Санкт-Петербургский государственный политехнический университет

Институт компьютерных наук и технологий

**Кафедра «Компьютерные системы и программные технологии»**

**КУРСОВОЙ ПРОЕКТ**

**Интерпретатор SmallBasic**

по дисциплине «Прикладное программирование»

Выполнил

студент гр.23531/1 Д.И.Графов

Руководитель М.А.Петров

«\_\_\_» \_\_\_\_\_\_\_\_\_\_ 2018 г.

Санкт-Петербург

2018

Санкт-Петербургский государственный политехнический университет

**ЗАДАНИЕ**

**НА ВЫПолнение курсовоГО ПРОЕКТА**

студенту группы 23531/1 Графову Денису Ивановичу

***1. Тема проекта (работы):*** Интерпретатор SmallBasic

***2. Срок сдачи законченного проекта (работы)*** 31.05.18 г.

***3. Исходные данные к проекту (работе)***:

Реализовать интерпретатор выбранного диалекта Basic. Выполнить минимальные требования:

­ целочисленные переменные;

­ инфиксные выражения с “+”, “­“, “\*”, “/”, скобками;

­ операторы сравнения “<”, “<=”, “=”, “<>”, “>=”, “>”;

­ операторы TextWindow.Write(), TextWindow.Read(), If, GoTo, Sub..

Разработка кода и документации должна вестись в репозитории git.

Разработанное ПО должно собираться с помощью команды make компилятором gcc (mingw) с опциями -std=c11 -pedantic -Wall -Wextra без предупреждений компилятора.

***4. Содержание пояснительной записки***: введение, основная часть (раскрывается структура основной части), заключение, список использованных источников, приложения.

***Дата получения задания***: «5» апреля 2018 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.А.Петров

*(подпись)*

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.И.Графов

*(подпись студента)*

05.04.2018г.

**СОДЕРЖАНИЕ**

|  |  |
| --- | --- |
| Введение. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . …………….. | 4 |
| 1. Алгоритм работы…….……………………………………………................ | 5 |
| 2. Структура интерпретатора………………..…………………….................... | 6 |
| 3. Код программы…………..………………..…………………….................... | 7 |
| 3.1. main.c ……………………………………………………………………. | 7 |
| 3.2. analyzer.c…………. …………………………………………………..... | 8 |
| 3.3. Makefile…….…………………………...…………………………... | 17 |
| 4. Методика испытаний……………………………………………………….. | 18 |
| Заключение ...................................................................................................................... | 19 |
| Список использованных источников…......................................................................... | 20 |

**ВВЕДЕНИЕ**

В данном курсовом проекте мною будет написана программа, являющаяся интерпретатором SmallBasic, что соответствует моему индивидуальному заданию по прикладному программированию.

В ней будут реализованы основные возможности интерпретатора, такие чтение команд из входных файлов, хранение и использование целочисленных переменных, обработка инфиксных выражений, операторов сравнения, а также выполнение команд из входных файлов.

1. **Алгоритм работы интерпретатора**

Интерпретатор SmallBasic должен уметь построчно анализировать, обрабатывать и выполнять исходный код программы или запроса.

Алгоритм работы:

1) прочитать инструкцию;

2) проанализировать инструкцию и определить соответствующие действия;

3) выполнить соответствующие действия;

4) если не достигнуто условие завершения программы, прочитать следующую инструкцию и перейти к пункту 2.

Для выполнения пункта 2 создан синтаксический анализатор выражений, который разбивает исходное выражение на составные части, компоненты (неделимые элементы выражения – лексемы).

Функция, разбивающая выражение на составные части, решает несколько задач:

1) игнорировать пробелы и символы табуляции;

2) извлекать каждую лексему из текста;

3) определять тип лексемы.

Анализатор использует шесть типов лексем: DELIMITER, VARIABLE, NUMBER, COMMAND, STRING, MARK (разделитель, переменная, число, команда, строка, метка).

Все ключевые команды имеют внешнее представление (строковое) и внутреннее (число).

Также при создании интерпретатора были учтены особенности и грамматика языка. Например, все числа – целые (integers: -32767...32767).

