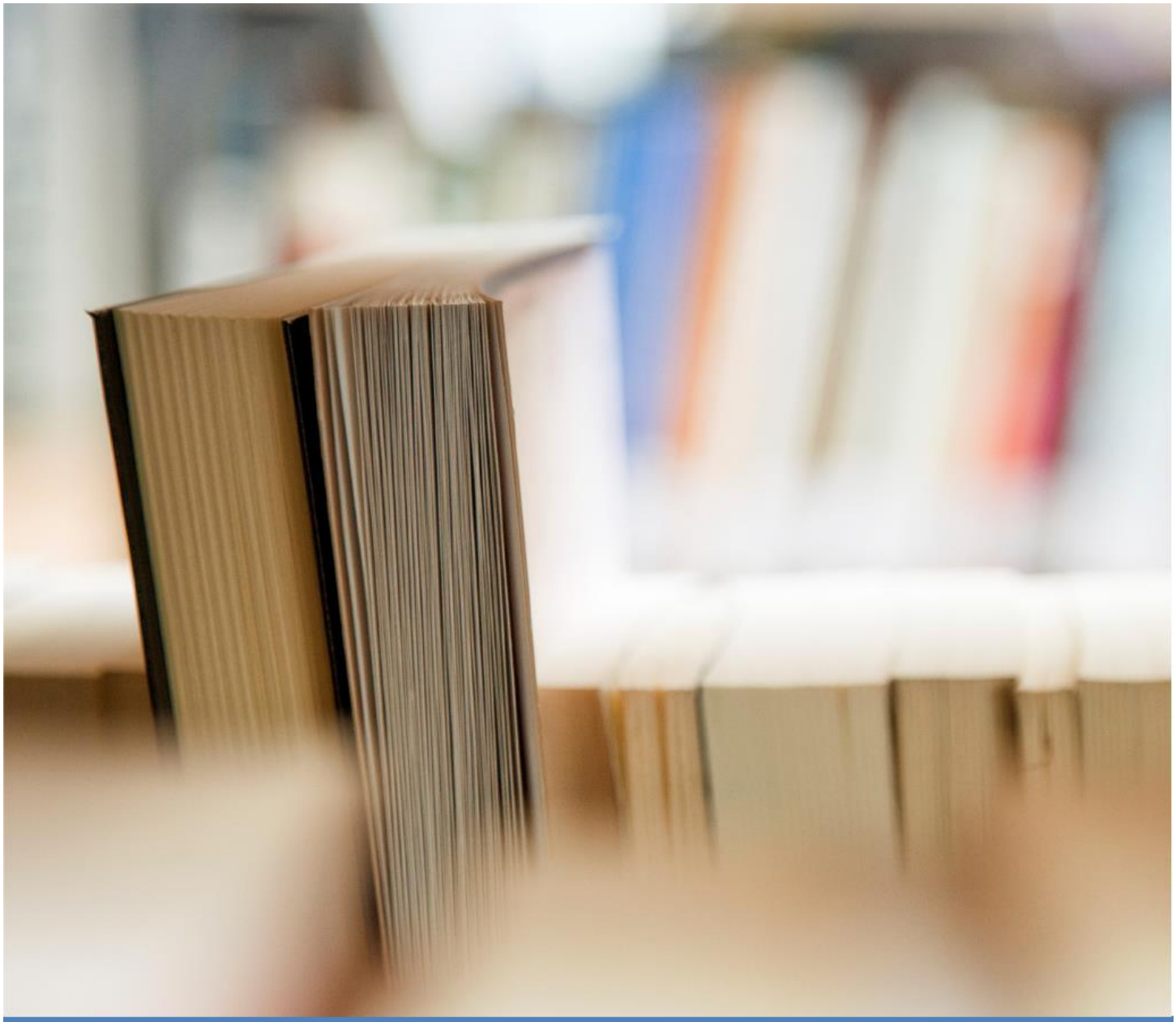


DATA STRUCTURES

BY : AMAR PANCHAL

1. INTRODUCTION TO DATA STRUCTURE-

2. STACK

3. QUEUES

4. LINKED LIST

5. TREES

6. GRAPHS

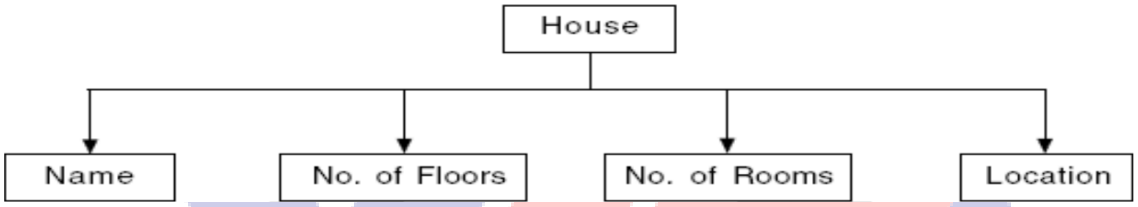
7. SORTING AND SEARCHING

CHAPTER 1
INTRODUCTION TO DATA STRUCTURE

DATA STRUCTURE AND TYPES

Data structures are a method of representing of logical relationships between individual data elements related to the solution of a given problem. Data structures are the most convenient way to handle data of different types including abstract data type for a known problem.
The components of data can be organized and records can be maintained. Further, the record formation leads to the development of abstract data type and database systems.

Figure. Information about a house



In data structures, we also have to decide on the storage, retrieval and operation that should be carried out between logically related items. For example, the data must be stored in memory in computer-understandable format, i.e. 0 and 1 and the data stored must be retrieved in human-understandable format, i.e. ASCII. In order to transform data various operations have to be performed.

TYPES OF DATA STRUCTURES

A data structure is a structured set of variables associated with one another in different ways, cooperatively defining components in the system and capable of being operated upon in the program. As stated earlier, the following operations are done on data structures:

1. Data organisation or clubbing
2. Accessing technique
3. Manipulating selections for information.

Data structures are the basis of programming tools and the choice of data structures should provide the following:

1. The data structures should satisfactorily represent the relationship between data elements.
2. The data structures should be easy so that the programmer can easily process the data.

Data structures have been classified in several ways. Different authors classify it differently. Fig.(a) shows different types of data structures. Besides these data structures some other data structures such as lattice, Petri nets, neural nets, semantic nets, search graphs, etc., can also be used. The reader can see Figs. (a) and (b) for all data structures.

Linear

In linear data structures, values are arranged in linear fashion. Arrays, linked lists, stacks and queues are examples of linear data structures in which values are stored in a sequence.

Non-Linear

This type is opposite to linear. The data values in this structure are not arranged in order. Tree, graph, table and sets are examples of non-linear data structures.

Homogenous

In this type of data structures, values of the same types of data are stored, as in an array.

Non-homogenous

In this type of data structures, data values of different types are grouped, as in structures and classes.

Dynamic

In dynamic data structures such as references and pointers, size and memory locations can be changed during program execution.

Static

Static keyword in C is used to initialize the variable to 0 (NULL). The value of a static variable remains in the memory throughout the program. Value of static variable persists. In C++ member functions are also declared as static and such functions are called as static functions and can be invoked directly.

