CS2023

THA-WEEK 5

Weijith Wimalasiri

210730B

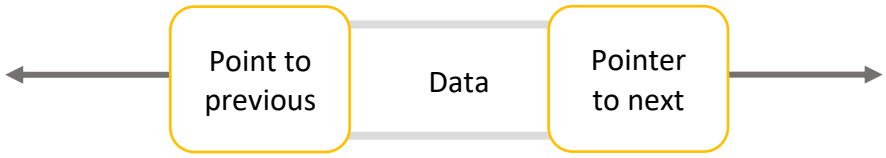
Table of Contents

Question 1

Explain briefly about what a double linked list and a circular linked list is?

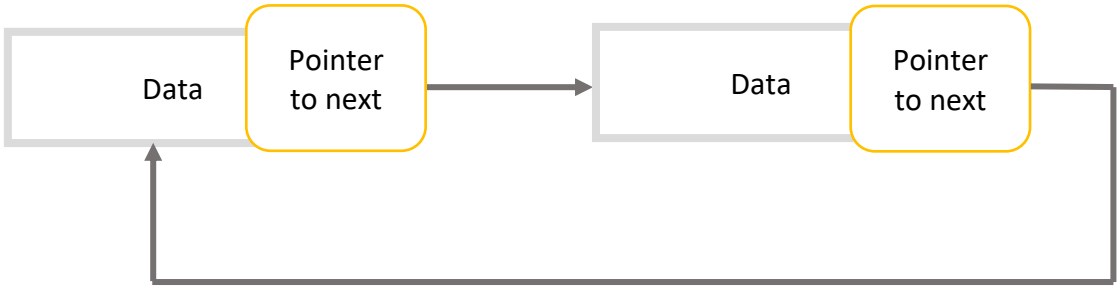
Double Linked List

Double linked list is very much similar to the Linked List data structure, except Double linked list has a pointer to the previous node which means unlike in Linked lists here we can traverse to the forward direction as well as the backward direction.



Circular Linked list

Circular Linked list is an extraction of Linked List where all are connected to form a circle. Unlike linked lists, in circular linked lists' last node points to the first node. Hence we can traverse through the Circular Linked List and reach the node where we started. Furthermore Circular Linked list can be implemented from both single linked list and double linked list.



Question 2

Write pseudocode for the operation of a single linked list.

Traversal

The traversal of the Single Linked list is possible starting from the head node. If there are 'n' nodes then the time complexity of the traversal is O(n) because it traverses through all the elements.

Pseudocode for Traversal

```
node *pointer = head;
while(pointer!= NULL)
      print(pointer -> data<< " ");
      pointer = pointer -> nxt;
end while;
```

Insertion

Insertion could be done at the beginning of the list, at some position in the list and the end of the list.

Pseudocode for insertion at the beginning

```
node *new_node = new node;
new_node -> data = data;
new_node -> nxt  = (*head);
(*head)  = new_node;
```

Pseudocode for insertion at some position

```
  if(pos==1)
     //Insertion at the Beginning
  else
     node *pos_node = new node;
     pos_node = (*head);
     node *new_node = new node;
     new_node -> data = data;
     new_node -> nxt  = NULL;
     while(--pos>1)
        pos_node = pos_node ->nxt;
     new_node->nxt = pos_node->nxt;
     pos_node->nxt = new_node;
```

Pseudocode for insertion at the end

```
node *end_node = new node;
  end_node = (*head);
  while(end_node->nxt != NULL)
     end_node = end_node -> nxt;
  end while
  node *new_node = new node;
  new_node -> data = data;
  new_node -> nxt = NULL;
  end_node -> nxt = new_node;
```

Deletion

Deletion also could be done at the beginning, at some and at the end.

Pseudocode for deletion at the beginning

```
if((*head)== NULL){
      print(“Error, Can't delete from an empty list");
   node *head_node = new node;
   head_node = (*head);
   (*head) = (*head)->nxt;
   delete head_node;
```

Pseudocode for deletion at some position

```
   if(pos == 1)
      //Deletion at the beginning

   node *pos_node = new node;
   pos_node =(*head);
   pos--;
   while(pos > 1)
      pos_node = pos_node->nxt;
      pos--;
   end while;
   node *new_node = new node;
   new_node = pos_node->nxt;
   pos_node->nxt = new_node->nxt;
   delete new_node;
```

Pseudocode for deletion at the end

```
if(head==NULL)
      print("The list is empty”)
   if((*head) -> nxt = NULL)
      delete (*head);
   node *pointer = new node;
   pointer = (*head);
   while(pointer->nxt->nxt != NULL)
      pointer= pointer->nxt;
   end while;
   node *del_node = new node;
   del_node = pointer->nxt;
   delete pointer;
   pointer->nxt = NULL;
```

Search

If a particular key is given then we have to find the node whose node matches the key.

```
node *pointer = new node;
   pointer  = (*head);
   int pos = 0;
   while(pointer != NULL)
      pos ++;
      if(pointer-> data == element){
          print(pos)
      pointer = pointer -> nxt;
    end while;
   else:
         print("No element like that is in the list")
```

Question 3

How can you implement a stack and a queue using linked list?

A Stack using Linked List

Stack is a linear data structure that follows the Last in, First out method. Stacks support various functions like push(), pop(), peek(). Advantage of using a linked list over an array is that it can grow dynamically.

Push

```
if the top is equal to NULL
  new_node -> nxt = NULL
else
  new_node -> nxt = top
```

Pop

```
if top == NULL
    print "Stack Underflow"
else
   create temp node, *temp = top
   create temp variable, tempData = top->data
   top = top->nxt
   free(temp)

return tempData
```

Peek

```
If top != NULL
  return top
else
  exit
```

A Queue using Linked List

Queue is a data structure which follows First-In-First-Out method. A queue support basic function like enqueue and dequeue.
A linked queue consists of two pointers, one for the front and one of the rear.

Enqueue

Insertion is performed to the rear end.

```
*pointer- > data = val
if  front == NULL
   front = pointer
   rear = pointer
   front - > nxt = NUL;
   rear - > nxt = NULL
else
   rear -> nxt = ptr
   rear = ptr
   rear ->nxt = NULL
```

Dequeue

Deletion is performed at the front end of the queue.

```
If front == NULL
   print “Underflow”
else
   *ptr = front;
   front = front -> next;
   free(ptr);
```

References

1. https://www.scaler.com/topics/doubly-linked-list-cpp/
2. https://www.scaler.com/topics/data-structures/circular-linked-list/
3. https://www.scaler.com/topics/linked-list-in-cpp/
4. https://www.scaler.com/topics/c/stack-using-linked-list-in-c/
5. https://www.scaler.com/topics/c/implementation-of-queue-using-linked-list/