Содержание

В данной работе описывается создание языка программирования, его синтаксис, грамматика и компилятора.

Постановка задачи

Необходимо разработать язык программирования и компилятор, транслирующий программу на этом языке в эквивалентную программу на языке C++.

Выбор языка программирования и средств разработки

В работе использовалось два языка программирования C++ и Python.

На языке C++ реализован компилятор, который обрабатывает входящий файл с программой и исполняет его, выводя в консоль ответ или ошибку, если в поданной программе такая содержится.

На языке Python разработан скрипт для перевода программы с русского языка на английский язык, для удобства обработки файла с программой.

Реализованный компилятор включает в себя:

Базовые типы данных включают в себя:

- целочисленный тип
- логический тип
- строковый тип

В качестве базовых операций для целого и строкового типов доступны:

- сложение
- вычитание
- умножение
- деление
- остаток от деления
- плюс равно
- минус равно
- сравнения
    - больше
    - меньше
    - равно
    - больше или равно
    - меньше или равно
    - неравно

В качестве базовых операторов реализованы:

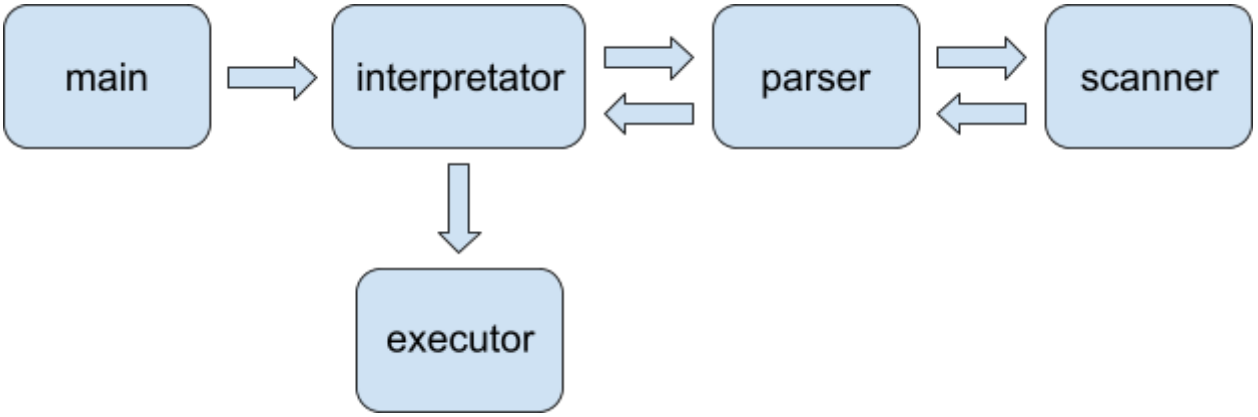
- оператор ввода значения переменной с клавиатуры
- оператор вывода значения переменной на экран
- условный оператор: if
- оператор цикла: while, do while

Также в качестве дополнительной возможности в программе можно оставлять комментарии:

- /* комментарий */

Алгоритмизация

Основная схема работы алгоритма устроена следующим образом:



1. **main.cpp** - данный файл принимает на вход текстовый файл в формате *.txt* и передает его в *interpretator.cpp,* где далее происходит работа с классом Interpretator, который устроен следующим образом:

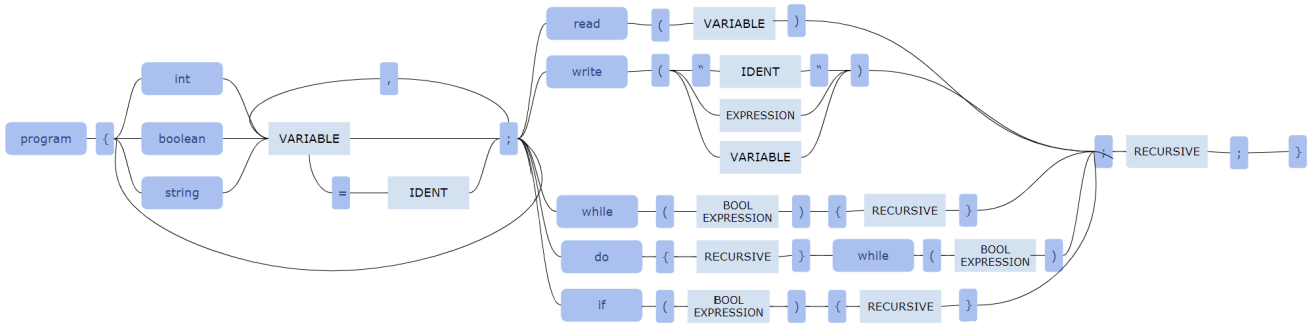
```
#pragma once
#include "executer.h"
class Interpretator
{
    Parser pars;
    Executer E;
public:
    Interpretator (const char * program);
    void interpretation();
};
```