Figure (a). Types of data structures

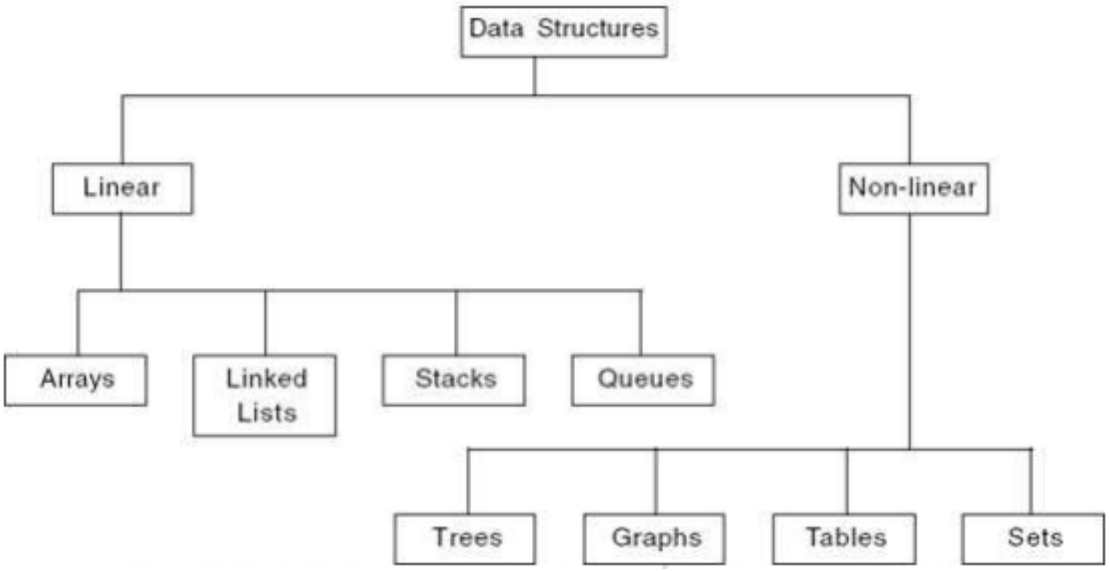
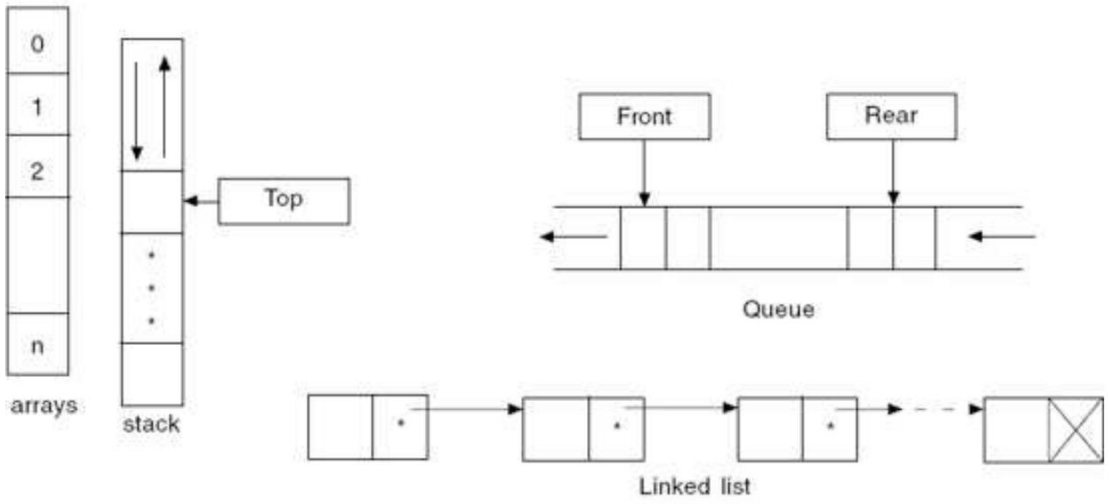
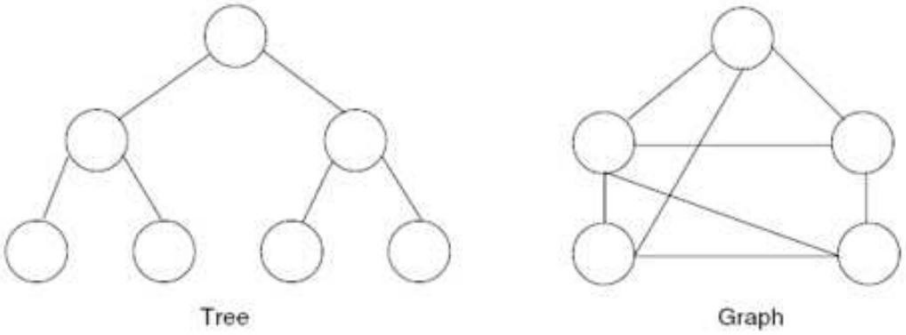
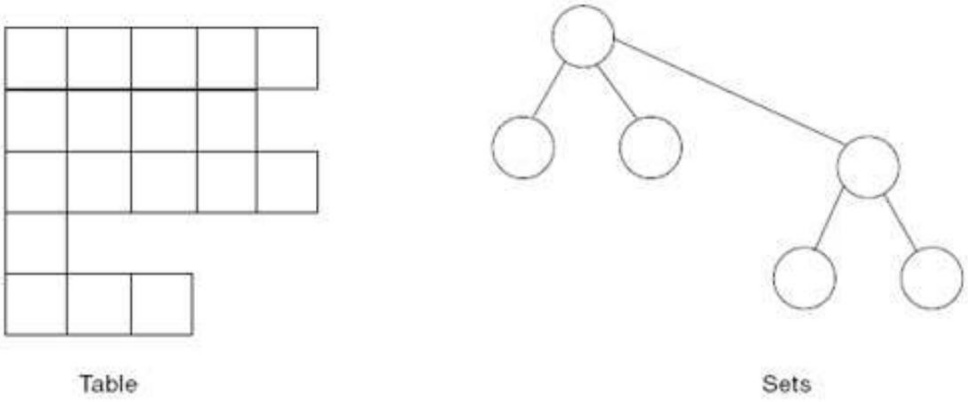


Figure (b). Linear and non-linear data structures



1. Linear data structures





2. Non-linear data structures

PRIMITIVE AND NON-PRIMITIVE DATA STRUCTURES AND OPERATIONS

Primitive Data Structures

The integers, reals, logical data, character data, pointer and reference are primitive data structures. Data structures that normally are directly operated upon by machine-level instructions are known as primitive data structures.

Non-primitive Data Structures

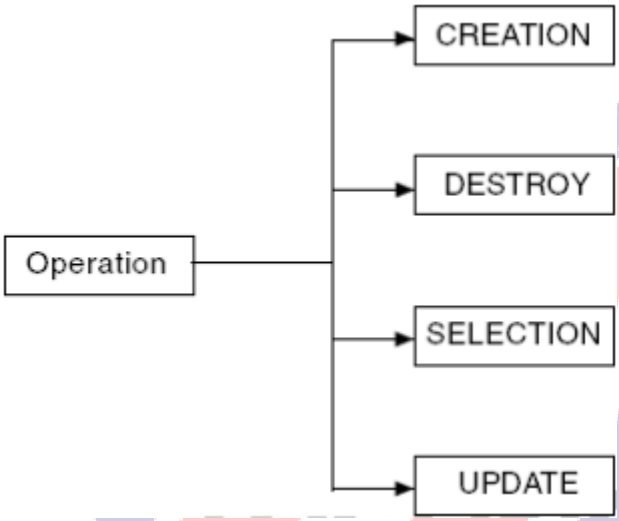
These are more complex data structures. These data structures are derived from the primitive data structures. They stress on formation of sets of homogeneous and heterogeneous data elements.

