# **D Compiler Design Question Compilation**

### **(Questions from 2013-2016)**

## **Question 1**

<E> -> <E> + <T>   
<E> -> <T>   
<T> -> <T> \* <P>   
<T> -> <P>   
<P> -> (<E>)   
<P> -> IDENT  
<P> -> CONST

1. Extend the grammar to include subtraction, division, exponentiation (where exponentiation is right associative and has higher precedence than multiplication and division). [6 marks]

<E> -> <E> + <T>

<E> -> <E> - <T>   
<E> -> <T>   
<T> -> <T> \* <P>

<T> -> <T> / <P>   
<T> -> <F>

<F> -> <P> ^ <F>

<F> -> <P>  
<P> -> (<E>)   
<P> -> IDENT  
<P> -> CONST

1. Produce an equivalent LL(1) grammar. [6 marks]

é = empty, # - endmarker

<E> -> <T><ELIST>

<ELIST> -> +<T><ELIST>

<ELIST> -> -<T><ELIST>

<ELIST> -> é

<T> -> <F><TLIST>

<TLIST> -> \*<F><TLIST>

<TLIST> -> /<F><TLIST>

<TLIST> -> é

<F> -> <P><EXPO>

<EXPO> -> ^<F>

<EXPO> -> é

<P> -> (<E>)

<P> -> IDENT

<P> -> CONST

1. Compute the selection sets of the LL(1) grammar. [8 marks]

|  |  |  |
| --- | --- | --- |
| FIRST | FOLLOW | SELECT |
| { (, IDENT, CONST} |  | { (, IDENT, CONST} |
| { + } |  | { + } |
| { - } |  | { - } |
|  | {#} | {#} |
| { (, IDENT, CONST} |  | { (, IDENT, CONST} |
| { \* } |  | { \* } |
| { / } |  | { / } |
|  | {+, -, # } | {+, -, # } |
| { (, IDENT, CONST} |  | { (, IDENT, CONST} |
| ^ |  | ^ |
|  | {+, -, \*, / # } | {+, -, \*, / # } |
| { ( } |  | { ( } |
| {IDENT} |  | {IDENT} |
| {CONST} |  | {CONST} |

1. Add attributes and evaluation rules to the productions for the grammar for the grammar from (1.) and also for the grammar from (2.) so that each of their starting non-terminal symbols will have a single synthesised attribute whose value is that of the expression being parsed. [8 marks]
2. Design an interpreter for arithmetic expressions by adding attributes and erecursive descent parser:valuation rules to the productions to the LL(1) grammar obtained in (2.) so that the starting non-terminal symbol will have a single synthesised attribute whose value is that of the expression being parsed. [8 marks]
3. Show how both of the grammars produce the same translation by drawing fully decorated derivation-trees for the expression 2^3^2+A/B where A and B have values 256 & 32 respectively, and ^ is the (right-associative) exponentiation operator.
4. [12 marks]
5. Rewrite the productions from the LL(1) grammar in simple assignment form.
6. [8 marks]
7. Design the corresponding stack replacement operations for an augmented pushdown machine for the LL(1) grammar.
8. [12 marks]
9. Show the contents of the augmented pushdown machine's stack at each step along the way while parsing the expression (A-B)\*C.
10. [8 marks]
11. Discuss the relationship between the first, follow and select sets in the context of LL(1) parsing, and explain how follow sets are calculated.

https://www.jambe.co.nz/UNI/FirstAndFollowSets.html

First(α) is the set of terminal symbols that can occur at the start of strings derived from α.

Follow<x> is the set of terminal symbols that can immediately follow an <x> in any intermediate string derivable from <s> ㅓ.

For productions of the form <A> -> aα, where a is a terminal symbol and α is a string of terminal and non-terminals.

SELECT(<A> -> aα) = {a}.

SELECT(<A> -> ϵ) = FOLLOW(<A>).

