**Содержание**

1. Цель 3
2. Синтаксис языка в форме Бэкуса-Наура 3-4
3. Используемые технологии 4
4. Структура анализатора 5
5. Таблица имен 6-7
6. Лексический анализатор 7-10
7. Синтаксический анализатор 11-12
8. Генератор кода 13-17
9. Интерфейсная логика 18
10. Тестирование 19-21
11. Вывод 22
12. Используемая литература 22

**Цель**

Целью курсового проектирования является разработка транслятора с ограниченного подмножества языка высокого уровня.

**Синтаксис языка в форме Бэкуса-Наура**

Вариант 15 (исходный вариант)

|  |
| --- |
| <***Программа***> ::= <Объявление переменных> <Описание вычислений><Оператор печати>  <***Описание*** ***вычислений***> ::= Begin <Список присваиваний> End  <***Объявление*** ***переменных***> ::= Logical <Список переменных>  <***Список*** ***переменных***> ::= <Идент> | <Идент> , <Список переменных>  <***Список*** ***присваиваний***>::= <Присваивание> |  <Присваивание> <Список присваиваний> <***Присваивание***> ::= <Идент> := <Выражение> ;  <***Выражение***> ::= <Ун.оп.><Подвыражение> | <Подвыражение> <***Подвыражение***> :: = ( <Выражение> ) | <Операнд> |  < Подвыражение > <Бин.оп.> <Подвыражение>  <***Ун.оп***.> ::= "!"  <***Бин.оп***.> ::= "&" | "|" | "^"  <***Операнд***> ::= <Идент> | <Const>  <***Идент***> ::= <Буква> <Идент> | <Буква>  <***Const***> ::= 0 | 1  <***Оператор*** ***печати***>::=Print <Идент> |

Внесенные изменения

|  |  |  |
| --- | --- | --- |
| ***Case*** <Выражение> ***of***  <Вариант>: <Оператор>  ***End*** | ***Until*** <Выражение> ***do***  <Список операторов>  ***EndUntil*** | Тип данных ***integer*** |

Результат

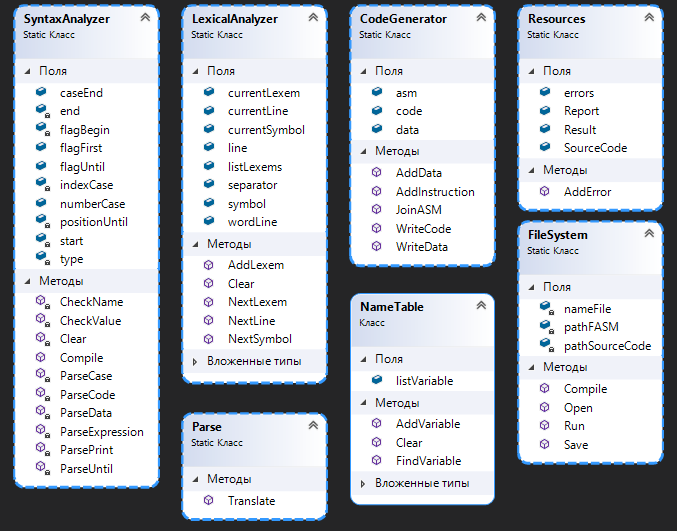
|  |
| --- |
| <***Программа***> ::= <Объявление переменных> <Описание вычислений><Оператор печати>  <***Описание*** ***вычислений***> ::= Begin <Список присваиваний> End  <***Объявление*** ***переменных***> ::= Logical <Список переменных> | Integer <Список переменных>  <***Список*** ***переменных***> ::= <Идент> | <Идент> , <Список переменных>  <***Список*** ***присваиваний***>::= <Присваивание> |  <Присваивание> <Список присваиваний> <***Присваивание***> ::= <Идент> := <Выражение> ;  <***Выражение***> ::= <Ун.оп.><Подвыражение> | <Подвыражение> <***Подвыражение***> :: = ( <Выражение> ) | <Операнд> |  < Подвыражение > <Бин.оп.> <Подвыражение>  <***Цикл***>::=Until <Выражение> do <Список операторов>  <***Список*** ***операторов***>::=<Оператор> |<Оператор> <Список операторов>  <***Оператор***>::= <Присваивание> | <Оператор печати> | <Цикл> | <Условие>  <***Условие***>::=Case <Выражение> of <Вариант>: < Список условий >  <***Список*** ***условий***>::=<Операнд>: <Присваивание> | <Вариант>: <Присваивание> <Список условий>  <***Ун.оп***.> ::= "!"  <***Бин.оп***.> ::= "&" | "|" | "^"|"+"|"-"|"\*"|"/"|  <***Операнд***> ::= <Идент> | <Const>  <***Идент***> ::= <Буква> <Идент> | <Буква>  <***Const***> ::= 0 | 1 | <Цифра> | <Const>  <***Оператор*** ***печати***>::=Print <Идент> |

