Санкт-Петербургский политехнический университет Петра Великого

Кафедра компьютерных систем и программных технологий

**Отчёт по курсовой работе**

**Дисциплина**: Прикладное программирование

**Тема**: Компилятор языка Оберон-0

Выполнила студентка гр. 23531/2 А.С. Судакова

Преподаватель Т.Л. Сидорина

Проект защищен с оценкой

“\_\_\_”\_\_\_\_\_\_\_\_\_\_\_\_\_ 2018 г.

Санкт-Петербург

2018

Оглавление

1. **Техническое задание3**
2. **Алгоритм и структура программы4**
3. **Тесты7**
4. **Использованная литература18**
5. **Приложение18**
   1. Синтаксис языка Оберон-018
   2. Код целевой RISC-машины на языке Оберон18
   3. main.c19
   4. parser.h21
   5. parser.c22
   6. scanner.h40
   7. scanner.c41
   8. codeGenerator.h45
   9. codeGenerator.c46
6. Техническое задание

Целью курсового проекта является реализация компилятора простого языка высокого уровня для простой стековой или RISC машины. В качестве компилируемого языка выбран язык Оберон-0, в качестве целевой машины – RISC машина.

В поставленной задаче можно выделить несколько взаимосвязанных подзадач, а именно: лексический анализ, синтаксический анализ, генерация кода. Во время лексического и синтаксического анализа, строится синтаксическое дерево, на основе которого генерируется необходимый код.

Структура проекта:

* Входной и выходные файлы:
  + Путь входного файла указывается пользователем в процессе работы программы. В данном файле ожидается код на языке Оберон-0
  + В выходном файле output.txt записан скомпилированный (в случае удачной компиляции) код. В случае неудачной компиляции файл пуст
  + В выходном файле report.txt содержится информация о процессе компиляции (сообщение об успешной компиляции либо сообщение об ошибке компиляции со списком ошибок)
* Файл с основной частью программы
* Файл, отвечающий за лексический анализ
* Файл, отвечающий за синтаксический анализ
* Файл, отвечающий за генерацию кода

Задачи разработанного компилятора:

1. Проведение лексического анализа, то есть, преобразование исходного кода в последовательность лексем
2. Проведение синтаксического анализа, сопоставление последовательности полученных лексем с формальной грамматикой языка Оберон-0. Получение синтаксического дерева, отражающего синтаксическую структуру исходного кода
3. В случае возникновения ошибок, окончание процесса компиляции, запись информации об ошибках в файл report.txt, завершение выполнения компиляции
4. При отсутствии ошибок, генерация кода для RISC-машины на основе синтаксического дерева. Запись сгенерированного кода в файл output.txt, запись времени компиляции и информации об успешной компиляции в файл report.txt
5. Вывод в консоль информации о компиляции (аналогична содержимому файла report.txt), пути выходных файлов. Завершение работы программы

Программа должна собираться с помощью команды make компилятором gcc (mingw) с опциями –std=c11 –pedantic –Wall –Wextra без предупреждений компилятора.

Вид входного файла:

MODULE Record;

CONST

k = 2;

TYPE

i = RECORD

j : INTEGER;

z : BOOLEAN;

END;

VAR

l : i;

BEGIN

l.j := 5;

l.z := TRUE;

l.j := k;

END Record.

Вид выходного файла output.txt:

entry 0

0 MOVI 13, 0, 1016

4 PSH 14, 13, 4

8 MOVI 0, 0, 5

12 STW 0, 15, -20

16 MOVI 0, 0, 1

20 STW 0, 15, -24

24 MOVI 0, 0, 2

28 STW 0, 15, -36

32 POP 14, 13, 4

36 RET 14

40 bytes

Вид выходного файла report.txt при возникновении ошибок компиляции:

Compilation begins.

Compiles module While.

error: line 5, position 8 Lost END

error: line 5, position 9 ;?

Compilation finished.

Вид выходного файла report.txt при удачной компиляции:

Compilation begins.

Compiles module While.

Code generated.

Compilation finished.

Формат командной строки:

compiler.txt outputfile.txt

При неверном формате командной строки в консоль выводится информация о программе и ее аргументах.

Ссылка на репозиторий github: https://github.com/Helnsidva/finalCompiler

1. Алгоритм и структура программы

Программа состоит за четырех файлов:

1. main.c.
2. parser.c (синтаксический анализатор)
3. scanner.c (лексический анализатор)
4. codeGenerator.c (генератор кода)

Работа программы начинается с файла main.c. Здесь читаются аргументы командной строки и, если формат аргументов корректен (один аргумент), производится чтение кода из файла (функция readCode), если нет – выводится информация о программе (функция manual). Если код считан успешно – вызывается функция compile из файла parser.c.

В функции compile создается структура данных (storage) для хранения различных данных, необходимых при компиляции: исходный код, индекс последнего считанного символа, индекс и флаг ошибки, информация о последнем считанном слове, указатель на файл report.txt, счетчик команд, текущий уровень вложенности процедуры, и т.д., и производится инициализация ее полей. Затем вызывается функция module, в которой происходит непосредственно компиляция исходного кода. В случае удачной компиляции, полученный код декодируется в функции decode (файл codeGenerator.c). На этом процесс компиляции завершается.

В ходе компиляции, для хранения объектов (переменная, выражение, функция) используется две структуры данных: object и item. В структуре object хранится полученный в синтаксическом анализаторе объект. При анализе объектов в генераторе кода используется структура item.

Для хранения созданных объектов используются «scopes» - области видимости. Для каждой функции и записи области видимости разные. В storage хранится объект topScope, указывающий на вершину текущей области видимости. В его поле nextObject начинается список его локальных объектов. Поле previousScope указывает на предыдущую область видимости. Закрывает каждую область видимости «пустой» объект guard, созданный для удобства поиска в списках. Так же при инициализации создается глобальная область видимости universe, хранящая глобальные типы данных (INTEGER и BOOLEAN) и глобальные функции (Write, WriteHex, WriteLn, Read). Она так же хранится в storage.

В целевой машине 4 зарезервированных регистра: FP – 12, SP – 13, LNK – 14, PC – 15. FP указывает на начало кадра активации, SP – вершина стека, LNK хранит адрес вершины стека, PC – счетчик команд.

Чтение слов и символов происходит в файле scanner.c в функции get. Здесь, пропуская пробелы и символы табуляции, читаются и различаются терминальные символы (их список в scanner.h), идентификаторы, числа, а также пропускаются комментарии (формат комментариев – (\*comment\*)). Так же здесь содержится функция mark, вызывающаяся при ошибке. Здесь флаг ошибки устанавливается в 1, фиксируется индекс ошибки и сообщение о ней выводится в файл report.txt и консоль. При возникновении ошибки оставляется сообщение о ней, а анализ кода продолжается для нахождения максимального количества ошибок.

В функции module последовательно анализируется код в порядке, соответствующем синтаксису языка: первое слово – MODULE, затем идентификатор – название модуля и символ ‘;’ (MODULE ident;). Затем – объявления объектов в порядке CONST, TYPE, VAR (другой порядок - ошибка). После этого – объявление процедур. Затем – слово BEGIN и последовательность операторов данного модуля. Оканчиваться файл должен словом END, идентификатором данного модуля и символом '.' (END ident.).

Чтение объявлений констант, типов и переменных происходит в функции declarations. Объекты заносятся в текущую область видимости. Вычисляется размер переменных, так как под них необходимо выделить память. Константы и типы должны быть инициализированы.

Чтение процедур происходит в функции procedureAnalyzer. Здесь для новой процедуры так же создается объект в текущей области видимости (уровень вложенности увеличивается). Объекты области, предшествующей процедуре – ее формальные (указанные в объявлении процедуры) параметры. Для каждого параметра указывается его смещение относительно кадра активации (указателя на начало стека процедуры). Для передаваемых в процедуру параметров смещение положительное. Затем, считываются объявления констант, типов, переменных и процедур, аналогично функции module. В поле value объекта записывается адрес точки входа в процедуру. Записывается пролог (функция prologue в codeGenerator; сохранение в стек значений регистров LNK, FP, запись полученного SP в FP, возврат SP к концу кадра активации). Затем, после слова BEGIN, ожидается последовательность операторов. Они анализируются в функции statements. У локальных переменных функции смещение относительно кадра активации отрицательное. В конце процедуры ожидается строка “END procname;”. После окончания процедуры записывается ее эпилог (функция epilogue; возврат значений SP, FP, LNK и возврат к месту вызова). Уровень вложенности уменьшается.

Допустимые операторы (statements) – присваивание, вызов процедуры, условный оператор if и цикл while. Здесь анализируется первый символ каждого оператора. Если это идентификатор – то должно быть либо присваивание, либо вызов процедуры. Идентификатор ищется среди существующих объектов и проверяется: переменная это, процедура, или что-то другое. Если переменная – после нее должен быть знак ':='. Вычисляется выражение после этого знака (функция expression) и затем, в функции store (codeGenerator), происходит присваивание. Если идентификатор – процедура, то осуществляется ее вызов. Здесь проверяются передаваемые параметры, они записываются в стек, затем процедура непосредственно вызывается (функция procedureCall). Вызов процедуры из universe осуществляется в функции globalCall, параметры здесь непосредственно записываются в команду. Если оператор – if, то вычисляется условное выражение, для него генерируется команда, и записывается переход в адрес после оператора, если условие не выполняется. Далее, записывается последовательность операторов и безусловный переход к концу всей цепочки elsif и else. Для операторов elsif генерируются те же переходы. Для else генерируется только последовательность операторов. Узнавая адрес очередного оператора elsif и else, происходит запись адреса перехода для предыдущего оператора. Переходы и выравнивание происходит в функциях falseJump, elseJump, fixLink. Для цикла while алгоритм похож: вычисляется условное выражение, для него генерируется команда, и записывается переход в адрес после оператора, если условие не выполняется. Затем генерируется последовательность операторов цикла, в конце которых добавляется безусловный переход в началу цикла (проверке условия; функция whileJump).

Анализ арифметических выражений происходит согласно приоритету арифметических операторов. В первую очередь, выражение анализируется в функции expression. Здесь читаются логические выражения со знаками =, #, <, <=, >, >=. Левая и правая части анализируются как сумма, разность, либо логическое или (‘|’) в функции simpleExpression. Здесь аргументы анализируются как результат умножения, деления, MOD и логического и (‘&’) (функция term). Аргументы term – числа и идентификаторы. Если происходит обращение к элементу массива и записи, вызывается функция factor.

После объявления процедур сохраняется адрес следующей команды – это адрес входа в модуль. По этому адресу записывается пролог модуля (функция codeHeader; запись адреса вершины стека и его сохранение в регистре LNK). Если встречено слово BEGIN – анализ последовательности операторов по описанному выше алгоритму (statements). После операторов ожидается строка “END modulename.”. В конце записывается эпилог модуля – функция codeEnding – возврат нуля).

В генераторе кода запись команд производится в функциях put и putJump. Здесь код команды и аргумента кодируются: команда занимает первые 4 бита числа, следующие 4 бита – первый аргумент, следующие 4 бита – второй аргумент, следующие 18 бит – третий аргумент для put; команда занимает первые 4 бита числа, следующие 26 бит – аргумент для putJump. В функции decode декодируется это число, получая сначала код команды, затем код аргументов.

1. Тесты

Программа собирается с помощью команды make компилятором gcc (mingw) с опциями –std=c11 –pedantic –Wall –Wextra без предупреждений компилятора:

Рисунок 3.1. Сборка проекта

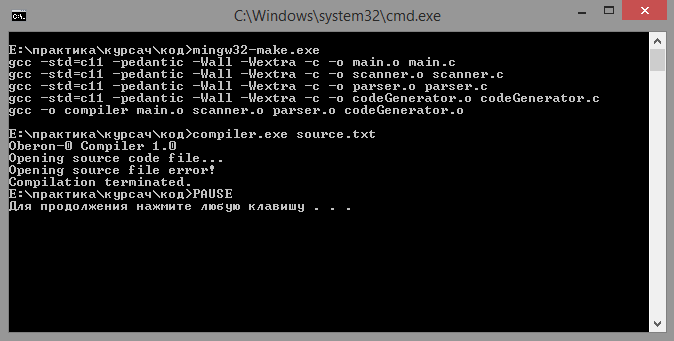
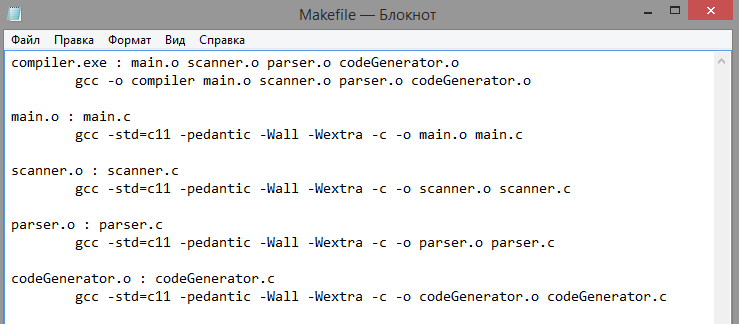
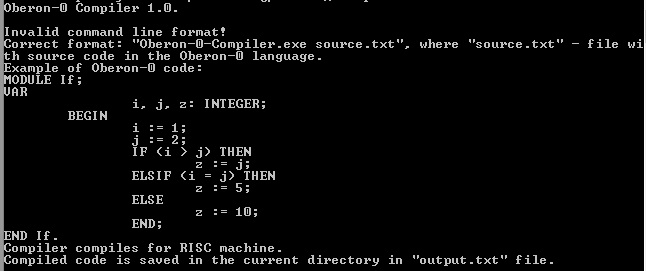


Рисунок 3.2. Makefile



Запуск с некорректными аргументами:

Рисунок 3.3. Некорректные аргументы



Пример работы программы:

Рисунок 3.4. Вывод в консоль при компиляции

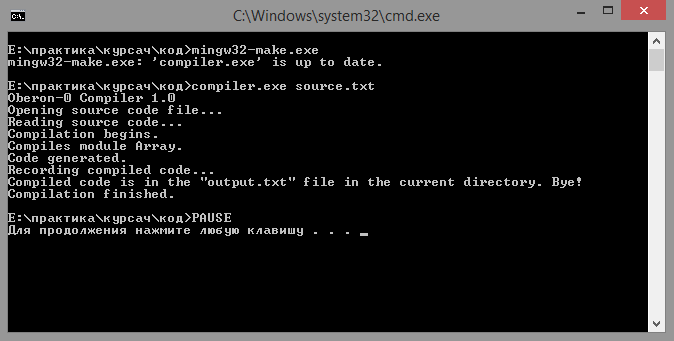


Рисунок 3.5. Пример файла report.txt при удачной компиляции

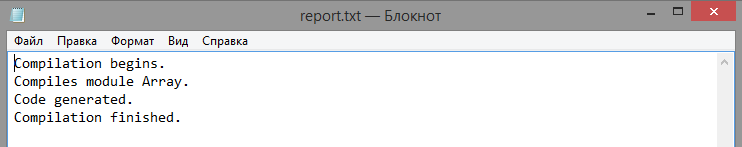
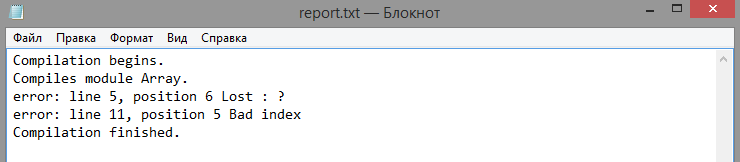


Рисунок 3.6. Пример файла report.txt при неудачной компиляции



Проведенные тесты:

1. Работа с массивом.

