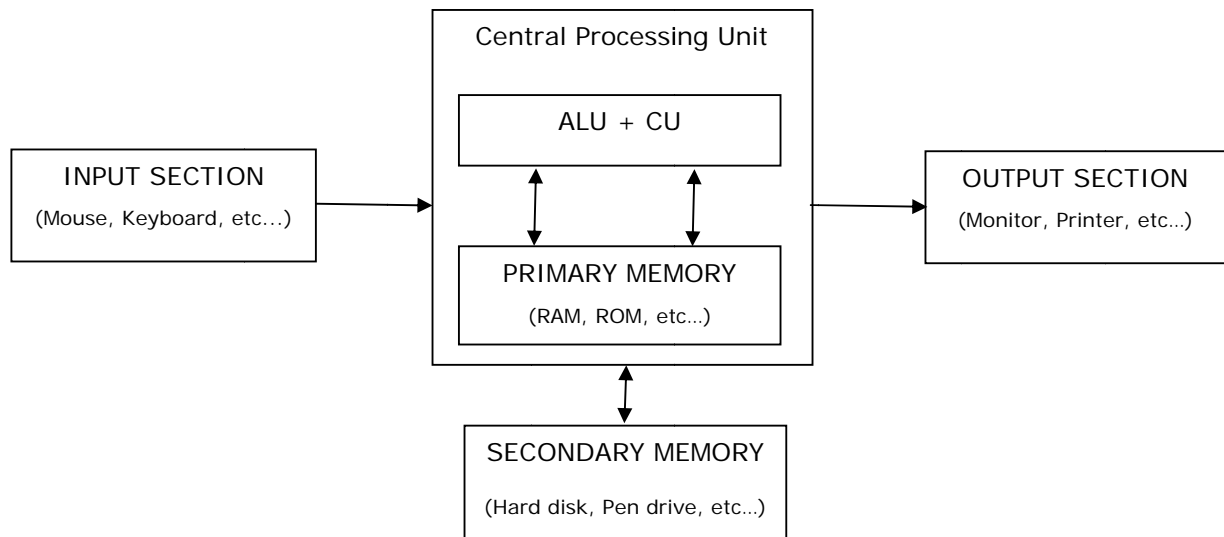


Computer Programming and Utilization (CPU) – 110003**A) Computer Fundamentals****1 Draw the block diagram of computer architecture and explain each block.**

Computer is made up of mainly four components,

- 1) Central processing unit (CPU)
- 2) Input section
- 3) Output section
- 4) Storage devices

**1) Central Processing Unit (CPU):-**

- Central processing unit is a main part of the computer system.
- It contains electronics circuitry that processes the data based on instructions.
- It also controls the flow of data in the system.
- It is also known as brain of the computer.
- CPU consists of,
 - 1.1) Arithmetic Logic Unit (ALU)
 - ✓ It performs all arithmetic calculations and takes logical decision.
 - ✓ It can perform add, subtract, multiply, compare, count, shift and other logical activities.
 - ✓ It calculates very fast.
 - ✓ It takes data from memory unit and returns data to memory unit, generally primary memory.
 - 1.2) Control Unit (CU)
 - ✓ It controls all other units in the computer system.
 - ✓ It manages all operations
 - ✓ It reads instruction and data from memory.
 - 1.3) Primary Memory:-
 - ✓ It is also known as main memory.
 - ✓ The processor or the CPU directly stores and retrieves information from it.



- ✓ This memory is accessed by CPU, in random fashion
- ✓ Generally currently executing programs and data are stored in primary memory
- ✓ Its storage capacity is very small compared to secondary storage.
- ✓ It is very fast in an operation compared to secondary storage
- ✓ RAM is Random Access Memory and it is volatile in nature.
- ✓ ROM is Read Only Memory and it can hold data permanently.
- ✓ PROM is Programmable Read Only Memory and it can hold data permanently. Programmer can store information only once. Modification is not allowed.
- ✓ EPROM is Erasable Programmable Read Only Memory. It can hold data permanently. Programmer can delete and write on it again and again.

2) Input Section:-

- The devices used to enter data in to computer system are called input devices.
- It converts human understandable input to computer controllable data.
- CPU accepts information from user through input devices.
- Examples: Mouse, Keyboard, Touch screen, Joystick etc...

3) Output Section:-

- The devices used to send the information to the outside world from the computer is called output devices.
- It converts data stored in 1s and 0s in computer to human understandable information
- Examples: Monitor, Printer, Plotter, Speakers etc...

4) Storage devices (Secondary memory):-

- Secondary memory is also called Auxilliary memory or External memory.
- User can store data permanently.
- It can be modified easily.
- It can store large data compared to primary memory. Now days, it is available in Terabytes.
- Examples: Hard disk, Floppy disk, CD, DVD, Pen drive, etc...

2 Describe advantages and limitations of computer. Or Explain characteristics of computer.

Advantages

- **Speed:** It can calculate millions of expressions within a fraction of second. The micro second and nano second units are used to measure the speed of computers. There are some problems which cannot be solved within specified time limit without computer.
- **Storage:** It can store data in large quantity using various storage devices. Millions of paper file's data can be stored in single small pen drive. Moreover, it reproduces data whenever we need and whatever format we need. Now a days, Gigabytes and Terabytes are units of data storage devises.
- **Accuracy:** Computer performs the computations at very high speed without any mistakes. For example, multiplication of two very large number takes more time for human and there are high probabilities of mistakes. Computer does it in parts of the second with accuracy level we want. Very high level of accuracy is must in financial transaction, medical surgery, nuclear plant, etc... which can be satisfied by computer only.
- **Reliability:** It is very reliable device. The information stored in computer is available after years in



same form. It works 24 hours without any problem as it does not feel tiredness.

- **Automation:** Once the one task is created in a computer, it can be repeatedly performed again by single click whenever we want. For example, once the software for banking application is installed in a computer, it computes the interest by sending one command.

Limitations

- **Lack of intelligence:** It cannot think while doing work. It does not have natural intelligence. It cannot think about properness or effect of work it is doing. It can only execute the instructions but it cannot think about the correctness of these instructions.
- **Unable to Correct Mistakes:** It cannot correct the mistakes by itself. So if we have provided wrong or incorrect data then it produces wrong results or performs wrong calculations.

3 Describe various types of computer languages and mention its advantages and disadvantages.

Computer languages may be classified in three categories,

1) Machine level language or Low level language:-

- Computer directly understands this language. It is a language of 0's and 1's (binary). Every CPU has its own machine language.
- ADVANTAGES:
 1. It is very fast in execution
 2. It does not require any extra system or software to run the program.
 3. Translation is not required.
 4. Suitable for low volume applications.
- DISADVANTAGES:
 1. Programs are long and difficult to write and understand for human.
 2. Debugging is very difficult task.
 3. It is not portable.
 4. Programmer requires detailed knowledge of architecture of microprocessor.

2) Assembly language:-

- Every machine language instruction is assigned to English word MNEMONIC such that it should describe function of instruction.
- System cannot understand this language directly so we require translator that convert assembly language to machine language. This translator is called assembler.
- Example : 8086 Instruction Set
- ADVANTAGES:
 1. Programs are easy to understand compared to machine level language.
 2. Programs are smaller in size compared to machine level language.
 3. Programs can be entered quickly using alphanumeric keyboard.
- DISADVANTAGES:
 1. It is not portable.
 2. Programmer should know structure of assembly language of microprocessor.
 3. It requires assembler as a translator.

3) Higher level language:-

- We can write programs in English like manner and it is more convenient to use.
- Programmer can perform complex task by using high level languages with less efforts.



- It is similar to natural language and mathematical notation.
- Example: C, C++, Java, etc...
- ADVANTAGES:
 - 1) Easier to learn.
 - 2) Requires less time to write.
 - 3) Provides better documentation.
 - 4) Easier to maintain.
 - 5) It is portable.
- DISADVANTAGES:
 - 1) It requires compiler or interpreter to convert higher level language to machine language.
 - 2) Programmers need to learn structure of high level language.
 - 3) It is bit slow compared to low level and medium level language.

4 Why C is called middle level language?

C is called middle level language because

- Syntax and keywords of C are just like higher level language (English).
- It gives advantages of higher level language through function, modular programming and breakup.
- It gives access to the low level memory through Pointers.
- Moreover it does support the Low Level programming i.e., Assembly Language.
- We can develop application specific programs in C and at the same time we can use features of assembly level language to give more speed and efficiency
- It is not hardware or system dependent. Hence portable programs can be written with C compiler.

5 Write a short note on types of software.

A set of instruction in a logical order to perform a meaningful task is called program and a set of program is called software.

System Software

System software is designed to operate the computer hardware efficiently. It provides and maintains a platform for running application software. Since system software runs at the most basic level of computer, it is called "low-level" software.

System software can be classified into three categories

- 1) Operating system: It controls hardware as well as interacts with users, and provides different services to user. It is a bridge between computer hardware and user. Ex: Windows XP, Linux, UNIX, etc...
- 2) System support software: It makes working of hardware more efficiently. For example drivers of the I/O devices or routine for socket programming, etc...
- 3) System development software: It provides programming development environment to programmers. Ex: Editor, pre-processor, compiler, interpreter, loader, etc.....

Application software

Application software is designed to help the user to perform general tasks (word processing, web browser ...) or some specific task (accounting, ticket booking ...). Example: Enterprise software, Accounting software, Office suites, Graphics software and media players.

Application software is classified into two categories.

- 1) General purpose software: It is used widely by many people for some common task, like word processing, web browser, excel, etc... It is designed on vast concept so many people can use it.



- 2) Special purpose software: It is used by limited people for some specific task like accounting software, tax calculation software, ticket booking software, banking software etc... It is designed as per user's special requirement.

6 Define the following terms:

1. Program

A set of instruction in a logical order to perform a meaningful task is called program.

2. Software

A set of instruction in a logical order to perform a meaningful task is called program, and set of program is called software.

3. Hardware

Physical parts of computer is known as Hardware. User can see and touch the hardware components.

4. Operating system

Operating system is system software which works as an interface between hardware and user and provides interactive platform to user.

5. Assembler

Assembler is system software which converts programs of assembly language to machine language.

6. Compiler

It translates programs of higher level language to machine language. It converts whole program at a time.

7. Interpreter

It translates programs of higher level language to machine language. It converts program line by line.



Computer Programming and Utilization (CPU) – 110003

B) Flowchart & Algorithm

1 What is Algorithm? What is Flowchart? Write down the advantages and disadvantages. Compare them.

Flowchart

Flowchart is a pictorial or graphical representation of a process. Each step in the process is represented by a different symbol and contains a short description of the process step. The flow chart symbols are linked together with arrows showing the process flow direction. This pictorial representation can give step-by-step solution of the given problem.

Advantages

- Easy to draw.
- Easy to understand logic.
- Easy to identify mistakes by non computer person.
- Easy to show branching and looping.

Disadvantages

- Time consuming.
- Difficult to modify.
- Very difficult to draw flowchart for big or complex problems.

Algorithm

An Algorithm is a finite sequence of well defined steps for solving a problem in systematic manner. It is written in the natural languages like English.

Advantages

- Easy to write.
- Human readable techniques to understand logic.
- Algorithms for big problems can be written with moderate efforts.

Disadvantages

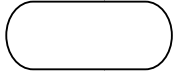
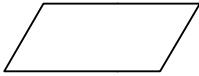

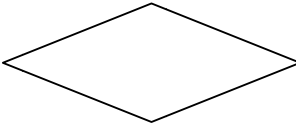

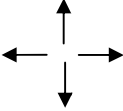
- Difficult to debug.
- Difficult to show branching and looping.
- Jumping (goto) makes it hard to trace some problems.

Comparison:

| Flowchart | Algorithm |
|--|--|
| It is a pictorial representation of a process. | It is step wise analysis of the work to be done. |
| Solution is shown in graphical format. | Solution is shown in non computer language like English. |
| Easy to understand as compared to algorithm. | It is somewhat difficult to understand. |
| Easy to show branching and looping. | Difficult to show branching and looping. |
| Flowchart for big problem is impractical | Algorithm can be written for any problem |



2 Explain various symbol used in flowchart.

| | |
|--|----------------------------------|
|  | Start / Stop |
|  | Input / Output (Read / Print) |
|  | Process |
|  | Decision making |
|  | Subroutine |
|  | Arrows |

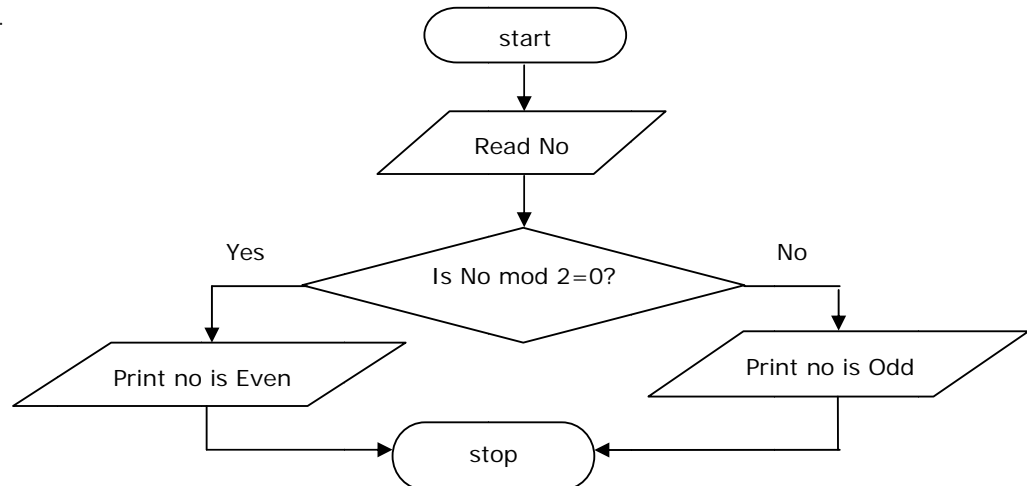
Write an algorithm and Draw a flowchart

3 To find whether given number is even or odd.

Algorithm:-

- Step 1 : Input no.
- Step 2 : If no mod 2=0, goto 4.
- Step 3 : Print given no is odd, goto 5.
- Step 4 : Print given no is even.
- Step 5 : Stop.

