Whatever you cannot understand, you cannot posses

LINEAR

**DATA STRUCTURES**

9.1 Introduction......

9.2 Linked lists..........

9.3 Basic operations in a singly linked list..................

9.4 Basic operations in a doubly linked list..

9.5 Basic operations in a circular singly linked list....

9.6 Basic operations in a circular doubly linked list.

**INTRODUCTION**

Programs consist of two things, algorithms, and data structures. A good program is combination of both algorithm and a data structure. An algorithm is a step-by-step recipe for solving an instance of a problem. Every single procedure that a computer performs is algorithm. An algorithm states the actions to be executed and the order in which the action are to be executed. A data structure represents the logical relationship that exists between individual elements of data to carry out certain tasks. In other words, a data structure defines a way of organising all data items that consider not only the elements stored but also stores the relationship between the elements. To develop a program for an algorithm, we should first decide the steps and select an appropriate data structure for that algorithm. The choice and implementation of a data structure is as important for easier manipulation of data. Therefore, algorithms, and data structures are closely related to each other for developing a program.

Arrays let you access lots of data fast with the ability to access its individual elements. You can have arrays of any data type. However, you cannot make arrays bigger if your program decides it needs more space, but we can build our own set data type out of arrays to solve this problem if necessary. A data structure is an actual implementation of an array. Fig 9.1, shows the various types of data structures.

**9.1.1 PRIMARY DATA STRUCTURES**

Primary data structures are the basic data structures that directly operate upon the machine instructions. They have representations on different computers. All the basic constants (integers, floating-point numbers character constants, string constants) and pointers are considered as primary data structures. We have spent Chapter 1 and Chapter 6 in understanding primary data structures.

**9.1.2 SECONDARY DATA STRUCTURES**

Secondary data structures are more complicated data structures derived from primary data structures. They emphasize on grouping same or different data items with relationship between each data item. Secondary data structures can be broadly classified as static data structures and dynamic data structures. If a data structure is created using static memory allocation, (i.e., a data structure formed when the number of data items are known in advance) it is known as static data structure or fixed size data structure. We have studied static data structures such as arrays, structures etc., in the previous chapters. If a data structure is created, using dynamic memory allocation (i.c., a data structure formed when the number of data items are not known in advance) is known as dynamic data structure or variable size data structure.

Dynamic data structures can be broadly classified as linear data structures and non- linear data structures. Linear data structures, have a linear relation ship between its adjacent elements. Linked lists are examples of linear data structures. A linked list is a linear dynamic data structure that can grow and shrink during its execution time. A circular linked list is similar to a linked list except that the first and last nodes are interconnected.

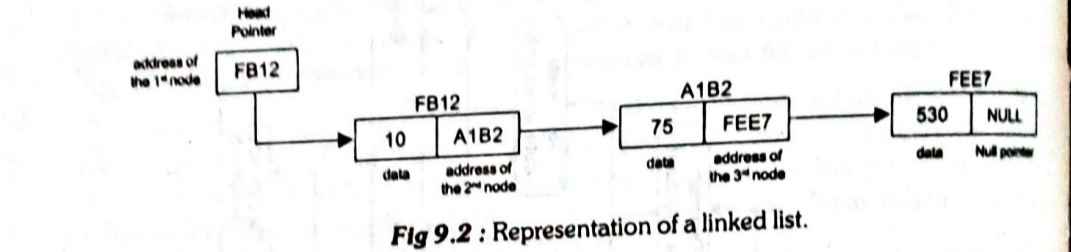
Non-linear data structures don't have a linear relationship between its adjacent elements. In a linear data structure, each node has a link which points to another node, where as in a non-linear data structure, each node may point to several other nodes. A tree is a non-linear dynamic data structure that may point to one or more nodes at a time. A tree graph is similar to tree except that it has no hierarchical relationship between its adjacent elements. In this chapter, we will discuss linear data structures in detail and we will see non- linear data structures in Chapter 10.

**LINKED LISTS**

Linked list or list is an ordered collection of elements. Each element in the list is referred as a node. Each node contains two fields namely,

* Data field
* Link field.

The data field contains the actual data (i.e., value) of the element to be stored in the list and the link field also referred as the next address field contains the address of the next node in the list. Fig 9.2, helps you understand things better.



The linked list shown in Fig 9.2, consists of three nodes, each with a data field and a link field. A linked list contains a pointer, referred as the head pointer, which points

to the first node in the list that stores the address of the first node of the list (i.e., FB12). The data field contains the actual information which is to be stored in the list. The data field of the first node stores the value 10. The link field of the first node contains the address of the second node (ie., A182). Similarly, the second node of the list stores the value 75 in the data field and the address of the third node (i.e., FEE7) in the link field. The last node of the list contains only. the information part (i.e., 530) in the data field and the address field stores the NULL pointer. This NULL pointer is used to indicate the end of a list.

**Note** that the nodes in a linked list are not ordered by their physical placement in memory (l.e., they may/may not be placed in contiguous memory locations) but by logical links stored as a part of the data in the node itself.

The address stored in a linked list are divided into three types namely,

* External address
* Internal address
* Null address.

External address is the address of the first node in the list. This address is stored in the head pointer which points to the first node in the list. The entire linked list can be accessed only with the help of the head pointer. Internal address is the address stored in each and every node of the linked list except the last node. The content stored in the link field (also referred as next address field) is the address of the

next node. Null address is the address stored by the NULL pointer of the last node of the list, which indicates the end of the list.

**TYPES OF LINKED LISTS**

There are different types of linked lists. They can be classified as,

* Singly linked list
* Doubly linked list
* Circular linked list.
* Singly Linked List

**Singly Linked List**

The list that we have seen so far is referred to as the singly linked list, in which each node has a single link to its next node. This list is also referred as a linear linked list. The head pointer points to the first node in the list and the null pointer is stored in the link field of the last node in the list, which indicates end of list. Fig 9.2, shows a singly linked list. You can traverse (move) in a singly linked list in only one direction (i.e., from head to null in a list). You cannot traverse the list in the reverse direction (i.e., from null to head in a list).

**Doubly Linked List**

For some applications, especially those where it is necessary to traverse lists in both directions, doubly linked lists work much better than singly linked lists. Doubly linked list is an advanced form of a singly linked list, in which each node contains three fields namely,

* Previous address field
* Data field
* Next address field.

The previous address field of a node contains address of its previous node. This field is also referred as the backward link field. The data field stores the information part of the node. The next address field contains the address of the next node in the list. This field is also referred as the forward link field. Fig 9.3, shows the structure of a doubly linked list.

**Circular Linked list**

circular linked list is another form of a linked list in which the last node of the list

is connected to the first node in the list. There are different types circular linked lists. They can be classified as,

* Circular singly linked list
* Circular doubly linked list.

Note that a circular linked list looks like a cyclic list wheretheir won't be any end-of-list (i.e., there is no NULL pointer). Fig 9.4, shows the structure of a circular linked singly list. Fig 9.5, shows the structure of a circular linked doubly list.

diagram

**BASIC OPERATIONS IN A SINGLY LINKED LIST**

The most commonly used linked list discussed so far is the singly linked list. The basic

operations that can be performed on singly linked lists are, Creation of a list

* Insertion of a node
* Modification of a node
* Deletion of a node
* Traversal of a list.

**9.3.1 Creation of a list**

**The**

Creating a singly linked list starts with creating a node. Sufficient memory has to be allocated for creating a node. The information is stored in the memory, allocated by using the malloc() function of type node. In program 9.1, the function getNode(), is used for creating a node.

Algorithm for declaration for the structure NODE

Struct NODE

DATA: Data Field

LINK: Link Field (Address of next struct NODE)

End Struct

Algorithm for GETNODEO

GETNODEO SIZE: INTEGER; NEWNODE: NODE

STEP 1: Set SIZE= get the size of the NODE

STEP 2: Set NEWNODE = Allocate space in memory for the size of SIZE and

return the initial address

STEP 3: Retum NEWNODE End GETNODE()

After allocating memory for the structure of type node, the information for the items (ie, roll and name) has to be read from the user. In program 9.1, the function readNode(), is used for reading details for the node from the user.

Algorithm for READNODEO

READNODE(NEWNODE: NODE) STEP 1: Read, NEWNODE-> DATA

STEP 2: Set NEWNODE-> LINK= NULL

STEP 3: Return

End READNODE()

Connect the new node with the existing list. If the list is empty, set the head pointer of the list to the new node, other wise connect the new node in the last position of the list. In program 9.1, the function createList (), is used to connect the new nodes with the list.

Algorithm for CREATELISTO

CREATELIST()

HEAD, LAST, NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE()

STEP 2: CALL READNODE(NEWNODE)

STEP 3: Set HEAD = NEWNODE STEP 4: Set LAST = NEWNODE

STEP 5: If you want to add another NODE proceed otherwise STOP.

STEP 6: Set NEWNODE = GETNODE() STEP 7: CALL READNODE(NEWNODE)

STEP 8: Assign LAST -> LINK= NEWNODE

STEP 9: Assign LAST = LAST -> LINK

STEP 10: Goto STEP 5

END CREATELISTO

INSERTION OF A NODE

One of the most primitive operations that can be done in a singly linked list is the insertion of a node. Memory is to be allocated for the new node (in a similar way that is done while creating a list) before reading the data. The new node will contain empty data field and empty link field. The data field of the new node is then stored with the information read from the user. The link field of the new node is assigned to NULL. The new node can then be inserted in the list at three different places namely,

Inserting as a first node in the list Inserting as a last node in the list

Inserting an intermediate node in the list.

Inserting as a first node in the list

The following steps are followed to insert a new node in the start of the list.

Get the new node using GETNODE(), and read the details of the node using READNODE).

Check whether the list is empty or not (i.e., check whether the head pointer is pointing to NULL or not). If the list is empty, assign new node as head. If the list is not empty, follow the next steps.

The link field of the new node is made to

point the data field of the first node (ie., head

node) in the list by assigning the address of the first node.

The head pointer is made to point the data field of the new node by assigning the

address of the new node. In program 9.1, the function insert First (), is used for inserting a new node in

the first position of the list.

Algorithm for INSERT\_FIRST()

INSERT\_FIRST(HEAD: NODE)

GOR 300 CAGAN

NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE() STEP 2: CALL READNODE (NEWNODE) 300AHNEES

4

STEP 3: If (HEAD==NULL)

Set HEAD = NEWNODE Motow

Return

MWMGCC.KM d

[End of If structure]

STEP 4: Assign NEWNODE-> LINK=HEAD

STEP 5: Set HEAD = NEWNODE

END INSERT FIRST()

Inserting as a last node in the list

The following steps are followed to insert a new node in the end of the list. Get the new node using GETNODE(), and read the details of the node using READNODE().

Check whether the list is empty or not (l.e., check whether the head pointer is pointing to NULL or not). If the list is empty, assign new node as head. If the list is not empty.

follow the next steps.

The link field of the last node is made to point the data field of the new node in the list by assigning the address of the new

node. The link field of the new node is set to NULL.

In program 9.1, the function insert Last (), is used for inserting a new node in the last position of the list.

Algorithm for INSERT\_LAST()

INSERT LAST(HEAD: NODE)

LAST, NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE() STEP 2: CALL READNODE(NEWNODE)

STEP 3: If (HEAD == NULL)

Set HEAD NEWNODE

Return

[End of If structure]

STEP 4: Set LAST = HEAD STEP 5: Repeat While (LAST -> LINK!= NULL)

Assign LAST LAST -> LINK [End of While Structure] =

STEP 6: Assign LAST-> LINK= NEWNODE END INSERT LAST()

Inserting an intermediate node in the list

The following steps are followed, to insert a new node in an intermediate position in the list.

Get the new node using GETNODE(), and read the details of the node using

READNODE(). Check whether the list is empty or not (i.e., check whether the head pointer is pointing to NULL or not). If the list is empty, assign new node as head. If the list is not empty. follow the next steps.

Get the address of the preceding node after which the new node is to be inserted. The link field of the new node is made to point the data field of the next node (link field

of the preceding node) by assigning its address.

The link field of the preceding node is made to point the data field of the new node by

assigning the address of the new node.

In program 9.1, the function insertMiddle (), is used for inserting a new node in the intermediate position of the list.

Algorithm for INSERT MIDDLE()

INSERT\_MIDDLE (HEAD: NODE) LAST, NEWNODE: NODE;

CONDITION: DATA of the any one NODE in the LIST for Insert STEP 1: Set NEWNODE=GETNODE()

STEP 2: CALL READNODE(NEWNODE)

STEP 3: If (HEAD NULL) Set HEAD = NEWNODE KA

Return

(End of If structure]

ATAL C-TER

STEP 4: Print, "Enter the data of NODE after which the insertion is to be made"

STEP 5: READ, CONDITION STEP 6: Set LAST = HEAD

STEP 7: Repeat While (LAST!= NULL) as

STEP 8: If (LAST-> DATA== CONDITION) then No Assign NEWNODE-> LINK=LAST -> LINK

Assign LAST-> LINK= NEWNODE

Return

Else

[End of If Structure]

STEP 9: [End of STEP 7 while structure]

Assign LAST LAST -> LINK

STEP 10: Print, "CONDITION IS NOT AVAILABLE"

END INSERT\_MIDDLE()

9.3.3 MODIFICATION OF A NODE

A node(s) can be modified in a list, for changing its information part. The following

steps are followed to modify a node in the list.