Входной файл:

MODULE Array;

VAR

a: ARRAY 6 OF INTEGER;

i: INTEGER;

BEGIN

a[0] := 7;

a[1] := 3;

Write(a[0]);

Write(a[1]);

i := 1;

a[i] := i;

a[5] := 0;

END Array.

Выходной файл:

entry 0

0 MOVI 13, 0, 996

4 PSH 14, 13, 4

8 MOVI 0, 0, 7

12 STW 0, 15, -36

16 MOVI 0, 0, 3

20 STW 0, 15, -40

24 LDW 0, 15, -48

28 WRD 0, 0, 0

32 LDW 0, 15, -52

36 WRD 0, 0, 0

40 MOVI 0, 0, 1

44 STW 0, 15, -72

48 LDW 0, 15, -76

52 CHKI 0, 0, 6

56 MULI 0, 0, 4

60 ADD 0, 15, 0

64 LDW 1, 15, -92

68 STW 1, 0, -92

72 MOVI 0, 0, 0

76 STW 0, 15, -80

80 POP 14, 13, 4

84 RET 14

88 bytes

1. Присваивания.

Входной файл:

MODULE Assignment;

VAR

i, j, z: INTEGER;

BEGIN

i := 10;

j := 5;

z := 0;

z := i + j;

z := i + 0;

z := i - j;

z := j - i;

z := z \* i;

z := z DIV j;

z := i MOD z;

END Assignment.

Выходной файл:

entry 0

0 MOVI 13, 0, 1012

4 PSH 14, 13, 4

8 MOVI 0, 0, 10

12 STW 0, 15, -16

16 MOVI 0, 0, 5

20 STW 0, 15, -28

24 MOVI 0, 0, 0

28 STW 0, 15, -40

32 LDW 0, 15, -36

36 LDW 1, 15, -44

40 ADD 0, 0, 1

44 STW 0, 15, -56

48 LDW 0, 15, -52

52 ADDI 0, 0, 0

56 STW 0, 15, -68

60 LDW 0, 15, -64

64 LDW 1, 15, -72

68 SUB 0, 0, 1

72 STW 0, 15, -84

76 LDW 0, 15, -84

80 LDW 1, 15, -84

84 SUB 0, 0, 1

88 STW 0, 15, -100

92 LDW 0, 15, -104

96 LDW 1, 15, -100

100 MUL 0, 0, 1

104 STW 0, 15, -116

108 LDW 0, 15, -120

112 LDW 1, 15, -120

116 DIV 0, 0, 1

120 STW 0, 15, -132

124 LDW 0, 15, -128

128 LDW 1, 15, -140

132 MOD 0, 0, 1

136 STW 0, 15, -148

140 POP 14, 13, 4

144 RET 14

148 bytes

1. Двойная вложенность процедур.

Входной файл:

MODULE DoubleLevel;

VAR

i, j, z: INTEGER;

PROCEDURE first(arg1 : INTEGER);

BEGIN

arg1 := 5;

END first;

PROCEDURE second(arg2 : INTEGER);

BEGIN;

arg2 := 6;

first(arg2);

END second;

BEGIN

second(5);

END DoubleLevel.

Выходной файл:

entry 92

0 PSH 14, 13, 4

4 PSH 12, 13, 4

8 MOV 12, 0, 13

12 SUBI 13, 13, 0

16 MOVI 0, 0, 5

20 STW 0, 12, 8

24 MOV 13, 0, 12

28 POP 12, 13, 4

32 POP 14, 13, 8

36 RET 14

40 PSH 14, 13, 4

44 PSH 12, 13, 4

48 MOV 12, 0, 13

52 SUBI 13, 13, 0

56 MOVI 0, 0, 6

60 STW 0, 12, 8

64 LDW 0, 12, 8

68 PSH 0, 13, 4

72 BSR -72

76 MOV 13, 0, 12

80 POP 12, 13, 4

84 POP 14, 13, 8

88 RET 14

92 MOVI 13, 0, 1012

96 PSH 14, 13, 4

100 MOVI 0, 0, 5

104 PSH 0, 13, 4

108 BSR -68

112 POP 14, 13, 4

116 RET 14

120 bytes

1. Запись процедуры.

Входной файл:

MODULE GlobalProc;

VAR

i, j, z: INTEGER;

PROCEDURE ExternalProc();

VAR

t, p, k: BOOLEAN;

BEGIN

t := TRUE;

p := t;

END ExternalProc;

END GlobalProc.

Выходной файл:

entry 48

0 PSH 14, 13, 4

4 PSH 12, 13, 4

8 MOV 12, 0, 13

12 SUBI 13, 13, 12

16 MOVI 0, 0, 1

20 STW 0, 12, -4

24 LDW 0, 12, -4

28 STW 0, 12, -8

32 MOV 13, 0, 12

36 POP 12, 13, 4

40 POP 14, 13, 4

44 RET 14

48 MOVI 13, 0, 1012

52 PSH 14, 13, 4

56 POP 14, 13, 4

60 RET 14

64 bytes

1. Оператор If.

Входной файл:

MODULE If;

VAR

i, j, z: INTEGER;

BEGIN

i := 1;

j := 2;

IF (i > j) THEN

z := j;

ELSIF (i = j) THEN

z := 5;

ELSE

z := 10;

END;

END If.

Выходной файл:

entry 0

0 MOVI 13, 0, 1012

4 PSH 14, 13, 4

8 MOVI 0, 0, 1

12 STW 0, 15, -16

16 MOVI 0, 0, 2

20 STW 0, 15, -28

24 LDW 0, 15, -28

28 LDW 1, 15, -36

32 CMP 0, 0, 1

36 BLE 16

40 LDW 0, 15, -48

44 STW 0, 15, -56

48 BR 40

52 LDW 0, 15, -56

56 LDW 1, 15, -64

60 CMP 0, 0, 1

64 BNE 16

68 MOVI 0, 0, 5

72 STW 0, 15, -84

76 BR 12

80 MOVI 0, 0, 10

84 STW 0, 15, -96

88 POP 14, 13, 4

92 RET 14

96 bytes

1. Вычисление логических выражений.

Входной файл:

MODULE LogAssign;

VAR

i, j: INTEGER;

z: BOOLEAN;

BEGIN

i := 5;

j := 4;

z := (i > j);

END LogAssign.

Выходной файл:

entry 0

0 MOVI 13, 0, 1012

4 PSH 14, 13, 4

8 MOVI 0, 0, 5

12 STW 0, 15, -16

16 MOVI 0, 0, 4

20 STW 0, 15, -28

24 LDW 0, 15, -28

28 LDW 1, 15, -36

32 CMP 0, 0, 1

36 BLE 12

40 MOVI 0, 0, 1

44 BR 8

48 MOVI 0, 0, 0

52 STW 0, 15, -64

56 POP 14, 13, 4

60 RET 14

64 bytes

1. Вложенные функции.

Входной файл:

MODULE NestedProc;

VAR

i, j, z: INTEGER;

PROCEDURE first(arg1 : INTEGER);

PROCEDURE second(arg2 : INTEGER);

BEGIN;

arg2 := 6;

END second;

BEGIN

arg1 := 5;

second(arg1);

END first;

BEGIN

first(5);

END NestedProc.

Выходной файл:

entry 92

0 PSH 14, 13, 4

4 PSH 12, 13, 4

8 MOV 12, 0, 13

12 SUBI 13, 13, 0

16 MOVI 0, 0, 6

20 STW 0, 12, 8

24 MOV 13, 0, 12

28 POP 12, 13, 4

32 POP 14, 13, 8

36 RET 14

40 PSH 14, 13, 4

44 PSH 12, 13, 4

48 MOV 12, 0, 13

52 SUBI 13, 13, 0

56 MOVI 0, 0, 5

60 STW 0, 12, 8

64 LDW 0, 12, 8

68 PSH 0, 13, 4

72 BSR -72

76 MOV 13, 0, 12

80 POP 12, 13, 4

84 POP 14, 13, 8

88 RET 14

92 MOVI 13, 0, 1012

96 PSH 14, 13, 4

100 MOVI 0, 0, 5

104 PSH 0, 13, 4

108 BSR -68

112 POP 14, 13, 4

116 RET 14

120 bytes

1. Работа с записью.

Входной файл:

MODULE Record;

CONST

k = 2;

TYPE

i = RECORD

j : INTEGER;

z : BOOLEAN;

END;

VAR

l : i;

BEGIN

l.j := 5;

l.z := TRUE;

l.j := k;

END Record.

Выходной файл:

entry 0

0 MOVI 13, 0, 1016

4 PSH 14, 13, 4

8 MOVI 0, 0, 5

12 STW 0, 15, -20

16 MOVI 0, 0, 1

20 STW 0, 15, -24

24 MOVI 0, 0, 2

28 STW 0, 15, -36

32 POP 14, 13, 4

36 RET 14

40 bytes

1. Передача параметров в процедуру.

Входной файл:

MODULE StatArgs;

VAR

i, j, z: INTEGER;

PROCEDURE ExternalProc(m, n : INTEGER);

VAR

t, p, k: BOOLEAN;

BEGIN

t := TRUE;

p := t;

k := (m < n);

END ExternalProc;

BEGIN

i := 5;

j := 6;

ExternalProc(i, j);

END StatArgs.

Выходной файл:

entry 80

0 PSH 14, 13, 4

4 PSH 12, 13, 4

8 MOV 12, 0, 13

12 SUBI 13, 13, 12

16 MOVI 0, 0, 1

20 STW 0, 12, -4

24 LDW 0, 12, -4

28 STW 0, 12, -8

32 LDW 0, 12, 12

36 LDW 1, 12, 8

40 CMP 0, 0, 1

44 BGE 12

48 MOVI 0, 0, 1

52 BR 8

56 MOVI 0, 0, 0

60 STW 0, 12, -12

64 MOV 13, 0, 12

68 POP 12, 13, 4

72 POP 14, 13, 12

76 RET 14

80 MOVI 13, 0, 1012

84 PSH 14, 13, 4

88 MOVI 0, 0, 5

92 STW 0, 15, -96

96 MOVI 0, 0, 6

100 STW 0, 15, -108

104 LDW 0, 15, -108

108 PSH 0, 13, 4

112 LDW 0, 15, -120

116 PSH 0, 13, 4

120 BSR -120

124 POP 14, 13, 4

128 RET 14

132 bytes

1. Оператор If в цикле While.

Входной файл:

MODULE TwoCycles;

VAR

i, j, z: INTEGER;

BEGIN

i := 1;

j := 2;

WHILE (i > j) DO

i := 1;

j := 2;

IF (i > j) THEN

z := j;

ELSIF (i = j) THEN

z := 5;

ELSE

z := 10;

END;

END;

i := 2;

END TwoCycles.

Выходной файл:

entry 0

0 MOVI 13, 0, 1012

4 PSH 14, 13, 4

8 MOVI 0, 0, 1

12 STW 0, 15, -16

16 MOVI 0, 0, 2

20 STW 0, 15, -28

24 LDW 0, 15, -28

28 LDW 1, 15, -36

32 CMP 0, 0, 1

36 BLE 88

40 MOVI 0, 0, 1

44 STW 0, 15, -48

48 MOVI 0, 0, 2

52 STW 0, 15, -60

56 LDW 0, 15, -60

60 LDW 1, 15, -68

64 CMP 0, 0, 1

68 BLE 16

72 LDW 0, 15, -80

76 STW 0, 15, -88

80 BR 40

84 LDW 0, 15, -88

88 LDW 1, 15, -96

92 CMP 0, 0, 1

96 BNE 16

100 MOVI 0, 0, 5

104 STW 0, 15, -116

108 BR 12

112 MOVI 0, 0, 10

116 STW 0, 15, -128

120 BR -96

124 MOVI 0, 0, 2

128 STW 0, 15, -132

132 POP 14, 13, 4

136 RET 14

140 bytes

1. Цикл While.

Входной файл:

MODULE While;

VAR

i, j, z: INTEGER;

BEGIN

i := 1;

j := 2;

WHILE (i > j) DO

z := j;

i := j;

END;

i := 2;

END While.

Выходной файл:

entry 0

0 MOVI 13, 0, 1012

4 PSH 14, 13, 4

8 MOVI 0, 0, 1

12 STW 0, 15, -16

16 MOVI 0, 0, 2

20 STW 0, 15, -28

24 LDW 0, 15, -28

28 LDW 1, 15, -36

32 CMP 0, 0, 1

36 BLE 24

40 LDW 0, 15, -48

44 STW 0, 15, -56

48 LDW 0, 15, -56

52 STW 0, 15, -56

56 BR -32

60 MOVI 0, 0, 2

64 STW 0, 15, -68

68 POP 14, 13, 4

72 RET 14

76 bytes

1. Работа с массивом.

Входной файл:

MODULE Array;

VAR

a: ARRAY 6 OF INTEGER;

i: INTEGER;

BEGIN

a[0] := 7;

a[1] := 3;

Write(a[0]);

Write(a[1]);

i := 1;

a[i] := i;

a[5] := 0;

END Array.

Выходной файл:

entry 0

0 MOVI 13, 0, 996

4 PSH 14, 13, 4

8 MOVI 0, 0, 7

12 STW 0, 15, -36

16 MOVI 0, 0, 3

20 STW 0, 15, -40

24 LDW 0, 15, -48

28 WRD 0, 0, 0

32 LDW 0, 15, -52

36 WRD 0, 0, 0

40 MOVI 0, 0, 1

44 STW 0, 15, -72

48 LDW 0, 15, -76

52 CHKI 0, 0, 6

56 MULI 0, 0, 4

60 ADD 0, 15, 0

64 LDW 1, 15, -92

68 STW 1, 0, -92

72 MOVI 0, 0, 0

76 STW 0, 15, -80

80 POP 14, 13, 4

84 RET 14

88 bytes

1. Использованная литература
2. Н.Вирт «Построение компиляторов», 2010
3. Н.Вирт «Алгоритмы и структуры данных», 2011
4. А.Ахо «Компиляторы. Принципы, технологии и инструментарий», 2016
5. Приложение
   1. Синтаксис языка Оберон-0

ident = letter {letter | digit}.

integer = digit {digit}.

selector = {"." ident | "[" expression "]"}.

factor = ident selector | integer | "(" expression ")" | "~" factor.

term = factor {("\*" | "DIV" | "MOD" | "&") factor}.

SimpleExpression = ["+"|"-"] term {("+"|"-" | "OR") term}.

expression = SimpleExpression [("=" | "#" | "<" | "<=" | ">" | ">=") SimpleExpression].

assignment = ident selector ":=" expression.

ActualParameters = "(" [expression {"," expression}] ")" .

ProcedureCall = ident [ActualParameters].

IfStatement = "IF" expression "THEN" StatementSequence {"ELSIF" expression "THEN" StatementSequence} ["ELSE" StatementSequence] "END".

WhileStatement = "WHILE" expression "DO" StatementSequence "END".

statement = [assignment | ProcedureCall | IfStatement | WhileStatement].

StatementSequence = statement {";" statement}.

IdentList = ident {"," ident}.

ArrayType = "ARRAY" expression "OF" type.

FieldList = [IdentList ":" type].

RecordType = "RECORD" FieldList {";" FieldList} "END".

type = ident | ArrayType | RecordType.

FPSection = IdentList ":" type.

FormalParameters = "(" [FPSection {";" FPSection}] ")".

ProcedureHeading = "PROCEDURE" ident [FormalParameters].

