

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ИВАНОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК

КАФЕДРА ПРИКЛАДНОЙ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК

ОТЧЁТ

о прохождении производственной практики,
научно-исследовательской работы

Направление подготовки:	02.04.01 Математика и компьютерные науки
Выполнил:	студент 1 курса магистратуры очной формы обучения _____ Фролов Павел Владимирович
Руководитель практики от ИвГУ:	зав. кафедрой прикладной математики и компьютерных наук, кандидат физико-математических наук _____ Соколов Евгений Викторович
Дата сдачи, оценка	

Содержание

Введение	3
§1. Рефакторинг кода и результат	4
Список литературы.....	5
Приложение. (Листинг кода)	
main.cpp.....	6
FM_Two_Groups.....	7
FiniteSubGroup.h.....	12
CyclicSubGroup.h.....	13
FiniteCyclicSubGroup.h.....	13
FiniteCyclicSubGroup.cpp	13
FG_AbelianGroupCSG.h	14
FG_AbelianGroupCSG.cpp.....	14
FCSG_FGAbelianGroup.cpp.....	15

Введение

Целью работы была программная реализация решения проблемы сопряженности для свободного произведения двух групп с объединённой подгруппой. За основу была взята работа [3].

В начале работы был проведён анализ кода из [3], были устранены ошибки. В дальнейшем был проведён рефакторинг кода и тестирование результатов. В §1 описываются изменения кода. Конечный список изменённых классов и результат описан в приложении (стр.6).

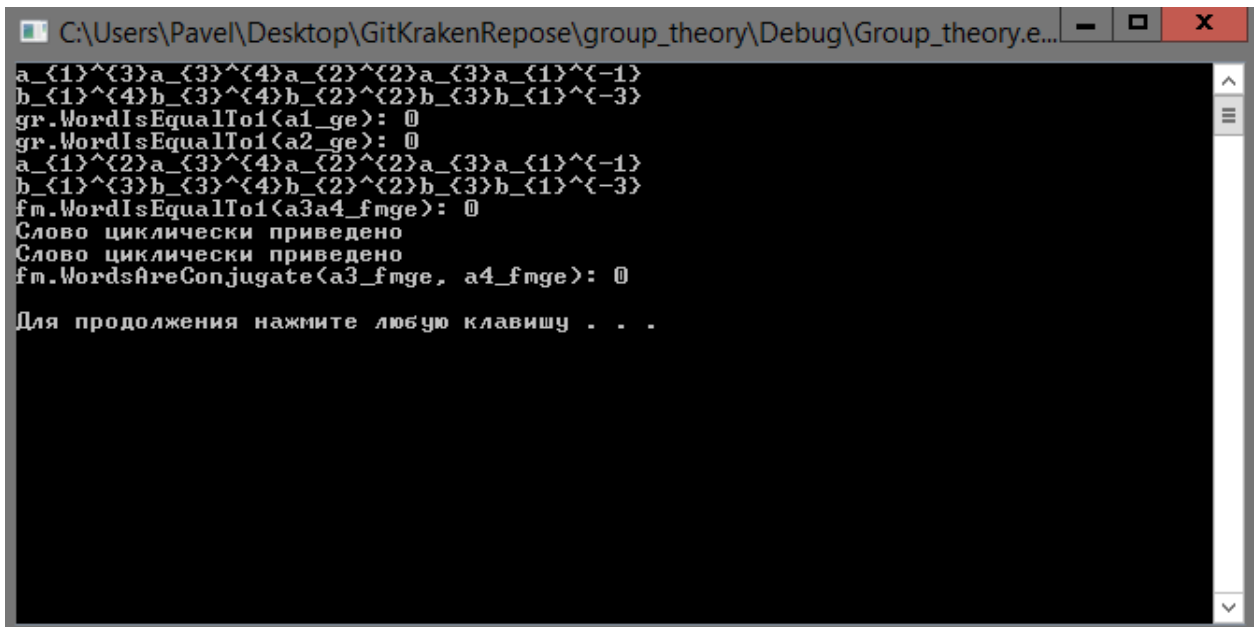
§1. Рефакторинг кода и результат.

