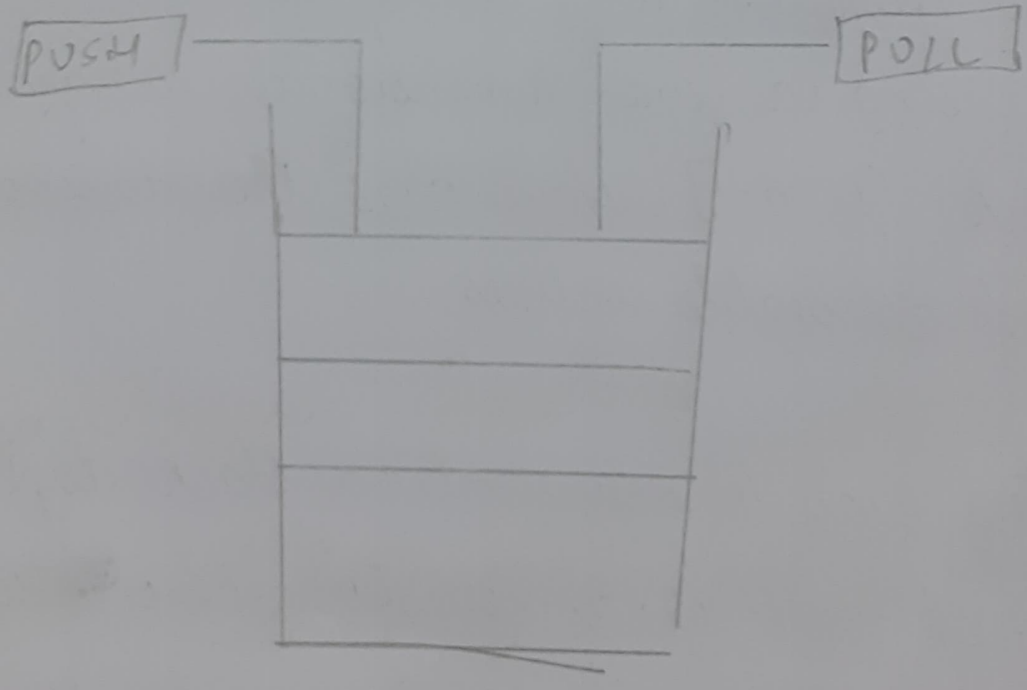
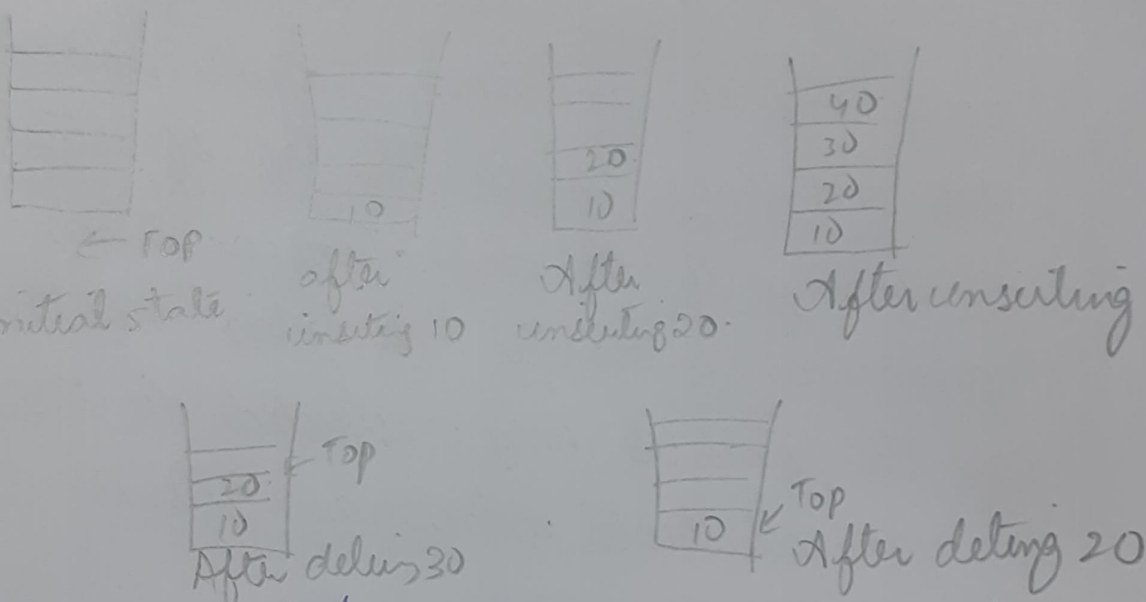
1) Define Stack