**Формат входных данных:**

Входным файлом является текстовый файл с программой на языке SmallBasic.

Имя данного файла передается в программу (интерпретатор) как аргумент командной строки.

Пример: **HelloWorld.sb**

|  |
| --- |
| TextWindow.Write("Hello, ") TextWindow.WriteLine("World!") |

**Результат работы интерпретатора:**

Выполнение команд входной программы

Пример: Результат выполнения **HelloWorld.sb**

Cmd



1. **Структура интерпретатора**

Проект состоит из директории **\main**, где хранятся файлы, реализующие Интерпретатор. В **\test** содержаться тестовые файлы для проверки работоспособности программы. **\result** – место сборки программы с помощью **Makefile** (SmallBasic\_Interpreter.exe). Также в проекте находится CMakeLists.txt и упомянутый ранее Makefile.

Программа состоит из 4 файлов (main.c, main.h, lexemes.h, analyzer.c).

В главном файле **main.c** происходит считывание аргументов командной строки и их обработка. Допустимый вариант аргументов: <executable>.exe <fileName>.txt. При некорректных входных аргументах программа выдаст ошибку и выведет пример допустимых аргументов.

Далее в файле main.c происходит обработка входных аргументов, выделение памяти, инициализация состояния интерпретатора. Функция load\_program(char \*) считывает входной файл, записывает в память и при необходимости выделяет еще памяти.

Далее вызывается функция start(), которая содержится в файле **analyzer.c**. В данной функции находится главный цикл программы. Также в Analyzer.c содержится рекурсивный синтаксический анализатор, который состоит из функций level1, level2, level3, level4, level5 и вспомогательных value(для определения типа переменная/число), unary(для установки знака), arithmetic(для выполнения математических операций). Функция getExp для входа в синтаксический анализатор. Также в analyzer.c содержится главная функция анализатора, которая определяет тип лексемы, устанавливает ее внутреннее представление и сохраняет ее в динамически расширяемую память для дальнейшей обработки лексемы. В этом файле находятся основные структуры, необходимые для работы программы, настройки интерпретатора, а также функции инициализации, отчистки памяти и функция для вывода ошибок. Виды ошибок:

* Syntax error
* Brackets requires
* Assignment needed
* Then required
* Undefined label

Все ошибки указывают на причину сбоя и завершают работу интерпретатора.

1. **Код программы**
   1. **main.c**

|  |
| --- |
| #include **<stdio.h>** #include **<stdlib.h>** #include **<setjmp.h>** #include **<mem.h>** #include **<ctype.h>** #include **"main.h"** #define **PROG\_SIZE** 16  *//Объявление переменных* **char** \*program;  *//Объявление функций* **int** loadProgram(**char** \*); *//Считывает программу* **int** main(**int** argc, **char** \*argv[]) {  **char** \*file\_name = argv[1]; *//Имя файла программы* **if** (argc != 2) {  printf(**"Use format: <executable file>.exe <program file>.sb"**);  exit(1);  }   *//Выделение памяти для программы* **if** (!(program = (**char** \*) malloc(**PROG\_SIZE**))) {  printf(**"Error allocating memory"**);  exit(1);  }   *//Загрузка программы* **if** (!loadProgram(file\_name)) exit(1);   start(program);  **return** 0; }  **int** loadProgram(**char** \*fname) {  FILE \*file;  **if** (!(file = fopen(fname, **"r"**)))  **return** 0;  **char** \*point = program;  **int** i = 0, k = 1;   **do** {  \*point = (**char**) getc(file);  point++;  i++;  **if** (i == k \* **PROG\_SIZE**) {  k++;  program = (**char** \*) realloc(program, (size\_t) (k \* **PROG\_SIZE**));  point = program;  point += i;  }  } **while** (!**feof**(file));  \*(point - 1) = 0;  fclose(file);  **return** 1; } |

