|  |  |
| --- | --- |
| МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ  РОССИЙСКОЙ ФЕДЕРАЦИИ  Федеральное государственное автономное образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | |
| *Кафедра математического обеспечения  вычислительных систем* | |
|  | |
| ОТЧЕТ  по дисциплине Формальный грамматики и методы трансляции  «Компилятор для Pascal» | |
| Работу выполнил студент группы *ПМИ-1* *3* курса механико-математического факультета  Понькин Н. А.  16 декабря 2022г. | Принял:  ассистент каф. МОВС  Пономарев Ф.А.  “\_\_\_\_\_” 2022 г. |
| Пермь 2022 | |

Модуль ввода-вывода

Описание

Необходимо разработать модуль, который будет выдавать лексемы из входного потока данных (исходный текст программы на языке Pascal), а также выводить семантические ошибки, которые встретились по ходу анализа программы.

Проектирование

Реализуем класс InOutModule:

Поля:

* Позиция лексемы,
* Текущая строка программы,
* Текущая лексема,
* Массив, содержащий строки программы (для вывода ошибок),
* Флаг, который > 0, если текущая лексема является частью комментария,
* Флаг, который > 0, если текущая лексема является частью строковой константы.

Методы:

* Получение следующей литеры с позицией,
* Вывод ошибок.

Класс ErrorHandler:

Поля:

* Вектор с ошибками (ошибка – класс, содержащий позицию ошибки и строку с информацией о ней).

Методы:

* Вернуть вектор ошибок
* Вернуть количество ошибок
* Добавление новой ошибки в вектор

Вывод семантических ошибок производится после анализа всего текста программы, ошибки перед выводом сортируются по позиции (строка, позиция в строке).

Реализация

class TextPos

{

public:

int line\_number, char\_number;

TextPos();

TextPos(int line\_number\_, int char\_number\_);

};

bool operator==(const TextPos a, const TextPos b);

bool operator!=(const TextPos a, const TextPos b);

class InOutModule

{

public:

InOutModule(const char\* In, const char\* Out = "");

std::pair<std::string, TextPos> GetNextLex();

bool ready;

void print\_semantic\_errors(ErrorHandler h);

private:

std::string s, cur;

std::vector<std::string> strs;

TextPos pos;

int idx, comment, quotes;

};

class Error

{

public:

std::string info;

TextPos pos;

Error(std::string info, TextPos pos);

};

class ErrorHandler

{

private:

std::vector<Error> errors;

public:

ErrorHandler() {};

void add\_error(std::string info, TextPos pos);

std::vector<Error> get\_errors();

int get\_errors\_count();

};

В функции GetNextLex происходит парс строк на лексемы. Вот, например, парс строковых констант:

if (s[idx] == '"') {

if (quotes == 0) {

if (cur != "") {

auto ans = std::make\_pair(cur, pos);

cur = "";

return ans;

}

pos.char\_number = idx;

cur = std::string(1, s[idx]);

quotes = 1;

}

else {

quotes = 0;

cur += s[idx];

auto ans = std::make\_pair(cur, pos);

cur = "";

idx++;

return ans;

}

continue;

}

if (quotes) {

cur += s[idx];

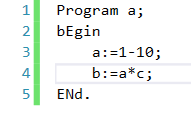
continue;

}

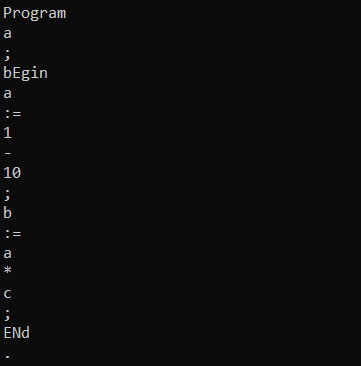
Тестирование

Тестируем парс программы на лексемы.

Программа:



Лексемы:



Лексический анализатор

Описание

Необходимо разработать модуль, который разбивает программу на токены.

Проектирование

Разработаем классы CLexer (Лексический анализатор), CToken (Токен), CVariant (Константа).

Класс CLexer:

Поля:

* Указатель на модуль ввода-вывода

Методы:

* Получение следующего токена

Класс CToken:

Поля:

* Позиция токена
* Тип токена (enum)

Методы:

* Получения строкового представления токена (лексемы)

От класса CToken наследуются классы CConstToken, CIdentToken, CKeyWordToken, COperatorToken, которые содержат непосредственное значение токена (указатель на экземпляр класса CVariant, значение идентификатора, тип ключевого слова - enum, тип оператора – enum, соответственно).