Flowchart:-



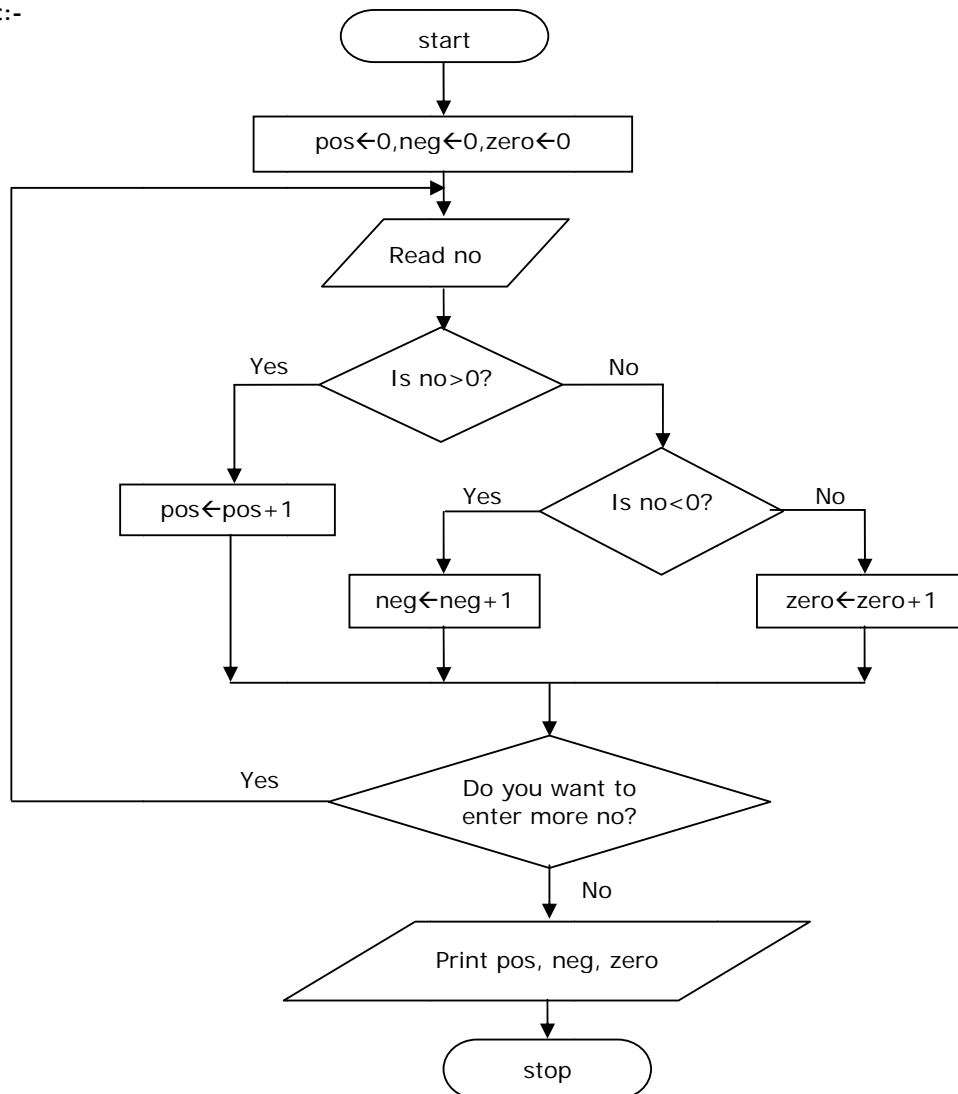


4 To enter number still user wants and at the end it should display count of total number of positive, negative and zero entered.

Algorithm:-

- Step 1 : Initialize $pos \leftarrow 0, neg \leftarrow 0, zero \leftarrow 0$
- Step 2 : Read no
- Step 3 : If $no > 0$, goto 6
- Step 4 : If $no < 0$, goto 7
- Step 5 : Increment zero by 1, $zero = zero + 1$. go to 8
- Step 6 : Increment pos by 1, $pos = pos + 1$. go to 8
- Step 7 : Increment neg by 1, $neg = neg + 1$.
- Step 8 : Print "Do you want to enter more number?"
- Step 9 : Read ans
- Step 10 : If ans = "y", goto 2
- Step 11 : Print pos, neg, zero.
- Step 12: Stop.

Flowchart:-



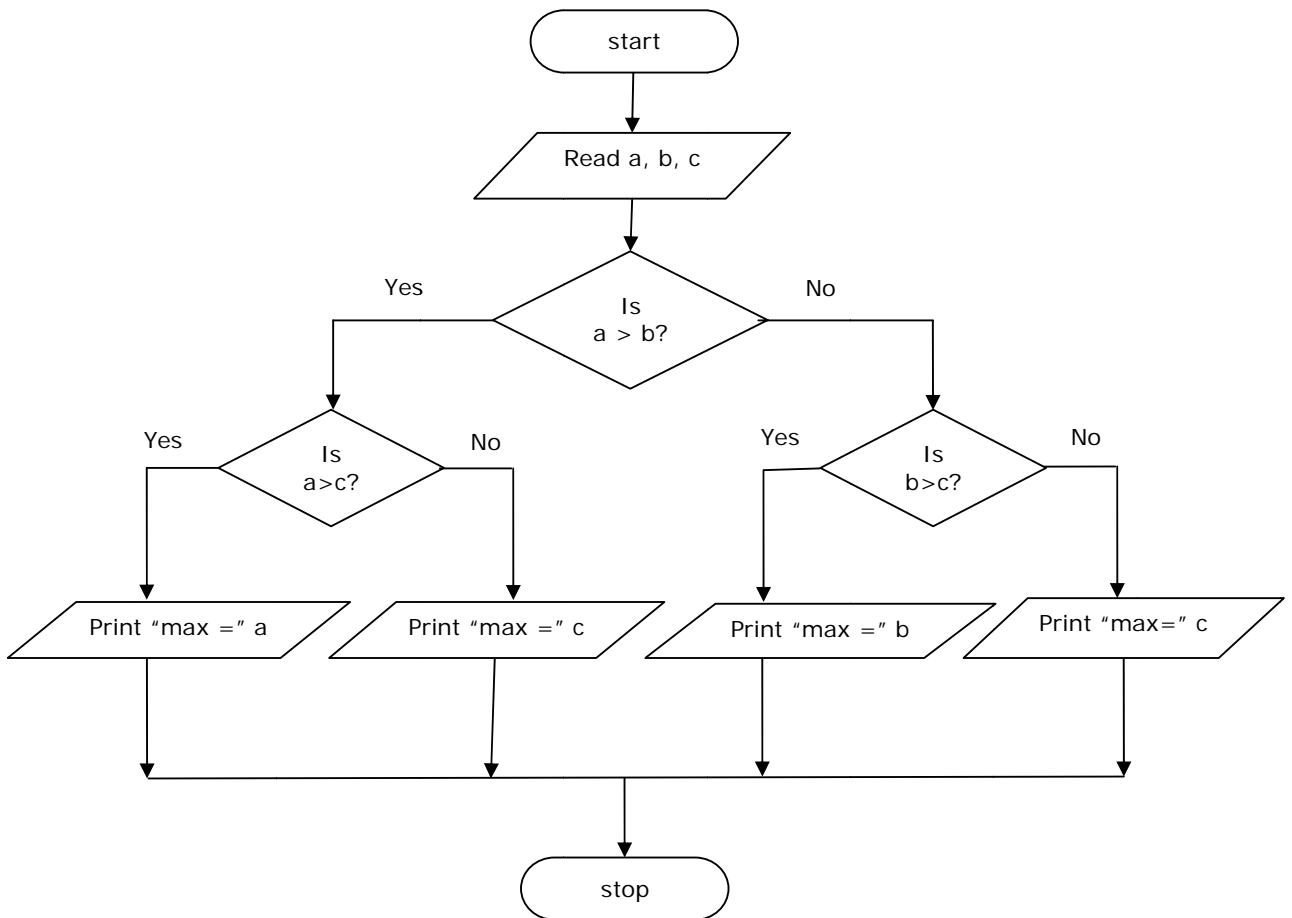


5 To print maximum number from a given 3 numbers.

Algorithm:-

- Step 1 : Read a ,b, c
- Step 2 : If a>b, goto 5
- Step 3 : If b>c, goto 8
- Step 4 : Print c is maximum goto 9
- Step 5 : If a>c, goto 7.
- Step 6 : Print c is maximum, goto 9
- Step 7 : Print a is maximum, goto 9
- Step 8 : Print b is maximum
- Step 9 : Stop.

Flowchart:-



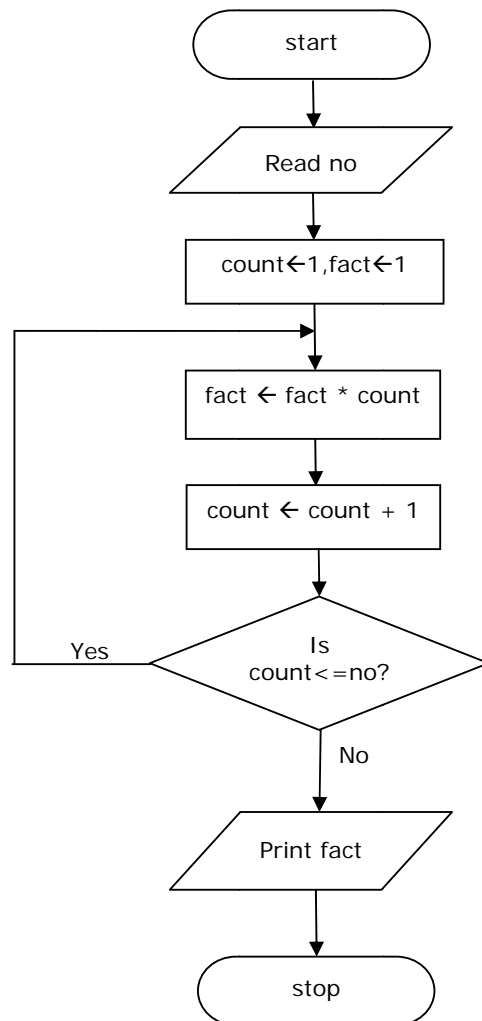


6 To find the factorial of a given number.

Algorithm:-

- Step 1 : Initialize $\text{count} \leftarrow 1, \text{fact} \leftarrow 1$
- Step 2 : Read no
- Step 3 : Calculate $\text{fact} \leftarrow \text{fact} * \text{count}$.
- Step 4 : Increment count by 1, $\text{count} \leftarrow \text{count} + 1$.
- Step 5 : If $\text{count} \leq \text{no}$, goto 3.
- Step 6 : Print fact.
- Step 7 : Stop.

Flowchart:-



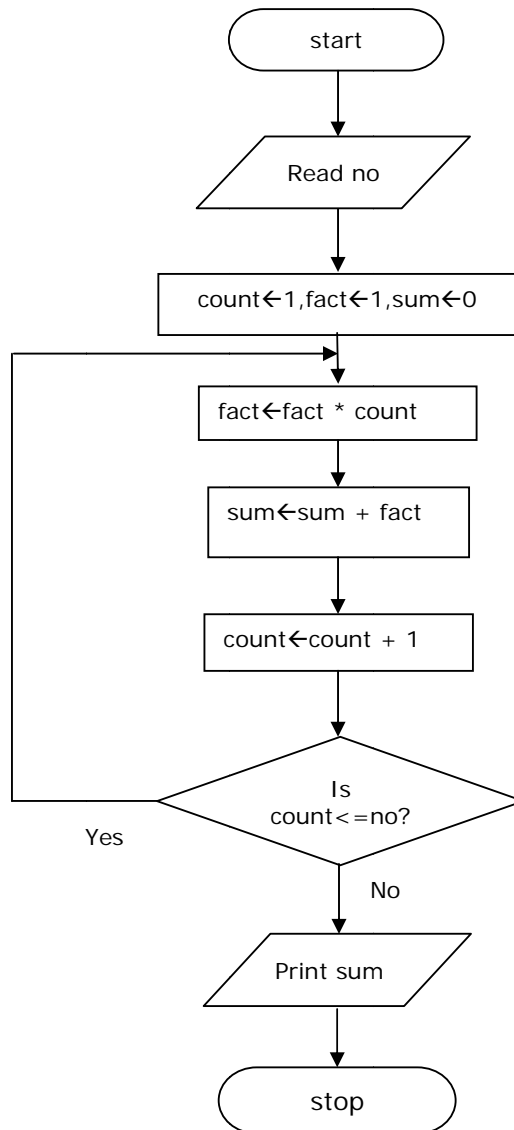


7 To solve series $1!+2!+3!+4!+\dots+n!$

Algorithm:

- Step 1 : Initialize count=1, fact=1, sum=0
- Step 2 : Read no
- Step 3 : Calculate fact=fact*count
- Step 4: Calculate sum=sum + fact.
- Step 5: Increment count by 1, count=count+1.
- Step 6: If count<=no, goto 3.
- Step 7: Print sum.
- Step 8: Stop.

Flowchart:





Computer Programming and Utilization (CPU) – 110003

C) C Fundamentals

1 Explain basic structure of 'C' program.

| Basic Structure of C Program is as below, | Example of C Program is as below, |
|---|--|
| Documentation Section | // This program is to find area of circle |
| Link Section | #include<stdio.h> |
| Definition Section | #define PI 3.14 |
| Global Declaration Section | int i; float areaofcircle(float); |
| main() { Declaration Part Executable Part } | void main() { //Declaration Part float r,area; //Executable Part scanf("%f",&r); area=areaofcircle(r); printf("Area of Circle is :- %f",area); } |
| Subprogram Section (User Defined Section) Function1() Function2() | float areaofcircle(float r) { return PI * r * r; } |

2 Explain printf() and scanf() function with syntax.

The stdio.h header file provides built in functions for reading (scanf) data from input devices (keyboard) and writing (printf) formatted data to output devices (monitor).

scanf():

- scanf() is a library function that reads data with specified format from standard input (keyboard).
- Syntax: `int scanf(const char *format, ...)`
- Example: `scanf("%d", &i);`
- First argument is the specification of format and other arguments are pointer variables to store data.
- The function returns the total number of items successfully read.
- scanf() stops when it exhausts its format string or when some input fails to match the control specification.

printf():

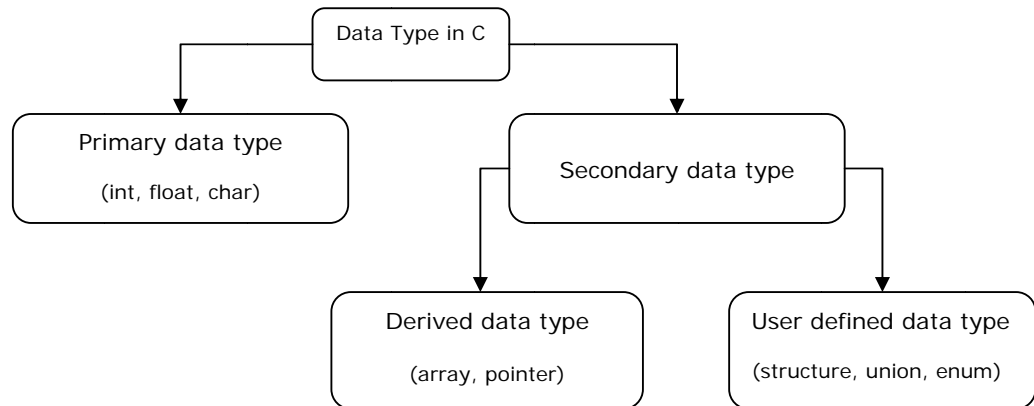
- printf() is a library function that prints formatted data to standard output, generally monitor.
- Syntax: `int printf(char * format, arg1, arg2, ...)`
- Example: `printf("Your marks are %d", mark);`



- The string format consists of two types of items - characters that will be printed on the screen, and format commands that define how the other arguments to printf() are displayed.
- printf() returns the number of characters printed.

