Министерство образования и науки Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и кибербезопасности
———————————————————
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

по дисциплине «Теория автоматов и формальных языков»

Тема: Расширение языка Milan циклом с пользовательским определением количества итераций

Выполнила:
Студент гр.: 5130202/20001 Флусова С.А.

Преподаватель: Герасимов А.С.

Санкт-Петербург, 2024

Содержание

1 Формулировка задания

Требуется внести изменения в компилятор согласно заданному варианту:

Вариант 16: Расширить язык Milan оператором цикла `loop ⟨statementList⟩ endloop`, который в начале своего исполнения сообщает пользователю номер своей первой строки в Milan-программе и запрашивает у пользователя, сколько раз исполнять своё тело (т. е. указанный `⟨statementList⟩`); а затем исполняет своё тело введённое пользователем число раз (детали поведения такого цикла, в частности при ошибочном вводе пользователя, специфицирует студент).

2 Изменения в грамматике

В процессе реализации оператора цикла **loop** в грамматику языка были внесены следующие изменения:

⟨statement⟩::=⟨ident⟩‘:=’⟨expression⟩
|‘if’ ⟨relation⟩ ‘then’ ⟨statementList⟩ [‘else’ ⟨statementList⟩] ‘fi’
|‘while’ ⟨relation⟩ ‘do’ ⟨statementList⟩ ‘od’
|‘write’ ‘(’ ⟨expression⟩ ‘)’
| **‘loop’ ⟨statementList⟩ ‘endloop’**

3 Семантика loop

- Введённая конструкция: `loop` *S* `endloop`.
  S - список операторов цикла `loop`.
- Описание:
  (1) Печать номера строки, соответствующей началу оператора цикла `loop`.
  (2) Считывание количества *N* итераций тела цикла.
  (3) Если *N* не является целым числом, программа завершается с ошибкой.
  (4) Иначе, если *N*<0, переход к шагу 2.
  (5) *S* выполняется *N* раз.
- Описание посредством эквивалентного этой конструкции и не содержащего её Milan-кода:

```
write(<номер_первой_строки>);
N := read;
while N < 0 do
    N := read
od;
while N != 0 do
    S;
    N := N - 1
od
```

Замечание: `<номер_первой_строки>` - обозначение для номера первой строки оператора цикла loop в Milan-программе.

4 Программная реализация

4.1 Схема VM-кода

<номер> - номер первой строки оператора цикла loop в Milan-программе.

```
                поместить <номер> на вершину стека и напечатать его

ИНИЦИАЛИЗАЦИЯ: считать <количество_итераций>
                инициализировать новую переменную iterationsAmount:=<количество_итераций>
                вычислить значение условия «iterationsAmount>=0»
                при ложном условии «iterationsAmount>=0» перейти (JUMP_NO)
                     к ИНИЦИАЛИЗАЦИЯ

НАЧАЛО:         вычислить значение условия «iterationsAmount!=0»
                при верном условии «iterationsAmount=0» перейти (JUMP_YES)
                     к КОНЕЦ
                код для S
                iterationsAmount--
                перейти (JUMP) к НАЧАЛО

КОНЕЦ:          первая команда после этого цикла
```

Замечание: для каждой конструкции loop (их может быть несколько в Milan-программе) инициализируется уникальная переменная iterationsAmount.

4.2 Алгоритм кодогенерации

Под обозначением `<текущий_адрес>` подразумевается адрес очередной ячейки памяти в буфере VM-команд.

