Project assignment on doubly and circular linked lists engineering essay

As a computer engineer I would like to deal with this topic following a step by step approach. Before going into the details and the programs of doubly and circular linked lists we need to concentrate upon the meaning of the term linked list.

The meaning of the terminology " link list" can further be dealed by dividing it into chunks . In a layman's term a link list is a logical collection of data stored in a system's memory in which every record that is the part of the given list has a pointer or link part that contains the address of the next record . This is how it works!

Linked lists are used to organize data in specific desired logical orders, independent of the memory address each record is assigned to.

Firstly, we would discuss linked lists in the terms of singly linked lists or what we call as SLLs . SLLs can be thought to be non-contiguous block of memory consisting of finite number of nodes with the address of the successor node stored in the link part of the preceding node. Each node has a DATA part and a LINK part. DATA PART STORES THE ACTUAL DATA ELEMENT WHILE THE LINK PART CONTAINS THE ADRESS OF THE NEXT NODE i. e. THE ONE JUST AFTER THAT NODE EXCEPT FOR THE LAST NODE THAT DOESN'T POINT TO ANYTHING OR WE CAN SAY NULL.

This is depicted in the following diagram:-

But to beginners this may sound confusing that if one node stores the address of the next node then where is the address of the first node stored?

However this question isn't a big deal . To counter this problem we allocate memory for a dummy node that will store the address of the first node . This head or the dummy node has only the link part (and not the data part). This node is always supposed to point to the first node of the list.

OPERATIONS POSSIBLE WITH SLLs

CREATE

- CREATING THE LIST FOR THE FIRST TIME.

INSERT

- INSERTING AN ELEMENT TO THE EXISTING LIST.

DELETE

- DELETING AN ELEMENT FROM THE EXISTING LIST.

SEARCH

- SEARCHING AN ELEMENT FROM THE EXISTING LIST.

REMEMBER: SLLs CAN BE TRAVERSED UNIDIRECTIONALLY ONLY i. e. ONLY IN FORWARD DIRECTION.

Since this assignment deals with doubly and circular linked list the programs on SLLs won't be discussed in detail. Only program on creating a SLL is included :-

THIS IS SIMPLE FUNCTION IN C++ DEPICTING HOW TO CREATE A SINGLY LINKED LIST

```
STRUCT nodeT {
```

INT data;

nodeT* link; };

nodeT* BUILD() {

nodeT *first= NULL,*new node;

INT num;

COUT <<" enter a list of integers ending with 178"< CIN>> num;

WHILE(num != 178) {

New node = new nodeT; //create a node

ASSERT (new node! = NULL); //program end if memory not allocated

New node -> data = num; //stores the data in the new node

New node -> link = first; //put new node at the start of list

First= new node; //update the dummy pointer of the list

Cin>> num ;}; //read the next number

RETURN first;}// this program is also called building list from backwards ITS OUTPUT CAN BE SEEN AS IN BELOW BLOCK DIAGRAM

C: Usersthe PANKESHAppDataLocalMicrosoftWindowsTemporary Internet FilesContent. IE57OPVY3E3MCj01978470000[1]. wmf

As we have discussed earlier that linked lists are such data structures that contain linked nodes each containing a DATA part and a LINK part. But contrary to SLLs, in doubly linked lists each node has two link parts one to store the address of the succeeding node and the other for the preceding node. This makes doubly linked lists bidirectional i. e. they can be traversed in either direction, forward or backward. This is shown in the following diagram:-

NODE 3 NODE 2 NODE 1

But there is a disadvantage to the use of doubly linked lists also, that is it requires more of memory space per node as two pointers will be needed but their advantage of sequential access in either direction make its manipulation quite easier which overcome its former shortcoming.

C: Usersthe PANKESHAppDataLocalMicrosoftWindowsTemporary Internet FilesContent. IE57OPVY3E3MPj04424300000[1]. jpg

OPERATIONS DONE ON DOUBLY LINKED LISTS ARE ALMOST THE SAME AS THAT ON SLLs BUT HAVE BEEN DISCUSSED IN DETAIL HERE

- Ÿ CREATING AN EMPTY LIST
- Ÿ ADDING A NODE AT THE END
- Ÿ ADDING A NODE IN THE BEGINNING
- Ÿ DELETING A NODE IN THE BEGINNING
- Ÿ DELETING A NODE AT THE END

Ÿ FORWARD TRAVERSAL

Ÿ REVERSE TRAVERSAL

Ÿ INSERTING A NODE AT A SPECIFIC PLACE

Ÿ DELETING A LIST

MISCELLENEOUS:

USE OF CONSTRUCTORS IN DOUBLY LINKED LISTS

IF THE LINKED LIST IS MADE USING CLASSES INSTEAD OF STRUCTURES THEN DEFAULT CONSTRUCTORS CAN BE USED TO INITIALISE THE WHOLE LIST TO A PARTICULAR VALUE

EG: IT SETS FIRST AND LAST TO NULL AND COUNT TO 0.

