Monsoon 2023 CS1319 A3

Bhumika Mittal

Introduction

The syntax analyser takes in a stream of tokens produced by the lexer and arranges them into a syntax tree that represents the grammatical structure of the token stream, and so, the program. A syntax tree is of the form such that interior nodes of subtrees are the operator tokens and their children are the operands.

Design Choices

We follow the given Phrase Structure Grammar of nanoC to design our parser. The following changes are made to accomodate for the some non-terminals which are optional. Given a non-terminal,

argument-expression-list

which is shown as optional on the right-hand-side of a production rule as:

postfix-expression: postfix-expression (argument-expression-list$_{opt}$ *)*

We introduce a new non-terminal, *argument-expression-list*$_{opt}$, and a pair of new productions:

argument-expression-list-opt: argument-expression-list $|\epsilon$

and change the above rule as:

postfix-expression: postfix-expression (argument-expression-list-opt)

Similarly, we change all the non-terminals which are optional on the right-hand-side of some production rule by introducing a new non-terminal and changing the rule a bit.

We also introduce a new production rule for constant as follows:

constant: INTEGER_CONSTANT | *CHARACTER_CONSTANT*

The output format is as follows: For a given production rule,

1. If it is present in as one of the rules in the specifications, print the **left-hand side non-terminal**.
2. If it is one of the rules you have defined yourself, print nothing.

To avoid errors or any unexpected behaviour, we replace - with _ in the name of non-terminals. Example - *postfix-expression* is replaced with *postfix_expression* We've also made modifications to our lexer design that was initially created for the previous assignment. Now, every keyword and punctuator is sent as an individual token to yyparse for distinct identification, allowing it to take the necessary actions accordingly.