# CSP3341: Programming Languages and Paradigms – Workshop 4

## Version E Answers

**Marks:** Marked out of 28, (7% of unit)

## Workshop Description

This workshop revises and applies topics from Chapters 3 and 4 of the textbook. It consists of a number of small tests and exercises. The content of this workshop is similar to that which may be included in your exam. **This workshop is assessed**. This workshop is to be completed **individually**.

### Task 1 – Grammars, Sentential Forms and Parse Trees (8 marks – 2 marks per part)

**A.** Looking at the BNF grammar for “race”, below, describe the structure of a program in the race language, in English. You may assume “dir” is a direction/instruction and “point” is a checkpoint.

**English description of “race”**

A program in “race” begins with “go”, followed by a list of directions, and ends with “and stop”.

The list of directions can be a single direction, or multiple directions connected by “then”.

Individual directions are in the form of either “from <point> to <point>”, “around <point>” or “to <point>”.

Points, i.e. checkpoints, are named “A”, “B” or “C”.

**Grammar for “race”**

<race> **→ go** <dir\_list> **and stop**

<dir\_list> **→** <dir>

**|** <dir> **then** <dir\_list>

<dir> **→ from** <point> **to** <point>

**| around** <point>

**| to** <point>

<point> **→ A**

**| B**

**| C**

**B.** Identify which of the sentences are not syntactically correct according to the grammar of “race”.

If a program is not valid, highlight the errors in the program.

|  |  |
| --- | --- |
| **Sentences in “race” Grammar** | **Valid?** |
| go to B and stop | **Yes** |
| go from A to B then around C then to A and stop | **Yes** |
| go around B then to A and then to C then from A to B and stop | **No** |
| go from B to A then to C then to A then around B then stop | **No** |
| go to C then to B then from A to B then from A to B and stop | **Yes** |
| go from A around B then to C then to B and stop | **No** |

**C.** Using leftmost derivation and the “race” grammar, derive the statement “go to A then around B then from B to C and stop” into its final sentential form. Show all sentential forms along the way.

**Derivation of “go to A then around B then from B to C and stop”**

NOTE: **bold** indicates the nonterminal that will be replaced in the next form

blue indicates the section of the statement that is new in that form

<race> => go **<dir\_list>** and stop

=> go **<dir>** then <dir\_list> and stop

=> go to **<point>** then <dir\_list> and stop

=> go to A then **<dir\_list>** and stop

=> go to A then **<dir>** then <dir\_list> and stop

=> go to A then around **<point>** then <dir\_list> and stop

=> go to A then around B then **<dir\_list>** and stop

=> go to A then around B then **<dir>** and stop

=> go to A then around B then from **<point>** to <point> and stop

=> go to A then around B then from B to **<point>** and stop

=> go to A then around B then from B to C and stop

**D.** Construct a parse tree of “go to A then around B then from B to C and stop” using the “race” grammar. It should match your derivation from part B of this task.

**Parse Tree of “go to A then around B then from B to C and stop”**

**go** <dir\_list> **and stop**

<dir> **then** <dir\_list>

**to** <point> <dir> **then** <dir\_list>

**A** **around** <point> <dir>

**B** **from** <point> **to** <point>

**B** **C**

### Task 2 – Converting BNF to EBNF (4 marks)

Translate the “race” grammar rules (Task 1) from BNF form to Extended BNF form, using the same notation to that used in the textbook. Your final EBNF grammar should describe the same language as the BNF grammar, and be highly readable.

**EBNF Grammar of “race”**

NOTE: There are numerous valid solutions to this question.

<race> **→ go** <dir> {**then** <dir>} **and stop**

<dir> **→ from** <point> **to** <point>

**| around** <point>

**| to** <point>

<point> **→** (**A** | **B** | **C**)

### Task 3 – Axiomatic Semantics & Weakest Preconditions (6 marks)

Identify the weakest preconditions and postconditions for all statements in the sequence. Assume only integer values. Also identify the weakest precondition for the “if” statement. Write the weakest preconditions/postconditions inside the braces provided.

