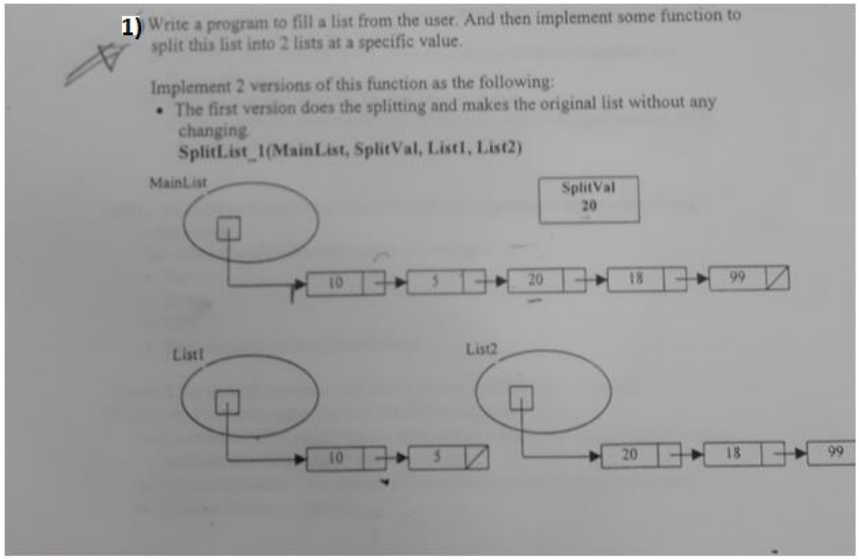
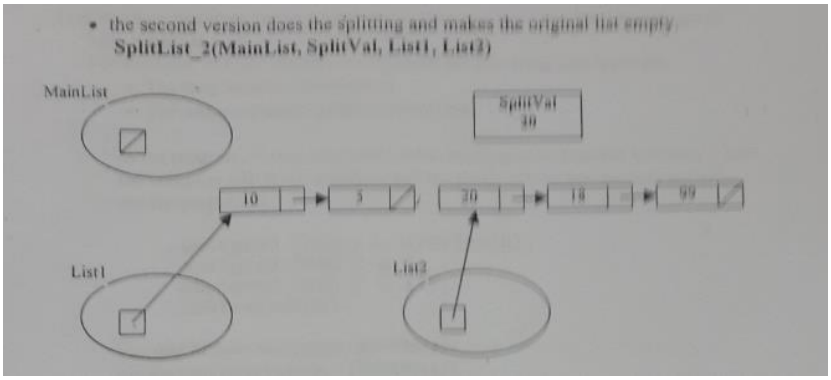



```
#include <iostream>
using namespace std;

class CNode
{
public:
	int info;
	CNode* pNext;
};

class CList
{
public:
	CNode* pHead;
	CNode* pTail;

	CList()
	{
		pHead = NULL;
		pTail = NULL;
	}

	void Attach(CNode* pnn)
	{
		if (pHead == NULL)
		{
			pHead = pnn;
			pTail = pnn;
		}
		else
		{
			pTail->pNext = pnn;
			pTail = pnn;
		}
	}

	~CList()
	{
```

```
		CNode* pTrav = pHead;
		while (pHead != NULL)
		{
			pHead = pTrav->pNext;
			pTrav->pNext = NULL;
			delete pTrav;
			pTrav = pHead;
		}
	}
};

void SplitList_1(CList MainL, int val, CList L1, CList L2)
{
	CNode* pTrav = MainL.pHead;
	CNode* pB = MainL.pHead;
	int ct = 0;

	while (pTrav->info != val)  //get node of target value(trav) & prev node(pB)
	{
		ct++;  //how many nodes to copy in L1
		pB = pTrav;
		pTrav = pTrav->pNext;

	}

	cout << "L1\n";
	CNode* pTrav2 = MainL.pHead;  //to traverse and copy info
	for (int i = 0; i < ct; i++)
	{
		CNode* pnn = new CNode;
		if (L1.pHead == NULL)
		{
			L1.pHead = pnn;
			L1.pTail = pnn;
			pnn->info = pTrav2->info;
			cout << pnn->info << " ";

		}
		else
		{
			L1.pTail->pNext = pnn;
			L1.pTail = pnn;
			pnn->info = pTrav2->info;
			cout << pnn->info << " ";


		}

		if (i == ct)
		{
			pnn->pNext = NULL;
		}

		pTrav2 = pTrav2->pNext;
	}


	ct = 0;
```