* 1. **analyzer.c**

|  |
| --- |
| #include **<stdio.h>** #include **<ctype.h>** #include **<stdlib.h>** #include **<memory.h>** #include **"lexemes.h"** *//Объявление переменных* #define **LENGTH\_LABEL** 32 *//длина имени метки* #define **NUM\_LABEL** 100 *//длина массива меток* **int** marks = 0; *//количество меток* **int** numOfSubs = 0; *//количество подпрограмм* **char** \*program; **struct** lexem {  **char** name[80]; *//Строковое представление лексемы* **int** id; *//Внутреннее представление лексемы* **int** type; *//Тип лексемы* } token;  **struct** command {  **char** name[32];  **int** token\_int; } tableCommand[] = {  {**"TextWindow.WriteLine"**, **WriteLine**},  {**"TextWindow.Write"**, **Write**},  {**"TextWindow.Read"**, **Read**},  {**"If"**, **If**},  {**"ElseIf"**, **ElseIf**},  {**"Then"**, **Then**},  {**"Else"**, **Else**},  {**"EndIf"**, **EndIf**},  {**"Goto"**, **Goto**},  {**"Sub"**, **Sub**},  {**"EndSub"**, **EndSub**} };  **struct** label {  **char** \*name[**LENGTH\_LABEL**]; *//Имя метки* **char** \*p; *//Указатель на место размещения в программе* };  **struct** label labels[**NUM\_LABEL**];  **struct** sub {  **char** name[**LENGTH\_LABEL**]; *//Имя подпрограммы* **char** \*p; *//Тело функции* };  **struct** sub subs[**NUM\_LABEL**];  **int** numOfValues = 0; **struct** variable {  **char** name[80];  **int** value; } \*p\_variable;  *//Объявление функций* **void** getToken(); *//Достает очередную лексему* **int** isWhite(**char**);  **void** putBack(); *//Возвращает лексему во вхожной поток (идёт на одну лексему назад)* **void** findEol(); *//Переходит на следующую строку* **int** isDelim(**char**); *//Проверяет является ли символ разделителем* **void** printError(**char** \*);  **int** getIntCommand(**char** \*); *//Возвращает внутреннее представление команды* **void** setAssignment(); *//Присваивает значение переменной* **void** level2(**int** \*), level3(**int** \*), level4(**int** \*), level5(**int** \*); *//Уровни анализа арифметической операции* **void** value(**int** \*); *//Определение значения переменной* **void** unary(**char**, **int** \*); *//Изменение знака* **void** arith(**char**, **int** \*, **int** \*); *//Примитивные операции* **void** getExp(**int** \*); *//Начало анализа арифметического выражения* **struct** variable \*findV(**char** \*); *//Поиск переменной по имени* **struct** variable \*addV(**char** \*); *//Добавление новой переменной* **void** scanLabels(); *//Находит все метки* **void** labelInit(); *//Заполняет массив с метками нулями* **char** \*findLabel(**char** \*); *//Возвращает метку* **char** \*findSub(**char** \*); *//Возвращает функицию* **void** print(), printLine(),  sbIf(), skipElse(), sbGoto();  **void** setSub();  **int** sbRead();  **void** start(**char** \*p) {  program = p;  scanLabels();   **do** {  getToken();  *//Проверка на присваивание* **if** (token.type == **VARIABLE**) {  **if** (findSub(token.name) != **NULL**) {  **if** (\*program == **'('**) {  program++;  **if** (\*program == **')'**) {  program++;  size\_t len = strlen(findSub(token.name)) + strlen(program);  **char** \*cpy = malloc(**sizeof**(**char**) \* (len + 1));  strcpy(cpy, findSub(token.name));  cpy[len] = **'\0'**;  program = strcat(cpy, program);  *//puts(program);* } **else** program--;  } **else** program--;  } **else** {  putBack(); *//откатиться на 1 лексему* setAssignment();  }  }   *//Проверка на команду* **if** (token.type == **COMMAND**) {  **switch** (token.id) {  **case WriteLine**:  printLine();  **break**;  **case Write**:  print();  **break**;  **case If**:  sbIf();  **break**;  **case Else**:  skipElse();  **break**;  **case Goto**:  sbGoto();  **break**;  **case Sub**:  setSub();  **break**;  **default**:  **break**;  }  }  } **while** (token.id != **FINISHED**);  }  **int** isWhite(**char** c) {  **if** (c == **' '** || c == **'\t'**) **return** 1;  **else return** 0; }  **void** getToken() {  **char** \*temp = token.name; *//Указатель на лексему* token.id = 0;  token.type = 0;   *//Пропускаем пробелы* **while** (isWhite(\*program))  program++;   *//Проверка закончился ли файл интерпретируемой программы* **if** (\*program == **'\0'**) {  temp = **'\0'**;  token.id = **FINISHED**;  token.type = **DELIMITER**;  **return**;  }   *//Проверка на конец строки программы* **if** (\*program == **'\n'**) {  \*temp++ = \*program++;  \*temp = **'\0'**;  token.id = **EOL**;  token.type = **DELIMITER**;  **return**;  }   *//Проверка на разделитель* **if** (strchr(**"+-\*/%=:,()><"**, \*program)) {  \*temp++ = \*program++;  \*temp = **'\0'**;  token.type = **DELIMITER**;  **return**;  }   *//Проверка на строку* **if** (\*program == **'"'**) {  program++;  **while** (\*program != **'"'** && \*program != **'\n'**)  \*temp++ = \*program++;  **if** (\*program == **'\n'**)  printError(**"Unpaired quotes"**);  program++;  \*temp = **'\0'**;  token.type = **STRING**;  **return**;  }   *//Проверка на число* **if** (isdigit(\*program)) {  **while** (!isDelim(\*program))  \*temp++ = \*program++;  \*temp = **'\0'**;  token.type = **NUMBER**;  token.id = 3;  **return**;  }   *//Переменная, метка или команда?* **if** (isalpha(\*program)) {  **while** (!isDelim(\*program)) {  \*temp++ = \*program++;  **if** (\*program == **':'**) { *//метка?* \*temp = **'\0'**;  token.type = **MARK**;  **return**;  }  }  \*temp = **'\0'**;  token.id = getIntCommand(token.name); *//Получение внутреннего представления команды* **if** (!token.id) {  token.type = **VARIABLE**;  } **else** token.type = **COMMAND**;  **return**;  }  printf(**"%s\n"**, token.name); }  **int** isDelim(**char** c) {  **if** (strchr(**" !;,+-<>\'/\*%=()\""**, c) || c == **'\r'** || c == **'\n'**)  **return** 1;  **return** 0; }  **void** printError(**char** \*error) {  printf(error);  exit(1); }  **int** getIntCommand(**char** \*t) {  *//Поиск лексемы в таблице операторов* **for** (**int** i = 0; \*tableCommand[i].name; i++) {  **if** (!strcmp(tableCommand[i].name, t))  **return** tableCommand[i].token\_int;  }   **return** 0; *//Незнакомый оператор* }  **void** putBack() {  **char** \*t;  t = token.name;  **for** (; \*t; t++) program--; }  *//Сложение или вычитание* **void** level2(**int** \*result) {  **char** op;  **int** hold;   level3(result);  **while** ((op = \*token.name) == **'+'** || op == **'-'**) {  getToken();  level3(&hold);  arith(op, result, &hold);  } }  *//Вычисление произведения или частного* **void** level3(**int** \*result) {  **char** operation;  **int** hold;   level4(result);   **while** ((operation = \*token.name) == **'/'** || operation == **'%'** || operation == **'\*'**) {  getToken();  level4(&hold);  arith(operation, result, &hold);  } }  *//Унарный + или -* **void** level4(**int** \*result) {  **char** operation;  operation = 0;  **if** (token.type == **DELIMITER** && (\*token.name == **'+'** || \*token.name == **'-'**)) {  operation = \*token.name;  getToken();  }  level5(result);  **if** (operation)  unary(operation, result); }  *//Обработка выражения в круглых скобках* **void** level5(**int** \*result) {  **if** ((\*token.name == **'('**) && (token.type == **DELIMITER**)) {  getToken();  level2(result);  **if** (\*token.name != **')'**)  printError(**"Unpaired parentheses"**);  getToken();  } **else** value(result); }  *//Определение значения переменной по ее имени* **void** value(**int** \*result) {  **struct** variable \*temp = findV(token.name);  **switch** (token.type) {  **case VARIABLE**:  **if** (temp == **NULL**)  printError(**"Variable not initialized"**);  **else** \*result = temp->value;  getToken();  **return**;  **case NUMBER**:  \*result = atoi(token.name);  getToken();  **return**;  **default**:  printError(**"Syntax error: value