3 Explain different data types available in C.

A data type is a classification of various types of data, as floating-point, integer, or string. C is rich in its data types to allow programmer to select appropriate type of data type.



1) Primary data types

Primary data types are built in data types. They are directly supported by machine. They are also known as fundamental data types.

a. int

int is integer which is whole number without fraction part. Its range is machine dependent values. C has 3 classes of integer storage namely short int, int and long int. All of these data types have signed and unsigned forms.

| | signed | | unsigned | |
|-----------|-------------|-----------------------------------|-------------|---------------------|
| | Size (bits) | Range | Size (bits) | Range |
| short int | 8 | -128 to 127 | 8 | 0 to 255 |
| int | 16 | -32768 to 32767 | 16 | 0 to 65535 |
| long int | 32 | -2,14,74,83,648 to 2,14,74,83,647 | 32 | 0 to 4,29,49,67,295 |

b. char

char data type can store single character of alphabet or digit or special symbol. Each character is assigned some integer value which is known as ASCII values.

| | signed | | Unsigned | |
|------|-------------|-------------|-------------|----------|
| | Size (bits) | Range | Size (bits) | Range |
| char | 8 | -128 to 127 | 8 | 0 to 255 |

c. float

float data type can store floating point number which represents a real number with decimal point and fractional part. When the accuracy of the floating point number is insufficient, we can



use the double to define the number. The double is same as float but with longer precision. To extend the precision further we can use long double which consumes 80 bits of memory space.

| | Size (bits) | Precision Digits | Range |
|------------|-------------|------------------|------------------------|
| float | 32 | 6 | 3.4E-38 to 3.4E+38 |
| double | 64 | 14 | 1.7E-308 to 1.7E+308 |
| long float | 80 | | 3.4E-4932 to 1.1E+4932 |

d. void

The void type has no value therefore we cannot declare it as variable as we did in case of int or float or char. The void data type is used to indicate that function is not returning anything.

2) Secondary data types

Secondary data types are not directly supported by the machine. It is combination of primary data types to handle real life data in more convenient way. It can be further divided in two categories,

a. Derived data type

Derived data type is extension of primary data type. It is built-in system and its structure cannot be changed. Examples: Array, Pointer, etc...

- i. Array: An array is a fixed-size sequenced collection of elements of the same data type.
- ii. Pointer: Pointer is a special variable which contains memory address of another variable

b. User defined data types

User defined data type can be created by programmer using combination of primary data type and/or derived data type. Use can design it as per special requirements

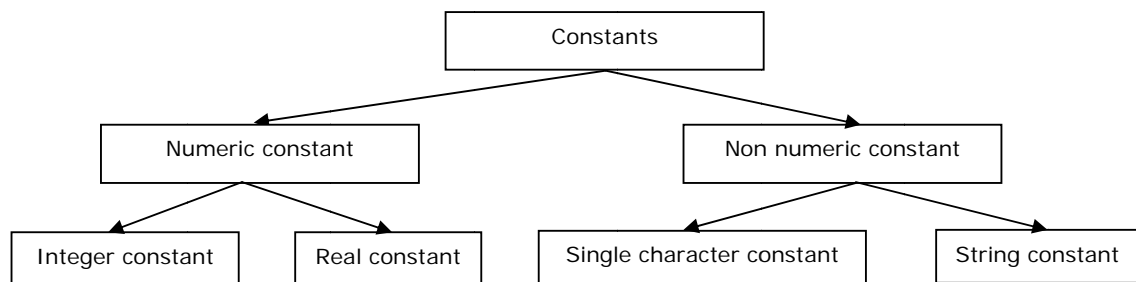
- i. structure: Structure is a collection of logically related data items of different data types grouped together and known by a single name.
- ii. union: Union is like a structure, except that each element shares the common memory.
- iii. enum: Enum is a user-defined type consisting of a set of named constants called enumerators. The enumerator names are usually identifiers that behave as constants in the language.

Example:

```
enum day {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
enum day week1stday ;
week1stday = Mon;
```

4 Explain types of constant in detail.

Constant is something whose value does not change throughout the program.





Integer constant:-

- Integer constant is a number without decimal point and fractional part
- There are three types of integers constant.
 - Decimal integer
Decimal integer consist of a set of digits, 0 to 9 having optional – or + sign. No other characters are allowed like space, commas, and non-digit charcters.
Ex: 123, -321, 0, +78
 - Octal integer
Octal integer consists of any combination of digits from the set 0 to 7. Octal numbers are always preceded by 0.
Ex: 037, 0, 0551
 - Hexadecimal integer
Hexadecimal integer consists of any combination of digits from the set 0 to 9 and A to F alphabets. It always starts with 0x or 0X. A represents 10, B represents 11... F represents 15.
Ex: 0X2A, 0x95, 0xA47C.

Real constant:-

- The number containing the fractional part is called real number. Ex: 0.0083, -0.75, +247.0, -0.75.
- A real number may also be expressed in exponential notation.
- The general form is: mantissa e exponent, ex: 215.65 can be written as 2.1565e2.
- In exponential form, e2 means multiply by 10².

Single character constant:-

- It contains single character enclosed within a pair of single quote mark.
- Ex: '5', 'A', ';', ' '

String constant:-

- A string constant is a sequence of characters enclosed within a double inverted comma.
- The characters may be letter, number, special character, blank space, etc...
- Ex: "DIET", "1988", "?A.B,! ", "5+3", etc...
- 'A' is character but "A" is string.

5 What is variable? Give the rules to define variable name.

A variable is a data name that is used to store a data value. A variable may take different values at different times during execution of the program.

Rules to define variable name:

1. It must consists of only alphabets (a to z & A to Z), digits (0 to 9) & underscore (_).
2. First character must be an alphabet or an underscore.
3. Only first 31 characters are significant.
4. Cannot use C keyword.

6 Define the following terms:

Token:

- Smallest unit of any language is known as token.



- The tokens are the basic building blocks which can be put together to construct programs.
- There are six classes of tokens in C: identifier, keywords, constants, string literals, operators and other separators

Identifier:

- An identifier is a sequence of letters and digits.
- The words which are defined by programmer in a program are known as identifier.
- Identifier refers to the name of variables, functions and arrays.

Keyword:

- Keywords are reserved words whose meaning is fixed and they are used for some special purpose by the compiler.
- They cannot be used by programmer for other purpose.

void - data type:

- void is a primary data type. void means nothing, no value.
- It is generally used to show that function is not returning anything.

7 Explain operators available in C

An operator is a symbol that tells the compiler to perform certain mathematical or logical operation. C has rich set of operators as below,

1. Arithmetic Operators

Arithmetic operators are used for mathematical calculation. C supports following arithmetic operators

| | | |
|---|----------------------------|---|
| + | Addition or unary plus | a+ b (addition), +7 (unary plus) |
| - | Subtraction or unary minus | a – b (subtraction), -8 (unary minus) |
| * | Multiplication | a * b |
| / | Division | a / b |
| % | Modulo division | a % b (this operator can be used with only integer data type) |

2. Relational Operators

Relational operators are used to compare two numbers and taking decisions based on their relation. Relational expressions are used in decision statements such as if, for, while, etc...

| | |
|----|--------------------------|
| < | less than |
| <= | less than or equal to |
| > | greater than |
| >= | greater than or equal to |
| == | is equal to |
| != | is not equal to |

3. Logical Operators

Logical operators are used to test more than one condition and make decisions

| | |
|----|--|
| && | logical AND (Both non zero then true, either is zero then false) |
| | logical OR (Both zero then false, either is non zero then true) |
| ! | logical NOT (non zero then false, zero then true) |

4. Assignment Operators

Assignment operators are used to assign the result of an expression to a variable. C also supports shorthand assignment operators which simplify operation with assignment



| | |
|----|--|
| = | Assigns value of right side to left side |
| += | a += 1 is same as a = a + 1 |
| -= | a -= 1 is same as a = a - 1 |
| *= | a *= 1 is same as a = a * 1 |
| /= | a /= 1 is same as a = a / 1 |
| %= | a %= 1 is same as a = a % 1 |

5. Increment and Decrement Operators

These are special operators in C which are generally not found in other languages.

| | |
|----|--|
| ++ | <p>Increments value by 1.</p> <p>a++ is postfix, the expression is evaluated first and then the value is incremented. Ex. a=10; b=a++; after this statement, a= 11, b = 10.</p> <p>++a is prefix, the value is incremented first and then the expression is evaluated. Ex. a=10; b=++a; after this statement, a= 11, b = 11.</p> |
| -- | <p>Decrements value by 1.</p> <p>a-- is postfix, the expression is evaluated first and then the value is decremented. Ex. a=10; b=a--; after this statement, a= 9, b = 10.</p> <p>--a is prefix, the value is decremented first and then the expression is evaluated. Ex. a=10; b=--a; after this statement, a= 9, b = 9.</p> |

6. Conditional Operator

A ternary operator is known as Conditional Operator.

exp1?exp2:exp3 if exp1 is true then execute exp2 otherwise exp3

Ex: x = (a>b)?a:b; which is same as

```
if(a>b)
```

```
    x=a;
```

```
else
```

```
    x=b;
```

7. Bitwise Operators

Bitwise operators are used to perform operation bit by bit. Bitwise operators may not be applied to float or double.

| | |
|----|--|
| & | bitwise AND |
| | bitwise OR |
| ^ | bitwise exclusive OR |
| << | shift left (shift left means multiply by 2) |
| >> | shift right (shift right means divide by 2) |

8. Special Operators

| | |
|--------|--|
| & | Address operator, it is used to determine address of the variable. |
| * | Pointer operator, it is used to declare pointer variable and to get value from it. |
| , | Comma operator. It is used to link the related expressions together. |
| sizeof | It returns the number of bytes the operand occupies. |
| . | member selection operator, used in structure. |
| -> | member selection operator, used in pointer to structure. |

8 Explain conditional operator (ternary operator) with example.

- C provides special conditional operator (? :) to evaluate conditional expression in single line.

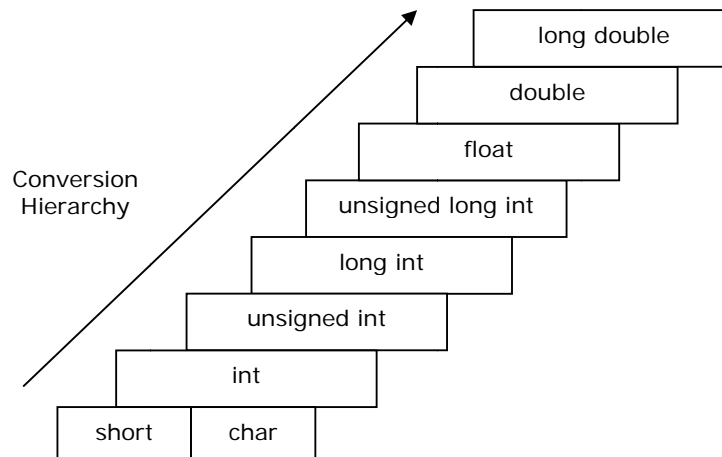
- It is also known as ternary operator because this is the only operator in C which requires three operands or expressions.
- Syntax: `expr1? expr2 : expr3`
- First of all `expr1` is evaluated, if it is nonzero (true) then the expression `expr2` is evaluated otherwise `expr3` is evaluated. Only one of `expr2` and `expr3` is evaluated.
- Example: `c = a>b? a : b; // If a is greater than b then a is assigned to c otherwise b.`

9 What is type conversion? Explain implicit and explicit type conversion with example.

- When an operator has operands of different types, they are converted to a common type, this is known as type casting or type conversion.
- Typecasting is making a variable of one data type to act like another data type such as an int to float.

Implicit Type Casting:

- C automatically converts any intermediate values to the proper type so that the expression can be evaluated without losing any significance.
- This automatic conversion is known as implicit type conversion.
- The lower type is automatically converted to the higher type before the operation proceeds. The result is higher type.
- Example:
 - If one operand is int and other is float then int will be converted to float.
 - If one operand is float and other is long double then float will be converted to long double.
- Thus conversion happens from low data type to high data type so that information is not lost. The conversion hierarchy is shown below.



Explicit Type Casting:

- Sometimes we want to force a type conversion in a way that is different from automatic conversion. The process of such a local conversion or casting is known as explicit casting.
- The general form of cast is: `(type-name) expression.`
- Where type-name is one of the standard C data type. The expression may be constant, variable, or an expression.