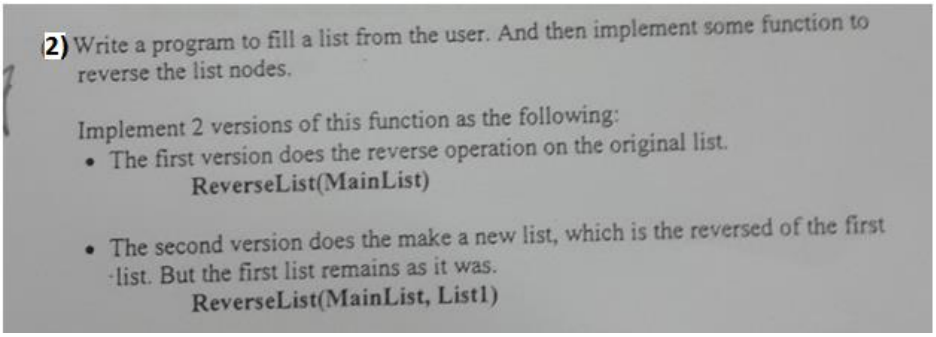
```
	while (pB->pNext != NULL)  //how many nodes to copy in L2
	{
		ct++;
		pB = pB->pNext;

	}

	cout << "L2\n";

	for (int i = 0; i <= ct; i++)
	{
		CNode* pnn = new CNode;
		if (L2.pHead == NULL)
		{
			L2.pHead = pnn;
			L2.pTail = pnn;
			pnn->info = pTrav->info;
			cout << pnn->info << " ";


		}
		else
		{
			L2.pTail->pNext = pnn;
			L2.pTail = pnn;
			pnn->info = pTrav->info;
			cout << pnn->info << " ";


		}

		if (i == ct)
		{
			pnn->pNext = NULL;
		}

		pTrav = pTrav->pNext;
	}



}

void main()
{
	CList MainL;
	CList L1;
	CList L2;
	int val;
	CNode* pnn;

	int N;
	cin >> N;

	for (int i = 0; i < N; i++)
	{
		pnn = new CNode;
```

```
		cout << "enter info\n";
		cin >> pnn->info;
		pnn->pNext = NULL;
		MainL.Attach(pnn);
	}



	cout << "enter val \n";
	cin >> val;

	SplitList_1(MainL, val, L1, L2);


}
```




```
void SplitList_2(CList& MainL, int val, CList& L1, CList& L2)
{
	CNode* pTrav = MainL.pHead;
	CNode* pB = MainL.pHead;


	while (pTrav->info != val)  //get node of target value(trav) & prev node(pB)
	{
		pB = pTrav;
		pTrav = pTrav->pNext;

	}

	L1.pHead = MainL.pHead;
	pB->pNext = NULL;
	L1.pTail = pB;

	L2.pHead = pTrav;
	L2.pTail = MainL.pTail;


	//empty the main list
	MainL.pHead = NULL;
	MainL.pTail = NULL;
```

```
	//output
	CNode* pOut = L1.pHead;
	cout << "L1\n";
	while (pOut != NULL)
	{
		cout << pOut->info << " ";
		pOut = pOut->pNext;
	}
	cout << "\n";
	pOut = L2.pHead;
	cout << "L2\n";
	while (pOut != NULL)
	{
		cout << pOut->info << " ";
		pOut = pOut->pNext;
	}

}
```



2) Write a program to fill a list from the user. And then implement some function to reverse the list nodes.

Implement 2 versions of this function as the following:

- The first version does the reverse operation on the original list.
  ReverseList(MainList)

- The second version does the make a new list, which is the reversed of the first list. But the first list remains as it was.
  ReverseList(MainList, List1)

VERSION ONE

```
#include <iostream>
using namespace std;

class CNode
{
public:
	int info;
	CNode* pNext;
};

class CList
{
public:
	CNode* pHead;
	CNode* pTail;

	CList()
	{
		pHead = NULL;
		pTail = NULL;
	}
```

