National Institute of Technology Karnataka- Surathkal.
Department of Computer Science & Engineering

Tentative Course Details: December 2014 – May 2015

Course:

No : CO 350
Title : Compiler Design
Credits : (3-1-0) 4

Prerequisites:

C0 204: Data Structures
CO 201: Computer Organisation
CO 254: Theory of Computation (desirable).

Instructor:

Name : P.Santhi Thilagam
Office : I floor, Computer Science & Engineering Building
Office Hours : Afternoons or by appointment
Email : santhi@nitk.ac.in

Objectives:

At the end of this course on Compiler Construction, the student should be able to

- Understand the two models of language processing, viz. compilers and interpreters, and realize their strengths and limitations.
- Understand the influence of various features of programming languages and machine architectures on the efficiency of language translation.
- Appreciate the role of lexical analysis in compilation, understand the theory behind design of lexical analysers and lexical analyser generators, and be able to use Lex to generate lexical analysers.
- Be proficient in writing grammars to specify syntax, understand the theories behind different parsing strategies - their strengths and limitations, understand how the generation of parsers can be automated, and be able to use Yacc to generate parsers.
- Understand the error detection capabilities of a compiler and how to report them in a user friendly manner.
- Understand the need for various static semantic analyses such as declaration processing, type checking, and intermediate code generation, and be able to perform such analysis in a syntax directed fashion through the use of attribute grammars.

- Understand the issues involved in allocation of memory to data objects, and the relation of such issues with higher level language features such as data types, scope and nesting rules, and recursion.
- Understand the key issues in the generation of efficient code for a given architecture.
- Understand the role played by code optimization.

References:

1. Aho A.V, Sethi R, and Ullman J.D. Compilers: Principles, Techniques, and Tools. Addison-Wesley, 1986.
2. Appel A.W., and Palsberg J. Modern Compiler Implementation in Java. Cambridge University Press, 2002.

Grading: Marks Distribution:

Mid-Term Exam	: 25%
Assignments & Homework	: 10%
Tests / Quizzes	: 25%
Final Exam	: 40%

Course Schedule:

	Topic
Week 01	Introduction to Programming Language Translation
Week 02	Lexical Analysis: Specification and Recognition of tokens
Week 03	Syntax Analysis: Predictive parsers
Week 04	Syntax Analysis: LR(0) parsers
Week 05	Syntax Analysis: SLR and LR(1) parsers
Week 06	Syntax Analysis: SLR and LR(1) parsers
Week 07	Syntax Analysis: LALR(1) parsers and Review
Week 08	Midterm Exam
Week 09	Semantic Analysis: Type Expressions, Type systems
Week 10	Semantic Analysis: Type checking
Week 11	Semantic Analysis: Symbol Tables, Error Recovery
Week 10	Intermediate Code Generation: Intermediate Representation
Week 11	Intermediate Code Generation: Syntax Directed Translation
Week 12	Intermediate Code Generation: Syntax Directed Translation
Week 13	Intermediate Code Generation: Syntax Directed Translation
Week 15	Runtime Environments: Storage organization, Activation records
Week 16	Machine Code Generation
Week 17	Introduction to Code Optimization
Week 18	Final Exam

Grading: Policies:

- Grading will be relative.
- There will be 3 tests & quizzes of 10, 10, 05 and marks respectively.
- Absence for exams/quizzes without prior written permission from the instructor will be equivalent to zero marks in the corresponding exam/quiz.
- There will be no makeup exams except in case of genuine reasons. In the event of such exceptional cases, the student must discuss the matter with the instructor and must get written permission before the date of exam. Normally the corresponding weight of the exam will be added to other exams.

Standard of Conduct:

Each student is expected to adhere to high standards of ethical conduct, especially those related to cheating and plagiarism. Any submitted work MUST BE an individual effort. Any academic dishonesty will result in zero marks in the corresponding exam or quiz and will be reported to the department board (DUGC) for record keeping.

