

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИИ
«Санкт-Петербургский Политехнический университет Петра Великого»
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

Курс: Методы разработки компиляторов

Отчет по лабораторной работе №1
«Доработка компилятора языка MiLan»

Выполнила
студентка группы 3530201/90101

Андреева Наталия Сергеевна

Принял

Востров Алексей Владимирович

Санкт-Петербург, 2022

Содержание

Введение	3
1 Математическое описание	4
1.1 Общие сведения о языке MiLan	4
1.2 Компилятор языка MiLan	4
1.3 Дополненная грамматика языка MiLan	6
1.4 Виртуальная машина MiLan	9
2 Особенности реализации	11
2.1 Распознавание новых лексем	11
2.2 Новые поля и методы класса Parser	11
2.3 Объявление массива	12
2.4 Доступ к элементам массива	13
2.5 Поэлементные операции над массивами	16
2.6 Объединение и пересечение массивов	18
2.7 Удаление последнего элемента	21
3 Результаты работы программы	23
3.1 Объявление массива и доступ к элементам	23
3.2 Поэлементные арифметические операции	24
3.3 Пересечение массивов	26
3.4 Объединение массивов	29
3.5 Удаление элемента	31
3.6 Ошибки, отслеживаемые компилятором	34
Заключение	37
Список литературы	38
Приложение	39

Введение

В данной работе необходимо дополнить компилятор языка MiLan так, чтобы была возможность создавать программы, содержащие одномерные массивы. Необходимо реализовать следующие операции, связанные с массивами:

- объявление массива;
- доступ к элементам массива;
- поэлементные арифметические операции над массивами;
- операция объединения массивов;
- операция пересечения массивов;
- удаление последнего элемента массива.

1 Математическое описание

1.1 Общие сведения о языке MiLan

MiLan — это учебный язык программирования.

Программа на языке MiLan представляет собой последовательность операторов, заключенных между ключевыми словами `begin` и `end`. Операторы отделяются друг от друга точкой с запятой, при этом после последнего оператора в блоке точка с запятой не ставится. Также программа может содержать комментарии, которые заключены между последовательностями символов `/*` и `*/`.

Базовая версия языка MiLan содержит следующие конструкции:

- знаковые целочисленные константы: последовательность цифр, перед которой может находиться знак `'-'`;
- идентификаторы: последовательность латинских букв и цифр, начинающаяся с буквы;
- арифметические операции над целыми числами: сложение `'+'`, вычитание `'-'`, умножение `'*'` и целочисленное деление `/'`; приоритет у операций умножения и деления, действия выполняются слева направо;
- чтение со стандартного ввода: ключевое слово `read`, может комбинироваться с арифметическими операциями;
- печать чисел на стандартный вывод: конструкция `write (параметр)`, где параметром могут быть все перечисленные выше конструкции;
- оператор присваивания: последовательность символов `':='`;
- условный оператор;
- цикл с предусловием.

1.2 Компилятор языка MiLan

Компилятор языка MiLan состоит из трех компонентов:

- лексический анализатор;
- синтаксический анализатор;
- генератор кода.

Лексический анализатор посимвольно читает из входного потока текст программы и преобразует группы символов в лексемы (терминальные символы грамматики языка MiLan). При этом регистр не учитывается. Также лексический анализатор отслеживает номер текущей строки, который используется синтаксическим

анализатором при формировании сообщений об ошибках. Также лексический анализатор удаляет пробельные символы и комментарии. При формировании лексем анализатор идентифицирует ключевые слова и последовательности символов, а также определяет значения числовых констант и имена переменных. Эти значения становятся значениями атрибутов, связанных с лексемами.[1]

Лексемы базовой версии языка MiLan приведены в таблице 1. Цветом отмечены лексемы, добавленные в язык MiLan при его расширении.

Таблица 1 – Лексемы языка MiLan

Имя лексемы	Значение	Атрибут
T_EOF	Конец текстового потока	
T_ILLEGAL	Недопустимая лексема	
T_IDENTIFIER	Идентификатор	Имя переменной
T_NUMBER	Константа	Значение константы
T_BEGIN	Ключевое слово 'begin'	
T_END	Ключевое слово 'end'	
T_IF	Ключевое слово 'if'	
T_THEN	Ключевое слово 'then'	
T_ELSE	Ключевое слово 'else'	
T_FI	Ключевое слово 'fi'	
T_WHILE	Ключевое слово 'while'	
T_DO	Ключевое слово 'do'	
T_OD	Ключевое слово 'od'	
T_WRITE	Ключевое слово 'write'	
T_READ	Ключевое слово 'read'	
T_ARRAY	Ключевое слово 'array'	
T_DELETE	Ключевое слово 'delete'	
T_ASSIGN	Оператор ':='	
T_ADDOP	Операция типа сложения	A_PLUS('+'), A_MINUS('-')
T_MULOP	Операция типа умножения	A_MULTIPLY('*'), A_DIVIDE('/')
T_ARROP	Операция над массивами	A_MULTIPLY('&'), A_PLUS(' ')
T_CMP	Оператор отношения	C_EQ('='), C_NE('!='), C_LT('<'), C_LE('<='), C_GT('>'), C_GE('>=')
T_LPAREN	Открывающая скобка '('	
T_RPAREN	Закрывающая скобка ')'	
T_LQPAREN	Открывающая скобка '['	
T_RQPAREN	Закрывающая скобка ']'	
T_SEMICOLON	Точка с запятой ';'	

Задачами **синтаксического анализатора** является проверка соответствия программы грамматике языка MiLan и формирование кода для виртуальной машины MiLan в соответствии со структурой программы. Синтаксический анализ выполняется методом рекурсивного спуска. Каждому нетерминальному символу грамматики сопоставлен метод, выполняющий проверку соответствия последовательности лек-

сем одному из тех правил грамматики, в левой части которых стоит данный нетерминальный символ. Семантические действия, т. е. генерация кода, встроены в код метода. [1]

Основной задачей **генератора кода** является хранение и заполнение буфера инструкций последовательностью команд для виртуальной машины MiLan. Генератор кода не отвечает за правильность этой последовательности и не выполняет никаких семантических преобразований. [1]

Лексический анализатор реализован в классе **Scanner**, синтаксический анализатор — в классе **Parser**, генератор кода — в классе **CodeGen**.

1.3 Дополненная грамматика языка MiLan

Далее приведена грамматика языка MiLan в форме Бэкуса-Наура. Цветом выделены конструкции, добавленные в процессе расширения языка.

```

<program> ::= 'begin' <statementList> 'end'
<statementList> ::= <statement> ';' <StatementList>
                  | ε
<statement> ::= <ident> ':' <expression>
               | <ident> ':' <arrExpression>
               | <ident> '[' <expression> ']' ':' <expression>
               | <ident> ':' '[' <ident> <arrop> <ident> ']'
               | 'if' <relation> 'then' <statementList> ['else' <statementList>] 'fi'
               | 'while' <relation> 'do' <statementList> 'od'
               | 'write' '(' <expression> ')'
               | 'array' <ident> '[' <number> ']'
               | 'delete' '(' <ident> ')'
<expression> ::= <term> { <addop> <term> }
<term> ::= <factor> { <mulop> <factor> }
<factor> ::= <ident>
           | <number>
           | '-' <factor>
           | '(' <expression> ')'
           | <ident> '[' <expression> ']'
<arrExpression> ::= <arrTerm> { <addop> <arrTerm> }
<arrTerm> ::= <arrFactor> { <mulop> <arrFactor> }
<arrFactor> ::= <ident>
              | '-' <arrFactor>
              | '(' <arrExpression> ')'
<relation> ::= <expression> <cmp> <expression>
<addop> ::= '+' | '-'
<mulop> ::= '*' | '/'
<arrop> ::= '&' | '|'
<cmp> ::= '=' | '!=' | '<' | '<=' | '>' | '>='
<number> ::= <digit> { <digit> }
<ident> ::= <letter> { <letter> | <digit> }

```

$\langle \text{letter} \rangle ::= 'a'|'b'|'c'| \dots |'x'|'y'|'z'|'A'|'B'|'C'| \dots |'X'|'Y'|'Z'$
 $\langle \text{digit} \rangle ::= '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'$

На рисунках 1-5 приведены синтаксические диаграммы для продукций, расширяющих БНФ-нотацию языка MiLan.

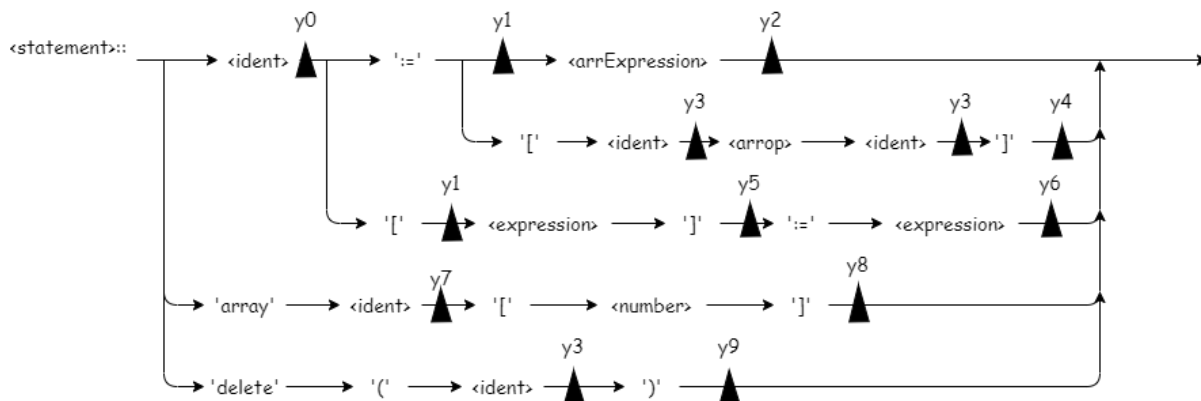


Рис. 1 – Синтаксическая диаграмма продукций нетерминала $\langle \text{statement} \rangle$

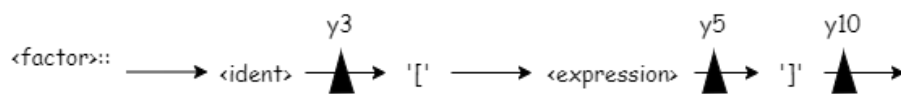


Рис. 2 – Синтаксическая диаграмма продукций нетерминала $\langle \text{factor} \rangle$



Рис. 3 – Синтаксическая диаграмма продукций нетерминала $\langle \text{arrExpression} \rangle$

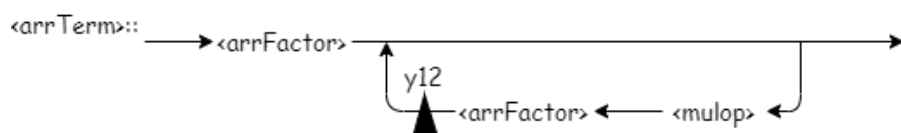


Рис. 4 – Синтаксическая диаграмма продукций нетерминала $\langle \text{arrTerm} \rangle$