**Используемые технологии**

Для реализации транслятора был выбран flat assembler. Он часто обновляется разработчиком и позволяет компилировать asm-код под 64 разрядную ОС Windows.

В качестве языка программирования используется c#. Разработка велась в среде Visual Studio.

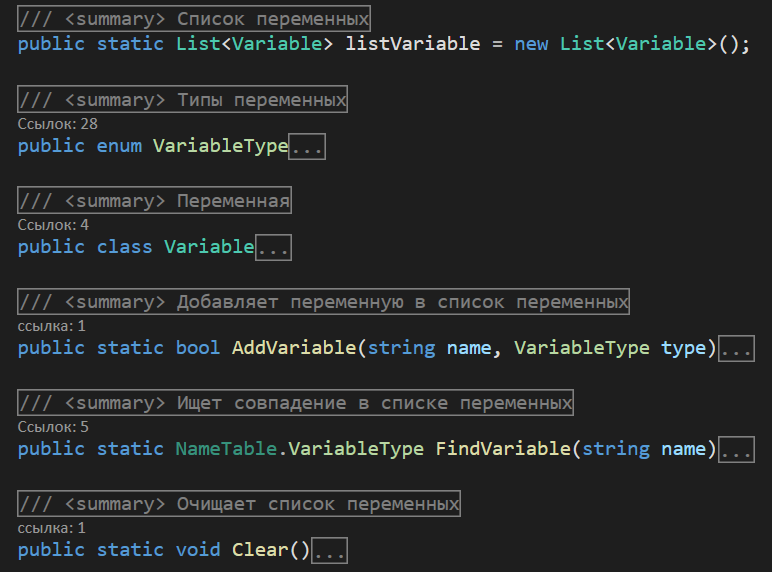
**Структура анализатора**



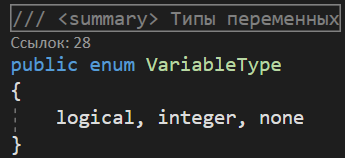
**Таблица имен**

Класс «TableName» служи для хранения и получения идентификаторов.

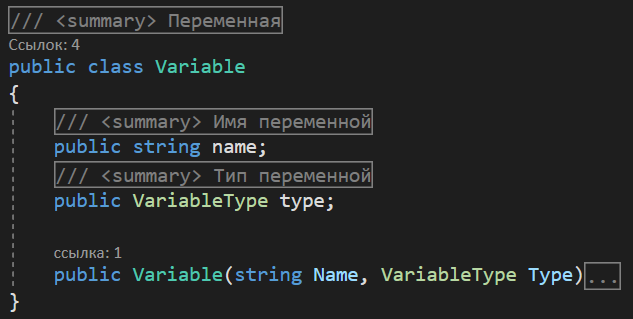
Структура класса

****

Типы переменных

****

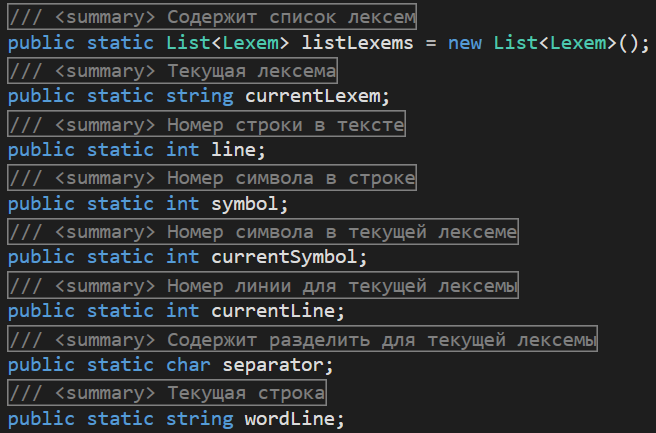
Структура класса «Variable»