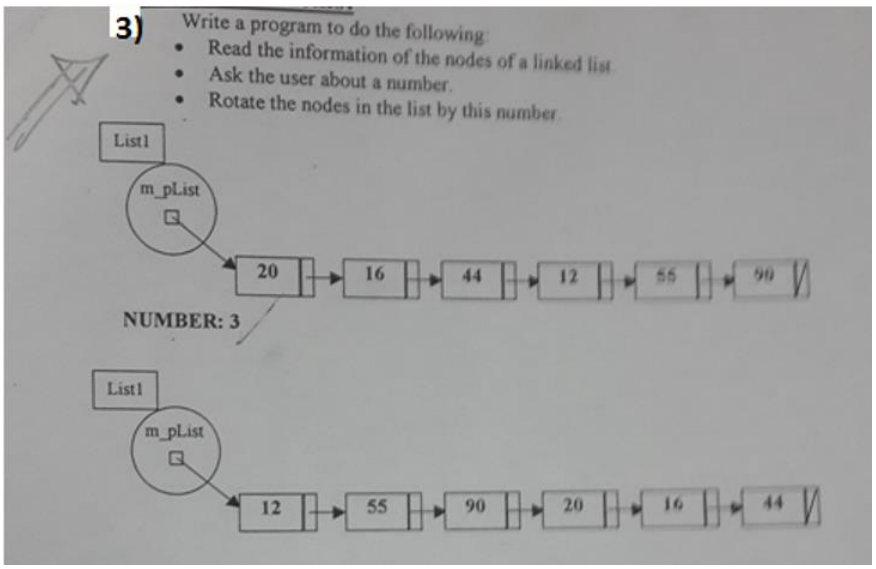
```
	void Attach(CNode* pnn)
	{
		if (pHead == NULL)
		{
			pHead = pnn;
			pTail = pnn;
		}
		else
		{
			pTail->pNext = pnn;
			pTail = pnn;
		}
	}

	~CList()
	{
		CNode* pTrav = pHead;
		while (pHead != NULL)
		{
			pHead = pTrav->pNext;
			pTrav->pNext = NULL;
			delete pTrav;
			pTrav = pHead;
		}
	}
};

void ReverseList(CList MainL)
{
	CNode* pB = MainL.pHead;    //current node to change arrow of
	CNode* pTrav = MainL.pHead;   //node after (next)
	CNode* pArrow = NULL;    //node to redirect the arrows(next) of nodes
	CNode* pLast = MainL.pTail;   //to store the pTail for reverse at the end

	while (pB != NULL)  //NULL is reaching after end of list
	{
		pTrav = pTrav->pNext;  //travserse
		pB->pNext = pArrow;
		pArrow = pB;   //redirects arrow to prev node
		pB = pTrav;
	}

	MainL.pTail = MainL.pHead;
	MainL.pHead = pLast;


	//output
	pTrav = MainL.pHead;
	while (pTrav != NULL)
	{
		cout << pTrav->info << " ";
		pTrav = pTrav->pNext;
	}
}

void main()
{
	CList MainL;
```

```
	CNode* pnn;

	cout << "Enter N \n";
	int N;
	cin >> N;

	for (int i = 0; i < N; i++)
	{
		pnn = new CNode;
		cout << "enter info\n";
		cin >> pnn->info;
		pnn->pNext = NULL;
		MainL.Attach(pnn);
	}

	ReverseList(MainL);

}
```

VERSION TWO

```
void ReverseList(CList MainL, CList L)
{
	CNode* pTrav = MainL.pHead;
	CNode* pB = L.pTail;

	while (pTrav != NULL)
	{
		CNode* pnn = new CNode;

		if (L.pHead == NULL)
		{
			pnn->info = pTrav->info;
			L.pHead = pnn;
			L.pTail = pnn;
			pB = L.pTail; //to keep track of the node that each new node
will point to

		}
		else
		{
			pnn->info = pTrav->info;
			L.pHead = pnn;
			pnn->pNext = pB;
			pB = pnn;
		}

		pTrav = pTrav->pNext;
	}


	//output
	pTrav = L.pHead;
	while (pTrav != NULL)
	{
		cout << pTrav->info << " ";
```