Check whether the list is empty or not (l.e.,

check whether the head pointer is pointing

to NULL or not). If the list is not empty, follow the next steps.

Search for the node to be modified.

banChange the Information part of the node.

In program 9.1, the function modifyNode (), is used for modifying an existing

in the list.

node

Algorithm for MODIFY\_NODE()

MODIFY\_NODE(HEAD: NODE) STEP 1: If (HEAD == NULL)

Return

[End of If structure]

STEP 2: Print, "Enter the data of NODE to be modified"

love onla

STEP 3: Read, CONDITION

STEP 4: Set LAST = HEAD STEP 5: Repeat While (LAST!= NULL)

COMAIR LA SEP

STEP 6 If (LAST-> DATA== CONDITION) then Read, LAST-> DATA

Return

Twist to br

Else

Assign LAST = LAST -> LINK MATION NA

[End of If Structure]

DATES TRAIN a se

STEP 7:

[End of STEP 6 while structure] MAWR ne

STEP 8: Print, "CONDITION IS NOT AVAILABLE"

END MODIFY\_NODE()

9.3.4

DELETION OF A NODE

Another primitive operation that can be done in a singly linked list is the deletion of a node. Memory is to be released for the node to be deleted. A node can be deleted from the list from three different places namely, 3

Deleting the first node from the list Deleting the last node from the list

an intermediate node from the list.

Deleting the first node from the list

The following steps are followed, to delete a node from the start of the list.

Check whether the list is empty or not (l.e.,

check whether the head pointer is pointing to NULL or not). If the list is not empty, follow the next steps.

Set the head pointer to the second node in the list (by assigning its address). Release the memory for the deleted node.

the list. After deleting the node release the memory occupied by the deleted node by using In program 9.1, the function deleteFirst (), is used for deleting the first node in the function releaseNode().

Algorithm for RELEASE\_NODE()

RELEASE\_NODE(NEWNODE: NODE) STEP 1: Deallocate the space for the NODE of NEWNODE

ON CARA\_1733

STEP 1: Return

End RELEASENODE()

Algorithm for DELETE\_FIRSTO

DELETE FIRST (HEAD: NODE)

ut to be

DELNODE: NODE STEP 1: If (HEAD == NULL)

Print, "List is Empty"

Return

[End of If structure]

STEP 2: Set DELNODE = HEAD

STEP 3: Assign HEAD = HEAD-> LINK STEP 4: Print, "Deleted Data is",

DELNODE-> DATA=72A28

STEP 5: CALL RELEASENODE (DELNODE)

END DELETE\_FIRST()

Deleting the last node from the list

MAL-12AJ

TEA

The following steps are followed, to delete a node from the end of the list. Check whether the list is empty or not (i.e., check whether the head pointer is pointing to NULL or not). If the list is not empty, follow

the next steps.

The link field of the previous node (from the end of the list) is set to NULL. Release the memory for the deleted node.

In program 9.1, the function deleteLast (), is used for deleting the last node in the list. After deleting the node release the memory occupied by the deleted node by using the function releaseNode().

Algorithm for DELETE\_LASTO)

DELETE\_LAST(HEAD: NODE) HOWEVE

PREV, DELNODE: NODE

STEP 1: If (HEAD == NULL)

030032ALERA

Print, "List is empty"

Return

[End of If structure]

(ЗДОИ ПАН) таянаталь

STEP 2: If (HEAD-> LINK== NULL) Then

Set DELNODE= HEAD

Set HEAD= NULL

Print, "Deleted Data is", DELNODE-> DATA

Return [End of If structure]

STEP 3: Set LAST = HEAD

STEP 4: Repeat While (LAST -> LINK!= NULL) 792

STEP 5: Assign PREV = LAST

sdr most when

STEP 6: Assign LAST = LAST -> LINK

STEP 7: [End of STEP 4 While loop) 1 hora ante ga

STEP 8: Set DELNODE = LAST

STEP 9: PREV-> LINK= NULL

STEP 10: Print, "Deleted Data is", DELNODE-> DATA STEP 11: CALL RELEASENODE (DELNODE)

END DELETE\_LAST() 20

Deleting an intermediate node from the list

The following steps are followed, to delete a node from an intermediate position in the list. Check whether the liet is empty or not (ie, check whether the head pointer is pointing to NULL or not). If the list is not empty, follow the next steps.

The link field of the previous node (following the node to be deleted) is made to point the data field of the next node (before the node to be deleted), by assigning its address.

Release the memory for the deleted node.

Algorithm for DELETE MIDDLE()

DELETE MIDDLE (HEAD: NODE) LAST,

PREV, DELNODE NODE

DELDATA Data of the NODE is the NODE to Delete STEP 1: If (HEAD == NULL)

Print, "List is Empty"

Return

[End of If Structure]

Step 2: Pint, "Enter the DATA of the node in the list for deletion"

STEP 3: Read, DELDATA STEP 4: If (HEAD> DATA DELDATA)

Set DELNODE - HEAD

Then

Assign HEAD HEAD. LINK Print, "Deleted Data is", DELNODE-> DATA

CALL RELEASENODE (DELNODE)

Return

(End of If Structure]

STEP 5: Set LAST - HEAD-> LINK

STEP 6: Set PREV- HEAD STEP 7: Repeat While (LAST!= NULL)

STEP 8: I (LAST DATA DELDATA) Then Set DELNODE - LAST

Assign PREV LINK LAST LINK

Print, "The Deleted Data is", DELNODE-> DATA CALL RELEASENODE (DELNODE)

Return

Assign LAST LAST LINK

Assign PREV PREV LINK

[End of If structure)

STEP 9 End of STEP 7 While Loop) STEP 10: Print, "DELDATA is Not Available in the List"

END DELETE MIDDLE()

program 9.1, the function deleteMiddle (), is used for deleting the intermed node in the list. After deleting the node release the memory occupied by the deleted node by using the function releaseNode().

TRAVERSAL OF A LIST

9.3.5

To read the information or to display the information in a linked list, you have to

traverse (move) a linked list, node by node from the first node, until the end of the list is reached. Traversing a list involves the following steps,

Assign the address of head pointer to a variable.

Display the information in the data field.

Traverse the list from one node to another by advancing the pointer.

In program 9.1, the function view (), is used

for traversing and to display the information stored in the list.

Algorithm for VIEWO

VIEW(HEAD: NODE)

LIST: NODE

STEP 1: LIST = HEAD

STEP 2: If (LIST == NULL) Print, "List is Empty"

Return

[End of If Structure]

STEP 3: While (LIST!= NULL)

STEP 4: Print "The data is ", LIST-> DATA

STEP 5: LIST LIST -> LINK

STEP 6: [End of STEP 3 While structure]

END VIEWO

PROGRAM 9.1

Program to create, insert, delete, modify and view in a singly linked list."/

/all.c / #include<stdio.h>

struct list

int roll,

char name [20]; struct list link;

)node; getNode();

node

void createList (node \*\*headptr);

void insertFirst (node \*headptr); void insert Last (node \*\*headptr);

void insertMiddle (node \*\*headptr);

void deleteFirst (node \*\*headptr);

void deleteLast (node \*\*headptr);

void deleteMiddle (node \*\*headptr); void modifyNode (node \*head);

void view (node \*head); void releaseNode (node \*newnode);

void displayMenu(); void main()

node head NULL; int ch;

displayMenu(); while (1)

printf("\n\n ?");

fflush(stdin);

scanf("%d", &ch); switch(ch)

(

case 0 :

createList (&head);

break;

case 1 :

insertFirst (&head); break;

case 2:

insert Last (&head); break;

insertMiddle (&head);

break;

case 3:

case 4 :

deleteFirst (&head);

break;

case 5:

deleteLast (&head); break;

case 6:

deleteMiddle (&head);

break;

modifyNode (head); break;

case 8: view (head);

break;

case 9:

displayMenu(); break;

default:

case 7 :

E

printf("End of run of your program exit (0);

void displayMenu() printf("\nBasic Operations in a Singly Linked

printf("\n\n\t 0. Create List ");

printf("\n\t 1. Insert First "); printf("\n\t 2. Insert Last "); printf("\n\t 3. Insert Middle ");

printf("\n\t 4. Delete First ");

printf("\n\t 5. Delete Last "); printf("\n\t 6. Delete Middle ");

printf("\n\t 7. Modify "); printf("\n\t 8. View ");

printf("\n\t 9. Show Menu "); printf("\n\t 10. Exit ");

node getNode()

int size;

node newnode; size sizeof (node);

newnode (node) malloc(size);

return (newnode);

void readNode (node newnode)

(

printf("\nEnter the Roll Number: "); scanf("%d", &newnode->roll);

printf("Enter the Name: "); scanf("s",

newnode->name);

newnode->link = NULL;

void releaseNode (node newnode)

free (newnode),

void createList (node \*\*headptr)

node head NULL, newnode, last;

do

char ch;

newnode getNode();

readNode (newnode); if (head NULL)

head newnode; last head;

else

last link newnode; last last -> link;

fflush (stdin);

printf("Do u wish to Add Data in the List (y/n) ? "); scanf ("c", &ch);

while (ch y') || (ch Y')); \*headptr head;

void insertFirst (node \*\*headptr)

node head, newnode;

head \*headptr;

newnode - getNode();

readNode (newnode);

if (head NULL)

head- newnode;

else

newnode> link= head; head newnode;

\*headptr head,

void insertLast (node headptr)

node head, last, newnode/

head headptr newnode getNode();

readNode (newnode);

if (head= NULL) head newnode;

else (

last head;

while (last -> link != NULL)

last last link; last link = newnode;

\*headptr= head;

} void insertMiddle (node \*\*headptr)

node \*head, last, \*newnode;

int insdata;

head headptr; if (head= NULL)

printf("Singly Linked list is Empty.

"So new node is head node.");

newnode getNode();

readNode (newnode);

head newnode;

else

2

printf("\n Enter the node roll no after which the" insertion is to be made: "); tenni blow

scanf("%d", &insdata); last head;

while (last != NULL)

if (lastroll==insdata)

newnode getNode();

readNode (newnode); newnode-> link last -> link; Labamen

last link = newnode;

return;

bomusa best

六

}

else last last -> link;

printf("Insert Node is Not Found");

\*headptr= head;

}

void deletePirat (node headptr) node head, delnode;

head headptr

if (head NULL)

printf("\n singly Linked List is Empty"))

return;

delnode head;

head head link: printf("\n Deleted Data is... \n");

printf("\n Roll Number 1d", delnode-> roll);

printf("\n Name: a", delnode->name); releaseNode (delnode);

headptr head;

void deleteLast (node headptr)

node head, delnode, laat, "prev head headptr;

if (head NULL)

printf("\n Singly Linked List is Empty");

return;

else if (head link NULL)

delnode head; head NULL:

else

last = head;

while (last link = NULL) (

}

prev last; last last link:

delnode laat; prev link NULL; ->

printf("\n Deleted Data in \n"); printf("\n Roll Number: d", delnode-> roll),

printf("\n Name: ta", delnode->name);

releaseNode (delnode); headptr head;

void deleteMiddle (node headptr) (

node head, delnode, last, prev; int deldata;

head headptr; if (head NULL)

printf("\n Singly Linked List is Empty ...");

return;

} printf("\n Enter the node roll no to be deleted: "); scanf("%d", &deldata); if (head roll == deldata)

delnode head;

head head -> link;

printf("\n Deleted Data is.. \n"); printf("\n Roll Number: %d", delnode-> roll); printf("\n Name: %s", delnode->name);

releaseNode (delnode);

\*headptr= head;

return;

last head -> link;

prev head; while (last != NULL)

if (last roll == deldata)

( delnode last;

prev link last> link;

printf("\n Deleted Data is \n");

printf("\n Roll Number: %d", delnode-> roll);

printf("\n Name: ts", delnode -> name); releaseNode (delnode);

return;

else

last last -> link;

prev = prev -> link;

printf("Delete Node is Not Found"); \*headptr= head;

void modifyNode (node \*head)

int moddata;

24

}

if (head= NULL)

printf("\n Singly Linked List is Empty ...");

return;

} printf("\n Enter the node roll no for modification : "); scanf("%d", &moddata); while (head != NULL)

if (head-> roll == moddata)

the Node.

printf("\n Modify the Data of the

printf("\n Enter the New Roll Number: "); scanf("%d", &head-> roll);

printf("\n Enter the New Name: "); scanf("%s", head->name);

return;

} else

202

head head-> link; al

\n");

}

printf("Modify Node is Not Found"); % of duty was

void view (node \*list)

if (list= NULL)

201 printf("\nSingly Linked List is Empty..."Do

return; } for(;list = NULL; list-list link)

{

printf("\n Roll Number: %d", list -> roll); printf("\n Name: %s", list -> name); }

αν

The program displays the following output

Basic Operations in a Singly Linked List

25

74

Singly Linked List is Empty ...

