MAY/JUNE 2012 EXAMINATION CIT 445Principles & Techniques of Compilers

1a) Define formal Grammar.) 4 marks

A **formal grammar** (sometimes called a **grammar**) is a set of rules of a specific kind, for forming strings in a formal language.

b) List and define the four basic types of grammars in the field of Computer Science) 8 marks

i. Type-0 grammars (unrestricted grammars) include all formal grammars.

ii. Type-1 grammars (context-sensitive grammars) generate the context-sensitive languages.

iii. Type-2 grammars (context-free grammars) generate the contextfree languages.

iv. Type-3 grammars (regular grammars) generate the regular languages.

c. Given the grammar *G* with following production rules, S → a | aS | bS, determine whether the string *bbaaba* can be generated by the grammar) 2marks

Using

S = bS

S = ba

S = bbaaS= bbaaba

2a. Define formal language) 3 marks

A **formal language** is a set of *words*, i.e. finite strings of *letters*, *symbols*, or *tokens*.

b. State three of the uses of formal languages) 3 marks

c. What is a translator?) 2marks

A translator is a programme that takes as input a programme written in one programming language (the source language)and produces as output a programme in another language (the object or target lan guage).

d. Why do we need a translator?) 3 marks

We need translators to overcome the rigour of programming in machine language, which involves communicating directly with a computer in terms of bits, register, and primitive machine operations.

e. Enumerate the functions performed by the lexical analyser) 4 marks

The main function is to simplify the overall design of the compiler. Lexical analyser also performs other functions such as keeping track of line numbers, producing an output listing if necessary, stripping out white space (such as redundant blanks and tabs), and deleting comments.

3a. Compare interpreter and compiler) 5 marks

Compiler: A *compiler* is a programme that translates a source programme written in some high-level programming language (such as Java) into machine code for some computer architecture (such as the Intel Pentium architecture).

Interpreter: An *interpreter* reads an executable source programme written in a high-level programming language as well as data for this programme, and it runs the programme against the data to produce some results.

b. State any five qualities of a compiler) 5 marks

i. the compiler itself must be bug-free

ii. it must generate correct machine code

iii. the generated machine code must run fast

iv. the compiler itself must run fast (compilation time must be proportional to programme size)

v. the compiler must be portable (i.e. modular, supporting separate compilation)

vi. it must print good diagnostics and error messages

vii. the generated code must work well with existing debuggers

viii. must have consistent and predictable optimisation.

c. State the knowledge needed to build a compiler) 4 marks

i. programming languages (parameter passing, variable scoping, memory allocation, etc.)

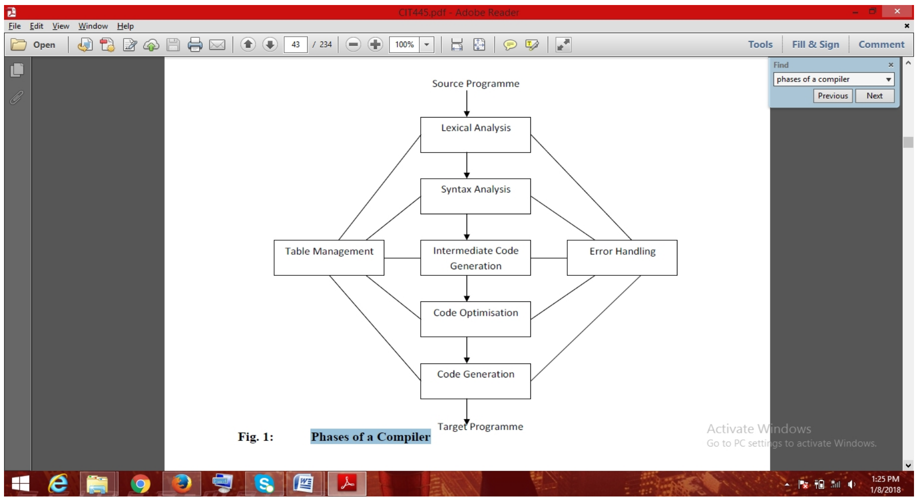
ii. theory (automata, context-free languages, etc.)

iii. algorithms and data structures (hash tables, graph algorithms, dynamic programming, etc.)

iv. computer architecture (assembly programming)

v. software engineering.

4) With the aid of illustrative diagram describe the phases of a compiler.) 14 marks



5) Consider the grammar G below:

G: E E + T / T

T T * F / F

F (E) / i

a) Generate the non-left recursive version of the grammar) 5 marks

E ⇒ E + T

⇒ T + T

⇒ F + T

⇒ id + T

⇒ id + T * F

⇒ id + F * F

⇒ id + id * F

⇒ id + id * id

b) Find FOLLOW of all the nonterminal symbols in the non-left recursive version of the gramm ar)

6a) What are the benefits of LR parsing?) 5 marks

b. List the common techniques for building tables for an “LR” parser stating the characteristics of each?) 6 marks

c. Consider the grammar,

G: E → E + T | T

T → T*F | F

F → (E) | i

What is the augmented grammar for this grammar.) 4 marks

Answer:

G’: E’ -> E

E -> E + T | T

T -> T*F | F

F _ (E) | i

7) Consider the grammar,

G: S → a | aS | bS

a) Find the LR(0) items for this grammar) 10 marks

b) Construct an NFA whose states are the LR(0) items from (a).

JANUARY/FEBRUARY 2013 EXAMINATION CIT 445 Principles & Techniques of Compilers

1a) Explain what is meant by the term handle?) 2 marks

A handle of a string is a substring that matches the right side of a production whose reduction to the no nterminal on the left represents one step along the reverse of a rightmost derivation.

b) Consider the following grammar for list structure:

S → a | ^ | (T)

T → T,S | S

i) find the rightmost derivations for:) 7marks

(i) (a, (a, a))

(ii) (((a, a), ^, (a)), a)

ii) Indicate the handle of each right sentential form for the derivations in (a) above) 5 marks

2a) Briefly describe the operation performed by the shift-reduce parser) 6 marks

The *shift-reduce parser* operates by shifting zero or more symbols onto the stack until a handle appears on the top. The parser then reduces the handle to the left side of the corresponding production. This process continues until an error occurs or the start symbol remains on the stack.

b) Given the context-free grammar G below:

G: E E + E

E E * E

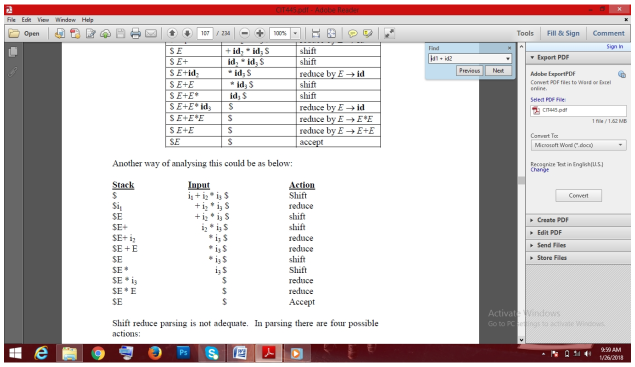
E (E)

E **id**

State the steps performed by the shift-reduce parser when analyzing the input string:

$\textbf{id}_1 + \textbf{id}_2 * \textbf{id}_3$) 8 marks

Answer:

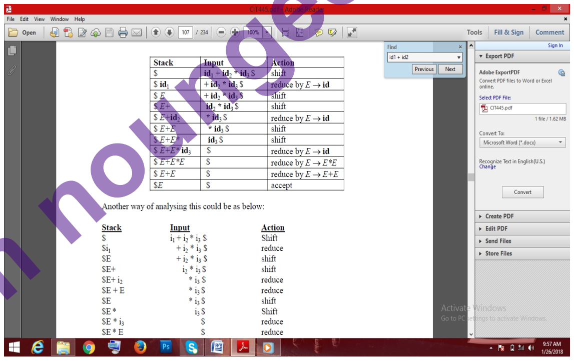


\$ E	+ $\mathbf{id}_2$ * $\mathbf{id}_3$ \$	shift
\$ E+	$\mathbf{id}_2$ * $\mathbf{id}_3$ \$	shift
\$ E+$\mathbf{id}_2$	* $\mathbf{id}_3$ \$	reduce by $E \rightarrow \mathbf{id}$
\$ E+E	* $\mathbf{id}_3$ \$	shift
\$ E+E*	$\mathbf{id}_3$ \$	shift
\$ E+E* $\mathbf{id}_3$	\$	reduce by $E \rightarrow \mathbf{id}$
\$ E+E*E	\$	reduce by $E \rightarrow E*E$
\$ E+E	\$	reduce by $E \rightarrow E+E$
\$$E$	\$	accept

Another way of analysing this could be as below:

Stack	Input	Action
\$	$i_1 + i_2 * i_3$ \$	Shift
\$$i_1$	$+ i_2 * i_3$ \$	reduce
\$E	$+ i_2 * i_3$ \$	shift
\$E+	$i_2 * i_3$ \$	shift
\$E+ i_2	$* i_3$ \$	reduce
\$E + E	$* i_3$ \$	reduce
\$E	$* i_3$ \$	shift
\$E *	i_3 \$	Shift
\$E * i_3	\$	reduce
\$E * E	\$	reduce
\$E	\$	Accept

Shift reduce parsing is not adequate. In parsing there are four possible actions:



Stack	Input	Action
\$	$\mathbf{id}_1$ + $\mathbf{id}_2$ * $\mathbf{id}_3$ \$	shift
\$ $\mathbf{id}_1$	+ $\mathbf{id}_2$ * $\mathbf{id}_3$ \$	reduce by $E \rightarrow \mathbf{id}$
\$ E	+ $\mathbf{id}_2$ * $\mathbf{id}_3$ \$	shift
\$ E+	$\mathbf{id}_2$ * $\mathbf{id}_3$ \$	shift
\$ E+$\mathbf{id}_2$	* $\mathbf{id}_3$ \$	reduce by $E \rightarrow \mathbf{id}$
\$ E+E	* $\mathbf{id}_3$ \$	shift
\$ E+E*	$\mathbf{id}_3$ \$	shift
\$ E+E* $\mathbf{id}_3$	\$	reduce by $E \rightarrow \mathbf{id}$
\$ E+E*E	\$	reduce by $E \rightarrow E*E$
\$ E+E	\$	reduce by $E \rightarrow E+E$
\$$E$	\$	accept

Another way of analysing this could be as below:

Stack	Input	Action
\$	$i_1 + i_2 * i_3$ \$	Shift
\$$i_1$	$+ i_2 * i_3$ \$	reduce
\$E	$+ i_2 * i_3$ \$	shift
\$E+	$i_2 * i_3$ \$	shift
\$E+ i_2	$* i_3$ \$	reduce
\$E + E	$* i_3$ \$	reduce
\$E	$* i_3$ \$	shift
\$E *	i_3 \$	Shift
\$E * i_3	\$	reduce
\$E * E	\$	reduce

OR

3a) Explain what is meant by the term Viable Prefix?) 3 marks

A viable prefix is so called because it is always possible to add terminal symbols to the end of a viable prefix to obtain a right sentential form.

b) Given the grammar *G* with following production rules, S → a | aS | bS, determine whether the string *aababbba* can be generated by the grammar) 5marks

c) Enumerate any three of the errors which can be detected during lexical analysis) 6 marks

Strange characters: Some programming languages do not use all possible characters, so any strange ones which appear can be reported.

Long quoted strings I: Many programming languages do not allow quoted strings to extend over more than one line; in such cases a missing quote can be detected.

Long quoted strings II: If quoted strings can extend over multiple lines then a missing quote can cause quite a lot of text to be 'swallowed up' before an error is detected.

Invalid numbers: A number such as 123.45.67 could be detected as invalid during lexical analysis (provided the language does not allow a full stop to appear immediately after a number).

4a) Explain what is meant by top-down parsing technique) 2 marks

Top-down parsing can be viewed as an attempt to find a leftmost derivation for an input string or as attempting to construct a parse tree for the input string starting from the root and creating the nodes of the parse tree in pre-order

b) State the difficulties in top-down parsing) 6 marks

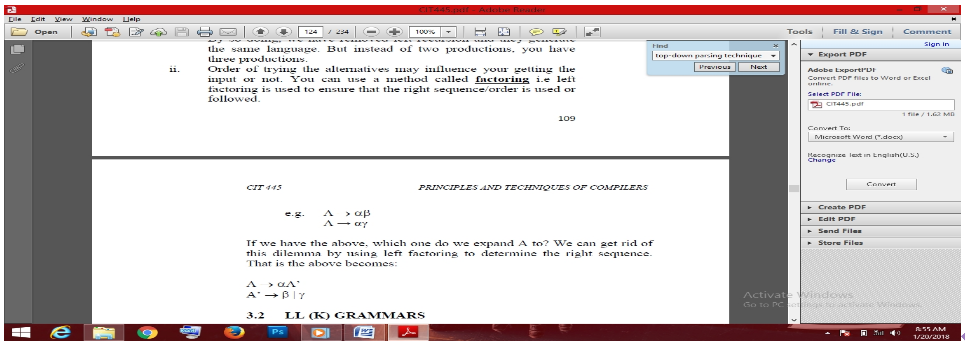
i. Which production will you apply? If the right production is not applied you will not get the input string.