A Stack can be defined as an ordered collections of an items may be added at the end only, called the top of stack. Awording to the defination a Stack is dynamic, constantly changing object where as to items is only restricted to the top we can illustrate Stack as follows.



Here, if any new items are added, they will be placed on top of the stack, and if any item is deleted it as last in the list of (LIFO) Operation.



The simplest way to represent a stack is by using a one-dimensional array say stack [0:n-1] where n is the maximum no of allowable entries.

Another way to represent a stack by using links. A node is a collection of data & link operation. A stack can be represent using nodes with two fields, possibly cal

data and link.

→ Stack operations.

- Insertion Operation is called push
- Deletion operation is called pop

2) Write the structure to create node for doubled linked list.

NODE CREATION:-

```
{
  struct node *prev;
  int data;
  struct node *next;
};
  struct node * head;
```

Long Answer Questions

3). Write stack program using linked

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

```
opnd1 = pop(&opndstk);
value = oper(c, opnd1, opnd2);
push(&opndstk, value);
}
return(pop(&opndstk));
}

double oper(int symb, double op1, double op2
{
switch(symb)
{
case '+': return(op1 op2);
case '-': return(op1-op2);
case '*': return(op1*op2);
case '/': return(op1/op2);
case '$': return(pow(op1, op2));
default: printf("illegal operations");
exit(1);
}
}
}
void push(struct stack *ps, double x)
```

```
printf ("\n Result = %lf", eval (expr));
getch ();
}
struct stack {
 int top;
 double items [MAXCOLS];
};
double eval (char expr[]);
{
 int c, position;
 double operand1, operand2, value;
 struct stack operandstk;
 operandstk. top = -1;
for (position = 0; (c = expr[position]) !
    = '\0'; position++).
if (isdigit(c)).
push (&operandstk, (double) (c - '0'));
else
{
opnd2 = pop (&opndstk);
```

4) Write a C program to Evaluate postfix expression?

```
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#define MAXCOLS 80
double eval (char []);
double pop (struct stack *);
void push (struct stack *, double);
double oper (int, double, double);
void main()
{
  char expr [MAXCOLS];
  int position = 0;
  clrscr ();
  printf ("enter expression : \n");
  gets (expr);
  printf ("%s%s", "the original postfix
          expression is ", expr);
```

```
int pop() {
int n;
node * c;
 if (top != NULL) {
 c = top;
 top = top -> link;
 n = c -> data;
 free (c);
 return n;
  }
return 0;
 }
 void display() {
 node * c;
 c = top;
while (c != NULL) {
printf ("%d\t", c -> data);
 c = c -> link;
  }

}
```

```
        switch (ch)5
        push (n);
        break,

case 2:
        printf ("Remove element : %d", pop(1),
        break;
case 3:
        display ();
        break;

case 4:
        exit(0);
          }
    }
     return 0;
    }
    void push (int n){
    node* c;
  c = (node*) malloc (sizeof (node));
  c→data = n;
 c→link = top;
  top = c;
  }.
```

```
struct stacknode *{
    int data;
    struct stacknode * link
    };
    typdef struct stacknode node;
    node * top;
    void push (int);
    int pop ( );
    void display ( );
int main ( ) {
int ch, x;
top = NULL;
while (1) {
printf ("\n MENU \n1. push \n2. pop \n3. display
        \n4. exit \).
printf( "enter your choice:");
scanf ("%d", &ch);
switch (ch) {
    case 1:
    printf ("enter element:");
    scanf ("%d", &n);
```

```
{
if (PS -> top == MAXCOLS - 1)
printf ("Stack Overflow");
else
PS -> items [++(PS -> top)] = x;
}
double pop (struct stack * PS).
{ if (PS -> top == -1).
{
printf ("Stack underflow");
exit (S);
}
else
return (PS -> items [PS -> stop--]);
}.
```

Q1) What is a hash function ?