The different operations that are to be carried out on data are nothing but designing of data structures.

The various operations that can be performed on data structures are shown in Fig. 1.5.

1. CREATE
2. DESTROY
3. SELECT
4. UPDATE

Figure 1.5. Data structures operations



An operation typically used in combination with data structures and that creates a data structure is known as creation. This operation reserves memory for the program elements. It can be carried out at compile time and run-time.

For example,

```
int x;
```

Here, variable x is declared and memory is allocated to it.

Another operation giving the balancing effect of a creation operation is destroying operation, which destroys the data structures. The destroy operation is not an essential operation. When the program execution ends, the data structure is automatically destroyed and the memory allocated is eventually de-allocated. C++ allows the destructor member function to destroy the object. In C free () function is used to free the memory. Languages like Java have a built-in mechanism, called garbage collection, to free the memory.

The most commonly used operation linked with data structures is selection, which is used by programmers to access the data within data structures. The selection relationship depends upon yes/no. This operation updates or alters data. The other three operations associated with selections are:

1. Sorting
2. Searching
3. Merging

Searching operations are used to seek a particular element in the data structures. Sorting is used to arrange all the elements of data structures in the given order: either ascending or descending. There is a separate chapter on sorting and searching in this book. Merging is an operation that joins two sorted lists.

An iteration relationship is nothing but a repetitive execution of statements. For example, if we want to perform any calculation several times then iteration, in which a block of statements is repetitively executed, is useful.

One more operation used in combination with data structures is update operation. This operation changes data of data structures. An assignment operation is a good example of update operation.

For example,

int x=2;-

Here, 2 is assigned to x.

x=4;--

Again, 4 is reassigned to x. The value of x now is 4 because 2 is automatically replaced by 4, i.e. updated.-

ABSTRACT DATA TYPES

In programming, a situation occurs when built-in data types are not enough to handle the complex data structures. It is the programmer's responsibility to create this special kind of data type. The programmer needs to define everything related to the data type such as how the data values are stored, the possible operations that can be carried out with the custom data type and that it must behave like a built-in type and not create any confusion while coding the program. Such custom data types are called Abstract data type. In C struct and in C++ struct/class keywords are used to create abstract data type.

For example, if the programmer wants to define date data type which is not available in C/ C++, s/he can create it using struct or class. Only a declaration is not enough; it is also necessary to check whether the date is valid or invalid. This can be achieved by checking using different conditions. The following program explains the creation of date data type:

Write a program to create abstract data type date.

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

struct date
{
 int dd;
 int mm;  int yy;
};

main ()
{
struct date d; // date is abstract data type
clrscr();
printf ("Enter date (dd mm yy) :");
scanf ("%d %d %d",&d.dd,&d.mm, &d.yy);
printf ("Date %d-%d-%d",d.dd,d.mm,d.yy);
}
```

OUTPUT