0. Create List 1. Insert First

2. Insert Last 3. Insert Middle

4. Delete First

5. Delete Last 6. Delete Middle

7. Modify 8. View

9. Show Menu 10. Exit

75 Singly Linked List is Empty 76

Singly Linked List is Empty

77

Singly Linked List is Empty

7 8 Singly Linked List is Empty

70

Enter the Roll Number: 101 Enter the Name : Madhu

Do u wish to Add Data in the List (y/n) ? y

Enter the Roll Number: 102

Enter the Name : Priya Do u wish to Add Data in the List (y/n) 7 Y

Enter the Roll Number: 103

Enter the Name : Raaji Do u wish to Add Data in the List (y/n) 7 n

7 8

Roll Number: 101

Name: Madhu Roll Number: 102

Name: Priya

Roll Number: 103 Name: Raaji

71

Enter the Roll Number: 100 Enter the Name : Sudha

72 Enter the Roll Number: 104 Enter the Name: Siva

78

Roll Number: 100

Name: Sudha Roll Number: 101

Name: Madhu

Roll Number : 102

Name: Priya

Roll Number: 103

Name: Raaji

Roll Number: 104 Name: Siva

74

Deleted Data is ...

Roll Number: 100

Name: Sudha

75

Deleted Data is Roll Number: 104

Name: Siva

78

Roll Number: 101

Name: Madhu

Roll Number: 102

Name: Priya Roll Number: 103

Name: Raaji

23

Enter the node roll no after which the insertion is to be made: 104 Insert Node is Not Found 73

Enter the node roll no after which the insertion is to be made 102

Enter the Roll Number: 105 Enter the Name : Ramya

78

Roll Number: 101 Name: Madhu

Roll Number: 102

Name: Priya Roll Number: 105

Name: Ramya

Roll Number: 103

Name: Raaji

76

Enter the node roll no to be deleted: 104 Delete Node is Not Found

76

Enter the node roll no to be deleted: 105 Deleted Data is ...

Roll Number: 105

Name: Ramya

? 8

Roll Number: 101 Name: Madhu

Roll Number: 102

Name: Priya

Roll Number: 103 Name: Raaji

? 7

Enter the node roll no for modification : 105 Modify Node is Not Found

? 7 Enter the node roll no for modification: 102

Modify the Data of the Node ... Enter the New Roll Number: 112 Enter the New Name: Geetha

7 8 Roll Number: 101

Name: Madhu

Roll Number: 112 Name: Geetha

Roll Number: 103

Name: Raaji

79

Basic Operations in a Singly Linked List

0. Create List 1. Insert First

2. Insert Last 3. Insert Middle

4. Delete First

5. Delete Last 6. Delete Middle

7. Modify

8. View

9. Show Menu 10. Exit

302

74

Deleted Data is Roll Number: 101

Name: Madhu

? 5

Deleted Data is

Roll Number: 103

Name: Raaji

74

Deleted Data is ...

28

Roll Number: 102

Name: Geetha

78

Singly Linked List is Empty

73 Singly Linked list is Empty. So new node is head node.

Enter the Roll Number: 1000 Enter the Name : Vanitha

? 8

Roll Number: 1000

Name: Vanitha

? 10

End of run of your program

9.4 BASIC OPERATIONS IN A DOUBLY LINKED LIST

The basic operations that can be performed on doubly linked lists are,

Creation of a list Insertion of a node

Modification of a node

Deletion of a node

Slat Nome

Traversal of a list.

9.4.1 CREATION OF A LIST

The linked list in program 9.2, has two items (i.e., roll and name). Creation of list involves three processes. They are,

Creating a node

Reading details for a node from user

Connect the node with the list.

Creating a doubly linked list starts with creating a node. Sufficient memory has to be allocated for creating a node. The information is stored in the memory, allocated by using the malloc() function of type node. In program 9.2, the function getNode(), is used for

creating a node.

Algorithm for declaration of the structure NODE

Struct NODE

DATA: Data Field FLINK: Link Field (Address of next Struct NODE)

BLINK: Link Field (Address of previous Struct NODE)

End Struct

Algorithm for GETNODE()

GETNODE()

SIZE: INTEGER

NEWNODE: NODE

Mallar

\*

5MM

otherwise Return.

9

STEP 1: Set SIZE= get the size of the NODE

STEP 2: Set NEWNODE = Allocate space in memory for the size of SIZE and

return the initial address STEP 3: Return NEWNODE

165

/ NN

End GETNODE()

After allocating memory for the structure of type node, the information for the items (i.e., roll and name) has to be read from the user. In program 9.2, the function readNode(),

is used for reading details for the node from the user.

Algorithm for READNODE()

READNODE(NEWNODE: NODE) STEP 1: Read, NEWNODE-> DATA

STEP 2: Set NEWNODE-> FLINK = NULL

STEP 3: Set NEWNODE-> BLINK= NULL

STEP 4: Retur End READNODE()

Connect the new node with the existing list. If the list is empty, set the head pointer of

the list to the new node, other wise connect the new node in the last position of the list. In program 9.2, the function createList (), is used to connect the new nodes with the hist

Algorithm for CREATELIST()

CREATELIST()

HEAD, LAST, NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE()

STEP 2: CALL READNODE(NEWNODE)

STEP 3: Set HEAD = NEWNODE

STEP 4: Set LAST = NEWNODE STEP 5: If you want to add another NODE proceed

STEP 6: Set NEWNODE = GETNODE()

STEP 7: CALL READNODE(NEWNODE) STEP 8: Assign LAST-> FLINK= NEWNODE

STEP 9: Assign NEWNODE-> BLINK = LAST STEP 10: Assign LAST LAST-> FLINK

STEP 11: Goto STEP 5 END CREATELIST()

INSERTION OF A NODE

One of the most primitive operations that can be done in a doubly linked list is the

insertion of a node. Memory is to be allocated for the new node (in a similar way that is done while creating a list) before reading the data. The new node will contain empty data field and empty forward and backward link fields. The data field of the new node is then stored with the information read from the user. Both the link fields of the new node are assigned to NULL. The new node can then be inserted in the list at three different places namely,

Inserting as a first node in the list Inserting as a last node in the list

Inserting an Intermediate node in the list.

Inserting as a first node in the list

The following steps are followed to insert

a new node in the start of the list. Get the new node using GETNODE(), and read the details of the node using READNODE().

Check whether the list is empty or not (i.e., check whether the head pointer is pointing to NULL or not). If the list is empty, assign new node as head. If the list is not empty. follow the next steps.

The forward link field of the new node is made to point the first node (head node) in the list by assigning the address of the first node.

The backward link field of the first node (head node) is made to point the new node, by

assigning the address of the new node. Assign the new node as the head pointer.

In program 9.2, the function insertFirst(), is used for inserting a new node in

the first position of the list.

Algorithm for INSERT\_FIRST()

INSERT\_FIRST(HEAD: NODE) NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE()

STEP 2: CALL READNODE (NEWNODE)

STEP 3: If (HEAD == NULL)

Set HEAD = NEWNODE

NN

Return

[End of If structure]

STEP 4: Assign NEWNODE-> FLINK = HEAD

STEP 5: Assign HEAD-> BLINK= NEWNODE STEP 6: Assign HEAD = NEWNODE

END INSERT FIRST()

Inserting as a last node in the list

The following steps are followed to insert a new node at the end of the list. Get the new node using GETNODE(), and read the details of the node using READNODE()

Check whether the list is empty or not (l.e., check whether the head pointer is pointing

to NULL or not). If the list is empty, assign

new node as a head node. If the list is not

empty, follow the next steps.

The forward link field of the last node in the list is made to point the new node, by assigning the address of the new node.

The backward link field of the new node is made to point the last node, by assigning

the address of the last node.

The forward link field of the new node is set to NULL. In program 9.2, the function insert Last (), is used for inserting a new node in the

last position of the list.

Algorithm for INSERT\_LAST()

INSERT\_LAST(HEAD: NODE) LAST, NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE() STEP 2: CALL READNODE(NEWNODE)

STEP 3: If (HEAD== NULL)

Set HEAD NEWNODE

Return

[End of If structure]

STEP 4: STEP 5:

Set LAST = HEAD Repeat While (LAST -> FLINK!= NULL}

Assign LAST LAST -> FLINK

[End of While Structure] STEP 6: Assign LAST -> FLINK= NEWNODE

STEP 7: Assign NEWNODE-> BLINK = LAST

END INSERT\_LASTO

Inserting an intermediate node in the list

The following steps are followed, to insert a new node in an intermediate position

in Get the new node using GETNODE(), and read the details of the node using READNODE().

the list.

Check whether the list is empty or not (l.e., check whether the head pointer is pointing

to NULL or not). If the list is empty, assign new node as a head node. If the list is not empty, follow the next steps.

Get the address of the preceding node after which the new node is to be inserted. The forward link field of the new node is made to point the next node (forward link field

of the preceding node) by assigning its address.

The backward link field of the new node is made to point the preceding node by assigning the address of the preceding node. The backward link field of the next node (node before which the new node is to be

inserted) is made to point the new node,

by assigning the address of the new node. The forward link field of the preceding node (node after which the new node is to be inserted) is made to point the new node, by assigning the address of the new node.

In program 9.2, the function insertMiddle(), is used for inserting a new node in the intermediate position of the list.

Algorithm for INSERT\_MIDDLE()

INSERT MIDDLE (HEAD: NODE) LAST, NEXT, NEWNODE: NODE

CONDITION: DATA of the any one NODE in the LIST for insert STEP 1: Set NEWNODE=GETNODE()

STEP 2: CALL READNODE(NEWNODE)

STEP 3: If (HEAD == NULL)

Set HEAD NEWNODE

Return [End of If structure]

STEP 4: Print, "Enter the data of NODE after which the Insertion is to be made"

STEP 5: READ, CONDITION STEP 6: Set LAST = HEAD

STEP 7: Repeat While (LAST!= NULL)

STEP 8 If (LAST-> DATA== CONDITION) then Assign NEXT LAST -> FLINK

Assign NEWNODE-> FLINK NEXT

Assign NEWNODE-> BLINK = LAST

Assign LAST-> FLINK = NEWNODE

If (NEXT!= NULL)

Assign NEXT-> BLINK = NEWNODE

[End of If Structure]

Return

Else

NNNX N/

IN PRIN

Assign LAST LAST -> FLINK

[End of If Structure]

STEP 9: [End of STEP 7 while structure] IS

NOT AVAILABLE"

STEP 10: Print, "CONDITION

STEP 11: Retum

END INSERT MIDDLE()

MODIFICATION OF A NODE

9.4.3 A node(s) can be modified in a list, for changing its information part. The following

steps are followed to modify a node in the list.

Check whether the list is empty or not (l.e., check whether the head pointer is pointing to NULL or not). If the list is not empty,

follow the next steps.

Search for the node to be modified. Change the information part of the node.

In program 9.2, the function modifyNode(), is used for modifying an existing node

in the list.

Algorithm for MODIFY \_NODEO)

MODIFY\_NODE(HEAD: NODE)

STEP 1: If (HEAD == NULL)

Return

[End of If structure]

STEP 2: Print, "Enter the data of NODE to

be modified"

STEP 3: Read, CONDITION

STEP 4: Set LAST = HEAD

STEP 5: Repeat While (LAST != NULL)

STEP 6: If (LAST-> DATA== CONDITION) then M

Read, LAST-> DATA

Return

Assign LAST = LAST -> FLINK

Else

[End of If Structure]

STEP 7: [End of STEP 5, While structure]

STEP 8: Print, "CONDITION IS NOT AVAILABLE"

9.3.4

END MODIFY\_NODE() DELETION OF A NODE

ON: CACI Teur utaso

Another primitive operation that can be done in a doubly linked list is the deletion of a bode. Memory is to be released for the node to be deleted. A node can be deleted from the list from three different places namely,

Deleting the first node from the list

Deleting the last node from the list

Deleting an intermediate node from the list.

Deleting the first node from the list

The following steps are followed, to delete a node from the start of the list.

Check whether the list is empty or not (ie., check whether the head pointer is pointing to NULL or not). If the list is not empty, follow the next steps.

Set the head pointer to the second node in the

list (by assigning its address)

Set the backward link field of the head node in the list to NULL.

Release the memory for the deleted node.

In program 9.2, the function deleteFirst (), is used for deleting the first node in the list. After deleting the node release the memory occupied by the deleted node by using

the function releaseNode().

Algorithm for RELEASE\_NODEO

RELEASENODE(NEWNODE: NODE)

STEP 1: Deallocate the space for the NODE of NEWNODE

STEP 2: Return

End RELEASENODE() Algorithm for DELETE\_FIRSTO

DELETE FIRST (HEAD: NODE) DELNODE: NODE

STEP 1: If (HEAD == NULL)

Print, "List is Empty"

Return [End of If structure]

STEP 2: Set DELNODE= HEAD

STEP 3:

Assign HEAD= HEAD-> FLINK

STEP 4: If (HEAD!= NULL)

Assign HEAD-> BLINK= NULL [End of If

Structure]

STEP 5: Print, "Deleted Data is", DELNODE-> STEP 6: CALL RELEASENODE(DELNODE)

DATA

STEP 7: Return END DELETE\_FIRST()

Deleting the last node from the list

The following steps are followed, to delete a node from the end of the list.

Check whether the list is empty or not (ie., check whether the head pointer is

pointing to NULL or not). If the list is not empty, follow the next steps. The forward link field of the previous node (from the

end of the list) is set to NULL

Release the memory for the deleted node.

In program 9.2, the function deleteLast (), is used for deleting the last node in the list. After deleting the node release the memory occupied by the deleted node by using the

function releaseNode().