A). * A fixed process converts a key to hash key is known as hash function.

* When a key range is too large to user, the ideal method is division method.

* The Division method is the most common hash function

* The Division method is :

$$f(K) = K \% D.$$

* where,

K → key.
D → size of hash table.
% → modulo operator.

* The positions in the hash table are 0 through D-1.

* Each position in the hash table are indexed. 0 through D-1.
* Each position is called as the "bucket"

2). Define Dictionary Data Structure.

* It is defined as a general purpose data structure for storing a group of objects.
* A dictonary is associated with a set of keys and each key has a single assoiated value.
* When it is presented with a key, the dictionary will simply return the associated value.

Example:

The Students list in the classroom could b represented as dictionary with students roll no and name.

where, roll no is key.
name is value.

Students = {

501 : "Kiran",
502 : "Raju",
503 : "Praveen" };

Long Answer Questions

Q3) What is hash function? Explain the type of hash functions

A). * A fixed process converts a key to a h key is known as hash function.

* When a key range is too large to use ideal method is used.
* The most common hash function is division method
* The Division method is.

$$f(K) = K \% D.$$

K → key
D → Size of hash table.
% → Modulo operator.

* The positions are indexed 0 through D-1.
* Each position is called as "bucket"
* Different types of hash functions:-
    - ⇒ Mid Squared hash function
    - ⇒ Division Hash function
    - ⇒ Folding Hash function

* Mid Squared Hash Function.
* Mid Squared hashing is a hashing technique in which unique keys are generat
* In this technique, an initial Seed valu is taken and is squared.
* Some digits from the middle are extrac
* These extracted digits form a no. whic is taken as new seed.
* Let us take 4536 as seed, its squared value is 20575296.
* Take the middle 4 digits are new seed ie

Its squared value is 33085504

* Take the middle 4 digits as new seed i.e 0855
* Repeat this process

* Division Hash function

This is the easiest method to create a hash function

The hash function is :-

$$h(K) = K \% m$$

where,

$K \rightarrow$ key

$m \rightarrow$ size of hash table.

$\Rightarrow$ hash table size is 11 and key is 321

$\Rightarrow h(321) = 321 \% 11$

$\Rightarrow h(321) = 2$

* Folding hash function.

* The key k is partitioned into a number of parts $k_1 k_2 \ldots k_n$ where each part except possibly the last has the same no. of digits as the required address.

⇒ key = 9246162517.

⇒ h(key) = 92+46+16+25+17.

⇒ h(key) = 196.

2). Write a C program to implement dictionary ADT.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
Struct Dic
{
int key;
```

```
char value [20];
struct Qui *link;
};
type def struct qui Node;
Qui Node * start;
void insert (int, char*);
void remove key (int);
void display ();
char * search (int);
int main () {
char ch;
int key
char value [20];
start = NULL;
while (1) {
print ("\n MENU\n! is not \n 2. Sea
n3. delete \n 4. display \n 5. exit \n");
print f ("Enter your choice:");
Scan f ("%d", &ch);
```

```
case 1
printf ("Enter key and value : ");
scanf ("%d %s", &key value);
insert (key value);
break;

case 2:
    printf ("Enter key for search : ");
    scanf ("%d", &key);
    printf ("value %s", search (key));
    break;

case 3:
    print f ("Enter key to delete:");
    scanf ("%d", &key);
    remove key (key);
    break;

case 4:
    display ();
    break;
```

```
case 5:
    exit(0);
}
return 0;
}
void main insert (int key, char value:){
Dii Node *c;
int flag = 0;
c = Start;
if (Start == NULL){
Start = (Dii Node*) Malloc (Size of (Dii
Start → key = key;
Strcpy (Star → value, value);
Start → link = NULL;
}
else {
while (c => link != NUL){
if (c → key == key){
Strcpy (c → value, value);
```