```
Enter date (dd/mm/yy): 08 12 2003
Date 08-12-2003
```

RECURSIVE FUNCTION

RULES

1. In recursion, it is essential for a function to call itself, otherwise recursion will not take place.
2. Only user defined function can be involved in the recursion. Library function cannot be involved in recursion because their source code cannot be viewed.
3. A recursive function can be invoked by itself or by other function. It saves return address in order with the intention that to return at proper location when return to a calling statement is made. The last-in-first-out nature of recursion indicates that stack data structure can be used to implement it.
4. Recursion is turning out to be increasingly important in non-numeric applications and symbolic manipulations.
5. To stop the recursive function it is necessary to base the recursion on test condition and proper terminating statement such as exit () or return must be written using if () statement.
6. Any function can be called recursively. An example is illustrated in amar sir's class.
7. When a recursive function is executed, the recursion calls are not implemented instantly. All the recursive calls are pushed onto the stack until the terminating condition is not detected. As soon as the terminating condition is detected, the recursive calls stored in the stack are popped and executed. The last call is executed first then second, then third and so on.
8. During recursion, at each recursive call new memory is allocated to all the local variables of the recursive functions with the same name.-
9. At each call the new memory is allocated to all the local variables, their previous values are pushed onto the stack and with its call. All these values can be available to corresponding function call when it is popped from the stack.

RECURSION VS. ITERATIONS

Recursion Vs. Iteration	
Recursion	**Iteration**
Recursion is the term given to the mechanism of defining a set or procedure in terms of itself.	The block of statement executed repeatedly using loops.
A conditional statement is required in the body of the function for stopping the function execution.	The iteration control statement itself contains statement for stopping the iteration. At every execution, the condition is checked.
At some places, use of recursion generates extra overhead. Hence, better to skip when easy solution is available with iteration.	All problems can be solved with iteration.
Recursion is expensive in terms of speed and memory.	Iteration does not create any overhead. All the programming languages support iteration.

Iterative Processes

Initialization

The variables involved in the iteration process are initialized. These variables are used to decide when to end the loop.

Decision

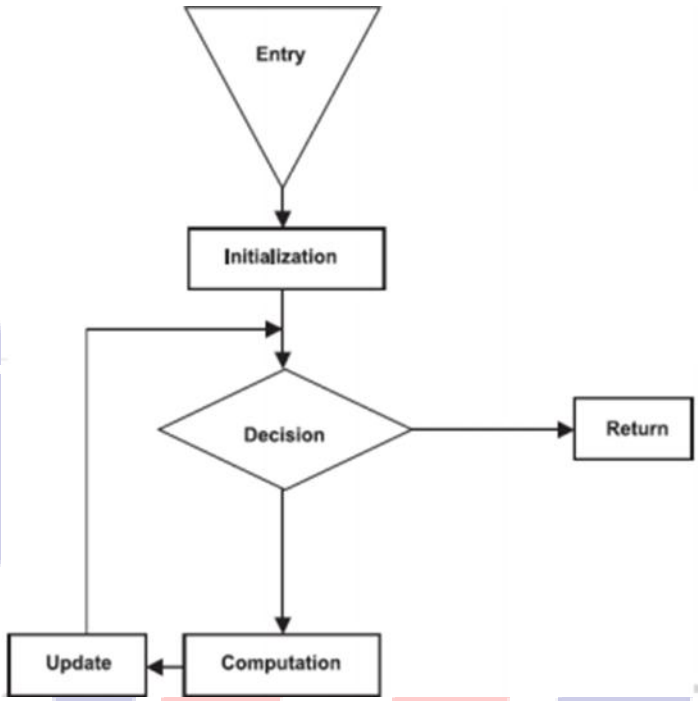
The decision variable is used to decide whether to continue or discontinue the loop. When the condition is satisfied, control goes to return, or else it goes to computation block.

Computation

The required processing or computation is carried out in this block.

Update

The decision argument is changed and shifted to next iteration.



Recursive Function

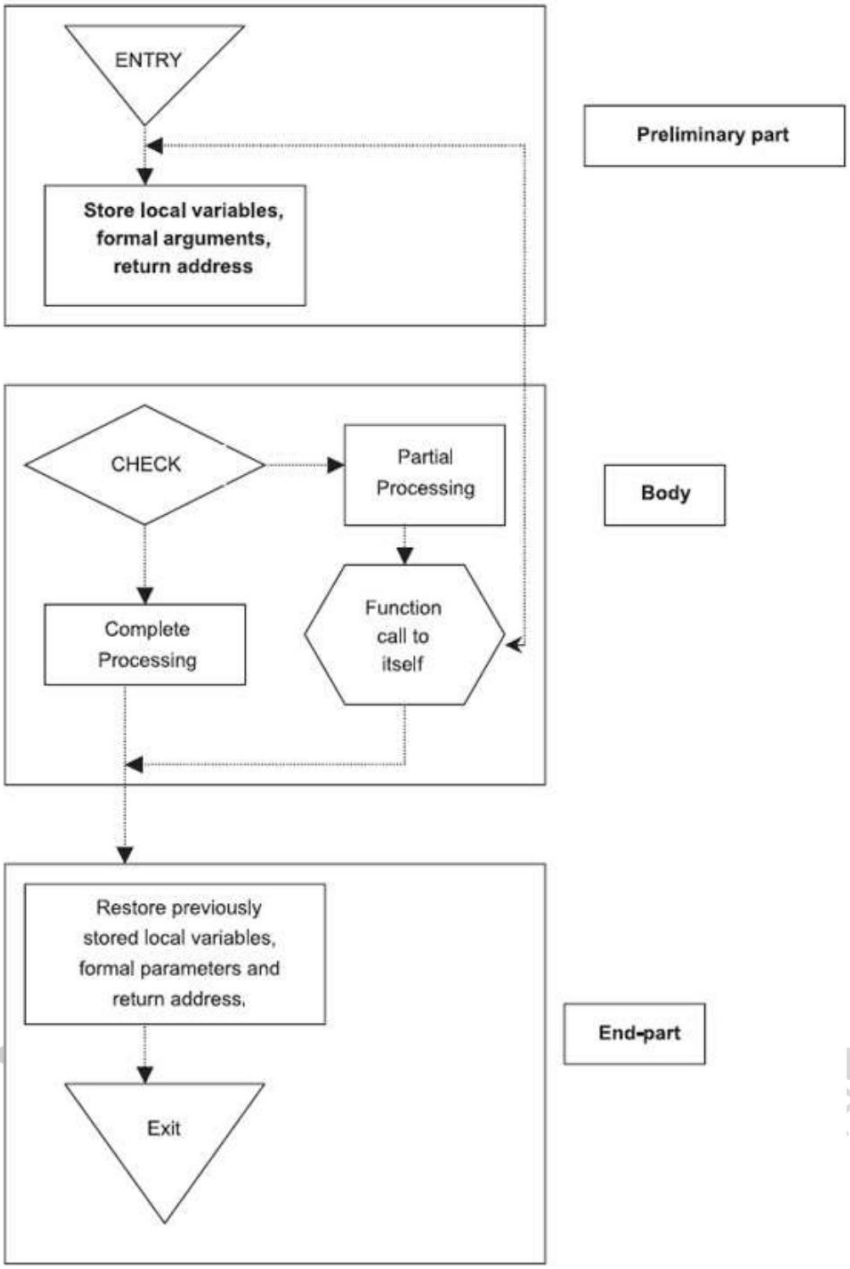
Preliminary part

The use of this block is to store the local variables, formal arguments and return address. The end-part will restore these data. Only recently saved arguments, local variables and return address are restored. The variables last saved are restored first.

Body

If the test condition is satisfied, it performs the complete processing and control passes to end-block. If not, partial processing is performed and a recursive call is made. The body also contains call to itself and one or more calls can be made. Every time a recursive call is made, the preliminary part of the function saves all the data. The body also contains two processing boxes i.e. partial processing and complete processing. In some programs, the result can be calculated after complete processing. For this, the recursive call may not be required. For example, we want to calculate factorial of one.

The factorial of one is one. For this, it is needless to call function recursively. It can be solved by transferring control to complete processing box.



In other case, if five is given for factorial calculation, the factorial of five can be calculated in one step. Hence, the function will be called recursively. Every time one step is solved, i.e. 5*4*3 and so on. Hence, it is called partial processing.

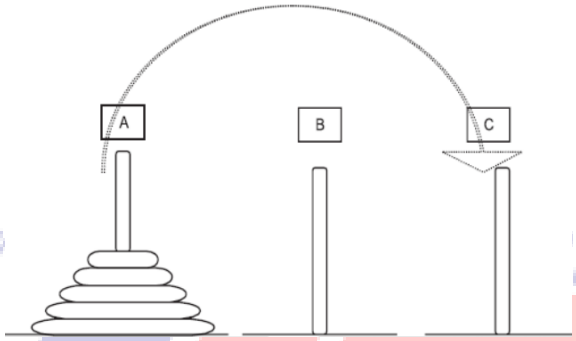
THE TOWERS OF HANOI

The Tower of Hanoi has historical roots in the ceremony of the ancient tower of Brahma. There are n disks of decreasing sizes mounted on one needle as shown in the Fig. (a). Two more needles are also required to stack the entire disk in the decreasing order. The use of third needle is for impermanent storage. While mounding the disk, following rules should be followed.

1. At a time only one disk may be moved.
2. The disk may be moved from any needle to another needle.
3. No larger disk may be placed on the smaller one.

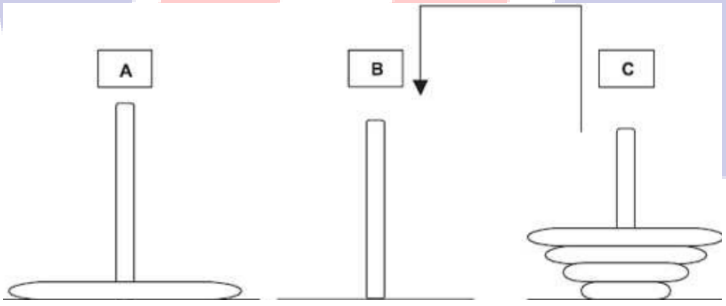
Our aim is to move the disks from A to C using the needle B as an intermediate by obeying the above three conditions. Only top-most disks can be moved to another needle. The following figures and explanation clear the process of Tower of Hanoi stepwise.

Figure (a). Recursive Towers of Hanoi



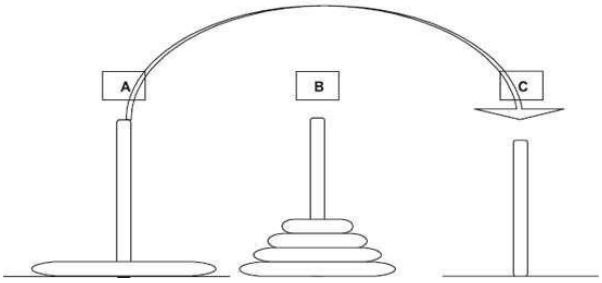
In Fig. (a), the three needles are displayed in their initial states. The needle A contains five disks and there are no disk on needle B and C. The arrow indicates the next operation to be performed, i.e. move first four disk from A to C.

Figure (b). Recursive Towers of Hanoi



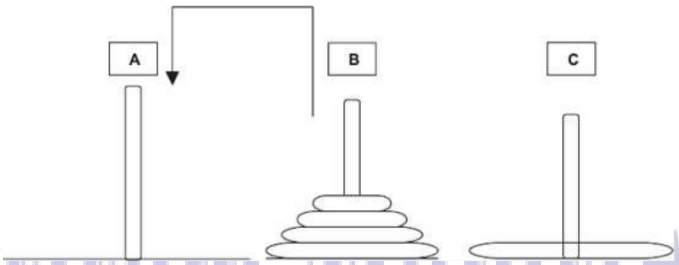
In Fig. (b) the needle, A has only one disk. The needle C contains four disks. The arrow indicates the next operation, i.e. move four disks from C to B.

Figure (c). Recursive Towers of Hanoi



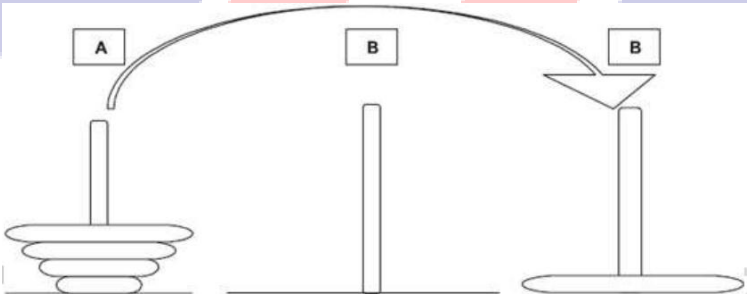
In Fig. (c), the needle B has four disks. The needle A has only one disk, which will be moved to needle C.

Figure (d). Recursive Towers of Hanoi



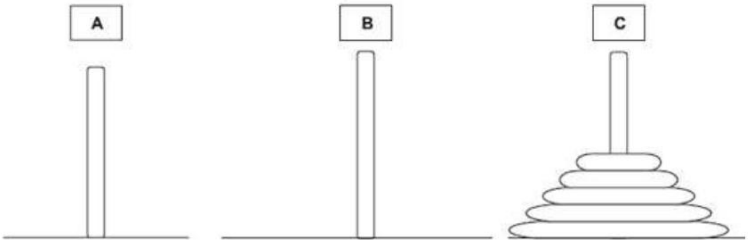
In Fig. (d), the needle A has no disk. The needle B contains four disks and C contains one disk.The four disks move from B to A.-

Figure (e). Recursive Towers of Hanoi-



In Fig. (e), the four disks from needle A are moved to needle C. Finally, the final position of needles A, B and C will be as shown in Fig. (f). Here, solution of Tower of Hanoi is complete.

Figure (f). Recursive Towers of Hanoi



The above process can be summarized as:

1. Move first four disks from needle A to C

2. Move four disks from C to B
3. Move the disk one from A to C
4. Move four disks from B to A
5. Move four disks from A to C.

Program to illustrate Towers of Hanoi.

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

void hanoi (int,char,char,char);

main ()

{
  int num;
  clrscr();
  printf ("\n Enter a Number: ");
  scanf ("%d",&num);
  clrscr();
  hanoi(num,'A','C','B');
}


void hanoi (int num, char f_peg,char t_peg, char a_peg)
{


 if (num==1)
 {
      printf ("\nMove disk 1 from Needle %c to Needle %c",f_peg,t_peg);
      return;
 }
 hanoi(num-1,f_peg,a_peg,t_peg);
 printf ("\nMove disk %d from Needle %c to Needle %c",num,f_peg,t_peg);
 hanoi(num-1,a_peg,t_peg,f_peg);
}
```