Algorithm for DELETE\_LASTO

DELETE\_LAST(HEAD: NODE)

LAST, PREV, DELNODE: NODE STEP 1: If (HEAD == NULL)

Print, "List is empty"

Return

[End of If structure]

STEP 2: If (HEAD-> FLINK== NULL) Then

Set DELNODE=HEAD Print, "Deleted Data is", DELNODE-> DATA

Return

[End of If structure]

STEP 3: Set LAST =

HEAD

STEP 4: Repeat While (LAST -> FLINK!= NULL) Assign LAST = LAST -> FLINK

[End of while loop]

STEP 5:

Set DELNODE = LAST

STEP 6: Assign LAST-> BLINK-> FLINK= NULL

STEP 7: Print, "Deleted Data is", DELNODE-> DATA STEP 8: CALL RELEASENODE (DELNODE)

STEP 9: Return END DELETE\_LAST()

Deleting an intermediate node from the list

The following steps are followed, to delete a node from an intermediate position in the

list.

Check whether the list is empty or not (i.e.,

check whether the head pointer is pointing to NULL or not). If the list is not empty, follow the next steps.

The forward link field of the previous node (following the node to be deleted) to point the next node (before the node to be deleted), by assigning its address.

is made

The backward link field of the next node (before the node to be deleted) is made to

point the previous node (following the node to be deleted), by assigning its address.

Release the memory for the deleted node.

In program 9.2, the function deleteMiddle

(), is used for deleting the intermediate node in the list. After deleting the node release the memory occupied by the deleted node by

using the function releaseNode(). Algorithm for DELETE\_MIDDLE()

DELETE\_MIDDLE(HEAD: NODE)

NEXT, PREV, LAST, DELNODE: NODE DELDATA: Data of the NODE is the List to Delete the b

STEP 1: If (HEAD == NULL) Print, "List is Empty"

Return [End of If Structure]

STEP 2: Print, "Enter the DATA of the any node in the List for Delete"

STEP 3: Read, DELDATA

STEP 4: If

(HEAD-> DATA== DELDATA) Then

Set DELNODE = HEAD Assign HEAD HEAD -> FLINK

If (HEAD!= NULL) Assign HEAD-> BLINK = NULL

[End of If Structure]

Print, "Deleted Data is", DELNODE-> DATA

CALL RELEASENODE (DELNODE) Return

[End of If Structure]

STEP 5: Set LAST = HEAD ->

FLINK STEP 6: Repeat While (LAST!= NULL)

STEP 7: If (LAST-> DATA- == DELDATA) Then Set DELNODE = LAST

Set PREV = LAST -> BLINK Set NEXT LAST -> FLINK

Assign PREV-> FLINK = NEXT

If (NEXT != NULL) Assign NEXT-> BLINK = PREV

[End of If Structure]

Print, "The Deleted Data is ", DELNODE-> DATA

CALL RELEASENODE (DELNODE)

Return

Assign LAST LAST -> FLINK

Else

[End of If structure]

STEP 8: [End of STEP 6 While Loop] STEP 9: Print, "DELDATA is Not Available in the List",

STEP 10: Return

END DELETE\_MIDDLE()

9.4.5 TRAVERSAL OF A LIST

To read the information or to display the information in a linked list, you have to traverse (move) a linked list, node by node

from the first node, until the end of the list is

reached. Traversing a list involves the following steps, Assign the address of head pointer to a variable.

Display the information in the data field. Traverse the list from one node to another by advancing the pointer.

In program 9.2, the function view (), is used for traversing and to display the information in the list.

Algorithm for VIEW()

VIEW(HEAD: NODE)

LIST: NODE

STEP 1: LIST= HEAD

STEP 2: If (LIST== NULL)

Print, "List is Empty"

Return

[End of If Structure]

STEP 3: While (LIST!= NULL)

STEP 4: Print "The data is ", LIST-> DATA STEP 5: LIST LIST -> FLINK =

STEP 6: [End of STEP 3 While structure]

ENDVIEW()

PROGRAM 9.2

/\* Program to create, insert, delete, modify and view in a doubly linked list. "/

/\* dll.c \*/

#include <stdio.h>

typedef struct list

( int roll;

char name [20];

struct list flink; struct list \*blink;

)node;

node getNode();

void createList (node \*\*headptr); void insertFirst (node \*\*headptr);

void insertLast (node \*\*headptr);

void insertMiddle (node \*\*headptr); void deleteFirst (node \*\*headptr);

void deleteLast (node \*headptr);

void deleteMiddle (node \*\*headptr);

void modifyNode (node \*head);

void view (node \*head); void releaseNode (node \*newnode);

void main()

void displayMenu();

int ch;

displayMenu();

while (1)

printf("\n\n?");

fflush(stdin);

scanf("%d", &ch),

switch(ch)

case 0:

createList (&head); break;

case 1:

case 2:

insertFirst (&head); break;

insertLast (&head); break;

case 3:

insertMiddle (&head); break;

case 4:

deleteFirst (&head); break;

case 5 :

deleteLast (&head); break;

deleteMiddle (&head);

break;

case 6 :

case 7 :

case 8 :

modifyNode (head); break;

view (head); break;

case 9 :

displayMenu(); break;

default:

");

printf("End of run of your program ...

exit(0);

void displayMenu()

printf("\nBasic Operations in a Doubly Linked List printf("\n\n\t 0. Create List ")

"); printf("\n\t 1. Insert First "); printf("\n\t 2. Insert Last "); printf("\n\t 3. Insert Middle ");

printf("\n\t 4. Delete First "); printf("\n\t 5. Delete Last ");

printf("\n\t 6. Delete Middle "), printf("\n\t 7. Modify ");

printf("\n\t 8. View "); printf("\n\t 9. Show Menu "), printf("\n\t 10. Exit ");

node\* getNode() (

int size: node newnode;

size sizeof (node),

newnode (node)malloc(size); return (newnode);

void readNode (node\* newnode)

printf("\nEnter the Roll Number: ");

scanf("%d", &newnode->roll); printf("Enter the Name: "); scanf("%a", newnode->name); newnode->flink= NULL;

newnode->blink = NULL; void releaseNode (node \*newnode)

free (newnode);

void createList (node \*\*headptr) (

node head NULL, newnode, last; char ch;

do

newnode getNode();

readNode (newnode); if (head NULL)

head newnode;

last head;

else (

newnode blink- last,

last flink = newnode; last last -> flink;

fflush(stdin); printf("Do u wish to Add Data

in the List

scanf ("c", &ch); while (chy') || (ch 'Y')); headptr head,

void insertFirst (node \*headptr)

node head, newnode; head \*headptr;

newnode getNode();

readNode (newnode);

if (head == NULL) head newnode;

else

(

newnode-> flink= head; head blink - newnode;

head newnode;

\*headptr= head;

void insertLast (node headptr)

node newnode, head, last; head headptr;

newnode getNode();

readNode (newnode); if (head NULL)

head newnode;

else

last head;

while (last -> flink = NULL) last last -> flink;

newnode-> blink last; last flink = newnode;

\*headptr= head;

void insertMiddle (node \*\*headptr)

node \*newnode, head, last, \*next;

int insdata; head headptr;

if (head= NULL)

printf("The Doubly Linked list is Empty. "So new node is head node.");

newnode getNode();

readNode (newnode);

head newnode;

} else

( printf("\n Enter the node roll no after which the"

insertion is to be made: "); scanf("%d", &insdata);

last head, while (last = NULL)

if (last rollinsdata)

newnode getNode(); readNode (newnode);

next last -> flink; newnode flink next;

newnode blink last, last flink = newnode; next blink newnode;

if (next = NULL)

return;

else

last last -> flink;

printf("Insert Condition is Not Found");

headptr head;

void deleteFirst (node headptr)

node head, delnode;

head -headptr; if (head NULL)

(

printf("\n Doubly Linked List is Empty...");

return;

delnode head;

head head -> flink;

if (head - NULL) head blink = NULL;

printf("\n Deleted Data is ...\n");

printf("\n Roll Number: %d", delnode-> roll);

printf("\n Names", delnode->name); releaseNode (delnode); headptr= head;

void deleteLast (node headptr)

node head, delnode, last, prev;

head headptr; if (head NULL)

printf("\n Doubly Linked List is Empty...");

return,

else if (head flink NULL)

delnode head, head NULL,

else

last head,

while (last flink = NULL)

last last flink,

prev last> blink, delnode last,

prev flink= NULL,

printf("\n Deleted Data is ... \n"); printf("\n Roll Number: d", delnode-> roll);

printf("\n Name: a", delnode->name); releaseNode (delnode); headptr head;

void deleteMiddle (node \*\*headptx)

node head, delnode, last, prev, next; int deldata

head headptr; if (head NULL)

( printf("\n Doubly Linked List is Empty.

return;

printf("\nEnter the node roll no for deleteion is to be made "); scanf("%d", &deldata), if (head roll deldata)

delnode head, head head -> flink;

if (head - NULL) head blink= NULL; ->

printf("\n Deleted Data is... \n"); printf("\n Roll Number: d", delnode-> roll); printf("\n Names", delnode name);

releaseNode (delnode);

headptr head, return,

last head flink,

while (last 1 NULL)

1

if (last roll del data)

delnode last;

prev last -> blink, next last -> flink;

prev flink next;

if (next = NULL)

next blink prev printf("\n Deleted Data is... \n");

printf("\n Roll Number: d", delnode-> roll);

printf("\n Name: ts", delnode->name); releaseNode (delnode);

return;

else

last last flink;

) printf("Delete Node is Not Found") d

head;

void modifyNode (node \*head)

(

int moddata; if (head= NULL)

printf("\n Doubly Linked List is Empty");

return;

printf("\n Enter the node roll no for modification: ");

scanf("%d", &moddata); while (head = NULL)

if (head roll moddata)

printf("\n Modify the Data of the Node ...

\n");

printf("\n Enter the New Roll Number: "); scanf ("d", &head roll); ->

printf("\n Enter the New Name: "); scanf ("s", head name);

return;

else

head head -> flink

printf("Modify Node is Not Found");

void view (node list)

if (list NULL) (

printf("\n Doubly Linked List is Empty...");

return; for (list = NULL; list list->flink)

printf("\n Roll Number: d", list->roll); printf("\n Name: ta", list->name);

The program displays the following output Basic Operations in a Doubly Linked List

0. Create List 1. Insert First

2. Insert Last

3. Insert Middle

4. Delete First

5. Delete Last

6. Delete Middle

7. Modify 8. View

9. Show Menu 10. Exit

74

Doubly Linked List is Empty

? 5

Doubly Linked List is Empty ...

76

Doubly Linked List is Empty ...

77

Doubly Linked List is Empty

78

Doubly Linked List is Empty

70 Enter the Roll Number: 201

Enter the Name: Kannan Do u wish to Add Data in the List (y/n) ? y

Enter the Roll Number: 202

Enter the Name: Lalitha Do u wish to Add Data in the List (y/n)? y

Enter the Roll Number: 203 Enter the Name : Karthik Do u wish to Add Data in the List (y/n) ? n

7 8

Roll Number: 201

Name: Kannan

Roll Number: Name: Lalitha

Roll Number: 203

Name: Karthik

? 1

Enter the Roll Number: 200

Enter the Name : Kamal

? 2

Enter the Roll Number: 204

Enter the Name: Priya

7 8

Roll Number: 200

Name: Kamal Roll Number: 201

Name: Kannan

Roll Number: 202

Name: Lalitha

Roll Number: 203

Name: Karthik

Roll Number: 204

Name: Priya

74

Deleted Data is ...

Roll Number: 200

Name: Kamal

75 Deleted Data is ...

Roll Number: 204 Name: Priya

7 8

Roll Number: 201 Name: Kannan

Roll Number: 202

Name: Lalitha

Roll Number: 203 Name: Karthik

73

Enter the node roll no after which the insertion is

to be made

: 204

Insert Condition is Not Found

73

Enter the node roll no after which the insertion is to be made 202

Enter the Roll Number: 205

Enter the Name : Ravi

78

Roll Number: 201 Name: Kannan

Roll Number: 202

Name: Lalitha

Roll Number: 205

Name: Ravi

Roll Number: 203

Name: Karthik

76

Enter the node roll no for deleteion is to be made: 204 Delete Node is Not Found

76

Enter the node roll no for deleteion is to be made: 205

Deleted Data is ...

Roll Number: 205

Name: Ravi

? 8

Roll Number: 201

Name: Kannan

Roll Number: 202 Name: Lalitha

Roll Number: 203

Name: Karthik

? 7

Enter the node roll no for modification: 205 Modify Node is Not Found

? 7

Enter the node roll no for modification: 202 Modify the Data of the Node ...

Enter the New Roll Number: 212 Enter the New Name: Lalli

78

Roll Number: 201 Name Kannan

Roll Number: 212

Name Lalli

Roll Number: 203

Name: Karthik

74

Deleted Data is Roll Number: 201

Name: Kannan

75

Deleted Data is

Roll Number: 203 Name: Karthik

74

Deleted Data is....

Roll Number: 2121 Name: Lalli

74

Deleted Data is..