ProcedureBody = declarations ["BEGIN" StatementSequence] "END".

ProcedureDeclaration = ProcedureHeading ";" ProcedureBody ident.

declarations = ["CONST" {ident "=" expression ";"}] ["TYPE" {ident "=" type ";"}] ["VAR" {IdentList ":" type ";"}] {ProcedureDeclaration ";"}.

module = "MODULE" ident ";" declarations ["BEGIN" StatementSequence] "END" ident "." .

* 1. Код целевой RISC-машины на языке Оберон

MODULE RISC;

IMPORT SYSTEM, Texts;  
 CONST

MemSize\* = 4096; ProgOrg = 2048; (\*in bytes\*)  
 MOV = 0; MVN = 1; ADD = 2; SUB = 3; MUL = 4; Div = 5; Mod = 6; CMP = 7;  
 MOVI = 16; MVNI = 17; ADDI = 18; SUBI = 19; MULI = 20; DIVI = 21; MODI = 22;

CMPI = 23; CHKI = 24; LDW = 32; LDB = 33; POP = 34; STW = 36; STB = 37;

PSH = 38; RD = 40; WRD= 41; WRH = 42; WRL = 43; BEQ = 48; BNE = 49; BLT = 50;

BGE = 51; BLE = 52; BGT = 53; BR = 56; BSR = 57; RET = 58;  
 VAR

IR: LONGINT;  
 N, Z: BOOLEAN;  
 R\*: ARRAY 16 OF LONGINT;  
 M\*: ARRAY MemSize DIV 4 OF LONGINT;  
 W: Texts.Writer;  
 PROCEDURE Execute\*(start: LONGINT; VAR in: Texts.Scanner; out: Texts.Text);  
 VAR opc, a, b, c, nxt: LONGINT;  
 BEGIN R[14] := 0; R[15] := start + ProgOrg;  
 LOOP (\*interpretation cycle\*)  
 nxt := R[15] + 4; IR := M[R[15] DIV 4];  
 opc := IR DIV 4000000H MOD 40H;  
 a := IR DIV 400000H MOD 10H;  
 b := IR DIV 40000H MOD 10H;  
 c := IR MOD 40000H;  
 IF opc < MOVI THEN c := R[IR MOD 10H]  
 ELSIF opc < BEQ THEN  
 c := IR MOD 40000H;  
 IF c >= 20000H THEN DEC(c, 40000H) END   
 ELSE c := IR MOD 4000000H;  
 IF c >= 2000000H THEN DEC(c, 4000000H) END   
 END ;  
 CASE opc OF  
 MOV, MOVI: R[a] := ASH(c, b) (\*arithmetic shift\*)  
 | MVN, MVNI: R[a] := -ASH(c, b)  
 | ADD, ADDI: R[a] := R[b] + c  
 | SUB, SUBI: R[a] := R[b] - c  
 | MUL, MULI: R[a] := R[b] \* c  
 | Div, DIVI: R[a] := R[b] DIV c  
 | Mod, MODI: R[a] := R[b] MOD c  
 | CMP, CMPI: Z := R[b] = c; N := R[b] < c  
 | CHKI: IF (R[a] < 0) OR (R[a] >= c) THEN R[a] := 0 END  
 | LDW: R[a] := M[(R[b] + c) DIV 4]   
 | POP: R[a] := M[(R[b]) DIV 4]; INC(R[b], c)  
 | STW: M[(R[b] + c) DIV 4] := R[a]  
 | STB: (\*not implemented\*)  
 | PSH: DEC(R[b], c); M[(R[b]) DIV 4] := R[a]  
 | RD: Texts.Scan(in); R[a] := in.i  
 | WRD: Texts.Write(W, " "); Texts.WriteInt(W, R[c], 1)  
 | WRH: Texts.WriteHex(W, R[c])  
 | WRL: Texts.WriteLn(W); Texts.Append(out, W.buf)  
 | BEQ: IF Z THEN nxt := R[15] + c\*4 END  
 | BNE: IF ~Z THEN nxt := R[15] + c\*4 END  
 | BLT: IF N THEN nxt := R[15] + c\*4 END  
 | BGE: IF ~N THEN nxt := R[15] + c\*4 END  
 | BLE: IF Z OR N THEN nxt := R[15] + c\*4 END  
 | BGT: IF ~Z & ~N THEN nxt := R[15] + c\*4 END  
 | BR: nxt := R[15] + c\*4  
 | BSR: nxt := R[15] + c\*4; R[14] := R[15] + 4  
 | RET: nxt := R[c MOD 10H];  
 IF nxt = 0 THEN EXIT END  
 END ;  
 R[15] := nxt  
 END  
 END Execute;  
 PROCEDURE Load\*(VAR code: ARRAY OF LONGINT; len: LONGINT);  
 VAR

i: INTEGER;  
 BEGIN i := 0;  
 WHILE i < len DO M[i + ProgOrg DIV 4] := code[i]; INC(i) END  
 END Load;  
BEGIN Texts.OpenWriter(W)  
END RISC.

* 1. main.c

#include <stdio.h>  
#include <malloc.h>  
#include "parser.h"  
  
char\* readCode(char\* fileName) {  
  
 //чтение исходного кода из файла  
 FILE\* sourceFile = NULL;  
 int fileSize = 0; //размер входного файла. нужен чтобы знать количество считываемых символов  
 char\* sourceCode;  
 printf("Oberon-0 Compiler 1.0\n");  
 printf("Opening source code file...\n");  
 if((sourceFile = fopen(fileName, "rb")) == NULL) {  
 printf("Opening source file error!\nCompilation terminated.");  
 return NULL;  
 }  
 printf("Reading source code...\n");  
 fseek(sourceFile, 0, SEEK\_END);  
 fileSize = ftell(sourceFile);  
 rewind(sourceFile); //возвращаемся в началу файла  
 sourceCode = (char\*)malloc((fileSize + 1) \* sizeof(char));  
 if(sourceCode == NULL)  
 return NULL;  
 fread(sourceCode, sizeof(char), fileSize, sourceFile);  
 sourceCode[fileSize] = '\0';  
 fclose(sourceFile);  
 return sourceCode;  
  
}  
  
void manual() {  
  
 //печать информации о программе и аргументах  
 printf("Oberon-0 Compiler 1.0.\n\n");  
 printf("Invalid command line format!\n");  
 printf("Correct format: \"Oberon-0-Compiler.exe source.txt\", where \"source.txt\" - file with source code in the Oberon-0 language.\n");  
 printf("Example of Oberon-0 code: \nMODULE If;\n"  
 "VAR \n"  
 "\t\ti, j, z: INTEGER;\n"  
 "\tBEGIN\n"  
 "\t\ti := 1;\n"  
 "\t\tj := 2;\n"  
 "\t\tIF (i > j) THEN \n"  
 "\t\t\tz := j;\n"  
 "\t\tELSIF (i = j) THEN\n"  
 "\t\t\tz := 5;\n"  
 "\t\tELSE \n"  
 "\t\t\tz := 10;\n"  
 "\t\tEND;\n"  
 "END If.\n");  
 printf("Compiler compiles for RISC machine.\n");  
 printf("Compiled code is saved in the current directory in \"output.txt\" file.\n");  
 return;  
  
}  
  
int main(int argc, char\* argv[]) {  
  
  
 //вид командной строки: compiler.exe имя\_входного\_файла  
 if(argc != 2)  
 manual(); //печать информации о программе и аргументах  
 else {  
 char\* inputFileName = NULL; //имя входного файла  
 char\* sourceCode = NULL; //исходный код  
 inputFileName = argv[1];  
 sourceCode = readCode(inputFileName); //чтение исходного кода из файла  
 if(sourceCode != NULL) {  
 compile(sourceCode); //если код считан успешно - начало компиляции  
 }  
 if(inputFileName != NULL)  
 free(inputFileName);  
 if(sourceCode != NULL)  
 free(sourceCode);  
 }  
 return 0;  
  
}

* 1. parser.h

#ifndef \_PARSER\_H  
#define \_PARSER\_H  
  
#define identLength 16 //максимальная длина идентификатора  
#define keyTabSize 34 //количество терминальных символов языка  
#define maxCodeSize 1000 //максимальная длина кода  
#define wordSize 4 //размер одного слова  
  
struct symbolLex { //структура для соответствия терминального символа и его кода  
  
 int symbol;  
 char identifier[12];  
  
};  
  
struct object { //структура для представления одного объекта  
  
 int class; //класс объекта: переменная, функция, константа, параметр, поле, тип, ...  
 int level; //уровень вложенности своего поля видимости  
 struct object\* nextObject; //указатель на следующий элемент в своем поле  
 struct object\* previousScope; //указатель на предыдущее поле  
 struct type\* classType; //тип класса (например, int или bool)  
 char name[identLength]; //имя объекта  
 int value; //значение/сдвиг относительно базового адреса  
  
};  
  
struct type { //структура для предствления типа  
  
 int classType; //bool, int, массив, либо запись  
 struct object\* fields; //поля записи  
 struct type\* baseType; //тип массива  
 int size; //размер  
 int length; //длина(массива)  
  
};  
  
struct item { //структура для представления объекта с учетом его синтаксиса  
  
 int mode; //режим выражения. var, const, reg  
 int level; //уровень вложенности своего поля видимости  
 struct type\* classType; //тип  
 int a; //первый аргумент команды  
 int b; //второй аргумент команды  
 int c; //третий аргумент команды  
 int storage; //регистр  
  
};  
  
struct parameters {  
  
 char\* sourceCode; //исходный код  
 int lastPosition; //последняя просмотренная позиция исходного кода  
 char lastLexeme[identLength]; //последний считанный идентификатор  
 int lastLexemeCode; //код считанного последнего символа  
 int lastLexemeValue; //последнее считанное число  
 int error; //флаг ошибки  
 int errorPosition; //флаг, указывающий на последнюю позицию ошибки  
 FILE\* reportFile; //файл с информацией о компиляции  
 struct symbolLex\* keyTab[keyTabSize]; //структура для сканера. хранит множество терминальных символов  
 int currentLevel; //текущий уровень вложенности  
 int PC; //счетчик команд  
 int registers[16]; //множество регистров  
 struct type\* boolType; //глобальный тип bool  
 struct type\* intType; //глобальный тип int  
 struct object\* topScope;  
 struct object\* guard;  
 struct object\* universe; //глобальные области видимости  
 int code[maxCodeSize]; //скомпилированный код  
 int entryAddress; //адрес входа в программу  
 int linesCounter; //счетчик строк  
 int posCounter; //счетчик позиции в строке  
 struct item\* emptyItem; //пустой item для возврата при ошибках памяти  
 struct object\* emptyObject; //пустой object для возврата при ошибках памяти  
 struct type\* emptyType; //пустой type для возврата при ошибках памяти  
  
};  
  
void compile(char\*);  
void initItem(struct item \*, struct parameters \*);  
  
#endif

* 1. parser.c

#include <stdio.h>  
#include <malloc.h>  
#include <string.h>  
#include "scanner.h"  
#include "codeGenerator.h"  
  
void signal(char msg[], struct parameters\* storage) {  
  
 //печать сообщения в report.txt и в консоль  
 fprintf(storage->reportFile, "%s\r\n", msg);  
 printf("%s\n", msg);  
  
}  
  
void freeObject(struct object \*obj, struct parameters\* storage) {  
  
 //освобождение памяти объекта  
 if(obj != NULL) {  
 if(obj->nextObject != NULL && obj->nextObject != obj) {  
 freeObject(obj->nextObject, storage);  
 free(obj->nextObject);  
 }  
 if(obj->previousScope != NULL && obj->previousScope != obj) {  
 freeObject(obj->previousScope, storage);  
 free(obj->previousScope);  
 }  
 }  
  
}  
  
void freeType(struct type \*typ, struct parameters\* storage) {  
  
 //освобождение памяти типа  
 if(typ != NULL) {  
 if(typ->fields != NULL && typ->fields != storage->guard) {  
 freeObject(typ->fields, storage);  
 free(typ->fields);  
 }  
 }  
  
}  
  
void initObject(struct object \*obj, struct parameters \*storage) {  
  
 //инициализация "пустого "object  
 obj->level = 0;  
 obj->classType = storage->emptyType;  
 obj->class = 0;  
 obj->previousScope = storage->emptyObject;  
 obj->nextObject = storage->emptyObject;  
 obj->value = 0;  
 strcpy(obj->name, "\0");  
  
}  
  
void initType(struct type \*typ, struct parameters \*storage) {  
  
 //инициализация "пустого" type  
 typ->classType = 0;  
 typ->baseType = storage->emptyType;  
 typ->fields = storage->emptyObject;  
 typ->length = 0;  
 typ->size = 0;  
  
}  
  
void initItem(struct item \*it, struct parameters \*storage) {  
  
 //инициализация "пустого" item  
 it->classType = storage->emptyType;  
 it->c = 0;  
 it->level = 0;  
 it->mode = 0;  
 it->b = 0;  
 it->a = 0;  
 it->storage = 0;  
  
}  
  
int enterUniverse(int class, int value, char \*name, struct type \*type, struct parameters \*storage) {  
  
 //создание объектов в universe  
 struct object\* newObject = (struct object\*)malloc(sizeof(struct object));  
 if(newObject == NULL) {  
 mark("Memory allocation error in function enterUniverse", storage);  
 return -1;  
 }  
 initObject(newObject, storage);  
 newObject->class = class;  
 newObject->value = value;  
 strcpy(newObject->name, name);  
 newObject->classType = type;  
 newObject->previousScope = NULL;  
 newObject->nextObject = storage->topScope->nextObject;  
 storage->topScope->nextObject = newObject;  
 return 0;  
  
}  
  
void enterKeyTab(int symbol, char \*name, struct symbolLex \*\*keyTab, int \*index) {  
  
 //инициализация элемента keyTab  
 keyTab[\*index]->symbol = symbol;  
 strcpy(keyTab[\*index]->identifier, name);  
 (\*index)++;  
  
}  
  
int initLexical(struct parameters \*storage) {  
  
 //инициализация структуры для сканнера  
 int i;  
 for(i = 0; i < 34; i++) {  
 storage->keyTab[i] = (struct symbolLex\*)malloc(sizeof(struct symbolLex));  
 if(storage->keyTab[i] == NULL) {  
 mark("Memory allocation error in function initLexical", storage);  
 return -1;  
 }  
 }  
 i = 0;  
 enterKeyTab(nullLexical, "BY", storage->keyTab, &i);  
 enterKeyTab(doLexical, "DO", storage->keyTab, &i);  
 enterKeyTab(ifLexical, "IF", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "IN", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "IS", storage->keyTab, &i);  
 enterKeyTab(ofLexical, "OF", storage->keyTab, &i);  
 enterKeyTab(orLexical, "OR", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "TO", storage->keyTab, &i);  
 enterKeyTab(endLexical, "END", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "FOR", storage->keyTab, &i);  
 enterKeyTab(modLexical, "MOD", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "NIL", storage->keyTab, &i);  
 enterKeyTab(varLexical, "VAR", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "CASE", storage->keyTab, &i);  
 enterKeyTab(elseLexical, "ELSE", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "EXIT", storage->keyTab, &i);  
 enterKeyTab(thenLexical, "THEN", storage->keyTab, &i);  
 enterKeyTab(typeLexical, "TYPE", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "WITH", storage->keyTab, &i);  
 enterKeyTab(arrayLexical, "ARRAY", storage->keyTab, &i);  
 enterKeyTab(beginLexical, "BEGIN", storage->keyTab, &i);  
 enterKeyTab(constLexical, "CONST", storage->keyTab, &i);  
 enterKeyTab(elsifLexical, "ELSIF", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "IMPORT", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "UNTIL", storage->keyTab, &i);  
 enterKeyTab(whileLexical, "WHILE", storage->keyTab, &i);  
 enterKeyTab(recordLexical, "RECORD", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "REPEAT", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "RETURN", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "POINTER", storage->keyTab, &i);  
 enterKeyTab(procedureLexical, "PROCEDURE", storage->keyTab, &i);  
 enterKeyTab(divLexical, "DIV", storage->keyTab, &i);  
 enterKeyTab(nullLexical, "LOOP", storage->keyTab, &i);  
 enterKeyTab(moduleLexical, "MODULE", storage->keyTab, &i);  
 return 0;  
  
