**Class/ Semester:** B.Tech – IT 3rd Sem

**Subject with Code:** Data Structures & Algorithms (BTIT301-18)

**Assignment No. II**

1. Write a function *TotalChars*  that returns the total number of occurrences of the character *ch* in each string in *list*.

// precondition: list has a header node

// postcondition: returns # of occurrences of ch in all nodes of list

Sol :

#include <iostream>

#include<string.h>

#include<stdlib.h>

#include<ctype.h>

int total;

using namespace std;

void clean()

{

system("cls");

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout<<"\n CREATED BY SACHIN PATWAL \n";

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n";

}

struct node

{

char info[50];

int n;

node \* next;

};

node \*temp=new node;

node\*header=new node;

class link

{

public: node \* start;

link()

{

start=NULL;

}

void input(char);

void display();

int TotalChars(char,node \*);

}l;

int link::TotalChars(char c,node \*temp) //CALCULATES AND RETURNS NO OF CHARACTERS

{

int total=0;

int n=strlen(temp->info);

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

{

if(temp->info[i]==c||temp->info[i]==toupper(c)||temp->info[i]==tolower(c))

total+=1;

}

return total; //RETURNS NUMBER OF CHARACTERS IN THE STRING

}

void link::input(char c) //TAKES INPUT

{

int n;

total++;

node \*temp=new node;

node \*ptr=NULL;

cout<<"\n\nENTER THE STRING:";

cin>>temp->info;

temp->n=0;

temp->next=NULL;

n=TotalChars(c,temp); //CALLS TOTOALCHAR FUNCTION

temp->n=n;

if(start->next==NULL)

{

header->next=temp;

}

else

{

ptr=start->next;

while(ptr->next!=NULL)

{

ptr=ptr->next;

}

ptr->next=temp;

}

cout<<"\n\nSUCCESFULLY INSERTED\n\n";

system("pause");

}

void link::display()

{

clean();

if(start->next==NULL)

{

cout<<"\nLINK LIST IS EMPTY\n";

}

else

{ node \*temp=start;

node \* ptr=start->next;

cout<<temp->info<<" CONTAINS TOTAL : "<<temp->n<<" NODES";

cout<<"\n\nLINK LIST IS:\n\n";

while(ptr!=NULL)

{

cout<<"==> "<<ptr->info<<" TOTAL OCCURENCE :"<<ptr->n;

cout<<"\n\n";

ptr=ptr->next;

}

}

cout<<"\n";

system("pause");

}

int main()

{ clean();

char a;

int i;

strcpy(header->info,"\nTHIS IS A HEADER NODE"); //CREATING HEADER NODE;

header->next=NULL;

l.start=header;

cout<<"\nENTER THE CHARACTER WHOES OCCURENCE YOU WANT TO FIND:";

cin>>a;

while(1)

{ clean();

cout<<"1.FOR INSERT\n\n";

cout<<"2.FOR DISPLAY\n\n";

cout<<"3.FOR EXIT\n\n\n";

cout<<"ENTER YOUR CHOISE:";

cin>>i;

switch(i)

{

case 1: l.input(a);

header->n=total; //HEADER NODE TAKING TO NUMBERS OF NODES IN LINK LIST

break;

case 2: l.display();

break;

case 3: cout<<"\nPROGRAM HAS BEEN SUCCESFULLY CLOSED\n";

exit(0);

default: cout<<"\nINVALID OPTION\n";

break;

}

}

return 0;

}

1. Write a function *RemoveAllBut*  that removes all nodes except for those nodes containing a specified item.

// postcondition: all nodes except those containing value have been removed from list

Sol:

#include<iostream>

#include<stdlib.h>

using namespace std;

void clean()

{

system("cls");

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout<<"\n CREATED BY SACHIN PATWAL \n";

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n";

}

struct node

{

int info;

node \* next;

};

class link

{

private: node \*start;

node \* last;

public: link()

{

start=NULL;

last=NULL;

};

void insertione();

void display();

void RemoveAllBut(int);

}l;

void link::insertione()

{

node \*temp=new node;

cout<<"\n\nENTER THE ELEMENT:";

cin>>temp->info;

temp->next=NULL;

if(start==NULL)

{

start=temp;

last=temp;

}

else

{

last->next=temp;

last=last->next;

}

}

void link:: RemoveAllBut(int x)

{

int f=0;

node\* ptr=start;

node \* prev=start;

node \* temp=NULL;

while(ptr!=NULL)

{

if(start->info!=x)

{

if(ptr->next==NULL)

{

start=NULL;

ptr=NULL;

f++;

}

else

{

temp=start;

start=start->next;

ptr=start;

prev=start;

delete(temp); }

}

else

{

if(ptr->info!=x)

{

while(ptr->info!=x&&ptr->next!=NULL)

{

temp=ptr;

ptr=ptr->next;

delete temp;

}

if(ptr->next==NULL)

prev->next=NULL;

else

{

prev->next=ptr;

prev=ptr; }

}

}

if(f==0)

{

prev=ptr;

last=ptr;

ptr=ptr->next;