Example:

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



```

void main()
{
    int sum=47, n=10;
    float avg;
    avg=sum / n; // implicit type casting
    printf("Result of Implicit Type Casting: %f", avg); // This will print 4
    avg=(float)sum / (float)n; // explicit type casting
    printf("Result of Explicit Type Casting: %f", avg); // This will print 4.7
}

```

10 Explain operator precedence and associativity.

- Precedence of an operator is its priority in an expression for evaluation.
- Operator precedence is why the expression $5 + 3 * 2$ is calculated as $5 + (3 * 2)$, giving 11, and not as $(5 + 3) * 2$, giving 16.
- We say that the multiplication operator ($*$) has higher "precedence" or "priority" than the addition operator ($+$), so the multiplication must be performed first.
- Associativity is the left-to-right or right-to-left order for grouping operands to operators that have the same precedence.
- Operator associativity is why the expression $8 - 3 - 2$ is calculated as $(8 - 3) - 2$, giving 3, and not as $8 - (3 - 2)$, giving 7.
- We say that the subtraction operator ($-$) is "left associative", so the left subtraction must be performed first. When we can't decide by operator precedence alone in which order to calculate an expression, we must use associativity.
- Following table provides a complete list of operator, their precedence level, and their rule of association. Rank 1 indicates highest precedence level and 15 is the lowest.

| Precedence | Associativity | Operator | Description |
|------------|---------------|--|---|
| 1 | Left to Right | () [] | Function Call Array Element Reference |
| 2 | Right to Left | +, - ++, -- ! ~ * & sizeof (type) | Unary Plus, Unary Minus Increment, Decrement Logical Negation Ones Complement Pointer Reference (Indirection) Address Size of an object Type Cast (Conversion) |
| 3 | Left to Right | * / % | Multiplication Division Modulus |
| 4 | Left to Right | + - | Addition Subtraction |
| 5 | Left to Right | << >> | Left Shift Right Shift |
| 6 | Left to Right | < | Less than |



| | | | |
|----|---------------|---|---|
| | | <= > >= | Less than or equal to Greater than Greater than or equal to |
| 7 | Left to Right | == != | Equality Inequality |
| 8 | Left to Right | & | Bitwise AND |
| 9 | Left to Right | ^ | Bitwise XOR |
| 10 | Left to Right | | Bitwise OR |
| 11 | Left to Right | && | Logical AND |
| 12 | Left to Right | | Logical OR |
| 13 | Right to Left | ?: | Conditional Expression |
| 14 | Right to Left | ==, *=, /=, %=, +=, -=, &=, ^=, =, <<=, >>= | Assignment Operators |
| 15 | Left to Right | , | Comma Operator |

11 Explain backslash and trigraph characters.

Backslash character:

Back slash constants are a special type of character constant which are consists of two characters. This is known as escape sequence. Escape sequence starts with backslash '\ ' character.

| Escape sequence | Meaning |
|-----------------|--|
| '\0' | End of string – NULL. |
| '\n' | End of line – takes the control to next line. |
| '\r' | Carriage return – takes the control to the next paragraph. |
| '\f' | Form feed- takes control to the next page. |
| '\t' | Horizontal tab |
| '\b' | Back space |
| '\\' | Prints backslash character \ |
| '\a' | Alert – provide edible alert. |

Trigraph character:-

C introduces concept of "trigraph" sequence to provide a way to enter certain characters which are not available in some keyboard of non English language. Each trigraph character consists of a three characters.

| Trigraph sequence | Translation |
|-------------------|-----------------|
| ??= | # number sign |
| ??(| [left bracket |
| ??) |] right bracket |
| ??< | { left bracket |
| ??> | } right bracket |
| ??! | vertical bar |
| ??/ | \ backslash |



Computer Programming and Utilization (CPU) – 110003

D) Decision Control Structure (if, if else, if else if, switch, ...)

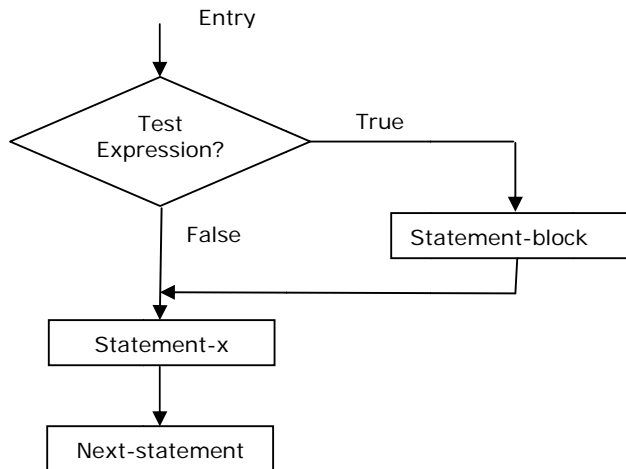
1 Explain *if* with example and draw flowchart.

- *if* is used to control the flow of execution of statements.
- It is two way decision making statement.
- It evaluates expression first and based on its result, the control is transferred to the particular statement.
- The general form of simple *if* statement is

```
if (test expression)
{
    statement-block
}
statement-x
```

- 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-x.

Flowchart:



Example:

```
#include<stdio.h>
void main()
{
    int no;
    printf("Enter the number:");
    scanf("%d", &no);
    if(no%2==0)
        printf("no is even");
    if(no%2!=0)
        printf("no is odd");
}
```



2 Explain if...else... with example and draw flowchart.

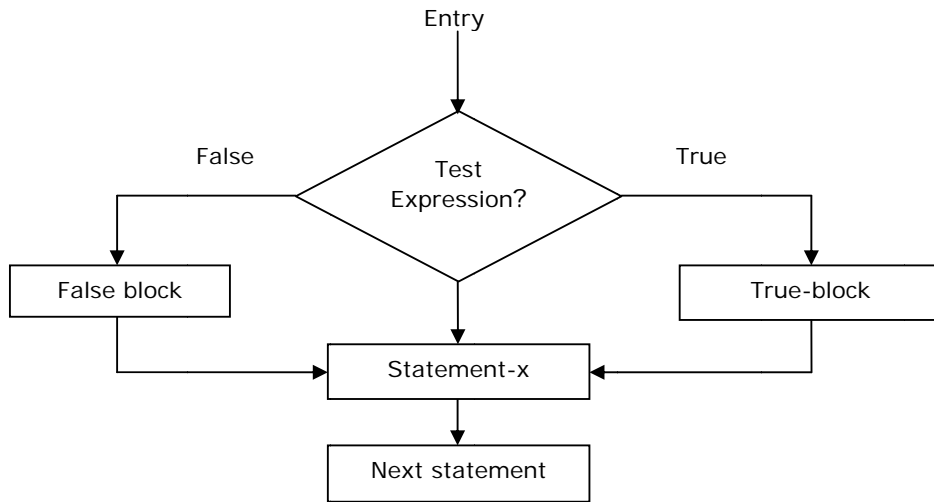
- *If* supports statements only for true part
- *If...else* supports statements for true part and false part.

The general format is given below.

```
if(test-expression)
{
    True-block statement.
}
else
{
    False-block statement
}
statement-x;
```

- If test-expression is true, then true-block statement is executed otherwise false-block statement is executed.
- Every time, either true block or false block will be executed.
- In both the cases, statement-x will be executed immediately after that block.

Flowchart:



Example: Check whether the given number is even or odd.

```
#include<stdio.h>
void main()
{
    int no;
    printf("Enter the number:");
    scanf("%d", &no);
    if(no%2==0)
        printf("the number is even");
    else
        printf("the number is odd");
}
```



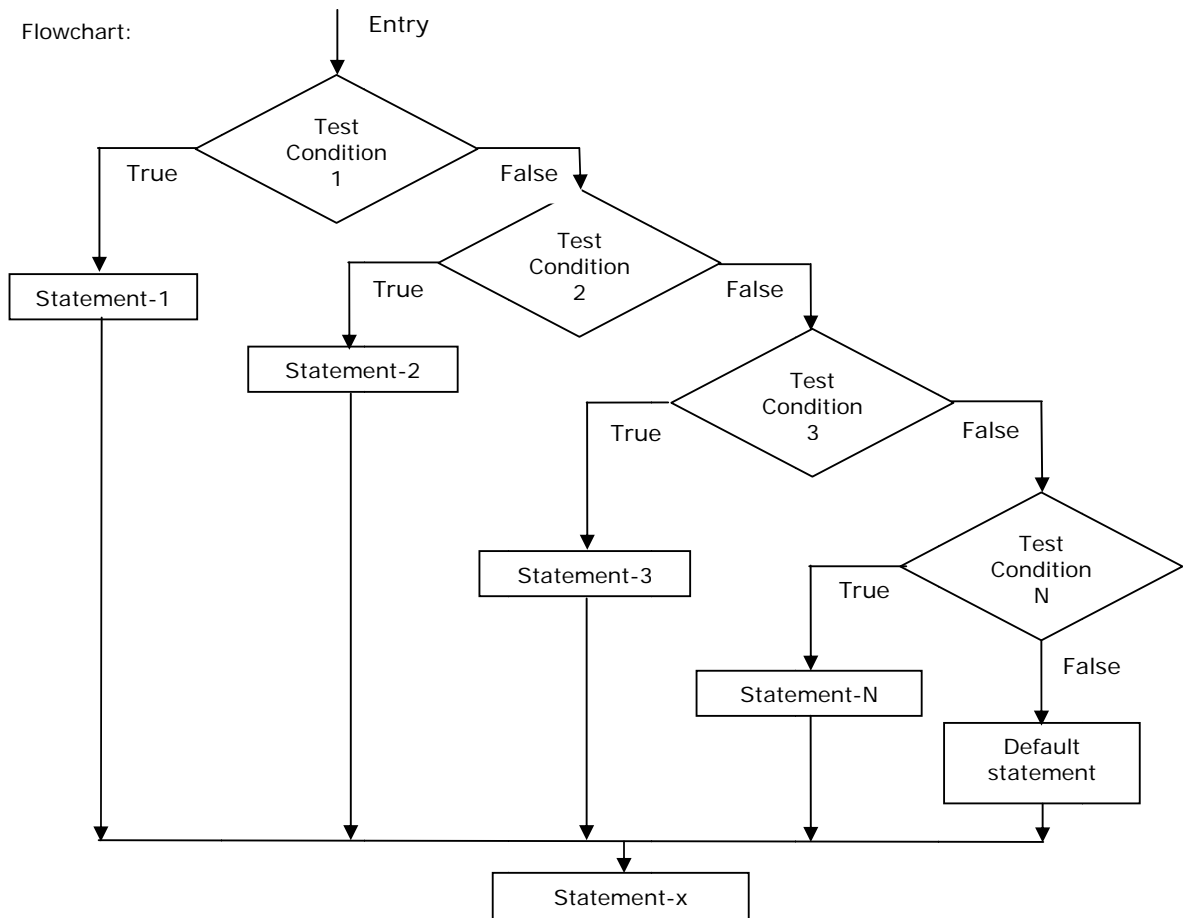
3 Explain if...else...if ladder with example and draw flowchart.

When multipath decisions are involved, we may use *if...else...if* ladder, which takes the following general form.

```
if(condition-1)
    statement-1;
else if(condition-2)
    statement-2;
else if(condition-N)
    statement-N;
else
    default-statement;
statement-x
```

- First condition-1 is checked and if it is true, then statement-1 will be executed and control goes to statement-x.
- If condition-1 is false, then condition-2 is checked and if it is true then statement-2 will be executed and control goes to statement-x.
- If condition-2 is false, then condition-3 is checked and process repeats
- This process is repeated until either it finds one of the conditions is true or all the conditions are false. If all the conditions are false, then default-statement will be executed.
- Else part is optional in if...else...if ladder.

Flowchart:





Example: Find the class of student based on average marks.

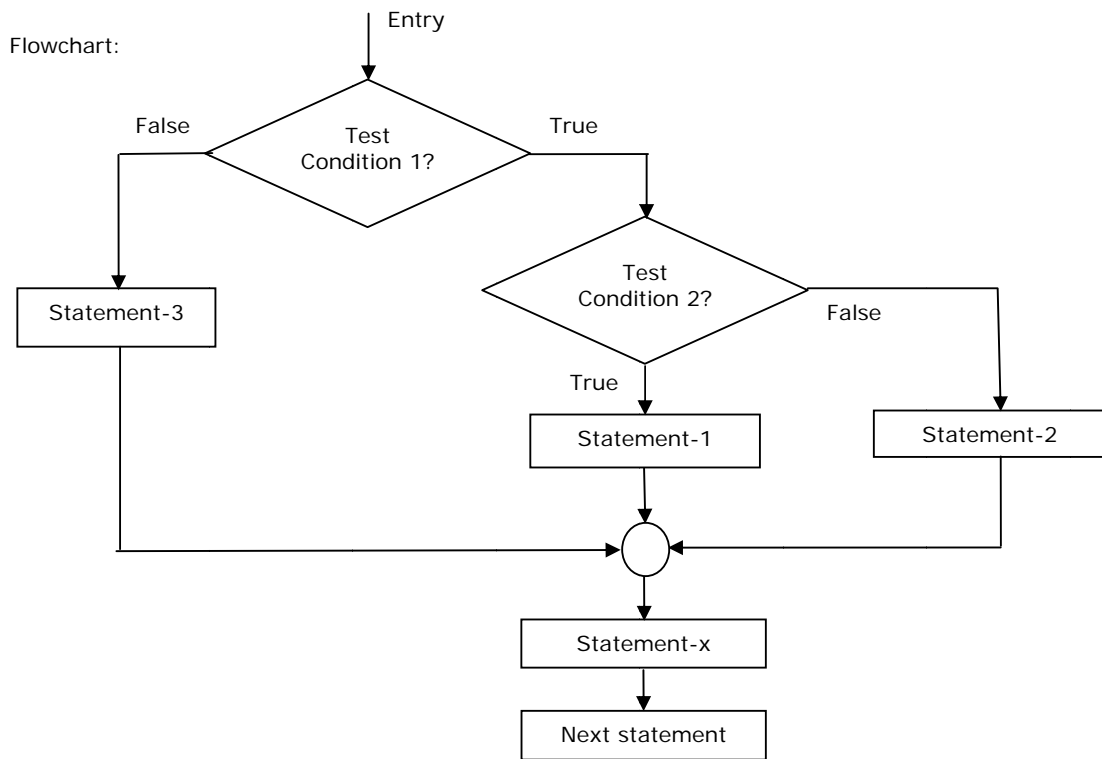
```
#include<stdio.h>
void main()
{
    int m1,m2,m3,m4,m5;
    float avg;
    printf("Enter m1,m2,m3,m4,m5:");
    scanf("%d%d%d%d%d", &m1, &m2, &m3, &m4, &m5);
    avg=(m1+m2+m3+m4+m5)/5;
    if(avg>=70)
        printf("Distinct");
    else if(avg>=60)
        printf("First class");
    else if(avg>=50)
        printf("Second class");
    else if(avg>=40)
        printf("Pass class");
    else
        printf("fail");
}
```

4 Explain nested if with example and draw flowchart.

When a series of decisions are involved, we may have to use more than one if...else statement in nested form as shown below.

```
if(test-condition-1)
{
    if(test-condition-2)
    {
        Statement-1;
    }
    else
    {
        Statement-2;
    }
}
else
{
    Statement-3;
}
Statement-x
```

- If *test-condition-1* is true then *test-condition-2* is evaluated. If it is true then *Statement-1* will be executed, if it is false then *Statement-2* will be executed.
- If *test-condition-1* is false then *Statement-3* will be executed.