****

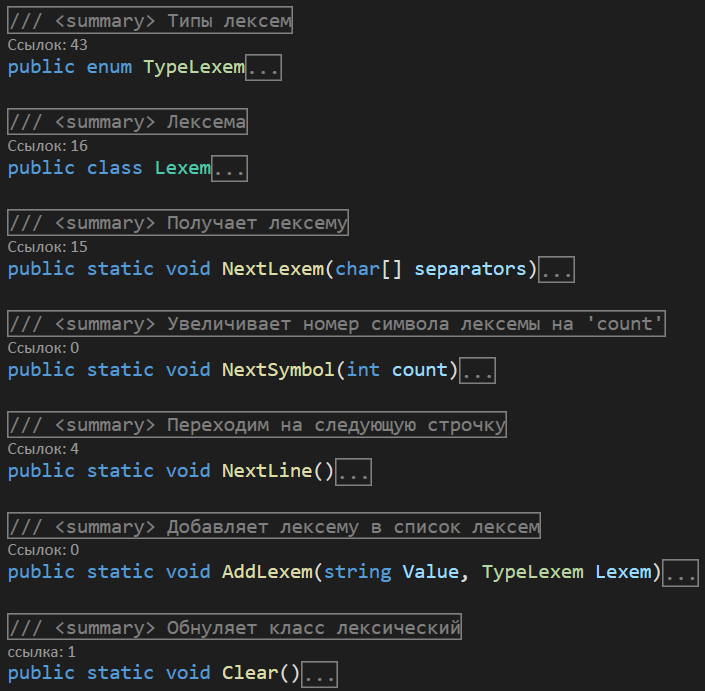
**Лексический анализатор**

Основной функцией является считывание лексемы из исходного файла.

