ALLAMA IQBAL OPEN UNIVERSITY, ISLAMABAD

(Department of Computer Science)

WARNING

1. **PLAGIARISM OR HIRING OF GHOST WRITER(S) FOR SOLVING THE ASSIGNMENT(S) WILL DEBAR THE STUDENT FROM AWARD OF DEGREE/CERTIFICATE, IF FOUND AT ANY STAGE.**
2. **SUBMITTING ASSIGNMENT(S) BORROWED OR STOLEN FROM OTHER(S) AS ONE'S OWN WILL BE PENALIZED AS DEFINED IN "AIOU PLAGIARISM POLICY".**

Course: Compiler Construction (3468) **Semester: Autumn, 2013**
Level: BS (CS) **Total Marks: 100**
Pass Marks: 50

ASSIGNMENT No. 1

(Units: 1–4)

Note: All questions are compulsory & carry equal marks.

Q. 1 (a) Define compiler? What are the different compiler construction tools explain in detail?

(b) Why intermediate code generation is not included in front end or back end?

(c) Consider the following grammar.

$S \longrightarrow XaYb$

$X \longrightarrow bXc \mid b$

$Y \longrightarrow dYa \mid d$

Find the first set for each non-terminal of the given grammar.

Q. 2 (a) Construct a Syntax-Directed Translation scheme that takes strings of a's, b's and c's as input and produces as output the number of substrings in the input string that correspond to the pattern **a(a|b)*c+(a|b)*b**. For example the translation of the input string "**abbcabcababc**" is "3".

Your solution should include:

1. A context-free grammar that generates all strings of a's, b's and c's
2. Semantic attributes for the grammar symbols

(b) What is Abstract Stack machine explain in detail with the help of suitable examples?

Q. 3 (a) Define lexical analysis? Also roles of lexical analyzer in detail.

(b) What is Input Buffer explain in detail?

Q. 4 (a) Define Parser. Elaborate Top down parser in detail with the help of suitable examples.

(b) Convert the following regular expression into NFA using Thompson's construction.
a(a|b)*c+(a|b)*b

Q. 5 (a) Convert NFA into DFA of the following:
i. (a | b)*
ii. (a* | b*)*

ASSIGNMENT No. 2

(Total Marks: 100)

Note: All questions are compulsory & carry equal marks.

Q. 1 (a) Define Type Checking. What is the difference between Static and Dynamic Type checking?
(b) What is the Specification of a simple Type Checker?

Q. 2 Explain unification algorithm in detail with the help of suitable examples.

Q. 3 (a) Explain Back patching in detail with the help of suitable examples.
(b) Explain Procedure Call with the suitable example.

Q. 4 Convert the following LR grammar to right recursive grammar:
E → E + T / E – T / T
T → T x F / T / F / F
F → (E) / Numbers
Numbers → 0/1/2……/9.

Q. 5 Write a short notes on the following topics:
(a) Peephole optimization
(b) Loops in flow graphs
(c) Iterative solution of data-flow equations

3468 Compiler Construction Credit Hours: 3(3, 0)

Recommended Book:
Compliers; Principles, Techniques, and Tools by Alfred V. Aho, Ravi Sethi, Jerrey D. Ullman

Course Outlines:

Unit No. 1 Introduction to Compiling

Compliers, analysis of the source program, The phases of a Complier, Cousins of the compiler, The grouping of phases, Complier-construction tools

Unit No. 2 A Simple One-pass Compiler

Overview, Syntax definition, Syntax-directed translation, parsing, A translator for simple expressions, Lexical analysis, Incorporating a symbol table, Abstract stack machines, Putting the techniques together

Unit No. 3 Lexical and Syntax Analysis

Lexical analysis (The role of the Lexical Analyzer, Input buffering, Specification of tokens, Recognition of tokens, a language for specifying Lexical analyzers, finite automata, From a regular expression to an NFA, Design of a Lexical analyzer Generator, Optimization of DFA-based pattern matchers), Syntax Analysis (The role of the parser, context-free grammars, Writing a grammar, Top-down parsing, Bottom-up parsing, Operator-precedence parsing, LR parsers, Using ambiguous grammars, parser Generators)

Unit No. 4 Syntax-Directed Translation

Syntax-directed definitions, construction of syntax trees, Bottom-up evaluation of S-attributed definitions, L-attributed definitions, Top-down translation, Bottom-up evaluation of inherited attributes, Recursive evaluators, Space for attribute values at compile time, Assigning space at complier-construction time, Analysis of syntax-directed definitions

Unit No. 5 Type Checking

Type systems, Specification of a simple type checker, Equivalence of type expressions, Type conversions, Overloading of functions and operators, Polymorphic functions, an algorithm for unification

Unit No. 6 Intermediate Code Generation

Intermediate Languages, Declarations, Assignment statements, Boolean expressions, Case statements, Back Patching, Procedure calls

Unit No. 7 Code Generations

Issues in the design of a code generator, The target machine, Run-time storage management, Basic blocks and flow graphs, Next-use information, A simple code generator, Register allocation and assignment, The dag representation of basic blocks, Peephole optimization, Generating code from dags, Dynamic programming code-generation algorithm, Code-generator generators

Unit No. 8 Code Optimization

Introduction, The principal sources of optimization, Optimization of basic blocks, Loops in flow graphs, Introduction to global data-flow analysis, Iterative solution of data-flow equations, Code-improving transformations, Dealing with aliases, Data-flow analysis of structured flow graphs, Efficient data-flow algorithms, A tool for data-flow analysis, Estimation of types, Symbolic debugging of optimized code

Unit No. 9 Writing a Complier

Planning a compiler, Approaches to compiler development, The compiler-development environment, Testing and maintenance, A Look at Some Compilers, EQN, a preprocessor for typesetting mathematics, Compilers for Pascal, The C compilers, The Fortran H compilers, The Bliss/11 compiler, Modula-2 optimizing compiler.

✡✡✡