OUTPUT

```
Move disk 1 from Needle A to Needle C
Move disk 2 from Needle A to Needle B
Move disk 1 from Needle C to Needle B
Move disk 3 from Needle A to Needle C
Move disk 1 from Needle B to Needle A
Move disk 2 from Needle B to Needle C
Move disk 1 from Needle A to Needle C
```

ADVANTAGES & DISADVANTAGES OF RECURSION

Advantages of recursion,

1. Sometimes, in programming a problem can be solved without recursion, but at some situations in programming it is must to use recursion. For example, a program to display the list of all files of the system cannot be solved without recursion.
2. The recursion is very flexible in data structure like stacks, queues, linked list and quick sort.
3. Using recursion, the length of the program can be reduced.

1. It requires extra storage space. The recursive calls and automatic variables are stored on the stack. For every recursive call separate memory is allocated to automatic variables with the same name.
2. If the programmer forgot to specify the exit condition in the recursive function, the program will execute out of memory. In such a situation user has to press ctrl+ break to pause or stop the function.
3. The recursion function is not efficient in execution speed and time.
4. If possible, try to solve problem with iteration instead of recursion.

CHAPTER 2:
STACKS

Stack is an important tool in programming languages. Stack is one of the most essential linear data structures. Implementation of most of the system programs is based on stack data structure.

We can insert or delete an element from a list, which takes place from one end. The insertion of element onto the stack is called as "push" and deletion operation is called "pop", i.e. when an item is added to the stack the operation is called "push" and when it is removed the operation is called "pop". Due to the push operation from one end, elements are added to the stack, the stack is also known as pushdown list.

The most and least reachable elements in the stack are respectively known as the "top" and "bottom" of the stack. A stack is an arranged collection of elements into which new elements can be inserted or from which existing new elements can be deleted at one end. Stack is a set of elements in a last-in-first-out technique. As per Fig. 2.1 the last item pushed onto the stack is always the first to be removed from the stack. The end of the stack from where the insertion or deletion operation is carried out is called top.
In Fig. 2.1(a) a stack of numbers is shown.

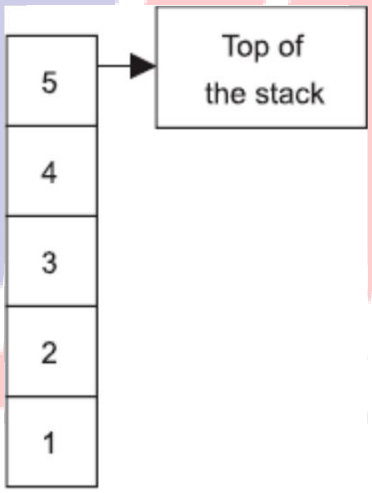
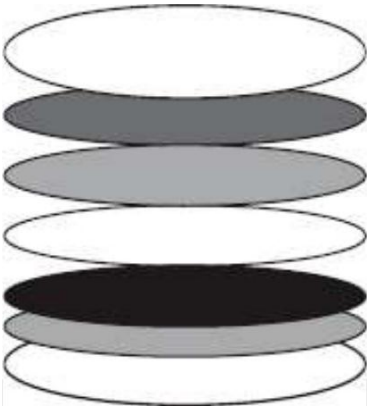


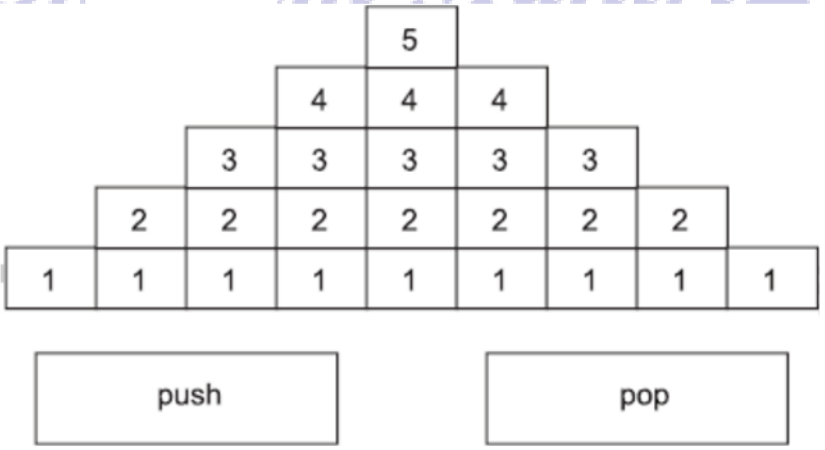
Figure 2.1(b). A pot with plates



As shown in Fig. 2.1(a) the stack is based on the rule last-in-first-out. The element appended lastly is deleted first. If we want to delete element 3 it is necessary to delete the top elements 5 and 4 first.

