Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Тверской государственный университет»

Факультет прикладной математики и кибернетики

Направление: 01.03.02 Прикладная математика и информатика

ОТЧЕТ ПО УЧЕБНО-ВЫЧИСЛИТЕЛЬНОЙ ПРАКТИКЕ

Выполнила:
студентка 1 курса
14 группы
Савинова Валентина
Константиновна
Проверил:
доцент кафедры информатики
Дадеркин Дмитрий Ольгердович

Тверь — 2015

Оглавление

Постановка задачи

Вариант №13.

1. Разработать программу с удобным пользовательским интерфейсом, реализующую следующие функции:

   Синтаксический и семантический анализ понятия список геометрических фигур

   $$\text{СГФ} ::= \left\{ \begin{array}{c} \text{геометрическая фигура} \\ \text{геометрическая фигура } \{; \text{ список геометрических фигур}\} \end{array} \right\}$$

   $$\text{геометрическая фигура} ::= \left\{ \begin{array}{c} \text{прямоугольник} \\ \text{трапеция} \end{array} \right\}$$

   $$\text{прямоугольник} ::= \left\{ \text{ вершина; вершина; вершина; вершина } \right\}$$

   $$\text{трапеция} ::= \left\{ \text{ вершина; вершина; вершина; вершина } \right\}$$

   $$\text{вершина} ::= \left\{ \text{ абсцисса, ордината } \right\}$$

   $$\text{абсцисса} ::= \left\{ \text{ число с точкой } \right\}$$

   $$\text{ордината} ::= \left\{ \text{ число с точкой } \right\}$$

   $$\text{число с точкой} ::= \left\{ \left\{ \begin{array}{c} \text{пробел} \\ \pm \end{array} \right\} \text{целое без знака.целое без знака} \right\}$$

   $$\text{целое без знака} ::= \left\{ \begin{array}{c} \text{цифра} \\ \text{цифра } \{\text{целое без знака}\} \end{array} \right\}$$

   В случае отсутствия синтаксических ошибок, вычисление площадей этих фигур и их покоординатное построение на экране.

2. Создать класс «матрица n, m», где - n,m - количество строк и столбцов в матрице. Каждый элемент, кроме элементов в крайних строках и столбцах, должен быть связан с соседними (звено 4-х связного списка). Элемент в позиции [i,j] (гарантируется, что он расположен не в крайних строках и столбцах матрицы) задает 4 квадрата, расположенных, соответственно, выше и левее, выше и правее, ниже и левее, ниже и правее этого элемента. Найти квадрат с максимальной суммой своих элементов.

Задание 1.

Описание программы

Программа включает в себя следующие классы «целое без знака», «число с точкой», «вершина», «геометрическая фигура» и «список геометрических фигур», а так же простой пользовательский интерфейс.
Входные данные считываются из файла, выбранного пользователем. В ходе работы программы осуществляется синтаксический и семантический анализ введенных данных.
В случае наличия ошибок во входных данных, на экран выводится окно с сообщением об ошибке, и пользователю предлагается повторить выбор файла с входными данными, прервать работу программы или закрыть окно.
В случае отсутствия ошибок, покоординатное построение фигур из списка на экране и вывод площадей фигур в информационном окне.После построения каждой из фигур, область изображения фигур очищается.

Описание классов

- Класс $UInt$: Реализует целое без знака число. Включает в себя одну переменную $value$ типа $integer$, отвечающую за хранение целого без знака числа.
  Класс включает в себя конструктор присваивающий переменной $value$ значение 0, и деструктор по умолчанию.
  А также, функцию $getValue$, которая позволяет получить значение целого без знака числа,
  функцию $digits$, которая возвращает количество цифр в целом без знака числе,
  функцию $isStopSymbol$ проверяющую, является ли символ, символом разделителем или пустым символом,
  и функцию $operator >>$, которая позволяет считывать целое без знака число.
- Класс $Number$: Реализует число с точкой. Включает в себя переменную $sign$ типа

integer, отвечающую за хранение знака числа с точкой, и переменные *integer* и $fraction$ типа $UInt$, отвечающие за хранение целой и дробной части числа с точкой соответственно.

Класс включает в себя конструктор присваивающий переменной *sign* значение 1, и деструктор по умолчанию.

