Curtin University — Department of Computing

Programming Languages / Advanced Computing Topics
(COMP2007 / COMP5004)

Semester 2, 2017

Assignment

Due: Friday 19th October

Weight: 20% of the unit mark

Hi! Welcome to the PL2017 assignment specification! Please read this document in it's entirety before starting, especially if you are an ACT student. There is a second part you need to be aware of after the Syntax Diagrams. Good luck!

Objective

This assignment will give you some practical experience in building parsers (and theoretically compilers) with Yacc and Lex, and also with working with EBNF specifications in general.

Task 1 – EBNF Specification

Attached to this document is a series of syntax diagrams specifying a language called PL-NEXT. You must translate these into EBNF statements. Your specification must be clear and consistent, using the same set of extentions throughout your program.

You will also be required to write a series of grammars (programs) that will test each aspect of the language. For each EBNF statement, you should have two corresponding programs, one that will compile under PL-NEXT and one that will not. You may combine the correct grammars into a single program that will test every aspect of the language. You may *not* combine the incorrect grammars.

Task 2 – Yacc and Lex Parser

You will need to design a Yacc and Lex parser and syntax checker for PL-NEXT (using bison and flex). This will need to be an executable called PL-NEXT-SYNTAX that will check for correct PL-NEXT grammars.

You will need to parse an input stream, ie, the console or a file (containing, for example, the test grammars/programs you wrote in Task 1), and convert it into reserved words, numbers, and identifiers using flex. This tokenised input will then be passed to bison, which will check to see if the tokens form a valid PL-NEXT program.

You should translate your EBNF to Yacc and Lex code. A correct PL-NEXT program should be parsed by your PL-NEXT-SYNTAX program successfully without errors. Your program should exit with a syntax error if there is an error in the syntax. In order to aid readability, your program should display all matched symbols to the screen as the input is being parsed.

Requirements

You are required to submit the following items for assessment:

- The typed EBNF description of PL-NEXT derived from the syntax graphs provided.
- The grammars for testing each of your EBNF statements.
- A typed report that contains a description of how you built the parser and syntax checker for the PL-NEXT language. The report must be presented in a professional manner.
- Fully documented YACC and Lex source code. Do NOT supply any object or executable files.
- A script to build the final executable.

Failure to provide any of these (or using the wrong executable name) may result in significant mark deductions or for your assignment to be treated as not reasonably attempted.

The script will be executed in the same directory as your source files, and should create an executable, which is to be named PL-NEXT-SYNTAX and must run on the Linux machines.

Once compiled, the program will be run as so: `./PL-NEXT-SYNTAX < test.txt`

Marking

Marks are assigned to EBNF [30] and the yacc and lex source code [10,10]. The script given to build the executable will be used and the tests performed on the executable covers the remaining marks [50]. If the code does not compile using the provided script (or not script is provided), the executable testing marks are lost.

All work is to be submitted via Blackboard in a single archive of type .tar.gz. Any archives not of .tar.gz will be considered not submitted and will result in a mark of zero.

You are responsible for ensuring that your submission is correct and not corrupted. You may make multiple submissions, but only your newest submission will be marked. The late submission policy (see the Unit Outline) will be strictly enforced.

Note: Making a reasonable attempt at the assignment is one of the conditions of passing the unit, as stated in the unit outline. I expect to see a full BNF/EBNF specification, as well as source files that can detect and parse at least some of the specification, compile and can run. If the submission is not up to this standard it will be deemed not to be a reasonable attempt.

Note PS: ACT students, I expect to see a reasonably attempted essay too. See ”Task for ACT Students Only” at the end of this document

Academic Misconduct – Plagiarism and Collusion

This is an assessable task, and as such there are strict rules. You must not ask for or accept help from *anyone* else on completing the tasks. You must not show your work to another student enrolled in this unit who might gain unfair advantage from it.

These things are considered **plagiarism** or **collusion.**

Staff can provide assistance in helping you understand the unit material in general, but nobody is allowed to help you solve the specific problems posed in this document. The purpose of the assignment is for *you* to solve them *on your own.*

Please see Curtin's Academic Integrity website for information on academic misconduct (which includes plagiarism and collusion).

