ЛАБОРАТОРНАЯ РАБОТА 4.

Шаблоны классов.
Использование шаблонного класса MyStack для хранения простых множителей целых чисел

Задание. Дано описание класса MyStack (Приложение 1, файл MyStack.h), который реализует на основе односвязного списка динамическую структуру данных типа стек.

1. Разработайте реализацию интерфейса класса в виде файла MyStack.cpp.

2. Разработайте функцию (глобальную), которая выполняет разложение на простые множители целого числа N. Для хранения множителей функция должна использовать класс MyStack. Прототип функции: void Multipliers(int n, MyStack<DATA> &stack).

3. В функции main( ) распечатайте множители, которые функция Multipliers( ) записывает в стек, сначала по убыванию, а потом по возрастанию. Например, для N=3960 программа должна вывести:

3960=11 * 5 * 3 * 3 * 2 * 2 * 2
3960=2 * 2 * 2 * 3 * 3 * 5 * 11

Нельзя изменять описание класса, приведенное в файле MyStack.h.

Пояснения.

Стек представляет собой динамическую структуру (то-есть структуру, размер которой может изменяться в процессе выполнения программы), предназначенную для временного хранения данных некоторого типа INF, который может быть как базовым, так и определяемым пользователем. Запись данных в стек и выборка данных из стека производятся путем обращения к его вершине.

В данной работе стек реализуется на базе односвязного списка. При записи в стек (операция PUSH) в начало списка добавляется узел, а при выборке из стека (операция POP) узел удаляется из начала списка. Чтобы получить доступ к следующей ячейке стека нужно удалить предыдущую. При выполнении операции POP данные из стека не считываются. Данные могут считываться только из вершины стека. Для чтения данных используется специальная функция, которая читает данные без удаления узла из вершины стека.

Для обеспечения доступа к данным, хранящимся в узлах типа ListNode, класс MyStack сделан дружественным по отношению к классу ListNode. В этом случае все методы класса MyStack получают доступ к скрытым данным класса ListNode.

Чтобы в узлах можно было бы хранить данные различных типов и чтобы узлы класса ListNode можно было бы использовать в различных структурах (например, для реализации списка или очереди), класс ListNode реализован в виде шаблона семейства классов с двумя формальными параметрами: типом хранимых данных (class INF) и дружественным классом, реализующим некоторую структуру данных, например стек (class FRIEND).

Методы шаблонного класса не должны зависеть от значений формальных параметров и должны быть одинаковыми для всех типов хранимых данных и дружественных классов. Для реализации этого требования в лабораторной работе класс ListNode сделан закрытым (то-есть в нем нет методов в разделе public:, хотя могли бы и быть), а доступ к его элементам осуществляется через интерфейс дружественного класса FRIEND, являющегося одним из двух формальных параметров класса ListNode (в данной работе это MyStack<INF>).

Указания:

1. Пример внешнего определения методов шаблонного класса:

```
template <class INF>
bool MyStack<INF>::empty(void)
{
if(top==NULL)
    return true;
else
    return false;
}
```

2. Определение (переименование) *типа* узла ListNode, хранящего данные класса INF и использующего в качестве дружественного класса FRIEND шаблонный класс MyStack<INF> :

typedef class ListNode < INF, MyStack <INF> > Node;

Теперь вместо **class ListNode < INF, MyStack <INF> >** он будет называться просто **Node**.

Требования к отчету. Отчет должен содержать следующие разделы:.

- «Постановка задачи», в котором на основании задания уточняются задачи, для решения которых предполагается использовать разрабатываемый класс.
- «Разработка алгоритма», в котором должна быть приведена блок-схема алгоритма разложения целого числа на простые множители.
- «Текст программы», в котором приведены исходные тексты разработанной программы. При защите лабораторной работы студент должен уметь объяснить назначение каждого оператора разработанной им программы.
- «Анализ результатов», в котором приводятся разработка контрольных примеров, распечатки результатов выполнения программой контрольных примеров и анализ результатов.

Приложение 1.

```
//файл MyStack.h
//Шаблонный класс MyStack на основе односвязного списка.
#ifndef MyStack_h                    // защита от повторной компиляции
#define MyStack_h                    // модуль подключен
#include "Data.h"

//Шаблонный класс ListNode (узел односвязного списка)
template <class INF,class FRIEND>
class ListNode                              //узел списка
{
private:
        INF d;                       //информационная часть узла
        ListNode* next;               //указатель на следующий узел списка
        ListNode(void) { next = NULL; } //конструктор
        friend FRIEND;
        };

//Шаблонный класс MyStack на основе односвязного списка.
template <class INF >
class MyStack
{
        typedef class ListNode < INF, MyStack <INF> > Node;
        Node* top;
public:
        MyStack(void);               // конструктор
        ~MyStack(void);              // освободить динамическую память
        bool empty(void);            // стек пустой?
        bool push(INF n);            // добавить узел в вершину стека
        bool pop(void);              // удалить узел из вершины стека
        INF* top_inf(void);          //считать информацию из вершины стека
};


#endif
```