1. Прочитать лексему `loop`.
2. Поместить номер первой строки оператора цикла `loop`, на вершину стека.
3. Напечатать в консоль номер первой строки, на которой находится оператор цикла loop.
4. Сгенерировать VM-код для записи в переменную `inputStr` значения `<текущий_адрес>` (в переменной `inputStr` будет храниться адрес инструкции ввода).
5. Считать с клавиатуры количество итераций тела цикла и сохранить полученное значение в переменную `iterationsAmount`.
6. Сгенерировать VM-код для вычисления значения условия «`iretationsAmount`$\geq$`0`».
7. Сгенерировать VM-команду «`JUMP_NO inputStr`».
8. Сгенерировать VM-код для записи в переменную `loopBeginningAddress` значения `<текущий_адрес>` (в переменной `loopBeginningAddress` хранится адрес начала тела цикла).
9. Сгенерировать VM-код для вычисления значения условия выхода из цикла «`iterationsAmount=0`».
10. Зарезервировать в буфере VM-команд ячейку памяти (для VM-команды `JUMP_YES`, которая осуществляет переход к коду после цикла) и записать полученный адрес в переменную `jumpExitAddress`.
11. Сгенерировать VM-команды для списка операторов S, вызвав метод `Parser::statementList()`.
12. Сгенерировать VM-код для записи в переменную `iterationsAmount` значения `iterationsAmount-1`.
13. Прочитать лексему `endloop`.
14. Сгенерировать VM-команду «`JUMP loopBeginningAddress`».
15. В буфере VM-команд по адресу `jumpExitAddress` записать VM-команду «`JUMP_YES <текущий_адрес>`».

4.3 Изменения в коде компилятора

В процессе реализации поставленной задачи в исходный код компилятора были внесены следующие изменения:

(a) Перечисление *Token* было расширено **следующими** ключевыми словами:

```
enum Token {

    ...

    T_LPAREN,
    T_RPAREN,
    T_SEMICOLON,
    T_LOOP,
    T_ENDLOOP,
}
```

(b) В массив указателей на символы *tokenNames_* были добавлены те же ключевые слова для вывода ошибок:

```
static const char *tokenNames_[] = {

    ...

    "'('",
    "')'",
    "':'",
    "'LOOP'",
    "'ENDLOOP'",
}
```

(c) К инициализации ассоциативного массива *keywords_* в конструкторе лексического анализатора *Scanner* был добавлен перевод цепочки символов для ключевых слов **loop** и **endloop**:

```
explicit Scanner(const string& fileName, istream&input)
 : fileName_(fileName), lineNumber_(1), input_(input)
{
    ...

    keywords_["od"] = T_OD;
    keywords_["write"] = T_WRITE;
    keywords_["read"] = T_READ;

    keywords_["loop"] = T_LOOP;
    keywords_["endloop"] = T_ENDLOOP;

    nextChar();
}
```

(d) Был расширен метод `statement` класса *Parser*:

```
void Parser::statement()
{
  ...

  else if(match(T_WRITE)) {
      mustBe(T_LPAREN);
      expression();
      mustBe(T_RPAREN);
      codegen_->emit(PRINT);
  }


  else if (see(T_LOOP)) {
       // Запоминаем номер строки, на которой находится инструкция loop.
       int loopStr = scanner_->getLineNumber();
       codegen_->emit(PUSH, loopStr);
       codegen_->emit(PRINT);
       next();

       // Считываем количество итераций тела цикла.
       int iterationsAmount = findOrAddVariable(codegen_->getCurrentAddress()
                                                + "iterationsAmount");
       int inputStr = codegen_->getCurrentAddress();
```

```
        codegen_->emit(INPUT);
        codegen_->emit(STORE, iterationsAmount);

        // Проверяем условие iterationsAmount >= 0.
        codegen_->emit(LOAD, iterationsAmount);
        codegen_->emit(PUSH, 0);
        codegen_->emit(COMPARE, 5);

        // Если введено отрицательное число, считываем iterationsAmount повторно.
        codegen_->emit(JUMP_NO, inputStr);

        // Запоминаем адрес начала выполнения тела цикла.
        int loopBeginningAddress = codegen_->getCurrentAddress();

        // Проверяем условие выполнения тела цикла.
        codegen_->emit(LOAD, iterationsAmount);
        codegen_->emit(PUSH, 0);
        codegen_->emit(COMPARE, 0);

        // Резервируем место под инструкцию условного перехода
        // для выхода из цикла.
        int jumpExitAddress = codegen_->reserve();

        // Выполняем список операторов.
        statementList();

        // Уменьшаем количество оставшихся итераций.
        codegen_->emit(LOAD, iterationsAmount);
        codegen_->emit(PUSH, 1);
        codegen_->emit(SUB);
        codegen_->emit(STORE, iterationsAmount);

        // Проверяем наличие лексемы endloop.
        mustBe(T_ENDLOOP);

        // Переходим на адрес проверки условия.
        codegen_->emit(JUMP, loopBeginningAddress);

        // Заполняем зарезервированный адрес инструкцией условного
        // перехода на следующий за циклом оператор.
        int exitStr = codegen_->getCurrentAddress();
        codegen_->emitAt(jumpExitAddress, JUMP_YES, exitStr);
    }
```

```
    else {
       reportError("statement expected.");
    }
}
```

