List of programs

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Name of the program | Date | Faculty sign |
| 1. | Write a C++ programs to implement 1) Linear search 2) Binary search |  |  |
| 2. | Write a C++ programs to implement i) Bubble sort ii) Selection sort iii) quick sort iv) insertion sort |  |  |
| 3. | Write a C++ programs to implement the following using an array.  a) Stack ADT b) Queue ADT |  |  |
| 4. | Write a C++ programs to implement list ADT to perform following operations a) Insert an element into a list.   1. Delete an element from list 2. Search for a key element in list 3. count number of nodes in list |  |  |
| 5. | Write C++ programs to implement the following using a singly linked list.Stack ADT b) Queue ADT |  |  |
| 6 | Write C++ programs to implement the deque (double ended queue) ADT using a doubly linked list and an array. |  |  |
| 7 | Write a C++ program to perform the following operations:   1. Insert an element into a binary search tree. 2. Delete an element from a binary search tree. 3. Search for a key element in a binary search tree. |  |  |
| 8 | Write C++ programs for implementing the following sorting methods:Merge sort b) Heap sort |  |  |
| 9 | .Write C++ programs that use recursive functions to traverse the given binary tree in a) Preorder b) inorder and c) postorder. |  |  |
| 10 | Write a C++ program to perform the following operations  a) Insertion into a B-tree b) Deletion from a B-tree |  |  |
| 11 | Write a C++ program to perform the following operations a)Insertion into an AVL-tree b) Deletion from an AVL-tree |  |  |
| 12 | Write a C++ program to implement all the functions of a dictionary (ADT) |  |  |

**Data Structures using C++ LAB FA23**

Week 1: write a C++ programs to implement recursive and non recursive i) Linear search ii) Binary

Aim: To implement Linear search and binary search

**Description:**

1. **LINEAR SEARCH (SEQUENTIAL SEARCH):** Search begins by comparing the first element of the list with the target element. If it matches, the search ends. Otherwise, move to the next element and compare. In this way, the target element is compared with all the elements until a match occurs. If the match do not occur and there are no more elements to be compared, conclude that the target element is absent in the list.

For example, consider the following list of elements.

#### 5 9 7 8 11 2 6 4

To search for element 11(i.e Key element = 11). first compare the target element with first element in list i.e. 5. Since both are not matching we move on the next elements in the list and compare. Finally found the match after 5 comparisons.

**Algorithm for Linear search**

Linear\_Search (A[ ], N, val , pos ) Step 1 : Set pos = -1 and k = 0 Step 2 : Repeat while k < N

Begin

Step 3 : if A[ k ] = val

Set pos = k print pos Goto step 5

End while

Step 4 : print “Value is not present” Step 5 : Exit

#### Source code: C++ program for Linear search

#include<iostream> using namespace std;

int Lsearch(int list[ ],int n,int key); int main()

{

int n,i,key,list[25],pos;

cout<<"enter no of elements\n"; cin>>n;

cout<<"enter "<<n<<" elements "; for(i=0;i<n;i++)

cin>>list[i]; cout<<"enter key to search"; cin>>key;

pos= Lsearch (list,n,key); if(pos==-1)

cout<<"\nelement not found";

else

}

cout<<"\n element found at index "<<pos;

# Data Structures using C++ LAB FA23

/\*function for linear search\*/

int Lsearch(int list[],int n,int key)

{

int i,pos=-1;

for(i=0;i<n;i++) if(key==list[i])

{

}

return pos;

}

pos=i; break;

Results

# Data Structures using C++ LAB FA23

1. **Binary Searching**: Before searching, the list of items should be sorted in ascending order. First compare the key value with the item in the mid position of the array. If there is a match, we can return immediately the position. if the value is less than the element in middle location of the array, the required value is lie in the lower half of the array.if the value is greater than the element in middle location of the array, the required value is lie in the upper half of the array. We repeat the above procedure on the lower half or upper half of the array. Algorithm:

Binary\_Search (A [ ], U\_bound, VAL)

Step 1 : set BEG = 0 , END = U\_bound , POS = -1 Step 2 : Repeat while (BEG <= END )

Step 3 : set MID = ( BEG + END ) / 2

Step 4 : if A [ MID ] == VAL then POS = MID

print VAL “ is available at “, POS GoTo Step 6

End if

if A [ MID ] > VAL then set END = MID – 1

Else

set BEG = MID + 1

End if End while

Step 5 : if POS = -1 then

print VAL “ is not present “ End if

Step 6 : EXIT

#### Source code: Non recursive C++ program for binary search

#include<iostream> using namespace std;

int binary\_search(int list[],int key,int low,int high); int main()

{

int n,i,key,list[25],pos;

cout<<"enter no of elements\n" ; cin>>n;

# Data Structures using C++ LAB FA23

cout<<"enter "<<n<<" elements in ascending order "; for(i=0;i<n;i++)

cin>>list[i];

cout<<"enter key to search" ; cin>>key; pos=binary\_search(list,key,0,n-1); if(pos==-1)

cout<<"element not found" ;

else

}

cout<<"element found at index "<<pos;

/\* function for binary search\*/

int binary\_search(int list[],int key,int low,int high)

{

int mid,pos=-1;

while(low<=high)

{

mid=(low+high)/2; if(key==list[mid])

{

pos=mid; break;

}

else if(key<list[mid])

high=mid-1;

else

}

low=mid+1;

return pos;

}

Results

# Data Structures using C++ LAB FA23

Assignment :-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | date | sign | Remarks |
| 1.Write a program to find an element in the list of elements using linear and binary search non recursively ,provide a provision to select between linear and binary searching. |  |  |  |
| 2. Write a program to find an element in the list of elements using linear and binary search recursively, provide a provision to select between linear and binary searching. |  |  |  |
| 3. write a program to implement searching using linked list |  |  |  |
| 4.Submit an analysis report of merits and demerits of linear and binary searching. |  |  |  |

# Data Structures using C++ LAB FA23

Week 2: write a C++ programs to implement i) Bubble sort ii) Selection sort iii) quick sort iv) insertion

**Aim:** To implement i) Bubble sort ii) Selection sort iii) Quick sort iv) Insertion sort

Description:

#### Bubble sort

The bubble sort is an example of exchange sort. In this method, repetitive comparison is performed among elements and essential swapping of elements is done. Bubble sort is commonly used in sorting algorithms. It is easy to understand but time consuming i.e. takes more number of comparisons to sort a list . In this type, two successive elements are compared and swapping is done. Thus, step-by-step entire array elements are checked. It is different from the selection sort. Instead of searching the minimum element and then applying swapping, two records are swapped instantly upon noticing that they

are not in order.

#### ALGORITHM:

**Bubble\_Sort ( A [ ] , N )**

Step 1: Start

Step 2: Take an array of n elements Step 3: for i=0,………….n-2

Step 4: for j=i+1,…….n-1

Step 5: if arr[j]>arr[j+1] then Interchange arr[j] and arr[j+1] End of if

Step 6: Print the sorted array arr Step 7:Stop

**Source code:** Write a program to sort a list of numbers using bubble sort

#include<iostream> using namespace std;

void bubble\_sort(int list[30],int n); int main()

{

int n,i;

int list[30];

cout<<"enter no of elements\n"; cin>>n;

cout<<"enter "<<n<<" numbers "; for(i=0;i<n;i++)

cin>>list[i]; bubble\_sort (list,n);

cout<<" after sorting\n"; for(i=0;i<n;i++) cout<<list[i]<<endl;

return 0;

}

void bubble\_sort (int list[30],int n)

{

int temp ; int i,j;

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

# Data Structures using C++ LAB FA23

for(j=0;j<n-1;j++) if(list[j]>list[j+1])

{

temp=list[j]; list[j]=list[j+1]; list[j+1]=temp;

}

}

Results

1. **Selection sort ( Select the smallest and Exchange ):**

The first item is compared with the remaining n-1 items, and whichever of all is lowest, is put in the first position. Then the second item from the list is taken and compared with the remaining (n-2) items, if an item with a value less than that of the second item is found on the (n-2) items, it is swapped (Interchanged) with the second item of the list and so on.

#### Algorithm:

Selection\_Sort ( A [ ] , N ) Step 1 :start

Step 2: Repeat For K = 0 to N – 2 Begin

Step 3 : Set POS = K

Step 4 : Repeat for J = K + 1 to N – 1 Begin

If A[ J ] < A [ POS ]

Set POS = J

End For

Step 5 : Swap A [ K ] with A [ POS ] End For

Step 6 : stop

**Source code:** Program to implement selection sort

#include<iostream> using namespace std;

void selection\_sort (int list[],int n); int main()

{

int n,i;

int list[30];

# Data Structures using C++ LAB FA23

cout<<"enter no of elements\n"; cin>>n;

cout<<"enter "<<n<<" numbers "; for(i=0;i<n;i++)

cin>>list[i]; selection\_sort (list,n); cout<<" after sorting\n"; for(i=0;i<n;i++) cout<<list[i]<<endl;

return 0;

}

void selection\_sort (int list[],int n)

{

int min,temp,i,j;

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

{

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

{

if(list[j]<list[min]) min=j;

}

temp=list[i]; list[i]=list[min]; list[min]=temp;

}

}

Results

1. **Quick sort:** It is a divide and conquer algorithm. Quick sort first divides a large array into two smaller sub-arrays: the low elements and the high elements. Quick sort can then recursively sort the sub-arrays.