Example: Find maximum number from given three numbers.

```
#include<stdio.h>
void main()
{
    int a,b,c;
    printf("Enter value of a, b, c:");
    scanf("%d%d%d", &a, &b, &c);
    if(a>b)
    {
        if(a>c)
            printf("a is max");
        else
            printf("c is max");
    }
    else
    {
        if(b>c)
            printf("b is max");
        else
            printf("c is max");
    }
}
```



5 Explain switch-case statement with example.

- The switch statement is a multi-way decision making.
- It tests whether an expression matches any one of the constant values or not.
- General form of switch-case is as below,

```
switch(expression)
{
    case const-expr 1:  statement 1;
                       break;
    case const-expr 2:  statement 2;
                       break;
    case const-expr 3:  statement 3;
                       break;
    default:
        statements
}
```

- *expression* in switch should be integer or character expression. Float or any other data type is not allowed.
- Each case is labeled by one or more integer-valued constant.
- If a case matches the expression value then execution starts at that case.
- Value of all case expressions must be different.
- If none of the cases are matched then default case is executed. default case is optional.
- The break statement causes an immediate exit from the switch.

Example:

```
switch (grade)
{
    case 1:
        printf("Fall (F)\n");
        break;
    case 2:
        printf("Bad (D)\n");
        break;
    case 3:
        printf("Good (C)\n");
        break;
    case 4:
        printf("Very Good (B)\n");
        break;
    case 5:
        printf("Excellent (A)\n");
        break;
    default:
        printf("You have inputted false grade\n");
        break;          // break isn't necessary here
}
```



Rules for switch statement

- The switch expression must be integral type. Float or other data types are not allowed.
- Case labels must be constant or constant expression.
- Case labels must be unique.
- Case labels must end with colons.
- The break statement is optional.
- The default case statement is optional.
- Nesting of switch statement is allowed.

6 State difference between if-else and switch-case

| if-else | switch |
|---|--|
| <i>If</i> statement is used to select among two alternatives. | The <i>switch</i> statement is used to select among multiple alternatives. |
| <i>If</i> can have values based on constraints. | <i>Switch</i> can have values based on user choice. |
| <i>If</i> implements Linear search. | <i>Switch</i> implements Binary search. |
| Float, double, char, int and other data types can be used in <i>if</i> condition. | Only int and char data types can be used in <i>switch</i> block. |

Write a C Program to....

1 Find out whether the given number is odd or even.

```
#include<stdio.h>
void main()
{
    int no;
    printf("enter no:");
    scanf("%d",&no);
    if(no%2==0)
        printf("no is even");
    else
        printf("no is odd");
}
```

2 Find maximum number from given 3 numbers.

```
#include<stdio.h>
void main()
{
    int a,b,c;
    printf("enter a,b,c:");
    scanf("%d%d%d",&a,&b,&c);
```



```
    if(a>b&& a>c)
        printf("a is maximum");
    else if(b>c)
        printf("b is maximum");
    else
        printf("c is maximum");
}
```

3 which asks day number and prints corresponding day name using switch-case.

```
#include<stdio.h>
void main()
{
    int day;
    printf("enter day number (1-7) \n");
    scanf("%d",&day);
    switch(day)
    {
        case 1:
            printf("sunday");
            break;
        case 2:
            printf("monday");
            break;
        case 3:
            printf("tuesday");
            break;
        case 4:
            printf("wednesday");
            break;
        case 5:
            printf("thursday");
            break;
        case 6:
            printf("friday");
            break;
        case 7:
            printf("saturday");
            break;
        default:
            printf("wrong input");
    }
}
```



Computer Programming and Utilization (CPU) – 110003

E) Loop Control Structure (for, while, do...while)

1 Explain loops available in C with example

- Loops are used to repeat execution of a block of code.
- During looping, a set of statements are executed until some condition for termination is encountered.

Generally, looping process would include the following four steps

- 1) Initialization of a counter
- 2) Test for a termination condition
- 3) Loop body statements
- 4) Increment the counter

C supports three types of looping

while loop

- The simplest of all looping structure is while statement.
- The general format of the while statement is:

```
Initialization;
while (test condition)
{
    body of the loop ;
    increment or decrement;
}
```
- Test condition is evaluated and if the condition is true then the body of the loop is executed.
- After the execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again.
- This process is repeated till the test condition is true. When it becomes false, the control is transferred out of the loop.
- On exit, the program continues with the statements immediately after the body of the loop.
- While loop is also known as entry control loop because first control-statement is executed and if it is true then only body of the loop will be executed.

Example: To print first 10 positive integer numbers

```
void main()
{
    int i;
    i = 1;                \\ initialization of i
    while(i <= 10)       \\ condition checking
    {
        printf("\t%d",i); \\ statement execution
        i++;             \\ increment of control variable
    }
}
```



do...while loop

- In contrast to while loop, the body of the do...while loop is executed first and then the loop condition is checked.
- The body of the loop is executed at least once because do...while loop tests condition at the bottom of the loop after executing the body.
- do...while loop is also known as exit control loop because first body statements are executed and then control-statement is executed, thus condition checking happens at exit point.
- The general format of the do...while statement is:

```
initialization;
do
{
    statement;
    increment or decrement;
}
while(test-condition);
```

Example: To print first 10 positive integer numbers

```
void main()
{
    int i;
    i = 1;           \\ initialization of i
    do
    {
        printf("\t%d",i);  \\ statement execution
        i++;              \\ increment of control variable
    } while(i <= 10);    \\ condition checking
}
```

for Loop

- *for* loop provides a more concise loop control structure.
- The general form of the for loop is:
for (initialization; test condition; increment)
{
 body of the loop;
}
- When the control enters for loop, the variables used in for loop is initialized with the starting value such as i=0, count=0. Initialization part will be executed exactly one time.
- Then it is checked with the given test condition. If the given condition is satisfied, the control enters into the body of the loop. If condition is not satisfied then it will exit the loop.
- After the completion of the execution of the loop, the control is transferred back to the increment part of the loop. The control variable is incremented using an assignment statement such as i++
- If new value of the control variable satisfies the loop condition then the body of the loop is again executed. This process goes on till the control variable fails to satisfy the condition.
- For loop is also entry control loop because first control-statement is executed and if it is true then only body of the loop will be executed.



Example: // The following is an example that finds the sum of the first ten positive integer numbers

```

void main()
{
    int i;           //declare variable
    int sum=0;
    for(i=1; i <= 10; i++) // for loop
    {
        sum = sum + i ; // add the value of i and store it to sum
    }
    printf("%d", sum);
}

```

- We can include multiple expressions in any of the fields of for loop provided that we separate such expressions by commas. For example: for(i = 0; j = 100; i < 10 && j>50; i++, j=j-10)

| <u>for Loop</u> | <u>while Loop</u> | <u>do..while Loop</u> |
|---|--|--|
| <pre> for(i=1; i <= 10; i++) { sum = sum + i ; } </pre> | <pre> i=1; while(i<=10) { sum = sum + i; i ++; } </pre> | <pre> i=1; do { sum = sum + i; i ++; } while(i<=10); </pre> |

2 State the difference between entry control loop and exit control loop.

| Entry control loop | Exit control loop |
|---|---|
| Entry control loop checks condition first and then body of the loop will be executed. | Exit control loop first executes body of the loop and checks condition at last. |
| Body of loop may or may not be executed at all. | Body of loop will be executed at least once because condition is checked at last. |
| for, while are example of entry control loop. | Do...while is example of exit control loop. |

3 Explain break, continue, goto with example.

- Sometimes it is required to quit the loop as soon as certain condition occurs.
- For example, consider searching a particular number in a set of 100 numbers, as soon as the search number is found it is desirable to terminate the loop.
- A *break* statement is used to jump out of a loop.
- A *break* statement provides an early exit from for, while, do...while and switch constructs.
- A *break* causes the innermost enclosing loop or switch to be exited immediately.

Example : Read and sum numbers till -1 is entered

```

void main()
{
    int i, num=0;

```



```
float sum=0,average;
printf("Input the marks, -1 to end\n");
while(1)
{
    scanf("%d",&i);
    if(i==-1)
        break;
    sum+=i;
}
printf("%d", sum);
}
```

continue;

- The *continue* statement can be used to skip the rest of the body of an iterative loop.
- The *continue* statement tells the compiler, "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".

Example: Find sum of 5 positive integers. If a negative number is entered then skip it.

```
void main()
{
    int i=1, num, sum=0;
    for (i = 0; i < 5; i++)
    {
        scanf("%d", &num);
        if(num < 0)
            continue;    // starts with the beginning of the loop
        sum+=num;
    }
    printf("The sum of positive numbers entered = %d",sum);
}
```

goto:

- The *goto* statement is a jump statement which jumps from one point to another point within a function.
- The *goto* statement is marked by label statement. Label statement can be used anywhere in the function above or below the *goto* statement.
- Generally *goto* should be avoided because its usage results in less efficient code, complicated logic and difficult to debug.

Example: Following program prints 10,9,8,7,6,5,4,3,2,1

```
void main ()
{
    int n=10;
loop:
    printf("%d,",n);
    n--;
    if (n>0)
        goto loop;
}
```

Please refer book for flowchart or diagram of break, continue, goto. Page No: 167



Computer Programming and Utilization (CPU) – 110003

F) Array and String

1 What is an array? Explain with Example. What are the advantages of using an array?

- An array is a fixed-size sequenced collection of elements of the same data type.
- An array is derived data type.
- The individual element of an array is referred by their index or subscript value.
- The subscript for an array always begins with 0.

Syntax : data_type array_name[size];

Example : int marks[5];

- The *data_type* specifies the type of the elements that can be stored in an array, like int, float or char.
- The *size* indicates the maximum number of elements that can be stored inside the array.
- In the example, data type of an array is int and maximum elements that can be stored in an array are 5.

Advantages:

- You can use one name to store many values with different indexes.
- An array is very useful when you are working with sequences of the same kind of data.
- An array makes program easier to read, write and debug.

Example:

```
#include<stdio.h>
void main()
{
    int a[5] = {5,12,20,54,68}, i;
    for(i=0;i<5;i++)
    {
        printf("%d", a[i]);
    }
}
```

Types of an array:

- 1) Single dimensional array
- 2) Two dimensional array
- 3) Multi dimensional array



2 Explain initialization and working of single and multi-dimensional array with example.

Single Dimensional Array

An array using only one subscript to represent the list of elements is called single dimensional array.

Syntax : `data_type array_name[size];`

Example : `int marks[5];`

- An individual array element can be used anywhere like a normal variable with a statement such as `g = marks [60];`
More generally if *i* is declared to be an integer variable, then the statement `g=marks[i];` will take the value contained at *i*th position in an array and assigns it to *g*.
- We can store value into array element by specifying the array element on the left hand side of the equals sign like `marks[60]=95;` The value 95 is stored at 60th position in an array.
- The ability to represent a collection of related data items by a single array enables us to develop concise and efficient programs.
- For example we can very easily sequence through the elements in the array by varying the value of the variable that is used as a subscript into the array.

```
for(i=0; i<66; i++);
```

```
    sum = sum + marks[i];
```

Above for loop will sequence through the first 66 elements of the marks array (elements 0 to 65) and will add the values of each marks into sum. When for loop is finished, the variable sum will then contain the total of first 66 values of the marks.

- The declaration `int values[5];` would reserve enough space for an array called values that could hold up to 5 integers. Refer to the below given picture to conceptualize the reserved storage space.

| | |
|-----------|--|
| values[0] | |
| values[1] | |
| values[2] | |
| values[3] | |
| values[4] | |

Initialization of Single Dimensional array:

The general form of initialization of array is:

```
data_type array_name[size]={list of values};
```

There are three ways to initialize single dimensional array,

1. `int number[3]={1, 5, 2};`
will initialize 0th element of an array to 1, 1st element to 5 and 2nd element to 2.
2. `int number[5] = {1, 7};`
will initialize 0th element of an array to 1, 1st element to 7 and rest all elements will be initialized to 0.
3. `int number[] = {1, 5, 6};`
first of all array size will be fixed to 3 then it will initialize 0th element to 1, 1st element to 5 and 2nd element to 6



Two dimensional arrays:

- Two dimensional arrays are also called table or matrix.
- Two dimensional arrays have two subscripts.
- First subscript denotes the number of rows and second subscript denotes the number of columns.

Syntax : data_type array_name[row_size][column_size];

Example : int marks[10][20];

- Here m is declared as a matrix having 10 rows (numbered from 0 to 9) and 20 columns (numbered 0 through 19). The first element of the matrix is m[0][0] and the last row last column is m[9][19]
- A two dimensional array marks[4][3] is shown below. The first element is given by marks[0][0] contains 35.5 & second element is marks[0][1] and contains 40.5 and so on.

| | | |
|----------------------|----------------------|----------------------|
| marks [0][0] 35.5 | marks [0][1] 40.5 | marks [0][2] 45.5 |
| marks [1][0] 66.5 | marks [1][1] 55.5 | marks [1][2] 60.5 |
| marks [2][0] 85.5 | marks [2][1] 78.5 | marks [2][2] 65.3 |
| marks [3][0] 25.6 | marks [3][1] 35.2 | marks [3][2] 76.2 |

Initialization of two dimensional array:

1. `int table[2][3] = {1,2,3,4,5,6};`
will initialize 1st row 1st column element to 1, 1st row 2nd column to 2, 1st row 3rd column to 3, 2nd row 1st column to 4, 2nd row 2nd column to 5, 2nd row 3rd column to 6 and so on.
2. `int table[2][3] = {{1,2,3},{4,5,6}};`
here, 1st group is for 1st row and 2nd group is for 2nd row. So 1st row 1st column element is 1, 2nd row 1st column element is 4, 2nd row 3rd column element is 6 so on.
3. `int table[2][3] = {{1,2},{4}}`
initializes as above but missing elements will be initialized by 0.