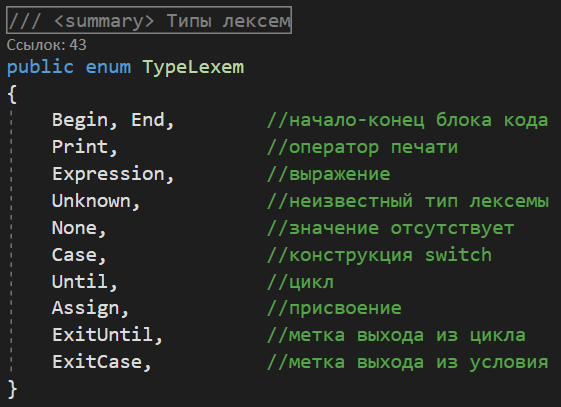
Список полей



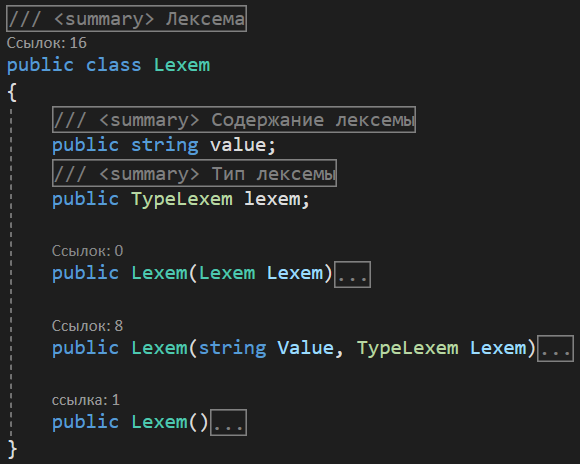
Структура методов



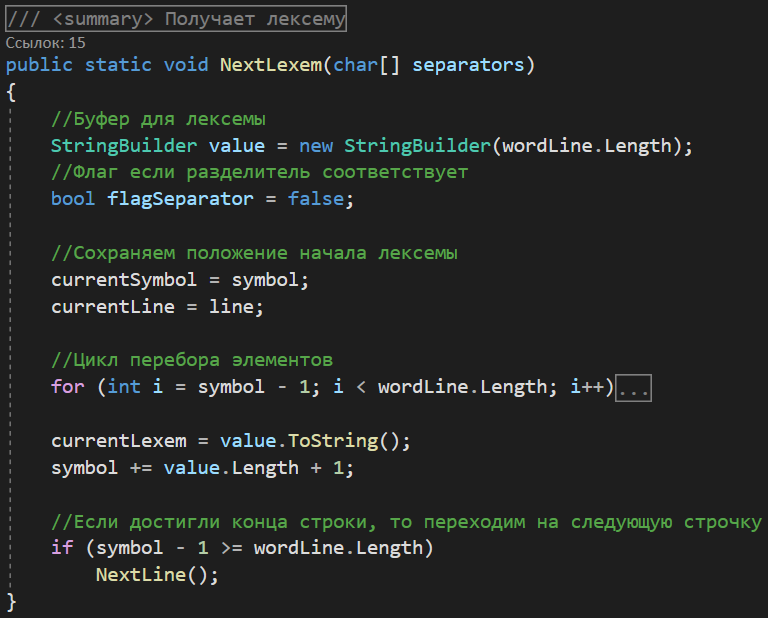
Типы лексем



Структура класса «Lexem»



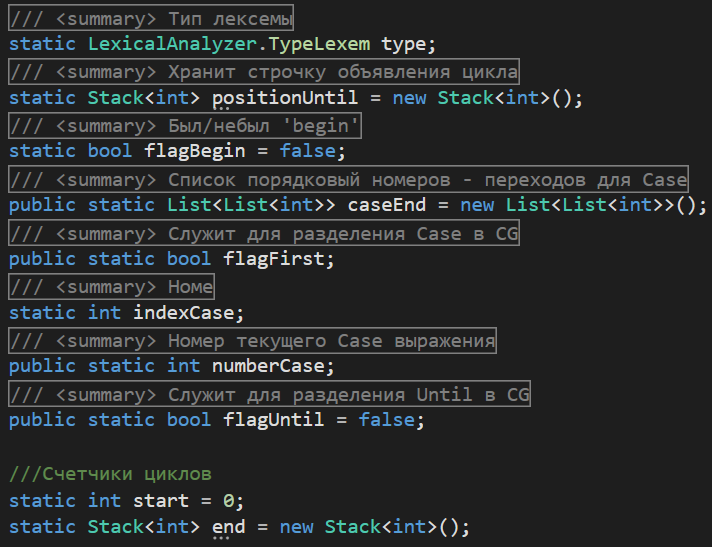
Основный метод лексического анализатора – получение лексемы



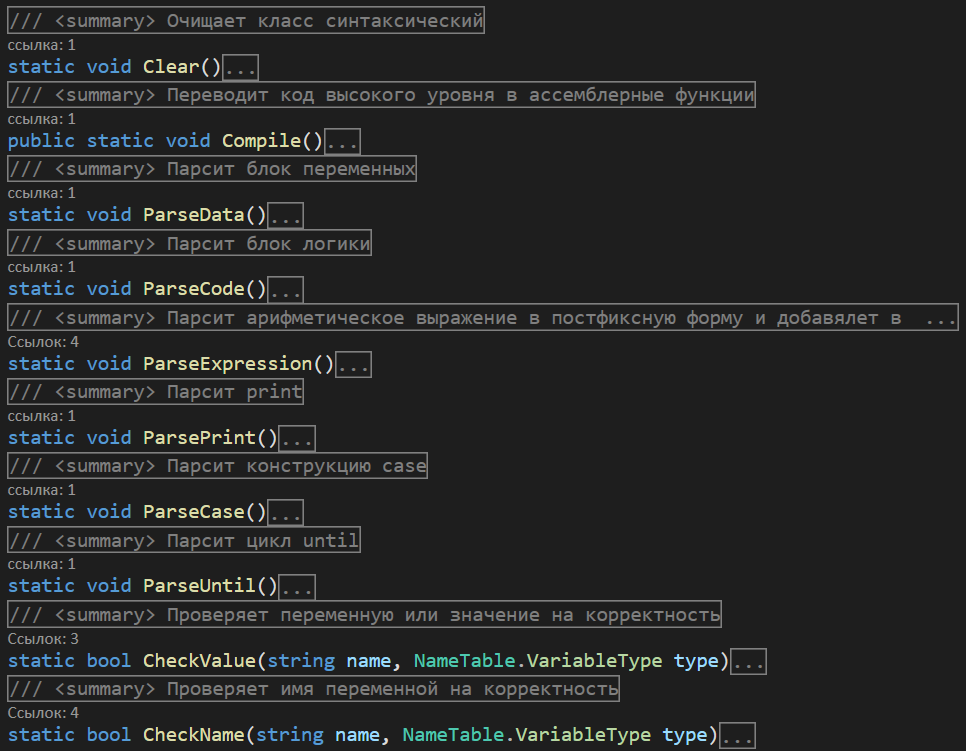
**Синтаксический анализатор**

Разбирает полученные лексемы и добавляет в список лексем для дальнейшей обработки.