ALGORITHM:

Step 1: Pick an element, called a pivot, from the array.

Step 2: Partitioning: reorder the array so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it (equal values can go either way). After this partitioning, the pivot is in its final position. This is called the partition operation.

# Data Structures using C++ LAB FA23

Step 3: Recursively apply the above steps to the sub-array of elements with smaller values and separately to the sub-array of elements with greater values.

**Source code:** program to implement Quick sort

#include<iostream> using namespace std;

void quicksort(int x[],int Lb,int Ub)

{

int down,up,pivot,t;

if(Lb<Ub)

{

down=Lb; up=Ub; pivot=down; while(down<up)

{

while((x[down]<=x[pivot])&&(down<Ub))down++; while(x[up]>x[pivot])up--;

if(down<up)

{

t=x[down]; x[down]=x[up]; x[up]=t;

}/\*endif\*/

}

t=x[pivot]; x[pivot]=x[up]; x[up]=t;

quicksort( x,Lb,up-1); quicksort( x,up+1,Ub);

}

}

int main()

{

int n,i;

int list[30];

cout<<"enter no of elements\n"; cin>>n;

cout<<"enter "<<n<<" numbers "; for(i=0;i<n;i++)

cin>>list[i]; quicksort(list,0,n-1); cout<<" after sorting\n"; for(i=0;i<n;i++) cout<<list[i]<<endl;

return 0;

}

# Data Structures using C++ LAB FA23

Results

1. **Insertion sort:** It iterates, consuming one input element each repetition, and growing a sorted output list. Each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain

#### ALGORITHM:

Step 1: start

Step 2: for i ← 1 to length(A) Step 3: j ← i

Step 4: while j > 0 and A[j-1] > A[j] Step 5: swap A[j] and A[j-1] Step 6: j ← j - 1

Step 7: end while Step 8: end for Step9: stop

**Source code: program to implement insertion sort**

#include<iostream> using namespace std;

void insertion\_sort(int a[],int n)

{

int i,t,pos;

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

{

t=a[i]; pos=i;

while(pos>0&&a[pos-1]>t)

{

a[pos]=a[pos-1]; pos--;

}

a[pos]=t;

}

}

int main()

{

# Data Structures using C++ LAB FA23

int n,i;

int list[30];

cout<<"enter no of elements\n"; cin>>n;

cout<<"enter "<<n<<" numbers "; for(i=0;i<n;i++)

cin>>list[i]; insertion\_sort(list,n); cout<<" after sorting\n"; for(i=0;i<n;i++) cout<<list[i]<<endl;

return 0;

}

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | date | sign | Remarks |
| 1.What are the various time complexities of different sorting algorithms |  |  |  |
| 2.Write a program to implement all above sorting techniques in a single program. provide a menu for selection of various sorting techniques  during runtime. |  |  |  |
| 3.Write a program to implement above sorting technique using dynamic  memory allocation |  |  |  |
| 4.write a program to count no of operations performed in each step in all sorting algorithms |  |  |  |

# Data Structures using C++ LAB FA23

Week 3: Write C++ programs to implement the following using an array. Stack ADT b) Queue ADT

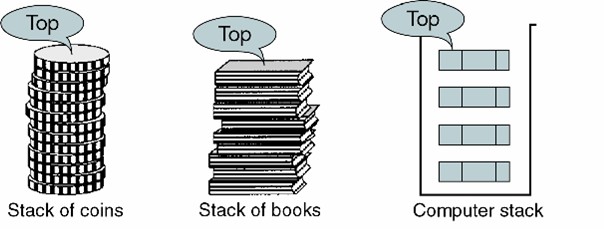
Aim: To implement Stack ADT and Queue ADT using an array

Description:

**Stack**:It is an ordered collection of data elements into which new elements may be inserted and from which elements may be deleted at one end called the “TOP” of stack.

* + A stack is a last-in-first-out ( LIFO ) structure.
  + Insertion operation is referred as “PUSH” and deletion operation is referred as “POP”.
  + The most accessible element in the stack is the element at the position “TOP”.
  + Stack must be created as empty.
  + Whenever an element is pushed into stack, it must be checked whether the stack is full or not.
  + Whenever an element is popped form stack, it must be checked whether the stack is empty or not.

We can implement the stack ADT either with array or linked list.



ALGORITHM: push()

Step 1: if top> =max-1 then

Step 2: Display the stack overflows Step 3: else then

Step 4: top ++

Step 5: assign stack[top]=x

Step 6: Display element is inserted ALGORITHM pop()

Step 1: if top = =-1 then

Step 2: Display the stack is underflows Step 3: else

Step 4: assign x=stack[top] Step 5: top- -

Step 6: return x

**Source code**: To implement Stack ADT using an array

#include<iostream> using namespace std; #include<stdlib.h> #define max 50 template <class T>

# Data Structures using C++ LAB FA23

class stack

{

private:

T top,stk[50],item;

public:

};

stack(); void push(); void pop(); void display();

template <class T> stack<T>::stack()

{

top=-1;

}

//code to push an item into stack; template <class T>

void stack<T>::push()

{

if(top==max-1)

cout<<"Stack Overflow...\n";

else

{

cout<<"Enter an item to be pushed:"; top++;

cin>>item; stk[top]=item;

cout<<"Pushed Sucesfully \n";

}

}

template <class T> void stack<T>::pop()

{

if(top==-1)

cout<<"Stack is Underflow";

else

}

{

item=stk[top]; top--;

cout<<item<<" is poped Sucesfully \n";

}

template <class T>

void stack<T>::display()

{

if(top==-1)

cout<<"Stack Under Flow";

else

{

for(int i=top;i>-1;i--)

# Data Structures using C++ LAB FA23

{

cout<<"|"<<stk[i]<<"|\n"; cout<<" \n";

}

}

}

int main()

{

int choice; stack<int>st;

while(1)

{

cout<<"\n\n\*\*\*\*\*Menu for Skack operations\*\*\*\*\*\n\n"; cout<<"1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n";

cout<<"Enter Choice:"; cin>>choice; switch(choice)

{

case 1:

case 2:

st.push(); break;

st.pop(); break;

case 3: cout<<"Elements in the Stack are \n";

st.display(); break;

case 4:

exit(0);

default:cout<<"Invalid choice...Try again...\n";

}

}

}

Results

# Data Structures using C++ LAB FA23

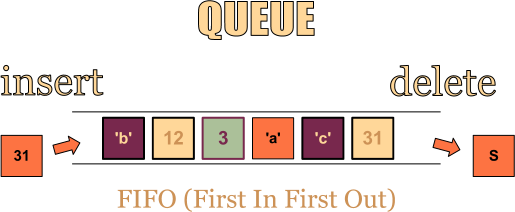
## QUEUE

DESCRIPTION:

Queue is a data structure in which the elements are added at one end, called the rear, and deleted from the other end, called the front. A First In First Out data structure (FIFO).The rear of the queue is accessed whenever a new element is added to the queue, and the front of the queue is accessed whenever an element is deleted from the queue. As in a stack, the middle elements in the queue are in accessible, even if the queue elements are sorted in an array.

BASIC QUEUE OPERATIONS:

1. initializeQueue(): Initializes the queue to an empty state.
2. Determines whether the queue is empty. If the queue is empty, it returns the value true; otherwise, it returns the value false.
3. Determines whether the queue is full. If the queue is empty, it returns the value true; otherwise, it returns the value false.
4. rear: Returns the last element of the queue. Prior to this operation, the queue must exit.
5. front: Returns the front, that is, the first element of the queue. Priority to this operation, the queue must exit.

Queue can be stored either in an array or in linked list. We will consider both implementations. Because elements are added at one end and remove from the other end, we need two pointers to keep track of the front and rear of the queue, called queueFront and queueRear. Queues are restricted versions of arrays and linked lists. The middle terms of queue should not be accessed directly.