ii. If you are not careful, and there is a left recursive production, it can lead to continue cycling without getting to the answer i.e. input string.

iii. The sequence in which you apply the production matters as to whether you are going to get the input string or not. That is, there is a particular sequence that will lead you to the input string.

iv. If you apply a production and find out that the production cannot work, you have to start all over again.

c) Using examples, state and illustrate how to minimized) 6 marks



the same language. But instead of two productions, you have three productions.

ii. Order of trying the alternatives may influence your getting the input or not. You can use a method called **factoring** i.e left factoring is used to ensure that the right sequence/order is used or followed.

109

CIT 445 *PRINCIPLES AND TECHNIQUES OF COMPILERS*

e.g. $A \rightarrow \alpha\beta$
$A \rightarrow \alpha\gamma$

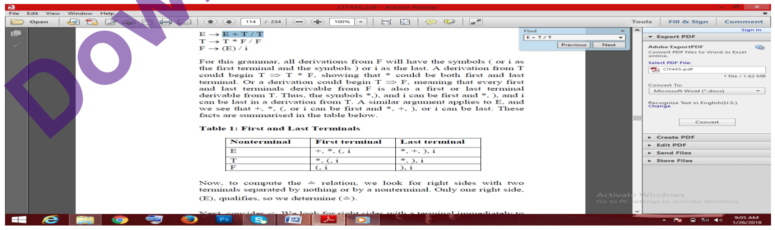
If we have the above, which one do we expand A to? We can get rid of this dilemma by using left factoring to determine the right sequence. That is the above becomes:

$A \rightarrow \alpha A'$
$A' \rightarrow \beta \mid \gamma$

3.2 LL (K) GRAMMARS

Factoring method

5) Consider the grammar Ggiven below:



$E \rightarrow E + T / T$
$T \rightarrow T * F / F$
$F \rightarrow (E) / i$

For this grammar, all derivations from F will have the symbols (or i as the first terminal and the symbols) or i as the last. A derivation from T could begin $T \Rightarrow T * F$, showing that * could be both first and last terminal. Or a derivation could begin $T \Rightarrow F$, meaning that every first and last terminals derivable from F is also a first or last terminal derivable from T. Thus, the symbols *,), and i can be first and *,), and i can be last in a derivation from T. A similar argument applies to E, and we see that +, *, (, or i can be first and *, +,), or i can be last. These facts are summarised in the table below.

Table 1: First and Last Terminals

Nonterminal	First terminal	Last terminal
E	+, *, (, i	*, +,), i
T	*, (, i	*,), i
F	(, i	), i

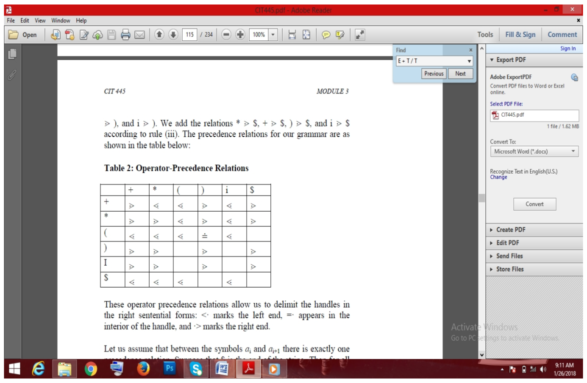
Now, to compute the $\doteq$ relation, we look for right sides with two terminals separated by nothing or by a nonterminal. Only one right side, (E), qualifies, so we determine ($\doteq$).

G: E E + T / T

T T * F / F

F (E) / i

a) Find all the first and last terminals in this grammar) 5 marks



CIT 445 MODULE 3

⋗), and i ⋗). We add the relations * ⋗ $, + ⋗ $,) ⋗ $, and i ⋗ $ according to rule (iii). The precedence relations for our grammar are as shown in the table below:

Table 2: Operator-Precedence Relations

	+	*	(	)	i	$
+	⋗	⋖	⋖	⋗	⋖	⋗
*	⋗	⋗	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	≐	⋖	
)	⋗	⋗		⋗		⋗
I	⋗	⋗		⋗		⋗
$	⋖	⋖	⋖		⋖	

These operator precedence relations allow us to delimit the handles in the right sentential forms: <· marks the left end, =· appears in the interior of the handle, and ·> marks the right end.

Let us assume that between the symbols a_i and a_{i+1} there is exactly one

b) Generate the operator precedence passing table for this grammar) 9 marks

6a) Define the following for any given grammar?) 5 marks

FOLLOW A: for non-terminal A to be the set of terminals *a*, that can appear immediately to the right of A in some sentential form.

I) FIRST(α)

b) Consider the grammar,

G: E → E + T | T

T → T*F | F

F → (E) | i

I) Find the FOLLOW(A) for all the terminal in G) 4 marks

II) Find the FIRST(α) for any string derivable from G) 5 marks

7a) Consider the grammar

S → L = R | R

L → *R | i

R → L

a) Compute all the LR(0) items for the above grammar)10 marks

Construct an NFA whose states are the LR(0) items from (a))4 marks

SCHOOL OF SCIENCE AND TECHNOLOGY

JUNE/JULY EXAMINATION

COURSE CODE: CIT445

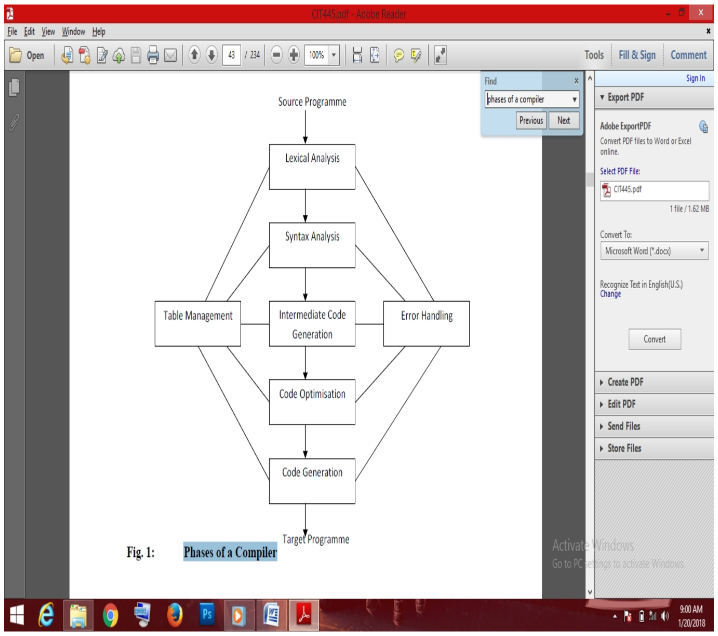
COURSE TITLE: Principles & Techniques of Compilers

TIME ALLOWED: 2½ hrs

INSTRUCTION: Answer any five (5) questions. Each question carries 14 marks

1) With the aid of an illustrative diagram describe the phases of a compiler.