In Fig. 2.1(b), you can see a pot containing plates kept one above the other. Plates can be inserted or removed from the top. The top plate can be removed first in case pop operation is carried out, otherwise plates are to be added on the top. In other words, the removal operation has to be carried out from the top. Thus, placing or removing takes place from top, i.e. from the same end.

Figure 2.2. Working of stack



As shown in the first half portion of Fig. 2.2, first integer 1 is inserted and then 2,3,4 and 5 are pushed on to the stack. In the second half portion of the same figure, the elements are deleted (pop) one by one from the top. The elements are deleted in the order from 5 to 1. The insertion and deletion operation is carried out at one end (top).
Hence, the recently inserted element is deleted first. If we want to delete a particular element of the stack, it is necessary to delete all the elements present above that element. It is not possible to delete element 2 without deleting elements 5,4 and 3. The stack expands or shrinks with the passage of time. In the above example, the stack initially expands until element 5 is

inserted and then shrinks after removal of elements. There is no higher limit on the number of elements to be inserted in the stack. The total capacity of stack depends on memory of the computer.

In practical life, we come across many examples based on the principle of stack.

Figure 2.3. Stack of books

Fig. 2.3 illustrates the stack of books that we keep in the order. Whenever we want to remove a book the removal operation is made from the top or new books can be added at the top.

STACK-RELATED TERMS

Stack

Stack is a memory portion, which is used for storing the elements. The elements are stored based on the principle of last-in-first-out. In the stack the elements are kept one above the other and its size is based on the memory.

Top

The top of the pointer points to the top element in the stack. The top of the stack indicates its door from where elements are entered or deleted. The stack top is used to verify stack's current position, i.e. underflow, overflow, etc. The top has value 0 when the stack is empty. Some programmers assign -1 to the top as initial value. This is because when the element is added, the top is incremented and it would become zero. The stack is generally implemented with the help of an array. In an array, counting of elements begins from 0 onwards. Hence, on the similar grounds stack top also begins from 0 and it is convenient to assign -1 to top as initial value

Stack Underflow

When there is no element in the stack or stack holds elements less than its capacity, the status of stack is known as stack underflow. In this situation, the top is present at the bottom of the stack. When an element is pushed, it will be the first element of the stack and top will be moved one step up.

Stack Overflow

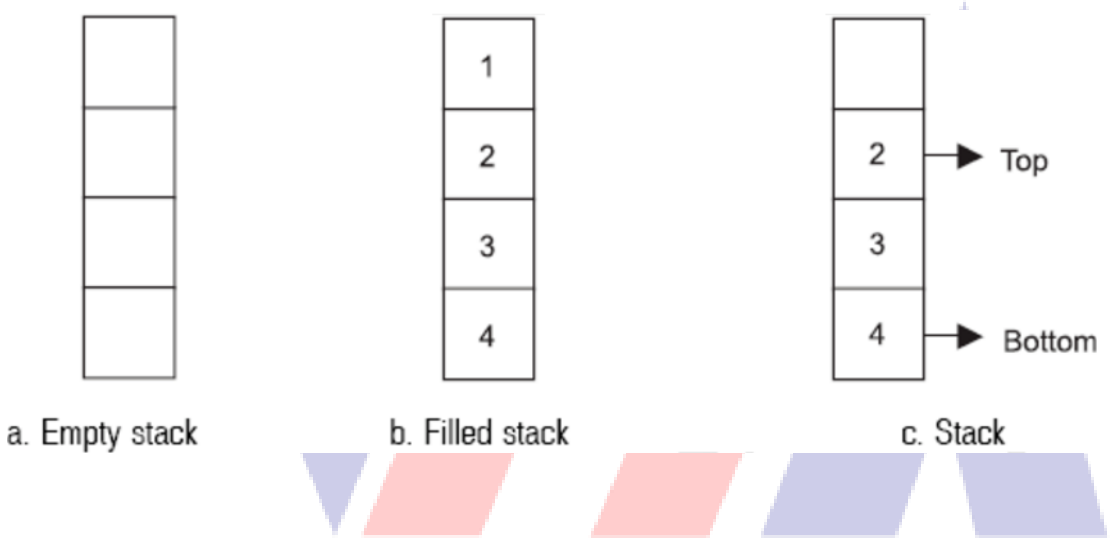
When the stack contains equal number of elements as per its capacity and no more elements can be added, the status of stack is known as stack overflow. In such a position, the top rests at the highest position.

2.2.1 Representation of Stack

The stack is represented in various ways. The stack can be shown as completely empty or fully filled or filled with few elements. When stack has no element, it is called empty stack which is shown in Fig. 2.5(a). The stack with completely filled elements is shown in the Fig. 2.5 (b) and no further elements can be inserted.

A top pointer maintains track of the top elements as shown in Fig. 2.5(c). For empty stack, the top value is zero. When stack has one element the value of top is one. Whenever an element is inserted (push operation) in the stack the value of pointer top is incremented by one. In the same manner, the value of pointer is reduced when an element is deleted from the stack (pop operation). The push operation can be applied to any stack, but before applying pop operation on the stack, it is necessary to make sure that the stack is not empty. If a pop operation is performed on empty stack, it results in underflow.

Figure 2.5. Representation of stack



Stack is very helpful in every ordered and chronological processing of functions. The most useful application of stack is in recursion (explained in previous chapter). It saves memory space. The mechanism of stack last-in-first-out (LIFO) is commonly useful in applications such as manufacturing and accounting calculations. It is a well-known accounting concept.

OPERATION ON STACK

Stack Methods

Create Stack: The fields of the stack comprise a variable to hold its maximum size (the size of the array), the array itself, and a variable top, which stores the index of the item on the top of the stack.

(Note that we need to specify a stack size only because the stack is implemented using an array. If it had been implemented using a linked list, for example, the size specification would be unnecessary.)

The push() method increments top so it points to the space just above the previous top, and stores a data item there. Notice that top is incremented before the item is inserted.

The pop() method returns the value at top and then decrements top. This effectively removes the item from the stack; it's inaccessible, although the value remains in the array (until another item is pushed into the cell).

The peek() method simply returns the value at top, without changing the stack.

The isEmpty() and isFull() methods return true if the stack is empty or full, respectively. The top variable is at –1 if the stack is empty and maxSize-1 if the stack is full.