Первым делом была устранена проблема с наследованием классов. Таким образом виртуальное наследование получили классы: **CyclicSubGroup**, **FG_AbelianGroupCSG**, **FiniteCyclicSubGroup**, **FiniteSubGroup**.

Также для этих классов пришлось добавить дополнительный параметр конструктора **SubGroup(g.GetGroup())** в исполняющих файлах.

Следующим этапом была проблема в методе **WordsAreConjugate** в классе **FM_Two_Groups**. Как оказалось, ошибка заключалась в вызываемой методом функции **ReducedFormOf**. В последнем методе была исправлена ошибка в итераторах, которая не позволяла произвести необходимые расчёты для решения проблемы сопряжённости для свободного произведения двух групп с объединённой подгруппой. Изменения этого метода можно увидеть в листинге кода(стр.6).

В файле **main.cpp** были добавлены соответствующие поля для проверки конечного результата.



```
C:\Users\Pavel\Desktop\GitKrakenRepose\group_theory\Debug\Group_theory.e...
a_{1}^{3}a_{3}^{4}a_{2}^{2}a_{3}a_{1}^{-1}
b_{1}^{4}b_{3}^{4}b_{2}^{2}b_{3}b_{1}^{-3}
gr.WordIsEqualTo1(a1_ge): 0
gr.WordIsEqualTo1(a2_ge): 0
a_{1}^{2}a_{3}^{4}a_{2}^{2}a_{3}a_{1}^{-1}
b_{1}^{3}b_{3}^{4}b_{2}^{2}b_{3}b_{1}^{-3}
fm.WordIsEqualTo1(a3a4_fmge): 0
Слово циклически приведено
Слово циклически приведено
fm.WordsAreConjugate(a3_fmge, a4_fmge): 0
Для продолжения нажмите любую клавишу . . .
```

Список литературы

1. Линдон Р., Шупп П. Комбинаторная теория групп. М.: Мир, 1980.
2. Магнус В., Каррас А., Солитэр Д. Комбинаторная теория групп. М.: Наука, 1974.
3. Фролов П. В. Программная реализация решения алгоритмических проблем для свободного произведения двух групп с объединённой подгруппой. ВКР бакалавра. Иваново: ИвГУ, 2019.

Приложение. (Листинг кода)

