

Linear List

Data Structure
Dr. Bassim

```
#include <iostream.h>

enum bool {FALSE,TRUE};

class LIST
{
protected:
    unsigned int Size;
    unsigned int Count;
    int *Array;
    bool Shift(int pos);

public:
    LIST(unsigned int size=20);
    ~LIST(){delete []Array;}
    bool Empty();
    bool Full();
    bool InsertFront(const int &item);
    bool InsertRear(const int &item);
    bool Insert(const int &item, int pos);
    bool GetFront(int &item);
    bool GetRear(int &item);
    bool Get(int &item,int pos);
    void Print();
    void Clear(){Count = 0;}
    bool Update(const int &item, int pos);
    bool Remove(int &item, int pos);
};
```

```
void main()
```

```
{  
    int y;  
    LIST l;  
    for(int j=0; j<10;j++)  
        l.InsertFront(j);  
    l.Remove(y,7);  
    l.Print();  
    cout<<y<<endl;  
}
```

```
LIST::LIST(unsigned int size)
```

```
{  
    Size = size;  
    Count = 0;  
    Array = new int[size];  
}
```

```
bool LIST::Insert(const int &item, int pos )
```

```
{  
    if ( Shift(pos) )  
    {  
        Array[pos]= item;  
        Count ++;  
        return TRUE;  
    }  
    else  
        return FALSE;  
}
```

bool LIST::InsertFront(const int &item)

```
{  
    if (Shift(0) )  
    {  
        Array[0]= item;  
        Count ++;  
        return TRUE;  
    }  
    else  
        return FALSE;  
}
```

bool LIST::InsertRear(const int &item)

```
{  
    if (Shift(Count) )  
    {  
        Array[Count]= item;  
        Count ++;  
        return TRUE;  
    }  
    else  
        return FALSE;  
}
```

bool LIST::Shift(int pos)

```
{  
    if(pos>=0 && pos<=Count && pos<Size)  
    {  
        for(int i=Count-1; i>= pos ; i--)  
            Array[i+1] = Array[i];  
    }  
}
```

```

        return TRUE;
    }
    else
        return FALSE;
}

void LIST::Print()
{
    for(int i=0;i<Count; i++)
        cout<<Array[i]<<endl;
}

```

bool LIST::GetFront(int &item)

```

{
    if(Empty())
    {
        item=0;
        return FALSE;
    }
    else
    {
        item=Array[0];
        return TRUE;
    }
}

```

bool LIST::GetRear(int &item)

```

{
    if(Empty())
    {
        item=0;
    }
}

```

```
        return FALSE;
    }
else
    {
        item=Array[Count-1];
        return TRUE;
    }
}
```

bool LIST::Get(int &item, int pos)

```
{
    if(Empty() || pos>=Count || pos<0)
        {
            item=0;
            return FALSE;
        }
    else
        {
            item=Array[pos];
            return TRUE;
        }
}
```

bool LIST::Empty()

```
{
    if(Count==0)
        return TRUE;
    else
        return FALSE;
}
```

bool LIST::Full()

```
{  
    if(Count==Size)  
        return TRUE;  
    else  
        return FALSE;  
}
```

bool LIST::Update(const int &item, int pos)

```
{  
    if(pos>=0 && pos<Count)  
    {  
        Array[pos]=item;  
        return TRUE;  
    }  
    else  
        return FALSE;  
}
```

bool LIST::Remove(int &item, int pos)

```
{  
    item= Array[pos];  
    if(pos>=0 && pos<Count)  
    {  
        for(int i=pos; i<Count-1;i++)  
            Array[i]=Array[i+1];  
        Count--;  
        return TRUE;  
    }  
    else  
        return FALSE;  
}
```