**Source code: To implement Queue ADT using an array**

#include<stdlib.h> #include<iostream> using namespace std; #define max 5 template <class T> class queue

{

private:T q[max],item;

int front,rear; public: queue();

void insert\_q(); void delete\_q(); void display\_q();

};

# Data Structures using C++ LAB FA23

template <class T> queue<T>::queue()

{

front=rear=-1;

}

//code to insert an item into queue; template <class T>

void queue<T> ::insert\_q()

{

if(rear>=max-1)

cout<<"queue Overflow...\n";

else

{

if(front>rear) front=rear=-1; else

{ if(front==-1) front=0;

rear++;

cout<<"Enter an item to be inserted:"; cin>>item;

q[rear]=item;

cout<<"inserted Sucesfully..into queue..\n";

}

}

}

template <class T>

void queue<T>::delete\_q()

{

if(front==-1||front>rear)

{

}

else

{

}

}

front=rear=-1;

cout<<"queue is Empty \n";

item=q[front]; front++;

cout<<item<<" is deleted Sucesfully \n";

template <class T>

void queue<T>::display\_q()

{

if(front==-1||front>rear)

{

front=rear=-1;

cout<<"queue is Empty \n";

}

# Data Structures using C++ LAB FA23

else

{

}

}

for(int i=front;i<=rear;i++) cout<<"|"<<q[i]<<"|<--";

int main()

{

int choice; queue<int> q;

while(1)

{

cout<<"\n\n\*\*\*\*\*Menu for QUEUE operations\*\*\*\*\*\n\n"; cout<<"1.INSERT\n2.DELETE\n3.DISPLAY\n4.EXIT\n";

cout<<"Enter Choice:"; cin>>choice; switch(choice)

{

case 1: q.insert\_q(); break;

case 2: q.delete\_q(); break;

case 3: cout<<"Elements in the queue are \n";

q.display\_q(); break;

case 4: exit(0);

default: cout<<"Invalid choice...Try again. \n";

}

}

return 0;

}

Results

Assignment :-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | remark |
| 1.Write a program to perform matching of parenthesis using stack. |  |  |  |
| 2.Write a program to perform evaluation of postfix expression |  |  |  |
| 3.Write a program to convert given infix expression to post fix |  |  |  |

# Data Structures using C++ LAB FA23

Week 4: C++ programs to implement list ADT to perform following operations

* 1. Insert an element into a list. b) Delete an element from list

c) Search for a key element in list d)count number of nodes in list

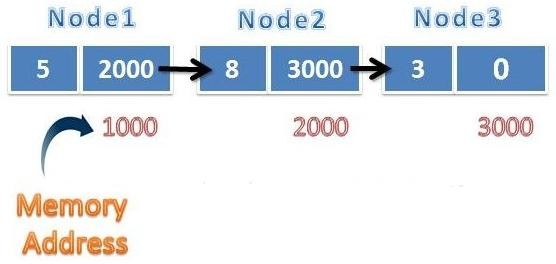
Aim: To implement list ADT to perform following operations

a) Insert an element into a list. b) Delete an element from list

c) Search for a key element in list d)count number of nodes in list

Description:

#### List ADT

A **linked list** is a data structure consisting of a group of nodes which together represent a sequence. Each node is composed of a data part and a reference (in other words, a *link*) to the next node in the sequence.

The Linked List is a collection of elements called nodes, each node of which stores two items of information, i.e., data part and link field.

The data part of each node consists the data record of an entity. The link field is a pointer and contains the address of next node.

The beginning of the linked list is stored in a pointer termed as head which points to the first node.

The head pointer will be passed as a parameter to any method, to perform an operation.

First node contains a pointer to second node, second node contains a pointer to the third node and so on.

The last node in the list has its next field set to NULL to mark the end of the list.

* There are several variants of linked lists. These are as follows:
  + Singly linked list
  + Circular linked list
  + Doubly linked list
  + Doubly circular linked list

**SINGLE LINKED LIST:**

Single Linked List is a collection of nodes. Each node contains 2 fields: I) info where the information is stored and ii) link which points to the next node in the list.

The node is like this:

|  |  |
| --- | --- |
| Node | |
| Info (or) data | Link (or) next |

# Data Structures using C++ LAB FA23

The operations that can be performed on single linked lists includes: insertion, deletion and traversing the list.

Various operations on a single linked list are 1.Insertion of a node into list

2.Deletion of a node from list 3.Traversal of the list

**Source code:**To Implement LIST ADT in C++ #include<stdlib.h>

#include<iostream.h>

#include<conio.h> class node

{

public:

int data; node \*next;

};

class List

{

int item; node \*head;

public: List( );

void insert\_front( ); void insert\_end( ); void delete\_front( ); void delete\_end( ); void display( );

int node\_count();

void delete\_before\_pos(); void delete\_after\_pos();

};

List::List( )

{

head=NULL;

}

//code to insert an item at front List; void List::insert\_front( )

{

node \*p;

cout<<"Enter an element to be inserted:"; cin>>item;

p=new node; p->data=item;

p->next=NULL; if(head==NULL)

{

head=p;

}

else

# Data Structures using C++ LAB FA23

{ p->next=head; head=p;

}

cout<<"\nInserted at front of Linked List Sucesfully \n";

}

//code to insert an item at end List void List::insert\_end( )

{

node \*p;

cout<<"Enter an element to be inserted:"; cin>>item;

p=new node; p->data=item;

p->next=NULL; if(head==NULL)

{

}

else

head=p;

{

node\*t; t=head;

while(t->next!=NULL) t=t->next;

1. >next=p;

}

cout<<"\nInserted an element at end of Linked List Sucesfully \n";

}

void List::delete\_front( )

{

node\*t; if(head==NULL)

cout<<"\nList is Underflow";

else

{ item=head->data; t=head; head=head->next;

cout<<"\n"<<item<<" is deleted Sucesfully from List \n";

delete(t);

}

}

void List::delete\_end( )

{

node\*t,\*prev;

if(head==NULL)

cout<<"\nList is Underflow";

# Data Structures using C++ LAB FA23

else

{

t=head;

if(head->next==NULL)

{

}

else

{

cout<<"\n"<<t->data<<" is deleted Sucesfully from List \n";

delete(t); head=NULL;

while(t->next!=NULL)

{

prev=t; t=t->next;

}

prev->next=NULL;

cout<<"\n"<<t->data<<" is deleted Sucesfully from List \n";

delete(t);

}

}

}

//Delete a node before a position void List::delete\_before\_pos( )

{

int i=1; int pos;

node\*t,\*prev;

if(head==NULL)

cout<<"\nList is Underflow";

else

{ cout<<"Enter position at which node has to be deleted:"; cin>>pos;

t=head;

int nc=node\_count(); if(pos>nc||pos<=0)

cout<<"invalid position ...try again\n";

else

{

cout<<"Before Deletion elements in the List are..\n"; display();

while(i<pos)

{

prev=t; t=t->next; i++;

}

if(i==1)

{

# Data Structures using C++ LAB FA23

cout<<"\n"<<t->data<<" is deleted Sucesfully from List \n";

if(head->next==NULL)

head=NULL;

else

{

t=head; head=head->next;

}

else

{

cout<<"\n"<<t->data<<"is deleted Sucesfully from List \n";

delete(t);

}

prev->next=t->next;

cout<<"\n"<<t->data<<" is deleted Sucesfully from List \n";

delete(t);

}

cout<<"After Deletion elements in the List are..\n"; display();

}

}

}

//Delete a node after a position void List::delete\_after\_pos( )

{

int i=1; int pos;

node\*t,\*prev;

if(head==NULL)

cout<<"\nList is Underflow";

else

{ cout<<"Enter position at which node has to be deleted:"; cin>>pos;

t=head;

int nc=node\_count(); if(pos>nc||pos<=0)

cout<<"invalid position ...try again\n";

else

{

cout<<"Before Deletion elements in the List are..\n"; display();

while(i<pos)

{

prev=t; t=t->next; i++;

}

if(i==1)

{

cout<<"\n"<<t->data<<" is deleted Sucesfully from List \n";

# Data Structures using C++ LAB FA23

if(head->next==NULL)

head=NULL;

else

{

t=head; head=head->next;

cout<<"\n"<<t->data<<" is deleted Sucesfully from List \n";

delete(t);

}

else

{

}

}

prev->next=t->next;

cout<<"\n"<<t->data<<" is deleted Sucesfully from List \n";

delete(t);

cout<<"After Deletion elements in the List are..\n"; display();

}

}

}

void List::display()

{

node\*t;

if(head==NULL)

cout<<"\nList Under Flow";

else

{

cout<<"\nElements in the List are \n";

t=head;

while(t!=NULL)

{

cout<<"|"<<t->data<<"|->"; t=t->next;

}

}

}

//code to count no of nodes int List::node\_count( )

{

int nc=0; node\*t;

if(head==NULL)

{

cout<<"\nList Under Flow"<<endl;

// cout<<"No Nodes in the Linked List are: "<<nc<<endl;

}

else

{

# Data Structures using C++ LAB FA23

t=head; while(t!=NULL)

{

nc++;

t=t->next;

}

// cout<<"No Nodes in the Linked List are: "<<nc<<endl;

}

return nc;

}

int main( )

{

int choice; List LL;

while(1)

{

cout<<"\n\n\*\*\*Menu for Linked List operations\*\*\*\n\n";

cout<<"1.Insert Front\n2.Insert end\n3.Delete front\n4.Delete End\n5.DISPLAY\n"; cout<<"6.Node Count\n7.Del before a position\n8.Del after position\n"; cout<<"9.Clear Scrn\n10.Exit\nEnter Choice:";

cin>>choice; switch(choice)

{

case 1: LL.insert\_front( ); break;

case 2: LL.insert\_end( );

break;

case 3: LL.delete\_front( ); break;

case 4: LL.delete\_end( );

break;

case 5: LL.display( ); break;

case 6:cout<<"No of nodes in List:"<<LL.node\_count(); break;

case 7:LL.delete\_before\_pos(); break;

case 8:LL.delete\_after\_pos(); break;

case 9:clrscr(); break;

case 10:exit(0);

default:cout<<"Invalid choice...Try again...\n";

}

}

}

# Data Structures using C++ LAB FA23

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1.Write a program to concatenate two linked lists |  |  |  |
| 2.Write a program to reverse a given linked list |  |  |  |
| 3.Write a program to generate two lists from a given linked list  such that first list contains all elements in odd places and second list consists of all elements in even places |  |  |  |

# Data Structures using C++ LAB FA23

Week 5: Write C++ programs to implement the following using a singly linked list.

a) Stack ADT b) Queue ADT

Aim: To implement Stack ADT and Queue ADT using a singly linked list.

