Методические указания к выполнению курсового проекта
по дисциплине «Теория языков программирования и методы трансляции»
Группа КИ10-11

1. Введение

В рамках часов, отведенных в учебном плане специальности «Программное обеспечение вычислительной техники и автоматизированных систем» на самостоятельную работу (общий объем – 72 часа) студентов по дисциплине «Теория языков программирования и методы трансляции», предусмотрено выполнение курсового проекта (общий объем – 72 часа) в соответствии с прилагаемым типовым заданием. Темы курсовых проектов даются на основе изложенного теоретического курса и выполненных лабораторных работ и практических занятий. Они направлены, главным образом, на закрепление полученных знаний, умений и владений (компетенций). Для выполнения курсовых проектов студенты могут объединяться в подгруппы. В отдельных случаях возможна выдача нетиповой темы. Также приветствуются инициативные студенческие проекты из области трансляции.

Темы и кандидатуры руководителей курсовых проектов обсуждаются на очередном заседании кафедры «Информатика» и утверждаются распоряжением заведующего кафедрой. Таким образом, до 28 февраля 2014 года (включительно) названия тем и кандидатуры руководителей курсовых проектов должны быть согласованы, и соответствующая информация передана на ведущую кафедру.

2. Постановка задачи

Необходимо разработать:

2.1. Язык императивного программирования высокого уровня с блочной структурой, включающий арифметические выражения и подвыражения в скобках с операциями сложения, вычитания, умножения, деления, унарными плюсом и/или минусом и другие; логические выражения и подвыражения в скобках (логические И, ИЛИ, НЕ и другие).

Основные конструкции языка:

- присваивание в виде операций либо операторов;
- условия с необязательной частью *else*;
- циклы с параметром, с предусловием и постусловием;
- множественный выбор;
- безусловные переходы (*goto*, *break* и *continue*);
- операции сравнения (<, >, =, !=, <=, >=);
- функции и/или процедуры с операторами вызова и возврата из них (*return*);
- простые средства ввода-вывода в виде операторов или встроенных функций;

- минимальные средства комментирования кода;
- другие по желанию разработчиков.

Язык должен поддерживать данные встроенных (примитивных) и конструируемых (пользовательских) типов. Встроенные типы – целые и вещественные числа, литерные, логические, строковые и др. Количество примитивных типов – не менее трех. Две или более разновидности пользовательских типов – классы/объекты, структуры, объединения, кортежи и другие со средствами описания структуры типа и операторами обращения к составным частям объектов пользовательских типов (например, «точка», как в языках C/C++).

Язык должен поддерживать операторы объявления программных объектов, включая скаляры и массивы встроенных и пользовательских типов.

Для этого языка необходимо составить описание, грамматику и примеры использования основных конструкций.

2.2. Разработать язык низкого уровня – набор элементарных команд, необходимых для работы некоторой виртуальной вычислительной машины. Этот язык должен работать с памятью регистрового (регистры общего и/или специального назначения), а также стекового, статического и по необходимости других типов.

В качестве дополнения и/или альтернативы можно рассматривать набор операций из системы команд реальной вычислительной машины.

2.3. Разработать программу – интерпретатор команд языка низкого уровня. Она должна принимать на вход файл с программой на разработанном языке низкого уровня (пп.2.2) и выполнять его инструкции. Требования и подход к разработке интерпретатора такие же, что и для компилятора (см. далее в пп.2.4).

Запуск интерпретатора осуществляется посредством командной строки с передачей всех необходимых аргументов. В числе обязательных – аргумент для вывода справки об использовании интерпретатора.

Данный пункт может не выполняться, если в качестве языка низкого уровня рассматривается исключительно система команд реальной вычислительной машины (см. выше пп.2.2).

2.4. Разработать программу – оптимизирующий компилятор императивного языка программирования высокого уровня на язык низкого уровня. На вход компилятора должен подаваться файл с текстом программы на разработанном языке высокого уровня (пп.2.1). Выход компилятора как минимум в одном из трех вариантах:

2.4.1) файл/набор файлов с эквивалентным «текстом» на разработанном языке низкого уровня (см. выше пп.2.2);

2.4.2) исполняемый файл в формате операционной системы (ELF, PE32/PE32+, COM/EXE и др.) для запуска на целевой вычислительной системе;

2.4.3) машинный и ассемблерный код для любых микропроцессорных архитектур, отличных от целевой системы.

Реализация одного из подпунктов 2.4.1-2.4.3 обязательна.

Компилятор должен осуществлять полный контроль лексики, синтаксиса и семантики исходной программы с выдачей обнаруженных ошибок как можно ближе к месту их возникновения. Структура компилятора в целом должна соответствовать структуре разработанной (пп.2.1) грамматики языка. Для проектирования отдельных компонент компилятора рекомендуется воспользоваться автоматизированными средствами типа flex/bison или аналогичных им.

Компилятор должен выполнять не менее двух разных видов локальных оптимизаций кода. Опциальной является поддержка глобальной и межпроцедурной оптимизаций. Пользователю должна быть предоставлена возможность включать и отключать с помощью параметров, передаваемых компилятору, каждый поддерживаемый вид оптимизации либо все виды сразу.