}  
  
int initTypes(struct parameters\* storage) {  
  
 //инициализация базовых типов. int и bool. их размер - 4  
 storage->boolType = (struct type\*)malloc(sizeof(struct type));  
 if(storage->boolType == NULL) {  
 mark("Memory allocation error in function initTypes", storage);  
 return -1;  
 }  
 initType(storage->boolType, storage);  
 storage->boolType->classType = BooleanGen;  
 storage->boolType->size = 4;  
 storage->intType = (struct type\*)malloc(sizeof(struct type));  
 if(storage->intType == NULL) {  
 mark("Memory allocation error in function initTypes", storage);  
 return -1;  
 }  
 initType(storage->intType, storage);  
 storage->intType->classType = IntegerGen;  
 storage->intType->size = 4;  
 return 0;  
  
}  
  
int initScopes(struct parameters\* storage) {  
  
 //инициализация областей видимости - scopes.  
 //topScope - указатель на "верхний" scope  
 //guard - "закрывающий" элемент. нужен для удобного поиска элементов  
 //universe - внешний scope  
 //в dsc хранится предыдущий scope  
 //в next хранятся объекты текущего scope  
 storage->topScope = (struct object\*)malloc(sizeof(struct object));  
 if(storage->topScope == NULL) {  
 mark("Memory allocation error in function initScopes", storage);  
 return -1;  
 }  
 initObject(storage->topScope, storage);  
 storage->guard = (struct object\*)malloc(sizeof(struct object));  
 if(storage->guard == NULL) {  
 mark("Memory allocation error in function initScopes", storage);  
 return -1;  
 }  
 initObject(storage->guard, storage);  
 storage->universe = (struct object\*)malloc(sizeof(struct object));  
 if(storage->universe == NULL) {  
 mark("Memory allocation error in function initScopes", storage);  
 return -1;  
 }  
 initObject(storage->universe, storage);  
 storage->guard->class = VarGen;  
 storage->guard->classType = storage->intType;  
 storage->guard->value = 0;  
 initObject(storage->topScope, storage);  
 openScope(storage); //открытие universe  
 if(enterUniverse(TypGen, 1, "BOOLEAN", storage->boolType, storage) != 0)  
 return -1;  
 if(enterUniverse(TypGen, 2, "INTEGER", storage->intType, storage) != 0)  
 return -1;  
 if(enterUniverse(ConstGen, 1, "TRUE", storage->boolType, storage) != 0)  
 return -1;  
 if(enterUniverse(ConstGen, 0, "FALSE", storage->boolType, storage) != 0)  
 return -1; //инициализация типов  
 if(enterUniverse(SProcGen, 1, "Read", NULL, storage) != 0)  
 return -1;  
 if(enterUniverse(SProcGen, 2, "Write", NULL, storage) != 0)  
 return -1;  
 if(enterUniverse(SProcGen, 3, "WriteHex", NULL, storage) != 0)  
 return -1;  
 if(enterUniverse(SProcGen, 4, "WriteLn", NULL, storage) != 0)  
 return -1; //добавление глобальных процедур в universe  
 storage->universe = storage->topScope;  
 return 0;  
  
}  
  
int init(struct parameters\* storage, char\* sourceCode) {  
  
 //инициализация storage  
 storage->sourceCode = sourceCode;  
 storage->lastPosition = 0;  
 strcpy(storage->lastLexeme, "\0");  
 storage->lastLexemeCode = -1;  
 storage->lastLexemeValue = -1;  
 storage->error = 0;  
 storage->errorPosition = -1;  
 storage->reportFile = NULL;  
 if((storage->reportFile = fopen("report.txt", "wb")) == NULL) {  
 printf("Opening report.txt file error!\n");  
 return -1;  
 }  
 FILE\* outputFile = NULL;  
 if((outputFile = fopen("output.txt", "wb")) == NULL) {  
 printf("Opening output.txt file error!\n");  
 return -1;  
 }  
 else  
 fclose(outputFile); //очистка файла output.txt  
 storage->emptyType = (struct type\*)malloc(sizeof(struct type));  
 if(storage->emptyType == NULL) {  
 mark("Memory allocation error in function init", storage);  
 return -1;  
 }  
 storage->emptyType->classType = 0;  
 storage->emptyType->baseType = storage->emptyType;  
 storage->emptyType->length = 0;  
 storage->emptyType->size = 0;  
 storage->emptyObject = (struct object\*)malloc(sizeof(struct object));  
 if(storage->emptyObject == NULL) {  
 mark("Memory allocation error in function init", storage);  
 return -1;  
 }  
 storage->emptyType->fields = storage->emptyObject;  
 initObject(storage->emptyObject, storage);  
 storage->emptyItem = (struct item\*)malloc(sizeof(struct item));  
 if(storage->emptyItem == NULL) {  
 mark("Memory allocation error in function init", storage);  
 return -1;  
 }  
 initItem(storage->emptyItem, storage);  
 if(initLexical(storage) != 0)  
 return -1; //инициализация keyTab  
 storage->currentLevel = 0;  
 storage->PC = 0;  
 memset(storage->registers, 0, sizeof(int) \* 16); //обнуление регистров. 0 - не занят  
 if(initTypes(storage) != 0)  
 return -1; //инициализация базовых типов  
 if(initScopes(storage) != 0)  
 return -1; //инициализация topScope, guard, universe  
 for(int i = 0; i < maxCodeSize; i++)  
 storage->code[i] = 0;  
 storage->entryAddress = 0;  
 storage->posCounter = 0;  
 storage->linesCounter = 0;  
 return 0;  
  
}  
  
struct object\* createNewObject(int class, struct parameters\* storage) {  
  
 //создание нового объекта в текущем scope  
 struct object\* newObject = (struct object\*)malloc(sizeof(struct object));  
 if(newObject == NULL) {  
 mark("Memory allocation error in function createNewObject", storage);  
 return storage->guard;  
 }  
 initObject(newObject, storage);  
 struct object\* buffer;  
 buffer = storage->topScope;  
 strcpy(storage->guard->name, storage->lastLexeme);  
 while(strcmp(buffer->nextObject->name, storage->lastLexeme))  
 buffer = buffer->nextObject;  
 if(buffer->nextObject == storage->guard) { //если дошли до guard - такого объекта в этом scope нет - создание объекта  
 strcpy(newObject->name, storage->lastLexeme);  
 newObject->class = class;  
 newObject->nextObject = storage->guard;  
 buffer->nextObject = newObject;  
 }  
 else { //иначе - ошибка  
 newObject = buffer->nextObject;  
 mark("Declared already", storage);  
 }  
 strcpy(storage->guard->name, "\0"); //"обнуление" guard  
 return newObject;  
  
}  
  
struct object\* findObject(struct parameters\* storage) {  
  
 //поиск объекта во всех scope  
 struct object\* bufferHead;  
 struct object\* buffer;  
 struct object\* object = (struct object\*)malloc(sizeof(struct object));  
 if(object == NULL) {  
 mark("Memory allocation error in function findObject", storage);  
 return storage->guard;  
 }  
 initObject(object, storage);  
 bufferHead = storage->topScope;  
 strcpy(storage->guard->name, storage->lastLexeme);  
 do {  
 buffer = bufferHead->nextObject;  
 while(strcmp(buffer->name, storage->lastLexeme))  
 buffer = buffer->nextObject;  
 if(buffer != storage->guard)  
 object = buffer; //если объект в scope найден  
 else if(bufferHead == storage->universe) { //если не найден и уже на внешнем scope  
 object = buffer;  
 mark("Not declared", storage);  
 buffer = NULL; //для выхода из цикла  
 }  
 else  
 bufferHead = bufferHead->previousScope; //иначе на уровень ниже  
 } while(buffer == storage->guard);  
 strcpy(storage->guard->name, "\0"); //"обнуление" guard  
 return object;  
  
}  
  
struct item\* selector(struct item\* item, struct parameters\* storage) { //передается уже инициализированный элемент  
  
 //чтение элемента массива / поля записи  
 struct item\* newItem = item; //возвращаемый item  
 //если нет обращения к элементам, то это просто переменная  
 while((storage->lastLexemeCode == lbrakLexical) || (storage->lastLexemeCode == periodLexical)) { // [ либо .  
 if(storage->lastLexemeCode == lbrakLexical) { //обращение к элементу массива  
 get(storage);  
 struct item\* indexExpression; //для выражения индекса массива  
 indexExpression = expression(storage); //x[y]. получение y  
 if(newItem->classType->classType == ArrayGen)  
 arrayElem(newItem, indexExpression, storage); //если переменная - массив, получение значение по индексу  
 else  
 mark("Not an array", storage); //иначе - ошибка  
 if(indexExpression != NULL)  
 free(indexExpression);  
 if(storage->lastLexemeCode == rbrakLexical)  
 get(storage);  
 else  
 mark("]?", storage);  
 } //если обращение к элементу массива  
 else {  
 get(storage);  
 if(storage->lastLexemeCode == identLexical) { //x.y. y - всегда индентификатор!  
 if(newItem->classType->classType == RecordGen) { //если тип x - запись  
 struct object\* fieldObject; //для поиска поля записи  
 fieldObject = findField(newItem->classType->fields, storage); //поиск, если ли такое поле у x  
 get(storage);  
 if(fieldObject != storage->guard) {  
 getField(newItem, fieldObject); //если есть - запись  
 }  
 else  
 mark("Undef record field", storage);  
 }  
 else  
 mark("Not a record", storage);  
 }  
 else  
 mark("Identifier?", storage);  
 } //если обращение к элементу записи  
 }  
 return newItem;  
  
}  
  
struct item\* factor(struct parameters\* storage) { //множители  
  
 //анализ идентификатора либо числа  
 struct item\* currentItem;  
 struct object\* currentObject;  
 if(storage->lastLexemeCode < lparenLexical) {  
 mark("Must be identifier or number", storage);  
 do {  
 get(storage);  
 } while(storage->lastLexemeCode < lparenLexical);  
 } //проверка допустимость символов  
 if(storage->lastLexemeCode == identLexical) {  
 currentObject = findObject(storage); //поиск объекта в существующих  
 get(storage);  
 currentItem = makeItem(currentObject, storage); //создание соответствующего item  
 currentItem = selector(currentItem, storage); //либо тот же идент, либо элемент массива, либо поле записи  
 } //идентификатор  
 else if(storage->lastLexemeCode == numberLexical) {  
 currentItem = makeConstItem(storage->intType, storage->lastLexemeValue, storage); //создание item с постоянным значением  
 get(storage);  
 } //число  
 else if(storage->lastLexemeCode == lparenLexical) {  
 get(storage);  
 currentItem = expression(storage); //( \*expression\* )  
 if(storage->lastLexemeCode == rparenLexical)  
 get(storage);  
 else  
 mark("Lost )", storage);  
 } //выражение в скобках  
 else if(storage->lastLexemeCode == notLexical) {  
 get(storage);  
 currentItem = factor(storage);  
 singleGenerate(notLexical, currentItem, storage); //выражение со знаком -  
 } //отрицание  
 else {  
 mark("Factor?", storage);  
 currentItem = makeItem(storage->guard, storage); //возвращение guard, если нет допустимой переменной  
 }  
 return currentItem;  
  
}  
  
struct item\* term(struct parameters\* storage) {  
  
 //анализ выражений: x \* y, x DIV y, x MOD y, x & y  
 struct item\* leftExpression;  
 int sign;  
 leftExpression = factor(storage); //получение левого выражения  
 while((storage->lastLexemeCode >= timesLexical) && (storage->lastLexemeCode <= andLexical)) {  
 sign = storage->lastLexemeCode;  
 get(storage);  
 if(storage->lastLexemeCode == andLexical)  
 singleGenerate(sign, leftExpression, storage); //загрузка в регистр для лог. операции  
 struct item\* rightExpression;  
 rightExpression = factor(storage); //получение правого выражения  
 termGenerate(sign, leftExpression, rightExpression, storage); //получение выражения с учетом знака  
 if(rightExpression != NULL)  
 free(rightExpression);  
 } //для умножения, div, mod, &  
 return leftExpression;  
  
}  
  
struct item\* simpleExpression(struct parameters\* storage) {  
  
 //анализ суммы, разности, |. учитывается знак перед первым аргументом (- или +)  
 struct item\* leftExpression;  
 int sign;  
 if(storage->lastLexemeCode == plusLexical) {  
 get(storage);  
 leftExpression = term(storage); //получение левого выражения  
 } //+item  
 else if(storage->lastLexemeCode == minusLexical){  
 get(storage);  
 leftExpression = term(storage);  
 singleGenerate(minusLexical, leftExpression, storage); //выражение со знаком -  
 } //-item  
 else {  
 leftExpression = term(storage);  
 } //нет знака  
 while((storage->lastLexemeCode >= plusLexical) && (storage->lastLexemeCode <= orLexical)) {  
 sign = storage->lastLexemeCode; //получение знака  
 get(storage);  
 if(sign == orLexical)  
 singleGenerate(sign, leftExpression, storage); //загрузка в регистры для лог. операции  
 struct item\* rightExpression = term(storage); //получение правого выражения  
 termGenerate(sign, leftExpression, rightExpression, storage); //получение выражения с учетом знака  
 if(rightExpression != NULL)  
 free(rightExpression);  
 } //анализ суммы/разности/OR  
 return leftExpression;  
  
}  
  
struct item\* expression(struct parameters\* storage) {  
  
 //анализ либо переменной, либо лог. выражения x ( = | # | < | <= | > | >= ) y  
 int sign;  
 struct item\* leftExpression = simpleExpression(storage); //получение левого выражения  
 if((storage->lastLexemeCode >= eqlLexical) && (storage->lastLexemeCode <= gtrLexical)) {  
 sign = storage->lastLexemeCode;  
 get(storage);  
 struct item\* rightExpression = simpleExpression(storage); //получение правого выражение  
 relation(sign, leftExpression, rightExpression, storage); //получение выражения с учетом знака  
 if(rightExpression != NULL)  
 free(rightExpression);  
 } //если след. знак ( = | # | < | <= | > | >= )  
 return leftExpression;  
  