Description:

A Stack is a collection of items in which new items may be deleted at end to implement stack using linked list we need to define a node which in turn consist of data a pointer to the next node. The advantage of representing stack using linked lists is that we can decide which end should be top of a stack. And since the array size is fixed, in the array (linear) representation of stack, only fixed number of elements can be pushed onto the stack. If in a program the number of elements to be pushed exceeds the size of the array, the program may terminate in an error. We must overcome these problems.

By using linked lists we can dynamically organize data (such as an ordered list).Therefore , ;ogically the stack is never full. The stack is full only if we run out of memory space. In the below program we select front end as top if stack in which we cab add or remove data.

#### Source code: To implement Stack ADT using a singly linked list.

#include<stdlib.h> #include<iostream> using namespace std; template <class T> class node

{

public:

T data; node<T>\*next;

};

template <class T> class stack

{

private:

T item; node<T> \*top;

public: stack();

void push(); void pop(); void display();

};

template <class T> stack<T>::stack()

{

top=NULL;

}

//code to push an item into stack; template <class T>

# Data Structures using C++ LAB FA23

void stack<T>::push()

{

node<T>\*t; node<T>\*p;

cout<<"Enter an item to be pushed:"; cin>>item;

p=new node<T>; p->data=item;

p->next=top; top=p;

cout<<"\nPushed Sucesfully \n";

}

template <class T> void stack<T>::pop()

{

node<T>\*t; if(top==NULL)

cout<<"\nStack is Underflow";

else

{

item=top->data; top=top->next;

cout<<"\n"<<item<<" is poped Sucesfully \n";

}

}

template <class T>

void stack<T>::display()

{

node<T>\*t;

if(top==NULL)

cout<<"\nStack Under Flow";

else

{ cout<<"\nElements in the Stack are \n";

t=top; while(t!=NULL)

{

cout<<"|"<<t->data<<"|\n"; cout<<" \n";

t=t->next;

}

}

}

int main()

{

int choice; stack<int>st;

while(1)

{

# Data Structures using C++ LAB FA23

cout<<"\n\n\*\*\*Menu for Skack operations\*\*\*\n\n"; cout<<"1.PUSH\n2.POP\n3.DISPLAY\n4.EXIT\n";

cout<<"Enter Choice:"; cin>>choice; switch(choice)

{

case 1:

case 2:

st.push(); break;

st.pop(); break;

case 3: st.display(); break;

case 4:

exit(0);

default:cout<<"Invalid choice...Try again...\n";

}

}

}

Results

**Description**: Queue is a data structure in which the elements are added at one end, called the **rear**, and deleted from the other end, called the **front**. A First In First Out data structure (FIFO). The rear of the queue is accessed whenever a new element is added to the queue, and the front of the queue is accessed whenever an element is deleted from the queue. As in a stack, the middle elements in the queue are in accessible, even if the queue elements are sorted in an array.

#### Source code: To implement QUEUE ADT using a singly linked list

#include<stdlib.h> #include<iostream.h> template <class T> class node

{

public:

T data; node<T>\*next;

};

# Data Structures using C++ LAB FA23

template <class T> class queue

{

private:

T item;

friend class node<T>; node<T> \*front,\*rear;

public: queue();

void insert\_q(); void delete\_q(); void display\_q();

};

template <class T> queue<T>::queue()

{

front=rear=NULL;

}

//code to push an item into queue; template <class T>

void queue<T>::insert\_q()

{

node<T>\*p;

cout<<"Enter an element to be inserted:"; cin>>item;

p=new node<T>; p->data=item;

* 1. >next=NULL; if(front==NULL)

{

rear=front=p;

}

else

{

rear->next=p;

rear=p;

}

cout<<"\nInserted into Queue Sucesfully \n";

}

//code to delete an element template <class T>

void queue<T>::delete\_q()

{

node<T>\*t; if(front==NULL)

cout<<"\nqueue is Underflow";

else

{

item=front->data; t=front; front=front->next;

# Data Structures using C++ LAB FA23

cout<<"\n"<<item<<" is deleted Sucesfully from queue \n";

}

delete(t);

}

//code to display elements in queue template <class T>

void queue<T>::display\_q()

{

node<T>\*t;

if(front==NULL)

cout<<"\nqueue Under Flow";

else

{

cout<<"\nElements in the queue are \n";

t=front;

while(t!=NULL)

{

}

}

int main()

{

cout<<"|"<<t->data<<"|<-"; t=t->next;

}

int choice; queue<int>q1;

while(1)

{

cout<<"\n\n\*\*\*Menu for Queue operations\*\*\*\n\n"; cout<<"1.Insert\n2.Delete\n3.DISPLAY\n4.EXIT\n"; cout<<"Enter Choice:";

cin>>choice; switch(choice)

{

}

}

return 0;

}

case 1: q1.insert\_q(); break;

case 2: q1.delete\_q(); break;

case 3: q1.display\_q(); break;

case 4: exit(0);

default:cout<<"Invalid choice...Try again...\n";

# Data Structures using C++ LAB FA23

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1.Submit an analysis report on stack implemented by static  allocation and dynamic allocation |  |  |  |
| 2.Submit an analysis report on Queue implemented by static  allocation and dynamic allocation |  |  |  |
| 3.Submit a report on application of stack and queue with  explanation |  |  |  |

# Data Structures using C++ LAB FA23

Week 6: Write C++ programs to implement the de queue (double ended queue) ADT using a doubly linked list and an array.

**Aim:** To implement the de queue (double ended queue) ADT using a doubly linked list and an array.

**Source code:** To implement the de queue (double ended queue) ADT

#include <iostream.h> #include<stdlib.h> #include <conio.h> template<class T> class node

{

public:

T data; node\*prev; node\*next;

};

template<class T> class dll

{

node<T>\*head; public:

dll();

void insert\_front(); void insert\_end(); void delete\_front(); void delete\_end(); void display();

void insert\_at\_pos(); int node\_count();

};

template<class T> dll<T>::dll()

{

head=NULL;

}

//code to insert node at front of list... template<class T>

void dll<T>::insert\_front()

{

node<T>\*new\_node;

int x;

new\_node=new node<T>; cout<<"Enter data into node:\n"; cin>>x;

new\_node->data=x;

new\_node->prev=NULL; new\_node->next=NULL;

# Data Structures using C++ LAB FA23

if(head==NULL) head=new\_node;

else

{

new\_node->next=head; head->prev=new\_node; head=new\_node;

}

cout<<"Inserted node sucesfully...";

}

//code to insert node at end of list... template<class T>

void dll<T>::insert\_end()

{

node<T>\*new\_node; node<T>\*t;

int x;

new\_node=new node<T>; cout<<"Enter data into node:\n"; cin>>x;

new\_node->data=x; new\_node->next=NULL; new\_node->prev=NULL; if(head==NULL)

head=new\_node; else

{

t=head;

while(t->next!=NULL) t=t->next;

t->next= new\_node; new\_node->prev=t;

}

cout<<"Inserted node sucesfully...";

}

template<class T>

void dll<T>::delete\_front()

{

node<T>\*temp;

if(head==NULL)

cout<<"List is empty \n ";

else if(head->next==NULL)

{

temp=head;

cout<<"Deleted element from Doubly Linked List is "<<temp->data<<endl; delete temp;

head=NULL;

}

else

# Data Structures using C++ LAB FA23

{

temp=head; head=head->next; head->prev=NULL;

cout<<"Deleted element from Doubly Linked List is "<<temp->data<<endl; delete temp;

cout<<"Elements after deletion from Front are...\n"; display();

}

}

template<class T>

void dll<T>::delete\_end()

{

node<T>\*t1; node<T>\*t2;

if(head==NULL)

cout<<"List is empty \n ";

else

{

t1=t2=head;

if(head->next==NULL)

{

head=NULL;

cout<<"Deleted element from Doubly Linked List is "<<t1->data<<endl; delete t1;

}

else

{

while(t1->next!=NULL)

{

t2=t1;

t1=t1->next;

}

t2->next=NULL;

cout<<"Deleted element from Doubly Linked List is "<<t1->data<<endl; delete t1;

cout<<"Elements after Deletion from End are...\n"; display();

}

}

}

template<class T>

void dll<T>::insert\_at\_pos()

{

node<T>\*new\_node; node<T>\*t1; node<T>\*t2;

int x,pos,nc;

new\_node=new node<T>; cout<<"Enter data into node:\n";

# Data Structures using C++ LAB FA23

cin>>x;

cout<<"enter Pos at which node has to be inserted:"; cin>>pos;

new\_node->data=x; new\_node->next=NULL; new\_node->prev=NULL; nc=node\_count();

cout<<"node count="<<nc<<endl; if(pos<=0||pos>nc+1)

cout<<"invalid position";

else

{

if(pos==1)