Класс CVariant содержит тип строковой константы (enum).

От класса CVariant наследуются классы CIntVariant, CFloatVariant, CStringVariant, CBoolVariant. Эти классы содержат непосредственное значение константы.

В данном модуле также реализуем функцию проверки, можно ли свести один тип к другому (понадобится в следующем модуле).

Реализация

class CLexer

{

public:

std::unique\_ptr<InOutModule> InOut;

CLexer(const char\* In, const char\* Out = "");

CToken\* GetNextToken();

void test();

};

enum TokenTp { Constant, Identifier, KeyWord, Operator };

enum VarType { DefaultType, IntegerType, FloatType, StringType, BoolType };

enum KeyWordType {

ProgramKW, VarKW, IfKW, ConstKW, ThenKW,

ElseKW, ForKW, WhileKW, DoKW, BeginKW, EndKW,

ReadKW, WriteKW, IntegerKW, FloatKW, StringKW, BoolKW, ReadlnKW, WritelnKW

};

enum OperatorType {

OType1, OType2, OType3, OType4, OType5, OType6, OType7,

OType8, OType9, OType10, OType11, OType12, OType13, OType14,

OType15, OType16, OType17, OType18, OType19, OType20, OType21,

OType22

};

extern std::map<std::string, OperatorType> from\_str\_to\_operator;

extern std::map<OperatorType, std::string> from\_operator\_to\_str;

extern std::map<std::string, KeyWordType> from\_str\_to\_kw;

extern std::map<KeyWordType, std::string> from\_kw\_to\_str;

std::string make\_low(std::string a);

bool can\_cast(VarType from, VarType to);

class CVariant;

class CToken

{

public:

TokenTp tt;

TextPos pos;

CToken();

CToken(TokenTp tt, TextPos pos);

virtual std::string ToString() = 0;

~CToken() {};

};

class CConstToken : public CToken

{

public:

std::unique\_ptr<CVariant> value;

CConstToken();

CConstToken(std::string lexem);

CConstToken(TextPos position, std::string lexem);

std::string ToString() override;

~CConstToken() {}

};

class CIdentToken : public CToken

{

public:

std::string value;

CIdentToken();

CIdentToken(std::string lexem);

CIdentToken(TextPos position, std::string lexem);

std::string ToString() override;

};

class CKeyWordToken : public CToken

{

public:

KeyWordType value;

CKeyWordToken(std::string lexem);

CKeyWordToken(TextPos position, std::string lexem);

std::string ToString() override;

};

class COperatorToken : public CToken

{

public:

OperatorType value;

COperatorToken();

COperatorToken(std::string lexem);

COperatorToken(TextPos position, std::string lexem);

std::string ToString() override;

};

class CVariant

{

public:

VarType vt;

CVariant();

CVariant(VarType vt);

virtual std::string ToString() = 0;

~CVariant() {}

};

class CIntVariant : public CVariant

{

public:

int source;

CIntVariant(int source);

std::string ToString() override;

};

class CFloatVariant : public CVariant

{

public:

float source;

CFloatVariant(float source);

std::string ToString() override;

};

class CStringVariant : public CVariant

{

public:

std::string source;

CStringVariant(std::string source);

std::string ToString() override;

};

class CBoolVariant : public CVariant

{

public:

bool source;

CBoolVariant(bool source);

std::string ToString() override;

};

В функции GetNextToken классификация лексемы на тип токена, после чего вызывается конструктор соответствующего наследника класса CToken. Вот, к примеру, конструктор класса CConstToken:

CConstToken::CConstToken(TextPos position, std::string lexem) : CToken(Constant, position){

tt = Constant;

pos = position;

int cnt\_c = 0, cnt\_p = 0;

bool is\_num = lexem[0] >= '0' && lexem[0] <= '9';

for (auto symb : lexem) {

if (symb >= '0' && symb <= '9')

cnt\_c++;

else if (symb == '.')

cnt\_p++;

}

if (cnt\_p == 0 && is\_num && cnt\_c == lexem.length()) {

value = std::make\_unique<CIntVariant>(stoi(lexem));

return;