not initialised"**);  } }  *//Изменение знака* **void** unary(**char** o, **int** \*r) {  **if** (o == **'-'**) \*r = -(\*r); }  *//Выполнение специфицированной арифметики* **void** arith(**char** o, **int** \*r, **int** \*h) {  **int** t;  **switch** (o) {  **case '-'**:  \*r = \*r - \*h;  **break**;  **case '+'**:  \*r = \*r + \*h;  **break**;  **case '\*'**:  \*r = \*r \* \*h;  **break**;  **case '/'**:  \*r = (\*r) / (\*h);  **break**;  **case '%'**:  t = (\*r) / (\*h);  \*r = \*r - (t \* (\*h));  **break**;  **default**:  **break**;  } }  **void** getExp(**int** \*result) {  getToken();  level2(result);  putBack(); }  **struct** variable \*findV(**char** \*name) {  **int** i = 1;  **struct** variable \*temp = p\_variable;  **while** (i <= numOfValues) {  **if** (!strcmp(name, temp->name)) {  **return** temp;  }  i++;  temp++;  }  **return NULL**; }  **struct** variable \*addV(**char** \*name) {  numOfValues++;  p\_variable = (**struct** variable \*) realloc(p\_variable, **sizeof**(**struct** variable) \* numOfValues);  **struct** variable \*temp = p\_variable;   **int** i = 1;  **while** (i < numOfValues) {  temp++;  i++;  }  strcpy(temp->name, name);  temp->value = **'\0'**;   **return** temp; }  *//Присваивание значения переменной* **void** setAssignment() {  **int** value;  getToken(); *//Получаем имя переменной* **struct** variable \*var;  **if** ((var = findV(token.name)) != **NULL**) {  getToken(); *//Считываем равно* getExp(&value);  var->value = value;  } **else** {  var = addV(token.name);  getToken(); *// Считываем равно* **if** (\*token.name == **'='**) {  getToken();  **if** (token.type == **NUMBER**) {  var->value = atoi(token.name);  } **else if** (token.id == **Read**) {  getToken();  **if** (\*token.name == **'('**) {  getToken();  **if** (\*token.name == **')'**) {  var->value = sbRead();  } **else** printError(**"Brackets required"**);  } **else** printError(**"Brackets required"**);  } **else if** (token.type == **VARIABLE**) {  var->value = findV(token.name)->value;  } **else** printError(**"Assignment needed"**);  }  } }  *//Переход на следующую строку программы* **void** findEol() {  **while** (\*program != **'\n'** && \*program != **'\0'**)  program++;  **if** (\*program)  program++; }  **void** print() {  **int** answer;  getToken();  **int** type = 0;  **char** \*str;  **if** (\*token.name == **'('**) {  getToken();  type = token.type;  **if** (token.type == **STRING**) {  str = token.name;  } **else** { *//Значит выражение* putBack();  getExp(&answer);  getToken();  putBack();  }  **if** (\*program == **')'**) {  **if** (type == **STRING**) {  printf(str);  } **else** printf(**"%d"**, answer);  getToken();  } **else** printError(**"Brackets required"**);  } **else** printError(**"Brackets required"**); }   **void** printLine() {  print();  printf(**"\n"**); }  **int** sbRead() {  **int** result;  scanf(**"%d"**, &result); *//Чтение входных данных* **return** result; }  **void** sbIf() {  **int** x, y, cond;  **char** operation;  **char** operationSecond;  getExp(&x); *//Получаем левое выражение* getToken(); *//Получаем оператор* **if** (!strchr(**"=<>"**, \*token.name)) {  printError(**"Syntax error"**); *//Недопустимый оператор* **return**;  }  operation = \*token.name;  *//Определяем результат* cond = 0;  **switch** (operation) {  **case '='**:  getExp(&y);  **if** (x == y) cond = 1;  **break**;  **case '<'**:  getToken();  **if** (strchr(**"=<>"**, \*token.name)) {  operationSecond = \*token.name;  **switch** (operationSecond) {  **case '='**:  getExp(&y);  **if** (x <= y) cond = 1;  **break**;  **case '>'** :  getExp(&y);  **if** (x != y) cond = 1;  **break**;  **case '<'**:  printError(**"Syntax error"**);  **return**;  **default**:  **break**;  }  } **else** {  putBack();  getExp(&y);  **if** (x < y) cond = 1;  **break**;  }  **break**;  **case '>'**:  