14 marks



2a) What are the benefits of LR parsing? 4½ marks

i. it detects syntactic errors when the input does not conform to the grammar as soon as possible.

ii. It is more general than operator precedence or any other common shift-reduce techniques

iii. LR parsing also dominates the common forms of top-down parsing without backtrack.\

iv. LR parsers can be constructed to recognise virtually all programming language constructs for which context-free grammars can be written

b) List the common techniques for building tables for an "LR" parser stating the characteristics of each?

Simple LR

i. smallest class of grammars ii. smallest tables (number of states) iii. simple, fast construction

Canonical LR

i. full set of LR(1) grammars ii. largest tables iii. slow, expensive and large construction

Lookahead LR

i. intermediate sized set of grammars ii. same number of states as SLR(1)

iii. canonical construction is slow and large   iv. better construction techniques exist

b) Consider the grammar,

G: E → E + T | T

T → T*F | F

F → (E) | i

What is the augmented grammar for this grammar. 3 marks

3a) Briefly describe the operations performed by the shift-reduce parser

6 marks

i. shift - the next input symbol is shifted onto the top of the stack

ii. *reduce* - the parser knows the right end of the handle is at the top of thestack. It must then locate the left end of the handle within the stack and decide with what nonterminal to replace the handle

iii. *accept* - the parser announces successful completion of parsing

iv. *error* - the parser discovers that a syntax error has occurred.

b) Given the context-free grammar G below:

G: E E + E

E E * E

E (E)

E **id**

4) Consider the grammar,

G: S → a | aS | bS

a) Find the LR(0) items for this grammar

10 marks

b) Construct an NFA whose states are the LR(0) items from (a). 4 marks

5a) Explain what is meant by the term Viable Prefix? 3 marks

b) Given the grammar *G* with following production rules, S → a | aS | bS, determine whether the string ***aababbba*** can be generated by the grammar 5 marks

c) Enumerate any three of the errors which can be detected during lexical analysis 6 marks

6) Consider the grammar G below:

G: E E + T / T

T T * F / F

F (E) / i

a) Generate the non-left recursive version of the grammar

5 marks

b) Find FOLLOW of all the nonterminal symbols in the non-left recursive version of the grammar 9 marks

7a) Define formal language 3 marks

b) State three of the uses of formal languages 3 marks

c) What is a translator?

2 marks

d) Why do we need a translator?

3 marks

e) Enumerate the functions performed by the lexical analyser

4 marks

October, 2013 Examination CIT 445 Principles& Techniques of Compilers

1.(a) Define the term parsing (3 marks)

(b) State and discuss four examples of analytic grammar formalisms (8 marks)

the Language Machine: directly implements unrestricted analytic grammars.

top-down parsing language (TDPL): a highly minimalist analytic grammar formalism developed in the early 1970s to study the behaviour of top-down parsers.

link grammars: a form of analytic grammar designed for linguistics, which derives syntactic structure by examining the positional relationships between pairs of words.

parsing expression grammars (PEGs): a more recent generalisation of TDPL designed around the practical expressiveness needs of programming language and compiler writers.

(c) Given the grammar *G* with the following production rules, S → a | aS | bS, determine whether the string *babbaa* can be generated by the grammar (3 marks)

2.(a) What is the difference between a translator and a compiler (6 marks)

A *compiler* is a programme that translates a source programme written in some high-level programming language (such as Java) into machine code for some computer architecture.

A translator is a programme that takes as input a programme written in one programming language (the source language) and produces as output a programme in another language

(b) State and describe four components of the structure of a compiler (8 marks)

1. Front end
2. Back-end
3. Tables of information
4. Runtime library

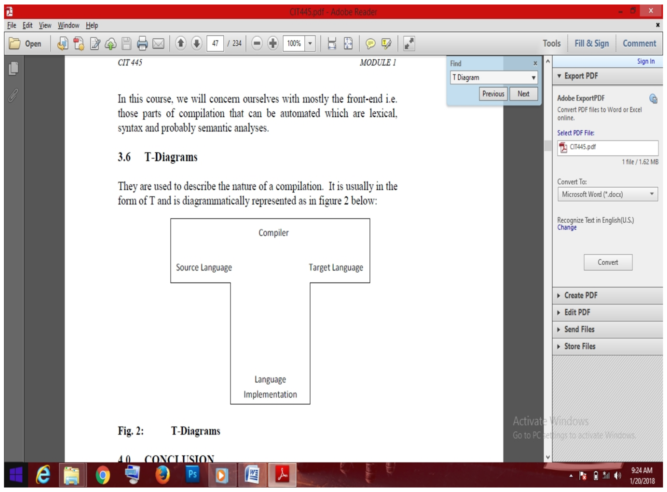
i) Front-End: the front-end is responsible for the analysis of the structure and meaning of the source text.

ii) Back-End: The back-end is responsible for generating the target language.

iii) Tables of Information: It includes the symbol-table and there are some other tables that provide information during compilation process.

iv) Run-Time Library: It is used for run-time system support.

3.(a) With the aid of a diagram describe the functions of a T.diagram (8 marks)



CIT 445 *MODULE 1*

In this course, we will concern ourselves with mostly the front-end i.e. those parts of compilation that can be automated which are lexical, syntax and probably semantic analyses.

