p=&x[0];
```

The name of array refers the zeroth element of an array. now we can access every value of x using p++ to move from one element to another element.

Pointer and structure:

The name of array refers the zeroth element of an array. the same is true of name of array of structure variables.

Suppose the 'product' is array of variable of structure type. The name 'product' represent address of zeroth element

Ex:

```
struct inventory

{

char name[10];

int number;

float price;

} product[2],*ptr;
```

The above statement declares product as array of two elements, each of the type struct inventory and ptr as pointer to data object of type struct inventory. The assignment

```
ptr=product;
```

would assign the address of zeroth element of product to ptr. That is pointer will now point to product[0]. Its member can be accessed using the following notation.

ptr->name

ptr -> number

ptr ->price

the symbol -> is called arrow operator and is made up of minus and greater than sign.

Drawbacks of console oriented I/O operations:

1. It is time consuming to handle large volume of data through the terminal.
2. The entire data is lost when either the program is terminated or the computer is turned off.

So C provide the concept of file to store the data on disk and read when ever necessary,

The basic file operations:

C supports many function to perform basic file operations, which include

1. naming a file

2. opening a file

3. reading data from a file

4. writing data from a file

5. closing a file

Defining and opening a file:

If we want to store the data in a file in secondary memory, we must specify certain things about the file to operating system, they include

1. filename
2. data structure
3. purpose

1. file name:
   the file name is string of character that make up a valid file name for the operating system. It may contain two parts, a primary name and optional period with extension.

   Example:

   Input.data

   Store

   Student.c

2. Data structure:

The data structure of file is defined as FILE in the library of standard I/O function definitions. There fore all files should be declared as type FILE before they are used.

When we open a file, we must specify what we want to do with the file. For example, we may write data into file or read already existing data.

The general format of declaring and opening a file is as follows.

```
FILE *fp;

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

the first statement declares the variable fp as a "pointer to the data type FILE". The second statement opens the file named filename and assigns an identifier to FILE type pointer fp. This pointer which contains all the information about the file is subsequently used as a communication link between system and program.

3. Purpose:

The second statement also specifies the purpose of opening this file. the mode can be one of the following.

r – opening the file for reading only

w – opening the file for writing only

a – open a file for appending data to it.

For example:

```
FILE *p1,*p2;

p1=fopen(“data”,”r”);

p2=fopen(“results”,”w”);
```

the file data is opened for reading and results is opened for writing. In case results file already exists, its contents are deleted and the file is opened as new. If dat file does not exist, an error will occur.

Closing a File:

A file must be closed as soon as al operations on it have been completed to

1. Prevent any accidental misuse of the file.

2. reopen same file in another mode.

The general form is

```
fclose(file_pointer);
```

Exmaple:

```
fclose(p1);
```

the file associated with the file pointer p1 is closed.

Input/ Output operations on file:

1. The getc and putc function:

The simplest file I/O functions are getc and putc, they handle one character at a time.

The general form of putc function is

```
putc(c.fp1);
```

the above function writes the character contained in the character variable c to the file associated with FILE pointer fp1.

The general form of getc function is

```
c = getc(fp1);
```

the above function used to read a character from file associated with the FILE pointer fp1 and character is assigned to left hand side character variable c.

2. the getw and putw functions:

The getw and putw functions are integer oriented functions. They handle an integer value at a time.

The general form of putw is

```
putw(integ,fp);
```

the above function writes the content of integer variable 'integ' into the file that is associated with the file pointer fp

the general form of getw is

```
i  = getw(fp);
```

the above function read an integer value from the file associated with the file pointer fp and assign the integer value into the left hand side variable i.

3. The fprintf and fscanf function:

These functions can handle group of mixed data simultaneously.

The general form of fprintf is

```
fprintf(fp,”control string”,list of variables);
```

where fp is the file pointer associated with a file that has been opened for writing. The control string contains output specifications for the items in the list. The list may include variables and constants.

Example:

```
fprintf(fp,”%s%d%f”,name,age,7.5);
```

the general form of fscanf is

```
fscanf(fp,”control string”,list);
```

where fp is the file pointer associated with the file that is opened in read mode. The control string specifies format specifications of variable in the list. The list may include variables.

Example:

```
fscanf(fp,”%s%d”,item,&quantity);
```

Error handling during I/O operations:

The following are the error situations that may occur during I/O operation on file.

1. Trying to read beyond the end-of-file mark
2. Device overflow.
3. Trying to use file that has not been opened
4. Trying to perform operation on file, when the file is opened in another mode.
5. Opening a file with an invalid filename
6. Attempt to write to a write-protected file.

(1) The feof function can be used to test for an end of file condition. The general form is

```
feof(fp)
```

Where fp is the FILE pointer and this function returns a nonzero integer if all of the data from the specified file has been read and, return zero otherwise..

Example:

```
if(feof(fp))
printf(“End of Data”);
```

would display the message “End of Data” on reaching end of file condition.

(2) we know that whenever file is opened using fopen function, a file pointer is returned. If the file cannot be opened for some reason, then the function returns a null value. This facility can be used to test whether a file has been opened or not.

Example:

```
fp=fopen(“tests”,”r”);
if(fp == NILL)
printf(“File could not be opened”);
```

in this case, “File could not be opened” is displayed, because the file file ‘tests’ does not exists.

Random access to file:

This is the process of accessing only a particular part of a file and not in reading other parts. This can be achieved with the help of functions

1. fseek
2. ftell

3. rewind

fseek()

this function can be used to move the file pointer to the desired location. The general form is

fseek(fp,offset,position)

fp is a pointer to file concerned.

offset is a number or variable of long type that specifies number of positions to be moved from the location specified by the position.

position ca take one of the following three values.

Values	meaning
0	Beginning of file
1	Current position
2	End of file

Example:

```
fseek(fp,m,0)
```

file pointer moved to (m+1)th byte in the file.

ftell()

this function takes a file pointer and returns number, that corresponds to the current position of file pointer. the general form is

```
n=ftell(fp);
```

n would give the relativeoffset of current position. This means that n bytes have already been read.

rewind()

takes the file pointer and resets the position to start of the file. the general form is

```
rewind(fp);
```

Example:

```
rewind(fp);

n=ftell(fp)
```

where fp is the file pointer and would assign 0 to n because the file position has been set to start of file by rewind.