getToken();  **if** (strchr(**"=<>"**, \*token.name)) {  operationSecond = \*token.name;  **switch** (operationSecond) {  **case '='**:  getExp(&y);  **if** (x >= y) cond = 1;  **break**;  **case '>'** :  printError(**"Syntax error"**);  **return**;  **case '<'**:  printError(**"Syntax error"**);  **return**;  **default**:  **break**;  }  } **else** {  putBack();  getExp(&y);  **if** (x > y) cond = 1;  **break**;  }  **break**;  **default**:  **break**;  }  **if** (cond) { *//Если значение If "истина"* getToken();  **if** (token.id != **Then**) {  printError(**"Then required"**);  **return**;  }  } **else** {  getToken(); *//Пропускаем Then* getToken();  **if** (strchr(**"\n"**, \*token.name)) {  **do** {  getToken();  **if** (token.id == **EndIf**) **return**;  } **while** (token.id != **Else**);  } **else** {  findEol();*//Если ложь - переходим на следующую строку* }  } }  **void** skipElse() {  **do** {  getToken();  } **while** (token.id != **EndIf**); }  *//Инициализация массива хранения меток* **void** labelInit() {  **for** (**int** i = 0; i < **NUM\_LABEL**; i++)  labels[i].name[0] = **'\0'**; }  *//Поиск всех меток* **void** scanLabels() {  **char** \*temp;   labelInit(); *//Инициализация массива меток* temp = program; *//Указатель на начало программы* getToken();  *//Если первая лексема является меткой* **if** (token.type == **MARK**) {  findEol();  strcpy((**char** \*) labels[0].name, token.name);  labels[0].p = program;  marks++;  putBack(); *//чтобы не терять следующую метку* }   *//Вторая строка и далее* findEol();  **do** {  getToken();  **if** (token.type == **MARK**) {  findEol();  strcpy((**char** \*) labels[marks].name, token.name);  labels[marks].p = program; *//Текущий указатель программы* marks++;  putBack(); *//чтобы не терять следующую метку* }  *//Если строка не помечена, переход к следующей* **if** (token.id != **EOL**) {  findEol();  }  } **while** (token.id != **FINISHED**);  program = temp; *//Восстанавливаем начальное значение* }  **char** \*findLabel(**char** \*s) {  **for** (**int** i = 0; i < **NUM\_LABEL**; i++)  **if** (!strcmp((**const char** \*) labels[i].name, s))  **return** labels[i].p;  **return '\0'**; *//Ошибка* }  **void** sbGoto() {  **char** \*location;  getToken(); *//Получаем метку перехода  //Поиск местоположения метки* location = findLabel(token.name);  **if** (location == **'\0'**) {  printError(**"Undefined label"**); *//Метка не обнаружена* } **else** program = location; *//Старт программы с указанной точки* }  **void** setSub() {  **int** counter = 0;  getToken(); *//получаем имя* **struct** sub \*sub = malloc(**sizeof**(**struct** sub));  size\_t size = 1000;  size\_t len = 1;  size\_t program\_len = strlen(program);  sub->p = malloc(**sizeof**(**char**) \* 1000); *//Выделили память под тело программы* sub->p[0] = **'\0'**; *//Затерли память, если в ней были значения* sub->name[0] = **'\0'**;  strcpy(sub->name, token.name); *//добавляем в список функций* findEol();  **char** \*istr;  **char** \*copy = (**char** \*) malloc(program\_len + 1);  strcpy(copy, program);  copy[program\_len] = **'\0'**;  istr = strtok(program, **"\n"**);  *// Выделение последующих частей* **while** (strcmp(istr, **"EndSub"**) != 0) {  *// Вывод очередной выделенной части* len += (strlen(istr) + 1);  **if** (len >= size) {  size = len \* 2;  sub->p = realloc(sub->p, size);  }  strcat(sub->p, istr);  strcat(sub->p, **"\n"**);  *// Выделение очередной части строки* istr = strtok(**NULL**, **"\n"**);  counter++;  }  sub->p = realloc(sub->p, len);  subs[numOfSubs] = \*sub;  program = copy;  **for** (**int** i = 0; i < counter + 1; i++) {  findEol();  }  free(istr);  numOfSubs++; }  **char** \*findSub(**char** \*s) {  **for** (**int** i = 0; i < **NUM\_LABEL**; i++)  **if** (strcmp(subs[i].name, s) == 0) {  **return** subs[i].p;  }  **return '\0'**; *//Ошибка* } |