Roll Number: 212 Name: Lalli

7 8

Doubly Linked List is Empty

73 The Doubly Linked list is Empty. So new node is head node.

Enter the Roll Number: 2000 Enter the Name: Jeyabal

7 8

Roll Number: 2000

Name: Jeyabal

7 10

End of run of your program

BASIC OPERATIONS IN A CIRCULAR BINGLY LINKED LIST

The basic operations that can be performed on circular singly linked list are similar to the singly linked list, except that the last node is made to point the first node in the list. The basic operations that can be performed on circular singly linked lists are,

Creation of a list

Insertion of a node Modification of a node

Deletion of a node

Traversal of a list.

9.5.1 CREATION OF A LIST

The circular singly linked list in program 9.3, has two items (ie., roll and name). Creation of list involves three processes.

Creating a node

Reading details for a node from user Connect the node with the list.

Creating a circular singly linked list starts with creating a node. Sufficient memory has to be allocated for creating a node. The information is stored in the memory,

allocated by using the malloc() function of type node. In program 9.3, the function getNode(), is used for creating a node.

Algorithm for declaration of the structure NODE

Struct NODE

DATA: Data Field

LINK: Link Field (Address of next struct NODE)

End Struct Algorithm for GETNODE()

GETNODE()

SIZE: INTEGER; NEWNODE: NODE

STEP 1: Set SIZE = get the size of the NODE

STEP 2: Set NEWNODE- Allocate space in memory for the size of SIZE and

return the initial address STEP 3: Retum NEWNODE

End GETNODE()

After allocating memory for the structure of type node, the information for the items (ie., roll and name) has to be read from the user. In program 9.3, the function readNode(), is used for reading details for the node from the user.

Algorithm for READNODEO

READNODE(NEWNODE: NODE) STEP 1: Read, NEWNODE-> DATA

STEP 2: Set NEWNODE-> LINK = NEWNODE

STEP 3: Retur

End READNODE()

Connect the new node with the existing list. If the list is empty, set the head pointer of the list to the new node, other wise connect the new node in the last position of the list. la program 9.3, the function createList (), is used to connect the new nodes with the list

Algorithm for CREATELISTO

CREATELIST()

HEAD, LAST, NEWNODE: NODE STEP 1: Assign NEWNODE = GETNODE()

STEP 2: CALL READNODE(NEWNODE)

STEP 3: Set HEAD = NEWNODE

STEP 4: Set LAST = NEWNODE

STEP 5: If you want to add another NODE proceed otherwise Return HEAD. STEP 6: Set NEWNODE = GETNODE()

STEP 7: CALL READNODE(NEWNODE)

STEP 8: Assign LAST-> LINK= NEWNODE

STEP 9: Assign NEWNODE-> LINK = HEAD

STEP 10: Assign LAST LAST -> LINK

STEP 11: Goto STEP 5

END CREATELISTO

9.5.2

INSERTION OF A NODE

One of the most primitive operations that can be done in a singly linked list is the insertion of a node. Memory is to be allocated for the new node (in a similar way that is done while creating a list) before reading the data. The new node will contain empty data field and empty link field. The data field of the new node is then stored with the information read from the user. The link field of the new node is assigned to NULL. The new node can then be inserted in the list at three different places namely,

Inserting as a first node in the list

Inserting as a last node in the list Inserting as an intermediate node in the list.

Inserting as a first node in the list

The following steps are followed to insert a new node in the start of the list.

Get the new node using GETNODE(), and read the detalls of the node using READNODE() Check whether the list is empty or not (i.e., check whether the head pointer is pointing to NULL or not). If the list is empty, assign new node as head. If the list is not empty.

follow the next steps. The link field of the new node is made to point the data field of the first node (ie., head

node) in the list by assigning the address of the first node. The head pointer is made to point the data field of the new node by assigning the

address of the new node.

In program 9.3, the function insertFirst (), is used for inserting a new node in

the first position of the list.

Algorithm for INSERT\_FIRST()

INSERT FIRST(HEAD: NODE)

NEWNODE, LAST: NODE

STEP 1: Set NEWNODE = GETNODE()

STEP 2: CALL READNODE (NEWNODE)

STEP 3: If (HEAD==NULL)

Set HEAD NEWNODE

Return

[End of If structure]

STEP 4: Set LAST = HEAD

STEP 5: Repeat While (LAST-> LINK! HEAD) type ye Assign LAST LAST -> LINK

(End of While Structure]

STEP 6: Assign LAST -> LINK= NEWNODE

STEP 7: Assign NEWNODE-> LINK= HEAD STEP 8: Assign HEAD = NEWNODE

END INSERT FIRST()

Inserting as a last node in the list

The following steps are followed to insert a new node in the end of the list.

Get the new node using GETNODE(), and read the details of the node using READNODE().

Check whether the list is empty or not (l.e., check whether the head pointer is pointing to NULL or not). If the list is empty, assign new node as head. If the list is not empty, follow the next steps.

The link field of the last node is made to point the data field of the new node in the list by assigning the address of the new

node.

The link field of the new node is set to NULL.

In program 9.3, the function insertLast (), is used for inserting a new node in the

last position of the list.

Algorithm for INSERT\_LAST()

INSERT LAST(HEAD: NODE) LAST, NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE()

STEP 2: CALL READNODE(NEWNODE)

STEP 3: If (HEAD==NULL)

Set HEAD =

NEWNODE 1:09

aber won a Return of bourat Yo fub con rodin

[End of If structure] STEP 4: Set LAST = HEAD

STEP 5: Repeat While (LAST -> LINK!= HEAD)

Assign LAST LAST-> LINKCOM

[End of While Structure] STEP 6: Assign LAST -> LINK= NEWNODE

STEP 7: Assign NEWNODE-> LINK= HEAD

END INSERT LAST()

Inserting an intermediate node in the list

The following steps are followed, to insert a new node in an intermediate position in

the list.

Get the new node using GETNODE(), and read the details of the node using READNODE(). Check whether the list is empty or not (i.e., check whether the head pointer is pointing

to NULL or not). If the list is empty, assign new node as head. If the list is not empty, follow the next steps.

Get the address of the preceding node after which the new node is to be inserted.

The link field of the new node is made to point the data field of the next node (link field of the preceding node) by assigning its address.

The link field of the preceding node is made to point the data field of the new node by assigning the address of the new node. HL 12

In program 9.3, the function insertMiddle (), is used for inserting a new node in the intermediate position of the list.

Algorithm for INSERT\_MIDDLEO

INSERT\_MIDDLE (HEAD: NODE) AL TA

LAST, NEWNODE: NODE

CONDITION: DATA of the any one NODE in

the LIST for insert STEP 1: Set NEWNODE= GETNODE()

STRAL TRA

STEP 2: CALL READNODE(NEWNODE)

STEP 3: If (HEAD == NULL)

Set HEAD NEWNODE Return

[End of If structure]

STEP 4: Print, "Enter the data of NODE after which the insertion is to be made"

STEP 5: Read, CONDITION

STEP 6: Set LAST = HEAD

STEP 7: Repeat

STEP 8:

If (LAST-> DATA== CONDITION) then

Assign NEWNODE-> LINK=LAST-> LINK

Assign LAST-> LINK NEWNODE

Return

Else

Assign LAST = LAST -> LINK

(End of If Structure]

STEP 9: Until (LAST == HEAD)

STEP 10: Print, "CONDITION IS NOT AVAILABLE"

END INSERT MIDDLE()

MODIFICATION OF A NODE

steps A node(s) can be modified in a list, for changing its information part, The following are followed to modify a node in the list.

Check whether the list is empty or not (l.e., check whether the head pointer is pointing

to NULL or not). If the list is not empty, follow the next steps.

Search for the node to be modified.

Change the Information part of the node.

In program 9.3, the function modifyNode(), is used for modifying an existing node

in the list.

Algorithm for MODIFY\_NODE()

MODIFY\_NODE(HEAD: NODE) gefst

STEP 1: If (HEAD == NULL)

Return

[End of If structure]

STEP 2: Print, "Enter the data of NODE to be modified"

STEP 3: Read, CONDITION STEP 4: Set LAST = HEAD

STEP 5: Repeat

STEP 6: If (LAST-> DATA CONDITION) then Read, LAST-> DATA

Return

Else

Assign LAST LAST

[End of If Structure]

STEP 7: Until (LAST == HEAD)

STEP 8: Print, "CONDITION IS NOT AVAILABLE"

END MODIFY\_NODE()

9.5.4 DELETION OF ANODE

Another primitive operation that can be done in a singly linked list is the deletion of a node. Memory is to be released for the node to be deleted. A node can be deleted from the list from three different places namely,

Deleting the first node from the list. Deleting the last node from the list.

Deleting an intermediate node from the list.

Deleting the first node from the list

The steps are followed, to delete a node from the start of the list Check whether the list is emply or not (Le, check whether the head pointer

to NULL or not). If the list is not empty,

follow the next steps. Set the head pointer to the second node in the list (by te address) Release the memory for the deleted node

In program 9.3, the function deleterirat (), is used for deleting the first rods i the list. After deleting the node release the memory occupied by the deleted node by using

the function releaseNode().

Algorithm for RELEASE NODE()

RELEASE\_NODE(NEWNODE NODE)

STEP 1: Deallocate the space for the NODE of NEWNODE V STEP 2: Return

End RELEASENODE()

Algorithm for DELETE FIRST()

DELETE FIRST(HEAD: NODE) LAST, PREV, DELNODE NODE STEP 1: (HEAD == NULL)

Print, "List is empty" Return

(End of If structure)

STEP 2:

I (HEAD> LINK HEAD) Then DELNODE HEAD

Print, "Deleted Data is", DELNODE DATA

Set HEAD NULL CALL RELEASENODE(DELNODE)

Return

(End of If structure)

STEP 3: Sel LAST = HEAD STEP 4: Repent

Assign PREV LAST Assign LAST LAST-> LINK

Until (LAST LINK== HEAD) STEP 5: DELNODE- LAST

STEP 6: PREV-> LINK= HEAD

STEP 7: Print, "Deleted Data is", DELNODE-> DATA

STEP 8: CALL RELEASENODE (DELNODE) END DELETE PIRST()

Deleting the last node from the list

The following steps are followed, to delete a node from the end of the list.

Check whether the list is empty or not (ie., check whether the head pointer is pointing

to NULL or not). If the list is not empty, follow the next steps. The link field of the previous node (from the end of the list) is set to NULL.

Release the memory for the deleted node.

In program 9.3, the function deleteLast (), is used for deleting the last node in the list. After deleting the node release the memory occupied by the deleted node by using the function releaseNode().

Algorithm for DELETE LAST()

DELETE LAST(HEAD: NODE) LAST, PREV DELNODE: NODE

Step 1: (HEAD-- NULL)

Print, "List is empty"

Return

End of If structure]

STEP 2: (HEAD LINK HEAD) Then

Set DELNODE-HEAD

Set HEAD NULL

Pint, "Deleted Data is", DELNODE-> DATA CALL RELEASENODE(DELNODE)

-

Return

-

End of If structure)

Step 3: Set LAST-HEAD

STEP 4: Repeat While (LAST LINK!-HEAD)

STEP 5: Assign PREVLAST

STEP 6: Assign LAST LAST LINK

STEP 7 (End of STEP 4 While loop]

Ster 8: Set DELNODE-LAST

Ster 9: PREV-> LINK-HEAD

STEP 10: Print, "Deleted Data is", DELNODE> DATA

STEP 11: CALL RELEASENODE (DELNODE)

END DELETE LASTO

Deleting an intermediate node from the list

The following steps are followed, to delete a node from an intermediate position in the

list.

Check whether the list is empty or not (ie, check whether the head pointer is pointing

to NULL or not) If the list is not empty, follow the next steps.

The link field of the previous node (following the node to be deleted) is made to point the data field of the next node (before the node to be deleted), by assigning its address

Release the memory for the deleted node.

In program 9.3, the function deleteMiddle(), is used for deleting the intermediate node in the list. After deleting the node release the memory occupied by the deleted node by using the function releaseNode().

Algorithm for DELETE\_MIDDLE()

DELETE\_MIDDLE(HEAD: NODE) LAST,

PREV, DELNODE: NODE

DELDATA: Data of the NODE is the List to Delete

STEP 1: If (HEAD == NULL) Print, "List is Empty"

Return

[End of If Structure]

STEP 2: Print, "Enter the DATA of the any node in the List for Details"

STEP 3: Read, DELDATA

STEP 4: If (HEAD-> DATA== DELDATA) Then

DELNODE = HEAD

If (HEAD -> LINK== HEAD)

HEAD= NULL

Else

LAST = HEAD

WHILE(LAST-> LINK != HEAD) LAST LAST -> LINK

HEAD = HEAD -> LINK LAST -> LINK=HEAD

[End of If Structure]

Print, "Deleted Data Is", DELNODE-> DATA CALL RELEASENODE (DELNODE)

Return

[End of If Structure]

STEP 5:

Set LAST = HEAD -> LINK STEP 6: Set PREV = HEAD

STEP 7:

Assign PREV-> LINK LAST-> LINK Print, "The Deleted Data is ", DELNODE-> DATA

CALL RELEASENODE (DELNODE)

Return

Else

Assign LAST LAST -> LINK Assign PREV PREV-> LINK

[End of If structure]

STEP 9: [End of STEP 7 While Loop]

STEP 10: Print," DELDATA is Not Available in the List",

END DELETE\_MIDDLE()

9.5.5 TRAVERSAL OF A LIST

To read the information or to display the information in a linked list, you have to traverse (move) through a linked list, node by node from the first node, until the end of the list is reached. Traversing a list involves the following steps,

Assign the address of head pointer to a variable Display the information in the data field

Traverse the list from one node to another by advancing the pointer.