Запуск компилятора осуществляется посредством командной строки с передачей всех необходимых аргументов. В числе обязательных – аргумент для вывода справки об использовании компилятора.

2.5. Написать отчет о проделанной работе согласно требованиям, предъявляемым в СФУ к письменным студенческим работам[1].

3. Содержание отчета

В отчете должны быть обязательно отражены следующие вопросы:

3.1) *Аннотация* – реферативное описание дальнейшей информации в отчете на русском и английском языках.

3.2) *Введение* – название языка высокого уровня, его возможности, достоинства и недостатки, требования к аппаратной части, операционному окружению и другим параметрам среды, в которой работают компилятор и интерпретатор, краткое руководство пользователя по компилятору и интерпретатору.

[1] См. http://www.sfu-kras.ru/node/8127.

3.3) *Описание языка*

3.3а) Лексические соглашения – алфавит, ключевые слова, комментарии, идентификаторы с использованием нотации расширенных регулярных выражений.

3.3б) Синтаксис языка – перечислить все конструкции, допустимые в данной версии языка, правила (в любой подходящей нотации: форма Бэкуса-Наура, расширенная форма Бэкуса-Наура, синтаксическая диаграмма Вирта) и семантические ограничения на их применение, примеры в виде фрагментов кода;

3.3в) Грамматика языка – полное описание грамматики языка в виде форм Бэкуса-Наура, либо расширенных форм Бэкуса-Наура, либо синтаксических диаграмм Вирта, либо в виде ссылки на соответствующее приложение к отчету с входной спецификацией для инструментария, если таковой применялся для реализации блока синтаксического анализа;

3.3г) Полный перечень отслеживаемых лексических, синтаксических и семантических ошибок на русском языке (если компилятор выдает сообщения на другом языке, то привести также их).

3.4) *Описание компилятора* – архитектура программы, замечания по ее реализации, руководство пользователя.

3.5) *Описание языка низкого уровня* – перечень команд и их описание (синтаксис). Дается полный перечень и описание при необходимости, либо делаются ссылки на источники информации, где это уже сделано.

3.6) *Описание интерпретатора*. Дается полное руководство пользователя при необходимости, либо делаются ссылки на источники информации, где это уже сделано.

3.7) *Список использованных источников*. В обязательном порядке согласно СТО СФУ и ГОСТ Р 7.0.5 – 2008.

3.8) *Приложение* – листинги наиболее интересных частей компилятора и примеров его использования, а также несколько полных примеров программ, входные спецификации для инструментальных средств реализации компилятора. Должны приводиться примеры как корректных программ, так и программ, содержащих ошибки. В последнем случае необходимо продемонстрировать, как реагирует компилятор на те или иные ошибки.

4. Система оценивания

При выполнении этих условий курсовой проект оценивается на оценку «удовлетворительно». Также применяется система «поощрений» и «штрафов».

Поощрения (выполнение хотя бы одного из следующих требований увеличивает оценку на 1 балл):

– промежуточное представление программы на входном языке компилятором в виде конструкций на языке XML;

– наличие в языке высокого уровня и поддержки компилятором возможностей по раздельной компиляции исходных текстов;

– поддержка многоцелевой кодогенерации (см. выше п.2.4);

– использование компиляторной инфраструктуры промышленного уровня (LLVM, Parrot, GCC и другие);

– поддержка глобальных и межпроцедурных оптимизаций кода.

Штрафы (невыполнение одного из следующих требований ведет к снижению оценки на 1 балл):

– неполнокомплектная пояснительная записка;

– некорректная диагностика ошибок;

– некорректная работа компилятора;

– невыполнение графика учебного процесса (в том числе защита курсового проектов по окончании семестра).

5. Альтернативные и нетиповые темы

Альтернативные и нетиповые темы курсовых проектов **согласовываются** с преподавателем **в обязательном порядке**. В качестве таковых, в частности, могут разрабатываться компиляторы существующих и вновь создаваемых языков функционального и логического программирования, а также предметно-ориентированные языки программирования. Можно так же представлять к защите интерпретаторы языков, считающихся преимущественно компилируемыми.

7. Дополнительные источники информации

- Executable and Linkable Format (ELF) - http://www.skyfree.org/linux/references/ELF_Format.pdf
- Microsoft PE and COFF Specification - http://msdn.microsoft.com/en-us/windows/hardware/gg463119.aspx
- EXE Format - http://www.delorie.com/djgpp/doc/exe/

- Hexadecimal Object File Format Specification - http://microsym.com/editor/assets/intelhex.pdf
- flex: The Fast Lexical Analyzer - http://flex.sourceforge.net/
- Bison - GNU parser generator - http://www.gnu.org/software/bison/
- BYACC – Berkeley Yacc – generate LALR(1) parsers - http://invisible-island.net/byacc/byacc.html
- The Compiler Generator Coco/R - http://www.ssw.uni-linz.ac.at/Research/Projects/Coco/
- ANTLR - http://www.antlr.org/
- The LLVM Compiler Infrastructure - http://www.llvm.org/
- From Source to Binary: The Inner Workings of GCC - http://www.redhat.com/magazine/002dec04/features/gcc/
- Parrot VM - http://www.parrot.org/

Составитель: А.С.Кузнецов (askuznetsov@sfu-kras.ru).

Красноярск, 18.02.2014