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, 2012**
Level: BS (CS) **Total Marks: 100**

ASSIGNMENT No. 1

Note: All questions carry equal marks.

Q. 1 (a) Define Compiler, using a diagram describes the three phases of analysis of source program.

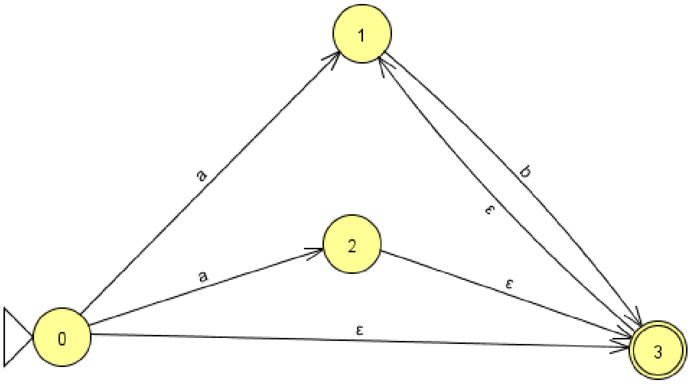
(b) Explain all the phases of Compiler.

(c) Consider the following grammar.
S —> XaYb
X —> bXc | b
Y —> dYa | d
Find the first sets for each non-terminal of the given grammar.

Q. 2 (a) Explain the error detection and reporting mechanisms.

(b) Write the intermediate representation code of the following position: = initial + rate * 60

Q. 3 (a) Convert the following NFA into equivalent DFA using subset construction Algorithm.



Note: Show all necessary steps that are involved in subset construction algorithm.

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

Q. 4 (a) Given the following grammar.
G → E
E → T + E | T
T → F * T | F
F → a
i) Is this grammar ambiguous? Explain!
ii) Draw all parse trees for sentence "a+a*a+a".

(b) Consider the following grammar.
S→ A
A→ A+A | B++
B → y
Draw parse tree for the input "y+++y++"

Q. 5 (a) Explain the role of the Lexical Analyzer and Parser in detail.

(b) Differentiate between Top-down parsing and Bottom-up parsing.

ASSIGNMENT No. 2

Total Marks: 100

Note: All questions carry equal marks.

Q. 1 (a) Rewrite the following SDT:
A A {a} B | *A B {b}* | 0
B -> *B* {c} *A* | *B A {d}* | 1
so that the underlying grammar becomes non-left-recursive. Here, a, 6, c, and *d* are actions, and 0 and 1 are terminals.

(b) This grammar generates binary numbers with a "decimal" point:
S-* L . L | L
L-+LB\B
B -> 0 | 1
Design an L-attributed SDD to compute *S.val,* the decimal-number value of an input string. For example, the translation of string 101.101 should be the decimal number 5.625.

Q. 2 (a) Translate the following expressions using the goto-avoiding translation scheme.
i) if (a==b *kk* c==d |I **e**==f) x == 1;
ii) if (a==b II c==d || e==f) x == 1;
iii) if (a==b && c==d *kk* e==f) x == 1;

(b) Construct the DAG and identify the value numbers for the sub expressions of the following expressions, assuming + associates from the left.
i) *a + b+ (a + b).*
ii) *a + b + a + b.*
iii) *a* + *a* + ((fl + *a* + *a* + *(a* + *a* + *a* + a)).

Q. 3 (a) Explain the following

i) Back Patching

ii) Procedure Calls

(b) Generate code for the following three-address statements, assuming all variables are stored in memory locations.

i) x = 1

ii) x = a

iii) x = a + 1

iv) x = a + b

v) The two statements

x = b * c
y = a + x

Q. 4 (a) The programming language C does not have a Boolean type. Show how a C compiler might translate if-statement into three-address code.

(b) Construct the DAG for the basic block

d = b * c
e = a + b
b = b * c
a = e - d

Q. 5 (a) Generate code for the following three-address statements assuming *a* and *b* are arrays whose elements are **4**-byte values.

i) The four-statement sequence

x = a [i]
y = b [j]
a [i] = y
b [j] = x

ii) The three-statement sequence

x = a [i]
y = b [i]
z = x * y

iii) The three-statement sequence

x = a [i]
y = b[x]
a [i] = y

(b) Suppose a basic block is formed from the C assignment statements

x = a + b + c + d + e + f;
y = a + c + e;

i) Give the three-address statements (only one addition per statement) for this block.

ii) Use the associative and commutative laws to modify the block to use the fewest possible number of

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