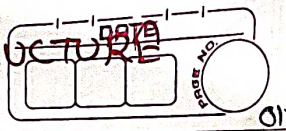
1. INTRODUCTION TO DATA - STRUCTURE



01st lecture

*. What is data structures

If the data is very huge, we need to organise that data properly, so that we can use access it efficiently and the mechanism which is used for arranging such data is called as "data-structures".

* Types of data structures:

Following are two types of data structures.

1. PRIMITIVE DATA STRUCTURE:
It contains those data types which are already created by programming language.
example, int, float, char, etc.

2. NON-PRIMITIVE DATA STRUCTURE:
It contains these data structures which are created by combining different primivitive data structures.
eg. it has 2 types.

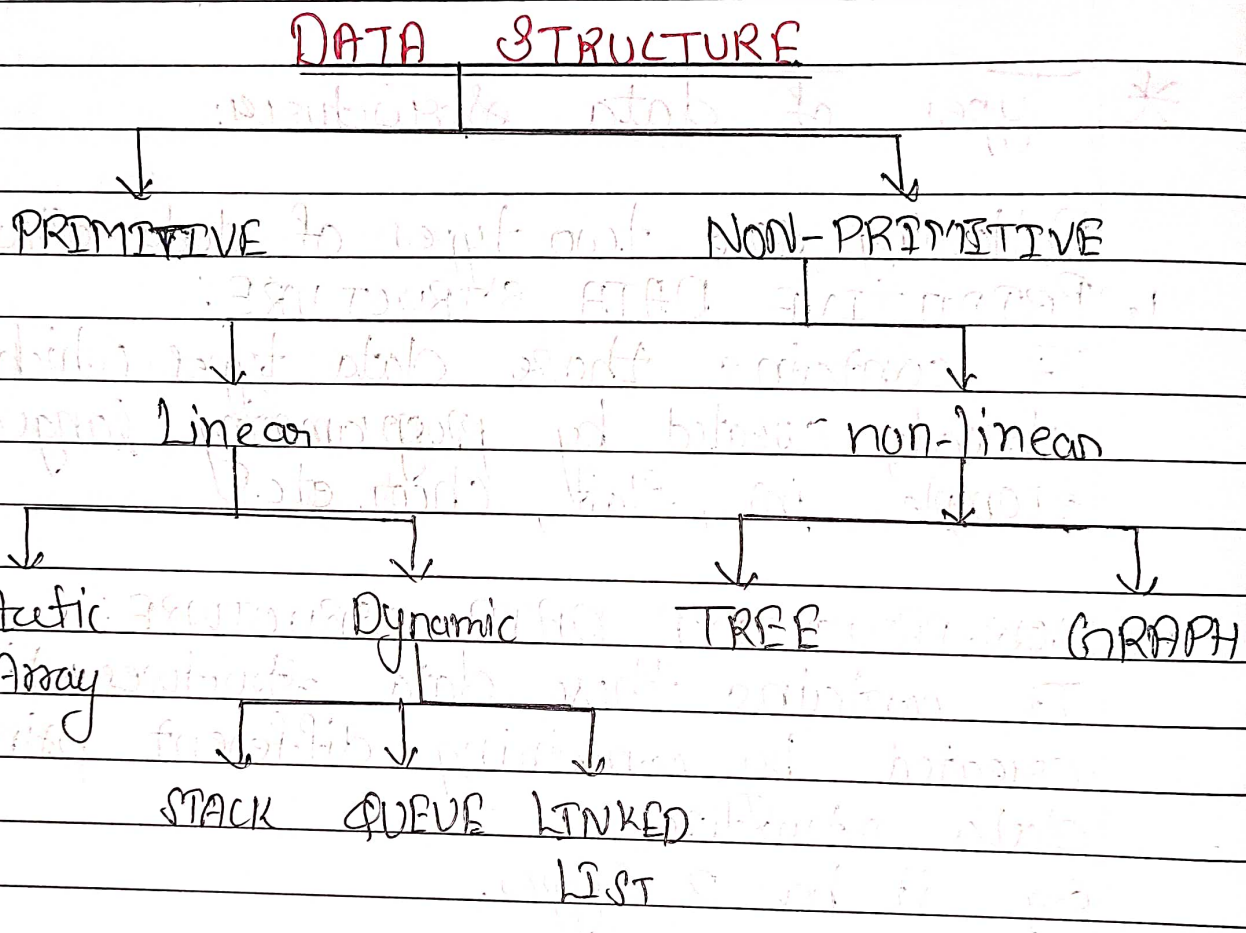
(a) Linear data structure:
If the data arranged in linear or in sequential manner, then it is called as linear data structure. (it has 2 types).