SYNTAX:

Template

Doubly_linked_list :: doubly_linked_list()

{ first= null;

Last= null;

Count= 0; } ;

FOLLOWING IS A C++ PROGRAM DEPICTING ALL THE OPERATIONS MENTIONED ABOVE THAT CAN BE PERFORMED USING DOUBLY LINKED LISTS

#include // header files in c++

```
#include

typedef struct doubly //self referential structure for making a linked list

{ int info;

struct doubly*frwd; //frwd and back are the two pointers of the linked list

struct doubly*back;

} node; //node is a global object

void create empty (node**frwd, node**back) //create empty is to set the

pointers to null

{
*frwd=*back= NULL;

}
void add_at_end(node**frwd, node**back, int element) // add_at_end is to

add a node in the end

{
node*ptr;

ptr= new node;

ptr-> info= element;

ptr-> frwd= NULL;

if(*frwd== NULL)
```

```
{
ptr-> back= NULL;

*frwd= ptr;

*back= ptr;

}
else

{
ptr-> back=*back;

(*back)-> frwd= ptr;

*back= ptr;

}

}
void add_at_beg(node**frwd, node**back, int item) // add_at_beg is to add a

node in the start

{ node*ptr;

ptr= new node;

ptr-> info= item;

ptr-> back=(node*)NULL;

ptr-> frwd=*frwd;
```

```
if(*frwd==(node*)NULL;

*back= ptr;

*frwd= ptr;

}
void delete_at_beg(node**frwd, node**back) // delete_at_beg is to delete a
node in the start

}
node*ptr;

if(*frwd==(node*)NULL)

return();

if((frwd))-> frwd==(node*)NULL)

{ ptr=*frwd;

*frwd=*back=(node*)NULL;

delete(ptr):

} else

{
ptr=*frwd

*frwd=(*frwd-> frwd;(*frwd)-> back=(node*)NULL;
```

```
delete(ptr);

}}
void delete_at_end(node**frwd, node**back)) // delete_at_beg is to delete a
node in the end

{
node*ptr;

if(*frwd==(node*)NULL)

return;

if((*frwd)-> frwd==(node*)NULL)

{ ptr=*frwd;

*frwd=*back=(node*)NULL

delete (ptr);

} else

{ ptr=*back;

*back= ptr-> back;

(*back)-> frwd=(node*)NULL;

delete(ptr);
```

}}

void inordertrav(node*)NULL; // inordertrav is to traverse the list in the forward direction

{ while(frwd!= NULL)

{ printf("%d", frwd-> info);

frwd= frwd-> frwd;

}

}

void reverse ordertrav(node*back) // reverseordertrav is to traverse the list in the back direction

{ while(back!= node*)NULL)

{ cout < info;

back= back-> back;

}

}

node*search(node*frwd, int item) //search is to search a given element in the list

{ while(!=(node*)NULL &&frwd-> info!= item)

frwd= frwd-> frwd;