}  
  
struct type\* getType(struct parameters\* storage) {  
  
 //чтение типа. (array, record, intType, boolType, созданный тип)  
 struct type\* type = (struct type\*)malloc(sizeof(struct type)); //полученный тип  
 if(type == NULL) {  
 mark("Memory allocation error in function getType", storage);  
 return storage->intType;  
 }  
 initType(type, storage);  
 if((storage->lastLexemeCode != identLexical) && (storage->lastLexemeCode < arrayLexical)) {  
 mark("No type?", storage);  
 do {  
 get(storage);  
 } while((storage->lastLexemeCode != identLexical) && (storage->lastLexemeCode < arrayLexical));  
 } //проверка допустимых символов  
 if(storage->lastLexemeCode == identLexical) { //если это объявленный тип  
 struct object\* objectBuffer;  
 objectBuffer = findObject(storage); //поиск в существующих объектах  
 get(storage);  
 if(objectBuffer->class == TypGen) //если класс найденного объекта - тип, то присваивание  
 type = objectBuffer->classType;  
 else  
 mark("Type identifier?", storage);  
 }  
 else if(storage->lastLexemeCode == arrayLexical) { //ARRAY \*expression\* OF \*type\*  
 get(storage);  
 struct item\* expressionItem;  
 expressionItem = expression(storage); //получение индекса  
 if((expressionItem->mode != ConstGen) || (expressionItem->a < 0)) //размер массива - ПОСТОЯННАЯ, значение - НЕ ОТРИЦАТЕЛЬНОЕ  
 mark("Bad index", storage);  
 if(storage->lastLexemeCode == ofLexical)  
 get(storage);  
 else  
 mark("Lost OF", storage);  
 type->classType = ArrayGen;  
 type->baseType = getType(storage); //баз. тип  
 type->length = expressionItem->a; // "длина" массива  
 if(expressionItem != NULL)  
 free(expressionItem);  
 type->size = type->length \* type->baseType->size; //размер массива = длина \* размер типа  
 }  
 else if(storage->lastLexemeCode == recordLexical) { //RECORD \*ident1\*, \*ident2\* : \*type1\* {; \*identN\* : \*typeN\*} END  
 get(storage);  
 type->classType = RecordGen;  
 type->size = 0;  
 openScope(storage); //открытие scope для записи  
 while((storage->lastLexemeCode == semicolonLexical) || (storage->lastLexemeCode == identLexical)) { //чтение полей  
 if(storage->lastLexemeCode == identLexical) {  
 struct object\* objectBuffer;  
 struct object\* listHeader;  
 listHeader = identifiersList(FldGen, storage); //запись объектов поля в scope  
 struct type\* typeBuffer;  
 typeBuffer = getType(storage); //получение типа поля  
 objectBuffer = listHeader;  
 while(objectBuffer != storage->guard) { //запись типа для всех идентификаторов  
 objectBuffer->classType = typeBuffer;  
 objectBuffer->value = type->size; //значение - смещение относительно базового адреса записи  
 type->size += typeBuffer->size; //увеличение смещения  
 objectBuffer = objectBuffer->nextObject;  
 }  
 }  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else if(storage->lastLexemeCode == identLexical)  
 mark("Lost ;", storage);  
 }  
 type->fields = storage->topScope->nextObject; //->fields - указатель на первое поле. структура как в scope  
 closeScope(storage); //закрытие scope  
 if(storage->lastLexemeCode == endLexical)  
 get(storage);  
 else  
 mark("Lost END", storage);  
 }  
 else  
 mark("Identifier type?", storage);  
 if(type != NULL)  
 return type;  
 else  
 return storage->intType; //если тип не идентифицирован, считается, что это int  
  
}  
  
struct object\* identifiersList(int class, struct parameters \*storage) {  
  
 //чтение всех идентификаторов вида ident1, ... , ident2 : . возврат указателя на первый элемент  
 struct object\* headerObject = (struct object\*)malloc(sizeof(struct object));  
 if(headerObject == NULL) {  
 mark("Memory allocation error in function identifiersList", storage);  
 return storage->guard;  
 }  
 initObject(headerObject, storage);  
 if(storage->lastLexemeCode == identLexical) {  
 headerObject = createNewObject(class, storage); //в first первый идентификатор  
 get(storage);  
 while(storage->lastLexemeCode == commaLexical) { //пока , . ident1, ident2, ...  
 get(storage);  
 if(storage->lastLexemeCode == identLexical) {  
 createNewObject(class, storage);  
 get(storage);  
 }  
 else  
 mark("Identifier?", storage);  
 }  
 if(storage->lastLexemeCode == colonLexical)  
 get(storage);  
 else  
 mark("Lost :", storage);  
 }  
 return headerObject;  
  
}  
  
void openScope(struct parameters\* storage) {  
  
 //открытие нового scope  
 struct object\* newScope = (struct object\*)malloc(sizeof(struct object));  
 if(newScope == NULL) {  
 mark("Memory allocation error in function openScope", storage);  
 return;  
 }  
 initObject(newScope, storage);  
 newScope->class = HeadGen; //первый элемент scope - head  
 newScope->previousScope = storage->topScope; //dsc - указатель на предыдущий scope  
 newScope->nextObject = storage->guard; //след. элемента нет  
 storage->topScope = newScope; //новая вершина scopes  
  
}  
  
int declarations(struct parameters\* storage) { //возвращается размер переменных  
  
 //чтение объявлений CONST, TYPE, VAR  
 int varSize = 0;  
 while((storage->lastLexemeCode >= constLexical) && (storage->lastLexemeCode <= varLexical)) { //пока const, type, var  
 if(storage->lastLexemeCode == constLexical) {  
 get(storage);  
 while(storage->lastLexemeCode == identLexical) { //CONST \*ident1\* = \*expr\*; \*ident2\* = \*expr\*; ...  
 struct object\* currentObject; //структура для рассматриваемого объекта  
 struct item\* currentExpression; //структура для анализа выражения  
 currentObject = createNewObject(ConstGen, storage); //создание объекта для константы  
 get(storage);  
 if(storage->lastLexemeCode == eqlLexical)  
 get(storage);  
 else  
 mark("Lost :=", storage); //так как это const, должна быть инициализация  
 currentExpression = expression(storage); // CONST \*ident1\* = \*expr\*;  
 if(currentExpression->mode == ConstGen) {  
 currentObject->value = currentExpression->a; //поле значения  
 currentObject->classType = currentExpression->classType; //поле типа  
 }  
 else  
 mark("Expression is not a constant", storage);  
 if(currentExpression != NULL)  
 free(currentExpression);  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else  
 mark("Lost ;", storage); //после каждой константы ;  
 }  
 }  
 if(storage->lastLexemeCode == typeLexical) {  
 get(storage);  
 while(storage->lastLexemeCode == identLexical) { //TYPE \*ident1\* = \*type\*; \*ident2\* = \*type\*; ...  
 struct object\* currentObject = createNewObject(TypGen, storage); //создание объекта типа  
 get(storage);  
 if(storage->lastLexemeCode == eqlLexical)  
 get(storage);  
 else  
 mark("Lost :=", storage); //должно быть объявление типа  
 currentObject->classType = getType(storage); //получение объявления типа  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else  
 mark("Lost ;", storage); //после каждого типа ;  
 }  
 }  
 if(storage->lastLexemeCode == varLexical) {  
 get(storage);  
 while(storage->lastLexemeCode == identLexical) { //\*identList1\* : \*type; \*identList2\* : \*type\*; ...  
 struct object\* currentObject; //структура для рассматриваемого объекта  
 struct object\* identListHead; //для множества идентификаторов. указатель на первый элемент  
 struct type\* currentType; //тип переменной  
 identListHead = identifiersList(VarGen, storage); //получение списка ВСЕХ идентификаторов через запятую, запись  
 //в текущий scope, получение указателя на первый записанный объект  
 currentType = getType(storage); //получение типа  
 currentObject = identListHead;  
 while(currentObject != storage->guard) { //для всех полученных объектов  
 varSize += currentType->size; //размер типа + к varSize  
 currentObject->classType = currentType; //тип  
 currentObject->level = storage->currentLevel; //текущий уровень. для отличия переменных разных scope  
 currentObject->value = -varSize; //смещение относительно базового адреса  
 currentObject = currentObject->nextObject;  
 }  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else  
 mark("Lost ;", storage); //после каждой переменной ;  
 }  
 }  
 if((storage->lastLexemeCode >= constLexical) && (storage->lastLexemeCode <= varLexical))  
 mark("Wrong declarations order", storage); //const->type->var !  
 }  
 return varSize;  
  
}  
  
int parametersBlockAnalyzer(struct parameters \*storage) {  
  
 //чтение блока формальных параметров функции  
 struct object\* listHead; //указатель на первый объект  
 struct object\* objectBuffer;  
 struct type\* currentType; //тип параметров  
 int typeSize; //размер типа  
 int parametersSize = 0; //размер блока  
 listHead = identifiersList(VarGen, storage); //VAR - VarGen, - параметр-значение  
 if(storage->lastLexemeCode == identLexical) { //ident - тип переменных  
 objectBuffer = findObject(storage); //поиск типа в существующих  
 get(storage);  
 if(objectBuffer->class == TypGen)  
 currentType = objectBuffer->classType; //если тип существует - присваивание  
 else {  
 mark("Identifier is not a type", storage);  
 currentType = storage->intType; //иначе - ошибка. присваевание int  
 }  
 }  
 else {  
 mark("Identifier type?", storage);  
 currentType = storage->intType; //если тип не указан - присваивание int  
 }  
 typeSize = currentType->size;  
 if(currentType->classType >= ArrayGen)  
 mark("Incorrect type of parameter", storage); //параметром может быть только int и bool  
 objectBuffer = listHead;  
 while(objectBuffer != storage->guard) { //инициализация типа для всех новых объектов. подсчет их размера  
 objectBuffer->classType = currentType;  
 parametersSize += typeSize;  
 objectBuffer = objectBuffer->nextObject;  
 }  
 return parametersSize; //возврат размера блока  
  
}  
  
void procedureAnalyzer(struct parameters \*storage) {  
  
 //анализ процедур. PROCEDURE \*ident\*(\*FormalParameters\*) \*ProcBody\*  
 struct object\* currentProcedure; //объект процедуры  
 char procedureName[identLength]; //имя процедуры  
 int currentBlockSize, parametersSize;  
 int markSize = 8; //размер "следа" процедуры  
 get(storage);  
 if(storage->lastLexemeCode == identLexical) { //PROCEDURE \*ident\*  
 strcpy(procedureName, storage->lastLexeme); //получение идент. процедуры  
 currentProcedure = createNewObject(ProcGen, storage); //создание нового объекта для процедуры в текущем scope  
 get(storage);  
 parametersSize = markSize; //размер параметров - след процедуры  
 changeLevel(1, storage); //увеличение уровня вложенности  
 openScope(storage); //открытие scope для новой процедуры  
 currentProcedure->value = -1;  
 if(storage->lastLexemeCode == lparenLexical) { //параметры функции (?? : ??; ?? : ??; ...)  
 get(storage);  
 if(storage->lastLexemeCode == rparenLexical) //если ()  
 get(storage);  
 else {  
 parametersSize += parametersBlockAnalyzer(storage); //чтение ident1, ident2, .. : type. увеличение размера блока параметров  
 while(storage->lastLexemeCode == semicolonLexical) { //между 2 и > должно быть ;  
 get(storage);  
 parametersSize += parametersBlockAnalyzer(storage);  
 }  
 if(storage->lastLexemeCode == rparenLexical)  
 get(storage);  
 else  
 mark("Lost )", storage);  
 }  
 }  
 struct object\* objectBuffer;  
 objectBuffer = storage->topScope->nextObject; //переход к первому параметру  
 currentBlockSize = parametersSize;  
 while(objectBuffer != storage->guard) { //полю val каждого объекта присваивается значение смещения относительно  
 //кадра активации. смещение положительное  
 objectBuffer->level = storage->currentLevel;  
 if(objectBuffer->class == ParGen)  
 currentBlockSize -= wordSize;  
 else  
 currentBlockSize -= objectBuffer->classType->size; //инкремент. смещение каждую итерацию  
 objectBuffer->value = currentBlockSize; //присваивание смещения  
 objectBuffer = objectBuffer->nextObject;  
 }  
 currentProcedure->previousScope = storage->topScope->nextObject; //дескриптор процесса - его параметры  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else  
 mark(";?", storage); //после PROCEDURE ident() должно быть ;  
 currentBlockSize = declarations(storage); //лок. параметры процедуры  
 while(storage->lastLexemeCode == procedureLexical) {  
 procedureAnalyzer(storage);  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else  
 mark(";?", storage);  
 } //внутр. функции функции  
 currentProcedure->value = storage->PC; //value - точка входа в процедуру  
 prologue(currentBlockSize, storage); //запись пролога процедуры  
 if(storage->lastLexemeCode == beginLexical) { //тело процедуры BEGIN procbody END \*procident\*  
 get(storage);  
 statements(storage); //последовательность операторов  
 }  
 if(storage->lastLexemeCode == endLexical)  
 get(storage);  
 else  
 mark("Lost END", storage);  
 if(storage->lastLexemeCode == identLexical) {  
 if(strcmp(procedureName, storage->lastLexeme))  
 mark("No match procedure identifier", storage); //идент. в конце должен совпадать с идент. процедуры  
 get(storage);  
 }  
 epilogue(parametersSize - markSize, storage); //запись эпилога процедуры  
 closeScope(storage); //закрытие scope  
 changeLevel(-1, storage); //-1 уровень вложенности  
 }  
  
}  
  
struct object\* parametersMatch(struct object \*formalParameter, struct parameters \*storage) {  
  
 //проверка параметров процедуры на корректность  
 struct item\* parameterExpression;  
 struct object\* nextFormalParameter;  
 nextFormalParameter = formalParameter;  
 parameterExpression = expression(storage); //получение выражения параметра  
 if(parameterFormat(nextFormalParameter)) { //если параметр корректен  
 parameterGen(parameterExpression, nextFormalParameter->classType, storage); //запись параметра в стек  
 nextFormalParameter = nextFormalParameter->nextObject; //переход к след. параметру  
 }  
 else  
 mark("Wrong parameters", storage);  
 if(parameterExpression != NULL)  
 free(parameterExpression);  
 return nextFormalParameter;  
  
}  
  
int parameterFormat(struct object \*object) {  
  
 //проверка, является ли объект допустимым параметром (проверка только формата)  
 //параметр - Par либо Var. значение val - положительное, так как положительное смещение  
 return ((object->class == ParGen) || ((object->class == VarGen) && (object->value > 0)));  
  
}  
  
struct item\* getSingleParameter(struct parameters \*storage) {  
  
 //получение выражения для глоб. процедур. ( ВЫРАЖЕНИЕ )  
 struct item\* currentParameter;  
 if(storage->lastLexemeCode == lparenLexical)  
 get(storage);  
 else  
 mark("(?", storage);  
 currentParameter = expression(storage);  
 if(storage->lastLexemeCode == rparenLexical)  
 get(storage);  
 else  
 mark(")?", storage);  
 return currentParameter;  
  