The unit coordinator may require you to provide an oral justification of, or to answer questions about, any piece of written work submitted in this unit. Your response(s) may be referred to as evidence in an Academic Misconduct inquiry. In addition, your assignment submission may be analysed by Turnitin and/or other systems to detect plagiarism and/or collusion.

Task for ACT Students Only

This part of the assignment is for the students of Advanced Computing Topics *only* and should not be done by PL students. If you are a PL student **IGNORE THIS PART**

Objective

The objective of this section is to give ACT students experience with dealing with languages they ***don't*** know, and extends their knowledge of the unit by applying the principles and requirements of programming languages to a language they don't know.

Writing an Essay

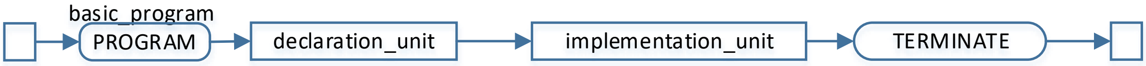
Choose one language out of the following; BASIC (not to be confused with the newer visual basic), FORTH, COBOL, or Common Lisp . For this language, discuss its Criteria (Readability, Writablity and Reliability) in terms of the characteristics discussed in the textbook. You should write one or two paragraphs for each characteristic (e.g., orthogonality), stating the degree to which the language supports this. You should summarise by concluding the degree to which the language meets the three main criteria.

Offset

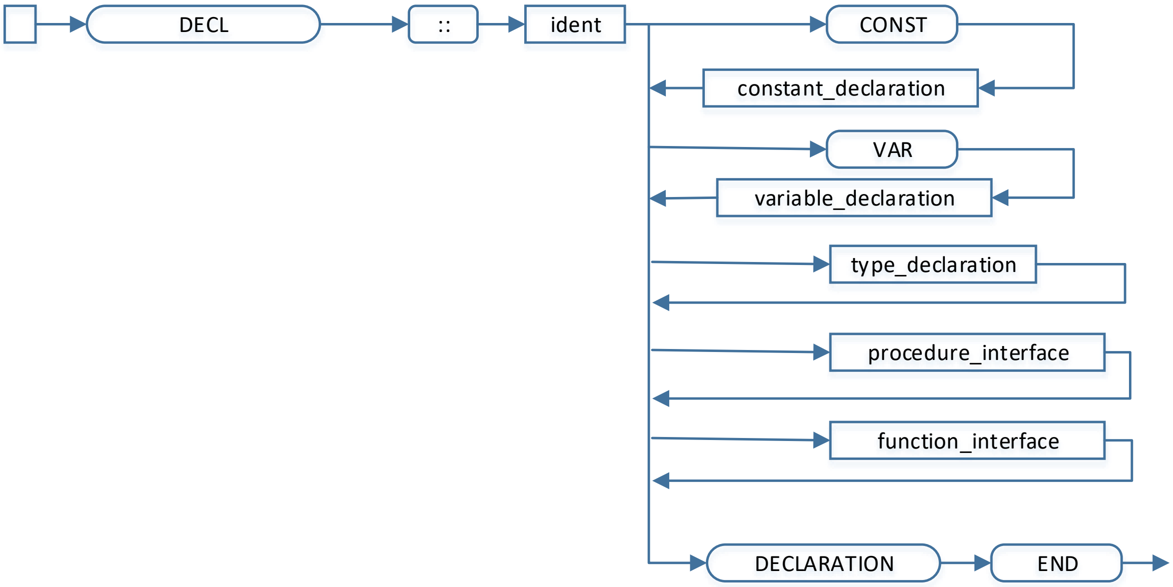
In order to partially reduce your workload in order to allow for this, you may ignore the following part of the specification in the main assignment:

- declaration_unit
- any parts that are used only in the above

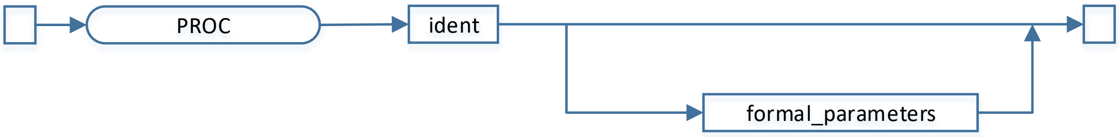
You will be marked out of 100 as usual for the main assignment, and your mark will then be scaled to 75, with the marks from this part making up the other 25. The 25 marks for this part of the assignment will be roughly 2 marks for each characteristic (less for a characteristic that is not supported at all by a particular language, in which case the marks are spread across more important characteristics), 2 marks per criteria and one mark for the overall work basic_program



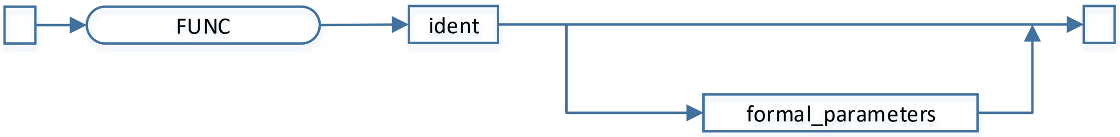
declaration_unit



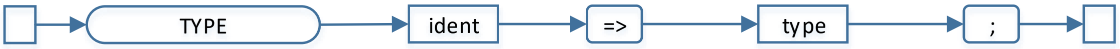
procedure_interface



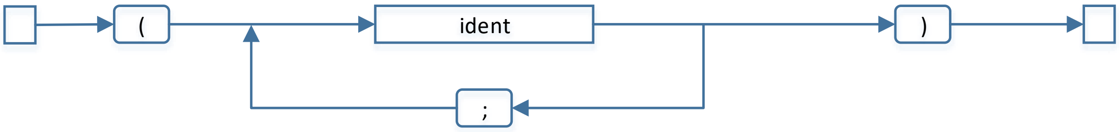
function_interface



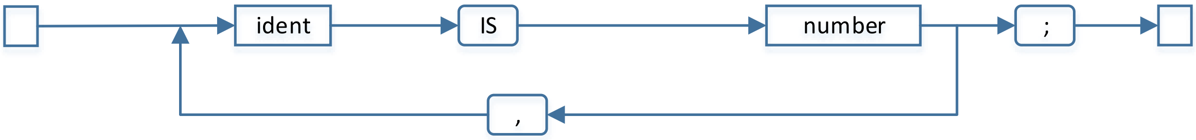
type_declaration



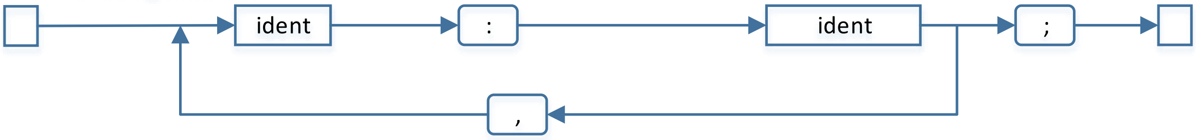
formal_parameters



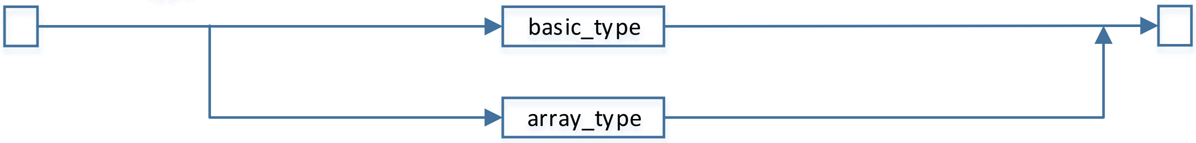
constant_declaration



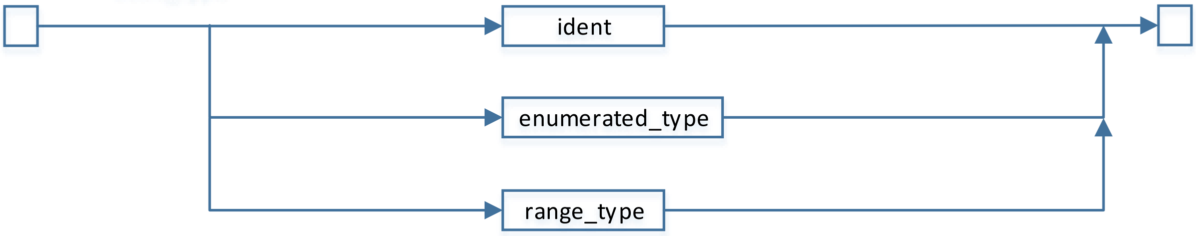
variable_declaration



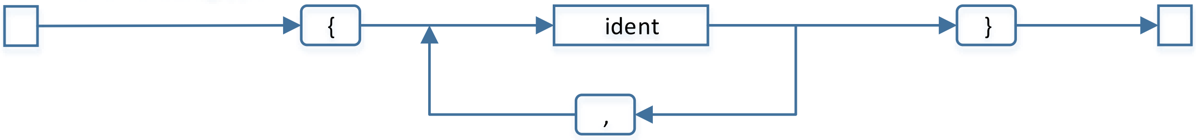
type



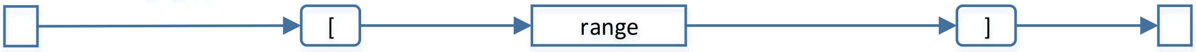
basic_type



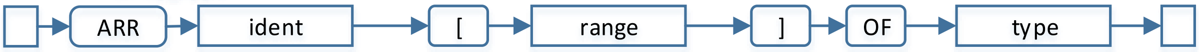
