Singly Linked List (Basic)

{Insertion}

Insert At Beginning ()

```
struct Node *node=(struct Node *) malloc (sizeof(struct Node));
node->next=NULL;
node->data=data;
if(start==NULL)
{
start=node;
head=start;
}
else
{
node->next=head;
head=node;
}
```

Insert At End ()

```
struct Node *node=(struct Node *) malloc (sizeof(struct Node));
node->next=NULL;
node->data=data;
if(start==NULL)
{
start=node;
head=node;
}
else
{
struct Node *temp=head;
while(temp)
{
if(temp->next==NULL) break;
temp=temp->next;
}
temp->next=node;
```

Singly Linked List

```
start=node;
}
```

Insert At Position ()

```
struct Node *node=(struct Node *) malloc (sizeof(struct Node));
node->next=NULL;
node->data=data;
if(position==1)
{
node->next=head;
head=node;
}
else
{
int i=1;
struct Node *p,*q;
p=head;
while(p && i<position)
{
i++;
q=p;
p=p->next;
}
q->next=node;
node->next=p;
}
```

{Deletion}

deleteAtBegining ()

```
struct Node *temp=head;
if(temp->next!=NULL) head=temp->next;
else start=head=NULL;
free(temp);
```

Singly Linked List

deleteAtEnd ()

```
struct Node *temp,*p;
temp=head;
p=start;
if(head==start)
{
head=start=NULL;
free(temp);
```

deleteAtPosition()

```
struct Node *temp,*p;
int position,i;
printf("Enter position: ");
scanf("%d",&position);
i=1;
temp=head;
if(position==1)
{
start=head=NULL;
free(temp);
printf("Node deleted successfully\n");
return;
}
else
{
while(temp && i<position)
{
i++;
p=temp;
temp=temp->next;
}
if(temp==NULL)
{
```

```
printf("Position invalid\n");
}
else
{
p->next=temp->next;
free(temp);
}
```

Singly Linked List (OOPS + Template)

Basic Structure

```
#include<iostream>
#define true 1
#define false 0
#define TRUE 1
#define FALSE 0
#define bool int
#define boolean int
namespace tmstl
{
template<class T>
class SinglyLinkedList;

template<class T>
class SinglyLinkedListNode;

template<class T>
class SinglyLinkedListIterator;

//SinglyLinkedListNode
template<class T>
class SinglyLinkedListNode
{
```

Singly Linked List

```
private:
SinglyLinkedListNode *next;
T data;
SinglyLinkedListNode(T);
friend class SinglyLinkedList<T>;
friend class SinglyLinkedListIterator<T>;
};

//SinglyLinkedListIterator
template<class T>
class SinglyLinkedListIterator
{
private:
SinglyLinkedListNode<T> *node;
SinglyLinkedListIterator(SinglyLinkedListNode<T> *);

public:
boolean hasNext();
T next();
friend class SinglyLinkedList<T>;
};

//SinglyLinkedList
template<class T>
class SinglyLinkedList
{
private:
SinglyLinkedListNode<T> *start;
SinglyLinkedListNode<T> *end;
int size;

public:
SinglyLinkedList();
```

Singly Linked List

```
SinglyLinkedList(const SinglyLinkedList<T> &);
virtual ~SinglyLinkedList();
SinglyLinkedList<T> & operator=(SinglyLinkedList<T>);

//Functionalities for singly linked list....
void add(T);
void insert(int,T);
T remove(int);
void clear();
T get(int);
int getSize();
SinglyLinkedListIterator<T> * getIterator();
};          //SinglyLinkedList ends here.....
};          //Namespace ends here.....
```

Functionality Code

SinglyLinkedListNode :-

```
template<class T>
SinglyLinkedListNode<T>::SinglyLinkedListNode(T data)
{
this->data=data;
this->next=NULL;
}
```

SinglyLinkedListIterator :-

```
template<class T>
SinglyLinkedListIterator<T>::SinglyLinkedListIterator(SinglyLinkedListNode<T> *node)
{
this->node=node;
}
template<class T>
boolean SinglyLinkedListIterator<T>::hasNext()
{
```

```
return this->next!=NULL;
}
template<class T>
T SinglyLinkedListIterator<T>::next()
{
if(this->node==NULL) return 0;
T data;
data=this->node->data;
this->node=this->node->next;
return data;
}
```

SinglyLinkedList :-

```
template<class T>
SinglyLinkedList<T>::SinglyLinkedList()
{
this->start=NULL;
this->end=NULL;
this->size=0;
}
template<class T>
SinglyLinkedList<T>::SinglyLinkedList(const SinglyLinkedList<T> &otherSinglyLinkedList)
{
this->size=0;
this->start=NULL;
this->end=NULL;
SinglyLinkedList<T> *node;
node=otherSinglyLinkedList.start;
while(node!=NULL)
{
this->add(node->data);
node=node->next;
}
```

```
}
template<class T>
SinglyLinkedList<T>:: ~SinglyLinkedList()
{
this->clear();
}
template<class T>
SinglyLinkedList<T> & SinglyLinkedList<T>::operator = (SinglyLinkedList<T> otherSinglyLinkedList)
{
this->clear();
SinglyLinkedList<T> *node;
node=otherSinglyLinkedList.start;
while(node!=NULL)
{
this->add(node->data);
node=node->next;
}
return *this;
}
template<class T>
void SinglyLinkedList<T>::add(T data)
{
SinglyLinkedListNode<T> *node;
node=new SinglyLinkedListNode<T>(data);
if(this->start==NULL)
{
this->start=node;
this->end=node;
}
else
{
this->end->next=node;
this->end=node;
```

```
}
this->size++;
}
template<class T>
void SinglyLinkedList::insert(int position,T data)
{
if(position<0) position=0;
if(position>this->size) position=this->size;
if(position==this->size)
{
this->add(data);
return;
}
SinglyLinkedListNode<T> *node;
node=new SinglyLinkedListNode<T>(data);
if(position==0)
{
node->next=start;
start=node;
}
else
{
SinglyLinkedListNode<T> *p,*q;
p=start;
int i=0;
while(i<position)
{
q=p;
p=p->next;
i++;
}
node->next=p;
q->next=node;
```

```
}
this->size++;
}
template<class T>
T SinglyLinkedList<T>::delete(int position)
{
if(position<0 || position>this->size) return 0;
T data;
SinglyLinkedListNode<T> *node;
if(this->size==1)
{
node=this->start;
data=node->data;
this->start=NULL;
this->end=NULL;
delete node;
this->size=0;
return data;
}
SinglyLinkedListNode<T> *p;
node=this->start;
int x=0;
while(x<position)
{
p=node;
node=node->next;
x++;
}
if(this->start==node)
{
this->start=this->start->next;
data=node->data;
delete node;
```

```
this->size--;
return data;
}
if(this->end==node)
{
end=p;
end->next=NULL;
data=node->data;
delete node;
this->size--;
return data;
}
p->next=node->next;
data=node->data;
delete node;
this->size--;
return data;
}
template<class T>
void SinglyLinkedList<T>::clear()
{
while(this->size>0) this->remove(0);
}
template<class T>
T SinglyLinkedList<T>::get(int index)
{
if(index<0 || index>=this->size) return 0;
SinglyLinkedListNode<T> *node;
node=this->start;
int x=0;
while(x<index)
{
node=node->next;
```

```
x++;
}
return node->data;
}
template<class T>
int SinglyLinkedList<T>::getSize()
{
return this->size;
}
template<class T>
SinglyLinkedList<T> * SinglyLinkedList<T>::getIterator()
{
return new SinglyLinkedListIterator<T>(this->start);
}
```