{

if(head==NULL)

head=new\_node;

else

{

new\_node->next=head; head->prev=new\_node; head=new\_node;

}

}

else

{

t1=t2=head; int i=1; while(i<pos)

{

}

t2=t1;

t1=t1->next; i++;

if(t1==NULL)

{

new\_node->next=NULL;

}

else

{

t1->prev=new\_node;

}

t2->next= new\_node; new\_node->prev=t2;

}

cout<<"Inserted node sucesfully...";

}

# Data Structures using C++ LAB FA23

}

template<class T>

int dll<T>:: node\_count()

{

int i=0; node<T> \*t; t=head; while(t!=NULL)

{

t=t->next; i++;

}

return i;

}

template<class T> void dll<T>::display()

{

node<T>\*t; int count;

t=head; if(head==NULL)

{

}

else

{

cout<<"Doubly linked list is empty \n";

cout<<"Elements in the list are \n";

while(t!=NULL)

{

cout<<"|"<<t->data<<"|-> "; t=t->next;

}

}

count=node\_count();

cout<<"\n Total No of nodes in Doubly linked List are:"<<count<<endl;

}

int main()

{

dll<int> d; int choice;

while(1)

{ cout<<"\n\*\*\*Menu for Doubly linked list operations\*\*\*\n"; cout<<"\n1.insert front";

cout<<"\n2.insert end"; cout<<"\n3.delete front"; cout<<"\n4.delete end"; cout<<"\n5.Display"; cout<<"\n6.insert at pos";

# Data Structures using C++ LAB FA23

cout<<"\n7.Exit"; cout<<"\nEnter Choice:"; cin>>choice; switch(choice)

{

case 1:d.insert\_front();

break; case 2:d.insert\_end();

break; case 3:d.delete\_front();

break; case 4:d.delete\_end();

break; case 5:d.display();

break; case 6:d.insert\_at\_pos();

break;

case 7:exit(0);

}

}

//return 0;

}

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1.Implement queue using doubly linked list |  |  |  |
| 2.Implement circular Queue using doubly linked list |  |  |  |
| 3. |  |  |  |

# Data Structures using C++ LAB FA23

Week 7: Write a C++ program to perform the following operations:

* + 1. Insert an element into a binary search tree.
    2. Delete an element from a binary search tree.
    3. Search for a key element in a binary search tree.

#### Description:

Binary Search Tree:

So to make the searching algorithm faster in a binary tree we will go for building the binary search tree. The binary search tree is based on the binary search algorithm. While creating the binary search tree the data is systematically arranged.

That means values at

#### left sub-tree < root node value < right sub-tree values.

10

7

15

5

9

12

18

**Source code:**

#include<stdlib.h> #include<iostream.h> class node

{

public:

int data; node\*lchild; node\*rchild;

};

class bst:public node

{ int item; node \*root;

public: bst();

void insert\_node(); void delete\_node(); void display\_bst(); void inorder(node\*);

# Data Structures using C++ LAB FA23

};

bst::bst()

{

root=NULL;

}

void bst:: insert\_node()

{

node \*new\_node,\*curr,\*prev; new\_node=new node; cout<<"Enter data into new node"; cin>>item;

new\_node->data=item; new\_node->lchild=NULL; new\_node->rchild=NULL; if(root==NULL)

root=new\_node;

else

{ curr=prev=root; while(curr!=NULL)

{ if(new\_node->data>curr->data)

{ prev=curr; curr=curr->rchild;

}

else

{ prev=curr; curr=curr->lchild;

}

}

cout<<"Prev:"<<prev->data<<endl; if(prev->data>new\_node->data)

prev->lchild=new\_node;

else

}

}

prev->rchild=new\_node;

//code to delete a node void bst::delete\_node()

{

if(root==NULL)

cout<<"Tree is Empty";

else

{

int key;

cout<<"Enter the key value to be deleted"; cin>>key;

node\* temp,\*parent,\*succ\_parent; temp=root;

while(temp!=NULL)

{ if(temp->data==key)

{ //deleting node with two childern

# Data Structures using C++ LAB FA23

if(temp->lchild!=NULL&&temp->rchild!=NULL)

{ //search for inorder sucessor node\*temp\_succ; temp\_succ=temp->rchild; while(temp\_succ->lchild!=NULL)

{

succ\_parent=temp\_succ; temp\_succ=temp\_succ->lchild;

}

temp->data=temp\_succ->data; succ\_parent->lchild=NULL;

cout<<"Deleted sucess fully"; return;

}

//deleting a node having one left child

if(temp->lchild!=NULL&temp->rchild==NULL)

{

if(parent->lchild==temp)

parent->lchild=temp->lchild;

else

parent->rchild=temp->lchild;

temp=NULL; delete(temp);

cout<<"Deleted sucess fully"; return;

}

//deleting a node having one right child

if(temp->lchild==NULL&temp->rchild!=NULL)

{

if(parent->lchild==temp)

parent->lchild=temp->rchild;

else

parent->rchild=temp->rchild;

temp=NULL; delete(temp);

cout<<"Deleted sucess fully"; return;

}

//deleting a node having no child

if(temp->lchild==NULL&temp->rchild==NULL)

{

if(parent->lchild==temp) parent->lchild=NULL;

else

parent->rchild=NULL;

temp=NULL; delete(temp);

cout<<"Deleted sucess fully"; return;

}

# Data Structures using C++ LAB FA23

}

else if(temp->data<key)

{ parent=temp; temp=temp->rchild;

}

else if(temp->data>key)

{ parent=temp; temp=temp->lchild;

}

}//end while

}//end if

}//end delnode func void bst::display\_bst()

{

if(root==NULL)

cout<<"\nBST Under Flow";

else

}

inorder(root);

void bst::inorder(node\*t)

{

if(t!=NULL)

{

inorder(t->lchild); cout<<" "<<t->data; inorder(t->rchild);

}

}

int main()

{

bst bt; int i;

while(1)

{

cout<<"\*\*\*\*BST Operations\*\*\*\*"; cout<<"\n1.Insert\n2.Display\n3.del\n4.exit\n"; cout<<"Enter Choice:";

cin>>i; switch(i)

{

case 1:bt.insert\_node();

break; case 2:bt.display\_bst();

break; case 3:bt.delete\_node();

break; case 4:exit(0);

default: cout<<"Enter correct choice";

}

# Data Structures using C++ LAB FA23

}

}

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1.What is a binary tree& binary search tree |  |  |  |
| 2.Applications of binary search tree |  |  |  |

# Data Structures using C++ LAB FA23

Week 8 : Write C++ programs for implementing the following sorting methods:

a) Merge sort b) Heap sort

**Aim:** To implement Merge sort and Heap sort

#### Description:

Merge sort is an O(*n* log *n*) comparison-based sorting algorithm. It is stable, meaning that it preserves the input order of equal elements in the sorted output. It is an example of the divide and conquer algorithmic paradigm. Merge sort is so inherently sequential that it's practical to run it using slow tape drives as input and output devices. It requires very little memory, and the memory required does not change with the number of data elements. If you have four tape drives, it works as follows:

1. Divide the data to be sorted in half and put half on each of two tapes
2. Merge individual pairs of records from the two tapes; write two-record chunks alternately to each of the two output tapes
3. Merge the two-record chunks from the two output tapes into four-record chunks; write these alternately to the original two input tapes
4. Merge the four-record chunks into eight-record chunks; write these alternately to the original two output tapes
5. Repeat until you have one chunk containing all the data, sorted --- that is, for log *n*

passes, where *n* is the number of records.

Conceptually, merge sort works as follows:

1. Divide the unsorted list into two sublists of about half the size
2. Divide each of the two sublists [recursively](http://en.wikipedia.org/wiki/Recursion) until we have list sizes of length 1, in which case the list itself is returned
3. [Merge](http://en.wikipedia.org/wiki/Merge_algorithm) the two sublists back into one sorted list.

**Source code:**

#include<iostream> using namespace std; #define max 15 template<class T>

void merge(T a[],int l,int m,int u)

{

T b[max];

int i,j,k;

i=l; j=m+1; k=l;

while((i<=m)&&(j<=u))

{

# Data Structures using C++ LAB FA23

if(a[i]<=a[j])

{

}

if(i>m)

{

}

else

{

}

++k;

b[k]=a[i];

++i;

b[k]=a[j];

++j;

while(j<=u)

{

}

}

else

{

b[k]=a[j];

++j;

++k;

while(i<=m)

{

b[k]=a[i];

++i;

++k;

}

}

for(int r=l;r<=u;r++) a[r]=b[r];

}

template <class T>

void mergesort(T a[],int p,int q)

{

int mid; if(p<q)

{

mid=(p+q)/2; mergesort(a,p,mid); mergesort(a,mid+1,q); merge(a,p,mid,q);

}

}

int main()

{

# Data Structures using C++ LAB FA23

int n,i;

int list[30];

cout<<"enter no of elements\n"; cin>>n;

cout<<"enter "<<n<<" numbers "; for(i=0;i<n;i++)

cin>>list[i]; mergesort (list,0,n-1);

cout<<" after sorting\n"; for(i=0;i<n;i++) cout<<list[i]<<endl;

return 0;

}

Results

#### HEAP SORT

Heap sort is a method in which a binary tree is used. In this method first the heap is created using binary tree and then heap is sorted using priority queue.