```
flag = 1;
break;
}
c = c -> link;
}
if (flag == 0) {
c -> link = (Dll Node *) Malloc (Dll Node));
c -> link -> key = key;
strcpy (c -> link -> value, value);
c -> link -> link = NULL;
}
}
}
void main remove key (int key) {
Dll Node * c, * pre;
c = start;
while (c != NULL) {
if (c -> key == key) {
if (c == start) {
```

```
start = start -> link;
free (c);
break;
}
else {
pre -> link = c -> link;
free (c);
break;
}
pre = c;
c = c -> link;
}
}
char * search (int key) {
Qui Node * c;
int flag = 0;
c = start;
while (c != NULL) {
if (c -> key == key) {
return c -> value;
}
```

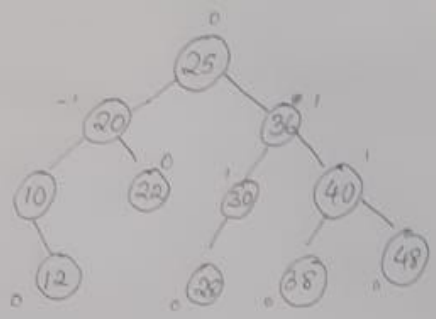
```
void display () {
Qui Node * c
c = start;
while (c != NULL)
printf ("%d %
c -> key, c -> v
c = c -> link;
}
} //
```

1) Define an AVL tree.

*AVL (Adelson, Velski & Landis) tree is a height balanced binary search tree. That means, an AVL tree is also a binary search tree but its a balanced tree

* A binary tree is said to be balanced if the difference between the heights of the left & right subtree of every node is either 0, -1 or +1.

* In an AVL tree, every node maintains extra information known as the balance factor



2) What is a Splay tree?

* Splay tree is another version/variant of binary tree
* In a Splay tree, recently accessed element is placed at the root of the tree
* Splay tree is self adjusted BST in which every operation on elements rearranges the tree so that the element is placed at root position

* All operations in a splay tree are combined with a common operation called "splaying"
* Splaying an element is the process of bringing it to the root position by performing suitable rotation operations

3). Construct the BST for following elements
{30, 15, 35, 20, 10, 45, 35, 15, 10, 50}

A. The BST will be constructed as explained below

i) Insert 30

(30)

ii) Insert 15.

15 < 30, so insert 15 to the left of 30

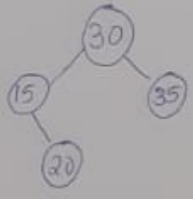
iii) Insert 35

35 > 30, so insert 35 to the right of 30

(iv) Insert 20

20 < 30, so insert 20 to the left of 30

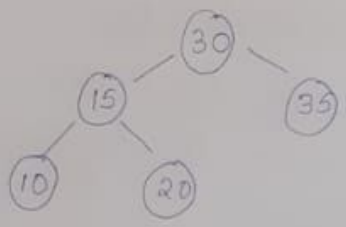