National Institute of Technology Karnataka- Surathkal.
Department of Computer Science & Engineering

Tentative Course Details : December 2014 – May 2015

Course:

No : CO 351
Title : Compiler Design LAB
Credits : (0-0-3) 2

Prerequisites:

CO 203: Unix programming

C0 206: Data Structures LAB

Instructor:

Name : P.Santhi Thilagam
Office : I floor, Computer Science & Engineering Building
Office Hours : Afternoons or by appointment
Email : santhi@nitk.ac.in

Objectives:

The students will undertake a sequence of experiments aimed at the design and implementation of the various phases of a compiler for the C programming language. This subset includes a sufficiently rich collection of data types and control structures.

General Requirements :

Team work

- Work in pairs, i.e., teams of no more than two students. (self-selected; team name list due on January 10).
- In this project, tasks should not be divided between the team members - the pair should tackle problems together, on the same computer, at the same time.

On grading

- For each phase, but the last, the marks are distributed as follows:
  - Report, 30% of the marks
  - Correctness and completeness, 60%
  - Program Documentation, 10%
- For the last phase the marks are distributed as follows:
  - Report, 10% of the marks
  - Correctness and completeness, 50%
  - Program Documentation, 10%
  - Presentation, 30%

Project 1: Scanner for C- language

Assigned: January 10, 2015

Due Date for Test Cases: January 15, 2015

Due Date for Report and Code: January 20, 2015

Use LEX/Flex to implement a lexical analyzer for C language. Your lexical analyzer should support nested comments, and return appropriate error messages that are as meaningfull as possible, such as comments and strings that don't end until the end of the file, etc. Your documentation should explain how did you deal with comments, strings, errors etc.

- Study the grammar of the source language, and identify the lexical components of the language specifications.
- Study the input format for Flex and clearly understand how tokens are described.
- Identify the tokens to be returned by the scanner..
- Develop Flex script for the tokens identified above. Be careful with the definition of constants, the string literal and comments.
- Design the symbol table and the constants table. Use hash organization for both.
- Design routines to produce the scanner output in the desired format as described in `Guidelines on how to present the results'.
- Generate the scanner.

Guidelines on how to present the results

- When the token is identified, print its details. Thus your scanner should produce the list of tokens for the source program input to it.
- At the end of the program, neatly print the symbol and constants tables.
- Examples should include some programs containing lexical errors. The error should be properly identified and indicated.

You are also required to submit test cases for the scanner.

Project 2: Parser for C- language

Assigned: January 25, 2015

Due Date for Test Cases: February 10, 2015

Due Date for Report and Code: March 1, 2015

The objective of this assignment is familiarization with parser generators and development of a LALR parser for the source language.

Use YACC/Bison to implement a parser for C language. Your parser should have as few shift-reduce conflicts as possible, and no reduce-reduce conflics.

Explain in your documentation what shift-reduce conflics have been left unresolved (if any) and why aren't they harmful.

For this assignement you will use the parser generator YACC/Bison and the grammar specification given above.

- Study the format of a typical yacc script.
- Enter the grammar in the yacc format, integrate with the scanner generated in the previous stage, and generate the parser.
- Test the generated parser on sample programs.
- Augment the grammar by adding production for the following features. Add enough production to ensure that the feature can be meaningfully used. These are being added just for this stage to test your grammar writing ability. They need not be implemented in the final compiler.
- Generate the parser once again for the augmented grammar.

Some changes are required in the structure of your compiler when you interface your scanner with the parser. The symbol table routines should now become a part of the parser. The parser makes a call to the scanner whenever it needs to look at the next token. After returning from the scanner, the parser must update the appropriate table. For example an identifier should be entered in the symbol table, a constant in the constants' table, etc.

In case of an error, abort after reporting the line number where the error has occured. You need not recover from errors.
You are also required to submit test cases for the parser.

Project 4: Semantic checker for C- language

Assigned: March 25, 2015

Due Date for Test Cases: April 5, 2015

Due Date for Report and Code: April 20, 2015

PART I:
Add a semantic checker to your compiler. It should do the checks on the list in class (repeated below) as well as build a symbol table for use during checking and in code generation. Use good style (including meaningful variable names) in your coding and be liberal with comments. You may use either C, C++.
Your parser should be written so that input is the abstract syntax tree from project 3 and output includes any error messages or a symbol table containing all functions and variables in the program. Your symbol table should include an option to print out its contents (symbol, data type, line number where declared). Error messages should include the line number where they occurred.