}

else if (cnt\_p == 1 && is\_num && cnt\_c == lexem.length() - 1) {

value = std::make\_unique<CFloatVariant>(stof(lexem));

return;

}

std::string help = make\_low(lexem);

if (help == "true" || help == "false") {

value = std::make\_unique<CBoolVariant>(help == "true");

}

else

value = std::make\_unique<CStringVariant>(lexem);

}

Функция проверки сводимости типов:

bool can\_cast(VarType from, VarType to) {

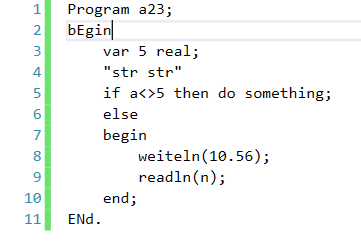
return to == DefaultType || from == to || from == IntegerType && to == FloatType

|| from == BoolType && to != StringType;

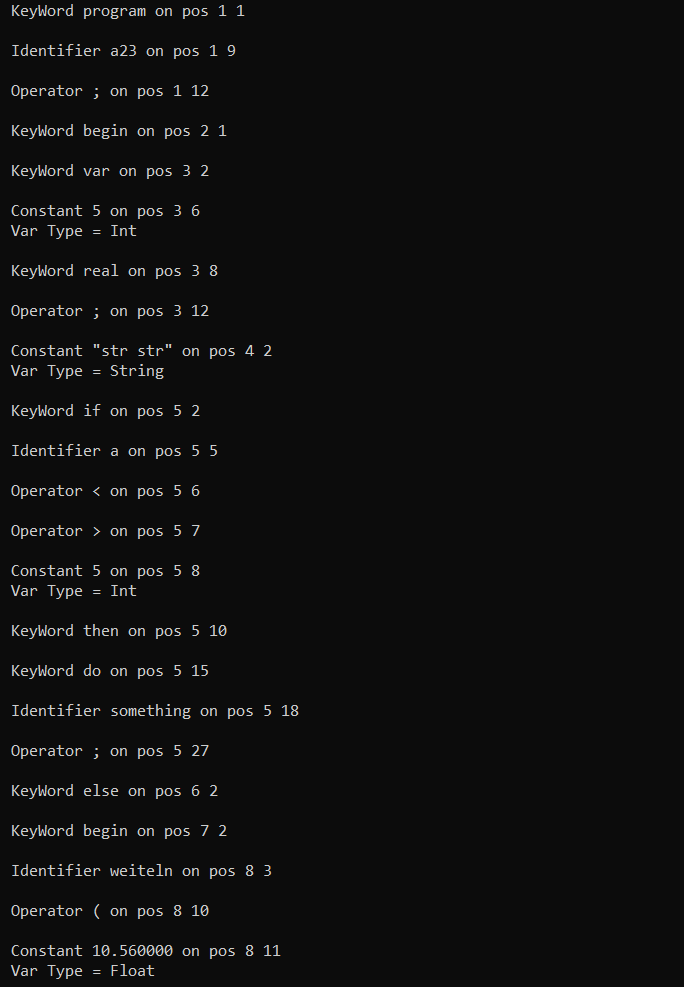
}

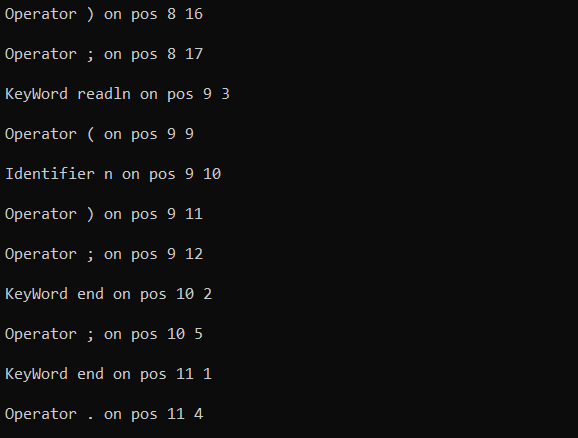
Тестирование

Программа:



Вывод токенов:





Синтаксический и семантический анализ

Описание

Необходимо разработать модуль, который по исходной программе проводит синтаксический анализ (преобразует в структурированный формат) и семантический анализ (семантические правила языка), а также добавляет семантические ошибки в менеджер ошибок.