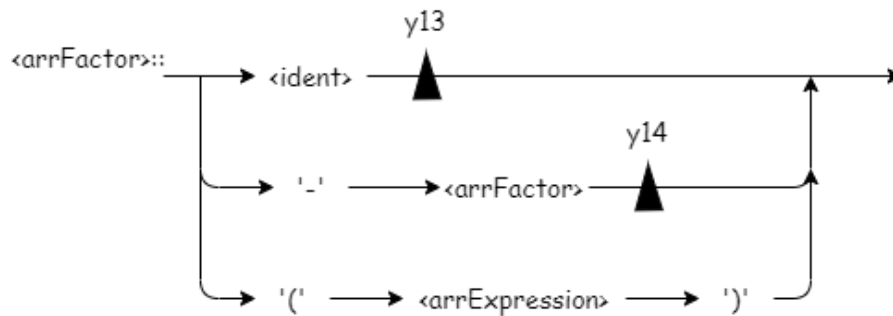


Рис. 5 – Синтаксическая диаграмма продукций нетерминала $\langle arrFactor \rangle$

Треугольниками отмечены следующие семантические операции:

- y0: сохранить адрес переменной или массива с именем $\langle ident \rangle$
- y1: проверить, что $\langle ident \rangle$ - массив, сохранить адрес его размера
- y2: записать значения в стеке в массив по адресу из y0
- y3: проверить, что $\langle ident \rangle$ - массив, сохранить его адрес и адрес его размера
- y4: сгенерировать код для операции $\langle arrOp \rangle$, сохранить результат в необходимый массив, если позволяет размер
- y5: сравнить индекс массива с 0 и размером массива, сохранить значение индекса
- y6: сохранить значение с вершины стека по адресу 'адрес $\langle ident \rangle$ ' + $\langle expression \rangle$
- y7: проверить наличие переменной с именем $\langle ident \rangle$ (должно отсутствовать)
- y8: сохранить адрес нового массива, зарезервировать место в памяти, сохранить размер
- y9: обнулить значение последнего элемента, уменьшить размер на 1
- y10: загрузить значение по адресу 'адрес $\langle ident \rangle$ ' + $\langle expression \rangle$ на вершину стека
- y11: выполнить операцию $\langle addOp \rangle$
- y12: выполнить операцию $\langle mulOp \rangle$
- y13: проверить, что $\langle ident \rangle$ - массив, сравнить его размер с размером массива, в который будет записан результат (должны совпадать), загрузить значение элемента массива в стек
- y14: инвертировать значение на вершине стека

1.4 Виртуальная машина MiLan

Виртуальная машина Милана состоит из памяти команд, памяти данных и стека. В памяти команд находятся исполняемые инструкции. Память данных используется для хранения значений переменных. Стек является рабочей областью: команды виртуальной машины считывают свои аргументы из стека и затапливают в стек результаты.

В виртуальной машине MiLan используются следующие команды:

- **NOP** — пустая команда;
- **STOP** — остановка работы программы;
- **LOAD <адрес>** — загрузка в стек значения из памяти по адресу <адрес>;
- **STORE <адрес>** — загрузка в память по адресу <адрес> значения на вершине стека;
- **BLOAD <адрес>** — загрузка в стек значения, адрес которого вычисляется как <адрес> + <значение с вершины стека>. Значение с вершины стека удаляется.
- **BSTORE <адрес>** — вычисляет адрес в памяти по формуле: <адрес в памяти> = <адрес> + <значение с вершины стека>. Значение с вершины стека при этом удаляется. Выталкивает из стека слово и записывает его по адресу <адрес в памяти>.
- **PUSH <значение>** — загрузка <значение> в стек;
- **POP** — удаление значения с вершины стека;
- **DUP** — дублирование значения на вершине стека;
- **ADD** — сумма двух верхних значений, удаляемых из стека, загружается в стек;
- **MULT** — произведение двух верхних значений, удаляемых из стека, загружается в стек;
- **SUB** — работа с двумя верхними значениями стека: верхнее вычитается из нижнего;
- **DIV** — работа с двумя верхними значениями стека: нижнее делится на верхнее, деление целочисленное;
- **INVERT** — изменение знака числа на вершине стека на противоположный;
- **COMPARE <код>** — сравнение двух чисел (с выталкиванием их из стека) на вершине стека и загрузка в стек 1 при выполнении равенства/неравенства и 0 в противоположном случае (a — число на вершине, b — следующее за ним)
 - 0: $b = a$;
 - 1: $b \neq a$;

- 2: $b < a$;
 - 3: $b > a$;
 - 4: $b \leq a$;
 - 5: $b \geq a$;
- JUMP <адрес> — переход к команде по адресу <адрес>;
 - JUMP_YES <адрес> — переход к команде по адресу <адрес>, если на вершине стека лежит ненулевое значение (выталкивается из стека);
 - JUMP_NO <адрес> — переход к команде по адресу <адрес>, если на вершине стека лежит нулевое значение (выталкивается из стека);
 - INPUT — считывание числа со стандартного ввода и загрузка его в стек;
 - PRINT — выводит на стандартный вывод значение с вершины стека (выталкивается из стека).

2 Особенности реализации

2.1 Распознавание новых лексем

Чтобы лексический анализатор мог распознать ключевые слова 'array' и 'delete' их достаточно добавить в ассоциативный массив `keywords_`, содержащий все ключевые слова. Добавление происходит в конструкторе класса `Scanner` с помощью следующих инструкций:

```
keywords_["delete"] = T_DELETE;
keywords_["array"] = T_ARRAY;
```

Для распознавания остальных новых лексем был модифицирован метод `nextToken` класса `Scanner`. В конструкцию `switch` были внесены следующие варианты:

```
case '[':
    token_ = T_LQPAREN;
    nextChar();
    break;
case ']':
    token_ = T_RQPAREN;
    nextChar();
    break;
case '&':
    token_ = T_ARROP;
    arithmeticValue_ = A_MULTIPLY;
    nextChar();
    break;
case '|':
    token_ = T_ARROP;
    arithmeticValue_ = A_PLUS;
    nextChar();
    break;
```

2.2 Новые поля и методы класса `Parser`

В класс `Parser` были добавлены следующие поля:

- `VarTable arrays_` — таблица массивов, связывающая имя массива с адресом его нулевого элемента;
- `VarTable arraySizes_` — таблица размеров массивов, связывающая имя массива и адрес его размера в памяти машины `MiLan`;
- `int reserveAddress_` — адрес ячейки памяти, которая зарезервирована для вычислений, инициализируется 0 в конструкторе.

Также в конструкторе поле `lastVar_` вместо 0 инициализируется числом 3.

В класс `Parser` были добавлены методы:

- `int findVariable(const string&)` — метод для поиска адреса переменной, если переменной с переданным именем нет, то возвращается -1;
- `int findArray(const string&)` — метод для поиска адреса нулевого элемента массива, если массива с переданным именем нет, то возвращается -1;
- `int addArray(const string&, int offset)` — если метод находит массив с переданным именем, то возвращает -1, иначе добавляет в `arrays_` массив с адресом `lastVar_`, увеличивает значение `lastVar_` на `offset`, а также добавляет в `arraySizes_` имя массива и новое значение `lastVar_`;
- `int findSize(const string&)` — метод для поиска размера массива по его имени;
- `void arrExpression()` — метод для разбора поэлементных операций над массивами;
- `void arrTerm()` — метод для разбора слагаемого массивов;
- `void arrFactor()` — метод для разбора произведения массивов;
- `void clear()` — метод, генерирующий код для освобождения памяти после выполнения операций объединения или пересечения;
- `void copyToDest(int address, int size)` — метод, генерирующий код для копирования массива, полученного объединением или пересечением массивов, по адресу `address`;
- `void orCode(int arrAddress, int sizeAddress)` — метод, генерирующий код для объединения массивов без повторов;
- `void andCode(int arrAddress1, int sizeAddress1, int arrAddress2, int sizeAddress2)` — метод, генерирующий код для пересечения массивов без повторов.

Код методов приведен в приложении.

2.3 Объявление массива

Объявлению массива соответствует следующая конструкция:

```
<statement> ::= 'array' <ident> '[' <number> '']'
```

Данная конструкция реализована в методе `statement` класса `Parser`. В ветвление, представленном в данном методе, была добавлена новая ветка:

```
1  if (match(T_ARRAY)) {
2      string ident = "";
3      int size = 0;
4      if (!see(T_IDENTIFIER)) {
5          reportError("identifier expected.");
```

```

6      }
7      else {
8          ident = scanner_->getStringValue();
9          if (findVariable(ident) != -1) {
10             reportError("variable with such name already exists.");
11         }
12     }
13     next();
14     mustBe(T_LQPAREN);
15     if (!see(T_NUMBER) || scanner_->getIntValue() <= 0) {
16         reportError("positive number expected.");
17         size = 1;
18     }
19     else {
20         size = scanner_->getIntValue();
21     }
22     next();
23     mustBe(T_RQPAREN);
24     if (!ident.empty() && size != 0) {
25         int arrAddress = addArray(ident, size);
26         if (arrAddress < 0) {
27             reportError("redefining an existing array.");
28         }
29         else {
30             int addr = findSize(ident);
31             codegen_->emit(PUSH, size);
32             codegen_->emit(STORE, addr);
33         }
34     }
35 }

```

Если встречено ключевое слово 'array' (строка 1), то за ним должен следовать идентификатор массива. Если это не так нужно сообщить об ошибке (строки 4-6). Если был встречен идентификатор, то он не должен быть таким же, как и у же существующих переменных (строки 9-11). Далее должно следовать положительное число, окруженное квадратными скобками, если число неположительное, то генерируется сообщение об ошибке (строки 14-23). Также необходимо проверить, что не существует массива с таким же именем (строки 26-28).

Результатом работы является сохранение имени массива и его адреса в таблице массивов, сохранение адреса размера массива в таблице размеров. Также генерируются инструкции для загрузки размера массива в память.

2.4 Доступ к элементам массива

Доступ к элементам массива происходит по схеме следующих выражений из БНФ-нотации:

```

<statement> ::= <ident> '[' <expression> ']' ':' <expression>
<factor> ::= <ident> '[' <expression> ']'

```

В первом случае необходимо загрузить значение по адресу элемента массива, а во втором — извлечь из памяти значение элемента массива.

