# Генератор кода для языка Паскаль

## Постановка задачи

Требуется написать генератор кода для разработанного анализатора языка Паскаль. Генерацию можно производить в один из следующих языков: Ассемблер, LLVM, CLR, JVM-байт код.

В качестве языка генерации был выбран язык Ассемблер – NASM.

## Проектирование

Поскольку все процессы компиляции (анализ синтаксиса, семантики) работают параллельно, то и генератор кода тоже должен работать одновременно со всеми процессами.

Был создан класс, который реализует возможности генерации кода – CodeGenerator. Основная задача этого класса – сформировать ассемблерный код для вычисления выражений, работы с переменными и переходами в программе, а также запись готовой программы в файл. Интерфейс этого класса представлен на рисунке 2.1.

class CodeGenerator

{public:

CodeGenerator(const char\* path);

~CodeGenerator();

//создание переменных

void generateScope(const Scope\* scope);

//вычисление выражения через постфиксную запись

void pushVariable(const std::string& variable, bool negate);

void pushValue(long long value);

void ExecuteBinaryOperation(PascalKeyword operation);

void SaveToVariable(const std::string& variable);

//ввод/вывод

void InputIntVariables(const std::vector<Data\*>& variables);

void OutputIntVariables(const std::vector<Data\*>& variables);

//переходы в программе

void ReserveLabel(std::string& label);

int createLabel();

void createLabel(const std::string& label);

void Jump(const std::string& label);

void JumpWithCondition(int then\_label\_id, int else\_label\_id);

private:

const char\* path;

std::stringstream data\_section;

std::stringstream bss\_section;

std::stringstream code\_section;

int last\_label\_id = 1;

};

Рисунок 2.1 – Интерфейс класса CodeGenerator

## Реализация

Программа на NASM состоит из трех секций:

* Data – инициализированные данные
* Bss – неинициализированные данные
* Text – код программы

Для удобства заполнения, все три секции заполняется параллельно в оперативной памяти в виде трех отдельных строк (stringstream, чтобы избежать конкатенации строк). Когда код сохраняется в файл, то все три строки последовательно записываются в файл.

При генерации кода встает проблема – в программе на ассемблере нет вложеностей видимости и все переменные находятся на одном уровне, а имена переменных могут повторяться в разных областях видимости. Вопрос: как записать переменные в ассемблер чтобы они не конфликтовали друг с другом? Было принято решение- дописывать к названию переменной ее уровень вложенности.

Самым важным при генерации – генерация вычислений выражений. Чтобы стало возможным выполнение самых сложных выражений – было принято решение переводить выражение в постфиксную запись на лету во время генерации. Притом стек операндов находится в стеке процессора, а стек операций – в классе компилятора – opstack

Алгоритм перевода выражения из инфиксной в постфиксную:

1. Если входящий элемент – операнд (число или переменная), то положить его в стек на процессоре
2. Если входящий элемент оператор(+, -, \*, /, div, mod, =, <>, ….),
   1. Если входящий оператор имеет более высокий приоритет чем вершина стека, то поместить оператор на вершину стека opstack
   2. Иначе
      1. пока prior(входящий\_оператор) <= prior(opstack.top())
         1. Достать оператор с вершины стека и выполнить его
      2. Положить входящий\_оператор на вершину стека opstack
3. Если входящий оператор – (, то положить ее на вершину стека
4. Если входящий оператор - ), то достаем операторы из стека opstack и выполняем их пока на вершине не окажется (
5. В конце выражения достаем операторы из стека opstack и выполняем их, пока стек не опустеет

Также возникла проблема с расстановкой меток для операций (для if и while). Проблема заключается в том, что у каждого условия должны быть свои уникальные метки, чтобы не было конфликтов между ними. Было принято решение ввести счетчик меток и каждой новой добавленной метке – приписывать число из счетчика, так решилась проблема уникальности меток.

Для операций ввода/вывода существует набор макросов io.inc, куда включены макросы считывания целых чисел, строк и символов.

## Тестирование

Для тестирования программы на ассемблере была использована IDE SASM, где есть встроенный отладчик программы.

### Тест 1

Тест 1 посвящен основным инструкциям языка: вычислению выражения, присваиванию, циклу while и условию if.

Код программы на языке паскаль приведен на рисунке 4.1.

program test;

var a,b,c: Integer;

d,e,f,g,h,i: Boolean;

begin

readln(a, b);

d := a = b;

e := a <> b;

f := a < b;

g := a > b;

h := a <= b;

i := a >= b;

writeln(d,e);

writeln(f,g);

writeln(h,i);

c := 0;

if (a < b) or True then

c := (a + b) / (a - b) / (a - b) + a + b;

writeln(c);

while (a < b) do

begin

a := a + 1;

writeln(a);

end;

end.

Рисунок 4.1 – Код программы, тестирующей все основные инструкции языка Паскаль

Вывод программы зависит от входных данных, которые подаются на вход. Поэтому протестируем программу на различных входных данных. Результаты тестирования представлены в таблице 4.1.

Таблица 4.1 – Результаты тестирования программы

|  |  |
| --- | --- |
| Ввод | Вывод |
| 3 5 | 0 1 //a = b, a <> b  1 0 //a < b, a >b  1 0 // a <=b, a >= b  10 // c  4 //while a < b  5 |
| -15 5 | 0 1  1 0  1 0  -10  -14  -13  -12  -11  -10  -9  -8  -7  -6  -5  -4  -3  -2  -1  0  1  2  3  4  5 |
| 6 5 | 0 1  0 1  0 1  22 |
| 5 5 | 1 0  0 0  1 1  Division by zero |