main.cpp

```
#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <locale>
#include <iostream>
#include "Word.h"
#include "FG_AbelianGroup.h"
#include "FG_AbelianGroupCSG.h"
#include "FCSG_FGAbelianGroup.h"
#include "ContainerAbelianGroup.h"
#include "CyclicSubGroupIzomorphizm.h"
#include "FM_Two_Groups.h"
#include "GroupElement.h"

using namespace std;

void print_word(const Word &w) {
    for (Word::const_iterator iter = w.begin(); iter != w.end(); iter++)
        cout << iter->GetValue() << ' ';
    cout << endl;
}

int main()
{
    setlocale(LC_CTYPE, "rus");

    {
        FG_AbelianGroup::FG_AbelianGroupElement order1(4);
        order1[0] = 8;
        order1[1] = 8;
        order1[2] = 5;
        order1[3] = 7;

        FG_AbelianGroup gr1(order1, 'a');

        FG_AbelianGroup::FG_AbelianGroupElement order2(4);
        order2[0] = 8;
        order2[1] = 8;
        order2[2] = 5;
        order2[3] = 7;

        FG_AbelianGroup gr2(order2, 'b');

        string a1_input = "a_{1}^{3}a_{3}^{4}a_{2}^{2}a_{3}a_{1}^{-1}";
        cout << a1_input << endl;
        TexString a1_Tex_input(a1_input);
        Word a1_word = ConvertingTexToWord(a1_Tex_input);
        GroupElement a1_ge = GroupElement(a1_word, &gr1);

        FCSG_FGAbelianGroup sgp1(a1_ge);

        string a2_input = "b_{1}^{4}b_{3}^{4}b_{2}^{2}b_{3}b_{1}^{-3}";
        cout << a2_input << endl;
        TexString a2_Tex_input(a2_input);
        Word a2_word = ConvertingTexToWord(a2_Tex_input);
        GroupElement a2_ge = GroupElement(a2_word, &gr2);
```

```

FCSG_FGAbelianGroup sgp2(a2_ge);

cout << "gr.WordIsEqualTo1(a1_ge): " << gr1.WordIsEqualTo1(a1_ge) << endl;
cout << "gr.WordIsEqualTo1(a2_ge): " << gr2.WordIsEqualTo1(a2_ge) << endl;

CyclicSubGroupIzomorphism iz(&sgp1, &sgp2);

FM_Two_Groups fm(&iz);

string a3_input = "a_{1}^{2}a_{3}^{4}a_{2}^{2}a_{3}a_{1}^{-1}";
cout << a3_input << endl;
TexString a3_Tex_input(a3_input);
Word a3_word = ConvertingTexToWord(a3_Tex_input);
GroupElement a3_ge = GroupElement(a3_word, &gr1);

string a4_input = "b_{1}^{3}b_{3}^{4}b_{2}^{2}b_{3}b_{1}^{-3}";
cout << a4_input << endl;
TexString a4_Tex_input(a4_input);
Word a4_word = ConvertingTexToWord(a4_Tex_input);
GroupElement a4_ge = GroupElement(a4_word, &gr2);

GroupElement a3_fmge = fm.StandartImageOf(a3_ge);
GroupElement a4_fmge = fm.StandartImageOf(a4_ge);

GroupElement a3a4_fmge = a3_fmge * a4_fmge;

cout << "fm.WordIsEqualTo1(a3a4_fmge): " << fm.WordIsEqualTo1(a3a4_fmge) <<
endl;

cout << "fm.WordsAreConjugate(a3_fmge, a4_fmge): " <<
fm.WordsAreConjugate(a3_fmge, a4_fmge) << endl;

}

cout << endl;
system("pause");
return 0;

}

```

FM_Two_Groups.cpp

```

#include "FM_Two_Groups.h"
#include "SubGroup.h"
#include "FiniteSubGroup.h"

FM_Two_Groups::FM_Two_Groups(SubGroupIzomorphism *_phi) : phi(_phi)
{
    _max_units_number = GetA()->GetMask().GetUnitsNumber();
    if (GetB()->GetMask().GetUnitsNumber() > _max_units_number)
        _max_units_number = GetB()->GetMask().GetUnitsNumber();

    _coding_units = 2;
    _freeproduct_mask.GenerateMask(0, _max_units_number + _coding_units);
    _extraction_factor_number_mask.GenerateMask(_max_units_number, _coding_units);
}

Mask FM_Two_Groups::GetMask() const

```

```

{
    return _freeproduct_mask;
}

unsigned int FM_Two_Groups::GetMaxUnitsNumber() const
{
    return _max_units_number;
}

unsigned int FM_Two_Groups::GetCodingUnits() const
{
    return _coding_units;
}

bool FM_Two_Groups::IsContain(const Symbol& input_value) const
{
    try
    {
        const ContainerGroup *result = GetGroupAddress(input_value);
        return result->IsContain(input_value);
    }
    catch (GP_Exception())
    {
        return false;
    }
};

Bitset FM_Two_Groups::GetGroupNumber(const Symbol& input_value) const
{
    ID result = input_value;

    result = result & (_extraction_factor_number_mask);
    result = result >> _max_units_number;

    if (result.GetValue() - 1 > 1)
        throw GP_Exception();
    //return result.GetValue();
    return (Bitset)result.GetValue();
}

GroupElement FM_Two_Groups::StandartImageOf(const GroupElement& input_value) const
{
    Word buffer(input_value);
    Bitset group_id = 0;
    if (input_value.GetGroup() == GetA()) group_id = 1;
    else if (input_value.GetGroup() == GetB()) group_id = 2;
    else throw GP_Exception();

    for (auto i = buffer.begin(); i != buffer.end(); ++i)
    {
        ID temp = *i;
        temp = temp & (~(ID(Bitset(3)) << (this->GetMaxUnitsNumber())));
        temp = temp | (ID(group_id) << (this->GetMaxUnitsNumber()));
        *i = temp;
    }

    return GroupElement(buffer, this);
}

const ContainerGroup *FM_Two_Groups::GetA() const {
    return phi->GetIzomorphizmDomain()->GetGroup(); //возвращаем указатель на группу
phi, от поля domain
}