3.6 T-Diagrams

They are used to describe the nature of a compilation. It is usually in the form of T and is diagrammatically represented as in figure 2 below:

Fig. 2: T-Diagrams

4.0 CONCLUSION

(b) State the Roles of a Parser (6 marks)

• The parser reads the sequence of tokens generated by the lexical analyser

• It verifies that the sequence of tokens obeys the correct syntactic structure of the programming language by generating a parse tree implicitly or explicitly for the sequence of tokens

• It enters information about tokens into the symbol table

• It reports syntactic errors to the user

4.(a) State four difficulties with Top-down parsing (6 marks)

i. Which production will you apply? If the right production is not applied you will not get the input string.

ii. If you are not careful, and there is a left recursive production, it can lead to continue cycling without getting to the answer i.e. input string.

iii. The sequence in which you apply the production matters as to whether you are going to get the input string or not. That is, there is a particular sequence that will lead you to the input string.

iv. If you apply a production and find out that the production cannot work, you have to start all over again.

(b) State five benefits of LR Parsing (7 marks)

i. it detects syntactic errors when the input does not conform to the grammar as soon as possible.

ii. It is more general than operator precedence or any other common shift-reduce techniques

iii. LR parsing also dominates the common forms of top-down parsing without backtrack.\

iv. LR parsers can be constructed to recognise virtually all programming language constructs for which context-free grammars can be written

5. With the aid of illustrative diagram describe the phases of a compiler (14 marks)

CHECK 2012 FOR THE DIAGRAM

6.(a) State and describe the three main techniques for loop optimisation (6 marks)

• *Strength Reduction* which replaces an expensive (time consuming) operator by a faster one;

• *Induction Variable Elimination* which eliminates variables from inner loops;

• *Code Motion* which moves pieces of code outside loops.

(b) State any sixqualities of a compiler (8 marks)

i. the compiler itself must be bug-free

ii. it must generate correct machine code

iii. the generated machine code must run fast

iv. the compiler itself must run fast (compilation time must be proportional to programme size)

v. the compiler must be portable (i.e. modular, supporting separate compilation)

vi. it must print good diagnostics and error messages

vii. the generated code must work well with existing debuggers

viii. must have consistent and predictable optimisation.

7. Consider the grammar G below:

G: E E + T / T

T T * F / F

F (E) / i

(a) Generate the non-left recursive version of the grammar (5 marks)

(b)Find FOLLOW of all the nonterminal symbols in the non-left recursive version of the grammar

(9 marks)

MARCH/APRIL 2014 EXAMINATION CIT 445 PRINCIPLES & TECHNIQUES OF COMPILERS

1. (a) What do you understand by the term Viable Prefix? (3 marks)

(b) Given the grammar *G* with following production rules, S → a | aS | bS, determine whether the string ***aababbba*** can be generated by the grammar (5marks)

(c) Enumerate any three of the errors which can be detected during lexical analysis (6 marks)

2. (a) What is the difference between a translator and a compiler (6 marks)

(b) State and describe four components of the structure of a compiler (8 marks)

3. (a) What do you understand by top-down parsing technique (2 marks)

(b) State the difficulties in top-down parsing (6 marks)

(c) Using examples state and illustrate how to minimize (6 marks)

4. (a) State and describe the three main techniques for loop optimisation (6 marks)

(b) State eight qualities of a compiler (8 marks)

5. With the aid of illustrative diagram describe the phases of a compiler (14 marks)

6. (a) With the aid of a diagram describe the functions of a T.diagram (8 marks)

(b) State the Roles of a Parser (6 marks)

7. Consider the grammar G below:

G: E E + T / T

T T * F / F

F (E) / i

(a) Generate the non-left recursive version of the grammar (5 marks)

(b) Find FOLLOW of all the nonterminal symbols in the non-left recursive version of the grammar (9 marks)

OCTOBER/NOVEMBER 2014 EXAMINATION CIT 445 Principles and Techniques of Compilers

1a. List the ten (10) types of information required by a compiler (10 marks)

• textual name • data type • dimension information (*for aggregates*)

• declaring procedure • lexical level of declaration • storage class (*base address*) • offset in storage

• if record, pointer to structure table • if parameter, by-reference or by-value?

• can it be aliased? to what other names? • number and type of arguments to functions

1b. Explain the term "formal system" (2 marks)

A *formal system* consists of a formal language together with a deductive apparatus

1c. Outline any two properties an optimising compiler should provide (2 marks)

• the transformations should preserve the semantics of the programmes, that is the changes should guarantee that the same input produces the same outputs (and should not cause errors)

• the transformations should speed up considerably the compiler on the average (although occasionally on some inputs this may not be demonstrated, on most of the inputs it should become faster)

• the transformation should be worth the intellectual effort.

2a. Describe each of the following parameter passing method:

Call by value: The calling procedure calculates and passes the address containing the value of the actual parameter.

Call by reference: The called procedure uses this address to refer to the parameter.

Call by name: This call implementation requires a textual substitution of the formal parameter name by the actual parameter.

Call by value result: A parameter can be stored as both value and result. In this case, the local location of the formal parameters is initialised to the valu contained in the address of the actual parameter and the called procedure returns the result back to the actual parameter.

2b. Define the term "Translators" (2 marks)

A translator is a programme that takes as input a programme written in one programming language (the source language)and produces as output a programme in another language (the object or target language).

3a. Complete the table by inserting the Translators and Target languages of the mentioned Source languages. (10 marks)

Source Language	**Translator**	**Target Language**
LaTeX	Text Formatter	PostScript
SQL	Database Query Optimiser	Query Evaluation Plan
Java	Javac Compiler	Java Byte Code
English text	Natural Language Understanding	Semantics (meaning)

Regular Expressions	JLex Scanner Generator	A Scanner in Java
BNF of a language	CUP parser generat or	A Parser in Java

3b. Explain the term "Formal Grammer"   2 marks

A **formal grammar** (sometimes called a **grammar**) is a set of rules of a specific kind, for forming strings in a formal language.

3c. Why Do We Need Translators?   2 marks

We need translators to overcome the rigour of programming in machine language, which involves communicating directly with a computer in terms of bits, register, and primitive machine

4a. Outline the challenges involved in developing compilers (11 marks)

i. Many variations:

- many programming languages (e.g. FORTRAN, C++, Java)

- many programming paradigms (e.g. object-oriented, functional, logic)

-many computer architectures (e.g. MIPS, SPARC, Intel, alpha)

-many operating systems (e.g. Linux, Solaris, Windows)

ii. Qualities of a compiler: these concerns the qualities that are compiler must possess in other to be effective and useful.

- the compiler itself must be bug-free

- it must generate correct machine code

- the generated machine code must run fast

- the compiler itself must run fast (compilation time must be proportional to programme size)

- the compiler must be portable (i.e. modular, supporting separate compilation)

- it must print good diagnostics and error messages

- the generated code must work well with existing debuggers

- must have consistent and predictable optimisation.

iii. In-depth knowledge:

Building a compiler requires in-depth knowledge of:

- programming languages (parameter passing, variable scoping, memory allocation, etc.)