**Weakest Precondition – If**

{ **b > 25** }

if ( a > 10 )

b = 2 \* b

else

b = b - 5

{ **b > 20** }

**Weakest Precondition - Sequence**

{ **y < x + 4** }

x = y - x + 1

{ **x < 5** }

y = 25 - (x \* 5)

{ **y > 2** }

x = 5 \* y;

{ **x > 12** }

### Task 4 – Lexical Analysis (2 marks)

The table below represents the tokens and character classes in a lexical analyser.

|  |  |
| --- | --- |
| **Tokens** | **Description** |
| VAR | LETTER, followed by one or more DIGIT or LETTER |
| INT | one or more DIGIT |
| FLOAT | one or more DIGIT, then DEC\_PNT, then one or more DIGIT |
| ADD\_OP | the + character |
| SUB\_OP | the - character |
| MULT\_OP | the \* character |
| DIV\_OP | the / character |
| ASSIGN\_OP | the = character |
| DEC\_PNT | the . character |
| L\_PAREN | the ( character |
| R\_PAREN | the ) character |
| SEMICOLON | the ; character |
|  |  |
| **Character Classes** | **Description** |
| LETTER | all upper and lower case letters ([A-Za-z]) |
| DIGIT | all numbers ([0-9]) |

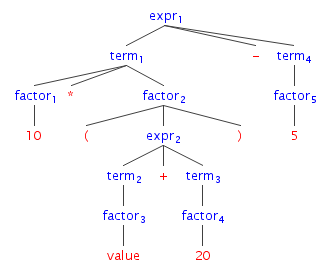
Using the definitions in the table, simulate the output (to the syntax analyser) of the lexical analyser given the following string of input. You may assume (and omit) the presence of EOF after the input.

**testProg = (12 + value1) \* 20.50 – value2;**

|  |  |
| --- | --- |
| **Token** | **Lexeme** |
| VAR | testProg |
| ASSIGN\_OP | = |
| L\_PAREN | ( |
| INT | 12 |
| ADD\_OP | + |
| VAR | value1 |
| R\_PAREN | ) |
| MULT\_OP | \* |
| FLOAT | 20.50 |
| SUB\_OP | - |
| VAR | value2 |
| SEMICOLON | ; |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

### Task 5 – Syntax Analysis (Recursive-Descent Parsing) (4 marks)

|  |
| --- |
| **NxtToken:** INT **NxtLexeme:** 10 |
| **Enter** expr |
| **Enter** term |
| **Enter** factor |
| **NxtToken:** MULT\_OP  **NxtLexeme:** \* |
| **Exit** factor |
| **NxtToken:** L\_PAREN  **NxtLexeme:** ( |
| **Enter** factor |
| **NxtToken:** VAR  **NxtLexeme:** value |
| **Enter** expr |
| **Enter** term |
| **Enter** factor |
| **NxtToken:** ADD\_OP  **NxtLexeme:** + |
| **Exit** factor |
| **Exit** term |
| **NxtToken:** INT  **NxtLexeme:** 20 |
| **Enter** term |
| **Enter** factor |
| **NxtToken:** R\_PAREN  **NxtLexeme:** ) |
| **Exit** factor |
| **Exit** term |
| **Exit** expr |
| **NxtToken:** SUB\_OP  **NxtLexeme:** - |
| **Exit** factor |
| **Exit** term |
| **NxtToken:** INT  **NxtLexeme:** 5 |
| **Enter** term |
| **Enter** factor |
| **NxtToken:** EOF  **NxtLexeme:** EOF |
| **Exit** factor |
| **Exit** term |
| **Exit** expr |

Below is a parse tree of “10 \* (value + 20) - 5”, using the same grammar that is used in the textbook’s section on Recursive-Descent Parsing (4.4.1). Complete the table next to the tree, illustrating a trace of the parser’s progress and use of expr, term and factor functions.

Tree generated in phpSyntaxTree - <http://ironcreek.net/phpsyntaxtree/>

Grammar from Section 4.4.1 of textbook:

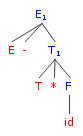
<expr> → <term> {(**+** | **-**) <term>}

<term> → <factor> {(**\*** | **/**) <factor>}

<factor> → **id** | **int\_constant** | **(** <expr> **)**

### Task 6 – Syntax Analysis (Bottom-Up Parsing), Phrases and Handles (4 marks)

Below left are a simple bottom-up grammar and a parse tree of a right sentential form of a derivation using the grammar. Identify and list the phrases, simple phrases and handle.



**Bottom-Up Grammar**

E **→** E **–** T **|** T

T **→** T **\*** F **|** F

F **→ (**E**)** **| id**

Right sentential form:

**E – T \* id**

**Handle**

Place a around the handle

**Simple Phrases**

id (from F)

**Phrases**

E – T \* id (from E1)

T \* id (from T1)

id (from F)