University of Anbar
College of Computer Science
and Information Technology
Computer Network Systems
Department

Data Structures

Lecture Ten

Second Stage

First Course - 2024-2025

Muhmmad shihab muayad Shihab

Master of Computer Engineering

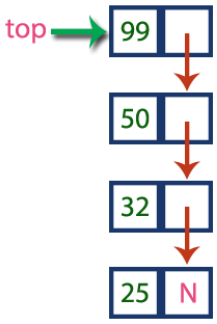
muhmmad.shihab@uoanbar.edu.iq

Linked Stack

The major problem with the stack implemented using an array is, it works only for a fixed number of data values. That means the amount of data must be specified at the beginning of the implementation itself. Stack implemented using an array is not suitable, when we don't know the size of data which we are going to use. A stack data structure can be implemented by using a linked list data structure. The stack implemented using linked list can work for an unlimited number of values. That means, stack implemented using linked list works for the variable size of data. So, there is no need to fix the size at the beginning of the implementation. The Stack implemented using linked list can organize as many data values as we want.

In linked list implementation of a stack, every new element is inserted as '**top**' element. That means every newly inserted element is pointed by '**top**'. Whenever we want to remove an element from the stack, simply remove the node which is pointed by '**top**' by moving '**top**' to its previous node in the list. The **next** field of the first element must be always **NULL**.

Example



In the above example, the last inserted node is 99 and the first inserted node is 25. The order of elements inserted is 25, 32,50 and 99.

Stack Operations using Linked List

To implement a stack using a linked list, we need to set the following things before implementing actual operations.

- Step 1 - Include all the **header files** which are used in the program. And declare all the **user defined functions**.
- Step 2 - Define a '**Node**' structure with two members **data** and **next**.

- Step 3 - Define a **Node** pointer '**top**' and set it to **NULL**.
- Step 4 - Implement the **main** method by displaying Menu with list of operations and make suitable function calls in the **main** method.

➢ *push(value) - Inserting an element into the Stack*

We can use the following steps to insert a new node into the stack...

- Step 1 - Create a **newNode** with given value.
- Step 2 - Check whether stack is **Empty** (**top** == **NULL**)
- Step 3 - If it is **Empty**, then set **newNode** → **next** = **NULL**.
- Step 4 - If it is **Not Empty**, then set **newNode** → **next** = **top**.
- Step 5 - Finally, set **top** = **newNode**.

➢ *pop() - Deleting an Element from a Stack*

We can use the following steps to delete a node from the stack...

- Step 1 - Check whether **stack** is **Empty** (**top == NULL**).
- Step 2 - If it is **Empty**, then display **"Stack is Empty!!! Deletion is not possible!!!"** and terminate the function
- Step 3 - If it is **Not Empty**, then define a **Node** pointer '**temp**' and set it to '**top**'.
- Step 4 - Then set '**top** = **top** → **next**'.
- Step 5 - Finally, delete '**temp**'. (**free(temp)**).

➢ *display() - Displaying stack of elements*

We can use the following steps to display the elements (nodes) of a stack...

- Step 1 - Check whether stack is **Empty** (**top** == **NULL**).
- Step 2 - If it is **Empty**, then display **'Stack is Empty!!!'** and terminate the function.
- Step 3 - If it is **Not Empty**, then define a Node pointer **'temp'** and initialize with **top**.
- Step 4 - Display '**temp** → **data** --->' and move it to the next node. Repeat the same until **temp** reaches to the first node in the stack. (**temp** → **next** != **NULL**).
- Step 5 - Finally! Display '**temp** → **data** ---> **NULL**'.

Implementation of Stack using Linked List | C Programming

```
#include<iostream>
```

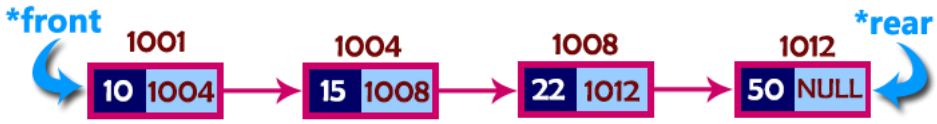
```
using namespace std;
struct Node
{
   int data;
   struct Node *next;
}*top = NULL;
void push(int);
void pop();
void display();
int main()
{
   int choice, value;
   cout<<"\n:: Stack using Linked List ::\n";
   while(1){
      cout<<"\n****** MENU ******\n";
      cout<<"1. Push\n2. Pop\n3. Display\n4. Exit\n";
      cout<<"Enter your choice: ";
      cin>>choice;
      switch(choice){
          case 1: cout<<"Enter the value to be insert: ";
                  cin>>value;
                  push(value);
                  break;
          case 2: pop(); break;
          case 3: display(); break;
          case 4: exit(0);
          default:  cout<<"\nWrong  selection!!!  Please  try
again!!!\n";
      }
   }
return 0;
}
void push(int value)
{
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(top == NULL)
      newNode->next = NULL;
```

```
    else
      newNode->next = top;
    top = newNode;
    cout<<"\nInsertion is Success!!!\n";
}
void pop()
{
    if(top == NULL)
      cout<<"\nStack is Empty!!!\n";
    else{
      struct Node *temp = top;
      cout<<"\nDeleted element: "<<temp->data;
      top = temp->next;
      free(temp);
    }
}
void display()
{
    if(top == NULL)
      cout<<"\nStack is Empty!!!\n";
    else{
      struct Node *temp = top;
      while(temp->next != NULL){
          cout<<temp->data<<"--->";
          temp = temp -> next;
      }
      cout<<temp->data<<"--->NULL";
    }
}
```

