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 |  |

Иваново, 2019

**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 ).

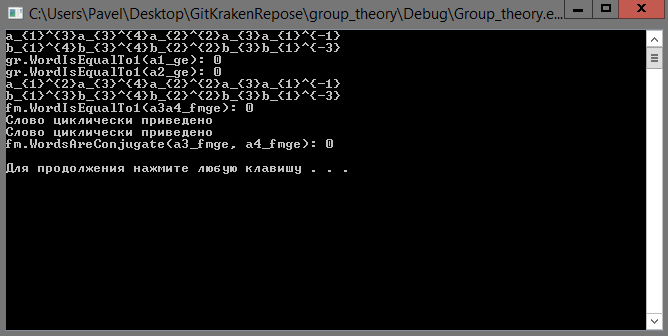
**§1. Рефакторинг кода и результат.**

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.



**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 <clocale>

#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;

CyclicSubGroupIzomorphizm 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(SubGroupIzomorphizm \*\_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->GetIzomorphizmImage()->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 А

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 возвращаем А

if (GetGroupNumber(input\_value) == 2) return GetB(); //если номер группы=2 возвращаем В

throw GP\_Exception(); //иначе исключение

}

const ContainerGroup \*FM\_Two\_Groups::GetMultiplier(const GroupElement &input\_value) const {

return GetGroupAddress(\*(input\_value.begin())); //возвращаем адрес свободного множителя

}

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> syliables; //создаем список

if (input\_value.size() == 0)

return syliables;

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)));

syliables.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)));

syliables.emplace\_back(ge);

return syliables;

}

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 возвращаем 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 итератор указывает на конец списка, то

\*iter = \*iter \* \*itern; //старый итератор умножаем 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 умножаем 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 несократимой 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()), GetMultiplier(\*(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 четно 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 слог

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 циклически 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 нахождения 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 возведения 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) : SubGroup(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