5 Примеры

1. Код программы

```
begin
    k := 3;
    loop
        i:=read;
        write(k+i)
    endloop
end
```

VM-код:

```
Переменные:
;  k находится по адресу 0,
;  iterationsAmount - по адресу 1,
;  i - по адресу 2

0:    PUSH     3
1:    STORE    0   ; k:=3

; начало цикла "loop ... endloop"
2:    PUSH     3   ; загрузка номера строки начала иструкции loop
3:    PRINT        ; write(3)
4:    INPUT
5:    STORE    1   ; iterationsAmount:=read
6:    LOAD     1   ; iterationsAmount
7:    PUSH     0
8:    COMPARE  5   ; проверка условия "iterationsAmount>=0"
9:    JUMP_NO  4   ; переход к инструкции ввода, если условие
                   ; "iterationsAmount>=0" ложно
10:   LOAD     1   ; iterationsAmount
11:   PUSH     0   ;
12:   COMPARE  0   ; проверка условия "iterationsAmount=0"
13:   JUMP_YES 25  ; переход на команду сразу после цикла
                   ; "loop ... endloop"
14:   INPUT
15:   STORE    2   ; i:=read
16:   LOAD     0   ; k
17:   LOAD     2   ; i
18:   ADD          ; k+i
19:   PRINT        ; write(k+i)
20:   LOAD     1   ; iterationsAmount
21:   PUSH     1
22:   SUB          ; iterationsAmount-1
```

```
23:   STORE    1   ; iterationsAmount:=iterationsAmount-1
24:   JUMP     10  ; переход на адрес проверки условия исполнения цикла
; loop
25:   STOP
```

Результат работы программы:

```
Reading input from test.txt
3
> 3
> 2
5
> 2
5
> 33
36
```

Результат работы программы при вводе отрицательного числа:

```
Reading input from test.txt
3
> -2
> -3
> -1
> 0
```

Результат работы программы при вводе не числа:

```
Reading input from test.txt
3
> ghj
Error: illegal input
Code:

        4       INPUT
VM error
```

2. Код программы, содержащей конструкцию "`loop loop` S_2 `endloop,` S_1 `endloop`":

```
begin
    loop
        loop
            write(1)
        endloop;
        write(2)
    endloop
end
```