Реализация

Реализуем класс CCompilier, в котором и будет происходить анализ последовательности токенов.

Поля:

* Указатель на лексический анализатор
* Указатель на текущий токен
* Указатель на менеджер ошибок
* Хэш таблица «идентификатор – тип данных»

Методы:

* Запуск анализа
* Проверка текущего токена на соответствие необходимому токену, запрос следующего токена в случае соответствия
* Проверка текущего оператора на соответствие необходимому оператору, запрос следующего токена в случае соответствия
* Проверка текущего ключевого слова на соответствие необходимому ключевому слову, запрос следующего токена в случае соответствия
* Проверка текущего типа данных константы на соответствие необходимому типу данных константы, запрос следующего токена в случае соответствия
* Добавление идентификатора с его типом в хэш таблицу «идентификатор – тип данных»
* Сведение двух типов
* Перевод из std::string в String^
* Перевод из VarType (enum) в Type^
* Перевод из Type^ в VarType

Далее рекурсивно запускаемые методы в соответствии с БНФ:

* Заголовок и точка
* Основной блок
* Раздел переменных
* Раздел однотипных переменных
* Составной оператор
* Оператор
* Простой оператор
* Сложный оператор
* Оператор присваивания
* Выражение
* Простое выражение
* Слагаемое
* Множитель
* Условный оператор
* Цикл с предусловием
* Вывод на консоль
* Ввод с консоли

Синтаксический анализ будет проходить по БНФ для языка Pascal, которые были взяты из книги: «Разработка ПАСКАЛЬ-компилятора» Л. А. Залогова.

Будем рекурсивно ходить по БНФ по соответствующим методам.

Семантический анализ будет заключаться в заполнении хэш-таблиц «идентификатор – тип данных» в разделе переменных, а затем проверке, были ли объявлены используемые в программе идентификаторы и сводимы ли типы в различных выражениях и оператор.

Реализация

ref class CCompilier

{

private:

CToken\* token;

ErrorHandler\* handler;

Dictionary<String^, Type^> vars;

String^ convert\_to\_mstr(std::string s);

Type^ GetType(VarType v);

VarType GetType(Type^ v);

CToken\* GetNextToken();

void Accept(TokenTp expected\_token\_type);

void Accept(OperatorType expected\_operator);

void Accept(KeyWordType expected\_keyword);

void Accept(VarType expected\_var\_type);

void Add\_var(std::string& name, VarType type);

VarType Derive(VarType left, VarType right, OperatorType last\_operation, TextPos pos\_for\_error);

// БНФ

void ProgramBlock(); //Заголовок и точка

void MainBlock(); //Программа

// void ConstBlock(); //Раздел констант

void VarBlock(); //Раздел переменных

void SimilarVars(); //Однотипные переменных

void CompositeOperatorBlock(); //Составной оператор (begin, end)

void OperatorBlock(); //Оператор

void SimpleOperatorBlock(); //Простой оператор

void ComplicatedOperatorBlock(); //Сложный оператор

void AssignOperatorBlock(); //Оператор присваивания

VarType Expression(); //Выражение

VarType SimpleExpression(); //Простое выражение

VarType Term(); //Слагаемое

VarType Multiplier(); //Множитель

void ConditionalOperatorBlock(); //Условный оператор

void WhileBlock(); //Цикл с предусловием

void WriteLnBlock(); //writeln - вывод на консоль

void ReadLnBlock(); //readln - ввод с консоли

public:

CLexer\* lexer = nullptr;

CCompilier(const char\* In, const char\* Out);

void Run();

};

Метод проверки соответствия текущего типа токена необходимому:

void CCompilier::Accept(TokenTp expected\_token\_type) {

if (token->tt != expected\_token\_type)

throw CTokenExpectedException(expected\_token\_type, token->pos);

token = GetNextToken();

}

Раздел однотипных переменных (БНФ):

void CCompilier::SimilarVars() {

std::vector<std::string> vs;

vs.push\_back(token->ToString());

Accept(Identifier);

while (token->ToString() == ",") {

Accept(from\_str\_to\_operator[","]);

vs.push\_back(token->ToString());

Accept(Identifier);

}

Accept(from\_str\_to\_operator[":"]);

if (token->tt != KeyWord) {

Accept(KeyWord);

return;

}

auto our = static\_cast<CKeyWordToken\*>(token);

auto my\_type\_kw = our->value;