- theory (automata, context-free languages, etc.)

- algorithms and data structures (hash tables, graph algorithms, dynamic programming, etc.)

- computer architecture (assembly programming)

- software engineering.

4b. What is a Compiler? 3 marks

a *compiler* is a programme that translates a source programme written in some high-level programming language (such as Java) into machine code for some computer architecture (such as the In tel Pentium architecture)

5a. Explain the four (4) components of a compiler (8 marks)

i) Front-End: the front-end is responsible for the analysis of the structure and meaning of the source text.

ii) Back-End: The back-end is responsible for generating the target language.

iii) Tables of Information: It includes the symbol-table and there are some other tables that provide information during compilation process.

iv) Run-Time Library: It is used for run-time system support.

5b. Mention six (6) reasons why there is a need to study LR grammars (6 marks)

• LR (1) grammars are often used to construct parsers. We call these parsers LR(1) parsers and it is everyone’s favourite parser

• virtually all context-free programming language constructs can be expressed in an LR(1) form

• LR grammars are the most general grammars parse-able by a deterministic, bottom-up parser

• efficient parsers can be implemented for LR(1) grammars

• LR parsers detect an error as soon as possible in a left-to-right scan of the input

• LR grammars describe a proper superset of the languages recognised by predictive (i.e., LL) parsers

6a. Clearly explain the four (4) benefits of parsing (8 marks)

i. it detects syntactic errors when the input does not conform to the grammar as soon as possible.

ii. It is more general than operator precedence or any other common shift-reduce techniques

iii. LR parsing also dominates the common forms of top-down parsing without backtrack.\

iv. LR parsers can be constructed to recognise virtually all programming language constructs for which context-free grammars can be written

6b. Consider porting a compiler for C written in C from an existing machine A to an existing machine B. Write out the steps to cross compilation? (6 marks)

a. Write new back-end in C to generate code for computer B

b. Compile the new back-end and using the existing C compiler running on computer A generating code for computer B.

c. We now have a compiler running on computer A and generating code for computer B.

d. Use this new compiler to generate a complete compiler for computer B. In other words, we can compile the new compiler on computer A to generate code for computer B

e. We now have a complete compiler for computer B that will run on computer B.

f. Copy this new compiler across and run it on computer B (this is cross Compilation).

7a. What are the errors that could occur during Lexical Analysis (8 marks)

Long quoted strings I: Many programming languages do not allow quoted strings to extend over more than one line; in such cases a missing quote can be detected.

Strange characters: Some programming languages do not use all possible characters, so any strange ones which appear can be reported.

• **Long quoted strings II:** If quoted strings can extend over multiple lines then a missing quote can cause quite a lot of text to be 'swallowed up' before an error is detected.

Invalid numbers: A number such as 123.45.67 could be detected as invalid during lexical analysis (provided the language does not allow a full stop to appear immediately after a number).

7b. Discuss the following: (6 marks)

I. Loop Optimizations

Strength Reduction: This concept refers to the compiler optimisation method of substituting some machine instruction by a cheaper one and still maintaining equivalence in results.

Function chunking: is a compiler optimisation for improving code locality.

MARCH/APRIL 2015 CIT 445 PRINCIPLES AND TECHNIQUES OF COMPILERS

1a) In the context of Computer Science, what do you understand by the word "Formal Grammar"

A **formal grammar** (sometimes called a **grammar**) is a set of rules of a specific kind, for forming strings in a formal language.

b) In the classic formalization of generative grammars first proposed by Noam Chomsky, list any three component of a grammar "G"?

a. Type-0 grammars (unrestricted grammars) include all formal grammars.

b. Type-1 grammars (context-sensitive grammars) generate the context-sensitive languages.

c. Type-2 grammars (context-free grammars) generate the contextfree languages.

d. Type-3 grammars (regular grammars) generate the regular languages.

c) Discuss the four basic types of grammars

Type-0: Unrestricted Grammars

These grammars generate exactly all languages that can be recognized by a Turing machine.

Type-1: Context-Sensitive Grammars

These grammars have rules of the form with being a non-terminal and strings of terminals and non-terminals

Type2: Context-Free Grammars

A *context-free grammar* is a grammar in which the left-hand side of each production rule consists of only a single non-terminal symbol

Type-3: Regular Grammars

In regular grammars, the left hand side is again only a single nonterminal symbol, but now the right-hand side is also restricted.

d) What is a formal language?

A **formal language** is a set of *words*, i.e. finite strings of *letters*, *symbols*, or *tokens*.

2a) It is customary to partition the compilation process into a series of sub-process called phases. What do you understand by the term "phase"?

A phase is a logically cohesive operation that takes as input one representation of the source programme and produces as output another representation.

b) With the aid of a suitable diagram list the different phases of the compilation process

CHECK 2012 FOR ANSWER

c) Discuss the operation of a compiler

Front End:

• The lexical analysis (or scanning)

• The syntax analysis

• Semantic analysis

Back End:

• Intermediate code optimisation

• Code generation

• Code optimisation

3a) Briefly describe and distinguish among the following: i) Translator (ii) compiler (iii) interpreter

A translator is a programme that takes as input a programme written in one programming language (the source language)and produces as output a programme in another language (the object or target language).

Compiler: A *compiler* is a programme that translates a source programme written in some high-level programming language (such as Java) into machine code for some computer architecture (such as the Intel Pentium architecture).

Interpreter: An *interpreter* reads an executable source programme written in a high-level programming language as well as data for this programme, and it runs the programme against the data to produce some results.

b) List any five qualities of compiler

i. the compiler itself must be bug-free ii. it must generate correct machine code

iii. the generated machine code must run fast iv. the compiler itself must run fast (compilation time must be proportional to programme size) v. the compiler must be portable (i.e. modular, supporting s eparate compilation) vi. it must print good diagnostics and error message vii. the generated code must work well with existing debuggers viii. must have consistent and predictable optimisation.

c) With the aid of a suitable diagram, clearly state the use of T-d3iagrams

4a) List the steps to implement Lex

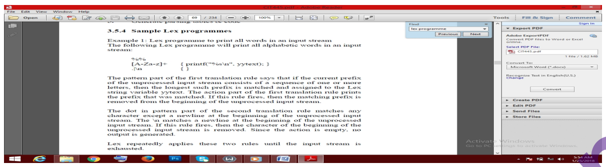
1. Read input lang. spec

2. Construct NFA with epsilon-moves (Can also do DFA directly)

3. Convert NFA to DFA

4. Optimise the DFA

5. Generate parsing tables & code



3.5.4 Sample Lex programmes

Example 1: Lex programme to print all words in an input stream
The following Lex programme will print all alphabetic words in an input stream:

```
%%
[A-Za-z]+      { printf("%s\n", yytext); }
.|\n           { }
```

The pattern part of the first translation rule says that if the current prefix of the unprocessed input stream consists of a sequence of one or more letters, then the longest such prefix is matched and assigned to the Lex string variable yytext. The action part of the first translation rule prints the prefix that was matched. If this rule fires, then the matching prefix is removed from the beginning of the unprocessed input stream.