This project is divided into two parts: the symbol table and implementation of type checking.

Symbol Table Implementation and Manipulation:

Among the best ways to implement the symbol table uses a stack. Each entry in the symbol table will contain information related to a symbol representing a variable, a procedure, or a parameter. The exact set of information stored in the symbol table entry will depend on the class of the symbol described by the entry.

The following fields should be included in a symbol table entry:
1. Symbol Name
2. Type
3. Class
4. Boundaries List
5. Array Dimensions
6. Parameters List
7. Procedure Definition Flag
8. Nesting Level

Semantic errors

- Variable declarations
    - Duplicate declaration of ID
    - Array size less than 1
- Function declarations
    - Duplicate declaration of ID
    - Params of type void
    - No functions defined
    - Function main is not last function
    - Int function has void return
    - Void function has int return
- Call expressions
    - Argument expression not of type integer
    - ID is not a function
    - ID undeclared in current scope
    - Type of paramaters does not match type of argument
    - Number of parameters and arguments do not match
- Select/while statements
    - Expression in test not of type int
- Expressions
    - ID undeclared in current scope

    - Array ID has no subscript
    - Single variable ID or function has subscript
    - Expression in subscript not of type int
    - Expression on rhs of assigment and in arithmetic ops must be int
    - Expression on lhs of assignment not single variable
- Return statement
    - Expression has type void, function or array

  Your project report may be submitted on paper (typed). At minimum, the report should include:
    - an overview explaining your code including info about what doesn't work (if anything). The overview should describe the general flow of your code.
    - listings of the code for your semantic checker
    - a description of the checks performed
    - a description of the symbol table
    - a discussion of any assumptions you made beyond what is in the basic language description.
    - a report on the test cases you ran along with the results. Report any failures and successes. For your test cases, you should describe what was being tested.

PART II:

The objective of this assignment is to perform semantic analysis such as type and scope analysis and declaration processing, and integrate such analyses with the parser. Add semantic actions to your parser to produce abstract syntax for the C language.

Notes on how to conduct the experiments:

- Study the way attributes are assigned to a nonterminal in YACCC
- Go back to the original grammar.
- For declaration processing, do the following:
    - Decide on the attribute information (i.e. type, dimension, length etc) for an identifier. Accordingly design the symbol table entry formats/conventions for storing information regarding variable identifiers, type identifiers and function identifiers.
    - Write semantic rules for processing of all declarations of data in a single block of the source program, and recording relevant information in the symbol table.
    - Perform storage allocation for data symbols (this will be an offset value in the activation record.
- For type analysis, write semantic rules to do the following:
    - For a use of a symbol, check for the declaration that the symbol is bound to using Pascal like scope rules. If the symbol is not bound

to any declaration, declare an error. The symbol table should be rich enough to implement the scope rules.
    - The symbol table entry for the declaration will provide all the attributes of the symbol.
    - Use this information to check that the use of a symbol is type compatible with its context.
    - The notion of type compatibility to be used is structural equivalence.
    - Perform type conversion if necessary.
- Detect and report semantic errors. You should detect as many semantic errors as possible in a single compilation of the program.

Your compiler should report semantic errors as accurately as possible. We shall give you a suit of programs to test your compiler against. At the end of the program, neatly print the symbol table. Each entry of the symbol table must contain the following information.

- The identifier whose information is stored in the entry.
- Its type, in case the identifier represents a data item.
- Its offset in the activation record.
- Other information such as number of dimension in the case of arrays.

You are also required to submit test cases for the semantic checker.

Project 4: Intermediate Code Generation for C- language

Assigned: April 21, 2015

Due Date for Test Cases: April 23, 2015

Due Date for Report and Code: April 24, 2015

PART I:

In this project you will generate a three address code for C language .We will test the results by compiling the resulting program, running it, and checking that it produces the expected output.

PART II:

Generate three address code for the original grammar.

Guidelines on how to present the results:

- You have been given a test suite of programs. You will test your compiler by generating assembly code which will be assembled, linked and executed to produce output.
- The evaluation will be done on the basis of another set of source programs, which will be given to you on the day of evaluation.