Далее приведена часть кода метода `statement` класса `Parser`, которая реализует первое выражение БНФ-нотации.

```

1  if(see(T_IDENTIFIER)) {
2      string ident = scanner_->getStringValue();
3      next();
4      if (match(T_LQPAREN)) {
5          int address = findArray(ident);
6          if (address == -1) {
7              std::ostringstream msg;
8              msg << "no such array: " << ident << ".";
9              reportError(msg.str());
10             recover(T_ASSIGN);
11             expression();
12         }
13         else {
14             expression();
15             mustBe(T_RQPAREN);
16             codegen_->emit(DUP);
17             codegen_->emit(DUP);
18             int sizeAddress = findSize(ident);
19             codegen_->emit(LOAD, sizeAddress);
20             codegen_->emit(COMPARE, 2);
21             codegen_->emit(JUMP_YES, codegen_->
22                 getCurrentAddress() + 2);
23             codegen_->emit(JUMP, -1);
24             codegen_->emit(PUSH, 0);
25             codegen_->emit(COMPARE, 5);
26             codegen_->emit(JUMP_YES, codegen_->
27                 getCurrentAddress() + 2);
28             codegen_->emit(JUMP, -1);
29             codegen_->emit(STORE, reserveAddress_);
30             mustBe(T_ASSIGN);
31             expression();
32             codegen_->emit(LOAD, reserveAddress_);
33             codegen_->emit(BSTORE, address);
34         }
35         ...

```

В рассматриваемом случае после идентификатора должна следовать открывающая квадратная скобка (строки 1-4). Если это не так, то рассматриваются другие варианты, соответствующие БНФ-нотации. В отличие от переменных, массив можно

использовать только после его объявления, поэтому необходимо проверить наличие указанного массива и в случае отсутствия такого массива сообщить об ошибке (строки 5-12). Далее внутри квадратных скобок следует конструкция **expression**, которая позволяет сгенерировать код для машины MiLan, который вычисляет значение индекса и кладет его на вершину стека. Это значение дважды дублируется (строки 16-17) для сравнения с 0 и размером массива. Если индекс больше размера массива или меньше 0 будет выполнена инструкция **JUMP -1**, которая вызовет ошибку работы машины MiLan. Далее индекс сохраняется по адресу **reserveAddress_** (строка 29), проверяется наличие оператора присваивания и конструкции **expression**. На вершину стека загружается индекс массива (строка 32) и происходит загрузка вычисленного выражения по адресу элемента массива (строка 33).

Далее приведена часть кода метода **factor** класса **Parser**, реализующая вторую конструкцию БНФ-нотации.

```
1  if(see(T_IDENTIFIER)) {
2      string ident = scanner_->getStringValue();
3      next();
4      if (match(T_LQPAREN)) {
5          int address = findArray(ident);
6          if (address == -1) {
7              std::ostringstream msg;
8              msg << "no such array: " << ident << ".";
9              reportError(msg.str());
10         }
11         else {
12             expression();
13             mustBe(T_RQPAREN);
14             codegen_->emit(DUP);
15             codegen_->emit(DUP);
16             int sizeAddress = findSize(ident);
17             codegen_->emit(LOAD, sizeAddress);
18             codegen_->emit(COMPARE, 2);
19             codegen_->emit(JUMP_YES, codegen_->
20                 getCurrentAddress() + 2);
21             codegen_->emit(JUMP, -1);
22             codegen_->emit(PUSH, 0);
23             codegen_->emit(COMPARE, 5);
24             codegen_->emit(JUMP_YES, codegen_->
25                 getCurrentAddress() + 2);
26             codegen_->emit(JUMP, -1);
27             codegen_->emit(BLOAD, address);
28         }
29     }
30     else {
31         int isArr = findArray(ident);
32         if (isArr != -1) {
```

```

33             std::ostringstream msg;
34             msg << "inappropriate use of array: " << ident << ".";
35             reportError(msg.str());
36         }
37         int varAddress = findOrAddVariable(ident);
38         codegen_->emit(LOAD, varAddress);
39     }
40 }

```

Если встречен идентификатор и за ним следует открывающая квадратная скобка, то идентификатор должен быть именем массива. Если такого массива нет, то генерируется ошибка (строки 6-8). Далее происходят те же действия, что и в предыдущем случае: вычисление значения индекса и сравнение с 0 и с размером массива (строки 12-26). После значение элемента массива загружается на вершину стека (строка 27).

В случае, когда открывающая квадратная скобка не найдена, идет работа с переменной. Перед ее добавлением или поиском необходимо проверить, что указанный идентификатор не массив (строки 31-36).

2.5 Поэлементные операции над массивами

На поэлементные операции над массивами накладывается следующее ограничение: размеры массивов, участвующих в операциях должны быть одинаковы.

По аналогии с переменными были созданы и реализованы выражения БНФ-нотации для поэлементных арифметических операций над массивами:

```

<statement> ::= <ident> ':' <arrExpression>
<arrExpression> ::= <arrTerm> { <addop> <arrTerm> }
<arrTerm> ::= <arrFactor> { <mulop> <arrFactor> }
<arrFactor> ::= <ident>
               | '-' <arrFactor>
               | '(' <arrExpression> ')'

```

Далее приведена часть кода метода `statement` класса `Parser`, реализующая первое выражение БНФ-нотации.

```

1  if(see(T_IDENTIFIER)) {
2      ...
3      if (match(T_LQPAREN)) {
4          ...
5      }
6  else {
7      int addr = findArray(ident);
8      if (addr == -1) {
9          int varAddress = findOrAddVariable(ident);
10         mustBe(T_ASSIGN);
11         expression();

```



```

12         codegen_->emit(STORE, varAddress);
13     }
14     else {
15         mustBe(T_ASSIGN);
16         if (match(T_LQPAREN)) {
17             ...
18         }
19         else {
20             codegen_->emit(PUSH, 0);
21             codegen_->emit(STORE, reserveAddress_);
22             codegen_->emit(LOAD, findSize(ident));
23             codegen_->emit(STORE, reserveAddress_ + 1);
24             int comandAddr = codegen_->getCurrentAddress();
25             arrExpression();
26             codegen_->emit(LOAD, reserveAddress_);
27             codegen_->emit(PUSH, 1);
28             codegen_->emit(ADD);
29             codegen_->emit(DUP);
30             codegen_->emit(STORE, reserveAddress_);
31             codegen_->emit(LOAD, reserveAddress_ + 1);
32             codegen_->emit(COMPARE, 2);
33             codegen_->emit(JUMP_YES, comandAddr);
34             codegen_->emit(LOAD, reserveAddress_);
35             codegen_->emit(PUSH, 1);
36             codegen_->emit(SUB);
37             codegen_->emit(STORE, reserveAddress_);
38             comandAddr = codegen_->getCurrentAddress();
39             codegen_->emit(LOAD, reserveAddress_);
40             codegen_->emit(BSTORE, addr);
41             codegen_->emit(LOAD, reserveAddress_);
42             codegen_->emit(PUSH, 1);
43             codegen_->emit(SUB);
44             codegen_->emit(DUP);
45             codegen_->emit(STORE, reserveAddress_);
46             codegen_->emit(PUSH, 0);
47             codegen_->emit(COMPARE, 5);
48             codegen_->emit(JUMP_YES, comandAddr);
49         }
50     }
51 }

```

В рассматриваемом случае до оператора присваивания должен стоять только идентификатор. Этот идентификатор может быть как массивом, так и переменной. Если массив с таким именем не найден, то идет работа с переменной (строки 7-13). В ином случае происходит работа с массивом.

Далее по схеме должен быть оператор присваивания и конструкция `arrExpression`.

Перед вызовом метода `arrExpression` происходит инициализация зарезервированной памяти: по нулевому адресу хранится индекс элементов, а по первому — адрес размера массива, в который будет записан результат (строки 20-23). В строках 26-33 индекс массива увеличивается на 1 и сравнивается с размером массива: если он меньше, то происходит переход к первой инструкции сгенерированной методом `arrExpression`. В строках 34-48 генерируется код для записи результата, который лежит в стеке, в массив. Запись происходит от последнего элемента массива до нулевого.

Методы `arrExpression` и `arrTerm` идентичны методам `expression` и `term` соответственно, за исключением вызываемых внутри них методов, соответствующих конструкциям БНФ-нотации.

Далее приведена часть кода метода `arrFactor`, которая отличается от метода `factor`.

```

1  if (see(T_IDENTIFIER)) {
2      int arrAddress = findArray(scanner_>getStringValue());
3      if (arrAddress == -1) {
4          std::ostream msg;
5          msg << "Unknown array " << scanner_>getStringValue();
6          reportError(msg.str());
7      }
8      else {
9          codegen_>emit(LOAD, reserveAddress_ + 1);
10         codegen_>emit(LOAD, findSize(scanner_>getStringValue()));
11         codegen_>emit(COMPARE, 0);
12         codegen_>emit(JUMP_YES, codegen_>getCurrentAddress() + 2);
13         codegen_>emit(JUMP, -1);
14         codegen_>emit(LOAD, reserveAddress_);
15         codegen_>emit(BLOAD, arrAddress);
16     }
17     next();
18 }

```

Если встречен идентификатор, то это должен быть существующий массив. Если такого массива нет, генерируется ошибка (строки 2-7). В ином случае размер найденного массива сравнивается с размером массива, в который будет записан результат (строки 9-13). В строках 14-15 генерируется код для загрузки элемента массива в стек.

Код методов `arrExpression`, `arrTerm` и `arrFactor` приведен в приложении.

2.6 Объединение и пересечение массивов

Для объединения и пересечения массивов в БНФ-нотацию были добавлены следующие конструкции:

```

<statement> ::= <ident> ':= ' '[' <ident> <arrop> <ident> ']'
<arrop> ::= '&' | '|'

```

Далее приведена часть кода метода `statement` класса `Parser`, которая реализует указанное выражение БНФ-нотации.

```
1  if(see(T_IDENTIFIER)) {
2    ...
3    if (match(T_LQPAREN)) {
4      ...
5    }
6    else {
7      int addr = findArray(ident);
8      if (addr == -1) {
9        ...
10     }
11    else {
12      mustBe(T_ASSIGN);
13      if (match(T_LQPAREN)) {
14        codegen_->emit(PUSH, 0);
15        codegen_->emit(STORE, reserveAddress_);
16        codegen_->emit(PUSH, 0);
17        codegen_->emit(STORE, reserveAddress_ + 1);
18        codegen_->emit(PUSH, 0);
19        codegen_->emit(STORE, reserveAddress_ + 2);
20        int addr1, addr2;
21        int size1, size2;
22        if (see(T_IDENTIFIER)) {
23          addr1 = findArray(scanner_->getStringValue());
24          if (addr1 == -1) {
25            reportError("the first argument must be array.");
26          }
27          else {
28            size1 = findSize(scanner_->getStringValue());
29          }
30        }
31        else {
32          std::ostringstream msg;
33          msg << "array identifier expected but found "
34            << tokenToString(scanner_->token()) << ' ';
35          reportError(msg.str());
36        }
37        next();
38        Arithmetic op;
39        if (see(T_ARROP)) {
40          op = scanner_->getArithmeticValue();
41        }
42        else {
43          std::ostringstream msg;
```

```

44         msg << tokenToString(T_ARROP) << " expected but found "
45         << tokenToString(scanner_->token()) << ' ';
46         reportError(msg.str());
47     }
48     next();
49     if (see(T_IDENTIFIER)) {
50         addr2 = findArray(scanner_->getStringValue());
51         if (addr2 == -1) {
52             reportError("the second argument must be array.");
53         }
54         else {
55             size2 = findSize(scanner_->getStringValue());
56         }
57     }
58     else {
59         std::ostringstream msg;
60         msg << "array identifier expected but found "
61         << tokenToString(scanner_->token()) << ' ';
62         reportError(msg.str());
63     }
64     next();
65     mustBe(T_RQPAREN);
66     if (op == A_PLUS) {
67         orCode(addr1, size1);
68         codegen_->emit(PUSH, 0);
69         codegen_->emit(STORE, reserveAddress_);
70         orCode(addr2, size2);
71     }
72     else {
73         andCode(addr1, size1, addr2, size2);
74     }
75     copyToDest(addr, findSize(ident));
76     clear();
77 }
78 ...