Linked Queue

The major problem with the queue implemented using an array is, It will work for an only fixed number of data values. That means, the amount of data must be specified at the beginning itself. Queue using an array is not suitable when we don't know the size of data which we are going to use. A queue data structure can be implemented using a linked list data structure. The queue which is implemented using a linked list can work for an unlimited number of values. That means, queue using linked list can work for the variable size of data (No need to fix the size at the beginning of the implementation). The Queue implemented using linked list can organize as many data values as we want.

In linked list implementation of a queue, the last inserted node is always pointed by '**rear**' and the first node is always pointed by '**front**'.



In above example, the last inserted node is 50 and it is pointed by '**rear**' and the first inserted node is 10 and it is pointed by '**front**'. The order of elements inserted is 10, 15, 22 and 50.

Operations

To implement queue using linked list, we need to set the following things before implementing actual operations.

- Step 1 - Include all the **header files** which are used in the program. And declare all the **user defined functions**.
- Step 2 - Define a '**Node**' structure with two members **data** and **next**.
- Step 3 - Define two **Node** pointers '**front**' and '**rear**' and set both to **NULL**.
- Step 4 - Implement the **main** method by displaying Menu of list of operations and make suitable function calls in the **main** method to perform user selected operation.

➢ *enQueue(value) - Inserting an element into the Queue*

We can use the following steps to insert a new node into the queue...

- Step 1 - Create a **newNode** with given value and set '**newNode → next**' to **NULL**.
- Step 2 - Check whether queue is **Empty** (**rear** == **NULL**)

- Step 3 - If it is **Empty** then, set **front** = **newNode** and **rear** = **newNode**.
- Step 4 - If it is **Not Empty** then, set **rear** → **next** = **newNode** and **rear** = **newNode**.

➢ *deQueue() - Deleting an Element from Queue*

We can use the following steps to delete a node from the queue...

- Step 1 - Check whether **queue** is **Empty** (**front == NULL**).
- Step 2 - If it is **Empty**, then display **"Queue is Empty!!! Deletion is not possible!!!"** and terminate from the function
- Step 3 - If it is **Not Empty** then, define a Node pointer '**temp**' and set it to '**front**'.
- Step 4 - Then set '**front** = **front** → **next**' and delete '**temp**' (**free(temp)**).

➢ *display() - Displaying the elements of Queue*

We can use the following steps to display the elements (nodes) of a queue...

- Step 1 - Check whether queue is **Empty** (**front** == **NULL**).
- Step 2 - If it is **Empty** then, display **'Queue is Empty!!!'** and terminate the function.
- Step 3 - If it is **Not Empty** then, define a Node pointer **'temp'** and initialize with **front**.
- Step 4 - Display '**temp** → **data** --->' and move it to the next node. Repeat the same until '**temp**' reaches to '**rear**' (**temp** → **next** != **NULL**).
- Step 5 - Finally! Display '**temp** → **data** ---> **NULL**'.

Implementation of Queue Datastructure using Linked List - C Programming

```
#include<stdio.h>
#include<conio.h>
#include<iostream>
using namespace std;
struct Node
{
   int data;
   struct Node *next;
}*front = NULL,*rear = NULL;
void insert(int);
void delet();
void display();
```

```
int main()
{
  int choice, value;
  cout<<"\n:: Queue Implementation using Linked List ::\n";
  while(1){
    cout<<"\n****** MENU ******\n";
    cout<<"1. Insert\n2. Delete\n3. Display\n4. Exit\n";
    cout<<"Enter your choice: ";
    cin>>choice;
    switch(choice){
        case 1: cout<<"Enter the value to be insert: ";
                cin>>value;
                insert(value);
                break;
        case 2: delet(); break;
        case 3: display(); break;
        case 4: exit(0);
        default: cout<<"\nWrong selection!!! Please try
again!!!\n";
    }
  }
return 0;
}
void insert(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  newNode -> next = NULL;
  if(front == NULL)
    front = rear = newNode;
  else{
    rear -> next = newNode;
    rear = newNode;
  }
  cout<<"\nInsertion is Success!!!\n";
}
void delet()
{
```

```
  if(front == NULL)
    cout<<"\nQueue is Empty!!!\n";
  else{
    struct Node *temp = front;
    front = front -> next;
    cout<<"\nDeleted element: "<< temp->data<<"\n";
    free(temp);
  }
}
void display()
{
  if(front == NULL)
    cout<<"\nQueue is Empty!!!\n";
  else{
    struct Node *temp = front;
    while(temp->next != NULL){
        cout<<temp->data<<"--->";
        temp = temp -> next;
    }
    cout<<temp->data<<"--->NULL\n";
  }
}
```

References:

- Frank Carrano, D.J. Henry: Data Abstraction and Solving with C++, 2012, 6th edition, Pearson Education, Inc.
- Mark Allen Weiss: Data Structures and Algorithm Analysis in C++, 2014, 4th edition, Pearson Education, Inc.