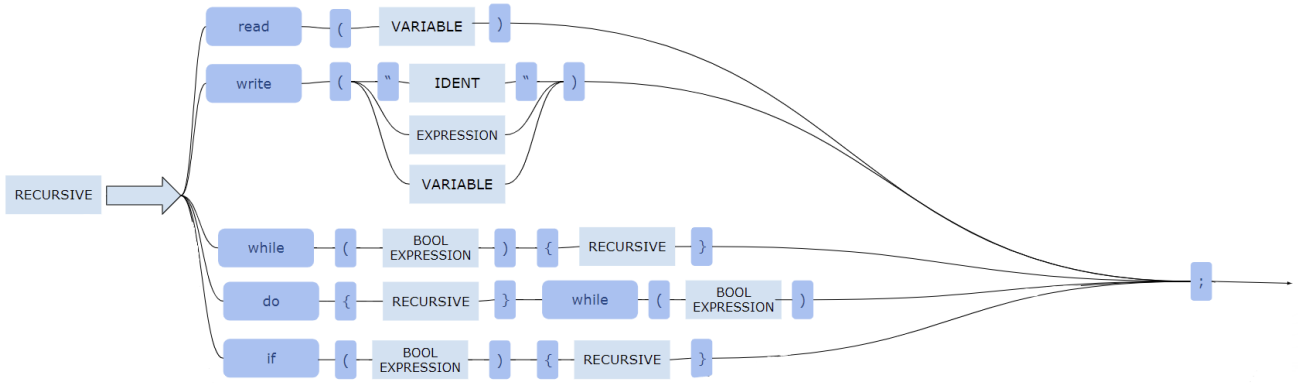
2. **interpretator.cpp** - данный файл обрабатывает и исполняет программу, а именно этим занимаются следующие функции:
    a. *analyze()* - в данной функции происходит проверка программы на ошибки, после чего происходит разбиение программы на лексемы.

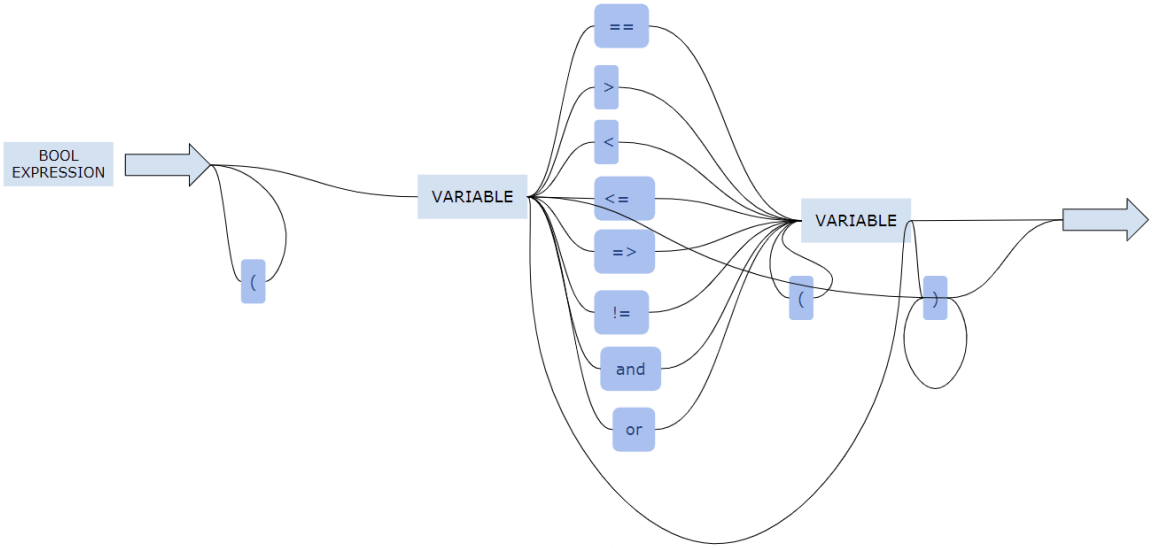
b. *execute()* - исполняет программу с помощью обратной польской записи.