return frwd;

```
} // insert-after-node is to insert a node at a specified position after the
provided node

void insert-after-node(node**frwd, node**frwd, int item, int after)

{ node*loc,*ptr;

ptr= new node;

ptr-> info= item;

loc=*frwd;

loc= search(loc, after);

if(loc==(node*)after)

{
cout <<" element doesnot exit"< return;

}
else if(loc-> frwd==(node*)NULL)

{ ptr-> frwd=(node*)NULL;

ptr-> frwd= ptr;

*frwd= ptr;

} else

{ ptr-> frwd= loc-> frwd;
```

```
ptr-> back= loc;

(loc-> frwd)-> pre= ptr;

loc-> frwd= ptr;

}
} // insert-before-node is to insert a node at a specified position before the
provided node

void insert-before-node(node**frwd, int item, int before)

{
node*ptr,*loc;

ptr= new node;

ptr-> info= item;

loc=*frwd;

loc= search(loc, before);

if(loc==(node*)NULL

{ cout <<" elements does not exit"< return;

}
else if(loc-> back==(node*)NULL)

{
ptr-> back=(node*)NULL:
```

```
loc-> back= ptr;

ptr-> frwd=*frwd;

*frwd= ptr;

} else

{ ptr-> back= loc-> back;

ptr-> frwd= loc;

(loc-> back)-> frwd= ptr;

loc-> back= ptr;

}

}
void delete-list(node**frwd**frwd) //delete-list is to destroy the created list

{ node*ptr;

while(*frwd!=(node*)NULL)

{
ptr=*frwd;

*frwd=(*frwd)-> frwd;

(*frwd)-> back=(node*)NULL;

delete(ptr);
```

```
} *
frwd=(node*)NULL;

}
void main()

{ node,*frwd,*frwd;

int ch, element, after;

create_empty(&frwd,&back);

while(1)

{
cout <<" 1. add at end"< cout <" 2. add at beggining n";

cout <<" 3. inorder traverse n";

cout <<" 4. reverse order traverse n";

cout <<" 5. insert after a node n";

cout <<" 6. insert before a node n";

cout <<" 7. delete from beggining n";

cout <<" 8. delete from end n");

cout <<" 9. delete entire list n");

cout <<" 10. exit n";
```

```
cout <<" enter your choice n";

cin>> ch;

switch(ch)

{ case 1:

cout <<" enter element");

cin>> element;

add_at_end(&frwd,&back, element);

getch();

break;

case 2:

cout <<" enter element");

cin>> element;

add_at_beg(&frwd,&back, element);

break;

case 3:

in ordertrav(frwd);

getch();
```

```
break;

case 4:

reverse-order-trav(back);

getch();

break;

case 5:

cout <<" enter element after which new element have to insert n");

cin>> after;

cout <<" enter element";

cin>> element;

insert-after-node(&frwd,&back, element, after);

break;

case 6:

cout <<" enter element before which new element have to insert n");

cin>> after;

cout <<" enter element";

cin>> element;
```

```
insert-before-node(&frwd, element, after);

break;

case 7:

delete-at-beg(&frwd,&back);

break;

case 8:

delete-at-end(&frwd,&back);

break;

case 9:

delete-list(&frwd,&back);

break;

case 10:

exit();
```

}

}

}

SOME INTERESTING FACTS :-

One byte means 8 bits and a nibble means 4 bits.

First hard disk available was of 5MB

Ethernet is the registered trademark of Xerox.

Google uses over 10000 network computers to crawl the web

Google can be queried in 26 languages

The floppy disk was patented by Allen sugar in 1946.

More than 80% of web pages are in English.

88% percent web pages have very low traffic rate.

An average American is dependent on 250 computers.

Internet is most fastest growing platform for advertisement.

About one third of CDs are pirated

About 76% soft wares used in India are pirated.

Only 10% of the WebPages are used by the search engines

" I feeling Lucky" This button is used by negligible number of people on net.

CONTINUED..

CIRCULAR LINKED LIST

A circularly linked list is just like representing an array that are supposed to be naturally circular , e. g. in this a pointer to any node serves as a handle to the whole list.

With a circular list, a pointer to the last node gives easy access also to the first node , by following one link. Using circular lists one has access to both ends of the list. A circular structure allows one to handle the structure by a single pointer, instead of two.

Thus we see , all nodes are linked in a continuous circle form without using any NULL pointer in the last node. Here the next node after the last node is the first node . Elements can be added to the back of the list and removed from the front in a constant period of time.

We can classify circularly linked lists into two kinds- singly linked and doubly linked. Both types have advantage of its own . either of them has the ability to traverse the full list beginning at any given node. this helps us to avoid storing any FIRSTnode ï¿½ LASTnode , although if the list is empty there dwells a need of a special representation for the empty list, such as a LASTnode variable which points to some node in the list or is NULL if it is empty; we use such a LASTnode here. This representation simplifies adding and removing nodes with a non-empty list, but empty lists are then a special case. See following figure:-