Source code:// C++ program for implementation of Heap Sort #include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is

// an index in arr[]. n is size of heap void heapify(int arr[], int n, int i)

{

int largest = i; // Initialize largest as root int l = 2\*i + 1; // left = 2\*i + 1

int r = 2\*i + 2; // right = 2\*i + 2

// If left child is larger than root if (l < n && arr[l] > arr[largest])

largest = l;

// If right child is larger than largest so far

# Data Structures using C++ LAB FA23

if (r < n && arr[r] > arr[largest]) largest = r;

// If largest is not root if (largest != i)

{

swap(arr[i], arr[largest]);

// Recursively heapify the affected sub-tree heapify(arr, n, largest);

}

}

// main function to do heap sort void heapSort(int arr[], int n)

{

// Build heap (rearrange array) for (int i = n / 2 - 1; i >= 0; i--)

heapify(arr, n, i);

// One by one extract an element from heap for (int i=n-1; i>=0; i--)

{

// Move current root to end swap(arr[0], arr[i]);

// call max heapify on the reduced heap heapify(arr, i, 0);

}

}

/\* A utility function to print array of size n \*/ void printArray(int arr[], int n)

{

for (int i=0; i<n; ++i) cout << arr[i] << " ";

cout << "\n";

}

int main()

{

int n,i;

int list[30];

cout<<"enter no of elements\n"; cin>>n;

cout<<"enter "<<n<<" numbers "; for(i=0;i<n;i++)

cin>>list[i]; heapSort(list, n);

cout << "Sorted array is \n";

# Data Structures using C++ LAB FA23

printArray(list, n); return 0;

}

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1.Write a program to explain time complexity in Merge sort |  |  |  |
| 2.Explain about heap sort |  |  |  |

# Data Structures using C++ LAB FA23

Week 9 : Write C++ programs that use recursive functions to traverse the given binary tree in Preorder b) inorder and c) postorder

Aim: To implement Binary tree traversals

#### Description:

It is often convenient to a single list containing all the nodes in a tree. This list may correspond to an order in which the nodes should be visited when the tree is being searched. We define three such lists here, the **preorder**, **postorder** and **inorder** traversals of the tree. The definitions themselves are recursive:

* if *T* is the empty tree, then the empty list is the preorder, the inorder and the postorder traversal associated with *T*;
* if *T* = [*N*] consists of a single node, the list [*N*] is the preorder, the inorder and the postorder traversal associated with *T*;
* otherwise, *T* contains a root node *n*, and subtrees *T*1,..., *T*n: and
  + the *preorder* traversal of the nodes of *T* is the list containing *N*, followed, in order by the preorder traversals of *T*1..., *T*n;
  + the *inorder* traversal of the nodes of *T* is the list containing the inorder traversal of *T*1 followed by *N* followed in order by the inorder traversal of each of *T*2,..., *T*n.
  + the *postorder* traversal of the nodes of *T* is the list containing in order the postorder traversal of each of *T*1,..., *T*n, followed by *N*.

Source code:

#include<stdlib.h> #include<iostream.h> class node

{

public:

int data; node\*Lchild; node\*Rchild;

};

class bst

{

int item; node \*root;

public: bst();

void insert\_node(); void delete\_node(); void display\_bst(); void preeorder(node\*); void inorder(node\*); void postorder(node\*);

};

# Data Structures using C++ LAB FA23

bst::bst()

{

root=NULL;

}

void bst:: insert\_node()

{

node \*new\_node,\*curr,\*prev; new\_node=new node; cout<<"Enter data into new node"; cin>>item;

new\_node->data=item; new\_node->Lchild=NULL; new\_node->Rchild=NULL; if(root==NULL)

root=new\_node;

else

{

curr=prev=root; while(curr!=NULL)

{

if(new\_node->data>curr->data)

{

}

else

{

}

}

prev=curr; curr=curr->Rchild;

prev=curr; curr=curr->Lchild;

cout<<"Prev:"<<prev->data<<endl; if(prev->data>new\_node->data)

prev->Lchild=new\_node;

else

}

}

prev->Rchild=new\_node;

//code to delete a node void bst::delete\_node()

{

if(root==NULL)

cout<<"Tree is Empty";

else

{

int key;

cout<<"Enter the key value to be deleted"; cin>>key;

node\* temp,\*parent,\*succ\_parent; temp=root;

# Data Structures using C++ LAB FA23

while(temp!=NULL)

{

if(temp->data==key)

{ //deleting node with two childern

if(temp->Lchild!=NULL&&temp->Rchild!=NULL)

{ //search for sucessor node\*temp\_succ; temp\_succ=temp->Rchild;

while(temp\_succ->Lchild!=NULL)

{

succ\_parent=temp\_succ; temp\_succ=temp\_succ->Lchild;

}

temp->data=temp\_succ->data; succ\_parent->Lchild=NULL;

cout<<"Deleted sucess fully"; return;

}

//deleting a node having one left child

if(temp->Lchild!=NULL&temp->Rchild==NULL)

{

if(parent->Lchild==temp)

parent->Lchild=temp->Lchild;

else

parent->Rchild=temp->Lchild;

temp=NULL; delete(temp);

cout<<"Deleted sucess fully"; return;

}

//deleting a node having one right child

if(temp->Lchild==NULL&temp->Rchild!=NULL)

{

if(parent->Lchild==temp)

parent->Lchild=temp->Rchild;

else

parent->Rchild=temp->Rchild;

temp=NULL; delete(temp);

cout<<"Deleted sucess fully"; return;

}

//deleting a node having no child

if(temp->Lchild==NULL&temp->Rchild==NULL)

{

if(parent->Lchild==temp) parent->Lchild=NULL;

else

parent->Rchild=NULL;

temp=NULL;

# Data Structures using C++ LAB FA23

delete(temp);

cout<<"Deleted sucess fully"; return;

}

}

else if(temp->data<key)

{

parent=temp; temp=temp->Rchild;

}

else if(temp->data>key)

{

}

}//end while

}//end if

}//end delnode func

void bst::display\_bst()

{

parent=temp; temp=temp->Lchild;

if(root==NULL)

cout<<"\nBinary Search Tree is Under Flow";

else

{

int ch;

cout<<"\t\t\*\*Binart Tree Traversals\*\*\n"; cout<<"\t\t1.Pree order\n\t\t2.Inorder\n\t\t3:PostOrder\n"; cout<<"\t\tEnter Your Chice:";

cin>>ch; switch(ch)

{

case 1: cout<<"Pree order Tree Traversal\n "; preeorder(root);

break;

case 2: cout<<"Inorder Tree Traversal is\n "; inorder(root);

break;

case 3: cout<<"Inorder Tree Traversal is\n"; postorder(root);

break;

}

}

}

void bst::inorder(node\*t)

{

if(t!=NULL)

{

inorder(t->Lchild); cout<<" "<<t->data;

# Data Structures using C++ LAB FA23

inorder(t->Rchild);

}

}

void bst::preeorder(node\*t)

{

if(t!=NULL)

{

cout<<" "<<t->data; preeorder(t->Lchild); preeorder(t->Rchild);

}

}

void bst::postorder(node\*t)

{

if(t!=NULL)

{

postorder(t->Lchild); postorder(t->Rchild); cout<<" "<<t->data;

}

}

int main()

{

bst bt; int i;

while(1)

{ cout<<"\n\n\*\*\*Operations Binary Search Tree\*\*\*\n"; cout<<"1.Insert\n2.Display\n3.del\n4.exit\n"; cout<<"Enter Choice:";

cin>>i; switch(i)

{

case 1:bt.insert\_node();

break; case 2:bt.display\_bst();

break; case 3:bt.delete\_node();

break; case 4:exit(0);

default:cout<<"Enter correct choice";

}

}

}

# Data Structures using C++ LAB FA23

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1.What is a Tree traversal. |  |  |  |
| 2.Application of Tree traversal |  |  |  |

# Data Structures using C++ LAB FA23

Week 10 : Write a C++ program to perform the following operations

a) Insertion into a B-tree b) Deletion from a B-tree

Aim: To implement B Tree Source code:

#include<iostream.h> #include<stdio.h> #include<string.h> #include<conio.h> #include<stdlib.h> #define MAX 4

#define MIN 2

typedef char Type[10]; typedef struct Btree

{

Type key;

}BT;

typedef struct treenode

{

int count;

BT entry[MAX+1]; treenode \*branch[MAX+1];

}node; class B

{

node \*root; public:

int LT(char \*,char \*); int EQ(char \*,char \*);

node \*Search(Type target,node \*root,int \*targetpos); int SearchNode(Type target,node \*current,int \*pos); node \*Insert(BT New,node \*root);

int MoveDown(BT New,node \*current,BT \*med,node \*\*medright); void InsertIn(BT med,node \*medright,node \*current,int pos);

void Split(BT med,node \*medright,node \*current,int pos,BT \*newmedian,node

\*\*newright);

void Delete(Type target,node \*\*root); void Del\_node(Type target,node \*current); void Remove(node \*current,int pos);

void Successor(node \*current,int pos); void Adjust(node \*current,int pos); void MoveRight(node \*current,int pos); void MoveLeft(node \*current,int pos); void Combine(node \*current,int pos); void InOrder(node \*root);