20 > 15, so insert 20 to the right of 15.



(v) Insert 10

10 < 30, so insert 10 to the left of 30.

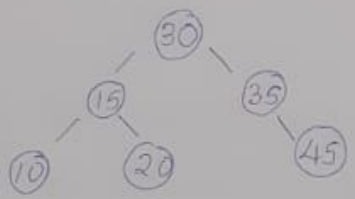
10 < 15, so insert 10 to the left of 15.



vi) Insert 45

45 > 30, so insert 45 to the right of 30,

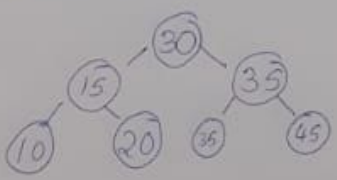
45 > 35, so insert 45 to the right of 35



vii) Insert 35

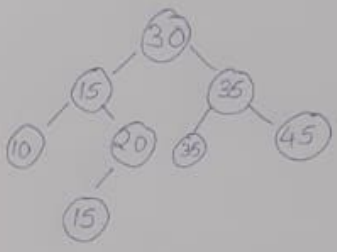
35 > 30, so insert 35 to the right of 30,

35 ≤ 35, so insert 35 to the left of 35



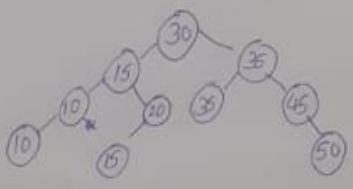
viii) Insert 15

15 < 30, so insert 15 to the left of 30
15 ≥ 15, so insert 15 to the right of 15
15 < 20, so insert 15 to the left of 20



(ix) Insert 10,
10 < 30, so insert 10 to the left of 30,
10 < 15, so insert 10 to the left of 15,
10 ≤ 10, so insert 10 at left most node

(x) Insert 50
50 < 30, 50 > 305, 50 > 45, insert 50 at right most node

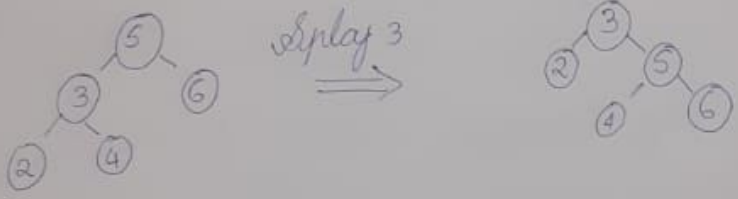


This is the required BST

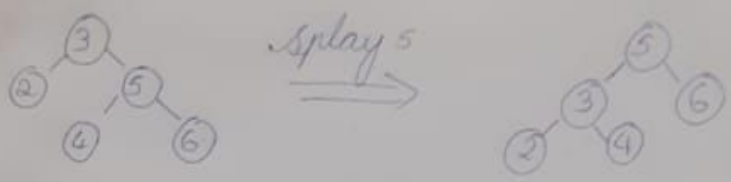
4) Explain the rotation in Splay tree

A) We use the following operations on a Splay tree.

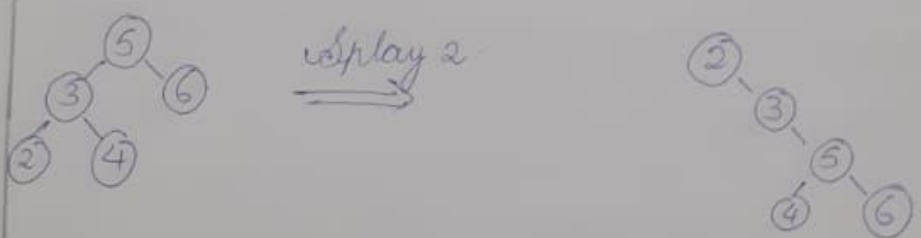
(i) Zig rotation : The Zig rotation in a Splay tree is similar to single right rotation in AVL tree. In Zig rotation, every node moves one position to the right



(ii) Zag Rotation : The Zag Rotation in Splay tree is similar to single left rotation in an AVL tree. In Zag rotation, every node moves one position to the left

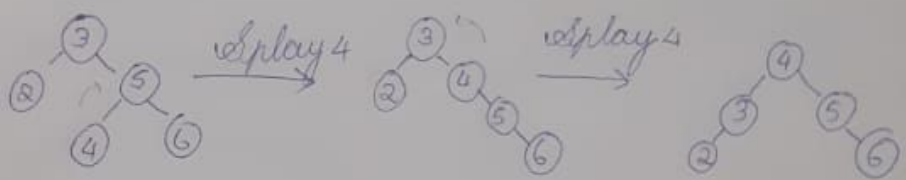


(iii) Zig-Zig Rotation: This is double Zig Rotation. In Zig-Zig Rotation, every node moves two positions to the right



(iv) Zag-Zag Rotation: This is double Zag Rotation

v) zig-zag Rotation: The zig-zag rotation in a splay tree is a sequence of zig rotation followed by zag Rotation. In this, every node moves one position to the right followed by one position to the left



vi) zag-zig rotation: The zag-zig rotation in a splay tree is a seq of zag rotation followed by zig rotation. In this, every node moves one position to the left followed by one position to right.