```



```

const ContainerGroup *FM_Two_Groups::GetB() const {
    return phi->GetIzomorphismImage()->GetGroup(); //возвращаем указатель на группу
    phi, от поля image
}
const ContainerGroup *FM_Two_Groups::GroupIDtoGroupPointer(Bitset id) const {
    if (id == 1) return GetA(); //если id=1, то A
    if (id == 2) return GetB(); //если id=2, то B
    throw GP_Exception(); //иначе исключение
}
Bitset FM_Two_Groups::GroupPointertoGroupID(const ContainerGroup *p) const {
    if (p == GetA()) return 1; //если p=A возвращаем 1
    if (p == GetB()) return 2; //если p=B возвращаем 2
    throw GP_Exception(); //иначе исключение
}
const ContainerGroup *FM_Two_Groups::GetGroupAddress(const Symbol& input_value) const{
    if (GetGroupNumber(input_value) == 1) return GetA(); //если номер группы=1 возвра-
щаем A
    if (GetGroupNumber(input_value) == 2) return GetB(); //если номер группы=2 возвра-
щаем B
    throw GP_Exception(); //иначе исключение
}

const ContainerGroup *FM_Two_Groups::GetMultiplier(const GroupElement &input_value) const
{
    return GetGroupAddress(*(input_value.begin())); //возвращаем адрес свободного мно-
жителя
}

GroupElement FM_Two_Groups::ListOfSyllablesToElement(const list<GroupElement> &l) const
{
    GroupElement t= *(l.begin()); //элемент t равный первому слогу
    for (list<GroupElement>::const_iterator iter = l.begin(); iter != l.end();
++iter) {
        t = t * *iter; //умножаем t на все элементы списка
    }
    return t; //получаем элемент
}

list<GroupElement> FM_Two_Groups::ListOfSyllablesOf(const GroupElement& input_value)
const {
    if (input_value.GetGroup() != this) throw GP_Exception(); //если группа не найдена,
то исключение
    list<GroupElement> syllables; //создаем список

    if (input_value.size() == 0)
        return syllables;

    auto word_iterator = input_value.begin();
    unsigned int current_group_id = this->GetGroupNumber(*word_iterator);

    auto _begin = word_iterator;
    auto _end = word_iterator;

    ++word_iterator;

    for (; word_iterator != input_value.end(); )
    {
        unsigned int new_group_id = this->GetGroupNumber(*word_iterator);

        if (current_group_id == new_group_id)
        {
            ++word_iterator;
        }
        else

```

```

        {
            _end = word_iterator;
            --_end;
            Word sub_word = input_value.GetSubWord(_begin, _end);
            GroupElement ge = StandartImageOf(GroupElement(sub_word,
GroupIDtoGroupPointer(current_group_id)));
            syllables.emplace_back(ge);

            _begin = word_iterator;
            current_group_id = new_group_id;

            ++word_iterator;
        }
    }
    _end = word_iterator;
    --_end;
    Word sub_word = input_value.GetSubWord(_begin, _end);
    GroupElement ge = StandartImageOf(GroupElement(sub_word,
GroupIDtoGroupPointer(current_group_id)));
    syllables.emplace_back(ge);

    return syllables;
}

list<GroupElement> FM_Two_Groups::ReducedFormOf(const GroupElement& _elem) const {
//алгоритм XI
    list<GroupElement> elem_syllables = ListOfSyllablesOf(_elem); // создаем список
    bool flag; //флажок
    do {
        if (elem_syllables.size() == 1) return elem_syllables; //если размер=1 воз-
вращаем список
        flag = false;
        for (list<GroupElement>::iterator iter = elem_syllables.begin(); iter !=
elem_syllables.end(); ++iter) {
            const ContainerGroup *syllable_group = GetMultiplier(*iter);
//создаем группу по её первому слогу
            GroupElement syllable_elem = GroupElement(*iter, syllable_group);
            if (syllable_group == GetA()) { //если A, то
                if (phi->GetIzomorphizmDomain()->IsContain(syllable_elem)) {
//если указатель на группу phi от поля domain, то
                    flag = true; // флаг в истину
                    *iter = StandartImageOf(phi->ImageOf(syllable_elem));
//преобразовываем итератор
                    list<GroupElement>::iterator itern = iter; itern++; //
создаем новый итератор равный текущему итератору
                    if (itern != elem_syllables.end()) { //если новый ите-
ратор указывает на конец списка, то
                        *iter = *iter * *itern; //старый итератор умно-
жаем на новый
                        elem_syllables.erase(itern); //стираем новый
итератор в списке
                    }
                    if (iter != elem_syllables.begin()) { // если итератор
указывает на начало списка, то
                        itern = iter; iter--; //шаг назад
                        *iter = *iter * *itern; //старый итератор умно-
жаем на новый
                        elem_syllables.erase(itern); //стираем новый
итератор в списке
                    }
                }
            }
        }
    }
    else if (syllable_group == GetB()) { //тоже самое для B
        if (phi->GetIzomorphizmImage()->IsContain(syllable_elem)) {