```

В рассматриваемом случае перед оператором присваивания должен стоять идентификатор, соответствующий массиву (строки 1, 7-8). После оператора присваивания должна стоять открывающая квадратная скобка (строка 13). В строках 14-19 генерируется код, который инициализирует зарезервированную память: по нулевому адресу хранится индекс первого массива, по первому — вспомогательный индекс, по второму — размер массива-результата.

В строках 20-36 ищутся адрес массива, определяемого идентификатором, и адрес, по которому хранится размер этого массива. Также генерируются ошибки, если идентификатор не был встречен или происходит обращение к необъявленному массиву. В строках 38-47 определяется тип операции: объединение или пересечение, а

также генерируются ошибки, если не указана ни одна из этих операций. В строках 49-63 ищутся адрес второго массива и его размера по аналогии с первым.

Если была использована операция объединения, то для каждого массива выполняется метод `orCode`, генерирующий код для объединения (строки 66-71). Если была использована операция пересечения, то вызывается метод `andCode`, работающий одновременно с двумя массивами (строка 73). Затем полученный массив с помощью метода `copyToDest` копируется в массив, стоящий перед оператором присваивания, а также методом `clear` освобождается память использованная при работе с массивами.

Код методов `orCode`, `andCode`, `copyToDest` и `clear` приведен в приложении.

2.7 Удаление последнего элемента

Для удаления последнего элемента массива в БНФ-нотацию была добавлена следующая конструкция:

`<statement> ::= 'delete' '(' <ident> ')'`

Далее приведена часть кода метода `statement` класса `Parser`, реализующая указанное выражение.

```
1  if (match(T_DELETE)) {
2      match(T_LPAREN);
3      if (see(T_IDENTIFIER)) {
4          string ident = scanner_->getStringValue();
5          int address = findArray(ident);
6          if (address == -1) {
7              std::ostringstream msg;
8              msg << "Unknown array " << ident << ' ';
9              reportError(msg.str());
10         }
11         next();
12         match(T_RPAREN);
13         int sizeAddress = findSize(ident);
14         codegen_->emit(LOAD, sizeAddress);
15         codegen_->emit(DUP);
16         codegen_->emit(PUSH, 0);
17         codegen_->emit(COMPARE, 3);
18         codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() + 2);
19         codegen_->emit(JUMP, -1);
20         codegen_->emit(PUSH, 1);
21         codegen_->emit(SUB);
22         codegen_->emit(STORE, sizeAddress);
23         codegen_->emit(PUSH, 0);
24         codegen_->emit(LOAD, sizeAddress);
25         codegen_->emit(BSTORE, address);
26     }
27 }
```

Согласно выражению сначала должно идти ключевое слово `'delete'`, за ним открывающая круглая скобка и за ней идентификатор (строки 1-3). Идентификатор должен соответствовать ранее объявленному массиву, если это не так, то генерируется ошибка (строки 6-10). После идентификатора должна быть закрывающая круглая скобка (строка 12).

В строках 14-19 происходит сравнение текущего размера массива с нулем. Если размер нулевой, то происходит ошибка, т. к. не осталось элементов, которые можно удалить. В строках 20-22 размер массива уменьшается на 1, после чего по адресу удаленного элемента записывается 0 (строки 23-25).

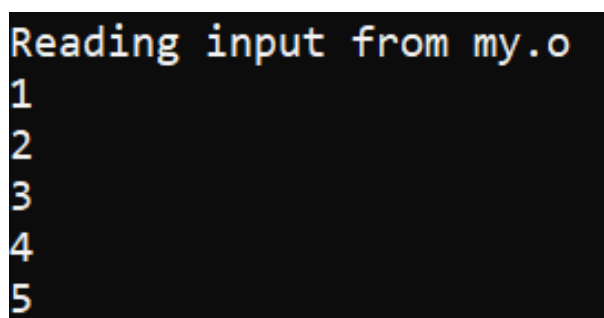
3 Результаты работы программы

3.1 Объявление массива и доступ к элементам

На рисунке 6 приведен пример программы, заполняющей массив и выводящей его в консоль. На рисунке 8 приведен код, сгенерированный компилятором. На рисунке 7 приведен результат работы виртуальной машины, которой был передан сгенерированный компилятором код.

```
BEGIN
    ARRAY arr[5];
    arr[0] := 1;
    i := 1;
    WHILE i < 5 DO
        arr[i] := arr[i-1] + 1;
        i := i + 1
    OD;
    i := 0;
    WHILE i < 5 DO
        WRITE(arr[i]);
        i := i + 1
    OD
END
```

Рис. 6 – Код программы на языке MiLan



```
Reading input from my.o
1
2
3
4
5
```

Рис. 7 – Выполнение программы виртуальной машиной MiLan

0:	PUSH	5	28:	JUMP_YES	30	56:	STORE	9
1:	STORE	8	29:	JUMP	-1	57:	JUMP	19
2:	PUSH	0	30:	PUSH	0	58:	PUSH	0
3:	DUP		31:	COMPARE	5	59:	STORE	9
4:	DUP		32:	JUMP_YES	34	60:	LOAD	9
5:	LOAD	8	33:	JUMP	-1	61:	PUSH	5
6:	COMPARE	2	34:	STORE	0	62:	COMPARE	2
7:	JUMP_YES	9	35:	LOAD	9	63:	JUMP_NO	82
8:	JUMP	-1	36:	PUSH	1	64:	LOAD	9
9:	PUSH	0	37:	SUB		65:	DUP	
10:	COMPARE	5	38:	DUP		66:	DUP	
11:	JUMP_YES	13	39:	DUP		67:	LOAD	8
12:	JUMP	-1	40:	LOAD	8	68:	COMPARE	2
13:	STORE	0	41:	COMPARE	2	69:	JUMP_YES	71
14:	PUSH	1	42:	JUMP_YES	44	70:	JUMP	-1
15:	LOAD	0	43:	JUMP	-1	71:	PUSH	0
16:	BSTORE	3	44:	PUSH	0	72:	COMPARE	5
17:	PUSH	1	45:	COMPARE	5	73:	JUMP_YES	75
18:	STORE	9	46:	JUMP_YES	48	74:	JUMP	-1
19:	LOAD	9	47:	JUMP	-1	75:	BLOAD	3
20:	PUSH	5	48:	BLOAD	3	76:	PRINT	
21:	COMPARE	2	49:	PUSH	1	77:	LOAD	9
22:	JUMP_NO	58	50:	ADD		78:	PUSH	1
23:	LOAD	9	51:	LOAD	0	79:	ADD	
24:	DUP		52:	BSTORE	3	80:	STORE	9
25:	DUP		53:	LOAD	9	81:	JUMP	60
26:	LOAD	8	54:	PUSH	1	82:	STOP	
27:	COMPARE	2	55:	ADD				

Рис. 8 – Код программы, сгенерированный компилятором

3.2 Поэлементные арифметические операции

На рисунке 9 приведен код программы на языке MiLan. В комментарии указан ожидаемый результат операций. На рисунке 11 приведен код, который был сгенерирован компилятором. На рисунке 10 изображен результат работы виртуальной машины.


```

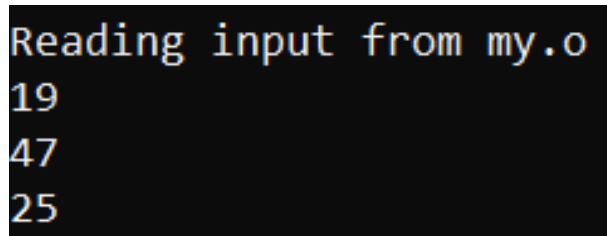
BEGIN
    ARRAY arr1[3];
    ARRAY arr2[3];
    ARRAY arr3[3];
    ARRAY arr4[3];
    ARRAY arr5[3];
    ARRAY arr6[3];

    arr1[0] := 2; arr1[1] := 7; arr1[2] := 10;
    arr2[0] := 5; arr2[1] := 3; arr2[2] := 4;
    arr3[0] := 3; arr3[1] := 5; arr3[2] := 2;
    arr4[0] := 4; arr4[1] := 9; arr4[2] := 18;
    arr5[0] := 2; arr5[1] := 3; arr5[2] := 6;

    arr6 := (arr1 + arr2) * arr3 - arr4 / arr5; /* { 19, 47, 25} */
    i:=0;
    WHILE i < 3 DO
        WRITE(arr6[i]);
        i:=i+1
    OD
END