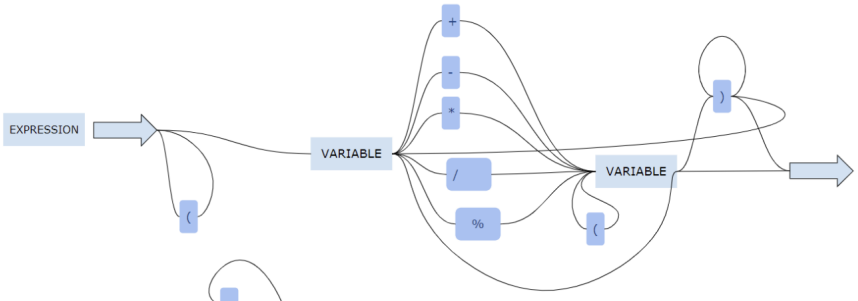
3. **parser.cpp** - в данном файле перед тем как производить разбиение на лексемы и запись из в список, построенный по правилам ПОЛИЗа — обратной польской записи. Сначала вызывается функция чтение лексемы “gl()” (из файла scanner.cpp), которая проверяет очередную полученную на вход лексему на допустимость в данном языке. Далее в parser.cpp лексемы проверяются на допустимый порядок (правильность написания операторов и использование лексем для данного оператора). После чего лексемы записываются в список “prog”. В случае успеха в *parser.cpp* возвращается по одной лексеме и на их основе создается список лексем, который будет подаваться в *executor.cpp*.
4. **scanner.cpp** - в данном файле происходит считывание и проверка поданной лексемы на принадлежность к классу служебных (“:”, “;”, “,” и т.п.), констант(“Hello world!”, “123”) или переменных (“целочисленный a;”). В случае успеха результат передается в *parser.cpp* и записывается в массив лексем.
5. **executor.cpp** - в данный файл передается список лексем и на их основе происходит исполнение программы с помощью обратной польской записи. Результат выводится в консоль.
6. Файлы не включенные в общую схему являются вспомогательными, где описаны объявления функций, которые использовались в основных файлах.

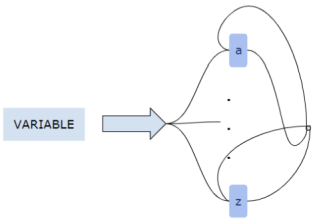
Далее в схематичном виде будет описан синтаксис реализованного языка, где подробно представлено описание каждого из блоков программы:

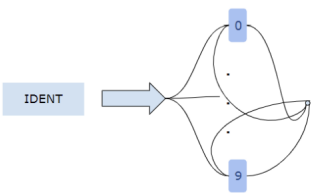


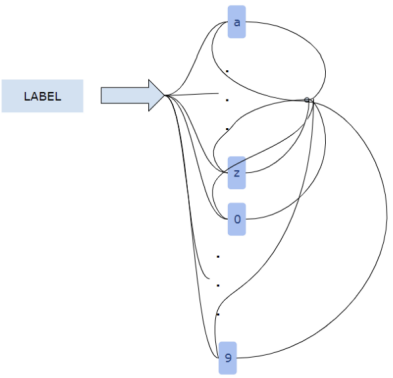












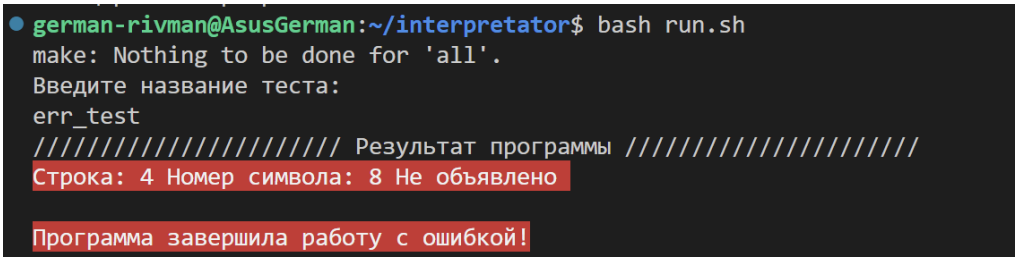
Руководство пользователя

Перед тем, как описывать взаимодействие с программой важно отметить, что для работы необходимо установить UNIX-подобную систему.