In program 9.3, the function view(), is used for traversing and to display the

information stored in the list.

Algorithm for VIEWO VIEW(HEAD: NODE)

LIST: NODE

STEP 1: LIST HEAD

STEP 2: If (LIST== NULL)

Print, "List is Empty"

Return [End of If Structure]

STEP 3: Repeat

STEP 4: Print "The data is ", LIST-> DATA

STEP 5: LIST LIST -> LINK

STEP 6: Until (LIST=-HEAD)

END VIEWO

PROGRAM 9.3

Program to create, insert, delete, modify and view in a circular singly linked list \*/

/cell.c/ #include <stdio.h>

typedef struct list

int roll;

char name [20]; struct list link:

node;

node getNode(); void createList (node headptr);

void insertFirst (node headptr); void insertLast (node \*headptr);

void insertMiddle (node \*\*headptr);

void deleteFirst (node headptr);

void deleteLast (node headptr); void deleteMiddle (node \*\*headptr);

void modifyNode (node head); void view (node head);

void releaseNode (node \*newnode);

void displayMenu();

void main()

node head NULL;

int ch; displayMenu();

while (1)

( printf("\n\n? ");

fflush(stdin);

scanf("%d", &ch); switch(ch)

case 0 : createList (&head);

break;

case 1 : insert First (&head);

break;

case 2:

insert Last (&head);

break;

case 3:

insertMiddle (&head); break;

case 4:

deleteFirst (head); break;

case 5:

deleteLast (&head); break;

case 6 1

deleteMiddle (&head);

break;

case 7 1

modifyNode (head);

break;

case 8:

view (head);

break;

case 9:

display\_menu(); break;

default:

printf("End of run of your program...");

exit(0);

void displayMenu()

printf("\nBasic Operations in a Circular Singly

Linked List..."); printf("\n\n\t 0. Create List ");

printf("\n\t 1. Insert First ");

printf("\n\t 2. Insert Last ");

printf("\n\t 3. Insert Middle ");

printf("\n\t 4. Delete First ");

printf("\n\t 5. Delete Last ");

printf("\n\t 6. Delete Middle ");

printf("\n\t 7. Modify "); printf("\n\t 8. View ");

printf("\n\t 9. Show Menu ");

printf("\n\t 10. Exit ");

node getNode()

int size; node newnode;

size sizeof (node); newnode (node) malloc(size);

return (newnode);

void readNode (node newnode)

printf("\nEnter the Roll Number:

");

scanf ("d", &newnode->roll), printf("Enter the Name: ");

scanf ("s", newnode->name); newnode->link newnode;

void releaseNode (node

free (newnode);

void createList (node

\*\*headptr)

node head NULL, newnode, last,

char ch;

do

newnode getNode(); readNode (newnode);

if (head NULL)

head newnode; last head;

else (

last link newnode; newnode-> link= head;

last last> link;

fflush (stdin); printf("Do u wish to Add Data in the List (y/n) ? ");

scanf("%c", &ch);

while((ch 'y') || (ch - 'Y')); \*headptr head;

void insertFirst (node \*headptr) (

node head, newnode, last;

head headptr;

newnode getNode(); readNode (newnode);

if (head NULL)

head newnode;

else

(

last head;

while (last link I head) last last -> link;

last link newnode;

newnode link head,

head newnode;

\*headptr -head,

void insertLast (node headptr)

node head, newnode, last; head headptr;

newnode getNode(); readNode (newnode);

if (head NULL)

head newnode;

else

last head; while (last link - head) last last -> link;

last link = newnode; newnode link= head;

\*headptr head;

void insertMiddle (node headptr) (

node head, newnode, last;

int insdata; head headptr;

if (head NULL)

printf("The Circular Singly Linked list is Empty..

"So new node is head node.");

newnode getNode();

readNode (newnode); head newnode;

else

printf("\n Enter the node roll no after which the" "insertion is to be made: ");

scanf ("d", &insdata);

do

if (lastroll==insdata)

last head;

newnode getNode(); readNode (newnode);

newnode-> link last -> link;

return;

last link = newnode;

else

last last -> link;

while (last I head); printf("The Insert Node is not Found");

\*headptr= head;

void deleteFirst (node headptr)

node head, delnode, last;

head headptr;

if (head NULL)

printf("\n Circular Singly Linked List is Empty ...");

return;

delnode head;

if (head link head) (

head= NULL;

else (

last head;

while (last -> link I head)

last last -> link;

head head -> link; last link= head;

printf("\n Deleted Data is ... \n"); printf("\n Roll Number: d", delnode-> roll);

printf("\n Name: ts", delnode->name);

releaseNode (delnode); headptr= head;

void deleteLast (node headptr)

node head, delnode, last, \*prev; head headptr;

if (head= NULL)

printf("\n Circular Singly Linked List is Empty ...");

return;

if (head> link head)

delnode head; head= NULL;

else

last = head;

while (last -> link 1 head) ( prev last;

last last link;

deinode last prev link head,

printf("\n Deleted Data is

\n");

printf("\n Roll Number Ad", delnode roll),

printf("\n Name to", delnode name);

releaseNode (delnode); headptr head,

void deleteMiddle (node headptr)

node head, delnode, last, prev

int deldata

head headptr (head NULL)

printf("\n Circular singly Linked List is impty

return;

printf("\n Enter the node Roll No for deleteion

la to be made");

scanf ("d", &deldata); last head,

if (head roll deldata)

delnode head; if (head link head)

head NULL

else

last head,

while (last link - head)

last last link head head link

last link- head,

prev head, last head link,

while(last head)

if (last roll deldata)

( delnode last/

prev link last link,

break;

prev prev-> link, last last -> link;

if (last head)

printf("Delete Node is Not Found"); return;

printf("\n Deleted Data is... \n");

printf("\n Roll Number: d", delnode roll);

printf("\n Name: te", delnode->name);

releaseNode (delnode); headptr= head;

void modifyNode (node \*head)

node last; int moddata;

if (head NULL)

(

printf("\n Circular Singly Linked List is Empty..."); return;

printf("\n Enter the node roll no for modification");

scanf("%d", &moddata);

last head;

do

if (last roll -- moddata)

printf("\n Enter the new data for the Node

printf("\n Enter the New Roll Number: ");

scanf("%d", &last roll);

printf("\n Enter the New Name: ");

scanf("%s", last->name);

return;

else

last last> link; )while (last = head);

printf("Modify Node is Not Found");

void view (node \*head)

node last; if (head NULL)

( printf("\nCircular Singly Linked List is Empty ...");

return;

last head;

do

printf("\n Roll Number: td, last->roll); last last link;

printf("\n Name: ts", last->name);

while (last - head);

The program displays the following output

Basic Operations in a Circular Singly Linked List...

0. Create List

1. Insert First

2. Insert Last 3. Insert Middle

4. Delete First

5. Delete Last

6. Delete Middle

7. Modify

8. View

9. Show Menu 10. Exit

74

Circular Singly Linked List is Empty

7 5

Circular Singly Linked List is Empty.

? 6

Circular Singly Linked List is Empty

? 7

Circular Singly Linked List is Empty

? 8 Circular Singly Linked List is Empty

20 Enter the Roll Number: 301 Enter the Name : Raj

Do u wish to Add Data in the List (y/n)? y

Enter the Roll Number: 302 Enter the Name : Rani

Do u wish to Add Data in the List (y/n)?y

Enter the Roll Number: 303

Enter the Name : Priya Do u wish to Add Data in the List (y/n) 7 n

78 Roll Number: 301

Name: Raj

Roll Number: 302

Name: Rani

Roll Number: 303 Name: Priya

71

Enter the Roll Number: 300 Enter the Name : Raju

72

Enter the Roll Number: 304 Enter the Name : Vasanth

7 8 Roll Number: 300

Name: Raju

Roll Number: 301

Name: Raj Roll Number: 302 Name: Rani

Roll Number: 303

Name: Priya Roll Number: 304

Name: Vasanth.

24 Deleted Data is

... Roll Number: 300

Name: Raju

? 5

Deleted Data is ...

Roll Number: 304 Name: Vasanth

? 8 Roll Number: 301

Name: Raj

Roll Number: 302

Name: Rani Roll Number: 303

Name: Priya

73

Enter the node roll no after which the insertion is to be made: 304

Insert Node is Not Found

73

Enter the node roll no after which the insertion is to be made 302

Enter the Roll Number 305

Enter the Name : Jai

7 8 Roll Number: 301

Name: Raj Roll Number: 302

Name: Rani

Roll Number: 305

Name: Jai

Roll Number: 303

Name Priya

76 Enter the node roll no to be deleted: 304 Delete Node is Not Found

76 Enter the node roll no to be deleted: 305

Deleted Data is ...

Roll Number: 305 Name Jai

28

Roll Number: 301

Name: Raj

Roll Number: 302

Name: Rani

Roll Number: 303

Name: Priya

77 Enter the node roll no for modification:

305 Modify Node is Not Found

77

Enter the node roll no for modification: 303

Modify the Data of the Node ... Enter the New Roll Number: 313 Enter the New Name: Geetha

7 8

Roll Number: 301

Name: Raj

Roll Number: 302 Name: Rani

Roll Number: 313

Name: Geetha

79

Basic Operations in a Circular Singly Linked List

0. Create List 1. Insert First

2. Insert Last

3. Insert Middle

4. Delete First

5. Delete Last

6. Delete Middle

7. Modify

8. View

9. Show Menu

10. Exit

24

Deleted Data is...

Roll Number: 301

Name: Raj

25

Deleted Data is Roll Number: 313

Name: Geetha

74

Deleted Data is... Roll Number: 302

Name: Rani

78

Circular Singly Linked List is Empty 73

Circular Singly Linked list is Empty. So new node is head node. Enter the Roll Number: 3000

Enter the Name : Ajith

7 8

Roll Number: 3000 Name: Ajith

? 10 End of run of your program ...

9.6

BASIC OPERATIONS IN A CIRCULAR DOUBLY LINKED LIST

The basic operations that can be performed on circular doubly linked lists are similar to the doubly linked lists, except that the last node is made to point the first node in the lis The basic operations that can be performed on circular doubly linked lists are,

Creation of a list Insertion of a node

Modification of a node

Deletion of a node

Traversal of a list.

9.6.1 CREATION OF A LIST

The circular doubly linked list in program 9.4, has two items (i.e., roll and name). Creation of list involves three processes.

Creating a node

Reading details for a node from user

Connect the node with the list.

Creating a doubly linked list starts with creating a node. Sufficient memory has to be allocated for creating a node. The information is stored in the memory, allocated by using the malloc() function of type node. In program 9.4, the function getNode(), is used for creating a node.

Algorithm for declaration of the structure NODE

Struct NODE

DATA:

Data Field

FLINK: Link Field (Address of next Struct NODE) BLINK: Link Field (Address of previous Struct NODE)

End Struct

Algorithm for GETNODE()

SIZE: INTEGER

NEWNODE: NODE

STEP 1: Set SIZE get the size of the NODE STEP 2: Set NEWNODE- Allocate space in memory for the size of SIZE and

return the initial address

STEP 3: Return NEWNODE End GETNODE()

After allocating memory for the structure of type node, the information for the items (ie, roll and name) has to be read from the user. In program 9.4, the function readNode (), is used for reading details for the node from the user.

Algorithm for READNODEO

READNODE(NEWNODE: NODE)

STEP 1: Read, NEWNODE-> DATA

STEP 2: Set NEWNODE-> FLINK= NEWNODE

STEP 3: Set NEWNODE-> BLINK= NEWNODE

STEP 4: Return End READNODE(

Connect the new node with the existing list. If the list is empty, set the head pointer of

the list to the new node, other wise connect the new node in the last position of the list. In program 9.4, the function createList(), is used to connect the new nodes with the list.

Algorithm for CREATELISTO

CREATELIST()

HEAD, LAST, NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE()

STEP 2: CALL READNODE(NEWNODE)

STEP 3: Set HEAD = NEWNODE

STEP 4: Set LAST = NEWNODE

STEP 5: If you want to add another NODE proceed otherwise Return HEAD.

STEP 6: Set NEWNODE = GETNODE()

STEP 7: CALL READNODE(NEWNODE)

STEP 8: Assign LAST-> FLINK= NEWNODE

STEP 9: Assign NEWNODE-> FLINK = HEAD

STEP 10: Assign NEWNODE-> BLINK = LAST

STEP 11: Assign HEAD-> BLINK NEWNODE STEP 12: Assign LAST LAST-> FLINK

=

STEP 13: Goto STEP 5

END CREATELISTO

9.6.2 INSERTION OF A NODE

One of the most primitive operations that can be done in a doubly linked list is the insertion of a node. Memory is to be

allocated for the new node (in a similar way that is done while creating a list) before reading the data. The new node will contain empty data field and empty forward and backward link fields. The data field of the new node is then stored with the information read from the user. Both the link fields of the new node are assigned to NULL The new node can then be inserted in the list at three different places namely,

Inserting as a first node in the list Inserting as a last node in the list

Inserting an intermediate node in the list.