The dot in pattern part of the second translation rule matches any character except a newline at the beginning of the unprocessed input stream. The \n matches a newline at the beginning of the unprocessed input stream. If this rule fires, then the character of the beginning of the unprocessed input stream is removed. Since the action is empty, no output is generated.

Lex repeatedly applies these two rules until the input stream is exhausted.

b) Write a Lexprogramme to print all alphabetic words in an input stream

Answer:

c) Explain the process to create a lexical processor with Lex

Put lex programme into a file, say file.l.

Compile the lex programme with the command:

• lex file.l

This command produces an output file lex.yy.c.

Compile this output file with the C compiler and the lex library -ll:gcc lex.yy.c -ll

The resulting a.out is the lexical processor.

5a) Briefly describe a “parser” and state any three roles of the parser

The Parser takes tokens from lexer (scanner or lexical analyser) and builds a parse tree

Role of the Parser

• The parser reads the sequence of tokens generated by the lexical

analyser

• It verifies that the sequence of tokens obeys the correct syntactic

structure of the programming language by generating a parse tree

implicitly or explicitly for the sequence of tokens

• It enters information about tokens into the symbol table

• It reports syntactic errors to the user.

b) State the need and constituents of a context-free Grammar

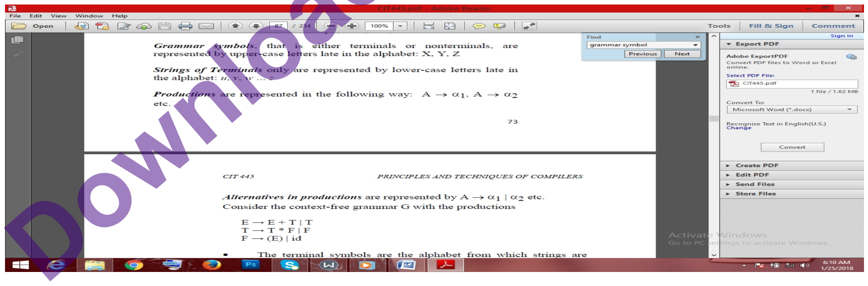
CFG's are very useful for representing the syntactic structure of programming languages.

1. A finite set of terminal symbols, 2. A finite nonempty set of nonterminal symbols,

3. One distinguished nonterminal called the start symbol, and 4. A finite set of rewrite rules, called

productions

c) How are the following represented?


Grammar symbols, that is either terminals or nonterminals, are represented by upper-case letters late in the alphabet: X, Y, Z

Strings of Terminals only are represented by lower-case letters late in the alphabet: $u, v, w \ldots z$

Productions are represented in the following way: $A \rightarrow \alpha_1$, $A \rightarrow \alpha_2$ etc.

73

CIT 445 PRINCIPLES AND TECHNIQUES OF COMPILERS

Alternatives in productions are represented by $A \rightarrow \alpha_1 \mid \alpha_2$ etc.
Consider the context-free grammar G with the productions

$$E \rightarrow E + T \mid T$$
$$T \rightarrow T * F \mid F$$
$$F \rightarrow (E) \mid id$$

• The terminal symbols are the alphabet from which strings are

i) Grammar symbols (ii) strings

of terminals (iii) productions

6a) Write short notes on the following:

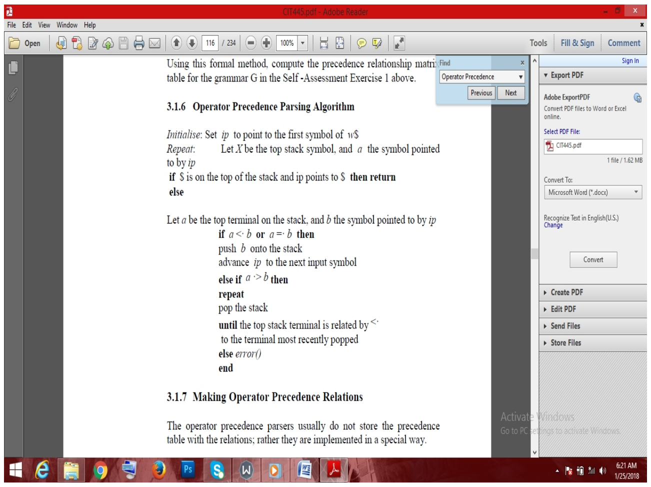
i) Operator precedence (ii) operator Grammar (iii) operator precedence e Grammar

An **operator precedence parser** is a bottom-up parser that interprets an operator-precedence grammar.

Operator grammars have the property that no production right side is empty or two or more adjacent non-terminals.

An operator precedence grammar is an -free operator grammar in which the precedence relations define above are disjoint

b) State the operator precedence parser algorithm


Using this formal method, compute the precedence relationship matrix table for the grammar G in the Self -Assessment Exercise 1 above.

3.1.6 Operator Precedence Parsing Algorithm

```
Initialise: Set ip to point to the first symbol of w$
Repeat:     Let X be the top stack symbol, and a the symbol pointed
to by ip
 if $ is on the top of the stack and ip points to $ then return
 else

Let a be the top terminal on the stack, and b the symbol pointed to by ip
            if a <· b or a =· b then
            push b onto the stack
            advance ip to the next input symbol
            else if a ·> b then
            repeat
            pop the stack
            until the top stack terminal is related by <·
             to the terminal most recently popped
            else error()
            end
```

3.1.7 Making Operator Precedence Relations

The operator precedence parsers usually do not store the precedence table with the relations; rather they are implemented in a special way.