};

# Data Structures using C++ LAB FA23

int B::LT(char \*a,char \*b)

{

if((strcmp(a,b))<(0)) return 1;

else return 0;

}

int B::EQ(char \*a,char \*b)

{

if((strcmp(a,b))==(0)) return 1;

else return 0;

}

node\* B::Search(Type target,node \*root,int \*targetpos)

{

if(root==NULL) return NULL;

else if(SearchNode(target,root,targetpos)) return root;

else

return Search(target,root->branch[\*targetpos],targetpos);

}

int B::SearchNode(Type target,node \*current,int \*pos)

{

if(LT(target,current->entry[1].key))

{

\*pos=0; return 0;

}

else

{

for(\*pos=current->count;

LT(target,current->entry[\*pos].key) && \*pos>1;(\*pos)--); return EQ(target,current->entry[\*pos].key);

}

}

node \*B::Insert(BT newentry,node \*root)

{

BT medentry; node \*medright; node \*New;

if(MoveDown(newentry,root, &medentry, &medright))

{

New=new node; New->count=1;

New->entry[1]=medentry; New->branch[0]=root; New->branch[1]=medright;

# Data Structures using C++ LAB FA23

return New;

}

return root;

}

int B::MoveDown(BT New,node \*current,BT \*med,node \*\*medright)

{

int pos; if(current==NULL)

{

\*med=New;

\*medright=NULL; return 1;

}

else

{

if(SearchNode(New.key,current,&pos)) cout<<"Duplicate key\n";

if(MoveDown(New,current->branch[pos],med,medright)) if(current->count<MAX)

{

InsertIn(\*med,\*medright,current,pos); return 0;

}

else

{

Split(\*med,\*medright,current,pos,med,medright); return 1;

}

return 0;

}

}

void B::InsertIn(BT med,node \*medright,node \*current,int pos)

{

int i;

for(i=current->count;i>pos;i--)

{

current->entry[i+1]=current->entry[i]; current->branch[i+1]=current->branch[i];

}

current->entry[pos+1]=med; current->branch[pos+1]=medright; current->count++;

}

void B::Split(BT med,node \*medright,node \*current,int pos,BT \*newmedian,node

\*\*newright)

{

int i;

int median; if(pos<=MIN)

median=MIN;

# Data Structures using C++ LAB FA23

else

median=MIN+1;

\*newright=new node; for(i=median+1;i<=MAX;i++)

{

(\*newright)->entry[i-median]=current->entry[i]; (\*newright)->branch[i-median]=current->branch[i];

}

(\*newright)->count=MAX-median; current->count=median; if(pos<=MIN)

InsertIn(med,medright,current,pos);

else

InsertIn(med,medright,\*newright,pos-median);

\*newmedian=current->entry[current->count];

(\*newright)->branch[0]=current->branch[current->count]; current->count--;

}

void B::Delete(Type target,node \*\*root)

{

node \*prev; Del\_node(target,\*root); if((\*root)->count==0)

{

prev=\*root;

\*root=(\*root)->branch[0]; free(prev);

}

}

void B::Del\_node(Type target,node \*current)

{

int pos; if(!current)

{

}

else

{

cout<<"Item not in the Btree\n"; return;

if(SearchNode(target,current,&pos)) if(current->branch[pos-1])

{

Successor(current,pos);

Del\_node(current->entry[pos].key,current->branch[pos]);

}

else

Remove(current,pos);

else

# Data Structures using C++ LAB FA23

Del\_node(target,current->branch[pos]); if(current->branch[pos])

if(current->branch[pos]->count<MIN) Adjust(current,pos);

}

}

void B::Remove(node \*current,int pos)

{

int i;

for(i=pos+1;i<=current->count;i++)

{

current->entry[i-1]=current->entry[i]; current->branch[i-1]=current->branch[i];

}

current->count--;

}

void B::Successor(node \*current,int pos)

{

node \*leaf;

for(leaf=current->branch[pos];leaf->branch[0]; leaf=leaf->branch[0]);

current->entry[pos]=leaf->entry[1];

}

void B::Adjust(node \*current,int pos)

{

if(pos==0)

if(current->branch[1]->count > MIN) MoveLeft(current,1);

else Combine(current,1);

else if(pos==current->count)

if(current->branch[pos-1]->count > MIN) MoveRight(current,pos);

else Combine(current,pos);

else if(current->branch[pos-1]->count > MIN) MoveRight(current,pos);

else if(current->branch[pos+1]->count > MIN) MoveLeft(current,pos+1);

else Combine(current,pos);

}

void B::MoveRight(node \*current,int pos)

{

int i;

# Data Structures using C++ LAB FA23

node \*t;

t=current->branch[pos]; for(i=t->count;i>0;i--)

{

t->entry[i+1]=t->entry[i];

t->branch[i+1]=t->branch[i];

}

t->branch[1]=t->branch[0]; t->count++;

t->entry[1]=current->entry[pos]; t=current->branch[pos-1];

current->entry[pos]=t->entry[t->count];

current->branch[pos]->branch[0]=t->branch[t->count]; t->count--;

}

void B::MoveLeft(node \*current,int pos)

{

int c; node \*t;

t=current->branch[pos-1]; t->count++;

t->entry[t->count]=current->entry[pos];

t->branch[t->count]=current->branch[pos]->branch[0]; t=current->branch[pos];

current->entry[pos]=t->entry[1]; t->branch[0]=t->branch[1];

t->count--;

for(c=1;c<=t->count;c++) t->entry[c]=t->entry[c+1];

t->branch[c]=t->branch[c+1];

}

void B::Combine(node \*current,int pos)

{

int c;

node \*right; node \*left;

right=current->branch[pos]; left=current->branch[pos-1]; left->count++;

left->entry[left->count]=current->entry[pos]; left->branch[left->count]=right->branch[0]; for(c=1;c<=right->count;c++)

{

left->count++;

left->entry[left->count]=right->entry[c]; left->branch[left->count]=right->branch[c];

}

# Data Structures using C++ LAB FA23

for(c=pos;c<current->count;c++)

{

current->entry[c]=current->entry[c+1]; current->branch[c]=current->branch[c+1];

}

current->count--; free(right);

}

void B::InOrder(node \*root)

{

int pos; if(root)

{

InOrder(root->branch[0]); for(pos=1;pos<=root->count;pos++)

{

cout<<" "<<root->entry[pos].key; InOrder(root->branch[pos]);

}

}

}

int main()

{

int choice,targetpos; Type inKey;

BT New; B obj;

node \*root,\*target; root=NULL; while(1)

{

cout<<"\n IMPLEMENTATION OF B-TREE\n"; cout<<"\n1.INSERT\n2.DELETE\n3.SEARCH\n4.DISPLAY\n5.EXIT\n";

cout<<"Enter Your Choice\n"; cin>>choice;

switch(choice)

{

case 1:cout<<"enter the key to be inserted\n"; fflush(stdin);

gets(New.key); root=obj.Insert(New,root); break;

case 2:cout<<"enter the key to be deleted\n"; fflush(stdin);

gets(New.key);

cout<<"Deleting the desired item\n"; obj.Delete(New.key,&root);

# Data Structures using C++ LAB FA23

break;

case 3:cout<<"enter the key to be searched\n"; fflush(stdin);

gets(New.key); target=obj.Search(New.key,root,&targetpos); if(target)

cout<<"The searched item"<<target->entry[targetpos].key<<endl; else

cout<<"Element not found\n"; break;

case 4:cout<<"\n InOrder Traversal\n"; obj.InOrder(root);

break; case 5:exit(0);

}

}

}

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1.What is a B-Tree |  |  |  |
| 2.Application of B-Tree |  |  |  |

# Data Structures using C++ LAB FA23

Week 11 : Write a C++ program to perform the following operations

a) Insertion into an AVL-tree b) Deletion from an AVL-tree

Aim: To implement AVL tree Source code:

# include <iostream.h> # include <stdlib.h>

# include <conio.h> struct node

{

int element; node \*left; node \*right; int height;

};

typedef struct node \*np; class bstree

{

public:

void insert(int,np &); void del(int, np &); int deletemin(np &); void find(int,np &); np findmin(np);

np findmax(np);

void copy(np &,np &); void makeempty(np &); np nodecopy(np &); void preorder(np);

void inorder(np); void postorder(np); int bsheight(np); np srl(np &);

np drl(np &); np srr(np &); np drr(np &); int max(int,int);

int nonodes(np);

};

// Inserting a node void bstree::insert(int x,np &p)

{

if (p == NULL)

{

p = new node; p->element = x; p->left=NULL;

p->right = NULL; p->height=0;

# Data Structures using C++ LAB FA23

}

else

{

if (p==NULL)

cout<<"Out of Space";

if (x<p->element)

{

insert(x,p->left);

if ((bsheight(p->left) - bsheight(p->right))==2)