FOLLOWING PROGRAM DEPICTS THE USE OF DOUBLY LINKED CIRCULAR LIST

#include

```
#include

class C_link //DEFINING A CLASS THAT

{
struct node //SELF REFERENTIAL STRUCTURE " node"

{
int data;

node *frwd;

node *back;

}*new1,*head,*tail,*ptr,*temp; //GLOBAL OBJECTS REQUIRED FOR

OPERATIONS

public:

C_link()

{
head= tail= NULL;

}
void CREATE(); //CREATE() , INSERT(), DELETE(), DISPLAYING() are the

various functions

void INSERT(); //that we operate using circular linked lists

void DELETE();
```

```
void DISPLAYING();

};
void C_link :: CREATE() //defining the CREATE() function to create a list

{
if(head== NULL)

{
new1= new node;

new1-> frwd= NULL;

new1-> back= NULL;

cout <<" enter student number :";

cin>> new1-> data;

head= new1;

tail= new1;

head-> frwd= tail;

head-> back= tail;

tail-> frwd= head;

tail-> back= head;
```

```
}
else

cout <<" CREATE done only once !";

}
void C_link :: INSERT() //INSERT() function for inserting a new node

{int i, pos;

new1= new node;

new1-> frwd= NULL;

new1-> back= NULL;

cout <<" enter student number :";

cin>> new1-> data;

cout <<" enter position you want to insert :";

cin>> pos;

if(pos== 1)

{new1-> frwd= head;

head= new1;

tail-> back= head;

tail-> frwd= head;
```

```
head-> back= tail;

}
else

{
i= 1;

temp= head;

while(i < pos-1 && temp-> frwd!= tail)

{i++;

temp= temp-> frwd;

}
if(temp-> frwd== tail)

{
new1-> frwd= tail-> frwd;

tail-> frwd= new1;

new1-> back= tail;

tail= new1;

head-> back= tail;

}
else
```

```
{
new1-> frwd= temp-> frwd;

new1-> back= temp;

temp-> frwd= new1;

new1-> frwd-> back= new1;

}}}
void C_link:: DELETE() //DELETE() function for deleting a particular node

{
int pos, i;

cout <<" Enter Position you want to Delete ?";

cin>> pos;

if(pos== 1 && head!= tail)

{ptr= head;

head= head-> frwd;

head-> back= tail;

tail-> frwd= head;

delete ptr;
```

```
}
else

{
i= 1;

temp= head;

while(i < pos-1 && temp-> frwd!= tail)

{
i++;

temp= temp-> frwd;

}
if(temp-> frwd!= tail)

{
ptr= temp-> frwd;

temp-> frwd= ptr-> frwd;

ptr-> frwd-> back= ptr-> back;

delete ptr;

}
else
```

```
{
if(temp-> frwd== tail && head!= tail)

{
ptr= tail;

tail= temp;

tail-> frwd= head;

head-> back= tail;

delete ptr;

}
else

{
head= NULL;

tail= NULL;

delete head;

delete tail;

}}}}
void C_link:: DISPLAYING() // DISPLAYING() function is used to DISPLAYING

the list in either direction

{int ch;
```

```
cout <<" 1. forward";

cout <<" 2. backward";

cout <<" Enter your choice <1/2>?";

cin>> ch;

switch(ch)

{
case 1: if(head!= NULL)

{
temp= head;

while(temp!= tail)

{
cout < data <<" ";

temp= temp-> frwd;

}
if(temp== tail)

cout < data;

}
break;

case 2 : if(tail!= NULL)
```

```
{
temp= tail;

while(temp!= head)

{
cout < data <<" ";

temp= temp-> back;

}
if(temp== head)

cout < data;

}
break;

}}
main()

{
C_link c1;

int ch;

char op;

do

{cout <<"----------Menu----------nn";
```

```
cout <<" 1. CREATE n 2. INSERT n3. DELETE n4. DISPLAYINGn";

cout <<" Enter Your choice <1.. 4> ?";

cin>> ch;

switch(ch)

{
case 1 : c1. CREATE();

break;

case 2 : c1. INSERT();

break;

case 3 : c1. DELETE();

break;

case 4 : c1. DISPLAYING();

break;

}
cout <<" Do you want to continue ?"; //while loop In case the user want to

continue using

cin>> op; //the functions that are declared formerly

}while(op=='y' || op=='Y');
```

```
getch();

}
```

OUTPUT:

1. on pressing F9

ass4. jpg

2. on pressing option 1 and entering 09173

now2. jpg

Continued...

3. pressing y and selecting option 2 ; entering 09175 ; storing at pos 2

now3. jpg

4. pressing y and selecting option 3 ; enter pos 1

now4. jpg

5. pressing y and selecting option 4 and then 1

ow6. jpg

Note: Number is 09175 ~ 9175

CONCLUSION

THIS ASSIGNMENT PURELY DESCRIBES HOW DOUBLY AND CIRCULAR LISTS CAN BE USED . WHERE DOUBLY USED TWO POITNTERS FOR THE SEQUENTIAL ACCESS OF THE NODES THE CIRCULAR LISTS MAY EITHER BE SINGLY LINKED OR DOUBLY LINKED DEPENDING UPON HOW IT SUITS OUR PURPOSE. THIS MAKES LINKED LIST AN IMPORTANT KIND OF DATA STRUCTURE. IT CAN BE USED TO IMPLEMENT BOTH STACKS AND QUEUES ALSO WHICH WE WILL STUDY ON LATER PART.

THANKS.