}  
  
void statements(struct parameters \*storage) {  
  
 //последовательность операторов. \*оператор\* {; \*оператор\*}  
 //оператор: присваивание / вызов процедуры / If / While  
 do {  
 struct object\* currentObject;  
 if(storage->lastLexemeCode < identLexical) {  
 mark("Statement?", storage);  
 do {  
 get(storage);  
 } while(storage->lastLexemeCode < identLexical);  
 } //поиск по коду, пока не будет корректных символов  
 if(storage->lastLexemeCode == identLexical) { //если слово - идентификатор  
 currentObject = findObject(storage); //поиск идент. среди объявленных  
 get(storage);  
 struct item\* currentItem;  
 currentItem = makeItem(currentObject, storage); //создание item для вычисления выражения  
 currentItem = selector(currentItem, storage); //получение значения по индексу/полю, если они есть  
 if(storage->lastLexemeCode == becomesLexical) { //x := y присваивание  
 get(storage);  
 struct item\* bufferItem;  
 bufferItem = expression(storage);  
 store(currentItem, bufferItem, storage); //запись y в x  
 if(bufferItem != NULL)  
 free(bufferItem);  
 }  
 else if(storage->lastLexemeCode == eqlLexical) {  
 mark("Lost : ?", storage);  
 get(storage);  
 expression(storage);  
 } //если =, предполагается, что пропущено :  
 else if(currentItem->mode == ProcGen) { //если объект - процедура, то это ее вызов  
 struct object\* parametersObject = currentObject->previousScope; //получение формальных параметров процедуры.   
 // parametersObject - указатель на первый  
 if(storage->lastLexemeCode == lparenLexical) { //procedure(params). чтение фактических параметров  
 get(storage);  
 if(storage->lastLexemeCode == rparenLexical)  
 get(storage); //если параметров нет  
 else { //чтение передаваемых параметров  
 do {  
 parametersObject = parametersMatch(parametersObject, storage); //проверка и запись передаваемых параметров  
 if(storage->lastLexemeCode == commaLexical)  
 get(storage); //неск. параметров - запятые  
 else if(storage->lastLexemeCode < semicolonLexical) {  
 if(storage->lastLexemeCode != rparenLexical)  
 mark(") or , ?", storage); //если встречен недопустимый символ и это не ) - ошибка  
 else  
 get(storage); //если ) - чтение след. символа - выход из цикла  
 }  
 } while((storage->lastLexemeCode != rparenLexical) && (storage->lastLexemeCode < semicolonLexical));  
 }  
 }  
 if(currentObject->value < 0)  
 mark("Forward procedure call", storage); //если вызов процедуры выше своего scope  
 else if(!parameterFormat(parametersObject)) //если все параметры корректны - запись вызова процедуры  
 procedureCall(currentItem, storage);  
 else  
 mark("Wrong parameters", storage); //иначе - ошибка  
 }  
 else if(currentItem->mode == SProcGen) { //если вызов одной из объявленных в universe процедур: Read, Write, WriteHex, WriteLn  
 struct item\* bufferItem;  
 if(currentObject->value <= 3) //для каждой процедуры свое значение: 1, 2, 3, 4. Для 1, 2, 3 есть аргументы,  
 bufferItem = getSingleParameter(storage); //для WriteLn - нет. здесь получение аргумента  
 globalCall(currentItem, bufferItem, storage); //запись вызова  
 if(bufferItem != NULL)  
 free(bufferItem);  
 }  
 else if(currentObject->class == TypGen)  
 mark("Illegal type assignment", storage); //если оператор начинается с типа  
 else  
 mark("Illegal statement", storage);  
 if(currentItem != NULL)  
 free(currentItem);  
 }  
 else if(storage->lastLexemeCode == ifLexical) { //если цикл if  
 get(storage);  
 struct item\* currentItem;  
 currentItem = expression(storage); //анализ выражения-условия ( exp )  
 falseJump(currentItem, storage); //переход через блок, если ложно  
 if(storage->lastLexemeCode == thenLexical)  
 get(storage);  
 else  
 mark("Lost THEN", storage); //IF \*exp\* THEN  
 statements(storage); //последовательность операторов  
 int jumpLevel = 0; //уровень ветвления переходов  
 while(storage->lastLexemeCode == elsifLexical) {  
 get(storage);  
 jumpLevel = elseJump(jumpLevel, storage); //запись адреса для списка переходов if  
 fixLink(currentItem->a, storage); //запись адреса для предыдущих операторов  
 currentItem = expression(storage); //получение очередного условия перехода  
 falseJump(currentItem, storage); //переход, если ложно  
 if(storage->lastLexemeCode == thenLexical)  
 get(storage);  
 else  
 mark("Lost THEN", storage); //ELSIF \*exp\* THEN  
 statements(storage); //последовательность операторов  
 }  
 if(storage->lastLexemeCode == elseLexical) { //последнее условие  
 get(storage);  
 jumpLevel = elseJump(jumpLevel, storage); //запись адреса для списка переходов if  
 fixLink(currentItem->a, storage); //запись адреса для предыдущих операторов  
 statements(storage); //последовательность операторов  
 }  
 else  
 fixLink(currentItem->a, storage); //запись адреса для предыдущих операторов  
 fixLink(jumpLevel, storage); //выравнивание адресов всего списка переходов  
 if(storage->lastLexemeCode == endLexical)  
 get(storage);  
 else  
 mark("Lost END", storage); //IF \*...\* END !  
 if(currentItem != NULL)  
 free(currentItem);  
 }  
 else if(storage->lastLexemeCode == whileLexical) { //WHILE \*cond\* DO \*statements\* END  
 get(storage);  
 int jumpAddress = storage->PC; //адрес возврата цикла  
 struct item\* currentItem;  
 currentItem = expression(storage); //получение условие  
 falseJump(currentItem, storage); //переход, если условие не выполняется  
 if(storage->lastLexemeCode == doLexical)  
 get(storage);  
 else  
 mark("Lost DO", storage);  
 statements(storage); //последовательность операторов  
 whileJump(jumpAddress, storage); //безусловный переход к началу цикла  
 fixLink(currentItem->a, storage); //выравнивание адреса перехода к концу цикла  
 if(storage->lastLexemeCode == endLexical)  
 get(storage);  
 else  
 mark("Lost END", storage);  
 if(currentItem != NULL)  
 free(currentItem);  
 }  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else if((storage->lastLexemeCode <= identLexical) || (storage->lastLexemeCode == ifLexical) || (storage->lastLexemeCode == whileLexical)) {  
 mark(";?", storage); //если идент, if или while, но между ними нет ;  
 }  
  
 } while((storage->lastLexemeCode <= identLexical) || (storage->lastLexemeCode == ifLexical) || (storage->lastLexemeCode == whileLexical));  
  
}  
  
void closeScope(struct parameters\* storage) {  
  
 //закрытие текущей области видимости  
 storage->topScope = storage->topScope->previousScope; //возврат к предыдущему scope  
  
}  
  
int module(struct parameters\* storage) {  
  
 //чтение модуля  
 signal("Compilation begins.", storage);  
 if(storage->lastLexemeCode == moduleLexical) { //модуль начинается с MODULE  
 char moduleName[identLength] = "\0"; //название модуля  
 int varSize = 0; //размер памяти для глобальных переменных  
 get(storage);  
 openScope(storage); //открытие нового scope  
 if(storage->lastLexemeCode == identLexical) { //MODULE \*ident\*  
 strcpy(moduleName, storage->lastLexeme);  
 //"Compiles module moduleName."  
 fprintf(storage->reportFile, "Compiles module %s.\r\n", moduleName);  
 printf("Compiles module %s.\n", moduleName);  
 get(storage);  
 }  
 else  
 mark("No module identifier", storage);  
 if(storage->lastLexemeCode == semicolonLexical) //MODULE \*ident\*;  
 get(storage);  
 else  
 mark("Lost ;", storage);  
 varSize = declarations(storage); //чтение объявлений CONST, TYPE, VAR. varSize - размер переменных  
 while(storage->lastLexemeCode == procedureLexical) { //чтение процедуры  
 procedureAnalyzer(storage);  
 if(storage->lastLexemeCode == semicolonLexical)  
 get(storage);  
 else  
 mark("Lost ;", storage); //PROCEDURE procbody; !  
 }  
 codeHeader(varSize, storage); //пролог. запись точки входа и указателя на вершину стека  
 if(storage->lastLexemeCode == beginLexical) { //начало модуля  
 get(storage);  
 statements(storage); //анализ последовательности операторов  
 }  
 if(storage->lastLexemeCode == endLexical)  
 get(storage);  
 else  
 mark("Lost END", storage); //конец модуля  
 if(storage->lastLexemeCode == identLexical) {  
 if(strcmp(moduleName, storage->lastLexeme))  
 mark("Wrong module identifier", storage);  
 get(storage);  
 }  
 else  
 mark("No module identifier", storage);  
 if(storage->lastLexemeCode != periodLexical)  
 mark("Lost .", storage); //MODULE \*ident\*.  
 closeScope(storage); //закрытие universe  
 if(!storage->error) { //если не было ошибок - генерация кода  
 codeEnding(storage); //эпилог. возврат вершины стека  
 signal("Code generated.", storage);  
 return 0;  
 }  
 }  
 else  
 mark("No MODULE", storage);  
 return 1; //возвращается 1, если ошибка  
  
}  
  
void freeStorage(struct parameters\* storage) {  
  
 freeObject(storage->universe, storage);  
 if(storage->universe != NULL)  
 free(storage->universe);  
 if(storage->topScope != NULL)  
 free(storage->topScope);  
 freeObject(storage->emptyObject, storage);  
 if(storage->emptyObject != NULL)  
 free(storage->emptyObject);  
 freeType(storage->emptyType, storage);  
 if(storage->emptyType != NULL)  
 free(storage->emptyType);  
 freeType(storage->intType, storage);  
 if(storage->intType != NULL)  
 free(storage->intType);  
 freeType(storage->boolType, storage);  
 if(storage->boolType != NULL)  
 free(storage->boolType);  
 freeObject(storage->guard, storage);  
 if(storage->guard != NULL)  
 free(storage->guard);  
 if(storage->emptyItem != NULL)  
 free(storage->emptyItem);  
 if(storage->sourceCode != NULL)  
 free(storage->sourceCode);  
 for(int i = 0; i < keyTabSize; i++) {  
 if(storage->keyTab[i] != NULL)  
 free(storage->keyTab[i]);  
 }  
  
}  
  
void compile(char \*sourceCode) {  
  
 struct parameters\* storage =  
 (struct parameters\*)malloc(sizeof(struct parameters));  
 if(storage == NULL) {  
 mark("Memory allocation error in function compile", storage);  
 return;  
 }  
 if(init(storage, sourceCode) != 0)  
 return; //важно, потому что могут быть ошибки обращения к памяти  
 get(storage); //получение первого символа  
 if (module(storage) == 0) //анализ кода, компиляция  
 decode(storage); //если компиляция прошла успешно - запись кода в output.txt  
 signal("Compilation finished.", storage);  
 if(storage != NULL) {  
 freeStorage(storage);  
 free(storage);  
 }  
  
}

* 1. scanner.h

#ifndef \_SCANNER\_H  
#define \_SCANNER\_H  
  
#include "parser.h"  
  
#define MAXINTLENGTH 10 //максимальная длина int  
#define MAXINT 2147483647 //максимальное значение int  
  
//терминальные символы  
#define nullLexical 0  
#define timesLexical 1  
#define divLexical 3  
#define modLexical 4  
#define andLexical 5  
#define plusLexical 6  
#define minusLexical 7  
#define orLexical 8  
#define eqlLexical 9  
#define neqLexical 10  
#define lssLexical 11  
#define geqLexical 12  
#define leqLexical 13  
#define gtrLexical 14  
#define periodLexical 18  
#define commaLexical 19  
#define colonLexical 20  
#define rparenLexical 22  
#define rbrakLexical 23  
#define ofLexical 25  
#define thenLexical 26  
#define doLexical 27  
#define lparenLexical 29  
#define lbrakLexical 30  
#define notLexical 32  
#define becomesLexical 33  
#define numberLexical 34  
#define identLexical 37  
#define semicolonLexical 38  
#define endLexical 40  
#define elseLexical 41  
#define elsifLexical 42  
#define ifLexical 44  
#define whileLexical 46  
#define arrayLexical 54  
#define recordLexical 55  
#define constLexical 57  
#define typeLexical 58  
#define varLexical 59  
#define procedureLexical 60  
#define beginLexical 61  
#define moduleLexical 63  
#define eofLexical 64  
  
void get(struct parameters\*);  
void mark(char[], struct parameters\*);  
  
#endif

* 1. scanner.c

#include "scanner.h"  
  
void mark(char msg[], struct parameters\* storage) {  
  
 //печать сообщения об ошибке в report.txt и в консоль  
 int p;  
 p = storage->lastPosition;  
 if (p > storage->errorPosition) {  
 fprintf(storage->reportFile, "error: line %d, position %d %s\r\n", storage->linesCounter, storage->posCounter, msg);  
 printf("error: line %d, position %d %s\n", storage->linesCounter, storage->posCounter, msg);  
 }  
 storage->errorPosition = p;  
 storage->error = 1; //обновление индексов ошибки  
  
}  
  
void comment(struct parameters\* storage) {  
  
 //пропуск комментария  
 int index = storage->lastPosition + 2; //(\* comment \*). здесь пропуск (\*  
 while(storage->sourceCode[index] != '\0') {  
 if (storage->sourceCode[index] == ')') {  
 if (storage->sourceCode[index - 1] == '\*')  
 break;  
 }  
 index++;  
 storage->posCounter++;  
 if(storage->sourceCode[index] == '\n') {  
 storage->linesCounter++;  
 storage->posCounter = 0;  
 }  
 }  
 if (storage->sourceCode[index] == '\0')  
 mark("Comment is not terminated", storage); //если достигнут до конца файла - комментарий не окончен  
 else  
 index++;  
 storage->lastPosition = index;  
  
}  
  
void number(struct parameters \*storage) {  
  
 //чтение числа  
 int index = storage->lastPosition; //первая позиция числа  
 int indexString = 0;  
 long long int getNumber = 0; //полученное число  
 char numberString[MAXINTLENGTH + 1]; //строка для записи числа  
 while((storage->sourceCode[index] != '\0') &&  
 isdigit(storage->sourceCode[index]) && (indexString < MAXINTLENGTH)) {  
 numberString[indexString] = storage->sourceCode[index];  
 numberString[indexString + 1] = '\0';  
 index++;  
 indexString++;  
 storage->posCounter++;  
 if(storage->sourceCode[index] == '\n') {  
 storage->linesCounter++;  
 storage->posCounter = 0;  
 }  
 } //чтение цифр, пока не конец строки и пока число не длиннее максимально допустимого  
 if (isdigit(storage->sourceCode[index])) { //если число длиннее допустимого  
 while ((storage->sourceCode[index] != '\0') &&  
 (isdigit(storage->sourceCode[index]))) {  
 index++; //до конца числа  
 storage->posCounter++;  
 if(storage->sourceCode[index] == '\n') {  
 storage->linesCounter++;  
 storage->posCounter = 0;  
 }  
 }  
 mark("Too big number", storage);  
 storage->lastLexemeCode = nullLexical;  
 }  
 else {  
 getNumber = atoll(numberString);  
 if (getNumber > MAXINT) { //если число превышает максимально допустимое  
 mark("Too big number", storage);  
 storage->lastLexemeCode = nullLexical;  
 }  
 else { //если ошибок нет - запись значения и кода  
 storage->lastLexemeCode = numberLexical;  
 storage->lastLexemeValue = getNumber;  
 }  
 }  
 storage->lastPosition = index;  
 if (isalpha(storage->sourceCode[index]))  
 mark("No space?", storage); //если остановлено на букве, к примеру, 7777IF  
  
}  
  
void identifier(struct parameters \*storage) {  
  
 //чтение идентификатора  
 int index = storage->lastPosition; //первая позиция идентификатора  
 int indexString = 0;  
 char identification[identLength + 1]; //строка для идентификатора  
 while((storage->sourceCode[index] != '\0') &&  
 (isalnum(storage->sourceCode[index])) && (indexString < identLength)) {  
 identification[indexString] = storage->sourceCode[index];  
 index++;  
 indexString++;  
 storage->posCounter++;  
 if(storage->sourceCode[index] == '\n') {  
 storage->linesCounter++;  
 storage->posCounter = 0;  
 }  
 }  
 identification[indexString] = '\0'; //считывание букв и цифр, пока не конец файла и до identLength  
  
 if (isalnum(storage->sourceCode[index])) { //если идентификатор длиннее identLength  
 while(isalnum(storage->sourceCode[index]))  
 index++; //идем до конца идентификатора  
 mark("Too large identification", storage);  
 storage->lastLexemeCode = nullLexical;  
 }  
 else {  
 int k = 0;  
 while((k < keyTabSize) && (strcmp(identification, storage->keyTab[k]->identifier)))  
 k++; //ищем идентификатор в таблице терминальных символов  
 if(k < keyTabSize)  
 storage->lastLexemeCode = storage->keyTab[k]->symbol; //если нашли - присваиваем его код  
 else {  
 storage->lastLexemeCode = identLexical;  
 strcpy(storage->lastLexeme, identification); //если нет - записываем идентификатор и код идентификатора  
 }  
 }  
 storage->lastPosition = index;  
  