VM-код:

```
Переменные:
;  iterationsAmount1 находится по адресу 0,
;  iterationsAmount2 - по адресу 1,

; начало цикла "loop ... endloop (1)"
0:    PUSH    2   ; загрузка номера строки начала иструкции loop (1)
1:    PRINT       ; write(2)
2:    INPUT
3:    STORE   0   ; iterationsAmount1:=read
4:    LOAD    0   ; iterationsAmount1
5:    PUSH    0
6:    COMPARE 5   ; проверка условия "iterationsAmount1>=0"
7:    JUMP_NO 2   ; переход к инструкции ввода, если условие
                  ;"iterationsAmount1>=0" ложно
8:    LOAD    0   ; iterationsAmount1
9:    PUSH    0   ;
10:   COMPARE 0   ; проверка условия "iterationsAmount1=0"
11:   JUMP_YES 38 ; переход на команду сразу после цикла
                  ; "loop ... endloop (1)"

; начало цикла "loop ... endloop (2)"
12:   PUSH     3  ; загрузка номера строки начала иструкции loop (2)
13:   PRINT       ; write(3)
14:   INPUT
15:   STORE    1  ; iterationsAmount2:=read
16:   LOAD     1  ; iterationsAmount2
17:   PUSH     0
18:   COMPARE  5  ; проверка условия "iterationsAmount2>=0"
19:   JUMP_NO  14 ; переход к инструкции ввода, если условие
                  ; "iterationsAmount2>=0" ложно
20:   LOAD     1  ; iterationsAmount2
21:   PUSH     0  ;
22:   COMPARE  0  ; проверка условия "iterationsAmount2=0"
```

```
23:   JUMP_YES 31 ; переход на команду сразу после цикла
                  ; "loop ... endloop (2)"
24:   PUSH     2
25:   PRINT       ; write(2)
26:   LOAD     1  ; iterationsAmount2
27:   PUSH     1
28:   SUB         ; iterationsAmount2 - 1
29:   STORE    1  ; iterationsAmount2:= iterationsAmount2 - 1
30:   JUMP     20 ; переход на адрес проверки условия исполнения цикла
                  ; loop (2)
; конец цикла "loop ... endloop (2)"

31:   PUSH     1
32:   PRINT       ; write(1)
33:   LOAD     0  ; iterationsAmount1
34:   PUSH     1
35:   SUB         ; iterationsAmount1 - 1
36:   STORE    0  ; iterationsAmount1:=iterationsAmount1 - 1
37:   JUMP     8 ; переход на адрес проверки условия исполнения цикла
                  ; loop (1)
; конец цикла "loop ... endloop (1)"

38:   STOP
```

Результат работы программы:

```
Reading input from test.txt
2
> 1
3
> 1
2
1
```

3. Код программы с ошибкой:

```
begin
    k:=read;
    loop
        if (k-k/2*2)=0 then
            write(1)
        else
            write(0)
        fi
    od
end
```

Результат компиляции:

```
Line 3: 'OD' found while 'ENDLOOP' expected.
Line 4: end of file found while 'END' expected.
```

Пояснение результата компиляции:
Компиляция завершилась неудачно, поскольку в Milan-программе специально была допущена синтаксическая ошибка. Синтаксический анализатор, встретив лексему `loop`, ожидает найти и её окончание - `endloop`, однако встречает `od`. Таким образом, кодогенерация для стековой машины прекращается, когда в `Parser::statement()` встречается метод `Parser::mustBe(T_ENDLOOP)`, который ожидает получить лексему `endloop`, но встречает `od`, из-за чего вылетает в метод `Parser::reportError()`, который выводит сообщение «`Line 3: 'OD' found while 'ENDLOOP' expected.`» в консоль. Далее происходит возврат в метод `mustBe`, где содержится метод `recover`, посредством которого происходит попытка восстановления компиляции после ошибки. Однако так как последовательность лексем уже нарушена, то в консоль выводится сообщение о новой ошибке: `Line 4: end of file found while 'END' expected.`

Результат работы `Debug`:

```
cmilan.exe!Parser::reportError(const std::string & message) Line 98
cmilan.exe!Parser::mustBe(t) Line 416
cmilan.exe!Parser::statement() Line 172
cmilan.exe!Parser::statementList() Line 40
cmilan.exe!Parser::program() Line 19
cmilan.exe!Parser::parse() Line 10
cmilan.exe!main(int argc, char ** argv) Line 25
```

Цепочка вызовов функций компилятора при этом такова:

`main()->parse()->program()->statementList()->statement()->mustBe()->reportError();`

6 Литература

1. Курс «Теория автоматов и формальных языков», Герасимов А. С., Карпов Ю. Г.
   https://dl.spbstu.ru/course/view.php?id=3759

2. Компилятор CMilan, Аллахвердиев Э.Ф., Тимофеев Д.А.
   https://dl.spbstu.ru/pluginfile.php/298156/mod_resource/content/6/milan_compiler_doc.pdf

3. Документация по виртуальной машине Милана, Тимофеев Д.А.
   https://dl.spbstu.ru/pluginfile.php/298157/mod_resource/content/3/milan_vm_doc.pdf

4. Язык программирования Си, Брайан Керниган, Деннис Ритчи, 3-е издание
   https://www.r-5.org/files/books/computers/languages/c/kr/Brian_Kernighan_Dennis_Ritchie-The_C_Programming_Language-RU.pdf