```

Рис. 9 – Код программы на языке MiLan



```

Reading input from my.o
19
47
25

```

Рис. 10 – Выполнение программы виртуальной машиной MiLan

0:	PUSH	3	66:	JUMP_YES	68	132:	PUSH	2	198:	JUMP	-1	264:	LOAD	1	
1:	STORE	6	67:	JUMP	-1	133:	DUP		199:	PUSH	0	265:	LOAD	18	
2:	PUSH	3	68:	STORE	0	134:	DUP		200:	COMPARE	5	266:	COMPARE	0	
3:	STORE	10	69:	PUSH	5	135:	LOAD	14	201:	JUMP_YES		203 267:	JUMP_YES	269	
4:	PUSH	3	70:	LOAD	0	136:	COMPARE	2	202:	JUMP	-1	268:	JUMP	-1	
5:	STORE	14	71:	BSTORE	7	137:	JUMP_YES		139 203:	STORE	0	269:	LOAD	0	
6:	PUSH	3	72:	PUSH	1	138:	JUMP	-1	204:	PUSH	2	270:	BLOAD	15	
7:	STORE	18	73:	DUP		139:	PUSH	0	205:	LOAD	0	271:	LOAD	1	
8:	PUSH	3	74:	DUP		140:	COMPARE	5	206:	BSTORE	19	272:	LOAD	22	
9:	STORE	22	75:	LOAD	10	141:	JUMP_YES		143 207:	PUSH	1	273:	COMPARE	0	
10:	PUSH	3	76:	COMPARE	2	142:	JUMP	-1	208:	DUP		274:	JUMP_YES	276	
11:	STORE	26	77:	JUMP_YES		79 143:	STORE	0	209:	DUP		275:	JUMP	-1	
12:	PUSH	0	78:	JUMP	-1	144:	PUSH	2	210:	LOAD	22	276:	LOAD	0	
13:	DUP		79:	PUSH	0	145:	LOAD	0	211:	COMPARE	2	277:	BLOAD	19	
14:	DUP		80:	COMPARE	5	146:	BSTORE	11	212:	JUMP_YES		214 278:	DIV		
15:	LOAD	6	81:	JUMP_YES		83 147:	PUSH	0	213:	JUMP	-1	279:	SUB		
16:	COMPARE	2	82:	JUMP	-1	148:	DUP		214:	PUSH	0	280:	LOAD	0	
17:	JUMP_YES		19 83:	STORE	0	149:	DUP		215:	COMPARE	5	281:	PUSH	1	
18:	JUMP	-1	84:	PUSH	3	150:	LOAD	18	216:	JUMP_YES		218 282:	ADD		
19:	PUSH	0	85:	LOAD	0	151:	COMPARE	2	217:	JUMP	-1	283:	DUP		
20:	COMPARE	5	86:	BSTORE	7	152:	JUMP_YES		154 218:	STORE	0	284:	STORE	0	
21:	JUMP_YES		23 87:	PUSH	2	153:	JUMP	-1	219:	PUSH	3	285:	LOAD	1	
22:	JUMP	-1	88:	DUP		154:	PUSH	0	220:	LOAD	0	286:	COMPARE	2	
23:	STORE	0	89:	DUP		155:	COMPARE	5	221:	BSTORE	19	287:	JUMP_YES	241	
24:	PUSH	2	90:	LOAD	10	156:	JUMP_YES		158 222:	PUSH	2	288:	LOAD	0	
25:	LOAD	0	91:	COMPARE	2	157:	JUMP	-1	223:	DUP		289:	PUSH	1	
26:	BSTORE	3	92:	JUMP_YES		94 158:	STORE	0	224:	DUP		290:	SUB		
27:	PUSH	1	93:	JUMP	-1	159:	PUSH	4	225:	LOAD	22	291:	STORE	0	
28:	DUP		94:	PUSH	0	160:	LOAD	0	226:	COMPARE	2	292:	LOAD	0	
29:	DUP		95:	COMPARE	5	161:	BSTORE	15	227:	JUMP_YES		229 293:	BSTORE	23	
30:	LOAD	6	96:	JUMP_YES		98 162:	PUSH	1	228:	JUMP	-1	294:	LOAD	0	
31:	COMPARE	2	97:	JUMP	-1	163:	DUP		229:	PUSH	0	295:	PUSH	1	
32:	JUMP_YES		34 98:	STORE	0	164:	DUP		230:	COMPARE	5	296:	SUB		
33:	JUMP	-1	99:	PUSH	4	165:	LOAD	18	231:	JUMP_YES		233 297:	DUP		
34:	PUSH	0	100:	LOAD	0	166:	COMPARE	2	232:	JUMP	-1	298:	STORE	0	
35:	COMPARE	5	101:	BSTORE	7	167:	JUMP_YES		169 233:	STORE	0	299:	PUSH	0	
36:	JUMP_YES		38 102:	PUSH	0	168:	JUMP	-1	234:	PUSH	6	300:	COMPARE	5	
37:	JUMP	-1	103:	DUP		169:	PUSH	0	235:	LOAD	0	301:	JUMP_YES	292	
38:	STORE	0	104:	DUP		170:	COMPARE	5	236:	BSTORE	19	302:	PUSH	0	
39:	PUSH	7	105:	LOAD	14	171:	JUMP_YES		173 237:	PUSH	0	303:	STORE	27	
40:	LOAD	0	106:	COMPARE	2	172:	JUMP	-1	238:	STORE	0	304:	LOAD	27	
41:	BSTORE	3	107:	JUMP_YES		109 173:	STORE	0	239:	LOAD	26	305:	PUSH	3	
42:	PUSH	2	108:	JUMP	-1	174:	PUSH	9	240:	STORE	1	306:	COMPARE	2	
43:	DUP		109:	PUSH	0	175:	LOAD	0	241:	LOAD	1	307:	JUMP_NO	326	
44:	DUP		110:	COMPARE	5	176:	BSTORE	15	242:	LOAD	6	308:	LOAD	27	
45:	LOAD	6	111:	JUMP_YES		113 177:	PUSH	2	243:	COMPARE	0	309:	DUP		
46:	COMPARE	2	112:	JUMP	-1	178:	DUP		244:	JUMP_YES		246 310:	DUP		
47:	JUMP_YES		49 113:	STORE	0	179:	DUP		245:	JUMP	-1	311:	LOAD	26	
48:	JUMP	-1	114:	PUSH	3	180:	LOAD	18	246:	LOAD	0	312:	COMPARE	2	
49:	PUSH	0	115:	LOAD	0	181:	COMPARE	2	247:	BLOAD	3	313:	JUMP_YES	315	
50:	COMPARE	5	116:	BSTORE	11	182:	JUMP_YES		184 248:	LOAD	1	314:	JUMP	-1	
51:	JUMP_YES		53 117:	PUSH	1	183:	JUMP	-1	249:	LOAD	10	315:	PUSH	0	
52:	JUMP	-1	118:	DUP		184:	PUSH	0	250:	COMPARE	0	316:	COMPARE	5	
53:	STORE	0	119:	DUP		185:	COMPARE	5	251:	JUMP_YES		253 317:	JUMP_YES	319	
54:	PUSH	10	120:	LOAD	14	186:	JUMP_YES		188 252:	JUMP	-1	318:	JUMP	-1	
55:	LOAD	0	121:	COMPARE	2	187:	JUMP	-1	253:	LOAD	0	319:	BLOAD	23	
56:	BSTORE	3	122:	JUMP_YES		124 188:	STORE	0	254:	BLOAD	7	320:	PRINT		
57:	PUSH	0	123:	JUMP	-1	189:	PUSH	18	255:	ADD		321:	LOAD	27	
58:	DUP		124:	PUSH	0	190:	LOAD	0	256:	LOAD	1	322:	PUSH	1	
59:	DUP		125:	COMPARE	5	191:	BSTORE	15	257:	LOAD	14	323:	ADD		
60:	LOAD	10	126:	JUMP_YES		128 192:	PUSH	0	258:	COMPARE	0	324:	STORE	27	
61:	COMPARE	2	127:	JUMP	-1	193:	DUP		259:	JUMP_YES		261 325:	JUMP	304	
62:	JUMP_YES		64 128:	STORE	0	194:	DUP		260:	JUMP	-1	326:	STOP		
63:	JUMP	-1	129:	PUSH	5	195:	LOAD	22	261:	LOAD	0				
64:	PUSH	0	130:	LOAD	0	196:	COMPARE	2	262:	BLOAD	11				
65:	COMPARE	5	131:	BSTORE	11	197:	JUMP_YES		199 263:	MULT					

Рис. 11 – Код программы, сгенерированный компилятором

3.3 Пересечение массивов

На рисунке 12 приведен код, в котором используется операция пересечения массивов. В коде программы в комментарии приведен ожидаемый результат. На рисунке 14 приведен код, сгенерированный компилятором для виртуальной машины MiLan. На рисунке 13 изображен результат работы виртуальной машины.

```

BEGIN
    ARRAY arr1[5];
    ARRAY arr2[4];
    ARRAY arr3[10];

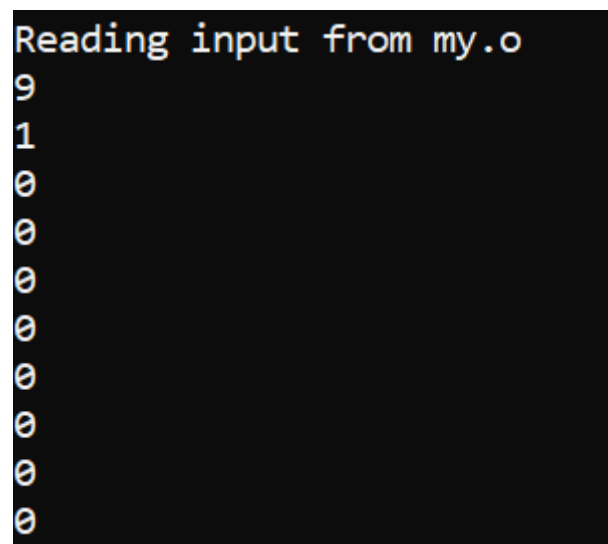
    arr1[0]:=9;
    arr1[1]:=2;
    arr1[2]:=1;
    arr1[3]:=2;
    arr1[4]:=5;

    arr2[0]:=8;
    arr2[1]:=1;
    arr2[2]:=1;
    arr2[3]:=9;

    arr3:=[arr1 & arr2];/*{9,1}*/
    i:=0;
    WHILE i < 10 DO
        WRITE(arr3[i]);
        i:=i+1
    OD
END

```

Рис. 12 – Код программы на языке MiLan



```

Reading input from my.o
9
1
0
0
0
0
0
0
0
0
0