3 Explain various string handling operations available in 'C' with example.

C has several inbuilt functions to operate on string. These functions are known as string handling functions. For Example: `char s1[]="Their", s2[]="There";`

| Function | Meaning |
|----------------------------|--|
| <code>strlen(s1)</code> | Returns length of the string. l = <code>strlen(s1)</code> ; it returns 5 |
| <code>strcmp(s1,s2)</code> | Compares two strings. It returns negative value if <code>s1 < s2</code> , positive if <code>s1 > s2</code> and zero if <code>s1 = s2</code> . |



| | |
|------------------|--|
| | <pre>printf("%d", strcmp(s1,s2));</pre> <p>Output : -9</p> |
| strcpy(s1,s2) | <p>Copies 2nd string to 1st string.</p> <p>strcpy(s1,s2) copies the string s2 in to string s1 so s1 is now "There".</p> <p>s2 remains unchanged.</p> |
| strcat(s1,s2) | <p>Appends 2nd string at the end of 1st string.</p> <p>strcat(s1,s2); a copy of string s2 is appended at the end of string s1. Now s1 becomes "TheirThere"</p> |
| strchr(s1,c) | <p>Returns a pointer to the first occurrence of a given character in the string s1.</p> <pre>printf("%s",strchr(s1,'i'));</pre> <p>Output : ir</p> |
| strstr(s1,s2) | <p>Returns a pointer to the first occurrence of a given string s2 in string s1.</p> <pre>printf("%s",strstr(s1,"he"));</pre> <p>Output : heir</p> |
| strrev(s1) | <p>Reverses given string.</p> <p>strrev(s1); makes string s1 to "riehT"</p> |
| strlwr(s1) | <p>Converts string s1 to lower case.</p> <pre>printf("%s", strlwr(s1));</pre> <p>Output : their</p> |
| strupr(s1) | <p>Converts string s1 to upper case.</p> <pre>printf("%s",strupr(s1));</pre> <p>Output : THEIR</p> |
| strncpy(s1,s2,n) | <p>Copies first n character of string s2 to string s1</p> <pre>s1=""; s2="There";</pre> <pre>strncpy(s1,s2,2);</pre> <pre>printf("%s",s1);</pre> <p>Output : Th</p> |
| strncat(s1,s2,n) | <p>Appends first n character of string s2 at the end of string s1.</p> <pre>strncat(s1,s2,2);</pre> <pre>printf("%s", s1);</pre> <p>Output : TheirTh</p> |
| strncmp(s1,s2,n) | <p>Compares first n character of string s1 and s2 and returns similar result as strcmp() function.</p> <pre>printf("%d", strcmp(s1,s2,3));</pre> <p>Output : 0</p> |
| strrchr(s1,c) | <p>Returns the last occurrence of a given character in a string s1.</p> <pre>printf("%s",strrchr(s2,'e'));</pre> <p>Output : ere</p> |



Computer Programming and Utilization (CPU) – 110003

G) Functions

1 What is user defined function? Explain with example. Define the syntax of function in C.

- A function is a block of code that performs a specific task.
- The functions which are created by programmer are called user-defined functions.
- The functions which are in-built in compiler are known as system functions.
- The functions which are implemented in header libraries are known as library functions.
- It has a unique name and it is reusable i.e. it can be called from any part of a program.
- Parameter or argument passing to function is optional.
- It is optional to return a value to the calling program. Function which is not returning any value from function, their return type is void.

While using function, three things are important

1. Function Declaration

- Like variables, all the functions must be declared before they are used.
- The function declaration is also known as function prototype or function signature. It consists of four parts,
 - 1) Function type (return type).
 - 2) Function name.
 - 3) Parameter list.
 - 4) Terminating semicolon

Syntax: <return type> FunctionName (Argument1, Argument2, Argument3.....);

Example: `int sum(int , int);`

- In this example, function return type is int, name of function is sum, 2 parameters are passed to function and both are integer.

2. Function Definition

- Function Definition is also called function implementation.
- It has mainly two parts.
 - Function header : It is same as function declaration but with argument name.
 - Function body : It is actual logic or coding of the function

3. Function call

- Function is invoked from main function or other function that is known as function call.
- Function can be called by simply using a function name followed by a list of actual argument enclosed in parentheses.



Syntax or general structure of a Function

```
<return type> FunctionName (Argument1, Argument2, Argument3.....)
{
    Statement1;
    Statement2;
    Statement3;
}
```

An example of function

```
#include<stdio.h>
int sum(int, int);           \\ Function Declaration or Signature
void main()
{
    int a, b, ans;
    scanf("%d%d", &a, &b);
    ans = sum(a, b);        \\ Function Calling
    printf("Answer = %d", ans);
}
int sum (int x, int y)      \\ Function Definition
{
    int result;
    result = x + y;
    return (result);
}
```

2 Explain different categories of functions.

Functions can be classified in one of the following category based on whether arguments are present or not, whether a value is returned or not.

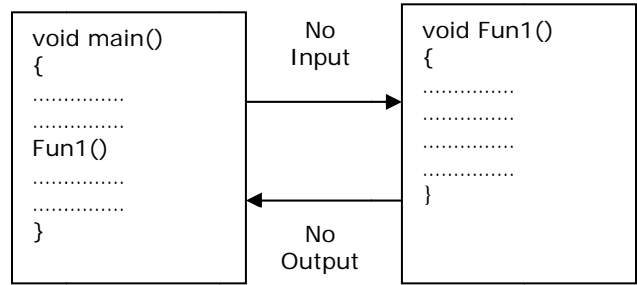
- 1. Functions with no arguments and no return value \\ void printline(void)
- 2. Functions with no arguments and return a value \\ int printline(void)
- 3. Functions with arguments and no return value \\ void printline(int a)
- 4. Functions with arguments and one return value \\ int printline(int a)
- 5. Functions that return multiple values using pointer \\ void printline(int a)

- 1. Function with no argument and no return value.
 - When a function has no argument, it does not receive any data from calling function.
 - When it does not return a value, the calling function does not receive any data from the called function.
 - In fact there is no data transfer between the calling function and called function.



Example:

```
#include<stdio.h>
void printline(void);      // No argument - No return value
void main()
{
    clrscr();
    printline();
    printf("\n GTU \n");
}
void printline(void)
{
    int i;
    for(i=0;i<10;i++)
    {
        printf("-");
    }
}
```

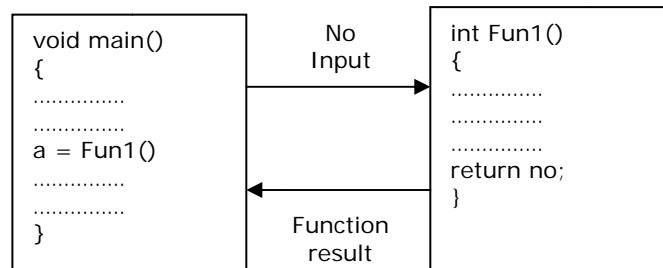


2. Function with no arguments and return a value

- When a function has no argument, it does not receive data from calling function.
- When a function has return value, the calling function receives one data from the called function.

Example:

```
#include<stdio.h>
int get_number(void);      // No argument
void main()
{
    int m;
    m=get_number();
    printf("%d",m);
}
int get_number(void)
{
    int number;
    printf("enter number:");
    scanf("%d",&number);
    return number;        // Return value
}
```



3. Function with arguments and no return values

- When a function has argument, it receives data from calling function.
- When it does not return a value, the calling function does not receive any data from the called function.

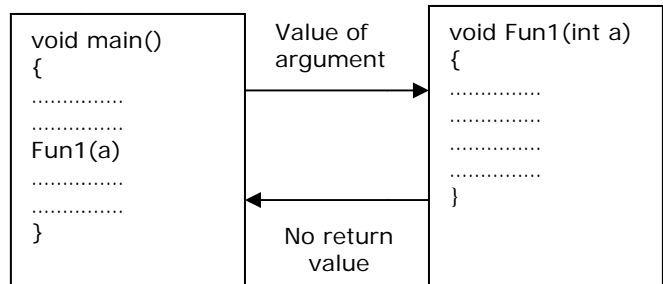
Example:



```

#include<stdio.h>
#include<conio.h>
void sum(int,int);          // Argument
void main()
{
    int no1,no2;
    printf("enter no1,no2:");
    scanf("%d%d",&no1,&no2);
    sum(no1,no2);
}
void sum(int no1,int no2)
{
    if(no1>no2)
        printf("\n no1 is gretest");
    else
        printf("\n no2 is gretest");
} // No return value

```



4. Function with arguments and one return value

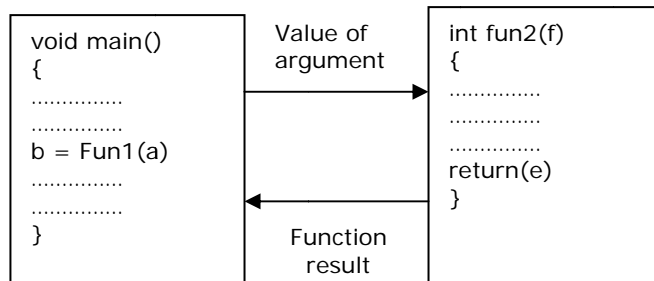
- When a function has argument, it receives data from calling function.
- When a function has return value, the calling function receives any data from the called function.

Example:

```

#include<stdio.h>
#include<conio.h>
int sum(int);          // Argument
void main()
{
    int no,x;
    clrscr();
    printf("enter no:");
    scanf("%d",&no);
    x=sum(no);
    printf("sum=%d",x);
    getch();
}
int sum(int no)
{
    int add=0,i;
    while(no>0)
    {
        i=no%10;

```





```
        add=add+i;
        no=no/10;
    }
    return add;           // Return value
}
```

5. Function that returns a multiple value

- Function can return either one value or zero value. It cannot return more than one value
- To receive more than one value from function, we have to use pointer.
- So function should be called with reference not with value

Example:

```
#include<stdio.h>
void mathoperation(int x, int y, int *s, int *d);
void main()
{
    int x=20,y=10,s,d;
    mathoperation(x,y,&s,&d);
    printf("s=%d \nd=%d", s,d);
}
void mathoperation(int a, int b, int *sum, int *diff)
{
    *sum = a + b;
    *diff = a - b;
}
```

3 Explain actual argument and formal argument with example.

- Arguments passed to the function during function calling are called actual arguments or parameters.
- Arguments received in the definition of a function are called formal arguments or parameters.

Example:

```
#include<stdio.h>
int max(int , int ) // Function Declaration.
void main()
{
    int a=5,b=3, ans;
    ans=max(a, b);           //a and b are actual arguments.
    printf("max=%d", ans);
}
int max(int x, int y)      // x and y are formal arguments.
{
    if(x>y)
        return x;
```



```
        else
            return y;
    }
```

4 Explain call by value (pass by value) and call by reference (pass by reference) with example.

The parameters can be passed in two ways during function calling,

- Call by value
- Call by reference

Call by value

- In call by value, the values of actual parameters are copied to their corresponding formal parameters.
- So the original values of the variables of calling function remain unchanged.
- Even if a function tries to change the value of passed parameter, those changes will occur in formal parameter, not in actual parameter.

Example:

```
#include<stdio.h>
void swap(int, int);
void main()
{
    int x, y;
    printf("Enter the value of X & Y:");
    scanf("%d%d", &x, &y);
    swap(x, y);
    printf("\n Values inside the main function");
    printf("\n x=%d, y=%d", x, y);
    getch();
}
void swap(int x,int y)
{
    int temp;
    temp=x;
    x=y;
    y=temp;
    printf("\n Values inside the swap function");
    printf("\n x=%d y=%d", x, y);
}
}
```

Output:

Enter the value of X & Y: 3 5
Values inside the swap function
X=5 y=3



Values inside the main function

X=3 y=5

Call by Reference

- In call by reference, the address of the actual parameters is passed as an argument to the called function.
- So the original content of the calling function can be changed.
- Call by reference is used whenever we want to change the value of local variables through function.

Example:

```
#include<stdio.h>
void swap(int *, int *);
void main()
{
    int x,y;
    printf("Enter the value of X & Y:");
    scanf("%d%d", &x, &y);
    swap(&x, &y);
    printf("\n Value inside the main function");
    printf("\n x=%d y=%d", x, y);
}
void swap(int *x, int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
    printf("\n Value inside the swap function");
    printf("\n x=%d y=%d", x, y);
}
```

Output:

Enter the value of X & Y: 3 5
Value inside the swap function
X=5 y=3
Value inside the main function
X=5 y=3

Difference between two program is marked as bold in call by reference program

4 What do you mean by recursive function? Explain with example.



- Recursive function is a function that calls itself.
- If a function calls itself then it is known as recursion.
- Recursion is thus the process of defining something in terms of itself.
- Suppose we want to calculate the factorial of a given number then in terms of recursion we can write it as $n! = n * (n-1)!$. First we have to find $(n-1)!$ and then multiply it by n . the $(n-1)!$ is computed as $(n-1)! = (n-1) * (n-2)!$. This process end when finally we need to calculate $1!$.which is 1.

Ex: $4! = 4*3!$

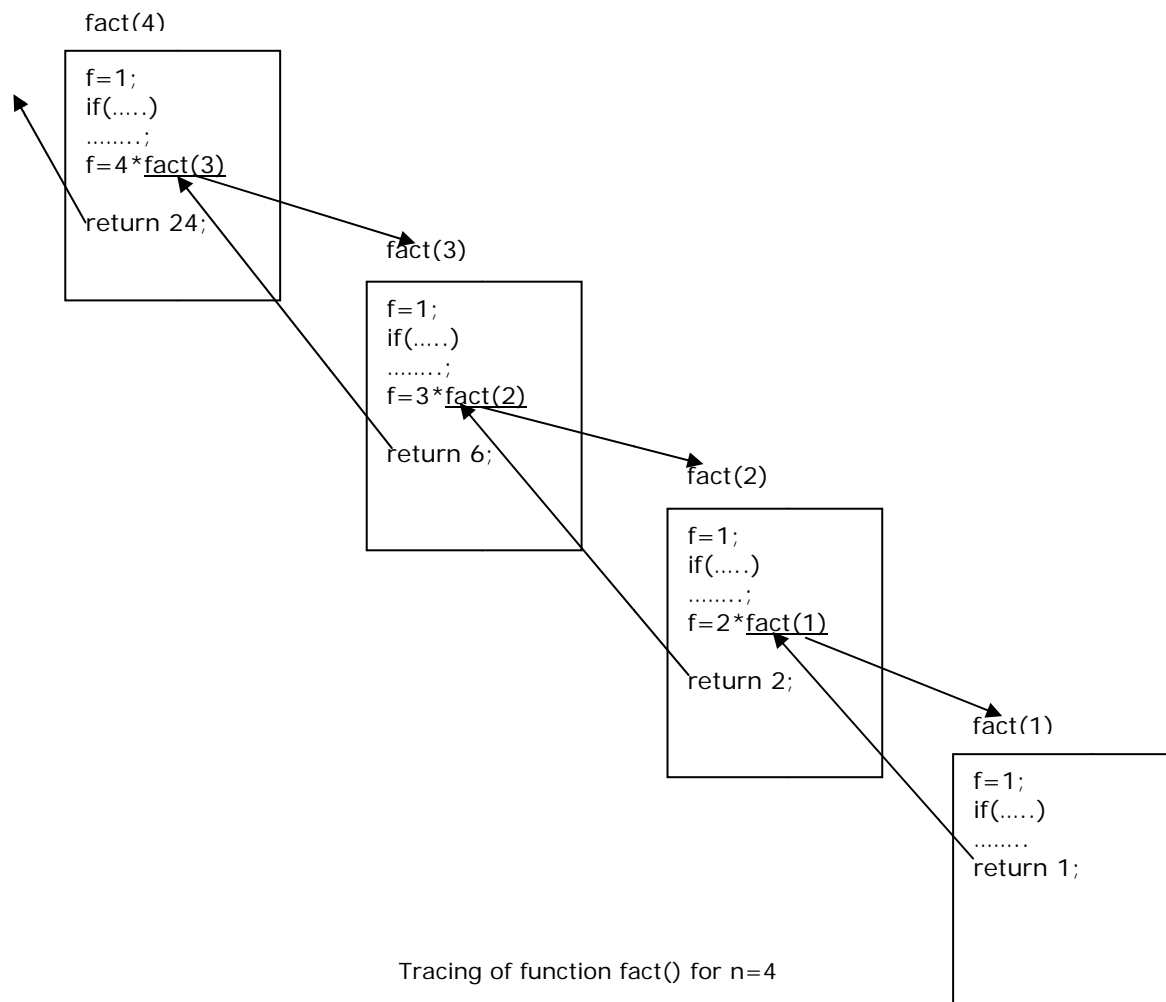
$$= 4*3*2!$$

$$= 4*3*2*1!$$

$$= 4*3*2*1$$

- Terminating condition must be there which can terminates the chain of process, otherwise it will lead to infinite number of process.