}  
  
void get(struct parameters\* storage) {  
  
 //получение следующего символа  
 char readChar; //текущий символ  
 int index = storage->lastPosition; //текущий индекс  
 storage->lastLexemeCode = -1;  
 strcpy(storage->lastLexeme, "\0");  
 storage->lastLexemeValue = -1; //"сбрасываем" все данные о предыдущем символе  
 while((storage->sourceCode[index] != '\0') && (storage->sourceCode[index] <= ' ')) {  
 index++;  
 storage->posCounter++;  
 if(storage->sourceCode[index] == '\n') {  
 storage->linesCounter++;  
 storage->posCounter = 0;  
 }  
 } //пропуск символов табуляции и пробелов  
 readChar = storage->sourceCode[index];  
 storage->lastPosition = index;  
 if (readChar == '\0')  
 storage->lastLexemeCode = eofLexical; //если дошли до символа конца строки - конец файла  
 else {  
 switch (readChar) {  
 case '&':  
 storage->lastLexemeCode = andLexical;  
 storage->lastPosition++;  
 break;  
 case '\*':  
 storage->lastLexemeCode = timesLexical;  
 storage->lastPosition++;  
 break;  
 case '+':  
 storage->lastLexemeCode = plusLexical;  
 storage->lastPosition++;  
 break;  
 case '-':  
 storage->lastLexemeCode = minusLexical;  
 storage->lastPosition++;  
 break;  
 case '=':  
 storage->lastLexemeCode = eqlLexical;  
 storage->lastPosition++;  
 break;  
 case '#':  
 storage->lastLexemeCode = neqLexical;  
 storage->lastPosition++;  
 break;  
 case '<':  
 if (storage->sourceCode[index + 1] == '=') {  
 storage->lastLexemeCode = leqLexical;  
 storage->lastPosition += 2; //либо <, либо <=  
 }  
 else {  
 storage->lastLexemeCode = lssLexical;  
 storage->lastPosition++;  
 }  
 break;  
 case '>':  
 if (storage->sourceCode[index + 1] == '=') {  
 storage->lastLexemeCode = geqLexical;  
 storage->lastPosition += 2; //либо >, либо >=  
 }  
 else {  
 storage->lastLexemeCode = gtrLexical;  
 storage->lastPosition++;  
 }  
 break;  
 case ';':  
 storage->lastLexemeCode = semicolonLexical;  
 storage->lastPosition++;  
 break;  
 case ',':  
 storage->lastLexemeCode = commaLexical;  
 storage->lastPosition++;  
 break;  
 case ':':  
 if (storage->sourceCode[index + 1] == '=') {  
 storage->lastLexemeCode = becomesLexical;  
 storage->lastPosition += 2; //либо :, либо :=  
 }  
 else {  
 storage->lastLexemeCode = colonLexical;  
 storage->lastPosition++;  
 }  
 break;  
 case '.':  
 storage->lastLexemeCode = periodLexical;  
 storage->lastPosition++;  
 break;  
 case '(':  
 if (storage->sourceCode[index + 1] == '\*') {  
 comment(storage);  
 get(storage); //либо комментарий (\* ... \*), либо   
 }  
 else {  
 storage->lastLexemeCode = lparenLexical;  
 storage->lastPosition++;  
 }  
 break;  
 case ')':  
 storage->lastLexemeCode = rparenLexical;  
 storage->lastPosition++;  
 break;  
 case '[':  
 storage->lastLexemeCode = lbrakLexical;  
 storage->lastPosition++;  
 break;  
 case ']':  
 storage->lastLexemeCode = rbrakLexical;  
 storage->lastPosition++;  
 break;  
 case '~':  
 storage->lastLexemeCode = notLexical;  
 storage->lastPosition++;  
 break;  
 default:  
 if (isdigit(readChar)) {  
 number(storage);  
 }  
 else if (isalpha(readChar)) {  
 identifier(storage);  
 }  
 else {  
 storage->lastLexemeCode = nullLexical;  
 storage->lastPosition++;  
 mark("Unknown symbol", storage);  
 }  
 break;  
 }  
 }  
  
}

* 1. codeGenerator.h

#ifndef \_CODEGEN\_H  
#define \_CODEGEN\_H  
  
//классы, режимы  
#define HeadGen 0  
#define VarGen 1  
#define ParGen 2  
#define ConstGen 3  
#define FldGen 4  
#define TypGen 5  
#define ProcGen 6  
#define SProcGen 7  
#define RegGen 10  
#define CondGen 11  
  
//идентификаторы типов  
#define BooleanGen 0  
#define IntegerGen 1  
#define ArrayGen 2  
#define RecordGen 3  
  
//ассемблерные команды  
#define MOVGen 0  
#define MVNGen 1  
#define ADDGen 2  
#define SUBGen 3  
#define MULGen 4  
#define DIVGen 5  
#define MODGen 6  
#define CMPGen 7  
#define MOVIGen 16  
#define MVNIGen 17  
#define ADDIGen 18  
#define SUBIGen 19  
#define MULIGen 20  
#define DIVIGen 21  
#define MODIGen 22  
#define CMPIGen 23  
#define CHKIGen 24  
#define LDWGen 32  
#define LDBGen 33  
#define POPGen 34  
#define STWGen 36  
#define STBGen 37  
#define PSHGen 38  
#define RDGen 40  
#define WRDGen 41  
#define WRHGen 42  
#define WRLGen 43  
#define BEQGen 48  
#define BNEGen 49  
#define BLTGen 50  
#define BGEGen 51  
#define BLEGen 52  
#define BGTGen 53  
#define BRGen 56  
#define BSRGen 57  
#define RETGen 58  
  
//зарезервированные регистры  
#define FPGen 12  
#define SPGen 13  
#define LNKGen 14  
#define PCGen 15  
  
void arrayElem(struct item \*, struct item \*, struct parameters \*);  
struct object\* findField(struct object \*, struct parameters \*);  
void getField(struct item \*, struct object \*);  
struct item\* makeItem(struct object \*, struct parameters \*);  
struct item\* makeConstItem(struct type \*, int, struct parameters \*);  
void singleGenerate(int, struct item \*, struct parameters \*);  
void termGenerate(int, struct item \*, struct item \*, struct parameters \*);  
void relation(int, struct item \*, struct item \*, struct parameters \*);  
void changeLevel(int, struct parameters \*);  
void prologue(int, struct parameters \*);  
void epilogue(int, struct parameters \*);  
void fixLink(int, struct parameters \*);  
void store(struct item \*, struct item \*, struct parameters \*);  
void procedureCall(struct item \*, struct parameters \*);  
void globalCall(struct item \*, struct item \*, struct parameters \*);  
void falseJump(struct item \*, struct parameters \*);  
void whileJump(int, struct parameters \*);  
int elseJump(int, struct parameters \*);  
void codeHeader(int, struct parameters \*);  
void codeEnding(struct parameters \*);  
void decode(struct parameters\*);  
void parameterGen(struct item \*, struct type \*, struct parameters \*);  
void openScope(struct parameters\*);  
void closeScope(struct parameters\*);  
void statements(struct parameters\*);  
struct item\* expression(struct parameters\*);  
struct object\* identifiersList(int, struct parameters\*);  
int parameterFormat(struct object\*);  
  
#endif

* 1. codeGenerator.c

#include <malloc.h>  
#include <string.h>  
#include <stdio.h>  
#include "scanner.h"  
#include "codeGenerator.h"  
  
void testRange(int x, struct parameters \*storage) {  
  
 //проверка на переполнение  
 if((x >= 0x20000) || (x < -0x20000)) //131072  
 mark("Value is too large", storage);  
  
}  
  
void put(int command, int a, int b, int c, struct parameters \*storage) {  
  
 //запись сгенерированной команды  
 //первые 5 бит - команда, след. 4 бита - a, след. 4 бита - b, след. 18 - c  
 if(command >= 32)  
 command -= 64; //если команда больше 5 бит. в декодере учитывается знак  
 storage->code[storage->PC] = (((command << 4 | a) << 4 | b) << 18) | (c & 0x3FFFF);  
 storage->PC++;  
  
}  
  
int getReg(struct parameters \*storage) {  
  
 //получение свободного регистра  
 int newReg = 0;  
 while((newReg < FPGen) && (storage->registers[newReg] == 1))  
 newReg++;  
 storage->registers[newReg] = 1;  
 return newReg;  
  
}  
  
void load(struct item\* x, struct parameters\* storage) {  
  
 //загрузка в регистр  
 if(x->mode == VarGen) { //загрузка переменной  
 int r = getReg(storage); //получение регистра  
 if(x->level == 0)  
 x->a -= storage->PC \* 4; //если это глобальные переменные, то увеличение сдвига относительно базового адреса  
 put(LDWGen, r, x->storage, x->a, storage); //загрузка в регистр значения переменной  
 storage->registers[x->storage] = 0;  
 x->storage = r; //обнуление регистра, в кот. хранился адрес переменной  
 }  
 else if(x->mode == ConstGen) { //загрузка константы  
 testRange(x->a, storage); //проверка на переполнение  
 x->storage = getReg(storage); //получение регистра  
 put(MOVIGen, x->storage, 0, x->a, storage); //загрузка константы в регистр  
 }  
 x->mode = RegGen; //mode - загружен в регистр  
  
}  
  
void arrayElem(struct item \*arrayVar, struct item \*index, struct parameters \*storage) {  
  
 //получение элемента массива  
 if(index->classType != storage->intType)  
 mark("Index is not an integer", storage); //индекс - только int  
 if(index->mode == ConstGen) { //если индекс - константа  
 if((index->a < 0) || (index->a >= arrayVar->classType->length))  
 mark("Bad index", storage);  
 arrayVar->a += index->a \* arrayVar->classType->baseType->size; //смещение относительно адреса массива  
 }  
 else { //если индекс - переменная  
 if(index->mode != RegGen)  
 load(index, storage); //если индекс не хранится в регистре - загрузка  
 put(CHKIGen, index->storage, 0, arrayVar->classType->length, storage); //обнуление индекса, если он больше допустимого  
 put(MULIGen, index->storage, index->storage, arrayVar->classType->baseType->size, storage); //получение смещения  
 put(ADDGen, index->storage, arrayVar->storage, index->storage, storage); //добавление адреса к смещению  
 storage->registers[arrayVar->storage] = 0;  
 arrayVar->storage = index->storage; //обнуление регистра элемента  
 }  
 arrayVar->classType = arrayVar->classType->baseType; //типа элемента - базовый тип массива  
  
}  
  
struct object\* findField(struct object \*list, struct parameters \*storage) {  
  
 //поиск поля записи в scope записи  
 strcpy(storage->guard->name, storage->lastLexeme);  
 while(strcmp(list->name, storage->lastLexeme))  
 list = list->nextObject;  
 return list; //возврат либо поля, либо guard  
  
}  
  
void getField(struct item \*record, struct object \*field) {  
  
 //инициализация поля записи  
 record->a += field->value; //добавление смещения поля  
 record->classType = field->classType; //присваивание типа поля  
  
}  
  
struct item\* makeItem(struct object \*argObject, struct parameters \*storage) {  
  
 //создание item из object  
 struct item\* newItem = (struct item\*)malloc(sizeof(struct item));  
 if(newItem == NULL) {  
 mark("Memory allocation error in function makeItem", storage);  
 return storage->emptyItem;  
 }  
 newItem->mode = argObject->class;  
 newItem->classType = argObject->classType;  
 newItem->level = argObject->level;  
 newItem->a = argObject->value;  
 newItem->b = 0;  
 if(argObject->level == 0)  
 newItem->storage = PCGen; //если глобальная переменная  
 else if(argObject->level == storage->currentLevel)  
 newItem->storage = FPGen; //если локальная - указывает на базовый адрес процедуры  
 else  
 newItem->storage = 0;  
 if(argObject->class == ParGen) { //если параметр процедуры  
 int reg = getReg(storage); //получение регистра для его хранения  
 put(LDWGen, reg, newItem->storage, newItem->a, storage); //загрузка в регистр параметра: баз. адрес + смещение  
 newItem->mode = VarGen;  
 newItem->storage = reg;  
 newItem->a = 0; //значение хранится в регистре. смещение - 0  
 }  
 return newItem;  
  
}  
  
struct item\* makeConstItem(struct type \*typ, int val, struct parameters \*storage) {  
  
 //создание item-константы без загрузки в регистр  
 struct item\* newItem = (struct item\*)malloc(sizeof(struct item));  
 if(newItem == NULL) {  
 mark("Memory allocation error in function makeConstItem", storage);  
 return storage->emptyItem;  
 }  
 initItem(newItem, storage);  
 newItem->mode = ConstGen;  
 newItem->classType = typ;  
 newItem->a = val;  
 return newItem;  
  
}  
  
int negated(int cond) {  
  
 //для команд переходов. если число четное - возврат нечетного и наоборот  
 if(cond % 2)  
 return cond - 1;  
 else  
 return cond + 1;  
  
}  
  
void loadBool(struct item\* x, struct parameters\* storage) {  
  
 //load с проверкой на тип boolean  
 //атрибуты cond: = 0; != 1; < 2; >= 3; <= 4; > 5  
 if(x->classType->classType != BooleanGen)  
 mark("Not a boolean type", storage);  
 load(x, storage);  
 x->mode = CondGen; //"режим" - логическое выражение  
 x->a = 0; //адрес незавершенной команды перехода  
 x->b = 0;  
 x->c = 1; //по умолчанию для выражения атрубут !=  
  
}  
  
void putJump(int command, int disp, struct parameters \*storage) {  
  
 //запись команд перехода/сравнения  
 //первые 5 бит - команда, след. 26 - смещение  
 storage->code[storage->PC] = ((command - 0x40) << 26) | (disp & 0x3FFFFFF);  
 storage->PC++;  
  
  
}  
  
void fix(int commandIndex, int jumpAddress, struct parameters\* storage) {  
  
 //запись адреса перехода в команду  
 storage->code[commandIndex] = (storage->code[commandIndex] & 0xFFC00000) | (jumpAddress & 0x3FFFFF);  
  
}  
  
void fixLink(int jumpEntry, struct parameters \*storage) {  
  
 //запись адреса перехода для всех ответвлений  
 int nextJump;  
 while(jumpEntry != 0) {  
 nextJump = storage->code[jumpEntry] & 0x3FFFF; //следующее ответвление  
 fix(jumpEntry, storage->PC - jumpEntry, storage);  
 jumpEntry = nextJump;  
 }  
  