1. [10 marks]
2. By computing the selection sets for the productions from the following context free grammars, determine which (if any) of them are LL(1):

* A

<S> -> <A> b   
<A> -> <B>   
<A> -> a   
<A> -> ϵ  
<B> -> b   
<B> -> ϵ

* B

<S> -> <A> <B> c   
<A> -> a   
<A> -> ϵ  
<B> -> b   
<B> -> ϵ

* C

<S> -> <A> <B> <B> <A>   
<A> -> a   
<A> -> ϵ  
<B> -> b   
<B> -> ϵ

1. Explain why the following two productions are not LL(1) and show how they may be handled by a recursive descent parser:

<statement> -> if <condition> then <statement> else <statment>  
<statement> -> if <condition> then <statement>

[10 marks]

1. Design an augmented pushdown machine to compute the value of an arithmetic expression of the form '\* 2 + 16 32' with this L-attributed translation grammar:
2. [25 marks]
3. Design an augmented pushdown machine (including all stack replacements) for an interpreter to compute the value of an arithmetic expression of the form '\* 2 + 16 32' with this L-attributed translation grammar:
4. [25 marks]
5. In relation to error processing, describe the prefix property and outline the differences between local and global error recovery, and explain in detail the function of the recursive descent procedure SKIP\_TO and its use in global error processing.
6. [15 marks: prefix property-2; differences-4; and SKIP-TO-9]

## **Question 2**

1. Design an L-attributed translation grammar production for a basic loop statement of the form:

for i := 1 to n do <stmts...>

where the control variable *i* is assigned successive values running from 1 to n. Describe the function of the action symbols and the information represented by the attributes. Care should be taken to ensure that the generated code will run correctly for all possible values of n.

[40 marks]

1. --Describe the information that should be maintained in the symbol table to record the properties of structured variables (e.g. arrays or records), and show how to design L-attribute translation grammar productions to cater for type and variable declarations of the form:

TYPE  
 CourseData = RECORD  
 code, mark: INTEGER  
 END;  
 StudentInfo = RECORD  
 name: STRING;  
 number: INTEGER;  
 course: ARRAY {} OF CourseData  
 END;  
 VAR  
 Students: ARRAY [1..32] OF StudentInfo;

[40 marks]

1. Describe the information that should be maintained in the symbol table at compile time to record the properties of multi (e.g. one and two) dimensional arrays, and design L-attributed translation grammar productions to parse variable declarations of the general form:

VAR  
 data: ARRAY [1..16] OF INTEGER;  
 table: ARRAY [1..8,1..32] of STRING;

Explain in detail the function of the action symbol, and sketch the symbol table's contents when compiling the variable declarations shown above.

[40 marks]

1. Design some form of switch (or case) statement and explain the underlying reason(s) behind your choice of syntax. Describe the flow of control and issues that might arise in the generation of object code for such statements. Finally, design l-attributed production(s) for your switch statement.
2. [40 marks: language design (including basis for syntax)-16; flow of control-8; attributed translation grammar-8; description of attributes-2; and description of action symbols-6]

## **Question 3s**

1. Design L-attributed translation grammar productions to process constant definitions and variable declarations of the following form:

CONST   
 max = 512;   
 marker = "$";   
VAR   
 ch: CHAR;   
 top: INTEGER;   
 stack: ARRAY [1..max] OF CHAR;

Describe the function of the action symbols and the information represented by the attributes, explain the mapping of arrays and show by example the structure and contents of the symbol table. [40 marks]

1. Discuss the relationship between the output action symbols {labelp and {jumpfp,q}, and describe in detail their use in L-attributed translation grammar productions for IF and REPEAT statements of the form:

<statement> --> IF <condition> THEN <statement>  
<statement> --> REPEAT <statement> UNTIL <condition>

Explain the role of the function "newl" and demonstrate, by example, how the processing of the address field in a generated branch instruction is dependent on the relative position of a {label} and its corresponding {jumpf} action. [40 marks]

1. Discuss the relationship between the output action symbols {labelp} and {jumpp} [or {jumptp,q}/{jumpfp,q}], and demonstrate their use by converting the context free productions

<statement> --> DO <statements> WHILE <condition>  
<statement> --> WHILE <condition> DO <statements>

into equivalent l-attributed translation productions. For each production sketch the flow of control through the generated object code as well as the sequence of actions performed during the parse. Describe the purpose of the function "newl" and show, by example, how the processing of the address field in a generated branch instruction is dependant on the relative position of a {label} and its corresponding {jump}, {jumpt} or {jumpf} action. [40 marks: attributed translation grammar-12; relationship between action symbols-2; sequence and flow control-8; purpose of function newl-2; and discussion about relative position of action symbols and address field-16]