VarType my\_type;

switch (my\_type\_kw)

{

case IntegerKW:

my\_type = IntegerType;

break;

case FloatKW:

my\_type = FloatType;

break;

case StringKW:

my\_type = StringType;

break;

case BoolKW:

my\_type = BoolType;

break;

default:

my\_type = DefaultType;

break;

}

Accept(my\_type\_kw);

for (auto v : vs) {

Add\_var(v, my\_type);

}

}

БНФ оператора присваивания с проверками соответствия типов:

void CCompilier::AssignOperatorBlock() {

TextPos ident\_pos = token->pos;

std::string ident = token->ToString();

Accept(Identifier);

TextPos op\_pos = token->pos;

Accept(from\_str\_to\_operator[":="]);

VarType expr\_type = Expression();

if (!vars.ContainsKey(convert\_to\_mstr(ident))) {

std::string error\_text = "Переменная не была объявлена";

handler->add\_error(error\_text, ident\_pos);

}

else if (!can\_cast(expr\_type, GetType(vars[convert\_to\_mstr(ident)]))) {

std::string error\_text = "Вычисленное выражение и переменная имеют разные типы";

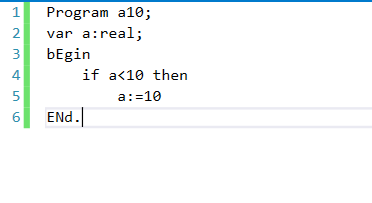
handler->add\_error(error\_text, op\_pos);

}

}

Тестирование

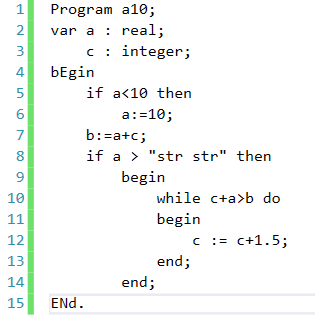
Программа с синтаксической ошибкой:



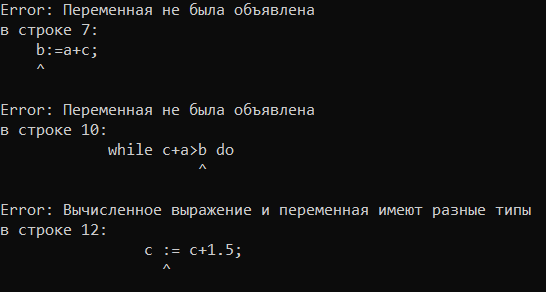
Вывод компилятора:



Программа с семантическими ошибками:



Вывод компилятора:



Генератор кода

Описание

Необходимо написать модуль, который переводит в IL код текст исходной программы.

Проектирование

Для реализации генерации кода добавим в класс CCompilier:

Поля:

* IlGenerator
* Хэш-таблица «идентификатор – LocalBuilder переменной»

Методы:

* Добавить операцию на стек
* Поменять знак последнего значения
* Загрузить значение переменной
* Вызвать ввод с консоли
* Вызвать вывод на консоль

Во время синтаксического и семантического анализа будем добавлять инструкции на стек.

Менеджер ошибок используется для того, чтобы после анализа посмотреть, есть ли ошибки в программе, если есть, то компиляцию надо закончить.

Реализация

Итоговая сигнатура CCompilier:

ref class CCompilier

{

private:

ILGenerator^ Gen;

Dictionary<String^, LocalBuilder^> vars\_lb;

Dictionary<String^, Type^> vars;

CToken\* token;

ErrorHandler\* handler;

void emit\_op(OperatorType op);

String^ convert\_to\_mstr(std::string s);

Type^ GetType(VarType v);

VarType GetType(Type^ v);

CToken\* GetNextToken();

void ChangeSign(System::Type^ type);

void LoadValue(CToken\* t);

void DoWriteLn(Type^ t);

void DoReadLn(CToken\* tk);

void Accept(TokenTp expected\_token\_type);

void Accept(OperatorType expected\_operator);

void Accept(KeyWordType expected\_keyword);

void Accept(VarType expected\_var\_type);

void Add\_var(std::string& name, VarType type);

VarType Derive(VarType left, VarType right, OperatorType last\_operation, TextPos pos\_for\_error);