```

```

        flag = true;
        *iter = StandartImageOf(phi-
>preImageOf(syllable_elem));

        list<GroupElement>::iterator itern = iter; itern++;
        if (itern != elem_syllables.end()) {
            *iter = *iter * *itern;
            elem_syllables.erase(itern);
        }
        if (iter != elem_syllables.begin()) {
            itern = iter; iter--;
            *iter = *iter * *itern;
            elem_syllables.erase(itern);
        }
    }
} while (flag == true);
cout << "слово несократимо\n";

return elem_syllables; //возвращаем получившийся список
}

bool FM_Two_Groups::WordIsEqualTo1(const GroupElement &_elem) const { //CI
    list<GroupElement> elem_syllables = ReducedFormOf(_elem); //приводим к несократи-
мой форме
    if (elem_syllables.size() > 1) return false; //если размер больше 1, то false
    else return GetMultiplier(*(elem_syllables.begin()))->WordIsEqualTo1(
        GroupElement(*(elem_syllables.begin()), GetMultipli-
er(*(elem_syllables.begin()))))
        );
    //иначе возвращаем метод GetMultiplier от начала списка
}

list<GroupElement> FM_Two_Groups::CyclicallyReducedFormOf(const GroupElement& input_word)
const { //XII
    list<GroupElement> w = ReducedFormOf(input_word); //приводим к несократимой форме
    do {
        if ((w.size() % 2 == 0) || (w.size() == 1)) { //если слово чётно или равно 1
            cout << "Слово циклически приведено \n";
            return w;
        }
        else {
            list<GroupElement>::iterator iter = w.end(); //итератор на конец слова
            iter--; //шаг назад
            *(w.begin()) = *(w.begin()) * *iter; //умножаем начало на последний
            w.erase(iter); //стираем последний слог
            if (GetMultiplier(*(w.begin()))->WordIsEqualTo1(*(w.begin())))
                w.erase(w.begin()); //стираем начальный слог
        }
    } while (true);
}

bool FM_Two_Groups::WordsAreConjugate(const GroupElement &u, const GroupElement &v) const
{ //CII
    const FiniteSubGroup *H = dynamic_cast<const FiniteSubGroup*>(phi-
>GetIzomorphizmDomain()); //указатель на поле с подгруппой H
    const FiniteSubGroup *K = dynamic_cast<const FiniteSubGroup*>(phi-
>GetIzomorphizmImage()); //указатель на поле с подгруппой K
    if (H == 0 || K == 0) throw; //если подгруппы пустые, то исключение
    list<GroupElement> u1 = CyclicallyReducedFormOf(u); //приводим слова u и v к цик-
лически несократимому виду
    list<GroupElement> v1 = CyclicallyReducedFormOf(v);
    for (unsigned int i = 0; i < v1.size(); i++) { //цикл по v1