(If you implement anything beyond the requirements, include a description of it so that I will know what to test for)

GOOD LUCK

National Institute of Technology Karnataka- Surathkal.
Department of Computer Science & Engineering

Tentative Course Details : December 2014 – May 2015

Course:

No : CO 351
Title : Compiler Design LAB
Credits : (0-0-3) 2

Prerequisites:

CO 203: Unix programming

C0 206: Data Structures LAB

Instructor:

Name : P.Santhi Thilagam
Office : I floor, Computer Science & Engineering Building
Office Hours : Afternoons or by appointment
Email : santhi@nitk.ac.in

Objectives:

The students will undertake a sequence of experiments aimed at the design and implementation of the various phases of a compiler for the C programming language. This subset includes a sufficiently rich collection of data types and control structures.

General Requirements :

Team work

- Work in pairs, i.e., teams of no more than two students. (self-selected; team name list due on January 10).
- In this project, tasks should not be divided between the team members - the pair should tackle problems together, on the same computer, at the same time.

On grading

- For each phase, but the last, the marks are distributed as follows:
  - Report, 30% of the marks
  - Correctness and completeness, 60%
  - Program Documentation, 10%
- For the last phase the marks are distributed as follows:
  - Report, 10% of the marks
  - Correctness and completeness, 50%
  - Program Documentation, 10%
  - Presentation, 30%

COMPILER CONSTRUCTION: LAB MANUAL

Objective: The students will undertake a sequence of experiments aimed at the design and implementation of the various phases of a compiler for the C programming language . This subset includes a sufficiently rich collection of data types and control structures.

General Requirements :

Team work

- Work in pairs, i.e., teams of no more than two students. (self-selected; team name list due on January 25).
- In this project, tasks should not be divided between the team members - the pair should tackle problems together, on the same computer, at the same time.
- The team members will get the same marks for the assignments they hand in.

On grading

- For each phase, but the last, the marks are distributed as follows:
    - Report, 30% of the marks
    - Correctness and completeness, 60%
    - Program Documentation, 10%
- For the last phase the marks are distributed as follows:
    - Report, 10% of the marks
    - Correctness and completeness, 50%
    - Program Documentation, 10%
    - Presentation, 30%

Project 1: Scanner for C- language

Use LEX/Flex to implement a lexical analyzer for C language. Your lexical analyzer should support nested comments, and return appropriate error messages that are as meaningfull as possible, such as comments and strings that don't end until the end of the file, etc. Your documentation should explain how did you deal with comments, strings, errors etc.

Project 2: Parser for C- language

The objective of this assignment is familiarization with parser generators and development of a LALR parser for the source language.
Use YACC/Bison to implement a parser for C language. Your parser should have as few shift-reduce conflicts as possible, and no reduce-reduce conflics.

Explain in your documentation what shift-reduce conflics have been left unresolved (if any) and why aren't they harmful.

Project 4: Semantic checker for C- language

PART I:

Add a semantic checker to your compiler. It should do the checks on the list in class (repeated below) as well as build a symbol table for use during checking and in code generation. Use good style (including meaningful variable names) in your coding and be liberal with comments. You may use either C, C++.

Symbol Table Implementation and Manipulation:

Among the best ways to implement the symbol table uses a stack. Each entry in the symbol table will contain information related to a symbol representing a variable, a procedure, or a parameter. The exact set of information stored in the symbol table entry will depend on the class of the symbol described by the entry.

PART II:

The objective of this assignment is to perform semantic analysis such as type and scope analysis and declaration processing, and integrate such analyses with the parser. Add semantic actions to your parser to produce abstract syntax for the C language.

Project 4: Intermediate Code Generation for C- language

PART I:

In this project you will generate a three address code for C language .We will test the results by compiling the resulting program, running it, and checking that it produces the expected output.

PART II:

Generate three address code for the original grammar.
Guidelines on how to present the results:

- You have been given a test suite of programs. You will test your compiler by generating assembly code which will be assembled, linked and executed to produce output.
- The evaluation will be done on the basis of another set of source programs, which will be given to you on the day of evaluation.