Inserting as a first node in the list

The following steps are followed to insert a new node in the start of the list.

Get the new node using GETNODE(), and read the details of the node using READNODE

Check whether the list is empty or not (l.e., check whether the head pointer is to NULL or not). If the list is empty, assign new node as head. If the list is not empty.

pointing

follow the next steps.

The forward link field of the new node is made to point the first node (head node) in the

list by assigning the address of the first node.

The backward link field of the first node (head node) is made to point the new node, by

assigning the address of the new node.

Assign the new node as the head pointer. In program 9.4, the function insertFirst (), is used for inserting a new

node in

the first position of the list.

Algorithm for INSERT FIRSTO

INSERT\_FIRST(HEAD: NODE)

NEWNODE: NODE

STEP 1: Set NEWNODE = GETNODE() 2:

CALL READNODE (NEWNODE)

STEP

STEP 3: If (HEAD==NULL)

Set HEAD=NEWNODE

Return

[End of If structure]

STEP 4: Assign NEWNODE-> FLINK=HEAD STEP 5: Assign NEWNODE-> BLINK= HEAD-> BLINK

STEP 6: Assign HEAD-> BLINK-> FLINK= NEWNODE

STEP 7: Assign HEAD-> BLINK= NEWNODE

STEP 8: Assign HEAD = NEWNODE END INSERT\_FIRSTO

Inserting as a last node in the list

The following steps are followed to insert a new node at the end of the list. Get the new node using GETNODE(), and read the details of the node using READNODE().

Check whether the list is empty or not (i.e., check whether the head pointer is pointing

to NULL or not). If the list is empty, assign new node as a head node. If the list is not empty, follow the next steps.

The forward link field of the last node in the list is made to point the new node, by

assigning the address of the new node.

The backward link field of the new node is made to point the last node, by assigning the address of the last node.

The forward link field of the new node is set to NULL. In program 9.4, the function insertLast (), is used for inserting a new node in the last position of the list.

Algorithm for INSERT\_LAST()

INSERT LAST(HEAD: NODE)

NEWNODE: NODE

STEP 1: STEP 2: CALL READNODE(NEWNODE)

Set NEWNODE = GETNODE()

STEP 3: If (HEAD==NULL)

Set HEAD NEWNODE

Return

[End of If structure]

STEP 4: Set LAST HEAD->

BLINK

STEP 5: Assign NEWNODE-> FLINK = HEAD STEP 6: Assign NEWNODE-> BLINK=LAST

STEP 7: Assign LAST-> FLINK= NEWNODE

STEP 8: Assign HEAD-> BLINK= NEWNODE

END INSERT\_LAST()

Inserting an intermediate node in the list

The following steps are followed, to insert a new node in an intermediate position in

the list.

Get the new node using GETNODE(), and read the details of the node using READNODE() Check whether the list is empty or not (i.e., check whether the head pointer is pointing to NULL or not). If the list is empty, assign new node as a head node. If the list is not

empty, follow the next steps.

Get the address of the preceding node after which the new node is to be inserted The forward link field of the new node is made to point the next node (forward link

field

of the preceding node) by assigning its address.

The backward link field of the new node is made to point the preceding node by assigning the address of the preceding node. The backward link field of the next node (node before which the new node is to be

Inserted) is made to point the new node, by assigning the address of the new node.

The forward link field of the preceding node (node after which the new node is to be Inserted) is made to point the new node, by assigning the address of the new node. In program 9.4, the function insertMiddle(), is used for inserting a new

node in

the intermediate position of the list.

Algorithm for INSERT\_MIDDLE()

INSERT\_MIDDLE (HEAD: NODE)

LAST, NEXT, NEWNODE: NODE

CONDITION: DATA of the any one NODE in the LIST for Insert

STEP 1: Set NEWNODE=GETNODE() STEP 2: READNODE(NEWNODE)

STEP 3: If (HEAD == NULL)

Set HEAD NEWNODE

Return

[End of If structure]

STEP 4: Print, "Enter the data of NODE after which the Insertion is to be made"

STEP 5: READ, CONDITION

STEP 6: Set LAST = HEAD

STEP 7: Repeat

STEP 8: If (LAST-> DATA CONDITION) then

Assign NEWNODE-> FLINK LAST-> FLINK

Assign NEWNODE-> BLINK= LAST Assign

LAST->FLINK-> BLINK NEWNODE

Assign LAST-> FLINK = NEWNODE

Return

Else

Assign LAST LAST -> FLINK

(End of If Structure]

STEP 9: Until (LAST == HEAD)

STEP 10: Print, "CONDITION IS NOT AVAILABLE"

STEP 11: Retur

END INSERT\_MIDDLE()

MODIFICATION OF A NODE

A node(s) can be modified in a list, for changing its information part. The following

9.6.3

steps are followed to modify a node in the list. Check whether the list is empty or not (l.e., check whether the head pointer is pointing

to NULL or not). If the list is not empty, follow the next steps.

Search for the node to be modified.

Change the information part of the node.

In program 9.4, the function modifyNode(),

is used for modifying an existing node

in the list.

Algorithm for MODIFY\_NODEO

MODIFY\_NODE(HEAD: NODE)

STEP 1: If (HEAD == NULL)

Return (End of If structure]

STEP 2: Print, "Enter the data of NODE to be modified"

STEP 3: Read, CONDITION STEP 4: Set LAST = HEAD

STEP 5:

Repeat

STEP 6: If (LAST-> DATA == CONDITION) then

Read, LAST-> DATA

Return

Else

Assign LAST LAST -> FLINK

[End of If Structure]

STEP 7: Until (LAST != HEAD) STEP 8: Print, "CONDITION IS NOT AVAILABLE"

END MODIFY\_NODE()

9.6.4

DELETION OF A NODE

Another primitive operation that can be done in a doubly linked list is the deletion of a node. Memory is to be released for the node to be deleted. A node can be deleted from the

list from three different places namely,

Deleting the first node from the list. 900)

Deleting the last node from the list.

Deleting an intermediate node from the list.

Deleting the first node from the list

The following steps are followed, to delete a node from the start of the list.

Check whether the list is empty or not (i.e., check whether the head pointer is pointing

to NULL or not). If the list is not empty, follow the next steps. Set the head pointer to the second node in the list (by assigning its address).

Set the backward link field of the head node in the list to NULL. Release the memory for the deleted node.

In program 9.4, the function deleteFirst (), is used for deleting the first node in

the list. After deleting the node release the memory occupied by the deleted node by using the function releaseNode().

Algorithm for RELEASE\_NODE()

RELEASENODE(NEWNODE: NODE)

STEP 1: Deallocate the space for the NODE of NEWNODE Retur

STEP 2:

End RELEASENODE()

Algorithm for DELETE\_FIRST()

DELETE FIRST (HEAD: NODE) DELNODE: NODE

STEP 1: If (HEAD == NULL)

Print, "List is Empty"

Return

[End of If structure]

STEP 2: Set DELNODE = HEAD

STEP 3: If (HEAD-> FLINK== HEAD)

Set HEAD= NULL

Print, "Deleted Data is", DELNODE-> DATA CALL RELEASENODE (DELNODE)

Return

[End of If structure]

STEP 4: Assign HEAD-> BLINK-> FLINK= HEAD-> FLINK STEP 5: Assign HEAD-> FLINK-> BLINK= HEAD-> BLINK

STEP 6: Assign HEAD = HEAD-> FLINK

STEP 7: Print, "Deleted Data is", DELNODE-

> DATA STEP 8: CALL RELEASENODE(DELNODE)

STEP 9: Return END DELETE\_FIRST()

Deleting the last node in the list

The following steps are followed, to delete a node from the end of the list. Check whether the list is empty or not (l.e., check whether the head pointer is pointing

to NULL or not). If the list is not empty, follow the next steps. The forward link field of the previous node (from the end of the list) is set to NULL.

Release the memory for the deleted node. In program 9.4, the function deleteLast (), is used for deleting the last node in the list.

After deleting the node release the memory occupied by the deleted node by using the function releaseNode().

Algorithm for DELETE\_LAST()

DELETE LAST(HEAD: NODE)

LAST, PREV, DELNODE: NODE

STEP 1: If (HEAD == NULL) Print, "List is Empty"

Return

[End of If structure]

STEP 2:

Set DELNODE= HEAD-> BLINK

STEP 3: If (HEAD-> FLINK= == HEAD) Set

HEAD= NULL

Print, "Deleted Data is", DELNODE-> DATA CALL RELEASENODE (DELNODE)

Return

[End of If structure]

STEP 4: Set LAST HEAD-> BLINK

STEP 5: Assign LAST-> BLINK-> FLINK= HEAD STEP 6: Assign HEAD-> BLINK= LAST-> BLINK

STEP 7: Print, "Deleted Data is", DELNODE-> DATA

STEP 8: CALL RELEASENODE (DELNODE)

STEP 9: Return END DELETE\_LAST()

Deleting an intermediate node from the list

The following steps are followed, to delete a node from an intermediate position in the

list.

Check whether the list is empty or not (i.e., check whether the head pointer is pointing to NULL or not). If the list is not empty, follow the next steps.

The forward link field of the previous node (following the node to be deleted) is made to point the next node (before the node to be deleted), by assigning its address.

The backward link field of the next node (before the node to be deleted) is made to point the previous node (following the

node to be deleted), by assigning its address.

Release the memory for the deleted node. In program 9.4, the function deleteMiddle (), is used for deleting the intermediate node in the list. After deleting the node release the memory occupied by the deleted node by

using the function releaseNode().

Algorithm for DELETE\_MIDDLE()

DELETE\_MIDDLE(HEAD: NODE)

NEXT, PREV, LAST, DELNODE: NODE DELDATA: Data of the NODE is the List to Delete

STEP 1: If (HEAD== NULL) Print, "List is

Empty"

Return

[End of If Structure]

STEP 2: Print, "Enter the DATA STEP 3: Read, DELDATA

of any node in the List for Details"

STEP 4: If (HEAD-> DATA DELDATA) Then

Set DELNODE HEAD

If (HEAD-> FLINK== HEAD)

1300)

Set HEAD NULL

Else

Assign HEAD-> FLINK -> BLINK= HEAD -> BLINK Assign HEAD-> BLINK -> FLINK HEAD -> FLINK

Assign HEAD HEAD -> FLINK

[End of If Structure]

Print, "Deleted Data is", DELNODE-> DATA CALL RELEASENODE (DELNODE)

Return

[End of If Structure] STEP 6: Set LAST HEAD->

FLINK

STEP 7: Repeat While (LAST!= HEAD)

STEP 8: If (LAST-> DATA== DELDATA)

Then

Set DELNODE= LAST

Assign LAST->FLINK-> BLINK LAST -> BLINK Assign LAST-> BLINK-> FLINK= LAST -> FLINK

Print, "The Deleted Data is ", DELNODE-> DATA RELEASENODE (DELNODE)

Return HEAD

Else

Assign LAST LAST -> FLINK

[End of If structure]

STEP 8: [End of STEP 6 While Loop]

STEP 9: Print," DELDATA is Not Available in the List".,

STEP 10: Return HEAD

END DELETE\_MIDDLE()

9.6.5 TRAVERSAL OF A LIST

To read the information or to display the information in a linked list, you have to

traverse (move) a linked list, node by node from the first node, until reached. Traversing a list involves the following steps,

Assign the address of head pointer to a variable. Display the information in the data field.

Traverse the list from one node to another by advancing the pointer.

In program 9.4, the function view (), is used for traversing and to display the

information in the list.