(i) Static data structure:
It contains data structures which has fix memory size
eg. array.

(ii) Dynamic data structure:
It contains those data structures, whose size can be dynamically de updated during run time. eg. stack, queue, linked list, etc.

(b) Non-linear data structures:

If the data is arranged in non-linear or hierachial manner, then it is called as linear data structure.



02 | Sept
Thursday

DATE
PAGE NO.

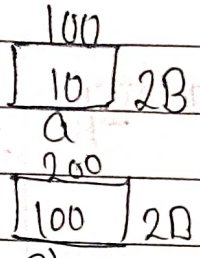
02nd lecture

* Operations on data structures.

Operations	Description
1. Traversing	In this we can traverse or move from one element to another element in data structure.
2. Searching	In this, we can check whether elements in data structure is present or not.
3. Insertion	In this, we can add or insert new element in data structure.
4. Deletion	In this we can delete or remove existing element from data structure.
5. Sorting	In this, we can arrange elements of data structure in ascending or descending order.
6. Merging	In this, we can join or merge 2 or more data structure.

* POINTERS *

Variables which is used to store address of another variable.



For eg.

```
int a=10
int *ptr
ptr = &a
printf ("%d", a); //10;
printf ("%u", &a); //100,
printf ("%u", ptr); //100
printf ("%u", &ptr); //200
printf ("%d", *ptr); // 10

*ptr = *ptr +10
printf ("%d", *ptr); //20
```

Address format specifier
%u, %x, %p

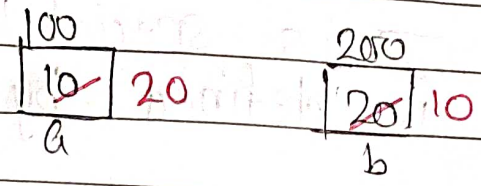
*ptr = ptr ke andar jo value hai, wo jis ka address hai wo value.

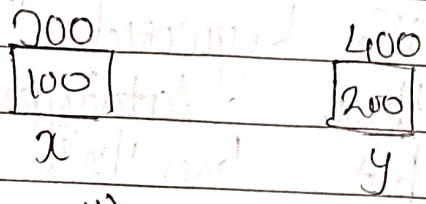
NOTE:

1. Call by Value: Whatever changes made on formal parameters, will not get reflected on actual parameters.

2. Call by Reference: Whatever changes made on formal parameters, will get reflected on actual parameters.

* CALL BY REFERENCE:






```
#include<stdio.h>
#
int main()
{
   int a, b;
   printf("Enter 2 numbers= ");
   scanf("%d %d", &a, &b);
Func call → swap(&a, &b); // Call by reference
   printf("a=%d", a);
   printf("b=%d", b);
   return 0;
}

Actual Parameters.

Func defin  void swap(int *x, int *y)
{
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
   printf = ("x=%d", *x);
   printf("y=%d", *y);
}

formal Parameters
```

Output

x=20 a=20

y=10 b=10.

2. STACK AND QUEUE

STACK:

* **Abstract DATA TYPE:**

It is special kind of datatype, whose behavior is defined by a set of operations.

The keyword "Abstract" is used as we can use these datatypes, to perform different operations but how those operations are working that is totally hidden from the user.

Example of ADT of Stack:

A. PUSH Operation.

Let 's' is the stack with size 'N' and 'x' be the element inserted in a stack.

```
if (top == N-1)
    print ("stack overflow");
else
    top = top + 1
    s[top] = x
```

B. POP operation.

Let 's' is the stack with size 'N'.

```
if (top == -1)
    printf("stack Underflow")
else
    x = s[top]
    top = top - 1
    print "x as deleted element".
```

STACK:

(i) A stack is a linear data structure that follows the LIFO (Last-In-First-Out) principle

(ii) It contains only one pointer TOP pointing to the topmost element of the stack.

(iii) Whenever an element is added in the stack, it is added on the top of the stack, and only the top element can be deleted from the stack.

(iv) Insertion of new element into stack is called as PUSH operation, whereas deletion of existing element from stack is called as POP. POP Operation.

(v) PEEK operation represents retrieving topmost element from stack.

(vi) While pushing element into stack we need to check "STACK OVERFLOW" condition and while popping element from stack we need to check "STACK UNDERFLOW" condition.