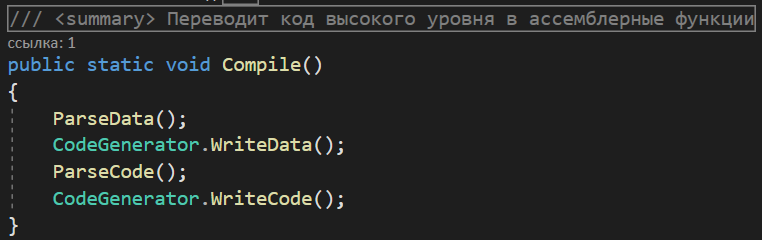
Основные поля класса



Методы класса

****

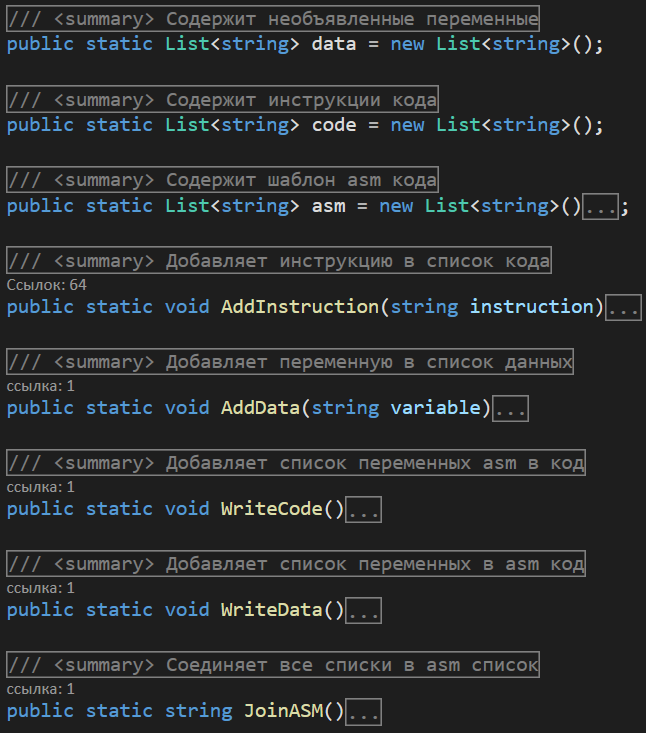
«Compile» - является основным методом так как запускает обработку блока данных и кода.

****

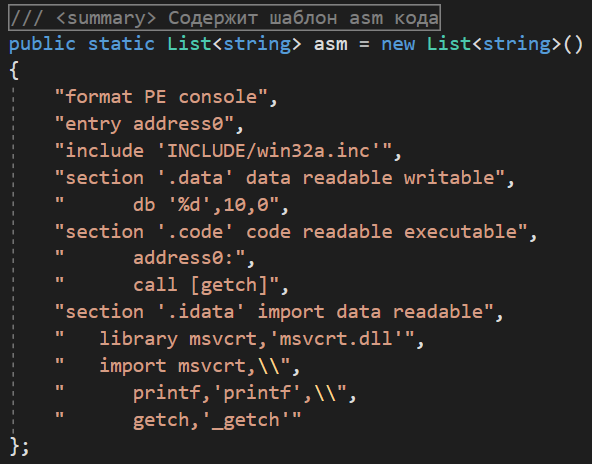
**Генератор кода**

Основная функция данного класса – перевести список лексем в asm код.