Pictorial presentation of recursion.





Example: Find factorial of a given number using recursion.

```
#include<stdio.h>
#include<conio.h>
int fact(int);
void main()
{
    int f,n;
    printf("enter number:");
    scanf("%d",&n);
    f=fact(n);
    printf("\n factorial=%d",f);
}
int fact(int n)
{
    int f=1;
    if(n==1)
    {
        return 1;
    }
    else
    {
        f=n*fact(n-1);
        return f;
    }
}
```

Advantages

- Easy solution for recursively defined solution.
- Complex programs can be easily written with less code.

Disadvantages

- Recursive code is difficult to understand and debug.
- Terminating condition is must; otherwise it will go in an infinite loop.
- Execution speed decreases because of function call and return activity many times.

5 What is scope, lifetime and visibility of variable?

Scope

- The scope of variable can be defined as that a part of a program where the particular variable is accessible
- In what part of the program the variable is accessible is depends on where the variable is declared.
- Local variables which are declared inside the body of function cannot be accessed outside the body of function.



- Global variables which are declared outside any function definition can be accessible by all the function in a program.

Lifetime

- Lifetime is a time limit during the program execution until which a variable exist in a memory.
- It is also referred to the longevity of variable.

Visibility

- Visibility is the ability of the program to access a variable from the memory.
- If variable is redeclared within its scope of variable, the variable losses the visibility in the scope of variable which is redeclared.



Computer Programming and Utilization (CPU) – 110003 Pointers

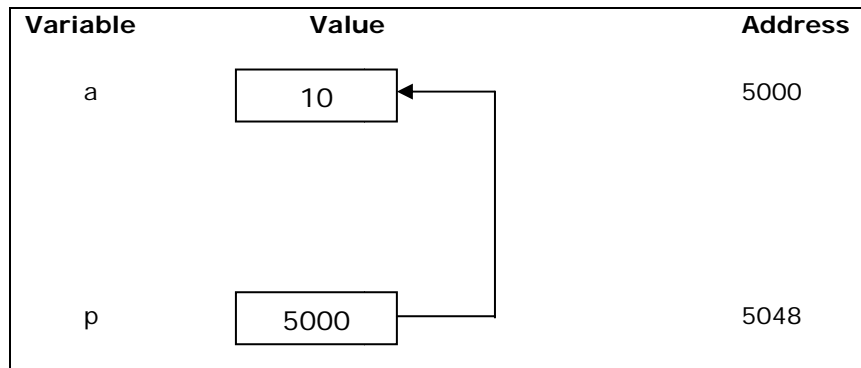
1 What is pointer? How to declare and initialize it?

- A pointer is a variable that contains address or location of another variable.
- Pointer is a derived data type in C.
- Pointers contain memory address as their values, so they can also be used to access and manipulate data stored in memory.

```
void main()
{
    int a=10, *p;
    p = &a;           \\ Assign memory address of a to pointer variable p
    printf("%d %d %d", a, *p, p);
}
```

Output: 10 10 5000

- p is integer pointer variable
- & is address of or referencing operator which returns memory address of variable
- * is indirection or dereferencing operator which returns value stored at that memory address
- & operator is the inverse of * operator (x = a is same as x = *(&a))



Declaration of pointer,

Syntax: data_type *pt_name;

Example: int *p, float *q, char *c;

- 1) The asterisk (*) tells that the variable pt_name is a pointer variable
- 2) pt_name needs a memory location to store address of another variable
- 3) pt_name points to a variable of type data_type

Initialization of the pointer,

int a=5, x, *p; // Declares pointer variable p and regular variable a and x

p = &a // Initializes p with address of a

x = *p; // p contains address of a and *p gives value stored at that address.



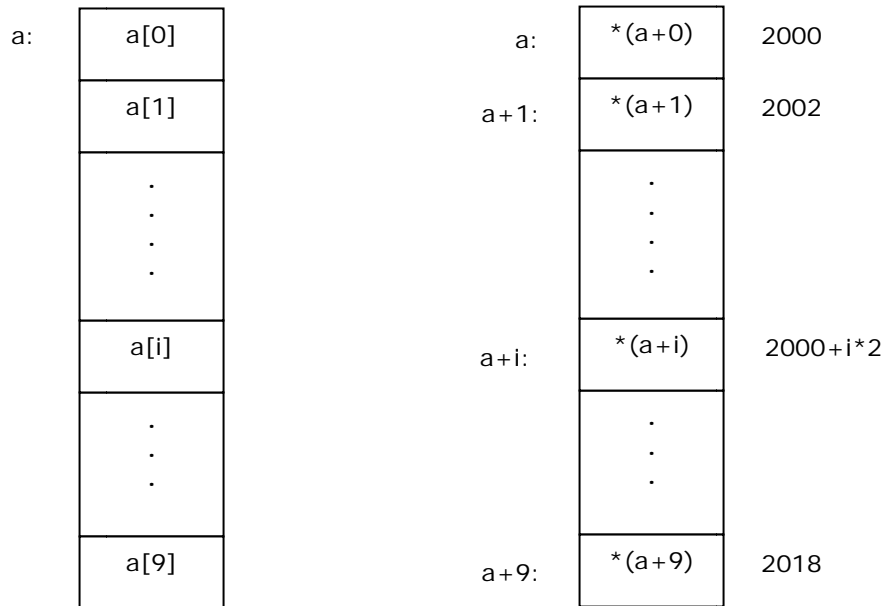
2 How pointer is different from array?

| Array | Pointer |
|---|---|
| Array is a constant pointer. | Pointer variable can be changed. |
| It refers directly to the elements. | It refers address of the variable. |
| Memory allocation is in sequence. | Memory allocation is random. |
| Allocates the memory space which cannot resize or reassigned. | Allocated memory size can be resized. |
| It is a group of elements. | It is not a group of elements. It is single variable. |

3 Discuss relationship between array and pointer.

```
int a[10], *p;
```

- Array name is constant pointer so a is constant pointer and it always points to the first element of an array.
- a[0] is same as *(a+0), a[2] is same as *(a+2), a[i] is same as *(a+i)
- So every program written using array can always be written using pointer.



4 Explain Array of Pointers

As we have an array of char, int, float etc..., same way we can have an array of pointers, individual elements of an array will store the address values. So, array of pointers is a collection of pointers of same type known by single name.



Syntax :

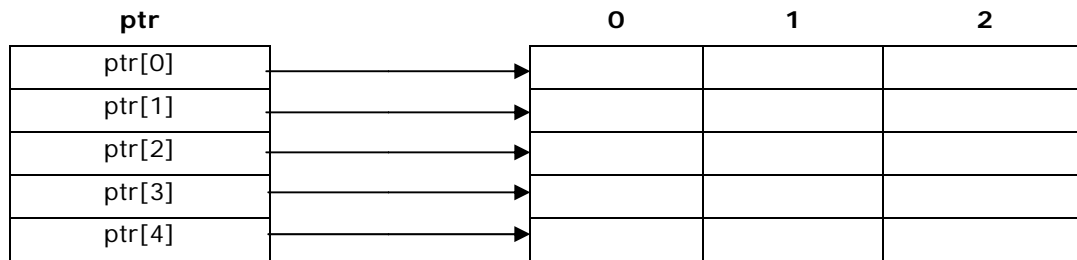
```
data_type *name[size];
```

Example :

```
int *ptr[5];    \\Declares an array of integer pointer of size 5
int mat[5][3]; \\Declares a two dimensional array of 5 by 2
```

Now, the array of pointers ptr can be used to point to different rows of matrix as follow

```
for(i=0;i<5;i++)
{
    ptr[i]=&mat[i][0];    \\ Can be re-written as ptr[i]=mat[i];
}
```



By using dynamic memory allocation, we do not require to declare two-dimensional array, it can be created dynamically using array of pointers.

**5 Swap value of two variables using pointer. OR
Swap value of two variables using call by reference**

```
#include<stdio.h>
void swap(int *,int *);
void main()
{
    int a,b;
    printf("Enter two numbers:");
    scanf("%d%d", &a, &b);
    swap(&a, &b);
    printf("a=%d b=%d", a, b);
    getch();
}
void swap(int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
```



6 Write a C program to calculate sum of 10 elements of an array using pointers

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int *p, a[10], sum=0, i;
    p=a;
    printf("Enter elements:");
    for(i=0; i<10; i++)
    {
        scanf("%d", &a[i]);
    }
    for(i=0; i<10; i++)
    {
        sum=sum+*p;
        p++;
    }
    printf("sum=%d", sum);
    getch();
}
```



Computer Programming and Utilization (CPU) – 110003

Structure, Union

1 What is structure? How to declare a Structure? Explain with Example

- Structure is a collection of logically related data items of different data types grouped together under a single name.
- Structure is a user defined data type.
- Structure helps to organize complex data in a more meaningful way.

Syntax of Structure:

```
struct structure_name
{
    data_type member1;
    data_type member2;
    .....
};
```

- struct is a keyword.
- structure_name is a tag name of a structure.
- member1, member2 are members of structure.

Example:

```
#include<stdio.h>
#include<conio.h>
struct book
{
    char title[100];
    char author[50];
    int pages;
    float price;
};
void main()
{
    struct book book1;
    printf("enter title, author name, pages and price of book");
    scanf("%s",book1.title);
    scanf("%s", book1.author);
    scanf("%d",&book1.pages);
    scanf("%f",&book1.price);
    printf("\n detail of the book");
    printf("%s",book1.title);
    printf("%s",book1.author);
```



```
printf("%d",book1.pages);  
printf("%f",book1.price);  
getch();  
}
```

- book is structure whose members are title, author, pages and price.
- book1 is a structure variable.

2 How do we declare and access structure variables?

Declaration of structure:

A structure variable declaration is similar to the declaration of variables of any other data type. It includes the following elements:

- 1) The keyword struct
- 2) The structure tag name
- 3) List of variable names separated by commas
- 4) A terminating semicolon

For example:

```
struct book  
{  
    char title[100];  
    char author[50];  
    int pages;  
    float price;  
} book1;  
struct book book2;
```

We can declare structure variable in two ways

- 1) Just after the structure body like book1
- 2) With struct keyword and structure tag name like book2

Accessing structure members:

The following syntax is used to access the member of structure.

```
structure_variable.member_name
```

- *structure_variable* is a variable of structure and *member_name* is the name of variable which is a member of a structure.
- The "."(dot) operator or 'period operator' connects the member name to structure name.
- for ex:
book1.price represents price of book1

We can assign values to the member of the structure variable book1 as below,

```
strcpy(book1.title,"ANSI C");  
strcpy(book1.author,"Balagurusamy");  
book1.pages=250;  
book1.price=120.50;
```



We can also use scanf function to assign value through a keyboard.

```
scanf("%s",book1.title);
scanf("%d",&book1.pages);
```

3 What is Union?

- Union is user defined data type just like structure.
- Each member in structure is assigned its own unique storage area where as in Union, all the members share common storage area.
- All members share the common area so only one member can be active at a time.
- Unions are used when all the members are not assigned value at the same time.

Example:

```
union book
{
    char title[100];
    char author[50];
    int pages;
    float price;
};
```

4 Difference between Structure and Union

| Structure | Union |
|---|--|
| Each member is assigned its own unique storage area. | All members share the same storage area. |
| Total memory required by all members is allocated. | Maximum memory required by the member is allocated. |
| All members are active at a time. | Only one member is active a time. |
| All members can be initialized. | Only the first member can be initialized. |
| Requires more memory. | Requires less memory. |
| Example: <pre>struct SS { int a; float b; char c; };</pre> <div style="display: flex; flex-direction: column; align-items: center;"> <div style="border: 1px solid black; width: 150px; height: 20px; margin-bottom: 2px; text-align: center;">1 byte for c</div> <div style="border: 1px solid black; width: 150px; height: 20px; margin-bottom: 2px; text-align: center;">2 bytes for a</div> <div style="border: 1px solid black; width: 150px; height: 20px; text-align: center;">4 bytes for b</div> </div> | Example: <pre>union UU { int a; float b; char c; };</pre> |
| Total bytes = 1 + 2 + 4 = 7 bytes. | 4 bytes are there between a,b and c because largest memory occupies by float which is 4 bytes. |



5 Explain nested structure with example

- A structure that contains another structure as a member variable is known as nested structure or structure within a structure.
- Structure which is part of other structure must be declared before the structure in which it is used.

Example:

```
#include<stdio.h>
#include<conio.h>
struct address
{
    char add1[50];
    char add2[50];
    char city[25];
};
struct employee
{
    char name[100];
    struct address a;
    int salary;
};
void main()
{
    struct employee e;
    printf("enter name,address,city,salary");
    scanf("%s",e.name);
    scanf("%s",e.a.add1);
    scanf("%s",e.a.add2);
    scanf("%s",e.a.city);
    scanf("%d",&e.salary);
    printf("detail of employaee:");
    printf("%s",e.name);
    printf("%s",e.a.add1);
    printf("%s",e.a.add2);
    printf("%s",e.a.city);
    printf("%d",e.salary);
    getch();
}
```



6 Explain Array of Structure with Example?

- As we have an array of basic data types, same way we can have an array variable of structure.
- It is better to use array of size 66 instead of 66 variables to store result of 66 student.

Following example shows how an array of structure can be used

```
#include<stdio.h>
#include<conio.h>
struct result
{
    char name[100];
    int rollno;
    float cpi;
};
void main()
{
    struct result r[66];
    int i;
    printf("enter detail of student :");
    for(i=0;i<66;i++)
    {
        printf("\nenter name,roll no,cpi");
        scanf("%s",r[i].name);
        scanf("%d%f",&r[i].rollno,&r[i].cpi);
    }
    printf("\n detail of student:\n");
    for(i=0;i<66;i++)
    {
        printf("%s",r[i].name);
        printf("\t%d",r[i].rollno);
        printf("\t%f\n",r[i].cpi);
    }
    getch();
}
```



Computer Programming and Utilization (CPU) -110003 J) File Handling

File Management

- In real life, we want to store data permanently so that later on we can retrieve it and reuse it.
- A file is a collection of bytes stored on a secondary storage device like hard disk, pen drive, and tape.
- There are two kinds of files that programmers deal with text files and binary files.
- Text file are human readable and it is a stream of plain English characters.
- Binary files are not human readable. It is a stream of processed characters and Ascii symbols.

File Opening Modes

- We want to open file for some purpose like, read file, create new file, append file, read and write file, etc...
- When we open any file for processing, at that time we have to give file opening mode.
- We can do limited operations only based on mode in which file is opened.

e.g. `fp = fopen("demo.txt", "r");` //Here file is opened in read only mode.

C has 6 different file opening modes for text files,

1. **r** open for reading only.
2. **w** open for writing (If file exists then it is overwritten)
3. **a** open for appending (If file does not exist then it creates new file)
4. **r+** open for reading and writing, start at beginning
5. **w+** open for reading and writing (overwrite file)
6. **a+** open for reading and writing, at the end (append if file exists)

Same modes are also supported for binary files by just adding **b**, e.g. **rb**, **wb**, **ab**, **r+b**, **w+b**, **a+b**

Write a C program to display file on screen.