Figure 2.6(a). Push operation with stack

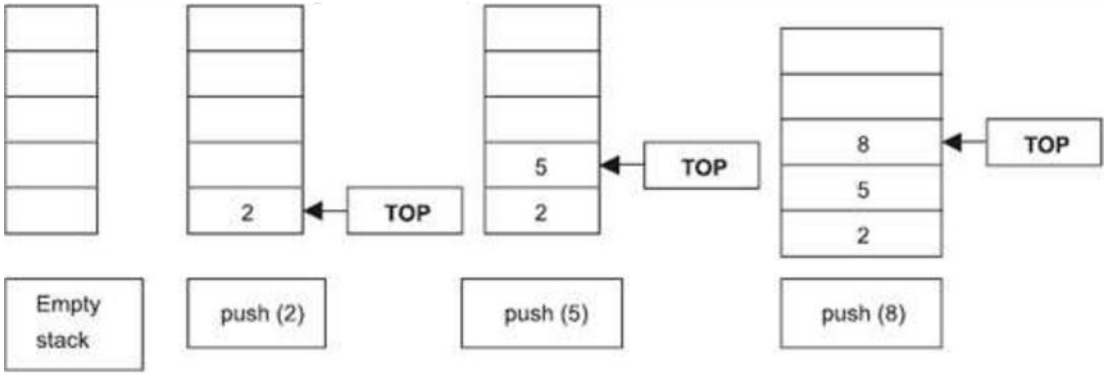
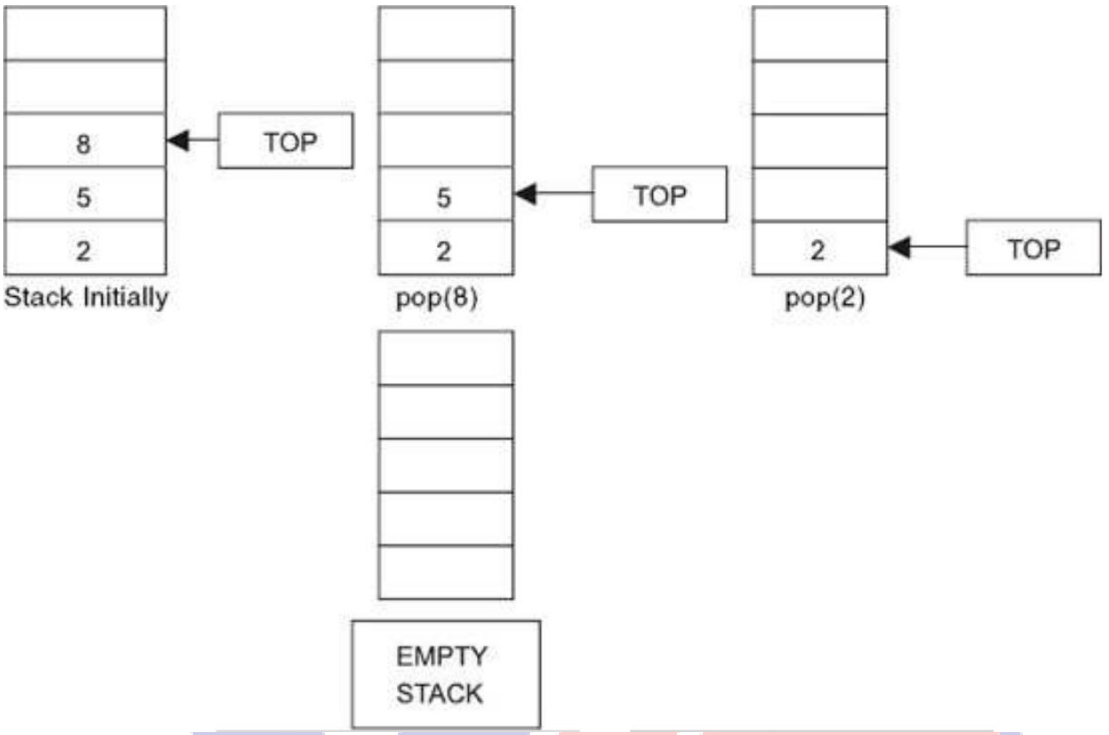


Figure 2.6(b). Pop operation with stack



STACK IMPLEMENTATION

The stack implementation can be done in the following two ways:

1 Static Implementation

Static implementation can be achieved using arrays. Though, it is a very simple method, it has few limitations. Once a size of an array is declared, its size cannot be modified during program execution. It is also inefficient for utilization of memory. While declaration of an array, memory is allocated which is equal to array size. The vacant space of stack (array) also occupies memory space. In both cases, if we store less argument than declared, memory is wasted and if we want to store more elements than declared, array cannot be expanded. It is suitable only when we exactly know the number of elements to be stored.

2 Dynamic Implementation

Pointers can also be used for implementation of stack. The linked list is an example of this implementation. The limitations noticed in static implementation can be removed using dynamic implementation. The dynamic implementation is achieved using pointers. Using pointer implementation at run time there is no restriction on the number of elements. The stack may be expandable. The memory is efficiently utilized with pointers. Memory is allocated only after element is pushed to the stack. Both the above implementations are illustrated with suitable examples in the next section.

REPRESENTATION OF ARITHMETIC EXPRESSIONS

An arithmetic expression contains operators and operands. Variables and constants are operands. The operators are of various kinds such as arithmetic binary, unary for example, + (addition), - (subtraction), *(multiplication), / (division) and % (modular division).

The precedence of operator also plays an important role in expression solving. The precedence of operators is given in Table 2.2.

We have already studied the different stack operations. Now, we will study how stack can be useful in problem solving. Consider a mathematical expression.
5 - *((A * ((B + K)/ (U-4))+K) / 8 (8-2.3))*

The common mistake, which can be made frequently by the programmer, is unbalance of parenthesis. For correct representation of mathematical expression, the following precautions must be taken:

Table 2.2. Precedence of operators

Operators	Precedence	Associativity
+ (unary), - (unary), NOT	6	—
^ (Exponentiation)	6	Right to left
* (multiplication), / (division)	5	Left to right
+ (addition), - (subtraction)	4	Left to right
<,<=,+,<,>,>=	3	Left to right

1. There must be equal number of left and right parenthesis.
2. Each left parenthesis must be balanced by right parenthesis.

INFIX, PREFIX AND POSTFIX NOTATIONS

Stack has various real life applications. Stack is very useful to keep sequence of processing. In solving arithmetic expressions, stacks are used. Stacks are used to convert bases of numbers. In a large program, various functions are invoked by the main () function; all these functions are stacked in the memory. Most of the calculators work on stack mechanism.

Arithmetic expressions can be defined in three kinds of notation: infix, prefix and postfix. The prefixes in, pre and post indicate the relative location of the operator with two operands.

Infix Notation

Expressions are generally expressed in infix notation with binary operator between the operands. The binary operator means the operator requires two operands such as +, *, / and %. In this type, the operator precedes the operands. Following are examples of infix notation:

1. *X+Y*
2. *X+Y*Z*
3. *(X+Y)*Z*
4. *(X+Y)*(P+Q)*

Every single letter (A-Z) and an unsigned integer is a legal infix expression. If X and Y are two valid expressions then (X+Y), then (X-Y) and (X/Y) are legal infix expressions.

Prefix Notation

The prefix notation is also called polish notation. The polish mathematician Lukasiewicx invented it. In this type also the operator precedes the operands. Following are the examples of prefix notation:

1. *+XY*
2. *+X*YZ*
3. **+XYZ*
4. **+XY+PQ*

The postfix notation is also called as reverse polish notation. The operator trails the operand. Following are the examples of postfix notation:

1. *+XY*
2. *XYZ *+*
3. *XY+ Z**
4. *XY+ PQ*+*

EVALUATION OF POSTFIX AND PREFIX EXPRESSION

An algorithm to evaluate postfix expression.

1. Scan the input from left to right.
2. If the number is an operand, push it on the stack.
3. If the number is an operator, pop the two operands from the stack, perform the operation and push the result back on the stack.
4. Continue from step 2 onwards till the end of the input.

An algorithm to evaluate prefix expression.

(Reverrse prreffiix and evalluatte it.)