enumerated_type



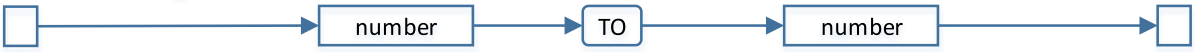
range_type



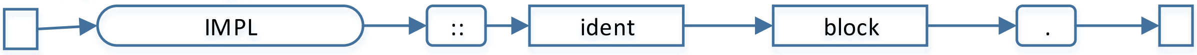
array_type



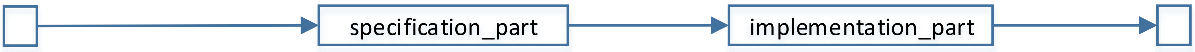
range



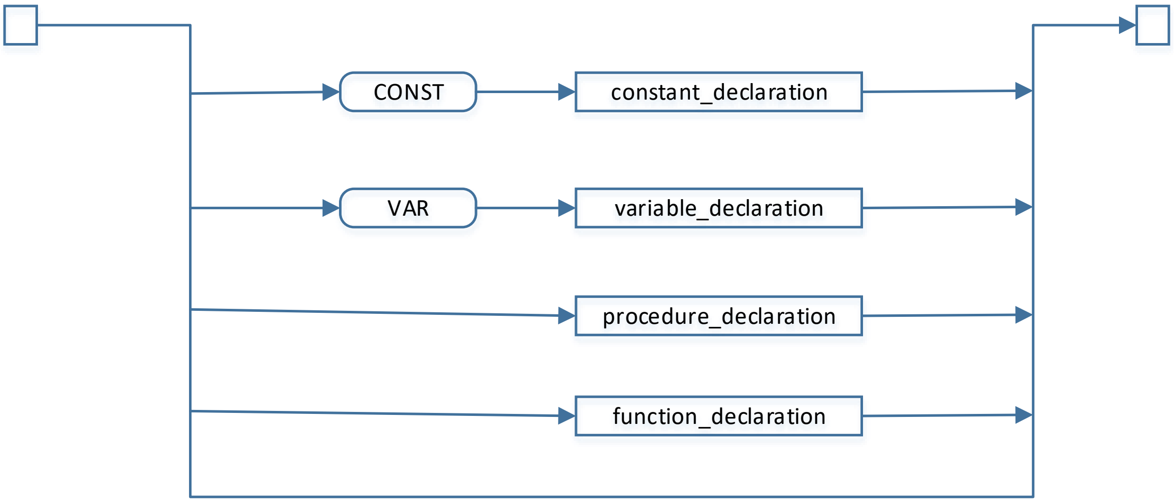
implementation_unit



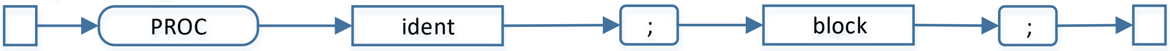
block



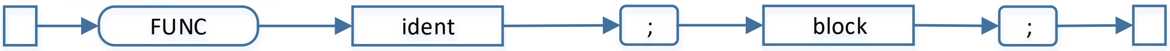
specification_part



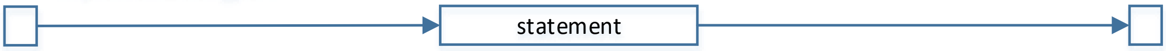
procedure_declaration



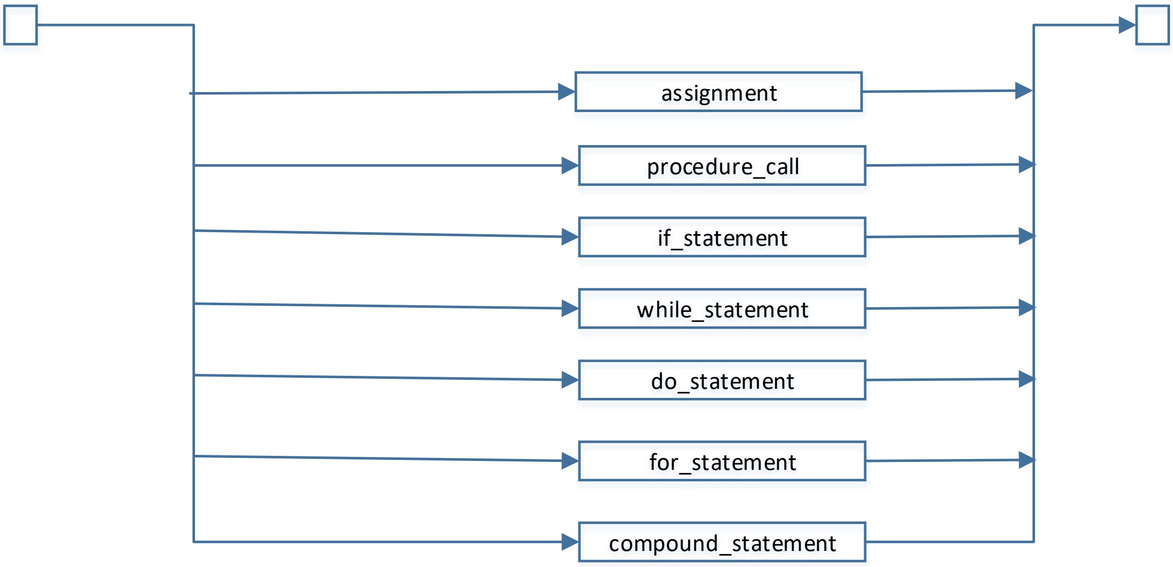
function_declaration



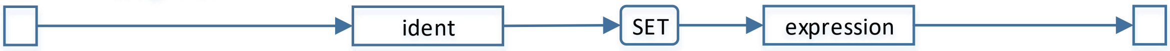
implementation_part