```

Рис. 13 – Выполнение программы виртуальной машиной MiLan

0:	PUSH	5	65:	BSTORE	3	130:	COMPARE	2	195:	ADD	
1:	STORE	8	66:	PUSH	4	131:	JUMP_YES		133 196:	DUP	
2:	PUSH	4	67:	DUP		132:	JUMP	-1	197:	STORE	0
3:	STORE	13	68:	DUP		133:	PUSH	0	198:	LOAD	8
4:	PUSH	10	69:	LOAD	8	134:	COMPARE	5	199:	COMPARE	2
5:	STORE	24	70:	COMPARE	2	135:	JUMP_YES		137 200:	JUMP_YES	147
6:	PUSH	0	71:	JUMP_YES		73 136:	JUMP	-1	201:	LOAD	2
7:	DUP		72:	JUMP	-1	137:	STORE	0	202:	LOAD	24
8:	DUP		73:	PUSH	0	138:	PUSH	9	203:	COMPARE	4
9:	LOAD	8	74:	COMPARE	5	139:	LOAD	0	204:	JUMP_YES	206
10:	COMPARE	2	75:	JUMP_YES		77 140:	BSTORE	9	205:	JUMP	-1
11:	JUMP_YES		13 76:	JUMP	-1	141:	PUSH	0	206:	PUSH	0
12:	JUMP	-1	77:	STORE	0	142:	STORE	0	207:	STORE	1
13:	PUSH	0	78:	PUSH	5	143:	PUSH	0	208:	LOAD	1
14:	COMPARE	5	79:	LOAD	0	144:	STORE	1	209:	BLOAD	25
15:	JUMP_YES		17 80:	BSTORE	3	145:	PUSH	0	210:	LOAD	1
16:	JUMP	-1	81:	PUSH	0	146:	STORE	2	211:	BSTORE	14
17:	STORE	0	82:	DUP		147:	LOAD	0	212:	LOAD	1
18:	PUSH	9	83:	DUP		148:	BLOAD	3	213:	PUSH	1
19:	LOAD	0	84:	LOAD	13	149:	PUSH	0	214:	ADD	
20:	BSTORE	3	85:	COMPARE	2	150:	STORE	1	215:	DUP	
21:	PUSH	1	86:	JUMP_YES		88 151:	DUP		216:	STORE	1
22:	DUP		87:	JUMP	-1	152:	LOAD	1	217:	LOAD	2
23:	DUP		88:	PUSH	0	153:	BLOAD	9	218:	COMPARE	2
24:	LOAD	8	89:	COMPARE	5	154:	COMPARE	0	219:	JUMP_YES	208
25:	COMPARE	2	90:	JUMP_YES		92 155:	JUMP_YES		166 220:	PUSH	0
26:	JUMP_YES		28 91:	JUMP	-1	156:	LOAD	1	221:	STORE	1
27:	JUMP	-1	92:	STORE	0	157:	PUSH	1	222:	PUSH	0
28:	PUSH	0	93:	PUSH	8	158:	ADD		223:	LOAD	1
29:	COMPARE	5	94:	LOAD	0	159:	DUP		224:	BSTORE	25
30:	JUMP_YES		32 95:	BSTORE	9	160:	STORE	1	225:	LOAD	1
31:	JUMP	-1	96:	PUSH	1	161:	LOAD	13	226:	PUSH	1
32:	STORE	0	97:	DUP		162:	COMPARE	2	227:	ADD	
33:	PUSH	2	98:	DUP		163:	JUMP_YES		151 228:	DUP	
34:	LOAD	0	99:	LOAD	13	164:	POP		229:	STORE	1
35:	BSTORE	3	100:	COMPARE	2	165:	JUMP	193	230:	LOAD	2
36:	PUSH	2	101:	JUMP_YES		103 166:	LOAD	2	231:	COMPARE	2
37:	DUP		102:	JUMP	-1	167:	PUSH	0	232:	JUMP_YES	222
38:	DUP		103:	PUSH	0	168:	COMPARE	0	233:	PUSH	0
39:	LOAD	8	104:	COMPARE	5	169:	JUMP_YES		187 234:	STORE	25
40:	COMPARE	2	105:	JUMP_YES		107 170:	PUSH	0	235:	LOAD	25
41:	JUMP_YES		43 106:	JUMP	-1	171:	STORE	1	236:	PUSH	10
42:	JUMP	-1	107:	STORE	0	172:	DUP		237:	COMPARE	2
43:	PUSH	0	108:	PUSH	1	173:	LOAD	1	238:	JUMP_NO	257
44:	COMPARE	5	109:	LOAD	0	174:	BLOAD	25	239:	LOAD	25
45:	JUMP_YES		47 110:	BSTORE	9	175:	COMPARE	0	240:	DUP	
46:	JUMP	-1	111:	PUSH	2	176:	JUMP_NO	179	241:	DUP	
47:	STORE	0	112:	DUP		177:	POP		242:	LOAD	24
48:	PUSH	1	113:	DUP		178:	JUMP	193	243:	COMPARE	2
49:	LOAD	0	114:	LOAD	13	179:	LOAD	1	244:	JUMP_YES	246
50:	BSTORE	3	115:	COMPARE	2	180:	PUSH	1	245:	JUMP	-1
51:	PUSH	3	116:	JUMP_YES		118 181:	ADD		246:	PUSH	0
52:	DUP		117:	JUMP	-1	182:	DUP		247:	COMPARE	5
53:	DUP		118:	PUSH	0	183:	STORE	1	248:	JUMP_YES	250
54:	LOAD	8	119:	COMPARE	5	184:	LOAD	2	249:	JUMP	-1
55:	COMPARE	2	120:	JUMP_YES		122 185:	COMPARE	2	250:	BLOAD	14
56:	JUMP_YES		58 121:	JUMP	-1	186:	JUMP_YES		172 251:	PRINT	
57:	JUMP	-1	122:	STORE	0	187:	LOAD	2	252:	LOAD	25
58:	PUSH	0	123:	PUSH	1	188:	BSTORE	25	253:	PUSH	1
59:	COMPARE	5	124:	LOAD	0	189:	LOAD	2	254:	ADD	
60:	JUMP_YES		62 125:	BSTORE	9	190:	PUSH	1	255:	STORE	25
61:	JUMP	-1	126:	PUSH	3	191:	ADD		256:	JUMP	235
62:	STORE	0	127:	DUP		192:	STORE	2	257:	STOP	
63:	PUSH	2	128:	DUP		193:	LOAD	0			
64:	LOAD	0	129:	LOAD	13	194:	PUSH	1			

Рис. 14 – Код программы, сгенерированный компилятором

3.4 Объединение массивов

На рисунке 15 приведен код, в котором используется операция объединения массивов. В коде программы в комментарии приведен ожидаемый результат. На рисунке 17 приведен код, сгенерированный компилятором для виртуальной машины MiLan. На рисунке 16 изображен результат работы виртуальной машины.

```
BEGIN
    ARRAY arr1[7];
    ARRAY arr2[5];
    ARRAY arr3[12];

    arr1[0]:=1;
    arr1[1]:=8;
    arr1[2]:=5;
    arr1[3]:=-5;
    arr1[4]:=10;
    arr1[5]:=2;
    arr1[6]:=1;

    arr2[0]:=8;
    arr2[1]:=3;
    arr2[2]:=10;
    arr2[3]:=3;
    arr2[4]:=9;

    arr3:= [arr1 | arr2];/*{1,8,5,-5,10,2,3,9}*/

    i:=0;
    WHILE i < 12 DO
        WRITE(arr3[i]);
        i:=i+1
    OD
END
```

Рис. 15 – Код программы на языке MiLan

```
Reading input from my.o
1
8
5
-5
10
2
3
9
0
0
0
0
```

Рис. 16 – Выполнение программы виртуальной машиной MiLan

0:	PUSH	7	66:	BSTORE	3	132:	JUMP_YES	134	198:	JUMP_YES	216	264:	DUP			
1:	STORE	10	67:	PUSH	4	133:	JUMP	-1	199:	PUSH	0	265:	STORE	0		
2:	PUSH	5	68:	DUP		134:	PUSH	0	200:	STORE	1	266:	LOAD	16		
3:	STORE	16	69:	DUP		135:	COMPARE	5	201:	DUP		267:	COMPARE	2		
4:	PUSH	12	70:	LOAD	10	136:	JUMP_YES	138	202:	LOAD	1	268:	JUMP_YES	232		
5:	STORE	29	71:	COMPARE	2	137:	JUMP	-1	203:	BLOAD	30	269:	LOAD	2		
6:	PUSH	0	72:	JUMP_YES		74	138:	STORE	0	204:	COMPARE	0	270:	LOAD	29	
7:	DUP		73:	JUMP	-1	139:	PUSH	3	205:	JUMP_NO	208	271:	COMPARE	4		
8:	DUP		74:	PUSH	0	140:	LOAD	0	206:	POP		272:	JUMP_YES	274		
9:	LOAD	10	75:	COMPARE	5	141:	BSTORE	11	207:	JUMP	222	273:	JUMP	-1		
10:	COMPARE	2	76:	JUMP_YES		78	142:	PUSH	2	208:	LOAD	1	274:	PUSH	0	
11:	JUMP_YES		13	77:	JUMP	-1	143:	DUP		209:	PUSH	1	275:	STORE	1	
12:	JUMP	-1	78:	STORE	0	144:	DUP		210:	ADD		276:	LOAD	1		
13:	PUSH	0	79:	PUSH	10	145:	LOAD	16	211:	DUP		277:	BLOAD	30		
14:	COMPARE	5	80:	LOAD	0	146:	COMPARE	2	212:	STORE	1	278:	LOAD	1		
15:	JUMP_YES		17	81:	BSTORE	3	147:	JUMP_YES	149	213:	LOAD	2	279:	BSTORE	17	
16:	JUMP	-1	82:	PUSH	5	148:	JUMP	-1	214:	COMPARE	2	280:	LOAD	1		
17:	STORE	0	83:	DUP		149:	PUSH	0	215:	JUMP_YES		201	281:	PUSH	1	
18:	PUSH	1	84:	DUP		150:	COMPARE	5	216:	LOAD	2	282:	ADD			
19:	LOAD	0	85:	LOAD	10	151:	JUMP_YES	153	217:	BSTORE	30	283:	DUP			
20:	BSTORE	3	86:	COMPARE	2	152:	JUMP	-1	218:	LOAD	2	284:	STORE	1		
21:	PUSH	1	87:	JUMP_YES		89	153:	STORE	0	219:	PUSH	1	285:	LOAD	2	
22:	DUP		88:	JUMP	-1	154:	PUSH	10	220:	ADD		286:	COMPARE	2		
23:	DUP		89:	PUSH	0	155:	LOAD	0	221:	STORE	2	287:	JUMP_YES		276	
24:	LOAD	10	90:	COMPARE	5	156:	BSTORE	11	222:	LOAD	0	288:	PUSH	0		
25:	COMPARE	2	91:	JUMP_YES		93	157:	PUSH	3	223:	PUSH	1	289:	STORE	1	
26:	JUMP_YES		28	92:	JUMP	-1	158:	DUP		224:	ADD		290:	PUSH	0	
27:	JUMP	-1	93:	STORE	0	159:	DUP		225:	DUP		291:	LOAD	1		
28:	PUSH	0	94:	PUSH	2	160:	LOAD	16	226:	STORE	0	292:	BSTORE	30		
29:	COMPARE	5	95:	LOAD	0	161:	COMPARE	2	227:	LOAD	10	293:	LOAD	1		
30:	JUMP_YES		32	96:	BSTORE	3	162:	JUMP_YES	164	228:	COMPARE	2	294:	PUSH	1	
31:	JUMP	-1	97:	PUSH	6	163:	JUMP	-1	229:	JUMP_YES		193	295:	ADD		
32:	STORE	0	98:	DUP		164:	PUSH	0	230:	PUSH	0	296:	DUP			
33:	PUSH	8	99:	DUP		165:	COMPARE	5	231:	STORE	0	297:	STORE	1		
34:	LOAD	0	100:	LOAD	10	166:	JUMP_YES	168	232:	LOAD	0	298:	LOAD	2		
35:	BSTORE	3	101:	COMPARE	2	167:	JUMP	-1	233:	BLOAD	11	299:	COMPARE	2		
36:	PUSH	2	102:	JUMP_YES		104	168:	STORE	0	234:	LOAD	2	300:	JUMP_YES		290
37:	DUP		103:	JUMP	-1	169:	PUSH	3	235:	PUSH	0	301:	PUSH	0		
38:	DUP		104:	PUSH	0	170:	LOAD	0	236:	COMPARE	0	302:	STORE	30		
39:	LOAD	10	105:	COMPARE	5	171:	BSTORE	11	237:	JUMP_YES		255	303:	LOAD	30	
40:	COMPARE	2	106:	JUMP_YES		108	172:	PUSH	4	238:	PUSH	0	304:	PUSH	12	
41:	JUMP_YES		43	107:	JUMP	-1	173:	DUP		239:	STORE	1	305:	COMPARE	2	
42:	JUMP	-1	108:	STORE	0	174:	DUP		240:	DUP		306:	JUMP_NO	325		
43:	PUSH	0	109:	PUSH	1	175:	LOAD	16	241:	LOAD	1	307:	LOAD	30		
44:	COMPARE	5	110:	LOAD	0	176:	COMPARE	2	242:	BLOAD	30	308:	DUP			
45:	JUMP_YES		47	111:	BSTORE	3	177:	JUMP_YES	179	243:	COMPARE	0	309:	DUP		
46:	JUMP	-1	112:	PUSH	0	178:	JUMP	-1	244:	JUMP_NO	247	310:	LOAD	29		
47:	STORE	0	113:	DUP		179:	PUSH	0	245:	POP		311:	COMPARE	2		
48:	PUSH	5	114:	DUP		180:	COMPARE	5	246:	JUMP	261	312:	JUMP_YES		314	
49:	LOAD	0	115:	LOAD	16	181:	JUMP_YES		183	247:	LOAD	1	313:	JUMP	-1	
50:	BSTORE	3	116:	COMPARE	2	182:	JUMP	-1	248:	PUSH	1	314:	PUSH	0		
51:	PUSH	3	117:	JUMP_YES		119	183:	STORE	0	249:	ADD		315:	COMPARE	5	
52:	DUP		118:	JUMP	-1	184:	PUSH	9	250:	DUP		316:	JUMP_YES		318	
53:	DUP		119:	PUSH	0	185:	LOAD	0	251:	STORE	1	317:	JUMP	-1		
54:	LOAD	10	120:	COMPARE	5	186:	BSTORE	11	252:	LOAD	2	318:	BLOAD	17		
55:	COMPARE	2	121:	JUMP_YES		123	187:	PUSH	0	253:	COMPARE	2	319:	PRINT		
56:	JUMP_YES		58	122:	JUMP	-1	188:	STORE	0	254:	JUMP_YES		240	320:	LOAD	30
57:	JUMP	-1	123:	STORE	0	189:	PUSH	0	255:	LOAD	2	321:	PUSH	1		
58:	PUSH	0	124:	PUSH	8	190:	STORE	1	256:	BSTORE	30	322:	ADD			
59:	COMPARE	5	125:	LOAD	0	191:	PUSH	0	257:	LOAD	2	323:	STORE	30		
60:	JUMP_YES		62	126:	BSTORE	11	192:	STORE	2	258:	PUSH	1	324:	JUMP	303	
61:	JUMP	-1	127:	PUSH	1	193:	LOAD	0	259:	ADD		325:	STOP			
62:	STORE	0	128:	DUP		194:	BLOAD	3	260:	STORE	2					
63:	PUSH	5	129:	DUP		195:	LOAD	2	261:	LOAD	0					
64:	INVERT		130:	LOAD	16	196:	PUSH	0	262:	PUSH	1					
65:	LOAD	0	131:	COMPARE	2	197:	COMPARE	0	263:	ADD						