06/09/21
Monday

OS lecture

Program 1:

1. WAP to implement stack using array to perform full operation on stack.

a) Push new element into stack.
b) Pop existing element from stack.
c) Peek topmost element from stack.
d) Display contents of stack.

```
#include<stdio.h>
#define N 20
typedef struct stack
{
    int a[N];
    int top;
}stack;

void push(stack *s, int x)
{
    if (s->top==N-1)
        printf("\n Stack Overflow...");
    else
    {
        s->top = s->top +1;
    s->a[s->top]= x;
    }
}

int isempty(stack *s)
{
    if (s->top==-1)
        return -1;
    else
        return 0;
}
```

```
int pop(stack *s)
{
    int x;
    if (isempty(s))
    {
        return -1;
    }
    else
    {
        x = s->a[s->top];
        s->top = s->top-1;
        return x;
    }
}

int peek(stack *s)
{
    if (isempty(s))
        return -1;
    else
        return s->a[s->top];
}

void display(stack *s)
{
    int i;
    if (isempty(s))
    {
        printf("\n stack is empty....");
    }
```

```
    else
    {
        for (i = s->top ; i>=0 ; i--)
        {
            printf ("\n %d", s->a[i]);
        }
    }
}

int main()
{
    int ch, x;
    stack s;
    s.top = -1;
    while(1)
    {
     printf("\n 1: Push \n 2: POP \n 3: Peek
            \n 4: Display \n 5: Exit \n Enter choice:
    if(ch == 5)
    break;
    switch (ch)
    {
     Case 1;
      {
       printf ("\n Enter element to be pushed");
       scanf ("%d", &x);
       push (&s, x);
      }
     break;
```

```
case 2:
  {
    x = pop(&s);
    if (x == -1)
      printf("\n Stack Underflow...");
    else
      printf("\n Popped Element = %d", x);
  }
  break;
case 3:
  {
    x = peek(&s);
    if (x == -1)
      printf("\n Stack is empty");
    else
      printf("\n Stack top element = %d", x);
  }
  break;

case 4:
  {
    display(&s);
  }
  break;
default:
  {
    printf("Invalid choice...");
  }
}
return 0;
}
```

* Application of stack:

1. Reversing a string:

While reversing a string we can push different of string into stack and after pushing all character, we start popping every character from stack one by one, to get reverse of string.

Example:

eg. S1 = "abcd"

d
c
b
a

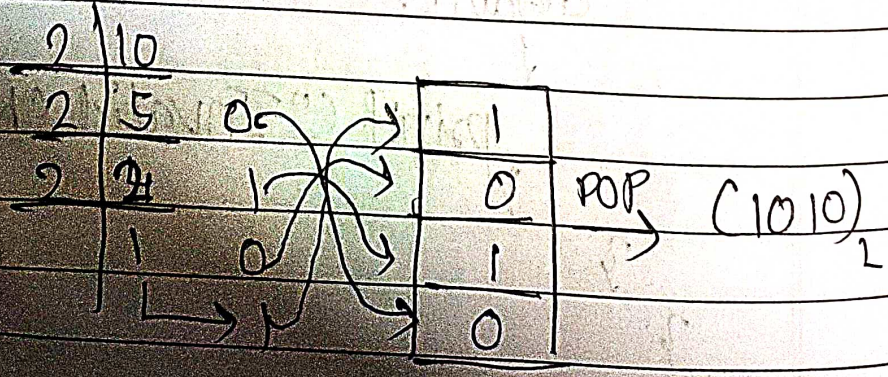
POP → "dcba"

2. Number conversion:

Stack is also used to convert decimal number to bincary or octal or hexadecimal. In that case we need to store all reminders into stacks and at the end & pop all reminders from stack to get appropriate converted value to decimal value.

Example: Decimal → Binary

$a == (10)_{10}$



3. Recursion:

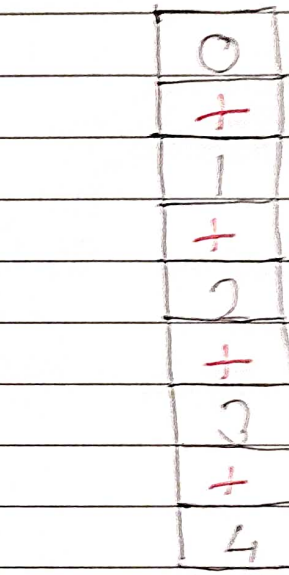
When a function calls itself repeatedly, it is called as "Recursion".

Recursive function use something called "the call stack". When a program calls a function, that function goes on top of the call stack. Then, when termination condition meets the values take off from top of stack.

Example:

```
#include<stdio.h>

int sum(int);
int main()
{
   int n, s;
   printf("\n Enter n=");
   scanf("%d", &n);
   s= sum(n);
   printf("\n Sum= %d", s);
   return 0;
}
int sum(int n)
{
  if (n==0)
    return 0;
  else
  return (n + sum(n-1));
}
```



0+1+2+3+4=10

4 + sum(3);
3 + sum(2);
2 + sum(1);
1 + sum(0);
0