```
#include <stdio.h>
void main()
{
    FILE *fp;           // fp is file pointer. FILE is a structure defined in stdio.h
    char ch ;

    fp = fopen("prog.c", "r"); // Open Prog.c file in read only mode.
    c = getc(fp) ;
    while (ch != EOF)       // EOF = End of File. Read file till end
    {
        putchar(ch);
        ch = getc (fp);
        //Reads single character from file and advances position to next character
    }
    fclose(fp);           // Close the file so that others can access it.
}
```




Write a C program to copy a file.

```
#include <stdio.h>
void main()
{
    FILE *p,*q;
    char ch;
    p = fopen("Prog.c","r");
    q = fopen("Prognew.c","w");
    ch = getc(p);
    while(ch != EOF)
    {
        putchar(ch,q);
        ch = getc(p);
    }
    printf("File is copied successfully. ");
    fclose(p);
    fclose(q);
    return 0;
}
```

Explain file handling functions with example.

C provides a set of functions to do operations on file. These functions are known as file handling functions. Each function is used for some particular purpose.

fopen() (Open file)

- fopen is used to open a file for operation.
- Two arguments should be supplied to fopen function,
- File name or full path of file to be opened
- File opening mode which indicates which type of operations are permitted on file.
- If file is opened successfully, it returns pointer to file else NULL.

Example: `fp = fopen("Prog.c","r"); // File name is prog.c and it is opened for reading only.`

fclose() (Close file)

- Opened files must be closed when operations are over.
- The function fclose is used to close the file i.e. indicate that we are finished processing this file.
- To close a file, we have to supply file pointer to fclose function.
- If file is closed successfully then it returns 0 else EOF.

Example: `fclose(fp);`

fprintf() (Write formatted output to file)

- The fprintf function prints information in the file according to the specified format.
- fprintf() works just like printf(), only difference is we have to pass file pointer to the function.
- It returns the number of characters outputted, or a negative number if an error occurs.

Example: `fprintf(fp, "Sum = %d", sum);`



fscanf() (Read formatted data from file)

- The function `fscanf()` reads data from the given file.
- It works in a manner exactly like `scanf()`, only difference is we have to pass file pointer to the function.
- If reading is succeeded then it returns the number of variables that are actually assigned values, or EOF if any error occurred.

Example: `fscanf(fp, "%d", &sum);`

fseek() (Reposition file position indicator)

- Sets the position indicator associated with the file pointer to a new position defined by adding offset to a reference position specified by origin.
- You can use `fseek()` to move beyond a file, but not before the beginning.
- `fseek()` clears the EOF flag associated with that file.
- We have to supply three arguments, file pointer, how many characters, from which location.
- It returns zero upon success, non-zero on failure.

The `origin` value should have one of the following values

| Name | Explanation |
|-----------------------|---------------------------------|
| <code>SEEK_SET</code> | Seek from the start of the file |
| <code>SEEK_CUR</code> | Seek from the current location |
| <code>SEEK_END</code> | Seek from the end of the file |

Example: `fseek(fp, 9, SEEK_SET);` // Moves file position indicator to 9th position from begging.

ftell() (Get current position in file)

It returns the current value of the position indicator of the file.

For binary streams, the value returned corresponds to the number of bytes from the beginning of the file.

Example: `position = ftell(fp);`

rewind() (Set position indicator to the beginning)

- Sets the position indicator associated with file to the beginning of the file.
- A call to `rewind` is equivalent to: `fseek(fp, 0, SEEK_SET);`
- On file open for update (read+write), a call to `rewind` allows to switch between reading and writing.

Example: `rewind(fp);`

getc() (Get character from file)

- `getc` function returns the next character from file or EOF if the end of file is reached.
- After reading a character, it advances position in file by one character.
- `getc` is equivalent to `getchar()`.
- `fgetc` is identical to `getc`.

Example: `ch = getc(fp);`

putc() (Write character to file)

- `putc` writes a character to the file and advances the position indicator.
- After reading a character, it advances position in file by one character.



- `putc` is equivalent to `putchar()`.
- `fgetc` is identical to `putc`.

Example: `putc(ch, fp);`

getw() (Get integer from file)

- `getw` function returns the next int from the file. If error occurs then EOF is returned.

Example: `i = getw(fp);`

putw() (Write integer to file)

- `putw` function writes integer to file and advances indicator to next position.
- It succeeded then returns same integer otherwise EOF is returned.

Example: `putw(I, fp);`

Write a program to count the number of lines, tabs, characters and words in a file.

```
#include <stdio.h>
void main()
{
    FILE *fp;
    int lines=0, tabs=0, characters=0, words = 0;
    char ch, filename[100] ;
    printf("Enter file name: ");
    gets( filename );    //You can also use scanf("%s",filename);
    fp = fopen( filename, "r" );
    if ( fp == NULL )    // File is not opened successfully then it returns NULL.
    {
        printf("Cannot open %s for reading \n", filename );
        exit(1);        /* terminate program */
    }
    ch = getc( fp ) ;
    while ( ch != EOF )
    {
        if ( ch == '\n' )
            lines++ ;
        else if ( ch == '\t' )
            tabs ++ ;

        else if ( ch == ' ' )
            words ++ ;
        else
            characters++;
        c = getc ( fp );
    }
    fclose( fp );
    printf("Lines=%d Tabs=%d Wods=%d Characters=%d", lines, tabs, words,
    characters);
}
```



Computer Programming and Utilization (CPU) -110003 K) Dynamic Memory Allocation, Storage Classes

Static Memory Allocation

- If memory is allocated to variables before execution of program starts then it is called static memory allocation.
- It is fast and saves running time.
- It allocates memory from stack.
- It is preferred when size of an array is known in advance or variables are required during most of the time of execution of program.
- Allocated memory stays from start to end of program.
- The storage space is given symbolic name known as variable and using this variable we can access value.
- e.g.
 int i;
 float j;

Dynamic Memory Allocation

- If memory is allocated at runtime (during execution of program) then it is called dynamic memory.
- It is bit slow.
- It allocates memory from heap
- It is preferred when number of variables is not known in advance or very large in size.
- Memory can be allocated at any time and can be released at any time.
- The storage space allocated dynamically has no name and therefore its value can be accessed only through a pointer.
- e.g.
 p = malloc(sizeof(int));

Explain various functions used in Dynamic Memory Allocation.

malloc()

- malloc() is used to allocate a certain amount bytes of memory during the execution of a program.
- malloc() allocates `size_in_bytes` bytes of memory from heap, if the allocation succeeds, a pointer to the block of memory is returned else NULL is returned.
- malloc() returns an uninitialized memory for you to use.
- Malloc() can be used to allocate space for complex data types such as structures.
- Syntax: `ptr_var = (cast_type *)malloc(size_in_bytes);`
- Example :

```
#include<stdio.h>
int main()
{
    int *p ;
    p = (int *)malloc(sizeof(int));
    *p =25;
    printf("%d" ,*p);
    free(P);
}
```

calloc()

- calloc() is used to allocate a block of memory during the execution of a program, e.g. for an array.
- calloc() allocates a region of memory large enough to hold `no_of_blocks` of size `size_of_block` each, if the allocation succeeds then a pointer to the block of memory is returned else NULL is returned.



- Syntax: `ptr_var=(cast_type *)calloc(no_of_blocks ,size_of_block);`
- Example:

```
#include<stdio.h>
int main ()
{
    int i,n;
    int *p;
    printf ("Enter how many numbers:");
    scanf ("%d",&n);
    p = (int*) calloc (n, sizeof(int));

    for (i=0; i<n; i++)
    {
        scanf("%d",p);
        p++;
    }
}
```

realloc()

- `realloc()` reallocates a memory block with a specific new size. If you call `realloc()`, the size of the memory block pointed to by the pointer is changed to the given size in bytes. This way you are able to expand and reduce the amount of memory you want to use.
- It is possible that the function moves the memory block to a new location; then the function returns address of new location. Old memory block is copied to new memory and old memory is released automatically.
- The content will remain unchanged means it is copied to new location.
- If the pointer is NULL then the `realloc()` will behave exactly like the `malloc()`. It will assign a new block of a size in bytes and will return a pointer to it.
- Syntax: `ptr_var=* realloc (void * ptr, size_t size);`
- Example:

```
#include<stdio.h>
int main()
{
    int *p ;
    p = (int *)malloc(sizeof(int));
    *p =25;
    p = (int *)realloc(p, 2 * sizeof(int));
    printf ("%d" ,*p);
    free(P);
}
```

free()

- When the memory is not needed anymore, you must release it calling the function `free`.
- Just pass the pointer of the allocated memory to `free` function and memory is released.
- Syntax: `void free(void *pointer);`
- Example: `free(p);`



Storage Class

- Storage class decides the scope, lifetime and memory allocation of variable.
- Scope of a variable is the boundary within which a variable can be used.
- Four storage classes are available in C,
 - Automatic (`auto`)
 - Register (`register`)
 - External (`extern`)
 - Static (`static`)

automatic:

- Variables which are declared in function are of automatic storage class.
- Automatic variables are allocated storage in the main memory of the computer.
- Memory is allocated automatically upon entry to a function and freed automatically upon exit from the function.
- The scope of automatic variable is local to the function in which it is declared.
- It is not required to use the keyword **auto** because by default storage class within a block is auto.

Example:

```
int a;  
auto int a;
```

register:

- Automatic variables are allocated storage in the main memory of the computer; However, for most computers, accessing data in memory is considerably slower than processing directly in the CPU.
- Register variables are stored in registers within the CPU where data can be stored and accessed quickly.
- Variables which are used repeatedly or whose access time should be fast may be declared to be of storage class **register**.
- Variables can be declared as a register: `register int var;`

external:

- Automatic and register variables have limited scope and limited lifetimes in which they are declared.
- Sometimes we need global variables which are accessible throughout the program.
- `extern` keyword defines a global variable that is visible to ALL functions.
- `extern` is also used when our program is stored in multiple files instead of single file.
- Memory for such variables is allocated when the program begins execution, and remains allocated until the program terminates. Memory allocated for an external variable is initialized to zero.
- Declaration for external variable is as follows: `extern int var;`

static:

- `static` keyword defines a global variable that is visible to ALL functions in same file.
- Memory allocated for static variable is initialized to zero.
- Static storage class can be specified for automatic as well as external variables such as:

```
static extern int varx; //static external  
static int var;        // static automatic
```
- Static automatic variables continue to exist even after the function terminates.
- The scope of static automatic variables is identical to that of automatic variables.



Explain Input / Output functions.

- C language provides a set of standard built-in functions which will do the work of reading or displaying data or information on the I/O devices during program execution.
- Such I/O functions establish an interactive communication between the program and user.

Formatted Input/Output functions

scanf()

- It is used to read all types of data.
- It cannot read white space between strings.
- It can read multiple data at a time by multiple format specifier in one scanf().
- Example: `scanf("%d%d", &a, &b);`

printf()

- It is used to display all types of data and messages.
- It can display multiple data at a time by multiple format specifier in one printf().
- Example: `printf("a=%d b=%d", a, b);`

Unformatted input output functions

gets()

- It is used to read a single string with white spaces.
- It is terminated by enter key or at end of line.
- Example:
`char str[10];`
`gets(str);`

getchar(), getche(), getch()

- It is used to read single character at a time.
- `getchar()` function requires enter key to terminate input while `getche()` and `getch()` does not require.
- `getch()` function does not display the input character while `getchar()` and `getche()` function display the input character on the screen.
- Example:
`char ch;`
`ch = getchar();`
`ch = getche();`
`ch = getch();`

puts()

- It is used to display a string at a time.
- Example:
`char str[]="Hello";`
`puts(str);`

putchar()

- It is used to display single character at a time.
- Example:



```
putchar(ch);
```

Character checking functions.

Character checking functions are available in ctype.h header file.

1. isdigit(int); for checking number (0-9)
2. isalpha(int); for checking letter (A-Z or a-z)
3. isalnum(int); for checking letter (A-Z or a-z) or digit (0-9)
4. isspace(int); for checking empty space
5. islower(int); for checking letter (a-z)
6. isupper(int); for checking letter (A-Z)
7. ispunct(int); for checking punctuation symbols (like - :, ; , { , } , ? , . etc.)

Example: Write a program to check the entered character is digit or not

```
#include<stdio.h>
#include<conio.h>
#include<ctype.h>
void main()
{
    char c;
    clrscr();
    scanf ("%c" ,&c);
    if(isdigit(c))
        printf("True");
    getch();
}
```