А также, функцию *toDouble*, которая позволяет получить значение числа с точкой, и функцию *operator* >>, которая позволяет считывать число с точкой.

- Класс *top*: Реализует вершину. Включает в себя переменные *abscissa* и *ordinate* типа $Number$, отвечающие за хранение абсциссы и ординаты вершины.

  Класс включает в себя конструктор и деструктор по умолчанию. А также, функцию *getx*, позволяющую получить значение абсциссы вершины,
  функцию *gety*, позволяющую получить значение ординаты вершины,
  и функцию *operator* >>, которая позволяет считывать вершину.

- Класс $GFigure$: Реализует геометрическую фигуру. Включает в себя переменные a, b, c, d типа *top*, отвечающие за хранение вершин четырехугольника.

  Класс включает в себя конструктор и деструктор по умолчанию. А также, функцию *geta*, позволяющую получить координаты вершины a,
  функцию *getb*, позволяющую получить координаты вершины b,
  функцию *getc*, позволяющую получить координаты вершины c,
  функцию *getd*, позволяющую получить координаты вершины d,
  функцию *area*, которая считает площадь фигуры,
  функцию $ParallelLines$, которая определяет параллельность 2-х прямых,
  функцию $identification$, которая определяет принадлежность фигуры списку,
  функцию $areEqual$, которая позволяет проверить на равенство 2 числа типа *double*,
  функцию $scalarProduct$, которая считает скалярное произведение 2-х векторов,
  функцию *length*, которая возвращает длинну отрезка между 2-мя точками,
  функцию *orthogonal*, которая проверяет 2 вектора на ортогональность,
  и функцию *operator* >>, которая позволяет считывать геометрическую фигуру.

- Класс $ListGF$: Реализует список геометрических фигур. Включает в себя указатель *top* типа $Node*$.

  Структура $Node$ (узел односвязного списка) в которую входит переменная *value* типа $GFigure$ и указатель *next* типа $Node*$.

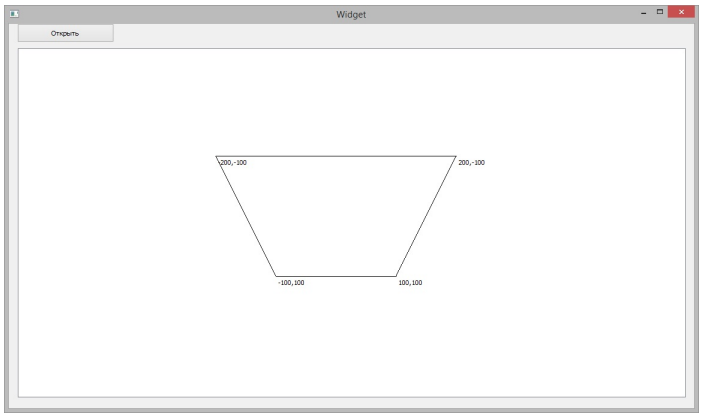
  Класс включает в себя конструкторы и деструктор. А также, функцию $addToEnd$, которая позволяет добавлять элементы в конец списка,

функцию *addToBegining*, которая позволяет добавлять элементы в начало списка, функцию *getValue*, которая позволяет получить по номеру в списке значение элемента, функцию *identification*, которая проверяет принадлежность фигур списку, функию *getCount*, возвращающую количество элементов в списке и функцию *operator >>*, которая позволяет считывать список геометрических фигур.

- Класс *Widget*: Реализует пользовательский интерфейс. Включает в себя конструктор и деструктор. А также, функцию *drawF*, которая позволяет покоординатно построить геометрическую фигуру и функцию *onOpenClicked*, которая позволяет считать, из выбранного пользователем файла, список геометрических фигур, выводит инфомационные сообщения и сообщения об ошибке.