}

} cout<<"\n\nDELETION SUCCESFUL\n\n";

system("pause"); }

void link::display()

{ clean();

node \* p=start;

if(start==NULL)

{

cout<<"\n\nLIST IS EMPTY\n\n\n";

}

else

{

while(p!=NULL)

{

cout<<p->info<<" ==> ";

p=p->next;

}

cout<<"NULL";

cout<<"\n\n\n";

}

system("pause");

}

int main()

{

clean();

while(1)

{ int a;

clean();

cout<<"PRESS 1 FOR INSERT AN ELEMENT\n\n";

cout<<"PRESS 2 FOR CALL REMOVE FUNCTION\n\n";

cout<<"PRESS 3 FOR DISPLAY\n\n";

cout<<"PRESS 4 FOR EXIT\n\n\n";

cout<<"ENTER YOUR CHOISE:";

cin>>a;

switch(a)

{

case 1: l.insertione();

break;

case 2: int x;

cout<<"\nENTER THE SPECIFIED ITEM:";

cin>>x;

l.RemoveAllBut(x);

break;

case 3: l.display();

break;

case 4: exit(0);

default: cout<<"\n\nINVALID OPTION";

break;

}

}

return 0;

}

1. Write a function *ListSplit* that will split a list containing an even number of nodes, say *2k*, into two lists each of which contains *k* nodes. The function should have three parameters and should work so that the function call

ListSplit(list,sub1,sub2);

will create new lists pointed to by parameters *sub1*, and *sub2* from the list initially referenced by

parameter list.

Sol:

#include <iostream>

#include<stdlib.h>

using namespace std;

void clean()

{

system("cls");

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n";

cout<<"\n CREATED BY SACHIN PATWAL \n";

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n\n";

}

struct node

{

int info;

node \* next;

};

node\* list1=NULL;

node\* sub1=NULL;

node\* sub2=NULL;

int sizeofll(node \*list1) //CALCULATES NUMBER OF NODES

{ node \*ptr=list1;

int n=0;

while(ptr!=NULL)

{

n++;

ptr=ptr->next;

}

return (n);

}

void insertione()

{

node \*temp=new node;

node \*temp1=list1;

cout<<"\n\nENTER THE ELEMENT:";

cin>>temp->info;

temp->next=NULL;

if(list1==NULL)

list1=temp;

else

{

while(temp1->next!=NULL)

temp1=temp1->next;

temp1->next=temp;

}

cout<<"\n\nINSETION SUCCESSFUL\n\n";

system("pause");

}

void display(node \*list1,node\*sub1,node\*sub2)

{

cout<<"\n\n"; int a;

node \*tmp;

clean();

cout<<"WHICH LIST U WANT TO SEE";

cout<<"\n\nPRESS 1.FOR DISPLAY FULL LIST\n";

cout<<"\nPRESS 2.FOR DISPLAY SUB LIST 1\n";

cout<<"\nPRESS 3.FOR DISPLAY SUB LIST 2\n";

cout<<"\n\nENTER YOUR CHOISE:";

cin>>a;

if(a==1)

tmp=list1;

if(a==2)

tmp=sub1;

if(a==3)

tmp=sub2;

if(tmp==NULL)

{

cout<<"\nLIST IS EMPTY\n\n";

}

else

{

while(tmp!=NULL)

{

cout<<tmp->info<<" => ";

tmp=tmp->next;

} cout<<"NULL"<<endl;

}

cout<<"\n\n";

system("pause");

}

void ListSplit(node \* &list1,node \* &sub1,node \* &sub2) //SPLIT THE LIST INTO TWO PARTS

{

int n=sizeofll(list1);

node \*ptr=list1;

node \*last=NULL;

node \*last1=NULL;

int i=0;

while(ptr!=NULL)

{

node \* temp=new node;

temp->info=ptr->info;

temp->next=NULL;

if(i<n/2)

{

if(sub1==NULL)

{

sub1=temp;

last=temp;

}

else

{

last->next=temp;

last=last->next;

}

}

else

{

if(sub2==NULL)

{

sub2=temp;

last1=temp;

}

else

{

last1->next=temp;

last1=last1->next;

}

}

ptr=ptr->next;

i++;

}

cout<<"\n\nSPLIT SUCCESSFULL\n\n";

system("pause");

}

int main()

{ clean();

int a;

while(1)

{

clean();

cout<<"1.FOR INSERTION IN THE LIST\n\n";

cout<<"2.FOR DISPLAYING ALL ELEMENTS IN LINK LIST\n\n";

cout<<"3.TO CALL SPLIT FUNCTION\n\n";

cout<<"4.FOR EXIT\n";

cout<<"\nENTER YOUR CHOISE:";

cin>>a;

switch(a)

{

case 1: insertione();

break;

case 2: display(list1,sub1,sub2);

break;

case 3: ListSplit(list1,sub1,sub2);

break;

case 4: clean();

cout<<"\nPROGRAM HAS CLOSED SUSSESFULLY\n\n";

exit(0);

default: cout<<"\n\nINVALID OPTION\n\n";

system("pause");

break;

}

}

return 0;

}