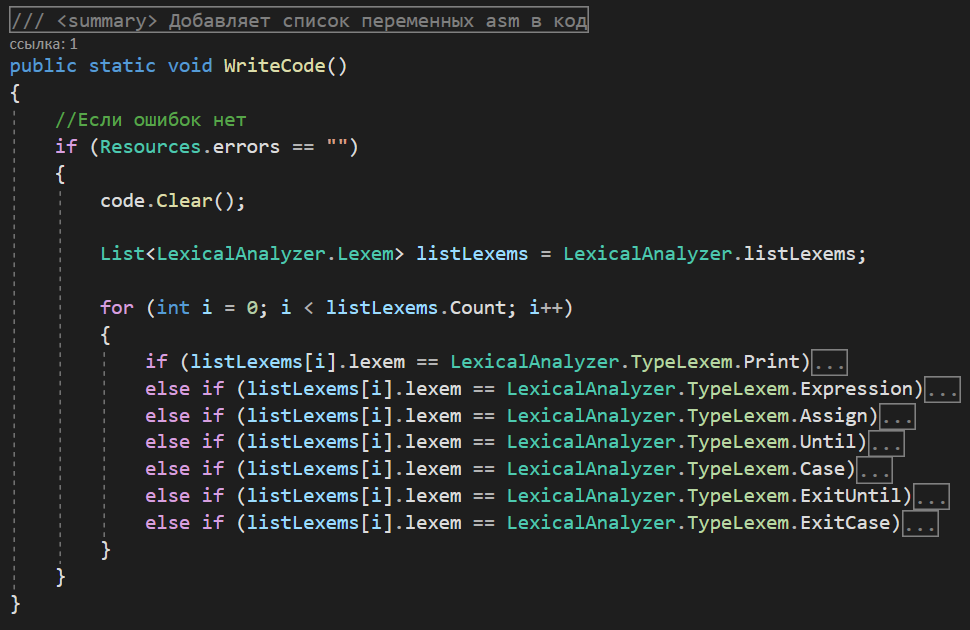
Структура класса «CodeGenerator»