Тестирование работы программы

При запуске программы с входными данными: -100.0,100.0;100.0,100.0;200.0,-100.0;-200.0,-100.0 На экране мы видим:



Исходный текст программы

```
#ifndef UINT_H
#define UINT_H
#include<iostream>
using namespace std;


class UInt
{
```

```
private:
    int value;
public:
    UInt();
    ~UInt();
    int getValue();
    friend istream& operator >> (istream&, UInt&);
    int digits() const;

};

#endif // UINT_H
```

```
#include "uint.h"

UInt::UInt()
{
    value = 0;
}

UInt::~UInt()
{

}
int UInt::getValue()
{
    return value;
}
bool isStopSymbol(char c)
{
    return c == ',' || c == '.' || c == ';' || isspace(c);
}

istream& operator >> (istream& in, UInt& x)
{
```

```
    char c = in.get();
    if(!isdigit(c))
    {
        throw "Undexpected input";
    }
    x.value = c-'0';
    c = in.peek();
    while (!isStopSymbol(c))
    {
        if (isdigit(c))
        {
            x.value*=10;
            x.value+=c-'0';
            in.get();
            c = in.peek();
        }
        else
        {
             throw "Undexpected input";
        }
    }
    return in;
}
int UInt:: digits() const
{
    if (this->value==0)
    {
        return 1;
    }
    int count = 0;
    int v = value;
    while (v!=0)
    {
        v/=10;
        count++;
```

```
    }
    return count;
}

#ifndef NUMBER_H
#define NUMBER_H
#include <uint.h>


class Number
{
private:
    int sign;
    UInt integer;
    UInt fraction;
public:
    Number();
    ~Number();
    friend istream& operator >> (istream&, Number&);
    double toDouble ();
};

#endif // NUMBER_H

#include "number.h"

Number::Number()
{
    sign = 1;
}

Number::~Number()
{

}
istream& operator >> (istream& in, Number& x)
```

```
{
    char c = in.peek();
    if (c=='+'||c==' ')
    {
        x.sign = 1;
        in.get();
    }
    else if (c=='-')
    {
        x.sign = -1;
        in.get();
    }
    else if(!isdigit(c))
    {
        throw "Unexpected input";
    }
    in>>x.integer;
    c = in.get();
    if (c!='.')
        throw "Unexpected input";
    in>>x.fraction;
    return in;
}

double Number::toDouble()
{
    double res = integer.getValue();
    double fract = fraction.getValue();
    for(int i=0; i<fraction.digits();i++)
    {
        fract/=10;
    }
    return (res+fract)*sign;

}
```

```
#ifndef TOP_H
#define TOP_H
#include <number.h>


class top
{
private:
    Number abscissa;
    Number ordinate;
public:
    top();
    ~top();
    friend istream& operator >> (istream&, top&);
    double getx();
    double gety();
};

#endif // TOP_H
```

```
#include "top.h"

top::top()
{

}

top::~top()
{

}
istream& operator >> (istream& in, top& v)
{
    in>>v.abscissa;
    char c = in.peek();
```

```
    while(isspace(c))
    {
        in.get();
        c = in.peek();
    }
    if (c!=',')
        throw "Unexpected input";
    c = in.get();
    while(isspace(c))
    {
        in.get();
        c = in.peek();
    }
    in>>v.ordinate;
    return in;
}
double top::getx()
{
    return abscissa.toDouble();
}
double top::gety()
{
    return ordinate.toDouble();
}

#ifndef GFIGURE_H
#define GFIGURE_H
#include <top.h>
#include <math.h>


class GFigure
{
private:
    top a;
```

```
    top b;
    top c;
    top d;
public:
    GFigure();
    ~GFigure();
    friend istream& operator >> (istream&, GFigure&);
    bool ParallelLines (top point1, top point2,
                top point3, top point4);
    bool identification ();
    double area ();
    top geta();
    top getb();
    top getc();
    top getd();
};

#endif // GFIGURE_H

#include "gfigure.h"

GFigure::GFigure()
{

}

GFigure::~GFigure()
{

}
istream& operator >> (istream& in, GFigure& f)
{
    in>>f.a;
    char c = in.peek();
    if (c!=';')
```

```
        throw "Unexpected input";
    c = in.get();
    in>>f.b;
    c = in.peek();
    if (c!=';')
        throw "Unexpected input";
    c = in.get();
    in>>f.c;
    c = in.peek();
    if (c!=';')
        throw "Unexpected input";
    c = in.get();
    in>>f.d;
    return in;
}

bool areEqual(double x, double y)
{
  static double SMALL_NUM = 1.0E-5;
  if (fabs(x) < SMALL_NUM && fabs(y) < SMALL_NUM)
    return fabs(x-y) < SMALL_NUM;
  else
    return fabs(x-y) < fabs(x) * SMALL_NUM;
}

double scalarProduct(top point1, top point2,
                     top point3, top point4)
{
    double v1 = point2.getx()-point1.getx();
    double v2 = point2.gety()-point1.gety();
    double u1 = point4.getx()-point3.getx();
    double u2 = point4.gety()-point3.gety();
    return v1*u1+v2*u2;
}
```

```
double length(top point1, top point2)
{
    return sqrt(scalarProduct(point1, point2, point1, point2));
}

bool GFigure::ParallelLines(top point1, top point2,
                            top point3, top point4)
{
    double product = scalarProduct(point1, point2,
                                   point3, point4);
    double l1 = length(point1,point2);
    double l2 = length(point3, point4);
    if(areEqual(l1,0)||areEqual(l2,0))
        throw "Invalid data";
    double absCos = fabs(product/(l1*l2));
    return areEqual(absCos,1);
}

bool orthogonal(top point1, top point2,
                               top point3, top point4)
{
    return areEqual(0,scalarProduct(point1,point2,
                            point3, point4));
}

bool GFigure:: identification()
{
    if(ParallelLines(a,b,c,d))
    {
        if(!ParallelLines(a,d,b,c))
        {
            return true;
        }
        else
        {
```

```
        if (a.getx()<b.getx()&&d.getx()<c.getx())
        {
            return orthogonal(a,d,a,b);
        }
        else if (a.getx()>b.getx()&&d.getx()>c.getx())
        {
            return orthogonal(a,d,a,b);
        }
        else if (a.getx()==b.getx()&&d.getx()==c.getx())
        {
            if (a.gety()<b.gety()&&d.gety()<c.gety())
            {
                return orthogonal(a,d,a,b);
            }
            else if (a.gety()>b.gety()&&d.gety()>c.gety())
            {
                return orthogonal(a,d,a,b);
            }
        }
        return false;
    }
}
else if(ParallelLines(a,d,b,c))
{
    if(!ParallelLines(a,b,c,d))
    {
        return true;
    }
    else
    {

        if (a.getx()<d.getx()&&b.getx()<c.getx())
        {
            return orthogonal(a,d,a,b);
        }
```

```
            else if (a.getx()>d.getx()&&b.getx()>c.getx())
            {
                return orthogonal(a,d,a,b);
            }
            else if (a.getx()==d.getx()&&b.getx()==c.getx())
            {
                if (a.gety()<d.gety()&&b.gety()<c.gety())
                {
                    return orthogonal(a,d,a,b);
                }
                else if (a.gety()>d.gety()&&b.gety()>c.gety())
                {
                    return orthogonal(a,d,a,b);
                }
            }
            return false;
        }
    }
    else return false;

}
top GFigure::geta()
{
    return this->a;
}
top GFigure::getb()
{
    return this->b;
}
top GFigure::getc()
{
    return this->c;
}
top GFigure::getd()
{
```

```
    return this->d;
}
double GFigure::area()
{
    double p = (length(a,b)+length(b,c)
                +length(c,d)+length(d,a))/2;
    double S = sqrt((p-length(a,b))*(p-length(b,c))
                *(p-length(c,d))*(p-length(d,a)));
    return S;
}

#ifndef LISTGF_H
#define LISTGF_H
#include<gfigure.h>

struct Node
{
    GFigure value;
    Node*next;
    Node(const GFigure value)
    {
        this->value = value;
        next = NULL;
    }
};

class ListGF
{
private:
    Node* top;
public:
    ListGF();
    ~ListGF();
    ListGF(const ListGF &other);
    void addToEnd(const GFigure n);
```

```
    void addToBegining (const GFigure n);
    friend istream& operator >> (istream& , ListGF&);
    GFigure getValue (int n);
    int getCount();
    void identification ();
};

#endif // LISTGF_H
```

```
#include "listgf.h"

ListGF::ListGF()
{
    top = 0;
}

ListGF::~ListGF()
{
    while(top!=NULL)
    {
        Node*temp = top;
        top = top->next;
        delete temp;
    }
}
ListGF::ListGF(const ListGF &other)
{
    Node*temp = other.top;
    while (temp!=NULL)
    {
        addToEnd(temp->value);
        temp = temp->next;
    }
}
void ListGF:: addToEnd(const GFigure n)
```

```
{
    if (top == NULL)
    {
        top = new Node(n);
    }
    else
    {
        Node* temp = top;
        while(temp->next!= NULL)
        {
            temp = temp->next;
        }
        temp->next = new Node(n);
    }
}
void ListGF::addToBegining(const GFigure n)
{
    Node* node = new Node(n);
    node->next = top;
    top = node;
}

istream& operator >> (istream& in, ListGF& f)
{
    while(f.top!=0)
    {
        Node*temp = f.top;
        f.top = f.top->next;
        delete temp;
    }
    f.top = NULL;
    GFigure figure;
    char c  = in.peek();
    if(c == -1)
    {
```

```
            throw "Unexpected input";
        }
        while (c!=-1)
        {
            in>>figure;
            f.addToEnd(figure);
            c = in.peek();
            if (c==';')
            {
                c = in.get();
                c = in.peek();
            }

            while(isspace(c))
            {
                in.get();
                c = in.peek();
            }
        }

        return in;
}
GFigure ListGF::getValue(int n)
{
        Node* temp = top;
        for (int i=0; i<n; i++)
        {
            temp = temp->next;
        }
        return temp->value;
}
int ListGF::getCount()
{
        Node* temp = top;
        int count = 0;
```

```
    while (temp!=NULL)
    {
        temp = temp->next;
        count++;
    }
    return count;
}
void ListGF::identification()
{
    for (int i = 0; i<this->getCount(); i++)
    {
        bool b = this->getValue(i).identification();
        if (!b)
        {
            throw "invalid figure";
        }
    }
}

#ifndef WIDGET_H
#define WIDGET_H

#include <QWidget>
#include <QMessageBox>
#include <QFileDialog>
#include <iostream>
#include <fstream>
#include <listgf.h>
#include<QGraphicsTextItem>

namespace Ui {
class Widget;
}

using namespace std;
```

```
class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    void drawF (GFigure);
    ~Widget();

private:
    Ui::Widget *ui;

private slots:
    void onOpenClicked();
};

#endif // WIDGET_H
```

```
#include "widget.h"
#include "ui_widget.h"

Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    connect(ui->pushButton, SIGNAL(clicked()),
            this, SLOT(onOpenClicked()));
    ui->graphicsView->setScene(new QGraphicsScene());
}

Widget::~Widget()
{
    delete ui;
}
```

```
void Widget::drawF(GFigure f)
{

    QGraphicsScene* scene = ui->graphicsView->scene();
    double ax,ay,bx,by,cx,cy,dx,dy;
    ax = f.geta().getx();
    ay = f.geta().gety();
    bx = f.getb().getx();
    by = f.getb().gety();
    cx = f.getc().getx();
    cy = f.getc().gety();
    dx = f.getd().getx();
    dy = f.getd().gety();
    scene->addLine(ax, ay, bx, by);
    QGraphicsTextItem * lbl1 = new QGraphicsTextItem;
    lbl1->setPos(ax,ay);
    lbl1->setPlainText(QString::number(ax)
                       +","+QString::number(ay));
    scene->addItem(lbl1);
    scene->addLine(bx, by, cx, cy);
    QGraphicsTextItem * lbl2 = new QGraphicsTextItem;
    lbl2->setPos(bx,by);
    lbl2->setPlainText(QString::number(bx)
                       +","+QString::number(by));
    scene->addItem(lbl2);
    scene->addLine(cx, cy, dx, dy);
    QGraphicsTextItem * lbl3 = new QGraphicsTextItem;
    lbl3->setPos(cx,cy);
    lbl3->setPlainText(QString::number(cx)
                       +","+QString::number(cy));
    scene->addItem(lbl3);
    scene->addLine(ax, ay, dx, dy);
    QGraphicsTextItem * lbl4 = new QGraphicsTextItem;
```

```
    lbl4->setPos(dx,dy);
    lbl4->setPlainText(QString::number(dx)
                       +","+QString::number(dy));
    scene->addItem(lbl4);
}



void Widget::onOpenClicked()
{
    QString fileName;
    fileName= QFileDialog::getOpenFileName(this,
                                           tr("Open file"), ".",
                                tr("Any file (*.*)"));
    ifstream in(fileName.toStdString().c_str());
    ListGF list;
    try
    {

        in>>list;
        list.identification();
    }
    catch( const char* err)
    {

        QMessageBox* pmbx;
        pmbx= new QMessageBox("Error",err,
                              QMessageBox::Information,
                              QMessageBox::Retry,
                              QMessageBox::Abort,
                              QMessageBox::Cancel);
        int n = pmbx->exec();
        delete pmbx;
        if (n == QMessageBox::Abort)
        {
```

```
            exit(0);
        }
        else if(n == QMessageBox::Cancel)
        {
            return;
        }
        if (n==QMessageBox::Retry)
        {
            return;
        }
    }

    for (int i=0; i<list.getCount(); i++)
    {
        ui->graphicsView->scene()->clear();
        GFigure f = list.getValue(i);
        drawF(f);
        QMessageBox::information(0,"Area",
                                 QString::number(f.area()));


    }
}
```

```
#include "widget.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    Widget w;
    w.show();

    return a.exec();
}
```

Задание 2.

Описание программы

Программа включает в себя класс «матрица» с конструкторами и методами, необходимыми для вывода матрицы на экран и вычисления суммы элементов матрицы. Класс «матрица» включает в себя структуру «элемент матрицы», в которую входит четыре указателя, связывающие элемент с верхним, нижним, правым и левым соседями, а также значение элемента матрицы.

Описание класса

Класс $Matrix$: Структура $Node$ (элемент матрицы) принадлежащая классу, включает в себя четыре указателя $up, down, left, right$ типа $Node*$ на соседние элементы матрицы, и переменную $value$ типа $integer$, которая используется для хранения значения элемента матрицы. Класс включает в себя указатель a типа $Node*$ (указатель на первый элемент матрицы), конструкторы и деструктор. А также,функцию $addLine$, которая позволяет добавить строку в матрицу,
функции $getColumns$ и $getRows$, которые используются для получения количества столбцов и строк матрицы соответсвенно,
функцию $element$, которая позволяет получить значение элемента матрицы находящегося в ячейке $[i, j]$,
функцию $elementSum$, которая считает сумму элементов матрицы,
функцию $split$, которая находит квадрат с наибольшей суммой элементов, и функции $operator << operator >>$ для ввода и вывода матриц.

Тестирование работы программы

На входных данных: 3 3 1 2 3 4 5 6 7 8 9 1 1, где первые 2 числа задают размер матрицы, последующие 9 чисел задают значение элементов матрицы по строкам, а последние 2 числа

задают элемент [i,j] относительно которого, исходная матрица разделяется на квадраты.

Программа выводит следующий результат: 3 3 $\begin{matrix}1&2&3\\4&5&6\\7&8&9\end{matrix}$ 2, где первые 2 числа это размер матрицы, затем исходная матрица, и номер квадрата с наибольшей суммой.

Исходный текст программы

```
#ifndef MATRIX_H
#define MATRIX_H
#include <stdlib.h>
#include <iostream>

using namespace std;
struct Node
{
    int value;
    Node *up;
    Node *down;
    Node *left;
    Node *right;
    Node(int x)
    {
        value = x;
        up = NULL;
        down = NULL;
        left = NULL;
        right = NULL;

    }
};
class Matrix
{
private:
    Node *a;
```

```
    void addLine (int lenght);
public:
    Matrix();
    ~Matrix();
    Matrix (const Matrix& );
    int getColumns () const;
    int getRows ()const;
    int& element(int i, int j)const;
    friend ostream& operator << (ostream&, const Matrix&);
    friend istream& operator >> (istream&, Matrix&);
    int elementSum (int rs, int re, int cs, int ce);
    int split (int i, int j);
};



#endif // MATRIX_H
```

```
#include "matrix.h"

Matrix::Matrix()
{
    a = NULL;
}
Matrix::Matrix(const Matrix &other)
{
    int columnsother = other.getColumns();
    int rowsother = other.getRows();
    for (int i=0; i<rowsother; i++)
    {
        addLine(columnsother);
    }
    for (int i = 0; i<rowsother; i++)
    {
        for (int j=0; j<columnsother; j++)
```

```
        {
            this->element(i,j) = other.element(i,j);
        }
    }
}

Matrix::~Matrix()
{
    while (a!=NULL)
    {
        Node *line = a;
        Node *next_line = a->down;
        while (line!=NULL)
        {
            Node *next_item = line->right;
            delete line;
            line = next_item;
        }
        a = next_line;
    }
}
int Matrix::getColumns()const
{
    int count = 0;
    Node *line = a;
    while (line!=NULL)
    {
        count++;
        line=line->right;
    }
    return count;
}
int Matrix::getRows()const
{
    int count = 0;
```

```
    Node *line = a;
    while (line!=NULL)
    {
        count++;
        line=line->down;
    }
    return count;
}
int& Matrix::element(int i, int j) const
{
    if (i<0||i>=getRows())
    {
        throw "Out of range";
    }
    if (j<0||j>=getColumns())
    {
        throw "Out of range";
    }
    Node *item = a;
    for (int k=0; k<i; k++)
    {
        item = item->down;
    }
    for (int k=0; k<j; k++)
    {
        item = item->right;
    }
    return item->value;
}
void Matrix::addLine(int lenght)
{
    if (a==NULL)
    {
        Node* newline = new Node(0);
        a = newline;
```

```
        for(int i=0; i<lenght-1; i++)
        {
            newline->right = new Node(0);
            newline->right->left = newline;
            newline = newline->right;
        }
        return;
    }
    Node* line = a;
    while (line->down!=NULL)
    {
        line = line->down;
    }
    Node* p = NULL;
    for (int i=0; i<lenght; i++)
    {
        line->down = new Node(0);
        line->down->up = line;

        line->down->left = p;
        if (p!=0)
        {
            p->right = line->down;
        }
        p = line->down;
        line = line->right;
    }
}
ostream& operator << (ostream& out, const Matrix& M)
{
    int rows = M.getRows();
    int columns = M.getColumns();
    out<<rows<<endl;
    out<<columns<<endl;
    for(int i=0; i<rows; i++)
```

```
    {
        for (int j=0; j<columns; j++)
        {
            out<<M.element(i,j)<<" ";
        }
        out<<endl;
    }
    return out;
}
istream& operator >> (istream& in, Matrix& M)
{
    while (M.a!=NULL)
    {
        Node *line = M.a;
        Node *next_line = M.a->down;
        while (line!=NULL)
        {
            Node *next_item = line->right;
            delete line;
            line = next_item;
        }
        M.a = next_line;
    }
    M.a = NULL;
    int rows,columns;
    in >> rows >> columns;
    for (int i=0; i<rows; i++)
    {
        M.addLine(columns);
    }
    for (int i = 0; i<rows; i++)
    {
        for (int j=0; j<columns; j++)
        {
            in >> M.element(i,j);
```

```
        }
    }
    return in;
}
int Matrix::elementSum(int rs, int re, int cs, int ce)
{
    int elementSum = 0;
    for (int i = rs; i<re; i++)
    {
        for (int j=cs; j<ce; j++)
        {
            elementSum = elementSum + this->element(i,j);
        }
    }
    return elementSum;
}
int Matrix::split (int i, int j)
{
    int rows = this->getRows();
    int columns = this->getColumns();
    int Sum1 = elementSum(0,i+1, 0, j+1);
    int Sum2 = elementSum(i, rows, j, columns);
    int Sum3 = elementSum(0, i+1, j, columns);
    int Sum4 = elementSum(i,rows,0,j+1);
    int maxSum = Sum1;
    int res = 1;
    if (maxSum<Sum2)
    {
        maxSum=Sum2;
        res = 2;
    }
    if (maxSum<Sum3)
    {
        maxSum=Sum3;
        res=3;
```

```
    }
    if (maxSum<Sum4)
    {
        maxSum=Sum4;
        res=4;
    }
    return res;
}

#include <iostream>
#include <matrix.h>

using namespace std;

int main()
{
    Matrix M;
    cin >> M;
    int i,j;
    cin >> i;
    cin >> j;
    cout << M;
    cout << M.split(i,j);
    return 0;
}
```