Singly Linked Lists

What is a linked list?

- General linear list in which operations such as insertion , retrievals, changes and deletions can be done anywhere in the list i.e at the beginning, in the middle or at the end of the list.

Advantages of linked list

- Since a linked list is a dynamic data structure, the size of the linked list can grow or shrink in size during execution of the program.
- It provides flexibility in rearranging the items efficiently.

Limitation of linked list

- It consumes extra space when compared to a array since each node must also contain the address of the next item in the list.

Types of Linked list

- **Types of linked list:**
  - Linear singly linked list
  - Circular singly linked list
  - Two way or doubly linked list
  - Circular doubly linked list.

Applications of Linked List

- Unpredicatable storage requirements.
- Extensive manipulation of stored data.
- Linked lists concepts are useful to model
  - queues
  - stacks
  - trees.

Algorithms of singly linked list

- Creating a linked list

Algorithm create_linked_list(struct link *header)

1. [initialize]
    1. header->next= NULL
    2. Point the node to the header
2. Repeat while(ch!='n')
    1. node->next= create a dynamic list of type struct link
    2. node=node->next
    3. Input the data for the node
    4. node->next = null
    5. Ask the user for continuing to create the list

Algorithms of singly linked list

- Displaying a linked list

Algorithm display_linked_list(struct link *header)

1. node=point to the first node
2. Repeat while(node)
    1. Display node->info
    2. node=node-> next

Algorithms of singly linked list

- Insert data at the beginning of the list

Algorithm insert_at_beg(struct link *header)

1. [declare]

   struct link *new1

2. [initialize]

   1. node= point to the first node
   2. previous=address of the header

3. new1 =create a dynamic list of type struct link

   point new1 to node

   point previous to new1

   Enter the value for the new node new1

Algorithm to Insert data at the end of the list

Algorithm insert_at_end(struct link *header)

1 . [declare]

struct link *new1

2. [initialize]

1. node= point to the first node
2. previous=address of the header

3. Repeat while(node)

1. node=point node to the next node
2. previous=point previous to the next node

if node=NULL

1. new1= create a dynamic list of type struct link
2. point new1 to node
3. point previous to new1
4. Enter the value for the new node new1

Algorithm to Insert data in the middle of the list

Algorithm insert_in_the_middle(struct link *header)

1. [initialize]

    int node_num=1;

    int insert_node;

    node=point to the next node

    previous=address of the header

2. Input the location at which the node is to be inserted.

Algorithm to Insert data in the middle of the list Continued..

3. if(insert_node<=node_counts)
   1. Repeat while(node)
      if(node_num= insert_node)
         1. new1 =create a dynamic list of type struct link
         2. point new1 to node
         3. point previous to new1
      4. Enter the value for the new node new1

else
   1. point node to the next node
   2. point previous to the next node

   node_num++;

else

   display " THERE ARE ONLY 'node_counts' NODES IN THE LIST"

Algorithm to delete first node in the list

Algorithm delete_first_node(struct link *header)

1. [initialize]
    1. node=point to the next node
    2. previous=address of the header
2. If node=NULL
    1. display “ THERE ARE NO NODES”
    2. return

    else
    1. previous->next= node->next
    2.free the node
3. node=header->next

Algorithm to delete last node in the list

Algorithm

delete_last_node(struct link *header)

1. [initialize]

   node_number=node_counts

   node=pointer to the first node

   previous=address of the header

2. If node=NULL

   1. display " THERE ARE NO NODES"

   2. return

else

1. Repeat while (node and node_number != 1)

   1. node= point node to the next node

   2. previous=point previous to the next node

   3. node_number - -

2. if(node_number=1)

   1. previous->next= node->next

   2.free the node

Algorithm to delete desired node from the list

Algorithm delete_desired_node(struct link *header)

1.[initialize]

node_number=1

delete_node=0

node=point to the first node

previous=address of the header

2. Enter the node number you want to delete

3. Repeat while(node)

1.if node_number=delete_node

1. previous->next=node->next

2. free(node)

else

1. node=point node to the next node

2. previous=point previous to the next node

node_number++

Operations on Singly Linked Lists

Algorithms

Searching for a data in the list

Algoritjm search_the_list(struct link *header)

1.[initialize]
   node_no=1
   flag=0
   node=point to the next node
2. [Input the data to be searched]
   input data for srch_info
3. if(node=NULL)
   1. display "LIST IS EMPTY"
4.Repeat while(node)
   1. if(node->info = srch_info)
      a. Display “INFO TO BE SEARCHED is srch_info”
      b. Display “ AT THE LOCATION : node_no
      c. Display " SEARCH IS SUCCESSFUL“
      d. flag=1
5.
   a. node=point to the next node
   b. increment node_no
6. if(flag=0)
   Display “SEARCH UNSUCCESSFUL"

Reversing the list

Algorithm reverse_the_list(struct link *header)

1. [Declare]

   struct link *s,*r

   node=point to the next node

   r=NULL

2. Repeat while(node)
   1. s=r
   2. r=node
   3. node=point to the next node
   4. r->next=s
3. node=r
4. Repeat while(node)
   1. Display node->info
   2. node=point to the next node

Concat two lists

Algorithm concat_two_lists(struct link *header1,struct link *header2)

1.{initialize]

struct link *node1,*previous1

previous1=point to header1

node1=point to the next node ofheader1

2. If node1=NULL

header1->next=header2->next;