7a)Suppose we have a grammar G:

EE + T

E T

T T*F

TF

F (E)

F a

SEPTEMBER/OCTOBER 2015 CIT 445 Principles & Techniques of Compilers

1) Define formal language) 3 marks

A **formal language** is a set of *words*, i.e. finite strings of *letters*, *symbols*, or *tokens*.

b. State three of the uses of formal languages) 3 marks

c. What is a translator?) 2marks

A translator is a programme that takes as input a programme written in one programming language (the source language)and produces as output a programme in another language (the object or target language).

d. Why do we need a translator?) 3 marks

We need translators to overcome the rigour of programming in machine language, which involves communicating directly with a computer in terms of bits, register, and primitive machine operations.

e. Enumerate the functions performed by the lexical analyser) 4 marks

The main function is to simplify the overall design of the compiler. Lexical analyser also performs other functions such as keeping track of line numbers, producing an output listing if necessary, stripping out white space (such as redundant blanks and tabs), and deleting comments.

2a) Briefly describe the operation performed by the shift-reduce parser) 6 marks

b) Given the context-free grammar G below:

G: E E + E

E E * E

E (E)

E **id**

State the steps performed by the shift-reduce parser when analyzing the input string:

$\mathbf{id}_1 + \mathbf{id}_2 * \mathbf{id}_3$) 8 marks

JULY 2017 EXAMINATION CIT 445 Principles & Techniques of Compilers Marking Scheme

1) a) With the aid of illustrative diagram describe the phases of a compiler. (14 marks)

b) Consider the grammar G below:

G: E E + T / T

T T * F / F

F (E) / i

Generate the non-left recursive version of the grammar (5 marks)

Answer:

$E \Rightarrow E + T$

$\Rightarrow T + T$

$\Rightarrow F + T$

$\Rightarrow id + T$

$\Rightarrow id + T * F$

$\Rightarrow id + F * F$

⇒ id + id * F

⇒ id + id * id

c) Explain what is meant by the term Viable Prefix? (3 marks)

A viable prefix is so called because it is always possible to add terminal symbols to the end of a viable prefix to obtain a right sentential form.

2a. What are the benefits of LR parsing?) 4 marks

i. it detects syntactic errors when the input does not conform to the grammar as soon as possible.

ii. It is more general than operator precedence or any other common shift-reduce techniques

iii. LR parsing also dominates the common forms of top-down parsing without backtrack.\

iv. LR parsers can be constructed to recognise virtually all programming language constructs for which context-free grammars can be written

b) List the common techniques for building tables for an "LR" parser stating the characteristics of each? (6 marks)

Simple LR

i. smallest class of grammars ii. smallest tables (number of states) iii. simple, fast construction

Canonical LR

i. full set of LR(1) grammars ii. largest tables iii. slow, expensive and large construction

Lookahead LR

i. intermediate sized set of grammars ii. same number of states as SLR(1)

iii. canonical construction is slow and large iv. better construction techniques exist

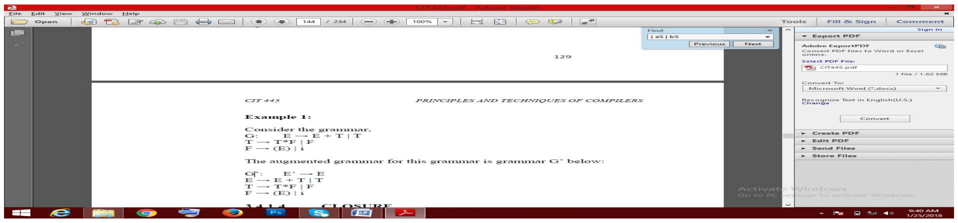
c) Consider the grammar,

G: E → E + T | T

T → T*F | F

F → (E) | i

What is the augmented grammar for this grammar (2 marks)


129

CIT 445 PRINCIPLES AND TECHNIQUES OF COMPILERS

Example 1:

Consider the grammar,
G: $E \longrightarrow E + T \mid T$
$T \longrightarrow T*F \mid F$
$F \longrightarrow (E) \mid i$

The augmented grammar for this grammar is grammar G' below:

G': $E' \longrightarrow E$
$E \longrightarrow E + T \mid T$
$T \longrightarrow T*F \mid F$
$F \longrightarrow (E) \mid i$

3 a) Briefly describe the operation performed by the shift-reduce parser (6 marks)

i. shift - the next input symbol is shifted onto the top of the stack

ii. *reduce* - the parser knows the right end of the handle is at the top of thestack. It must then locate the left end of the handle within the stack and decide with what nonterminal to replace the handle

iii. *accept* - the parser announces successful completion of parsing

iv. *error* - the parser discovers that a syntax error has occurred.

b) State three of the uses of formal languages (3 marks)

- Formal languages are often used as the basis for richer constructs endowed with semantics.

- In computer science they are used, among other things, for the precise definition of data formats and the syntax of programming languages.

- Formal languages play a crucial role in the development of compilers, typically produced by means of a compiler, which may be a single programme or may be separated in tools like lexical analyser generator s (e.g. lex), and parser generators (e.g. yacc).

- Formal languages are also used in logic and in foundations of mathematics to represent the syntax of formal theories.

c) Why do we need a translator? (3 marks)

We need translators to overcome the rigour of programming in machine language, which involves com municating directly with a computer in terms of bits, register, and primitive machine operations.

4a) What is a translator? (2marks)

A translator is a programme that takes as input a programme written in one programming language (

the source language)and produces as output a programme in another language (the object or target language).

b) Consider the grammar,

G: S → a | aS | bS

Find the LR(0) items for this grammar   (10 marks

5a) Enumerate the functions performed by the lexical analyser (4 marks)

The main function is to simplify the overall design of the compiler. Lexical analyser also performs other functions such as keeping track of line numbers, producing an output listing if necessary, stripping out white space (such as redundant blanks and tabs), and deleting comments.

b) Enumerate any three of the errors which can be detected during lexical analysis (6 marks)

Strange characters: Some programming languages do not use all possible characters, so any strange ones which appear can be reported.

Long quoted strings I: Many programming languages do not allow quoted strings to extend over more than one line; in such cases a missing quote can be detected.

Long quoted strings II: If quoted strings can extend over multiple lines then a missing quote can cause quite a lot of text to be 'swallowed up' before an error is detected.

Invalid numbers: A number such as 123.45.67 could be detected as invalid during lexical analysis (provided the language does not allow a full stop to appear immediately after a number)

6) a) Define formal language (3 marks)

A **formal language** is a set of *words*, i.e. finite strings of *letters*, *symbols*, or *tokens*.

b) Find FOLLOW of all the nonterminal symbols in the non-left recursive version of the grammar (9 marks)