Algorithm for VIEWO

VIEW(HEAD: NODE)

LIST: NODE

STEP 1: LIST HEAD =

STEP 2: If (LIST== NULL)

Print, "List is Empty"

Return

[End of If Structure]

STEP 3: Repeat

STEP 4: Print "The data is ", LIST-> DATA STEP 5: LIST LIST -> FLINK

STEP 6: Until (LIST != HEAD)

END VIEW

PROGRAM 9.4

/\* Program to create, insert, delete, modify and view in a Circular Doubly Linked List \*/

/ call.c \*/

#include <stdio.h> typedef struct list

int roll;

char name [20]; struct list flink;

struct list blink;

node;

node\* getNode();

void createList (node headptr);

void insertFirst (node headptr);

void insertLast (node \*\*headptr); void insertMiddle (node headptr);

void deleteFirst (node \*headptr);

void deleteLast (node \*headptr);

void modifyNode (node head); void deleteMiddle (node \*headptr);

void view (node head);

void releaseNode (node newnode); void displayMenu();

void main()

( node head NULL; int ch;

displayMenu(); while (1)

( printf("\n\n? ");

fflush(stdin); scanf("d", &ch);

switch (ch)

case 0:

createList (&head);

break;

case 1 :

insertFirst (&head); break;

case 2:

insert Last (&head);

break;

case 3:

insertMiddle (&head);

break;

case 4:

deleteFirst (&head) break;

case 5:

deleteLast (&head); break;

case 6:

deleteMiddle (&head);

break;

case 7:

modifyNode (head);

break;

case 8:

view (head); break;

case 9:

displayMenu(); break;

default:

printf("End of run of your program...");

exit (0);

void displayMenu()

printf("\nBasic Operations in a Circular Doubly Linked List printf("\n\n\t 0. Create

List ");

printf("\n\t 1. Insert First "); printf("\n\t 2. Insert Last ");

printf("\n\t 3. Insert Middle ");

printf("\n\t 4. Delete First ");

printf("\n\t 5. Delete Last "); printf("\n\t 6. Delete Middle "); printf("\n\t 7. Modify ");

printf("\n\t 8. View "); printf("\n\t 9. Show Menu ");

printf("\n\t 10. Exit ");

node\* getNode()

int size;

node newnode; size sizeof (node);

newnode (node)malloc(size);

return

void readNode (node newnode)

printf("\nEnter the Roll Number: "); scanf("%d", &newnode->roll);

printf("Enter the Name: ");

scanf("%s", newnode->name);

newnode->flink= newnode; newnode->blink newnode;

void releaseNode (node \*ptr)

free (ptr);

void createList (node headptr)

node head NULL, newnode, last;

char ch;

do

newnode getNode();

readNode (newnode); if (head NULL)

head newnode; last head;

else

newnode-> blink last,

newnode-> flink= head;

last flink= newnode; -> head blink = newnode;

last last -> flink;

fflush(stdin);

printf("Do u wish to Add Data in the List (y/n)

scanf("%c", &ch);

)while ((ch == 'y') || (ch = 'Y')); \*headptr= head;

7");

1

void insertFirst (node \*headptr)

(

node head, \*newnode;

head headptr; newnode getNode();

readNode (newnode);

if (head NULL)

head= newnode;

else

newnode-> flink head;

newnode-> blink head blink;

head blink -> flink=

head blink= newnode; head newnode;

\*headptr head;

void insertLast (node \*headptr)

node head, newnode, last; head headptr;

newnode getNode();

readNode (newnode); if (head= NULL)

head = newnode;

else

last head -> blink; newnode-> flink= head;

newnode-> blink last; last flink = newnode;

head blink = newnode;

} \*headptr= head;

void insertMiddle (node \*headptr)

node head, newnode, last, next;

int insdata;

head headptr; if (head= NULL)

printf("Circular Doubly Linked list is Empty.

"So new node is head node."); newnode getNode();

readNode (newnode); head newnode;

\*headptr head; return;

printf("\n Enter the node roll no after which

the" "insertion is to be made : : ");

scanf("%d", &insdata); last head;

do

if (last roll insdata)

newnode= getNode();

readNode (newnode); next last -> flink;

newnode-> flink= next;

newnode-> blink last;/

last flink newnode; next blink= newnode;

return;

else

last last flink;

while (last = head); printf("Insert Condition is Not Found");

void deleteFirst (node headptr)

node head, delnode, last; head headptr;

if (head NULL)

( printf("\n Circular Linked List is Empty...");

return;

delnode head;

if (head> flink

head NULL;

else

last head > blink, head head -> flink,

last flink= head; head blink last;

printf("\n Deleted Data is... \n"); printf("\n Roll Number: d", delnode-> roll),

printf("\n Name: 8", delnode->name)

releaseNode (delnode);

headptr head;

void deleteLast (node headptr)

node \*head, delnode, last, \*prev;

head \*headptr; if (head NULL)

printf("\n Circular Linked List is Empty ...");

return;

if (head flink head)

delnode head; head= NULL;

else

( last- head blinkke

delnode last; prev last -> blink;

prev flink= head;

head blink prev;

printf("\n Deleted Data is ... \n");

printf("\n Roll Number: d", delnode-> roll),

printf("\n Name: a", delnode->name);

releaseNode (delnode); \*headptr head;

void deleteMiddle (node headptr)

(

node head, last, prev, next, delnode/

int deldata;

head headptr

if (head NULL)

printf("\n Circular Linked List is Empty...");

return;

printf("\n Enter the node roll no for

deleteion is to be made :"); scanf("d", &deldata); last head;

do

if (last roll - deldata)

last- last flink; else

delnode last;

if (head> flink -- head) head= NULL;

else

prev last -> blink; next last -> flink;

if (last head) head head -> flink;

prev flink = next; next > blink = prev;

printf("\n Deleted Data is \n");

printf("\n Roll Number: %d", delnode-> roll); printf("\n Name: %s", delnode->name); headptr= head;

releaseNode (delnode);

return;

)while (last = head); printf("Delete Node is Not Found");

void modifyNode (node \*head)

node last; int moddata;

if (head= NULL)

printf("\n Circular Linked List is Empty ...");

return; printf("\n Enter the node roll no for modification:

scanf("%d", &moddata);

last - head;

do

if (last roll == moddata) -> (

printf("\n Modify the Data of the Node

printf("\n Enter the New Roll Number: "); scanf("%d", &last roll);

printf("\n Enter the New Name: "); scanf("%s", last->name);

return;

else

last last flink; while (last = head); printf("Modify Node is Not Found");

void view (node \*head)

node list; list = head;

if (list NULL)

} printf("\n Circular Doubly Linked List is Empty

return;

do

printf("\n Roll Number: d, list->roll);

printf("\n Name: %s", list->name); list list

flink;

while (list head);

The program displays the following output

Basic Operations in a Circular Doubly Linked List

0. Create List 1. Insert First

2. Insert Last 3. Insert Middle

4. Delete First

5. Delete Last 6. Delete Middle

7. Modify

8. View

9. Show Menu 10. Exit

...");

24

Circular Doubly Linked List is Empty 75

Circular Doubly Linked List is Empty

? 6

Circular Doubly Linked List is Empty

77

Circular Doubly Linked List is Empty..

Circular Doubly Linked List is Empty

Enter the Roll Number: 401 Enter the

Name: Anusha

Do u wish to Add Data in the List (y/n) ? y

Enter the Roll Number: 402 Enter the Name : Sussela

Do u wish to Add Data in the List (y/n) ? y

Enter the Roll Number: 403 Enter the Name Premi

Do u wish to Add Data in the List (y/n) ? n

78

Roll Number: 401 Name: Anusha

Roll Number: 402

Name: Sussela

Roll Number: 403 Name: Premi

71

Enter the Roll Number: 400 Enter the Name Ganesh

? 2

Enter the Roll Number: 404 Enter the Name : Sasi

7 8

Roll Number: 400

Name Ganesh

Roll Number: 401

Name: Anusha

Roll Number: 402

Name Suseela Roll Number: 403

Name Premi

Roll Number: 404 Name Sasi

74

Deleted Data la...

Roll Number: 400

Name Ganesh

76

Deleted Data is...

Holl Number: 404 Name: Basi

Roll Number: 401 Name Anusha

Roll Number: 402

Name: Sussela Roll Number: 403

Name Premi

93

Enter the node roll no after which the insertion is to be made 404

Insert Node is Not Found

Enter the node roll no after which the insertion is to be made 402

Enter the Roll Number 405 Enter the Name Suganya

Holl Number: 401

Name Anusha

Roll Number: 402

Name Suseela

Roll Number: 405

Name Suganya

Roll Number: 403

Name Premi

76

Enter the node roll no to be deleted 404 Delete Node is Not Found

76

Enter the node roll no to be deleted 405 Deleted Data is...

Roll Number 405 Name Buganya

7 8 Roll Number: 401

Name: Anusha

Roll Number: 402

Name: Sussela Roll Number: 403 Name: Premi

? 7

Enter the node roll no for modification: 405

Modify Node is Not Found 77

Enter the node roll no for modification : 403 Modify the Data of the Node

Enter the New Roll Number: 413

Enter the New Name: Selvi

28

Roll Number: 401 Anusha

Name: Roll Number:

Name :

Sussela

Roll Number: 413 Name: Selvi

79

Basic Operations in a Circular Doubly Linked List

0. Create List 1. Insert First

2. Insert Last 3. Insert Middle

4. Delete First

5. Delete Last 6. Delete Middle

7. Modify

8. View

9. Show Menu 10. Exit

? 4

Deleted Data is ... Roll Number: 401

Name: Anusha

75

Deleted Data is ...

Roll Number: 413 Name: Selvi

74 Deleted Data is

Roll Number: 402

Name: Sussela

78 Circular Doubly Linked List is Empty

? 3 Circular Doubly Linked list is Empty. So new node is head node. Enter the Roll Number: 4000

Enter the Name : Roy

? 8 Roll Number: 4000

Name: Roy

? 10

End of run of your program Advantages of linked lists

It is not necessary to specify the number of elements in a linked list during its

declaration (since memory can be allocated dynamically when a node is added to

the list). Linked list can grow and shrink in size depending upon the insertion and deletion

that occurs in the list. Insertions and deletions at any place in a list can be handled easily and efficiently.

A linked list does not waste any memory space.

Disadvantages of linked lists

Searching a particular element in a list is difficult and time consuming.

A linked list will use more storage space than an array to store the same number of elements (.. each element in a list needs additional memory space for storing the

address of the next node).

SUMMARY

An algorithm is a step-by-step recipe for solving an instance of a problem. Every single procedure that a computer performs is an algorithm.

A data structure represents the logical relationship that exists between individual elements of data to carry out basic tasks like insertion, deletion, searching, and so on.

\* Primary data structures are the basic data structures that directly operate upon the machine instructions. All the integers, floating-point number, character, string constants, and pointers are considered as primary data structures.

Secondary data structures emphasize on grouping same or different data items with relationship between each data item. They are broadly classified as static data structures and dynamic data structures.

If a data structure is created when the number of data items is known in advance, it is

known as static data structure. If a data structure is created, when the number of data items is not known in advance is known as dynamic data structure or variable size data structure. They are broadly

classified as linear data structures and non-linear data structures.

Linear data structures have a linear relation ship between its adjacent elements. Linked lists are examples of linear data structures.

A linked list is a linear dynamic data structure that can grow and shrink during its

execution time. A circular linked list is similar to a linked list except that the first and last nodes are

interconnected.

Non-linear data structures don't have a linear relationship between its adjacent elements. Trees and graphs are examples of non-linear data structures.

A tree is a non-linear dynamic data

structure that may point to one or more nodes at a time. A graph is similar to tree except that it has no hierarchical relationship between its adjacent elements.

Linked list or list is an ordered collection of elements in which each element in the list is referred as a node. Each node contains two fields namely, data field and Link field. The data field contains the actual data of the element to be stored in the list and the link field also referred as the next address field contains the address of the next node in the

list. External address is the address of the first node in the list stored in the head pointer

which points to the first node in the list. Internal address is the address stored in

the address field of every node except the last node.

Null address is the address stored by the Null pointer in the last node of the list. Singly linked list is a list in which each node as a single link to its next node.

Doubly linked list is a list in which each node as two links one its next node and the other to its previous node. Circular singly linked list is a list in which each node is connected to the next node as in the case of a singly linked list except that the list has no end i.e., the last node is connected

to the first node of the list.

Circular doubly linked list is a list in which each node is connected to the next node

as in the case of a doubly linked list except that the list has no end i.e., the last node is connected to the first node of the list.

The previous address field of a node contains address of its previous node. This field is also referred as the backward link field.

The next address field contains the address of the next node in the list. This field is also referred as the forward link field.

Some of the important applications of linked lists are, manipulation of polynomials, Stacks and Queues.

REVIEW QUESTIONS

Define data structures.

State the different types of data structures.

2

Define a node in a linked list.

What are the different types of fields in a linked list? What are the different types of linked lists?

State the advantages, disadvantages & applications of linked lists. State the basic operations performed in a lists.

Explain in brief insertion of nodes in various positions in a singly list. Explain in brief deletion of nodes in various positions in a singly list.

7.

10.

Explain in brief the modification of a node in a singly list.

11. Explain in brief the traversal of nodes in a singly list. Explain in brief insertion of nodes in various positions in a doubly linked list.

12.

13

14.

15.

16.

17.

18.

Explain in brief deletion of nodes in various positions in a doubly linked list.

Explain in brief the modification of a node in a doubly linked list.

Explain in brief the traversal of nodes in a doubly linked list.

Explain in brief insertion of nodes in various positions in a circular singly linked list. Explain in brief deletion of nodes in various positions in a circular singly linked list.

Explain in brief the modification of a node in a circular singly linked list. Explain in

brief the traversal of nodes in a circular singly linked list.

Explain in brief insertion of nodes in various positions in a circular doubly linked list.

Explain in brief deletion of nodes in various positions in a circular doubly linked list.

Explain in brief the modification of a node in a circular doubly linked list. Explain in brief the traversal of nodes in a circular doubly linked list.

EXERCISES

1.

Write a program to read N numbers from the user and store it in a singly linked list and sort them in ascending order.

2. Write a program to read N numbers from the user and store it in a doubly linked list and sort them in ascending order.

3.

Write a program to read N numbers from the user and store it in a circular singly

linked list and sort them in ascending order., Write a program to read N numbers from the user and store it in a circular doubly 4.

linked list and sort them in order. Write a program to read N names from the user

and store it in a sinlgy linked list and sort them in alphabetical order. 5.

6. Write a program to read regno and names of students in section A and section Bin two seperate singly linked lists and append the two singly linked lists into a single linked list.

7. Write a program to read regno and names of students in section A and section B in two seperate doubly linked lists and append the two singly linked lists into a single doubly linked list.

8.

Write a program to read regno and names of students in section A and section Bin two seperate circular singly linked lists and append the two singly linked lists into a

single circular linked list.