Рис. 17 – Код программы, сгенерированный компилятором

3.5 Удаление элемента

На рисунке 18 приведен код программы для удаления последнего элемента массива. На рисунке 20 приведен код, который был сгенерирован компилятором. На рисунке 19 изображен результат работы программы, запущенной на виртуальной машине. Во время выполнения возникла ошибка, т. к. произошел выход за границы массива.

```

BEGIN
    ARRAY arr[10];

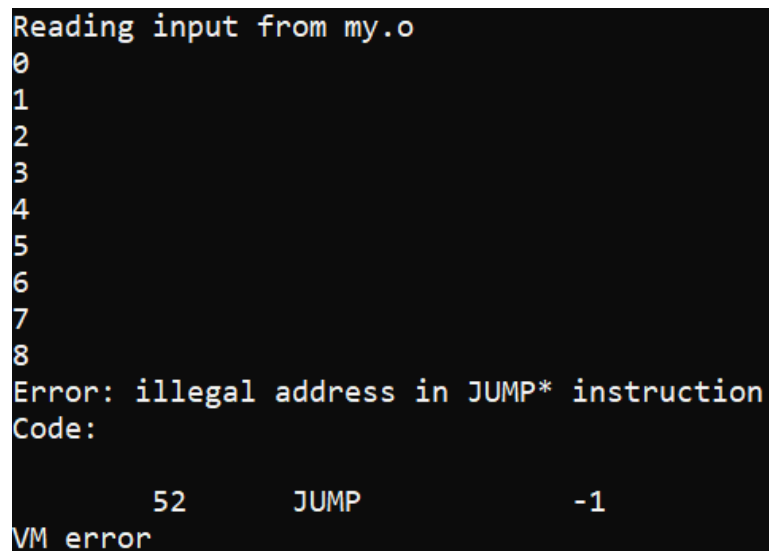
    i:=0;
    WHILE i < 10 DO
        arr[i] := i;
        i:=i+1
    OD;

    DELETE(arr);

    i:=0;
    WHILE i < 10 DO
        WRITE(arr[i]); /* при i=9 возникнет ошибка*/
        i:=i+1
    OD
END

```

Рис. 18 – Код программы на языке MiLan



```

Reading input from my.o
0
1
2
3
4
5
6
7
8
Error: illegal address in JUMP* instruction
Code:
          52      JUMP      -1
VM error

```

Рис. 19 – Выполнение программы виртуальной машиной MiLan

0:	PUSH	10	33:	JUMP	-1
1:	STORE	13	34:	PUSH	1
2:	PUSH	0	35:	SUB	
3:	STORE	14	36:	STORE	13
4:	LOAD	14	37:	PUSH	0
5:	PUSH	10	38:	LOAD	13
6:	COMPARE	2	39:	BSTORE	3
7:	JUMP_NO	28	40:	PUSH	0
8:	LOAD	14	41:	STORE	14
9:	DUP		42:	LOAD	14
10:	DUP		43:	PUSH	10
11:	LOAD	13	44:	COMPARE	2
12:	COMPARE	2	45:	JUMP_NO	64
13:	JUMP_YES	15	46:	LOAD	14
14:	JUMP	-1	47:	DUP	
15:	PUSH	0	48:	DUP	
16:	COMPARE	5	49:	LOAD	13
17:	JUMP_YES	19	50:	COMPARE	2
18:	JUMP	-1	51:	JUMP_YES	53
19:	STORE	0	52:	JUMP	-1
20:	LOAD	14	53:	PUSH	0
21:	LOAD	0	54:	COMPARE	5
22:	BSTORE	3	55:	JUMP_YES	57
23:	LOAD	14	56:	JUMP	-1
24:	PUSH	1	57:	BLOAD	3
25:	ADD		58:	PRINT	
26:	STORE	14	59:	LOAD	14
27:	JUMP	4	60:	PUSH	1
28:	LOAD	13	61:	ADD	
29:	DUP		62:	STORE	14
30:	PUSH	0	63:	JUMP	42
31:	COMPARE	3	64:	STOP	
32:	JUMP_YES	34			

Рис. 20 – Код программы, сгенерированный компилятором

3.6 Ошибки, отслеживаемые компилятором

На рисунках 21-25 приведены примеры работы компилятора при наличии ошибок в программе.

```
BEGIN
    i := 0;
    ARRAY i[8]
END
```

а) Код программы

```
Line 3: variable with such name already exists.
```

б) Реакция компилятора

Рис. 21 – Попытка создания массива с именем переменной

```
BEGIN
    ARRAY arr[5];
    i:=0;
    arr := i + j;
END
```

а) Код программы

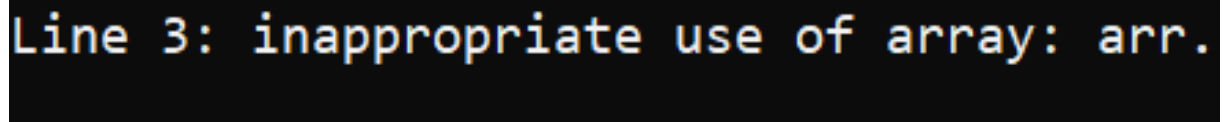
```
Line 4: Unknown array i
Line 4: Unknown array j
Line 5: statement expected.
```

б) Реакция компилятора

Рис. 22 – Использование переменных для операции над массивами

```
BEGIN
    ARRAY arr[5];
    i:= arr + arr[0]
END
```

а) Код программы



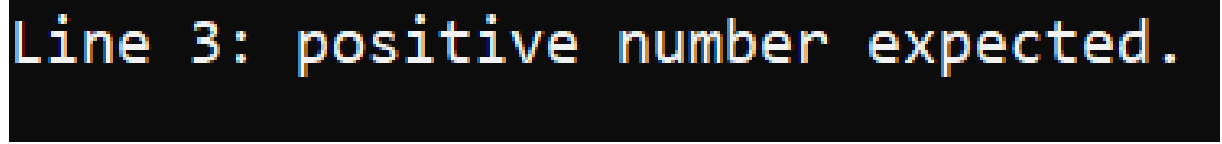
Line 3: inappropriate use of array: arr.

б) Реакция компилятора

Рис. 23 – Использование массива для операции над переменными

```
BEGIN
    i := 10;
    ARRAY arr[i]
END
```

а) Код программы



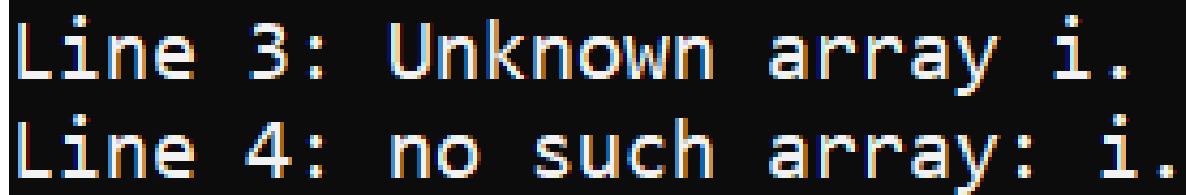
Line 3: positive number expected.