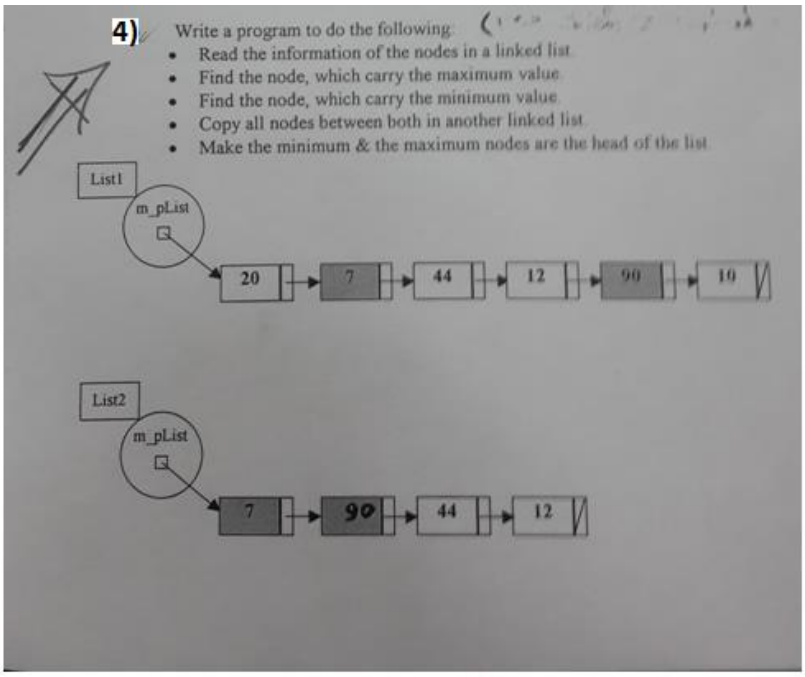
```
            pTrav = pTrav->pNext;
        }

}
```




```
#include <iostream>
using namespace std;

class CNode
{
public:
	int info;
	CNode* pNext;
};

class CList
{
public:
	CNode* pHead;
	CNode* pTail;

	CList()
	{
		pHead = NULL;
		pTail = NULL;
	}

	void Attach(CNode* pnn)
	{
		if (pHead == NULL)
		{
			pHead = pnn;
			pTail = pnn;
		}
		else
		{
			pTail->pNext = pnn;
			pTail = pnn;
		}
	}
```

```
	~CList()
	{
		CNode* pTrav = pHead;
		while (pHead != NULL)
		{
			pHead = pTrav->pNext;
			pTrav->pNext = NULL;
			delete pTrav;
			pTrav = pHead;
		}
	}
};


void main()
{
	CList L1;
	CNode* pnn;
	int N, num, ct=0;


	cout << "Enter N \n";
	cin >> N;

	for (int i = 0; i < N; i++)
	{
		pnn = new CNode;
		cout << "enter info\n";
		cin >> pnn->info;
		pnn->pNext = NULL;
		L1.Attach(pnn);
	}

	cout << "enter num\n";
	cin >> num;

	CNode* pTrav = L1.pHead;  //to point at target node
	CNode* pB = L1.pHead;   //to point at node before target


	while (pTrav != NULL)
	{
		if (ct == num)
		{
			pB->pNext = NULL;   //to make it last node
			L1.pTail->pNext = L1.pHead;   //to make last point to start node
			L1.pHead = pTrav;   //head points at target
			L1.pTail = pB;   //tail points at node before target
			break;

		}

		ct++;
		pB = pTrav;
		pTrav = pTrav->pNext;
	}
```

```
	//output
	pTrav = L1.pHead;
	while (pTrav != NULL)
	{
		cout << pTrav->info << " ";
		pTrav = pTrav->pNext;
	}

}
```




```
#include <iostream>
using namespace std;

class CNode
{
public:
	int info;
	CNode* pNext;
};

class CList
{
public:
	CNode* pHead;
	CNode* pTail;

	CList()
	{
		pHead = NULL;
		pTail = NULL;
	}
```

```
	void Attach(CNode* pnn)
	{
		if (pHead == NULL)
		{
			pHead = pnn;
			pTail = pnn;
		}
		else
		{
			pTail->pNext = pnn;
			pTail = pnn;
		}
	}

	~CList()
	{
		CNode* pTrav = pHead;
		while (pHead != NULL)
		{
			pHead = pTrav->pNext;
			pTrav->pNext = NULL;
			delete pTrav;
			pTrav = pHead;
		}
	}
};


void main()
{
	CList L1;
	CList L2;
	CNode* pnn;

	cout << "Enter N \n";
	int N;
	cin >> N;

	for (int i = 0; i < N; i++)
	{
		pnn = new CNode;
		cout << "enter info\n";
		cin >> pnn->info;
		pnn->pNext = NULL;
		L1.Attach(pnn);
	}

	CNode* pTrav = L1.pHead;
	CNode* pMin = L1.pHead;
	CNode* pMax = L1.pHead;
	int min = 9999, max = -9999, i=0,imin=0, imax=0;

	while (pTrav != NULL)
	{
		if (pTrav->info < min)
		{
			min = pTrav->info;
```

```
			pMin = pTrav;
			imin = i;
		}
		if (pTrav->info > max)
		{
			max = pTrav->info;
			pMax = pTrav;
			imax = i;
		}

		pTrav = pTrav->pNext;
		i++;
	}

	for (int i = 0; i < 2; i++) //add min and max into new list
	{
		CNode* pnn = new CNode;

		if (L2.pHead == NULL)
		{
			pnn->info = min;
			L2.pHead = pnn;
			L2.pTail = pnn;
		}
		else
		{
			pnn->info = max;
			L2.pTail->pNext = pnn;
			L2.pTail = pnn;
		}
	}

	if (imin < imax)
	{
		pTrav = pMin->pNext;

		while (pTrav != NULL && pTrav != pMax)
		{
			CNode* pnn = new CNode;
			pnn->info = pTrav->info;
			L2.pTail->pNext = pnn;
			L2.pTail = pnn;

			pTrav = pTrav->pNext;
		}
	}
	else
	{
		pTrav = pMax->pNext;

		while (pTrav != NULL && pTrav != pMin)
		{
			CNode* pnn = new CNode;
			pnn->info = pTrav->info;
			L2.pTail->pNext = pnn;
			L2.pTail = pnn;

			pTrav = pTrav->pNext;
```

```
		}
	}

	//output
	pTrav = L2.pHead;
	while (pTrav != NULL)
	{
		cout << pTrav->info << " ";
		pTrav = pTrav->pNext;
	}


}
```