}  
  
void singleGenerate(int op, struct item \*x, struct parameters \*storage) {  
  
 //генерация кода для выражений с одним аргументом. -x, ~x, x &, x |  
 if(op == minusLexical) { //если -x  
 if(x->classType->classType != IntegerGen)  
 mark("Type must be integer", storage); //отрицательный только int  
 else if(x->mode == ConstGen)  
 x->a = -(x->a); //если константа - просто отрицательное значение  
 else {  
 if(x->mode == VarGen)  
 load(x, storage); //если не загружена переменная - загрузка в регистр  
 put(MVNGen, x->storage, 0, x->storage, storage); //загрузка с реверс. знаком  
 }  
 }  
 else if(op == notLexical) { //если ~x  
 if(x->mode != CondGen)  
 loadBool(x, storage); //если не загружено в регист - загрузка  
 x->c = negated(x->c); //= меняется на != и наоборот  
 int t;  
 t = x->a;  
 x->a = x->b;  
 x->b = t; //перестваление списков F- и T- переходов  
 }  
 else if(op == andLexical) { //x &  
 if(x->mode != CondGen)  
 loadBool(x, storage); //загрузка в регистр  
 putJump(BEQGen + negated(x->c), x->a, storage); //переход дальше, если аргумент 1  
 storage->registers[x->storage] = 0;  
 x->a = storage->PC - 1;  
 fixLink(x->b, storage); //запись адреса перехода  
 x->b = 0;  
 }  
 else if(op == orLexical) { //x |  
 if(x->mode != CondGen)  
 loadBool(x, storage); //загрузка в регистр  
 putJump(BEQGen + x->c, x->b, storage); //переход дальше, если аргумент 0  
 storage->registers[x->storage] = 0;  
 x->b = storage->PC - 1;  
 fixLink(x->a, storage); //запись адреса перехода  
 x->a = 0;  
 }  
  
}  
  
void putArith(int cd, struct item \*x, struct item \*y, struct parameters \*storage) {  
  
 if(x->mode != RegGen)  
 load(x, storage); //x - всегда в регистре  
 if(y->mode == ConstGen) {  
 testRange(y->a, storage);  
 put(cd + 16, x->storage, x->storage, y->a, storage); //если y - константа, команда с I (E: MOVI)  
 }  
 else {  
 if(y->mode != RegGen)  
 load(y, storage); //иначе, y - в регистре  
 put(cd, x->storage, x->storage, y->storage, storage);  
 storage->registers[y->storage] = 0; //y освобождается  
 }  
  
}  
  
int merged(int rightExp, int leftExp, struct parameters\* storage) {  
  
 //соединение логических подвыражений  
 int rightBuff, secondBuff;  
 if(rightExp != 0) { //если правое выражение есть, анализируем его  
 rightBuff = rightExp;  
 do {  
 secondBuff = storage->code[rightBuff] & 0x3FFFF;  
 if(secondBuff != 0)  
 rightBuff = secondBuff;  
 } while(secondBuff != 0);  
 storage->code[rightBuff] = storage->code[rightBuff] + leftExp;  
 return rightExp;  
 }  
 else  
 return leftExp;  
  
}  
  
void termGenerate(int op, struct item \*x, struct item \*y, struct parameters \*storage) {  
  
 //генерирование кода для выражения x op y;  
 //должны быть одинаковые типы!  
 if((x->classType->classType == IntegerGen) && (y->classType->classType == IntegerGen)) {  
 if((x->mode == ConstGen) && (y->mode == ConstGen)) { //две константы. в регистрах не хранятся  
 if(op == plusLexical)  
 x->a += y->a;  
 else if(op == minusLexical)  
 x->a -= y->a;  
 else if(op == timesLexical)  
 x->a \*= y->a;  
 else if(op == divLexical)  
 x->a /= y->a;  
 else if(op == modLexical)  
 x->a = x->a % y->a;  
 else  
 mark("Bad expression operator", storage);  
 }  
 else { //если не константы - хранится в регистарах  
 if(op == plusLexical)  
 putArith(ADDGen, x, y, storage);  
 else if(op == minusLexical)  
 putArith(SUBGen, x, y, storage);  
 else if(op == timesLexical)  
 putArith(MULGen, x, y, storage);  
 else if(op == divLexical)  
 putArith(DIVGen, x, y, storage);  
 else if(op == modLexical)  
 putArith(MODGen, x, y, storage);  
 else  
 mark("Bad expression operator", storage);  
 }  
 }  
 else if((x->classType->classType == BooleanGen) && (y->classType->classType == BooleanGen)) {  
 if(y->mode != CondGen)  
 loadBool(y, storage); //второй аргумент - в регистр  
 if(op == orLexical) {  
 x->a = y->a;  
 x->b = merged(y->b, x->b, storage); //соединение логических подвыражений  
 x->c = y->c;  
 }  
 else if(op == andLexical) {  
 x->a = merged(y->a, x->a, storage); //соединение логических подвыражений  
 x->b = y->b;  
 x->c = y->c;  
 }  
 }  
 else  
 mark("Bad types", storage);  
  
}  
  
void relation(int op, struct item \*x, struct item \*y, struct parameters \*storage) {  
  
 //генерирование кода для выражения сравнения  
 if((x->classType->classType != IntegerGen) || (y->classType->classType != IntegerGen))  
 mark("Must be integer type", storage); //аргументы - только int  
 else {  
 putArith(CMPGen, x, y, storage); //команда сравнения  
 x->c = op - eqlLexical; //код сравнения (есть в loadBool)  
 storage->registers[y->storage] = 0;  
 }  
 x->mode = CondGen;  
 x->classType = storage->boolType; //результат сравнения - boolean  
 x->a = 0;  
 x->b = 0;  
  
}  
  
void changeLevel(int n, struct parameters \*storage) {  
  
 //изменение уровня вложенности scope  
 storage->currentLevel += n;  
  
}  
  
void prologue(int size, struct parameters \*storage) {  
  
 //команды пролога. сохранение адреса возврата  
 put(PSHGen, LNKGen, SPGen, 4, storage);  
 put(PSHGen, FPGen, SPGen, 4, storage);  
 put(MOVGen, FPGen, 0, SPGen, storage);  
 put(SUBIGen, SPGen, SPGen, size, storage);  
  
}  
  
void epilogue(int size, struct parameters \*storage) {  
  
 //команды эпилога. возврат стека  
 put(MOVGen, SPGen, 0, FPGen, storage);  
 put(POPGen, FPGen, SPGen, 4, storage);  
 put(POPGen, LNKGen, SPGen, size + 4, storage);  
 putJump(RETGen, LNKGen, storage);  
  
}  
  
void store(struct item \*x, struct item \*y, struct parameters \*storage) {  
  
 //присваивание  
 if((x->classType->classType == BooleanGen || x->classType->classType == IntegerGen) && (x->classType->classType == y->classType->classType)) {  
 if(y->mode == CondGen) { //присваивание результата сравнения. x := y  
 put(BEQGen + negated(y->c), y->storage, 0, y->a, storage); //переход по лог. операции  
 storage->registers[y->storage] = 0;  
 y->a = storage->PC - 1; //переход, если истинно  
 fixLink(y->b, storage); //запись адреса перехода, если не истинно  
 y->storage = getReg(storage);  
 put(MOVIGen, y->storage, 0, 1, storage); //загрузка истинного значения  
 putJump(BRGen, 2, storage); //переход  
 fixLink(y->a, storage);  
 put(MOVIGen, y->storage, 0, 0, storage); //загрузка ложного значения  
 }  
 else if(y->mode != RegGen)  
 load(y, storage); //если y не в регистре - в регистр  
 if(x->mode == VarGen) {  
 if(x->level == 0)  
 x->a = x->a - (storage->PC) \* 4;  
 put(STWGen, y->storage, x->storage, x->a, storage); //если x переменная - сохранение в памяти  
 }  
 else  
 mark("Illegal assignment", storage);  
 storage->registers[x->storage] = 0;  
 storage->registers[y->storage] = 0; //освобождение регистров, так как переменная уже в памяти  
 }  
 else  
 mark("Incompatible assignment", storage);  
  
}  
  
void procedureCall(struct item \*x, struct parameters \*storage) {  
  
 //вызов процедуры  
 putJump(BSRGen, x->a - storage->PC, storage);  
  
}  
  
void globalCall(struct item \*x, struct item \*y, struct parameters \*storage) {  
  
 //вызов глобальных процедур  
 struct item\* z = (struct item\*)malloc(sizeof(struct item));  
 if(z == NULL) {  
 mark("Memory allocation error in function globalCall", storage);  
 return;  
 }  
 initItem(z, storage);  
 if(x->a < 4) {  
 if(y->classType->classType != IntegerGen)  
 mark("Argument must be integer", storage);  
 } //read, write, writehex - аргумент integer  
 if(x->a == 1) { //read  
 z->storage = getReg(storage);  
 z->mode = RegGen;  
 z->classType = storage->intType;  
 put(RDGen, z->storage, 0, 0, storage);  
 store(y, z, storage); //в регистр значение аргумента  
 }  
 else if(x->a == 2) { //write  
 load(y, storage);  
 put(WRDGen, 0, 0, y->storage, storage);  
 storage->registers[y->storage] = 0; //печать числа  
 }  
 else if(x->a == 3) { //writehex  
 load(y, storage);  
 put(WRHGen, 0, 0, y->storage, storage);  
 storage->registers[y->storage] = 0; //печать числа в формате 0x  
 }  
 else //writeln  
 put(WRLGen, 0, 0, 0, storage); //печать \n  
 if(z != NULL)  
 free(z);  
  
}  
  
void falseJump(struct item \*x, struct parameters \*storage) {  
  
 //переход, если условие не выполняется  
 if(x->classType->classType == BooleanGen) {  
 if(x->mode != CondGen)  
 loadBool(x, storage); //загрузка в регистр  
 putJump(BEQGen + negated(x->c), x->a, storage); //переход  
 storage->registers[x->storage] = 0;  
 fixLink(x->b, storage); //запись адреса перехода  
 x->a = storage->PC - 1;  
 }  
 else {  
 mark("Not boolean type", storage);  
 x->a = storage->PC;  
 }  
  
}  
  
void whileJump(int retAddress, struct parameters \*storage) {  
  
 //безусловный переход в конце цикла while  
 putJump(BRGen, retAddress - storage->PC, storage);  
  
}  
  
int elseJump(int retAddress, struct parameters \*storage) {  
  
 //переход для elsif/else  
 int newAddress;  
 putJump(BRGen, retAddress, storage);  
 newAddress = storage->PC - 1;  
 return newAddress;  
  
}  
  
void codeHeader(int size, struct parameters \*storage) {  
  
 //запись пролога программы  
 storage->entryAddress = storage->PC; //сохранение точки входа  
 put(MOVIGen, SPGen, 0, 1024 - size, storage); //вершина стека = память машины - память глоб. переменных  
 put(PSHGen, LNKGen, SPGen, 4, storage); //сохранение адреса вершины стека  
  
}  
  
void codeEnding(struct parameters \*storage) {  
  
 //конец кода.  
 put(POPGen, LNKGen, SPGen, 4, storage);  
 putJump(RETGen, LNKGen, storage);  
  
}  
  
void decode(struct parameters\* storage) {  
  
 //декордер, запись кода в файл  
 FILE\* outputFile = NULL;  
 printf("Recording compiled code...\n");  
 if((outputFile = fopen("output.txt", "wb")) == NULL) {  
 printf("Opening output file error!\n");  
 fprintf(storage->reportFile, "Opening output file error!\r\n");  
 return;  
 } //открытие output.txt  
 char mnemo[64][6];  
 strcpy(mnemo[MOVGen], "MOV ");  
 strcpy(mnemo[MVNGen], "MVN ");  
 strcpy(mnemo[ADDGen], "ADD ");  
 strcpy(mnemo[SUBGen], "SUB ");  
 strcpy(mnemo[MULGen], "MUL ");  
 strcpy(mnemo[DIVGen], "DIV ");  
 strcpy(mnemo[MODGen], "MOD ");  
 strcpy(mnemo[CMPGen], "CMP ");  
 strcpy(mnemo[MOVIGen], "MOVI ");  
 strcpy(mnemo[MVNIGen], "MVNI ");  
 strcpy(mnemo[ADDIGen], "ADDI ");  
 strcpy(mnemo[SUBIGen], "SUBI ");  
 strcpy(mnemo[MULIGen], "MULI ");  
 strcpy(mnemo[DIVIGen], "DIVI ");  
 strcpy(mnemo[MODIGen], "MODI ");  
 strcpy(mnemo[CMPIGen], "CMPI ");  
 strcpy(mnemo[CHKIGen], "CHKI ");  
 strcpy(mnemo[LDWGen], "LDW ");  
 strcpy(mnemo[LDBGen], "LDB ");  
 strcpy(mnemo[POPGen], "POP ");  
 strcpy(mnemo[STWGen], "STW ");  
 strcpy(mnemo[STBGen], "STB ");  
 strcpy(mnemo[PSHGen], "PSH ");  
 strcpy(mnemo[BEQGen], "BEQ ");  
 strcpy(mnemo[BNEGen], "BNE ");  
 strcpy(mnemo[BLTGen], "BLT ");  
 strcpy(mnemo[BGEGen], "BGE ");  
 strcpy(mnemo[BLEGen], "BLE ");  
 strcpy(mnemo[BGTGen], "BGT ");  
 strcpy(mnemo[BRGen], "BR ");  
 strcpy(mnemo[BSRGen], "BSR ");  
 strcpy(mnemo[RETGen], "RET ");  
 strcpy(mnemo[RDGen], "READ ");  
 strcpy(mnemo[WRDGen], "WRD ");  
 strcpy(mnemo[WRHGen], "WRH ");  
 strcpy(mnemo[WRLGen], "WRL "); //заполнение массива кодов команд  
 fprintf(outputFile, "entry %d\r\n", storage->entryAddress \* 4);  
 for(int i = 0; i < storage->PC; i++) {  
 int codeCommand = storage->code[i];  
 int command = (codeCommand >> 26) & 0x3F; //получение первых 5 бит - команда  
 fprintf(outputFile, "%d %s", 4 \* i, mnemo[command]);  
 if(command < MOVIGen) {  
 //c - регистр  
 fprintf(outputFile, "%d, %d, %d\r\n", (codeCommand >> 22) & 0x0F, (codeCommand >> 18) & 0x0F,  
 codeCommand & 0x0F); //a - след. 4 бита, b - след. 4 бита, c - след. 18 бит  
 }  
 else if(command < BEQGen) {  
 //c - константа либо смещение  
 int c = codeCommand & 0x3FFFF;  
 if(c >= 0x20000)  
 c -= 0x40000; //если смещение  
 fprintf(outputFile, "%d, %d, %d\r\n", (codeCommand >> 22) & 0x0F, (codeCommand >> 18) & 0x0F, c);  
 }  
 else {  
 int c = codeCommand & 0x3FFFFFF;  
 if(command == RETGen)  
 fprintf(outputFile, "%d\r\n", c); //c - регистр  
 else {  
 //c - смещение  
 if(c >= 0x2000000)  
 c -= 0x4000000;  
 fprintf(outputFile, "%d\r\n", c \* 4);  
 }  
 }  
 }  
 fprintf(outputFile, "%d bytes\r\n", storage->PC \* 4);  
 fclose(outputFile);  
 printf("Compiled code is in the \"output.txt\" file in the current directory. Bye!\n");  
  
}  
  
void parameterGen(struct item \*x, struct type \*ftyp, struct parameters \*storage) {  
  
 //запись фактических передаваемых параметров функции в стек  
 if(x->classType == ftyp) { //если параметр верный  
 if(x->mode != RegGen)  
 load(x, storage); //загрузка в регистр  
 put(PSHGen, x->storage, SPGen, 4, storage); //загрузка в стек  
 storage->registers[x->storage] = 0; //обнуление регистра  
 }  
 else  
 mark("Bad parameter type", storage);  
  
}