// БНФ

void ProgramBlock(); //Заголовок и точка

void MainBlock(); //Программа

// void ConstBlock(); //Раздел констант

void VarBlock(); //Раздел переменных

void SimilarVars(); //Однотипные переменных

void CompositeOperatorBlock(); //Составной оператор (begin, end)

void OperatorBlock(); //Оператор

void SimpleOperatorBlock(); //Простой оператор

void ComplicatedOperatorBlock(); //Сложный оператор

void AssignOperatorBlock(); //Оператор присваивания

VarType Expression(); //Выражение

VarType SimpleExpression(); //Простое выражение

VarType Term(); //Слагаемое

VarType Multiplier(); //Множитель

void ConditionalOperatorBlock(); //Условный оператор

void WhileBlock(); //Цикл с предусловием

void WriteLnBlock(); //writeln - вывод на консоль

void ReadLnBlock(); //readln - ввод с консоли

public:

CLexer\* lexer = nullptr;

CCompilier(const char\* In, const char\* Out);

void Run();

};

Загрузка значения:

void CCompilier::LoadValue(CToken\* t) {

if (t->tt != Constant) {

return;

}

auto our = static\_cast<CConstToken\*>(token);

if (our->value->vt == IntegerType) {

Gen->Emit(OpCodes::Ldc\_I4, std::stoi(our->ToString()));

}

else if (our->value->vt == FloatType) {

Gen->Emit(OpCodes::Ldc\_R8, std::stod(our->ToString()));

}

else if (our->value->vt == StringType) {

Gen->Emit(OpCodes::Ldstr, convert\_to\_mstr(our->ToString()));

}

else if (our->value->vt == BoolType) {

Gen->Emit(OpCodes::Ldc\_I4, our->ToString() == "true");

}

}

Вызов readln:

void CCompilier::DoReadLn(CToken\* tk) {

String^ val = convert\_to\_mstr(tk->ToString());

Type^ type = vars[val];

LocalBuilder^ lb = vars\_lb[val];

MethodInfo^ readln\_method = Console::typeid->GetMethod("ReadLine", gcnew cli::array<Type^>{});

Gen->Emit(OpCodes::Call, readln\_method);

MethodInfo^ convert\_method = nullptr;

if (type == int::typeid)

{

convert\_method = Convert::typeid->GetMethod("ToInt32", gcnew cli::array<Type^>{std::string::typeid});

}

else if (type == double::typeid)

{

convert\_method = Convert::typeid->GetMethod("ToDouble", gcnew cli::array<Type^>{std::string::typeid});

}

else if (type == bool::typeid)

{

convert\_method = Convert::typeid->GetMethod("ToBoolean", gcnew cli::array<Type^>{std::string::typeid});

}

if (convert\_method != nullptr)

Gen->Emit(OpCodes::Call, convert\_method);

Gen->Emit(OpCodes::Stloc, lb);

}

Смена знака:

void CCompilier::ChangeSign(System::Type^ type) {

if (type == int::typeid) {

Gen->Emit(OpCodes::Ldc\_I4, -1);

Gen->Emit(OpCodes::Mul);

}

else if (type == double::typeid) {

Gen->Emit(OpCodes::Ldc\_R8, -1);

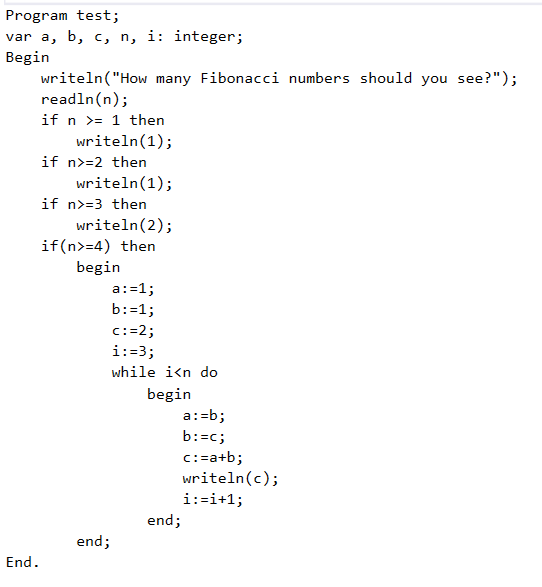
Gen->Emit(OpCodes::Mul);

}

}

Тестирование

Программа:



Сгенерировался exe файл, который можно запустить:

