ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО РЫБОЛОВСТВУ
Федеральное государственное образовательное учреждение
высшего профессионального образования

“Мурманский государственный технический университет”

Кафедра
высшей математики и
программного
обеспечения ЭВМ

Пояснительная записка к курсовой работе

по дисциплине

«Объектно-ориентированное программирование»

Выполнил:
студент группы П-371
Запорожцев И.Ф.

Проверил:
старший преподаватель
кафедры ВМ и ПО ЭВМ
Ефименко П.А.

Научный консультант:
к.ф.-м.н., доцент
кафедры ВМ и ПО ЭВМ
Богомолов Р.А.

Мурманск
2010

СОДЕРЖАНИЕ

Описание предметной области

Пусть F – алгебраически замкнутое поле нулевой характеристики, S_n – симметрическая группа на множестве символов $\{1,2,...,n\}$ и $F[S_n]$ – её групповая алгебра над полем F.

Известно, что минимальные левые идеалы в групповой алгебре симметрической группы $F[S_n]$ имеют вид:

$I = I_d(\nu) = F[S_n]\cdot e_d \cdot \sum_{\tau\in St(d)} \nu_\tau \tau^{-1}$, где

e_d – симметризатор Юнга на диаграмме Юнга *d*;

$St(d)\subset S_n$ ($Card\ St(d)$ вычисляется по формуле «крюков») – множество подстановок, являющихся нумерациями стандартных таблиц τd на диаграмме Юнга *d;*

$\nu = \sum_{\tau\in St(d)} \nu_\tau \tau^{-1}$ – точка проективного пространства, полученного проективизацией линейной оболочки $\left\{e_d\tau^{-1}\right\}_{\tau\in St(d)}$, то есть $\nu \in \mathbf{P}\left\langle\left\{e_d\tau^{-1}\right\}_\tau\right\rangle \cong \mathbf{P}\left\langle\left\{\tau e_d\right\}_\tau\right\rangle = \mathbf{P}\left(F[S_n]e_d\right)$.

Зададим в $F[S_n]$ F-линейный базис $\left\{e_\tau^\sigma(d)\right\}_{\tau,\sigma,d}$ [3], где $e_\tau^\sigma(d) = \sigma e_d \tau^{-1},\ \sigma,\tau\in St(d)$.

Ветвление идеала I есть $I_d(\nu) = \bigoplus_{(d',\nu')} I'_{d'}(\nu')$, где $I = I_d(\nu)\subset F[S_n],\ I'_{d'}(\nu')\subset F[S_{n+1}]$.

Будучи левым идеалом, $I_d(\nu)$ порождён элементом $\nu^* = \sum_{\tau\in St(d)} \nu_\tau e_d \tau^{-1}$.

После разложения $e_d\tau^{-1}$ по базису $\left\{e_{\tau'}^{\sigma'}(d^*)\right\}_{\sigma',\tau',d^*}$ элементы полученного разложения группируются в соответствии с диаграммами d':

$$e_d\tau^{-1} = \sum_{d'}\sum_{\sigma',\tau'\in St(d')} \beta_{\tau'}^{\sigma'}(d,\tau)\cdot e_{\tau'}^{\sigma'}(d').$$

Для отображения результатов (при фиксированной d') следует использовать символическую запись $e\tau d = \sum_{\tau'} \beta_{\tau'}^{\sigma'}(d,\tau) e\tau' d'$ вместо $e_d\tau^{-1} = \sum_{\tau'} \beta_{\tau'}^{\sigma'}(d,\tau) e_{\tau'}^{\sigma'}(d')$.

Алгоритм построения матрицы перехода

1. Генерация подстановок в антилексикографическом порядке.

2. Выбор инволюций из списка подстановок.

3. Генерация разбиений натурального числа (элементы в списке расположены в лексикографическом порядке).

4. Генерация стандартных таблиц по инволюциям (алгоритм *строчной вставки Шенстеда*). На каждом шаге k по таблице $\tau^* d^*$ и очередному элементу $\omega(k)$ из инволюции

ω алгоритм строит новую таблицу. Эта стандартная таблица содержит на одну клетку больше, чем $\tau^* d^*$, а множество элементов таблицы содержит все те же элементы, что и $\tau^* d^*$, а также $\omega(k)$. Нумерация определяется следующим образом: если $\omega(k)$ больше каждого элемента из первой строки, то устанавливаем его в новую пустую клетку в конце первой строки. В противном случае находим самый левый элемент первой строки, превосходящий $\omega(k)$. «Выбиваем» этот элемент из клетки и ставим $\omega(k)$ на его место. С выбитым элементом повторяем те же действия по отношению ко второй строке. Продолжаем действовать аналогично, пока очередной выбитый элемент не оказывается в конце очередной строки или не выбивается из последней – в этом случае он образует новую строку таблицы Юнга из одного элемента. Этот алгоритм ещё называют *строчным выбиванием*.

5. Сортировка таблиц в соответствиии с отношением порядка на диаграммах и нумерациях каждой диаграммы.

6. Вычисление симметризатора Юнга для каждой таблицы в соответствии с определением.

7. Вычисление элементов $e_\tau^\sigma(d)$.

Если упорядочить элементы $\theta_i \in S_n$ и соответственно слагаемые в разложении $\left\{e_\tau^\sigma(d)\right\}_{\tau,\sigma,d}$, то результаты можно представить матрицей перехода:

$$A_{(n!\times n!)} = \begin{array}{c|c} & \overbrace{\sigma_1 e_{d_1}\tau_1^{-1} \quad \sigma_1 e_{d_1}\tau_2^{-1} \quad .... \quad \sigma_p e_{d_1}\tau_p^{-1}}^{\sigma,\tau\in St(d_1),\ Card\ St(d_1)=p} \quad ..... \quad \overbrace{\sigma_1 e_{d_s}\tau_1^{-1} \quad \sigma_1 e_{d_s}\tau_2^{-1} \quad ..... \quad \sigma_{p'} e_{d_s}\tau_{p'}^{-1}}^{\sigma,\tau\in St(d_s),\ Card\ St(d_s)=p'} \\ \hline \theta_1 & \begin{pmatrix} ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... \\ ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... \\ ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... \\ ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... \\ ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... & ..... \end{pmatrix} \\ \theta_2 & \\ \theta_3 & \\ \vdots & \\ \theta_{n!} & \end{array}$$

Пример матрицы перехода для $n=3$:

$$A_{(6\times 6)} = \begin{array}{c|c} & s_1 \quad s_2 \quad s_3 \quad s_4 \quad s_5 \quad s_6 \\ \hline 123 & \begin{pmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & 0 & 1 & 1 \\ -1 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & -1 & -1 & 1 \\ 1 & -1 & -1 & 1 & 0 & 1 \\ -1 & 1 & 0 & -1 & -1 & 1 \end{pmatrix} \\ 213 & \\ 132 & \\ 312 & \\ 231 & \\ 321 & \end{array}$$

Столбец слева определяет представление подстановок, сгенерированных в антилексикографическом порядке. Если учесть, что упорядоченный список разбиений имеет вид $\left\{\{1,1,1\};\{2,1\};\{3\}\right\}$, то элементы базиса $\left\{s_i\right\}_i = \left\{e_\tau^\sigma(d)\right\}_{\tau,\sigma,d}$ будут:

$$s_1 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} e_{\{1,1,1\}} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}^{-1};$$

$$s_2 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} e_{\{2,1\}} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}^{-1};$$

$$s_3 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} e_{\{2,1\}} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}^{-1};$$

$$s_4 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} e_{\{2,1\}} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}^{-1};$$

$$s_5 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix} e_{\{2,1\}} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 2 \end{pmatrix}^{-1};$$

$$s_6 = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix} e_{\{3\}} \begin{pmatrix} 1 & 2 & 3 \\ 1 & 2 & 3 \end{pmatrix}^{-1}.$$

Постановка задачи

Цель: разработать программный продукт для выполнения разложения порождающих элементов минимальных левых идеалов по базису групповой алгебры, степень группы которой увеличена на единицу.

Входные данные. Возможны три варианта решаемой задачи с различной степенью общности:

1) для всех порождающих элементов заданной степени симметрической группы;
2) для всех порождающих элементов на заданной диаграмме;
3) для одного порождающего элемента, заданного таблицей Юнга.

Выходные данные: члены разложения по базису.

При обработке входных данных программа отыскивает решение в заранее созданных файлах, таким образом, запрос пользователя не инициирует трудоёмкий вычислительный процесс, а получает доступ к набору готовых решений.

Банк готовых решений следует организовать в виде набора файлов следующих типов:

1) Файл, хранящий данные о всех диаграммах и таблицах Юнга на них, которые отвечают степени симметрической группы (*СТЕПЕНЬ.bas*).
2) Файл, хранящий матрицу перехода от стандартного базиса подстановок групповой алгебры к нетривиальному базису, заявленному в описании предметной области (*СТЕПЕНЬ.txt*). Из теории известно, что эта матрица является сильно разреженной и имеет порядок, равный факториалу степени группы, поэтому для решения матричных уравнений преобразования базиса используется библиотека UMFPACK, имеющаяся в Интернете в свободном доступе. Способ хранения матрицы перехода определяется требованиями пакета UMFPACK.
3) Файл для хранения результатов решения (пар *Таблица-Линейная комбинация базисных элементов*). Имя файла – *СТЕПЕНЬ.ytf*.

В программном продукте должны быть доступные пользователю функции обновления (пересчёта) всех файлов банка готовых решений.

Декомпозиция задачи

Для достижения поставленной цели потребовалось разделение решения на следующие проекты, которые являются наиболее крупными этапами решения:

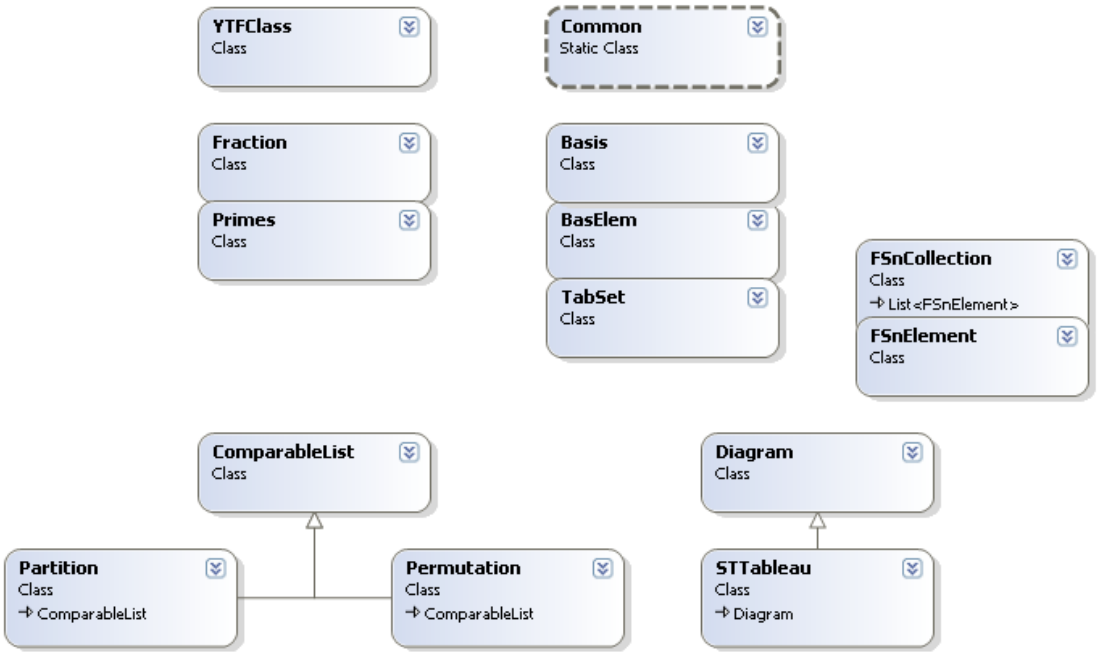
1) Библиотека *Algebra* предназначен для выполнения основных вычислительных задач.
2) Библиотека *UmfSolver* необходима для взаимодействия с UMFPACK
3) Библиотека *DrawingTool* осуществляет отрисовку графики в качестве выходных данных.
4) Библиотека *TabListBox* представляет элемент управления для организации взаимодействия с пользователем.
5) Оконное приложение *Client* является своего рода клиентом, с помощью которого пользователь формирует задания и получает ответы.

Проект Algebra

Сущности:

1) Перестановка (Permutation),
2) Разбиение натурального числа (Partition),
3) Диаграмма Юнга (Diagram),
4) Стандартная таблица Юнга (STTableau),
5) Простой элемент групповой алгебры (FSnElement),
6) Составной элемент групповой алгебры (FSnCollection),
7) Базисный элемент (BasElem),
8) Базис (Basis),
9) Рациональная дробь (Fraction)

Классы проекта



Класс Primes

Класс для генерации и хранения простых чисел. Результаты хранятся в файлах *НачалоДиапазона_КонецДиапазона.prm*

```
[Serializable]  public class Primes
    {
        public List<int> Elems;      Список простых чисел
        public Primes()
```

Генерация и сохранение

```
        public void CreatePrimes(int start,int stop)
```

Класс Fraction

Класс для работы с рациональными дробями

```
[Serializable]    public class Fraction
    {
        private int num;   Числитель
        private int den;   Знаменатель
        public int Num
        {
            get { return num; }
        }

        public int Den
        {
            get { return den; }
        }
```

Конструктор принимает на вход два значения, отыскивает минимальное из них, проверяет делимость на простые числа

```
        public Fraction(int ch, int zn)
```

Конструктор для единицы

```
        public Fraction()
```

Загрузка файла со списком сохранённых простых чисел, начиная с number

```
        private void UsePrimesSimplify(int number,int min)
```

Арифметика

```
        public static Fraction operator +(Fraction a, Fraction b)
        public static Fraction operator -(Fraction a, Fraction b)
        public static Fraction operator *(Fraction a, Fraction b)
        public static Fraction operator /(Fraction a, Fraction b)
```

Операторы сравнения

```
        public static bool operator ==(Fraction a, Fraction b)
        public static bool operator !=(Fraction a, Fraction b)
        public override bool Equals(object obj)
        public override int GetHashCode()
```

Представление строкой вида a/b

```
        public override string ToString()
```

Класс ComparableList

Класс для представления списка единым объектом. На множестве представителей этого класса установлено отношение порядка. Является родительским классом для класса перестановок и класса разбиений.

```
[Serializable]  public class ComparableList
    {
        private List<int> permNumbers;          Элементы списка
        public List<int> PermutationNumbers
        {
            get { return permNumbers; }
        }
        public int Count                        Число элементов
        {
            get { return permNumbers.Count; }
        }
```

Конструкторы:

```
        public ComparableList()
        public ComparableList(int number)
        public ComparableList(List<int> z)      Загрузка элементов из обычного списка
        public ComparableList(string str)       Загрузка элементов из строки через ‘,’

        protected void Init(List<int> z)        Функция заполнения списка
```

Отношение полного лексикографического порядка

```
        public static bool operator >(ComparableList a, ComparableList b)
        public static bool operator <(ComparableList a, ComparableList b)
        public static bool operator ==(ComparableList a, ComparableList b)
        public static bool operator !=(ComparableList a, ComparableList b)
        public override bool Equals(object obj)
        public override int GetHashCode()
```

Оператор индексирования (чтение)

```
        public int this[int index]
```

Добавление в список

```
        internal void Add(int c)
```

Представление списка строкой вида (a,a,..,a)

```
         public override string ToString()
```

Класс Partition

Класс разбиения натурального числа. Наследуется от ComparableList. При загрузке элементов из простого списка требуется, чтобы каждый следующий элемент был не больше предыдущего.

```
[Serializable]  public class Partition : ComparableList
    {
        private int number;       Сумма членов – число для разбиения

        public int Number
        {
            get{ return number;  }
        }
```

Конструкторы

```
        public Partition(): base() {}
```

```
        public Partition(int n): base(n){}
        public Partition(string str):base(str){}
        public Partition(List<int> z)
```
Загрузка элементов с проверкой условия

Класс Permutation

Класс, реализующий логику работы с элементами симметрической группы

```
[Serializable]   public class Permutation : ComparableList
    {
```

Конструкторы

```
        public Permutation():base(){}
        public Permutation(int n): base(n){}
        public Permutation(string str):base(str){}
```

Загрузка элементов с проверкой на то, чтобы они были были отрезком натурального ряда, начиная с единицы

```
        public Permutation(List<int> z)
```

Обращение (возведение в -1 степень)

```
        internal Permutation RevPerm()
```

Групповая операция

```
        public static Permutation operator *(Permutation a, Permutation b)
```

Проверка инволютивности

```
        internal bool IsInvolution()
```

Проверка чётности

```
        internal bool IsEven()
```

Класс FSnElement

Простой элемент групповой алгебры, представляемый как целое*подстановка

```
public class FSnElement
    {
        private int _FElement;
        private Permutation _SnElement;

        public Permutation SnElement
        {
            get {return _SnElement;}
            set {this._SnElement = value;}
        }
       public int FElement
        {
            get { return _FElement; }
            set { this._FElement = value; }
        }
```

Коэффициент из поля

Элемент симметрической группы

Конструкторы

```
        public FSnElement(){}
        public FSnElement(int num, Permutation per)
```

Класс FSnCollection

Составной элемент групповой алгебры – линейная комбинация простых

```
public class FSnCollection: List<FSnElement>

        public FSnCollection() {}
        public static FSnCollection operator *(FSnCollection a, Permutation b)
```

Класс Diagram

Диаграмма Юнга

```
[Serializable]  public class Diagram
    {
        private int width = 0;      //количество клеток в первой строке
        private int height = 0;     //количество клеток в первом столбце
        private int number = 0;     //количество клеток

        protected Partition diagElemRow;
        protected Partition diagElemCol;

        public Partition DiagElemRow
        {
            get { return diagElemRow; }
        }

        public Partition DiagElemCol
        {
            get { return diagElemCol; }
        }

        public int Width
        {
            get{ return width;}
        }
        public int Height
        {
            get { return height; }
        }
        public int Number
        {
            get { return number;}
        }
```

Разбиение построчное

Разбиение по столбцам

Конструкторы

```
        public Diagram(){}
```

Загрузка по строчному разбиению

```
        public Diagram(Partition partition)
```

Оператор индексирования

```
        internal int this[int index]
        {
            get { return (int)diagElemRow[index]; }
        }
```

Определение столбцового разбиения

```
        private Partition GetDiagElemCol()
```

Установка параметров при генерации по Шенстеду

```
        protected void SetNumber(int p)
        protected void SetWidth(int p)
```

```
protected void SetHeight(int p)
```

Определение равенства

```
public static bool operator ==(Diagram a, Diagram b)
public static bool operator !=(Diagram a, Diagram b)
public override bool Equals(object obj)
public override int GetHashCode()
```

Класс STTableau

Таблица Юнга – диаграмма Юнга, заполненая числами от 1 до Number

```
[Serializable]   public class STTableau : Diagram
    {
        private Permutation numeration;   Нумерация таблицы по столбцам
        public Permutation Numeration
        {
            get { return this.numeration; }
        }
```

Конструктор – задание разбиения и нумерации

```
public STTableau(Permutation perm, Partition diagelem):base(diagelem
```

Генерация по алгоритму Шенстеда (на входе инволюция)

```
public STTableau(Permutation perm):base()
```

Отношение полного лексикографического порядка по нумерациям

```
public static bool operator >(STTableau a, STTableau b)
public static bool operator <(STTableau a, STTableau b)
public static bool operator ==(STTableau a, STTableau b)
public static bool operator !=(STTableau a, STTableau b)
public override bool Equals(object obj)
public override int GetHashCode()
```

Преобразование в таблицу чисел, где отсутствующие клетки заполняются нулями

```
private Array ToMatrix()
```

Проверка того, является ли таблица горизонтальной

```
internal bool IsHorizontal()
```

Проверка того, является ли таблица вертикальной

```
internal bool IsVertical()
```

Вычисление симметризатора горизонтальной таблицы (= строковый стабилизатор)

```
private FSnCollection IfHorizontalStabilizer()
```

Вычисление симметризатора вертикальной таблицы (= столбцовый стабилизатор)

```
private FSnCollection IfVerticalStabilizer()
```

Вычисление строкового стабилизатора негоризонтальной таблицы

```
private List<Permutation> RowStabilizer(Array matr)
```

Вычисление столбцового стабилизатора невертикальной таблицы

```
private List<Permutation> ColStabilizer(Array matr)
```

Вычисление симметризатора Юнга

```
internal FSnCollection YoungSymmetrizer()
```

Класс TabSet

Класс для хранения списка нумераций стандартных таблиц на диаграмме с заданным разбиением

```
[Serializable]    public class TabSet
    {
        private Partition diagPart;                 Разбиение
        private List<Permutation> numerations;      Список нумераций

        public List<Permutation> TabGroupNumerations
        {
            get { return numerations; }
        }
        public Partition TabGroupPartition
        {
            get { return this.diagPart; }
        }
```

Конструктор с заданием разбиения, нумерации остаются пустыми

```
        public TabSet(Partition part)
```

Оператор индексации

```
        public Permutation this[int index]
        {
            get { return this.numerations[index]; }
        }
```

Класс BasElem

Класс, принимающий с выхода UmfSolver текстовый файл с решением матричного уравнения и преобразующий каждое ненулевое значение его в базисный элемент с коэффициентом из поля

```
[Serializable]    public class BasElem
    {
        public readonly STTableau Sigma;
        public readonly STTableau Tau;
        public readonly Fraction Coeff;
        public readonly int SigmaInt;   //нумерация с единицы
        public readonly int TauInt;     //нумерация с единицы
```