Программа компилируется и исполняется через консоль с помощью команды: *bash run.sh*

Скриншот запуска программы:

- В первую очередь вызывается команда *make* для сборки всей программы;
- Следующей командой вызывается файл *translator.py*, который переводит поданную программу на русском языке на английский язык для удобства работы. Пользователю необходимо ввести название программы, которую необходимо исполнить. Важно отметить, что расширение *.txt* писать НЕ НУЖНО.
- Последней командой исполняется программа, где вторым аргументом подается файл с переведенной программой и далее печатается результат. В случае ошибки выводятся сообщения, отмеченные красным цветом:



Продемонстрируем некоторые примеры работы программы:

- Поиск числа Фибоначчи

```
программа
{
    целочисленный n, i = 0, fibsum = 0, fib1 = 1, fib2 = 1;
    вывод("Вычисление числа Фибоначчи.");
    вывод("Введите номер числа Фибоначчи n: ");
    ввод(n);
    цикл(i < n + -2)
    {
        fibsum = fib1 + fib2;
        fib1 = fib2;
        fib2 = fibsum;
        i = i + 1;
    }
    вывод("Ответ: ");

    если(n <= 2)
    {
        вывод(fib1);
    }
    иначе
    {
        вывод(fib2);
    }
}

```

```
german-rivman@AsusGerman:~/interpretator$ bash run.sh
make: Nothing to be done for 'all'.
Введите название теста:
fib
////////////////////// Результат программы //////////////////////
Вычисление числа Фибоначчи.
Введите номер числа Фибоначчи n:

5
Ответ:
5

Конец работы программы
german-rivman@AsusGerman:~/interpretator$ bash run.sh
make: Nothing to be done for 'all'.
Введите название теста:
fib
////////////////////// Результат программы //////////////////////
Вычисление числа Фибоначчи.
Введите номер числа Фибоначчи n:

7
Ответ:
13

Конец работы программы
```

- Вычисление факториала числа

```
программа
{
    целочисленный n1,n2,n3,res;
    целочисленный j = 1;
    цикл(j <= 1)
    {
        res = 1;
        вывод("Вычисление факториала");
        вывод("Введите n1: ");
        ввод(n1);
        цикл(n1 > 0)
        {
            res = res * n1;
            n1 = n1 -1;
        }
        вывод("Ответ: ");
        вывод(res);
        j += 1;
    }

}

```

```
german-rivman@AsusGerman:~/interpretator$ bash run.sh
make: Nothing to be done for 'all'.
Введите название теста:
fact
////////////////////// Результат программы //////////////////////
Вычисление факториала
Введите n1:

5
Ответ:
120

Конец работы программы
german-rivman@AsusGerman:~/interpretator$ bash run.sh
make: Nothing to be done for 'all'.
Введите название теста:
fact
////////////////////// Результат программы //////////////////////
Вычисление факториала
Введите n1:

8
Ответ:
40320

Конец работы программы
```

- Программа с ошибкой

```
программа
{
    целочисленный j = 6, k;
    k = (i + j) * 2;  \* ERROR – нет описания переменной i *\
    вывод("ERROR not detected");
}

```

```
german-rivman@AsusGerman:~/interpretator$ bash run.sh
make: Nothing to be done for 'all'.
Введите название теста:
err_test
////////////////////// Результат программы //////////////////////
Строка: 4 Номер символа: 8 Не объявлено

Программа завершила работу с ошибкой!
```

Для запуска других вариантов программ существует папка *tests*, где можно выбрать любую другую программу. В случае необходимости ознакомления с тем, как работает программа существует папка *Description*, где схематично описана работа компилятора. Description Scanner описывает работу функции gl() файла scanner.cpp. Grammar описывает работу parser.cpp и всевозможные порядки использования лексем, допустимые в программе. Grammatika_interpretatora описывает допустимый вид всех операторов и лексем, а также общие правила написания программы в виде схемы. MainCycle описывает порядок возможных вызовов основных файлов в процессе работы программы.