```
#define _CRT_SECURE_NO_WARNINGS
#include <iostream>
#include <Windows.h>
#include "KiemTraGiuaKy1.h"
using namespace std;
/******* Node ********/
//Phương thức thiết lập
Node::Node()
{
	info = 0;
	next = NULL;
}
//Tạo node chứa giá trị x
Node* createNode(int x)
{
	Node* p = new Node;

	if (p == NULL)
	{
		cout << "Not enough memory to
allocate!";
		return NULL;
	}

	p->info = x;
	p->next = NULL;

	return p;
}
//Xóa node
void deleteNode(Node*& p)
{
	if (p == NULL)  return;

	p->next = NULL;
	delete p;
```

```
}
//Xuất nội dung nút
void showNode(Node* p)
{
	cout << p->info;
}

/***** SList ******/
//a. Khởi tạo DSLK
void init(SList& l)
{
	l.head = NULL;
	l.tail = NULL;
}
//b. Kiểm tra danh sách rỗng
bool isEmpty(SList l)
{
	return l.head == NULL;
}
//c. Duyệt danh sách để xuất nội dung ra màn
hình
void traverse(SList l)
{
	if (isEmpty(l))
	{
		cout << "List is empty\n";
		return;
	}

	for (Node* p = l.head; p != NULL; p =
p->next)
		cout << p->info << '\t';
}
//d. Thêm nút p có giá trị x vào đầu danh sách
void addHeadSList(SList& l, Node* p)
{
```

```
        if (p == NULL)  return;

        if (isEmpty(l))
                l.head = l.tail = p;
        else
        {
                p->next = l.head;
                l.head = p;
        }
}
//e, 1a. Thêm nút p có giá trị x vào cuối danh
sách
void addTailSList(SList& l, Node* p)
{
        if (p == NULL) return;

        if (isEmpty(l))
                l.head = l.tail = p;
        else
        {
                l.tail->next = p;
                l.tail = p;
        }
}
//f. Thêm nút p có giá trị x vào sau nút q có
giá trị y của danh sách
void addAfterNodeSList(SList& l, Node* q,
Node* p)
{
        if (p == NULL)  return;
        else
        {
                if (q == NULL)
                {
                        if (l.head ==
NULL)//list is empty
```

```
                                        l.head =
l.tail = p;
                                else//cannot find node
q in list
                                        return;
                        }
                        else
                        {
                                if (q == l.tail)
                                        addTailSList(l, p);
                                else
                                {
                                        p->next =
q->next;
                                        q->next = p;
                                }
                        }
                }
}
//g. Tìm node có giá trị x trong SList
Node* searchSList(SList l, int x)
{
        for (Node* q = l.head; q != NULL; q =
q->next)
                if (q->info == x)
                        return q;
        return NULL;
}
//h. Tạo danh sách nhập giá trị từ bàn phím
void createSList(SList& l)
{
        int n, x;
        init(l);
        do
        {
```

```
                cout << "Cho biết số phần tử
của danh sách (n > 0): ";
                cin >> n;
        } while (n <= 0);
        for (int i = 1; i <= n; i++)
        {
                cout << "Nhập phần tử thứ %d
là: " << i;
                cin >> x;
                Node* p = createNode(x);
                addTailSList(l, p);
                addHeadSList(l, p);
        }
}
//Khởi tạo danh sách có n phần tử
void createSList(SList& l, int n)
{
        int x;
        Node* q;
        l.head = l.tail = NULL;//can call init
(l);
        for (int i = 1; i <= n; i++)
        {
                cout << "Input the element
%d:" << i;
                cin >> x;
                q = createNode(x);

                if (q == NULL)
                {
                        printf("Not enough
memory to allocate!");
                        return;
                }

                //link q into list l
```

```
                if (l.head == NULL)
                        l.head = l.tail = q;
                else
                {
                        l.tail->next = q;
                        l.tail = q;
                }
        }
}
//Delete the head node of SList
void deleteHeadSList(SList& l)
{
        if (l.head == NULL)
                return;
        else
        {
                Node* p = l.head;
                l.head = p->next;
                p->next = NULL;
                delete p;
        }
}
//Delete the tail node of SList
void deleteTailList(SList& l)
{
        if (l.head == NULL)
                return;
        else
        {
                Node* p = l.tail;
                Node* q = l.head;
                if (p == q)//list has 1
element
                {
                        l.head = l.tail =
NULL;
```

```
                        delete p;
                }
                else
                {
                        //find the node
located right before tail
                        while (q->next !=
l.tail)
                                q = q->next;
                        l.tail = q;
                        l.tail->next = NULL;
                        delete p;
                }
        }
}
//Delete node p after node q of SList
void deleteMidSList(SList& l, Node* q)
{
        if (l.head == NULL || q == NULL || q
== l.tail)//list is empty or q is NULL or list
has 1 element
                return;
        else
        {
                if (q->next == l.tail)
                        deleteTailList(l);
                else
                {
                        Node* p = q->next;
                        q->next = p->next;
                        p->next = NULL;
                        delete p;
                }
        }
}
////////////////////////////////////////////////
```

```
////////
SNode* popS(Stack& s)
{
        SNode* p = NULL;
        if (s.top != NULL)
        {
                p = s.top;
                s.top = s.top->next;
                p->next = NULL;
        }
        return p;
}
void pushS(Stack& s, SNode* p)
{
        if (p != NULL)
                if (s.top == NULL) s.top = p;
                else
                {
                        p->next = s.top;
                        s.top = p;
                }
}
```