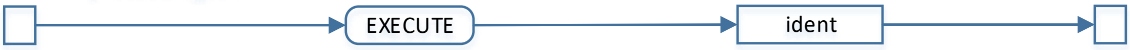
statement



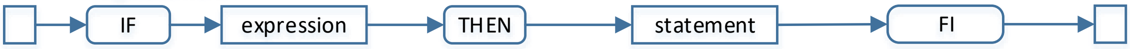
assignment



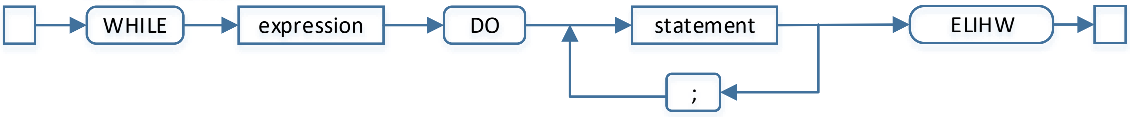
procedure_call



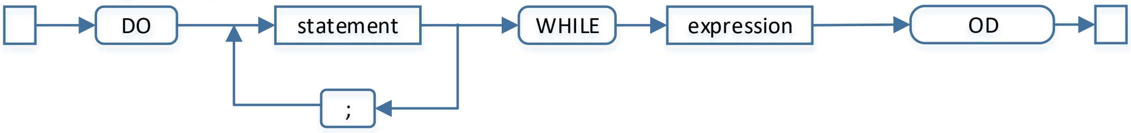
if_statement



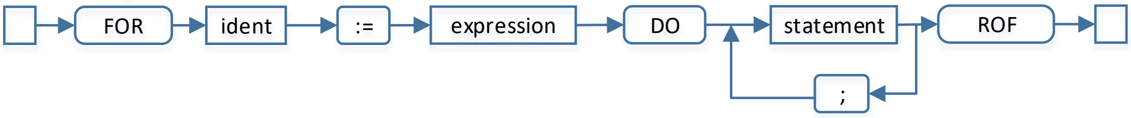
while_statement



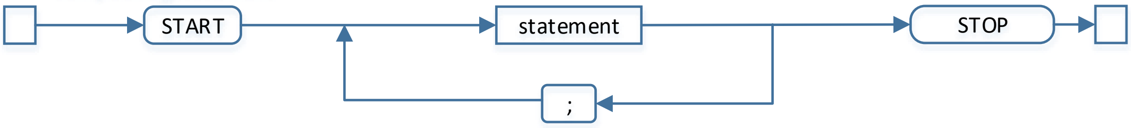
do_statement



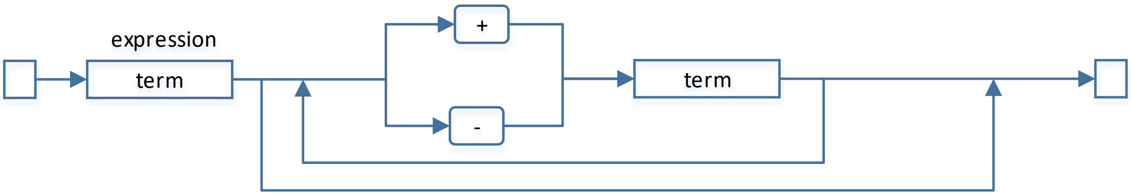
for_statement



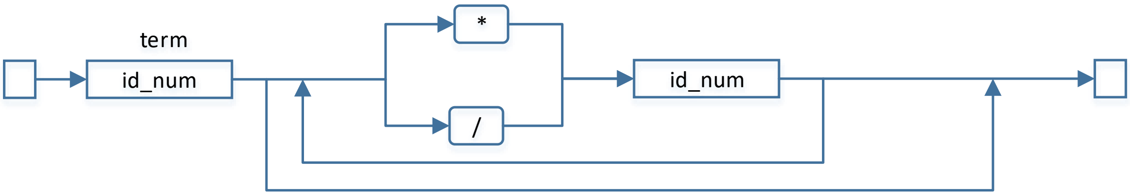
compound_statement



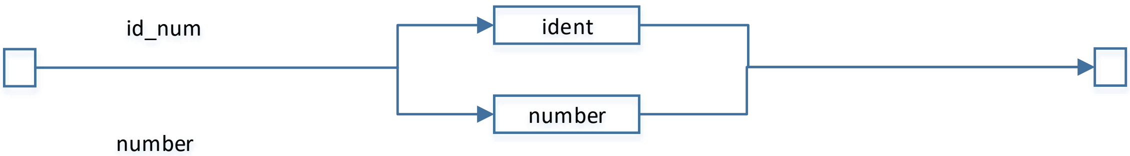
expression



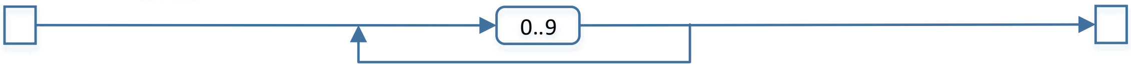
term



id_num



number



ident