* 1. **Makefile**

|  |
| --- |
| all: SmallBasic\_Interpreter SmallBasic\_Interpreter : result/main.o result/analyzer.o  gcc -std=c11 -pedantic -Wall -Wextra -o result/SmallBasic\_Interpreter result/main.o result/analyzer.o  result/main.o : main/main.c  gcc -std=c11 -pedantic -Wall -Wextra -c -o result/main.o main/main.c  result/analyzer.o : main/analyzer.c  gcc -std=c11 -pedantic -Wall -Wextra -c -o result/analyzer.o main/analyzer.c |

1. **Методика испытаний**

Работоспособность интерпретатора, а также его соответствие минимальным требованиям ТЗ были проверены с помощью набора тестовых файлов: HelloWorld.sb, Goto.sb, factorial.sb, test.sb.

Пример с программой “Hello, World!” был приведен в п.1.

|  |  |  |
| --- | --- | --- |
| **Файл на языке SmallBasic** | **Исходный код** | **Результат работы** |
| Goto.sb | Hello: x = 1 Start: TextWindow.WriteLine(x) x = x + 1 If x <= 5 Then  Goto Start EndIf Goto Finish a = TextWindow.Read() b = 10 TextWindow.WriteLine(a - b/2) Finish: TextWindow.WriteLine("Xex") | 1  2  3  4  5  Xex |
| factorial.sb  (введено число 5) | a = TextWindow.Read() f = 1 Start: If a > 1 Then  f = f \* a  a = a - 1  Goto Start EndIf TextWindow.Write("Result: ") TextWindow.WriteLine(f) | Result: 120 |
| test.sb | a = 10 c = a Sub hello  TextWindow.WriteLine("hello")  b = c + 10 EndSub Sub lol  TextWindow.WriteLine("lol")  hello() EndSub hello() lol() TextWindow.WriteLine("WOW") TextWindow.Write("a + b = ") TextWindow.WriteLine(a + b) TextWindow.Write("c = ") TextWindow.WriteLine(c) | hello  lol  hello  WOW  a + b = 30  c = 10 |

## ЗАКЛЮЧЕНИЕ

В данном курсовом проекте мною была написана программа, реализующая интерпретатор SmallBasic.

Работоспособность интерпретатора была проверена набором тестовых файлов.

Таким образом, программа полностью соответствует заданным в индивидуальном задании требованиям.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Документация по SmallBasic

(http://smallbasic-publicwebsite-dev.azurewebsites.net/Pages/DocumentReference.aspx)

1. Информация по make (http://pubs.opengroup.org/onlinepubs/9699919799/utilities/make.html#tag20761304)
2. Применение make (https://habr.com/post/211751/)
3. Общая информация об интерпретаторах (http://cpp.com.ru/shildt\_spr\_po\_c/01/0105.html)
4. M. Уэйт, С. Прата, Д. Мартин - Язык Си руководство для начинающих [Электронный ресурс] 1988г. – Режим доступа: <http://storage.ded32.net.ru/Lib/TX/CPrimerPlus.pdf>
5. Стандарт языка C[11] (http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1570.pdf)