б) Реакция компилятора

Рис. 24 – Использование переменной в качестве размера массива

```
BEGIN
    i := 0;
    DELETE(i);
    i[0] := j
END
```

а) Код программы

A screenshot of a compiler's error output. It shows two lines of text in a monospaced font with a color scheme of blue, green, and red on a black background. The first line says "Line 3: Unknown array i." and the second line says "Line 4: no such array: i.".

Line 3: Unknown array i.
Line 4: no such array: i.

б) Реакция компилятора

Рис. 25 – Использование переменной в качестве массива

Заключение

В результате данной работы был дополнен язык MiLan и его компилятор. Дополненная версия компилятора позволяет осуществлять следующие операции над одномерными массивами:

- объявление массива;
- получение доступа к элементам массива;
- поэлементные арифметические операции над массивами;
- пересечение и объединение массивов;
- удаление последнего элемента массива.

Достоинством разработанного решения является проверка того, что указанный индекс массива не выходит за границы массива, т. е. неотрицателен и меньше размера массива. Проверка осуществляется при выполнении программы.

Недостатком разработанного решения является дублирование кода для виртуальной машины. Так, например, если в программе несколько раз используется пересечение массивов, то будут сгенерированы последовательности инструкции, отличающиеся только параметрами.

В связи с хорошей архитектурой компилятора MiLan, его дополнение было простым. Распознавание новых ключевых слов предполагает лишь добавления этих слов в массив ключевых слов. Для распознавания остальных лексем достаточно внести изменения в метод `nextToken`. При синтаксическом анализе достаточно добавить семантические операции в методы, соответствующие нетерминалам грамматики языка, а также создать методы для новых нетерминалов, если они добавляются при расширении грамматики.

Возможны следующие варианты масштабирования:

- добавление элемента в конец массива, если позволяет размер, указанный при объявлении массива;
- инициализация массива при объявлении;
- вывод массива на стандартный вывод без использования доступа к элементам массива;
- операция умножения или деления массива на число.

Список литературы

- [1] Э. Ф. Аллахвердиев, Д. А. Тимофеев — Компилятор SMiLan

Приложение

Файл Parser.cpp

```
int Parser::findVariable(const string& var)
{
    VarTable::iterator it = variables_.find(var);
    if (it == variables_.end()) {
        return -1;
    }
    else {
        return it->second;
    }
}

int Parser::findArray(const string& arr) {
    VarTable::iterator it = arrays_.find(arr);
    if (it == arrays_.end()) {
        return -1;
    }
    else {
        return it->second;
    }
}

int Parser::addArray(const string& arr, int offset)
{
    VarTable::iterator it = arrays_.find(arr);
    if (it == arrays_.end()) {
        arrays_[arr] = lastVar_;
        int oldLastVar_ = lastVar_;
        lastVar_ += offset;
        arraySizes_[arr] = lastVar_++;
        return oldLastVar_;
    }
    else {
        return -1;
    }
}

int Parser::findSize(const string& var)
{
    VarTable::iterator it = arraySizes_.find(var);
    if (it == arraySizes_.end()) {
        return -1;
    }
}
```

```

    else {
        return it->second;
    }
}

void Parser::arrExpression() {
    arrTerm();
    while (see(T_ADDOP)) {
        Arithmetic op = scanner_->getArithmeticValue();
        next();
        arrTerm();
        if (op == A_PLUS) {
            codegen_->emit(ADD);
        }
        else {
            codegen_->emit(SUB);
        }
    }
}

void Parser::arrTerm() {
    arrFactor();
    while (see(T_MULOP)) {
        Arithmetic op = scanner_->getArithmeticValue();
        next();
        arrFactor();
        if (op == A_MULTIPLY) {
            codegen_->emit(MULT);
        }
        else {
            codegen_->emit(DIV);
        }
    }
}

void Parser::arrFactor() {
    if (see(T_IDENTIFIER)) {
        int arrAddress = findArray(scanner_->getStringValue());
        if (arrAddress == -1) {
            std::ostringstream msg;
            msg << "Unknown array " << scanner_->getStringValue();
            reportError(msg.str());
        }
        else {
            codegen_->emit(LOAD, reserveAddress_ + 1);
        }
    }
}

```



```

        codegen_->emit(LOAD, findSize(scanner_>getStringValue()));
        codegen_->emit(COMPARE, 0);
        codegen_->emit(JUMP_YES, codegen_>getCurrentAddress() + 2);
        codegen_->emit(JUMP, -1);
        codegen_->emit(LOAD, reserveAddress_);
        codegen_->emit(BLOAD, arrAddress);
    }
    next();
}
else if (see(T_ADDOP) && scanner_>getArithmeticValue() == A_MINUS) {
    next();
    arrFactor();
    codegen_>emit(INVERT);
}
else if (match(T_LPAREN)) {
    arrExpression();
    mustBe(T_RPAREN);
}
else {
    reportError("Array expected.");
}
}

void Parser::clear()
{
    //index:=0
    codegen_>emit(PUSH, 0);
    codegen_>emit(STORE, 1);
    //while index < size do mem[index]:=0; index++; done
    codegen_>emit(PUSH, 0);
    codegen_>emit(LOAD, 1);
    codegen_>emit(BSTORE, lastVar_);
    codegen_>emit(LOAD, 1);
    codegen_>emit(PUSH, 1);
    codegen_>emit(ADD);
    codegen_>emit(DUP);
    codegen_>emit(STORE, 1);
    codegen_>emit(LOAD, 2);
    codegen_>emit(COMPARE, 2);
    codegen_>emit(JUMP_YES, codegen_>getCurrentAddress() - 10);
}

void Parser::copyToDest(int address, int size) {
    //if size <= arraySize then OK else JUMP -1
    codegen_>emit(LOAD, 2);

```

```

    codegen_->emit(LOAD, size);
    codegen_->emit(COMPARE, 4);
    codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() + 2);
    codegen_->emit(JUMP, -1);
//index:=0
    codegen_->emit(PUSH, 0);
    codegen_->emit(STORE, 1);
//while index < size do array[index]:=mem[index];index++; done
    codegen_->emit(LOAD, 1);
    codegen_->emit(BLOAD, lastVar_);
    codegen_->emit(LOAD, 1);
    codegen_->emit(BSTORE, address);
    codegen_->emit(LOAD, 1);
    codegen_->emit(PUSH, 1);
    codegen_->emit(ADD);
    codegen_->emit(DUP);
    codegen_->emit(STORE, 1);
    codegen_->emit(LOAD, 2);
    codegen_->emit(COMPARE, 2);
    codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() - 11);
}

```

```

void Parser::orCode(int arrAddress, int sizeAddress)
{
    codegen_->emit(LOAD, 0);
    codegen_->emit(BLOAD, arrAddress);
    codegen_->emit(LOAD, 2);
    codegen_->emit(PUSH, 0);
    codegen_->emit(COMPARE, 0);
    codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() + 18);
    codegen_->emit(PUSH, 0);
    codegen_->emit(STORE, 1);
    codegen_->emit(DUP);
    codegen_->emit(LOAD, 1);
    codegen_->emit(BLOAD, lastVar_);
    codegen_->emit(COMPARE, 0);
    codegen_->emit(JUMP_NO, codegen_->getCurrentAddress() + 3);
    codegen_->emit(POP);
    codegen_->emit(JUMP, codegen_->getCurrentAddress() + 15);
    codegen_->emit(LOAD, 1);
    codegen_->emit(PUSH, 1);
    codegen_->emit(ADD);
    codegen_->emit(DUP);
    codegen_->emit(STORE, 1);
    codegen_->emit(LOAD, 2);
}

```

```

codegen_->emit(COMPARE, 2);
codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() - 14);
codegen_->emit(LOAD, 2);
codegen_->emit(BSTORE, lastVar_);
codegen_->emit(LOAD, 2);
codegen_->emit(PUSH, 1);
codegen_->emit(ADD);
codegen_->emit(STORE, 2);
codegen_->emit(LOAD, 0);
codegen_->emit(PUSH, 1);
codegen_->emit(ADD);
codegen_->emit(DUP);
codegen_->emit(STORE, 0);
codegen_->emit(LOAD, sizeAddress);
codegen_->emit(COMPARE, 2);
codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() - 36);
}

```

```

void Parser::andCode(int arrAddress1, int sizeAddress1,
                    int arrAddress2, int sizeAddress2) {
    codegen_->emit(LOAD, 0);
    codegen_->emit(BLOAD, arrAddress1);
    codegen_->emit(PUSH, 0);
    codegen_->emit(STORE, 1);
    codegen_->emit(DUP);
    codegen_->emit(LOAD, 1);
    codegen_->emit(BLOAD, arrAddress2);
    codegen_->emit(COMPARE, 0);
    codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() + 11);
    codegen_->emit(LOAD, 1);
    codegen_->emit(PUSH, 1);
    codegen_->emit(ADD);
    codegen_->emit(DUP);
    codegen_->emit(STORE, 1);
    codegen_->emit(LOAD, sizeAddress2);
    codegen_->emit(COMPARE, 2);
    codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() - 12);
    codegen_->emit(POP);
    codegen_->emit(JUMP, codegen_->getCurrentAddress() + 28);
    codegen_->emit(LOAD, 2);
    codegen_->emit(PUSH, 0);
    codegen_->emit(COMPARE, 0);
    codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() + 18);
    codegen_->emit(PUSH, 0);
    codegen_->emit(STORE, 1);
}

```

```

codegen_->emit(DUP);
codegen_->emit(LOAD, 1);
codegen_->emit(BLOAD, lastVar_);
codegen_->emit(COMPARE, 0);
codegen_->emit(JUMP_NO, codegen_->getCurrentAddress() + 3);
codegen_->emit(POP);
codegen_->emit(JUMP, codegen_->getCurrentAddress() + 15);
codegen_->emit(LOAD, 1);
codegen_->emit(PUSH, 1);
codegen_->emit(ADD);
codegen_->emit(DUP);
codegen_->emit(STORE, 1);
codegen_->emit(LOAD, 2);
codegen_->emit(COMPARE, 2);
codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() - 14);
codegen_->emit(LOAD, 2);
codegen_->emit(BSTORE, lastVar_);
codegen_->emit(LOAD, 2);
codegen_->emit(PUSH, 1);
codegen_->emit(ADD);
codegen_->emit(STORE, 2);
codegen_->emit(LOAD, 0);
codegen_->emit(PUSH, 1);
codegen_->emit(ADD);
codegen_->emit(DUP);
codegen_->emit(STORE, 0);
codegen_->emit(LOAD, sizeAddress1);
codegen_->emit(COMPARE, 2);
codegen_->emit(JUMP_YES, codegen_->getCurrentAddress() - 53);
}

```