**Санкт-Петербургский государственный политехнический университет**

**Институт Информационных Технологий и Управления**

**Кафедра распределенных вычислений и компьютерных сетей**

**Курсовая работа**

на тему: Расширение языка Milan для построения транслятора  
по дисциплине: Теория автоматов и формальных языков

Выполнил студент

гр. 23507/1 <подпись> В.Б. Борисов

Руководитель

зав. кафедры <подпись> Ю.Г. Карпов

Санкт-Петербург

2015

СОДЕРЖАНИЕ

[Введение 3](#_Toc417910269)

[Основная часть 4](#_Toc417910270)

[Постановка задачи 4](#_Toc417910271)

[Описание реализации 4](#_Toc417910272)

[Тестирование 5](#_Toc417910273)

[Заключение 8](#_Toc417910274)

[Список литературы 9](#_Toc417910275)

Введение

Язык Милан — учебный язык программирования. Программа на Милане представляет собой последовательность операторов, заключенных между ключевыми словами «begin» и «end». Операторы отделяются друг от друга точкой с запятой. После последнего оператора в блоке точка с запятой не ставится. Компилятор CMilan не учитывает регистр символов в именах переменных и ключевых словах.

В базовую версию языка Милан входят следующие конструкции: константы, идентификаторы, арифметические операции над целыми числами, операторы чтения чисел со стандартного ввода и печати чисел на стандартный вывод, оператор присваивания, условный оператор, оператор цикла с предусловием.

Программа может содержать комментарии, которые могут быть многострочными. Комментарий начинается символами «/\*» и заканчивается символами «\*/». Вложенные комментарии не допускаются.

Компилятор CMilan включает три компонента:

1. лексический анализатор;

2. синтаксический анализатор;

3. генератор команд виртуальной машины Милана.

Лексический анализатор посимвольно читает из входного потока текст программы и преобразует группы символов в лексемы (терминальные символы грамматики языка Милан). При этом он 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 (например, оператор присваивания), а также определяет значения числовых констант и имена переменных. Эти значения становятся значениями атрибутов, связанных с лексемами.

Синтаксический анализатор читает сформированную лексическим анализатором последовательность лексем и проверяет ее соответствие грамматике языка Милан. Для этого используется метод рекурсивного спуска. Если в процессе синтаксического анализа обнаруживается ошибка, анализатор формирует сообщение об ошибке, включающее сведения об ошибке и номер строки, в которой она возникла. В процессе анализа программы синтаксический анализатор генерирует набор машинных команд, соответствующие каждой конструкции языка.

Генератор кода представляет собой служебный компонент, ответственный за формирование внешнего представления генерируемого кода. Генератор поддерживает буфер команд и предоставляет синтаксическому анализатору набор функций, позволяющий записать указанную команду по определенному адресу. После того, как синтаксический анализатор заканчивает формирование программы, генератор кода используется для печати на стандартный вывод отсортированной по возрастанию адресов последовательности инструкций. Использование генератора кода несколько упрощает устройство компилятора, поскольку синтаксический анализатор может генерировать команды не в порядке их следования в программе, а по мере получения всей необходимой информации для их формирования. Это особенно важно при трансляции таких инструкций, как условные операторы или циклы.

# Основная часть

# Постановка задачи

Изучить компилятор языка Milan и добавить в него поддержку следующие языковые конструкции:

1. Цикл с постусловием:

**repeat** <список операторов> **until** <условие>

# Описание реализации

При реализации данного задания был выполнен анализ исходных кодов компилятора cmilan. Как результат этого грамматика языка Milan была расширена с помощью циклов с постусловием, т.е. repeat-until:

Первоначально нужно дополнить список лексем и ключевых слов

**Project cmilan -> scanner.h -> enum Token**

T\_REPEAT,

T\_UNTIL,

**Project cmilan -> scanner.h -> explicit Scanner**

keywords\_["repeat"] = T\_REPEAT;

keywords\_["until"] = T\_UNTIL;

**Project cmilan -> parser.cpp -> void Parser::statement()**

else if (match(T\_REPEAT)){

//запоминаем адрес начала проверки условия.   
int repeatAddress = codegen\_->getCurrentAddress();

statementList();

mustBe(T\_UNTIL);

relation();

codegen\_->emit(JUMP\_YES, repeatAddress);

}

# Тестирование

Для тестирования выполненной работы был создан файл: ***repeat-until.mil.***

Файл ***repeat-until.mil*** содержит код:

BEGIN

x:=0;

REPEAT

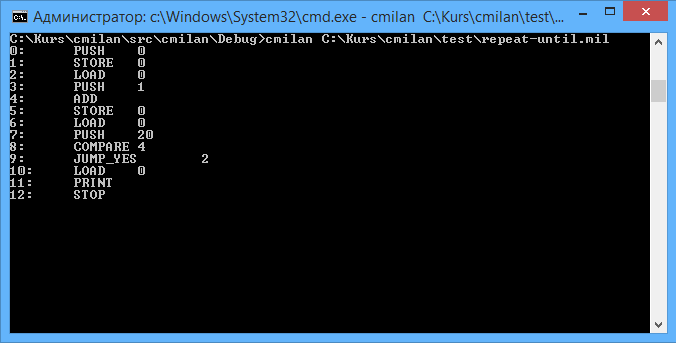
x:=x+1

UNTIL x <= 20;

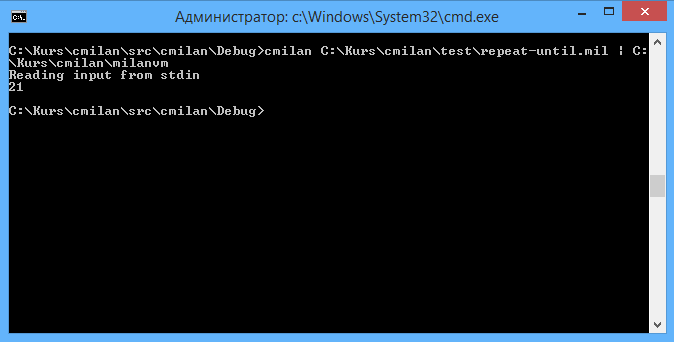
WRITE(x)

END

Для запуска исполняемого файла ***repeat-until.mil*** используем командную строку (консоль) windows и получаем следующий машинный код:



Теперь одновременно с этим полученным кодом, запустим файл «milanvm» для того чтобы получить результат.



# Заключение

Мы научились работать с языком Милан, впоследствии чего он был расширен полезными операторами, которые играют ключевую роль в программировании, а именно циклами. Были проведены различные тесты, которые показали корректную работу и выявили ожидаемые результаты. Поставленная задача решена. Полученные результаты следует считать успешными.

# Список литературы

[1] А. Ахо, М. Лам, Р. Сети, Дж. Ульман, Компиляторы: принципы, технологии и инструментарий, 2-е изд. М.: Вильямс, 2011.

[2] Ю.Г. Карпов, Теория и технология программирования. Основы построения трансляторов. СПб.: БХВ-Петербург, 2005.