3. Repeat while(node1)

1. node1=point to the next node
2. previous1=point to the next node

4. previous1->next =point to the next of header2

5. node1=point to the next of header1

6. Repeat while(node1)

1. Display node1->info
2. node1=node1->next

Sorting the list

Void sort_the_list(struct link *header)

1. [Initialize]
   struct link *temp1,*temp2
   // create a temporary node
   temp=0
2. if(temp1->next==NULL)
   1. Display “THERE IS NO LINKED LIST. ENTER CHOICE 1 to CREATE THE LINKED LIST"
3. Repeat for(temp1 = header->next ; temp1!=NULL ; temp1 = temp1->next)
   1. Repeat for(temp2 = temp1->next ; temp2!=NULL ; temp2 = temp2->next)
      1. if(temp1->info > temp2->info)
         a. temp = temp1->info
         b. temp1->info = temp2->info
         c. temp2->info = temp
4. node=header->next
5. node=point to the first node
6. Repeat while(node)
   1. Display node->info
   2. node=point to the next node

Merging the two Lists

```
void merge_two_lists(struct link
    *header1,struct link
    *header2,struct link *header3)
1.[Declare]
        struct link *node1,*node2,*z1
        struct link *z
2.[Initialize]
    node1=header1->next
    node2=header2->next
    z=header3
3.  if(node1=NULL  and node2 = NULL)
        return
4   Repeat  A,B,C
    while(node1!=NULL and
    node2!=NULL)
    A.  if(node1->info<node2->info)
        1. z->next=Create a
                dynamic linked list
        2. z=z->next
        3. z->info=node1->info
        4. z->next=NULL
        5. node1=node1->next
    B. if(node1->info>node2->info)
        1. z->next=Create a
                dynamic linked list
        2. z=z->next
        3. z->info=node2->info
        4. z->next=NULL
        5.node2=node2->next
```

C. if(node1->info=node2->info)

1. z->next=create a dynamic linked list
2. z=z->next
3. z->info=node2->info
4. z->next=NULL
5. node1=node1->next
6. node2=node2->next

4. Repeat while(node1)

1. z->next=Create a Dynamic linked list
2. z=z->next
3. z->info=node1->info
4. z->next=NULL
5. node1=node1->next

5. Repeat while(node2)

1. z->next=Create a Dynamic linked list
2. z=z->next
3. z->info=node2->info
4. z->next=NULL
5. node2=node2->next

6. z1=header3->next;

Display “MERGED LIST IS :”

7. Repeat while(z1)

1. Display z1->info
2. z1=z1->next

Linked List implementation of Stack Operations

Push into the stack

Algorithm pushstack(struct link *header,int x)

1.[Initialize]

pnode=point to the next of header

previous=address of the header

2. pnew=create a dynamic link

pnew->info=x

pnew->next=node

previous->next=pnew

Pop from the stack

Algorithm popstack(struct link *header)

return int

1. [Initialize]

   struct link *temp

   node=point to the next node

   previous=address of the header

   temp=node

2. no=temp->info

   previous->next=node->next

   free the node

   return no

Display the stack

Algorithm displaystack(struct link *header)

1. [Initialize]

   node=point to the next node

2. Repeat while(node)

   1. Display node->info
   2. node=point to the next node

Linked List implementation of queue Operations

Creating a queue

struct queue* createqueue()

1.[Declare]

struct queue *q

2. Dynamically allocate memory to q

3. if(q)

   1. q->front=NULL
   2. q->count=0
   3. q->rear=NULL

return q

Display the contents of queue

void displayqueue(struct queue* q)

1. [initialize]

   struct node* xnode

   xnode=q->front;

2. Repeat while(xnode)

   1. Display xnode->info
   2. xnode=point to the next xnode

Add data to the queue(Enqueue)

```
void enqueue(struct queue* q)
1.    [Initialize]
       struct node* newptr
       data,   ch=' '
2.   ch=get a character from the
     user
3. Repeat while(ch!='n')
     1. newptr=create a
          node dynamically
     2. Enter value for data
     3. newptr->info=data
     4. newptr->next=NULL
5.if(q->count=0)
        q->front=newptr
   else
        q->rear->next=newptr
6. (q->count)++
7.  q->rear=newptr
8. Display "do u want to
     continue(y/n)"
9. ch=get a character from
     the user
```

Delete data from the queue(Dequeue)

```
void dequeue(struct queue* q)
1.  [Initialize]
    struct node *deleteloc
    deleteloc=q->front
2.  if(q->count ==-1)
        q->rear=q->front=NULL
    else
        q->front=q->front->next
3.  (q->count)--
    if(q->count=0)
        Display ” no more data in the queue"
        free the deleteloc
```

Reading data from the front of the queue

```
void queuefront(struct queue* q)
1.[Initialize]
   result
2. if(q->count==0)
        result=0
   else
        result= q->front->info
3. Display "data at the front of the queue is : “,
   result
```

Reading data from the rear end of the queue

```
void queuerear(struct queue* q)
1.[Initialize]
   result
2. if(q->count==0)
        result=0
   else
        result= q->rear->info
3. Display "data at the front of the queue is : “,
   result
```

Count the data in the queue

```
void queuecount(struct queue* q)
1. [Initialize]
        result
2. if(q->count=0)
        result=0
   else
        result= q->count
3. Display "Total data in the queue is : ", result
```