```

```

        GroupElement h = H->FirstElem(); //указатель на первый элемент
        do {
            GroupElement h1 = StandartImageOf(h); //приводим слово к нормальному
виду
            if (WordsAreEqual(h1.GetInverse()*ListOfSyllablesToElement(v1)*h1,
ListOfSyllablesToElement(u1))) return true;
            //если слова равны, то возвращаем правду
        } while (H->NextElem(h));
        v1.emplace_back(*(v1.begin())); //создаем объект в конце
        v1.erase(v1.begin()); //стираем начало
    }
    return false;
}

BufferTex FM_Two_Groups::ConvertingWordToElementRecord(const GroupElement& input_value)
const
{
    BufferTex Result;
    BufferTex Temp;

    GroupElement BufferWord = input_value;
    while (BufferWord.DeleteTrivials());
    list<GroupElement> syllables_words = ListOfSyllablesOf(BufferWord);

    auto it = Result.end();

    for (auto it_syllable : syllables_words)
    {
        //Word Subword = BufferWord.GetSubWord(it_syllable._begin,
it_syllable._end);
        Temp = it_syllable.GetGroup()->ConvertingWordToElementRecord(it_syllable);

        for (auto& it_temp : Temp)

            it_temp.index.push_back(GroupPointertoGroupID(GetMultiplier(it_syllable)));

        Result.splice(it, Temp);
        Temp.clear();
    }
    return Result;
}

```

FiniteSubGroup.h

```

#ifndef FINITESUBGROUP
#define FINITESUBGROUP
#include "SubGroup.h"
#include "GroupElement.h"

class FiniteSubGroup : virtual public SubGroup
{
public:
    FiniteSubGroup(const ContainerGroup*);
    virtual GroupElement FirstElem() const = 0; //чисто виртуальный метод для находке-
ния первого элемента
    virtual bool NextElem(GroupElement &u) const = 0; //чисто виртуальный метод для
нахождения следующего элемента
};

#endif //FINITESUBGROUP

```

CyclicSubGroup.h

```
#ifndef CYCLICSUBGROUP_H
#define CYCLICSUBGROUP_H
#include "SubGroup.h"

typedef int ElementDegree;
const ElementDegree UndefinedDegree = 1 << (sizeof(ElementDegree) * 8 - 1);
// для отображения неопределенной степени

class CyclicSubGroup : virtual public SubGroup
{
public:
    const ContainerGroup *Group; //поле
    const GroupElement Element; //порожд элем
public:
    CyclicSubGroup(const GroupElement &g); //конструктор
    const GroupElement &GetGenerating() const; //метод получения порождающего элемент
    bool IsContain(const GroupElement& input_value) const; //содержится ли элемент в
    подгруппе или нет.
    virtual const ElementDegree GetPower(const GroupElement &_elem, ElementDegree
    MaxDegree = UndefinedDegree) const; //находит степень в которую нужно возвести Element
    virtual const GroupElement powerElement(ElementDegree p) const; //метод для возве-
    дения порожд элемента в степень
};
#endif // CYCLICSUBGROUP_H
```

FiniteCyclicSubGroup.h

```
#ifndef FINITECYCLICSUBGROUP
#define FINITECYCLICSUBGROUP
#include "FiniteSubGroup.h"
#include "CyclicSubGroup.h"
#include "GroupElement.h"

class FiniteCyclicSubGroup : public FiniteSubGroup, virtual public CyclicSubGroup
{
public:
    FiniteCyclicSubGroup(const GroupElement& g);
    GroupElement FirstElem() const;
    bool NextElem(GroupElement &u) const;
};

#endif //FINITECYCLICSUBGROUP
```

FiniteCyclicSubGroup.cpp

```
#include "FiniteCyclicSubGroup.h"

FiniteCyclicSubGroup::FiniteCyclicSubGroup(const GroupElement& g) : Sub-
Group(g.GetGroup()), FiniteSubGroup(g.GetGroup()), CyclicSubGroup(g) {
} //конструктор

GroupElement FiniteCyclicSubGroup::FirstElem() const {
    return GetGenerating(); //возвращаемое значение
}
```

```

bool FiniteCyclicSubGroup::NextElem(GroupElement &u) const {
    u = u * GetGenerating();
    if (u.GetGroup()->WordIsEqualTo1(u)) return false;
    else return true;
}