1. reverse given prefix expression;
2. scan the reversed prefix expression;
3. for each symbol in reversed prefix
4. if operand then push its value onto stack.
5. If the number is an operator, pop the two operands from the stack, perform the operation and push the result back on the stack
6. Continue from step 2 onwards till the end of the input

CONVERSION OF EXPRESSION FROM INFIX TO PREFIX AND POSTFIX

Algorithm to convert Infix Expression to Postfix Expression

Step 1. Push Left Parenthesis "(" onto stack and add right parenthesis ")" to end of the A
Step 2. Scan A from left to right and repeat steps 3 to 6 for each element of A until the stack is empty
Step 3. If an operand is encountered, add it to B
Step 4. If a left parenthesis is encountered push it onto the stack
Step 5. If an operator is encountered then

a. Repeatedly pop from the STACK and add to B each operator (on the top of the stack) which has the same precedence as or higher precedence than operator b. Add operator to STACK

Step 6. If a right parenthesis is encountered, then

a. Repeatedly pop from the STACK and add to B each operator (on the top of STACK) until a left parenthesis is encountered
b. Remove the left parenthesis. (Do not add left parenthesis to B)

Step 7. Exit

Algorithm to Convert Infix to Prefix Form

Suppose A is an arithmetic expression written in infix form. The algorithm finds equivalent prefix expression B.
Step 1. Push ")" onto STACK, and add "(" to end of the A

Step 2. Scan A from right to left and repeat step 3 to 6 for each element of A until the STACK is empty
Step 3. If an operand is encountered add it to B
Step 4. If a right parenthesis is encountered push it onto STACK
Step 5. If an operator is encountered then:
a. Repeatedly pop from STACK and add to B each operator (on the top of STACK) which has same or higher precedence than the operator.
b. Add operator to STACK
Step 6. If left parenthesis is encontered then
a. Repeatedly pop from the STACK and add to B (each operator on top of stack until a left parenthesis is encounterd)
b. Remove the left parenthesis
Step 7. Exit

APPLICATIONS OF STACKS

- Expression conversion--
- Expression evaluation-
- Reversing a string-
- Parsing-
- Well formed parentheses-
- Decimal to Binary Conversion-
- Recursive procedure call-
- We will be studying only two applications:
- Evaluating a postfix expression
- Converting an expression from infix to postfix.

Few applications of stack are narrated together with examples.

Reverse String

We know the stack is based on last-in-first-out rule. It can be achieved simply by pushing each character of the string onto the stack. The same can be popped in reverse fashion. Thus, reverse string can be done. Consider the following program.

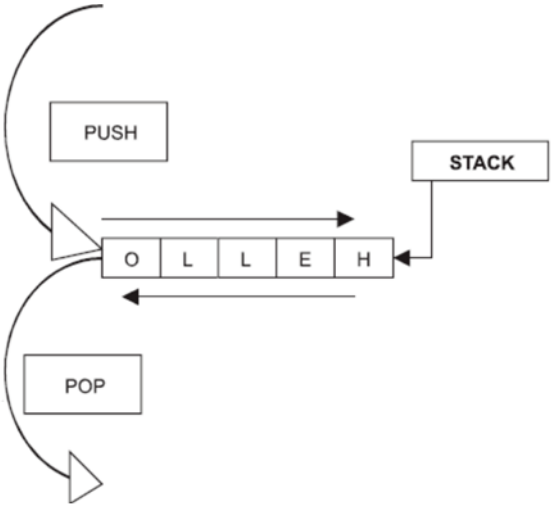
Write a program to reverse the string using stack.

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

char text[40];
void main ()
{
 char ch;
 void push (char,int);
 char pop(int);
 int j=39,k;
 clrscr();
 puts ("\n Enter a string (* to end): ");

while (ch!='*' && j>=0)
```

```
{
 ch=getche();
 push (ch,j);
 j--;
}
k=j;
j=0;

printf ("\n Reverse string is: ");

while (k!=40)
{
 ch=pop(k);
 printf ("%c",ch);
 k++;
}
}
void push (char c, int j)
{
 if (c!='*')
 text[j]=c;
}

char pop (int j)
{
 char c;  c=text[j];
 text[j]=text[j+1];
 return c;
}
```

OUTPUT
Enter a string (* to end):
HELLO*
Reverse string is: OLLEH

Figure 2.11. Reverse string



2.10.3 Conversion of Number System

Suppose, one wants to calculate binary number of a given decimal number, the given number is repeatedly divided by 2 until 0 is obtained. The binary number can be displayed in reverse order using stack rule last-in-first-out (LIFO). Consider the following program:

Example 2.13.

Write a program to convert a given decimal number to binary. Explain the role of stack mechanism.

Solution-

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


int num[7];

main ()
{
  int n,k,T=7;
  void show();
  void push(int,int);
  clrscr();
  printf ("\n Enter a number: ");
  scanf("%d",&n);
  printf ("\n The binary number is: ");
  while (n)
  {
  k=n%2;
  push(--T,k);
  n=n/2;
  }

show();
}

void push (int j, int b)
```

```
{
 num[j]=b;
}
void show ()
{
 int j;
 for (j=0;j<7;j++)
 printf (" %d ",num[j]);
}
```

OUTPUT
Enter a number: 9
The binary number is: 0 0 0 1 0 0 1

Explanation In this program the binary equivalent is obtained by repeatedly dividing by two. Here, the first binary digit obtained is pushed on the stack. This process is continued. The show () function displays the binary digits stored in the stack, i.e. an array.

2.10.4 Recursion

The recursion is the fundamental concept of the mathematical logic. Recursion is one of the important facilities provided in many languages. C also supports recursion. There are many problems, which can be solved recursively. The loop which performs repeated actions on a set of instructions can be classified as either iterative or recursive. The recursive loop is different from the iterative loop. In the recursion, the procedure can call itself directly or indirectly. In the directly called recursion the procedure calls itself repeatedly. On the other hand, if the procedure calls another procedure then it is an indirect recursion. In recursion, some statements, which are specified in the function, are executed repeatedly. Every time a new value is passed to the recursive function till the condition is satisfied. A simple programming example is given below.

Example 2.14.

Write a program and find the greatest common divisor of the given two numbers.

```
# include <stdio.h>
# include    <conio.h>
int stack[40],top=-
1; main ()
{
 void gcd(int,int);
 int n1,n2; clrscr();
 printf("\nEnter number:- ");
 scanf("%d",&n1);
 printf("\nEnter number:- ");
 scanf("%d",&n2);
 gcd(n1,n2);
 printf("\nThe gcd of %d & %d is:- %d",n1,n2,stack[top]);
}

void gcd(int a,int b)
```

```
{
if(a!=b)
 {
 if(a>b)
 {
 top++;
 stack[top]=a-b;
 printf("\nTop value is:- %d",stack[top]);
 gcd(a-b,b);
 }
else
 {
 top++;
 stack[top]=b-a;
 printf("\nTop value is:- %d",stack[top]);
 gcd(a,b-a);
 }
 }
}

OUTPUT:
Enter number:- 5
Enter number:- 25
Top value is:- 20
Top value is:- 15
Top value is:- 10
Top value is:- 5
The gcd of 5 & 25 is:-5
```


Example 2.15.

Write a program to convert decimal to binary by using the concept of recursion.

```
#include <stdio.h>
# include <conio.h> int
stack[40],top=-1; main
()
{
 void binary(int);
 int no; clrscr();
 printf("\nEnter number:-
 "); scanf("%d",&no);
 binary(no);

 printf("\nThe binary of the given number is:-");

 while(top>=0)
 {
```

```
 printf(" %d ",stack[top]);
 top--;
 }

}

void binary(int b)
{
  if(b>0)
  { top++;
  stack[top]=b%2;
  binary(b/2);
 }
}
```

OUTPUT:
Enter number:- 255
The binary of the given number is:- 1 1 1 1 1 1 1 1