{

if (x < p->left->element) p=srl(p);

else

p = drl(p);

}

}

else if (x>p->element)

{

insert(x,p->right);

if ((bsheight(p->right) - bsheight(p->left))==2)

{

if (x > p->right->element) p=srr(p);

else

}

}

p = drr(p);

else

}

cout<<"Element Exists";

int m,n,d; m=bsheight(p->left); n=bsheight(p->right); d=max(m,n);

p->height = d + 1;

}

//Finding the Smallest np bstree::findmin(np p)

{

if (p==NULL)

{

cout<<"Empty Tree "; return p;

}

else

{

while(p->left !=NULL) p=p->left;

return p;

}

# Data Structures using C++ LAB FA23

}

//Finding the Largest

np bstree::findmax(np p)

{

if (p==NULL)

{

cout<<"Empty Tree "; return p;

}

else

{

while(p->right !=NULL) p=p->right;

return p;

}

}

//Finding an element

void bstree::find(int x,np &p)

{

if (p==NULL)

cout<<" Element not found "; else

if (x < p->element) find(x,p->left);

else

if (x>p->element) find(x,p->right);

else

cout<<" Element found !";

}

//Copy a tree

void bstree::copy(np &p,np &p1)

{

makeempty(p1); p1 = nodecopy(p);

}

// Make a tree empty void bstree::makeempty(np &p)

{

np d;

if (p != NULL)

{

makeempty(p->left); makeempty(p->right); d=p;

free(d); p=NULL;

}

}

# Data Structures using C++ LAB FA23

// Copy the nodes np bstree::nodecopy(np &p)

{

np temp;

if (p==NULL)

return p; else

{

temp = new node;

temp->element = p->element; temp->left = nodecopy(p->left); temp->right = nodecopy(p->right); return temp;

}

}

// Deleting a node void bstree::del(int x,np &p)

{

np d;

if (p==NULL)

cout<<"Element not found "; else if ( x < p->element)

del(x,p->left);

else if (x > p->element) del(x,p->right);

else if ((p->left == NULL) && (p->right == NULL))

{

d=p; free(d); p=NULL;

cout<<" Element deleted !";

}

else if (p->left == NULL)

{

d=p; free(d); p=p->right;

cout<<" Element deleted !";

}

else if (p->right == NULL)

{

d=p;

p=p->left; free(d);

cout<<" Element deleted !";

}

else

p->element = deletemin(p->right);

}

# Data Structures using C++ LAB FA23

int bstree::deletemin(np &p)

{

int c;

cout<<"inside deltemin"; if (p->left == NULL)

{

c=p->element; p=p->right; return c;

}

else

{

c=deletemin(p->left); return c;

}

}

void bstree::preorder(np p)

{

if (p!=NULL)

{

cout<<p->element<<"-->"; preorder(p->left); preorder(p->right);

}

}

// Inorder Printing void bstree::inorder(np p)

{

if (p!=NULL)

{

inorder(p->left);

cout<<p->element<<"-->"; inorder(p->right);

}

}

// PostOrder Printing void bstree::postorder(np p)

{

if (p!=NULL)

{

postorder(p->left); postorder(p->right); cout<<p->element<<"-->";

}

}

int bstree::max(int value1, int value2)

{

return ((value1 > value2) ? value1 : value2);

}

int bstree::bsheight(np p)

# Data Structures using C++ LAB FA23

{

int t;

if (p == NULL)

return -1;

else

{

}

}

t = p->height; return t;

np bstree:: srl(np &p1)

{

np p2;

p2 = p1->left;

p1->left = p2->right; p2->right = p1;

p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1; p2->height = max(bsheight(p2->left),p1->height) + 1;

return p2;

}

np bstree:: srr(np &p1)

{

np p2;

p2 = p1->right;

p1->right = p2->left; p2->left = p1;

p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1; p2->height = max(p1->height,bsheight(p2->right)) + 1;

return p2;

}

np bstree:: drl(np &p1)

{

p1->left=srr(p1->left); return srl(p1);

}

np bstree::drr(np &p1)

{

p1->right = srl(p1->right); return srr(p1);

}

int bstree::nonodes(np p)

{

int count=0; if (p!=NULL)

{

nonodes(p->left); nonodes(p->right); count++;

}

return count;

# Data Structures using C++ LAB FA23

}

int main()

{

//clrscr();

np root,root1,min,max;//,flag;

int a,choice,findele,delele,leftele,rightele,flag; char ch='y';

bstree bst;

//system("clear"); root = NULL; root1=NULL;

while(1)

{

cout<<" \nAVL Tree\n"; cout<<" ========\n";

cout<<"1.Insertion\n2.FindMin\n"; cout<<"3.FindMax\n4.Find\n5.Copy\n"; cout<<"6.Delete\n7.Preorder\n8.Inorder\n"; cout<<"9.Postorder\n10.height\n11.EXIT\n"; cout<<"Enter the choice:";

cin>>choice; switch(choice)

{

case 1:

case 2:

case 3:

case 4:

cout<<"New node's value ?"; cin>>a;

bst.insert(a,root); break;

if (root !=NULL)

{

min=bst.findmin(root);

cout<<"Min element : "<<min->element;

}

break;

if (root !=NULL)

{

max=bst.findmax(root);

cout<<"Max element : "<<max->element;

}

break;

cout<<"Search node : "; cin>>findele;

if (root != NULL)

bst.find(findele,root);

break;

case 5:

# Data Structures using C++ LAB FA23

case 6:

case 7:

case 8:

case 9:

bst.copy(root,root1); bst.inorder(root1); break;

cout<<"Delete Node ?"; cin>>delele; bst.del(delele,root); bst.inorder(root);

break;

cout<<" Preorder Printing... :"; bst.preorder(root);

break;

cout<<" Inorder Printing ";

bst.inorder(root); break;

cout<<" Postorder Printing ";

bst.postorder(root); break;

case 10:

cout<<" Height and Depth is "; cout<<bst.bsheight(root);

//cout<<"No. of nodes:"<<bst.nonodes(root); break;

case 11:exit(0);

}

}

return 0;

}

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1. What is a height balanced tree |  |  |  |
| 2. Explain What is the necessity of rotations in AVL tree. |  |  |  |

# Data Structures using C++ LAB FA23

Week 12 : Write a C++ program to implement all the functions of a dictionary (ADT)

**Aim:** To to implement all the functions of a dictionary (ADT)

**Source code:** #include<stdlib.h> #include<iostream.h> class node

{

public: int key;

int value; node\*next;

};

class dictionary:public node

{ int k,data; node \*head;

public: dictionary();

void insert\_d( ); void delete\_d( ); void display\_d( );

};

dictionary::dictionary( )

{ head=NULL;

}

//code to push an val into dictionary; void dictionary::insert\_d( )

{

node \*p,\*curr,\*prev;

cout<<"Enter an key and value to be inserted:"; cin>>k;

cin>>data; p=new node; p->key=k;

p->value=data; p->next=NULL;

if(head==NULL) head=p;

else

{

curr=head;

while((curr->key<p->key)&&(curr->next!=NULL))

{ prev=curr; curr=curr->next;

}

if(curr->next==NULL)

{

if(curr->key<p->key)

{ curr->next=p;

prev=curr;

}

# Data Structures using C++ LAB FA23

else { p->next=prev->next; prev->next=p;

}

}

else

{

p->next=prev->next;

prev->next=p;

}

cout<<"\nInserted into dictionary Sucesfully \n";

}

}

void dictionary::delete\_d( )

{

node\*curr,\*prev;

cout<<"Enter key value that you want to delete..."; cin>>k;

if(head==NULL)

cout<<"\ndictionary is Underflow";

else

{ curr=head; while(curr!=NULL)

{

if(curr->key==k)

break; prev=curr; curr=curr->next;

}

}

if(curr==NULL)

cout<<"Node not found...";

else

{

if(curr==head)

head=curr->next;

else

delete curr;

prev->next=curr->next;

cout<<"Item deleted from dictionary...";

}

}

void dictionary::display\_d( )

{

node\*t;

if(head==NULL)

cout<<"\ndictionary Under Flow";

else

{

cout<<"\nElements in the dictionary are \n";

t=head;

# Data Structures using C++ LAB FA23

while(t!=NULL)

{

cout<<"<"<<t->key<<","<<t->value<<">"; t=t->next;

}

}

}

int main( )

{

int choice; dictionary d1;

while(1)

{

cout<<"\n\n\*\*\*Menu for Dictrionay operations\*\*\*\n\n"; cout<<"1.Insert\n2.Delete\n3.DISPLAY\n4.EXIT\n"; cout<<"Enter Choice:";

cin>>choice; switch(choice)

{

case 1: d1.insert\_d();

break; case 2: d1.delete\_d( );

break; case 3: d1.display\_d( );

break;

case 4: exit(0);

default:cout<<"Invalid choice...Try again...\n";

}

}

}

Results

Assignment:-

|  |  |  |  |
| --- | --- | --- | --- |
| Task | Date | Sign | Remark |
| 1. Write a program to implement hash table |  |  |  |
| 2.Explain DFS and BFS |  |  |  |