```

FG_AbelianGroupCSG.h

```

#ifndef FG_ABELIANGROUPCSG_H
#define FG_ABELIANGROUPCSG_H
#include "CyclicSubGroup.h"
#include "FG_AbelianGroup.h"

class FG_AbelianGroupCSG : virtual public CyclicSubGroup
{
public:
    FG_AbelianGroupCSG(const GroupElement &g); //конструктор
    const ElementDegree GetPower(const GroupElement &_elem, ElementDegree MaxDegree =
UndefinedDegree) const; //находит степень в которую нужно возвести Element
};

#endif // FG_ABELIANGROUPCSG_H

```

FG_AbelianGroupCSG.cpp

```

#include "FG_AbelianGroupCSG.h"
FG_AbelianGroupCSG::FG_AbelianGroupCSG(const GroupElement &g) : SubGroup(g.GetGroup()),
CyclicSubGroup(g)
{
    if (dynamic_cast<const FG_AbelianGroup*>(g.GetGroup()) == 0) throw GP_Exception();
} //если не имеет подгруппы, то исключение

const ElementDegree FG_AbelianGroupCSG::GetPower(const GroupElement &_elem, ElementDegree
MaxDegree) const {
    if (_elem.GetGroup() != this->GetGroup()) throw GP_Exception();
    const FG_AbelianGroup *gr = dynamic_cast<const
FG_AbelianGroup*>(_elem.GetGroup());
    FG_AbelianGroup::FG_AbelianGroupElement u = gr->ToFG_AbelianGroupElement(_elem);
    FG_AbelianGroup::FG_AbelianGroupElement v = gr-
>ToFG_AbelianGroupElement(GetGenerating());
    FG_AbelianGroup::FG_AbelianGroupElement order = (dynamic_cast<const
FG_AbelianGroup*>(this->GetGroup()))->GetOrders(); //?
    ElementDegree result = UndefinedDegree;
    for (unsigned int i = 0; i < order.size(); i++) {
        if (order[i] == 0) {
            if (v[i] == 0 && u[i] != 0) return UndefinedDegree;
            if (v[i] != 0 && u[i] != 0)
                if (u[i] % v[i] != 0) return UndefinedDegree;
            else
                if (result == UndefinedDegree) result = u[i] / v[i];
                else if (result != UndefinedDegree && result != (u[i] /
v[i])) return UndefinedDegree;
            if (v[i] != 0 && u[i] == 0)
                if (result == UndefinedDegree) result = 0;
                else if (result != 0) return UndefinedDegree;
        }
    }
    if (result == UndefinedDegree) {
        ElementDegree d = 1;
        for (unsigned int i = 0; i < order.size(); i++)

```

```

        if (order[i] != 0) d *= order[i];
    for (unsigned int p = 0; p < d; p++) {
        FG_AbelianGroup::FG_AbelianGroupElement vp;
        for (unsigned int i = 0; i < vp.size(); i++) {
            vp[i] = v[i] * p;
            if (order[i] != 0) vp[i] %= order[i];
        }
        bool res = true;
        for (unsigned int i = 0; i < vp.size(); i++)
            if (vp[i] != u[i]) res = false;
        if (res) return p;
    }
    //return CyclicSubGroup::GetPower(_elem, d);
}
for (unsigned int i = 0; i < order.size(); i++) {
    ElementDegree d = v[i] * result - u[i];
    d = d < 0 ? -d : d;
    if ((d % order[i]) != 0) return UndefinedDegree;
}
}

```

FCSG_FGAbelianGroup.cpp

```

#include "FCSG_FGAbelianGroup.h"
FCSG_FGAbelianGroup::FCSG_FGAbelianGroup(const GroupElement &g)
    : SubGroup(g.GetGroup()), CyclicSubGroup(g), FiniteCyclicSubGroup(g),
  FG_AbelianGroupCSG(g) {} //конструктор

```