****

Шаблон структуры asm файла

****

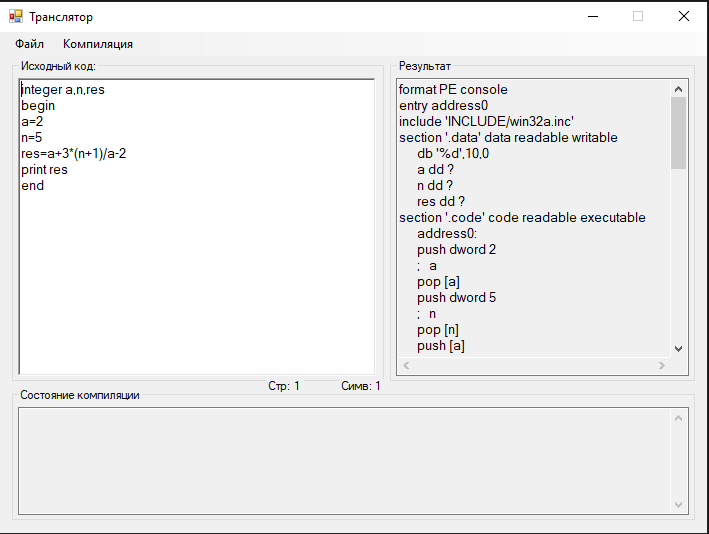
Метод обрабатывающий список лексем



|  |  |
| --- | --- |
| Лексемы | Asm инструкции |
| Print | AddInstruction("invoke printf,401000h," + listLexems[i].value); |
| Expression | switch (operations[j])  {  case "+":  AddInstruction("; + ");  AddInstruction("pop ebx");  AddInstruction("pop eax");  AddInstruction("add eax,ebx");  AddInstruction("push eax");  break;  case "-":  AddInstruction("; - ");  AddInstruction("pop ebx");  AddInstruction("pop eax");  AddInstruction("sub eax,ebx");  AddInstruction("push eax");  break;  case "/":  AddInstruction("; / ");  AddInstruction("pop ebx");  AddInstruction("pop eax");  AddInstruction("cdq");  AddInstruction("idiv ebx");  AddInstruction("push eax");  break;  case "\*":  AddInstruction("; \* ");  AddInstruction("pop eax");  AddInstruction("pop ebx");  AddInstruction("imul eax,ebx");  AddInstruction("push eax");  break;  case "^":  AddInstruction("; ^ ");  AddInstruction("pop ebx");  AddInstruction("pop eax");  AddInstruction("xor eax,ebx");  AddInstruction("push eax");  break;  case "%"://арифметическое отрицание  AddInstruction("; - ");  AddInstruction("pop eax");  AddInstruction("neg eax");  AddInstruction("push eax");  break;  case "!"://логическое отрицание  AddInstruction("; ! ");  AddInstruction("pop eax");  AddInstruction("btc eax,0");  AddInstruction("push eax");  break;  case "&":  AddInstruction("; & ");  AddInstruction("pop ebx");  AddInstruction("pop eax");  AddInstruction("and eax,ebx");  AddInstruction("push eax");  break;  case "|":  AddInstruction("; | ");  AddInstruction("pop ebx");  AddInstruction("pop eax");  AddInstruction("or eax,ebx");  AddInstruction("push eax");  break;  default:  if (NameTable.FindVariable(operations[j]) == NameTable.VariableType.none)  AddInstruction("push dword " + operations[j]);  else  AddInstruction("push [" + operations[j] + "]");  break; |
| Assign | AddInstruction("; " + listLexems[i].value + " ");  AddInstruction("pop [" + listLexems[i].value + "]"); |
| Until | if (SyntaxAnalyzer.flagUntil == false)  {  AddInstruction("; until ");  AddInstruction("startuntil" + listLexems[i].value + ":");  SyntaxAnalyzer.flagUntil = true;  }  else  {  AddInstruction("pop eax");  AddInstruction("cmp eax,dword 0");  AddInstruction("push eax");  AddInstruction("je enduntil" + listLexems[i].value);  SyntaxAnalyzer.flagUntil = false;  } |
| Case | AddInstruction("; case ");  if (SyntaxAnalyzer.flagFirst == false)  {  AddInstruction("pop eax");  SyntaxAnalyzer.flagFirst = true;  }  AddInstruction("cmp eax," + listLexems[i].value);  AddInstruction("jne endcase" + SyntaxAnalyzer.caseEnd[SyntaxAnalyzer.numberCase][0]); |
| ExitUntil | AddInstruction("; enduntil ");  AddInstruction("jmp startuntil" + listLexems[i].value);  AddInstruction("enduntil" + listLexems[i].value + ":"); |
| ExitCase | if (SyntaxAnalyzer.caseEnd[SyntaxAnalyzer.numberCase].Count != 1)  {  AddInstruction("jmp endcase" + SyntaxAnalyzer.caseEnd[SyntaxAnalyzer.numberCase]  [SyntaxAnalyzer.caseEnd[SyntaxAnalyzer.numberCase].Count - 1]);  }  AddInstruction("endcase" + SyntaxAnalyzer.caseEnd[SyntaxAnalyzer.numberCase][0] + ":");  SyntaxAnalyzer.caseEnd[SyntaxAnalyzer.numberCase].RemoveAt(0);  if (SyntaxAnalyzer.caseEnd[SyntaxAnalyzer.numberCase].Count == 0)  {  SyntaxAnalyzer.flagFirst = false;  SyntaxAnalyzer.numberCase++;  } |

**Интерфейсная логика**

Главная форма

****

На главной форме имеется три текстовых окошка. В «исходный код» вводятся данные пользователя. Под этим окошком показан номер строки и символа, на котором находится каретка.

В меню файл есть два пункта: «открыть файл» - считывает данные из выбранного пользователем файла, «сохранить файл» - сохраняет данные из окошка с исходным кодом в выбранный файл. Меню «компиляция» содержит три кнопки: «перевести» - переводит код из исходного окна в asm код и заносит в окошко с результатом, «выполнить» - делает перевод программы и её запуск (для корректного перевода необходимо чтобы путь к exe файлу был на латинском языке, файл FASM должен находиться вместе с запускаемым файлом иначе программа предложит выбрать путь к компилятору), «скомпилировать» - переводит asm код в exe файл, но не запускает его.

**Тестирование**

Написанный транслятор обрабатывает множество ошибок и указывает на их местоположение в тексте.

Проверка на ошибки

|  |  |
| --- | --- |
| Вход | Выход |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Проверка на вычисление выражения

|  |  |
| --- | --- |
| Вход | Выход |
|  | Результат    Проверка |
|  |  |

Пример работы циклов и условий

|  |  |
| --- | --- |
| Вход | Выход |
|  |  |

**Используемая литература**

1. Карпов, Ю. Г. Теория и технология программирования. Основы построения трансляторов / Ю.Г. Карпов. - М.: БХВ-Петербург, 2012.,

2. Постфиксная запись – префиксная форма [Электронный ресурс]. Режим доступа: https://steptosleep.ru/antananarivo-78/