Поля заполняются, исходя из значений номера столбца матрицы перехода и коэффициента (собственно компоненты вектора решения)

```
        public BasElem(Basis bas, int poss, Fraction coeff)
```

Класс Basis

```
Класс, выполняющий работу по формированию матрицы перехода, списков нумераций
таблиц и их упорядочения

[Serializable]  public class Basis
    {
        private int number;                 Степень симметрической группы
        public int Number
        {
            get { return number; }
        }
```

Упорядоченный список всех подстановок группы

```
        public List<Permutation> order;
```

Список диаграмм и таблиц на них (упорядочены и те и другие)

```
public List<TabSet> SaveTabGroups;
```

Представление матрицы перехода в формате UMFPACK

```
[NonSerialized] public List<int> Ap;  //для Umfpack
[NonSerialized] public List<int> Ai;  //для Umfpack
[NonSerialized] public List<int> Ax;  //для Umfpack
```

Конструкторы

```
public Basis(){}
```

Конструктор с инициализацией номера и списокв order & SaveTabGroups

```
public Basis(int num)
```

Сохранение order & SaveTabGroups в .bas (функция ядра должна быть выполнена)

```
internal void CreateForUMFmyFormat()
```

Сохранение матрицы перехода в текстовый файл для UMFPACK

```
internal void CreateForUMF()
```

Построение (сортированного по нумерациям и диаграммам) списка диаграмма-нумерации SaveTabGroups

```
internal void Core()
```

Класс Common

Статический класс, осуществляющий взаимодействие с остальными проектами. В то время как конструкторы классов библиотеки Algebra доступны извне, основные их вычислительные функции при необходимости вызываются с помощью методов класса Common

```
public static class Common
    {
        [DllImport("UmfSolver.dll")]
        public static extern void UmfpackSolve();
```

Инициирует работу с UMFPACK

Определение списка простых чисел в заданном диапазоне и сохранение в файл

```
public static void CalculatePrimes(int start, int stop)
```

Обычная генерация перестановок указанной длины в антилексикографическом порядке

```
public static List<Permutation> GeneratePermutations(int capacity)
```

Рекурсивная процедура для генерации перестановок

```
public static void PermAntilexBase(Permutation permutation,
       int RecursVar, List<Permutation> PermList)
```

Генерация перестановок из элементов списка, где длина перестановки указывается дополнительно; элементы, отсутствующие в списке занимают свои места в соответствии с номером (тождественно)

```
public static List<Permutation> GeneratePermutationsFromList(
       List<int> list, int capacity)
```

Аналогично предыдущему

```
public static void GeneratePermutationsFromList(List<int> input,
       List<Permutation> output, int capacity)
```

Рекурсивная процедура для генерации перестановок в антилексикографическом порядке из списка элементов

```
public static void PermAntilex(Permutation permutation,
          List<int> source, int RecursVar, int totalnumber,
          List<Permutation> permutationlist)
```

Генерация тривиальной таблицы на заданной диаграмме

```
public static STTableau CreateIdSTTableau(Partition part)
```

Факториал числа

```
public static int Factorial(int number)
```

Генерация тождественной подстановки заданной длины

```
public static Permutation CreateIdPermutation(int n)
```

Генерация упорядоченного списка разбиений числа

```
public static List<Partition> GeneratePartitions(int number)
```

Сортировка в лексикографическом порядке

```
public static void RadixSorting(int[] arr, int range, int length)
```

Генерация базисных файлов (.bas, .txt)

```
public static void CreateBasisFiles(int num)
```

Формирование файла правых частей для решения с помощью UMFPACK

```
private static void FromFSnCollectionToTXTFile(
     FSnCollection Collection, Basis bhigh)
```

Анализ текстового файла с выхода UmfSolver

```
private static List<BasElem> FromTXTFileToBasElems(Basis bhigh)
```

Разложение порождающего элемента идеала

```
public static List<BasElem> TranslateEdTauToBasElems(STTableau tab)
```

Генерация и сохранение .ytf

```
public static void CreateAndSaveYTF(int number)
```

Открытие файла .bas

```
public static Basis OpenBasFile(int n)
```

Поиск разложений порождающих элементов на заданной диаграмме в ytf

```
public static List<YTFClass> FindTabsOnDiagInYTF(Basis bas, Diagram d)
```

Поиск разложения порождающего элемента в ytf

```
public static List<YTFClass> FindTabInYTF(Basis bas, STTableau t)
```

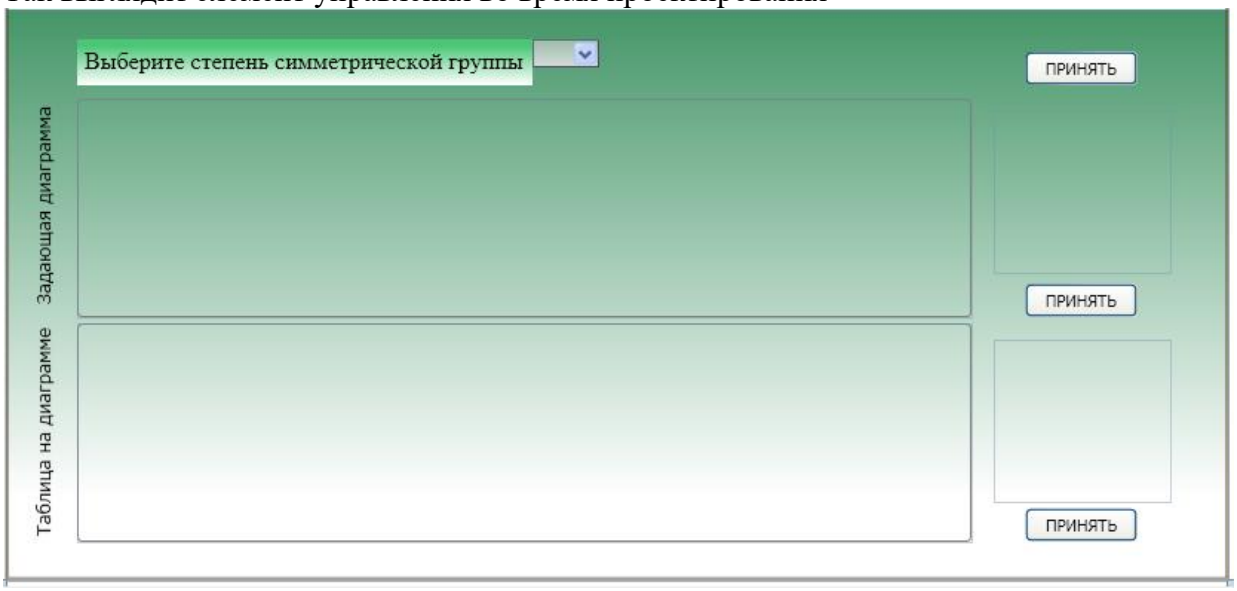
Проект UmfSolver

Данный проект является оболочной для библиотеки кода на C++ для решения систем линейных уравнений с разреженной матрицей. Входные данные: два текстовых файла – с матрицей перехода, записанной в формате UMFPACK и столбец правых частей right.txt. На выходе – файл, в каждой строке которого два числа – номер ненулевой компоненты вектора решения и её значение.

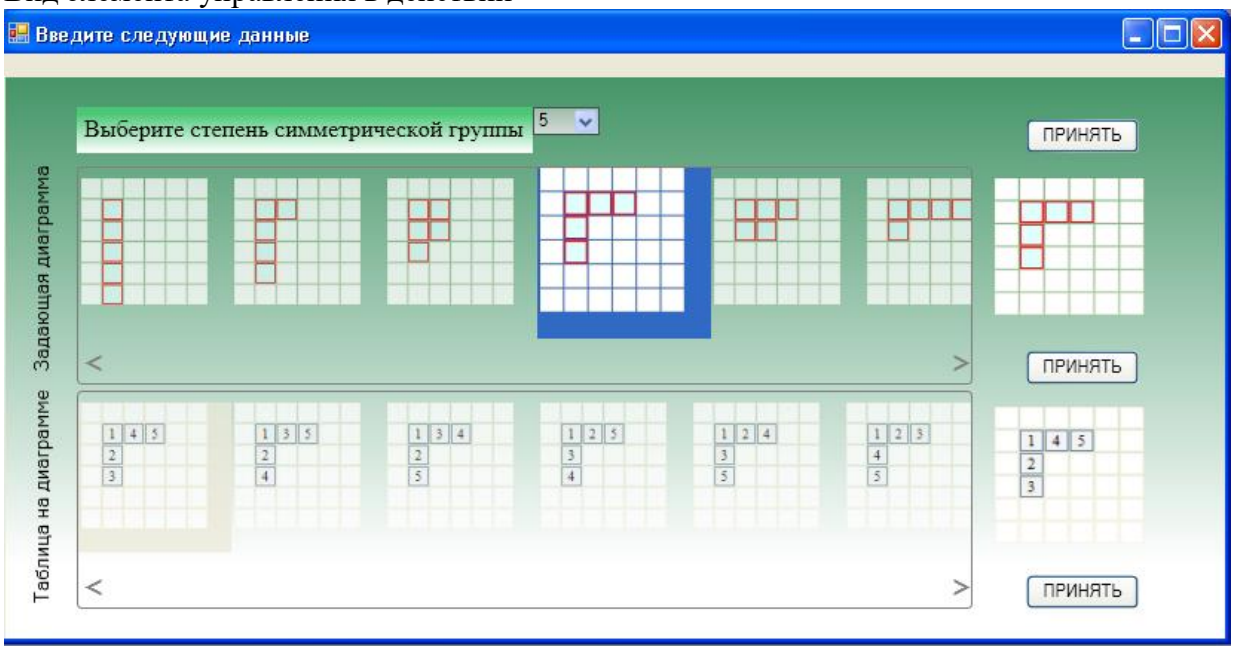
Проект TabListBox

Назначение проекта – создание декорированного элемента взаимодействия с пользователем по выбору задания. Кроме элемента управления, разработанного при помощи технологии WPF и языка разметки XAML, проект содержит классы для заполнения визуализированных списков.

Так выглядит элемент управления во время проектирования



Вид элемента управления в действии



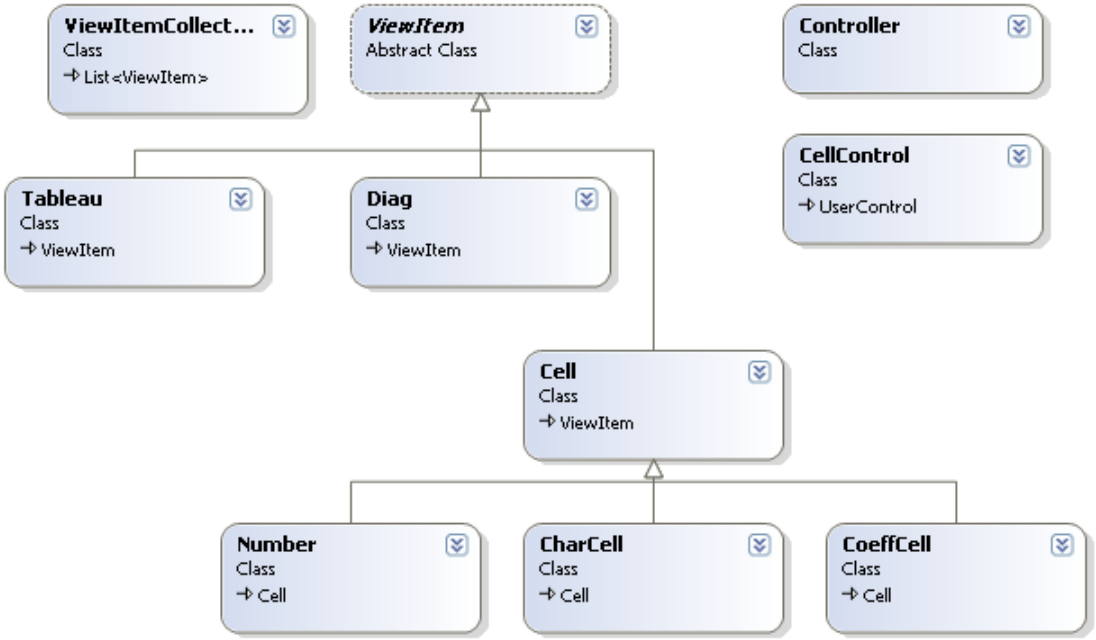
Проект DrawingTool

Основная задача проекта – визуализовать результаты расчётов, полученные с помощью функций Algebra на элементе управления CellControl. Гибкость настройки параметров самого контрола и изменение графических объектов осуществляется за счёт методов класс Controller, который является «владельцем» элемента управления. Он хранит список визуальных элементов ViewItem. Основным графическим классом является Cell. Остальные клетки отличаются от указанного размерами, цветом, наличием символьного заполнения. Сложные графические классы: таблица и диаграмма, -- представляют собой список других ViewItem со специфической компоновкой.

Сущности:

1) Обозреваемый элемент (ViewItem),
2) Таблица Юнга (Tableau),
3) Диаграмма Юнга (Diagram),
4) Клетка (Cell),
5) Клетка-коэффициент (CoeffCell),
6) Клетка с цифрой (Number),
7) Клетка с символом (CharCell),
8) Класс управления (Controller),
9) Элемент управления для отрисовки (CellControl)

Классы проекта



Класс ViewItem

Класс визуального элемента

```
public abstract class ViewItem
  {
      protected Region Region;
      internal Controller Master;
      protected int row;
      protected int column;
      internal int Row
```

Занимаемая область

Номер строки верхнего левого угла

Номер столбца верхнего левого угла

```
        {
            get { return row;}
        }
        internal int Column
        {
            get { return column;}
        }
```

Конструктор

```
        protected ViewItem(Controller c,int row,int col)

        public virtual void Draw(Graphics g)
        {
        }
```

Класс ViewItemCollection

Коллекция визуальных элементов

```
public class ViewItemCollection : List<ViewItem>
    {
        private Controller Master;
        public ViewItemCollection(Controller m)
```

Поиск элемента с указанными координатами

```
        public ViewItem GetViewItemAtXY(int x, int y)
```

Добавление в список с удалением того элемента, который занимал ранее позицию (x,y)

```
        public void SetViewItemAtXY(ViewItem item)
```

Класс Cell

Класс базовой клетки

```
    public class Cell : ViewItem
    {
        internal int CellHeight;    Высота
        internal int CellWidth;     Ширина
        internal int CellBorder;    Величина отступа между клетками этого типа
```

Задание области отображения

```
        private void CreateRegion()
```

Конструктор укрупнённой клетки

```
        public Cell(Controller owner, int row, int column,int mul)
            : base(owner, row, column)
```

Конструктор базовой клетки

```
        public Cell(Controller owner,int row, int column)
            : base(owner,row,column)
```

Функция рисования

```
        public override void Draw(Graphics g)
```

Класс CharCell

Клетка с символом. Отличается от базовой цветом границы, фона.

```
public class CharCell: Cell
    {
```

```
        public readonly char Value;       Символ
        public CharCell(Controller owner, int row, int column,char value)
            : base(owner, row, column)

        public override void Draw(Graphics g)
```

Класс CoeffCell

Клетка-коэффициент. Отличается от основной размером и цветом.

```
internal class CoeffCell:Cell
        public readonly Fraction Value;      Дробь в качестве коэффициента

        public CoeffCell(Controller owner, int row, int column, Fraction value)
            : base(owner, row, column,2)

        public override void Draw(Graphics g)
```

Класс Number

Клетка с номером используется для отображения таблиц Юнга

```
internal class Number : Cell
    {
        public readonly STTableau Tab;   Таблица Юнга
        public readonly int Index;       Номер числа в нумерации таблицы
        public readonly string Value;    Число

        public Number(Controller owner, int row, int column,
                STTableau T, string value, int index)
                : base(owner, row, column)

        public override void Draw(Graphics g)
```

Класс Diag

Визуализация Диаграммы Юнга – список клеток с символом пробел.

```
public class Diag: ViewItem
    {
        private List<CharCell> DiagCells;
        private Diagram Diagram;

        public Diag(Controller owner,int row, int col,Diagram d)
            : base(owner,row,col)

        public override void Draw(Graphics g)
```

Класс Tableau

Визуализация таблицы Юнга – список клеток с числом

```
public class Tableau: ViewItem
    {
        private List<Number> TableauCells;
        private STTableau Tab;

        public Tableau(Controller owner,int row, int col,STTableau t)
            : base(owner,row,col)
        public override void Draw(Graphics g)
```

Класс Controller

Класс, управляющий отрисовкой

```
public class Controller
    {
        public ViewItemCollection Cells;
        internal int BaseCellWidth = 15;          Ширина базовой клетки
        internal int BaseCellHeight = 15;         Высота базовой клетки
        internal int BaseCellBorder = 2;

Ширина элемента управления в базовых клетках
        internal int BaseCellControllerColNumber = 50;

Высота элемента управления в базовых клетках
        internal int BaseCellControllerRowNumber = 35;

Реальная высота изображения в базовых клетках
        internal int BaseCellRealRowNumber = 2;

Элемент управления для отрисовки
        internal CellControl WorkField;

Параметры отображения текста
        internal StringFormat Format;
        internal Font ControllerFont = new Font("Times New Roman", 8);
        internal Font ControllerLetterFont = new Font("Times New Roman", 10);

Текущие позиции в масштабе базовой клетки
        private int CurrentR = 1;
        private int CurrentC = 1;

Стартовые позиции в масштабе базовой клетки
        private int StartR = 1;
        private int StartC = 1;

Текущая величина будущего сдвига по высоте
        private int CurrentDH = 1;

Сдвиг по ширине
        private int ParagraphDW = 2;

        public void Init()

Инициализация элемента управления на родительской форме, начало отрисоки
        public void EnableWorkField(Form f)

        public Controller()
```

Конструктор контроллера с заданием на отрисовку разложения одного порождающего элемента

```
        public Controller(STTableau Tab, List<BasElem> Elems)
```

Конструктор контроллера с заданием на отрисовку разложения списка порождающих элементов

```
        public Controller(List<YTFClass> list)
```

Конструктор контроллера с заданием на отрисовку разложения всех порождающих элементов заданной степени группы

```
        public Controller(int number)
```

Добавление базовой клетки в заданную позицию

```
        private void AddCell(int row, int col)
```

Добавление базовой клетки в заданную позицию

```
private void AddTableau(int row,int col,STTableau tab)
```

Добавление клетки с символомв заданную позицию

```
private void AddChar(int row,int col,char c)
```

Добавление клетки-коэффициента в заданную позицию

```
private void AddCoeff(int row, int col,Fraction fr)
```

Добавление диаграммы в заданную позицию

```
private void AddDiag(int row, int col, Diagram d)
```

Аналогично добавление в текущую позицию с её дальнейшим пересчётом

```
private void AddTableau(STTableau tab)
private void AddChar(char c)
private void AddCoeff(Fraction fr)
```

Построение изображений для TabListBox

```
public void MakePictures(int number)
```

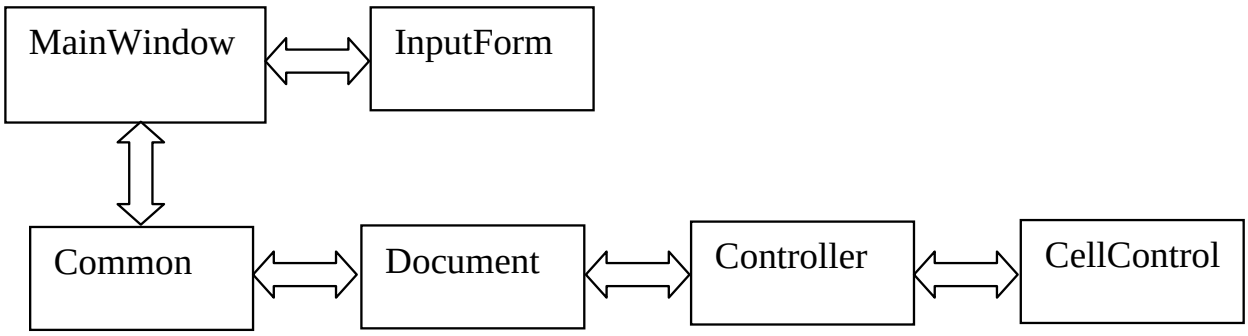
Проект Client

Проект состоит из трёх форм:

1) MainWindow – главная форма.
2) InputForm – форма ввода данных.
3) Document – форма отображения результата.

InputForm является владельцем элемента управления TabListBox, Document – владеет классом CellControl.

Схема взаимодействия проектов и классов



Выводы

В ходе выполнения курсовой работы были решены поставленные задачи в рамках предметной области.

Решение выполнено таким образом, что логически независимо работающие модули были выделены в качестве самостоятельных проектов, что повышает уровень абстрагирования, возможности отладки и модификации.

Программный продукт также удовлетворяет базовым концепциям объектно-ориентированного программирования, так как в нём активно используются принципы наследования, инкапсуляции и полиморфизма.

Список литературы

1. Троелсен, Э. C# и платформа .Net. Библиотека программиста. – СПб: Питер, 2007. – 796 с.
2. Мак-Дональд, М. WPF в .Net 3.5 с примерами на C# 2008 для профессионалов. – 2-е издание: Пер. с англ. – М. : Вильямс, 2008. – 928 с.
3. Фултон, У. Таблицы Юнга и их приложения к теории представлений и геометрии / Пер. с англ. – М. : МЦНМО, 2006. – 328 с.

ПРИЛОЖЕНИЕ. Пример работы программы

Разложение всех порождающих элементов при степени симметрической группы = 3