Assignment

Part B - Creating an Editorial Parser with the Parser BISON generator

Import

The role of the editorial analyst – SA (syntax analyzer) is to determine the syntactic correctness of the source code structures based on the language grammar. The input of the editorial SA (parser) is a sequence of verbal units (tokens), produced by the lexical analyzer LA after splitting (lexical analyzer), after recognizing the lexemes of the input string. Verbal units are returned by the LA not all at once but on continuous requests of the SA during the check it performs.

To develop the editorial parser through the BISON generator, study them INTRODUCTION TO BISON.pdf and Introduction to BISON - 2.pdf in the catalog 3-WORKSHOP » 3-BISON in the Open Class platform. Then for more FLEX & BISON Code examples information study the manual FLEX & BISON Guide.

After completing the assignment, you should be able to:

- design production rules with the BNF/EBNF notation for the description of the grammar of a programming language
- write bison code to generate a component-by-component independent editorial of semantic analysis
- adjust the flex codes to work together to create flex and bison to create integrated analysis environment
- you can rich the grammar of a language so that it can recognize valid and semantic words

The purpose of this paper is to gain relevant experience in describing grammars through BNF/EBNF semantics and in particular to and become familiar with the development of syntaxes. . To develop your editorial parser you will use t use the BISON generator following the instructions posted

Before proceeding to the individual tasks tasks, they must be described, with the help of , with the help BNF/EBNF notation, the exact language, i.e. the rules to create, the production rules of its grammar.

PART B-1: Developing bison code to create an independent SA **In this part you will first need to complete the simple-bison-code.y sample code from the compressed file named simple-bison-code.zip that you will find inside the folder JOB ANNOUNCEMENTS of folder 4-Material for Assignments.**

Once you understand how the generator works be sure to debug and complete the parser by replacing "FILL ME" with the correct bison code to get as many lab language expressions recognized as possible.

Be sure to thoroughly test the SA to be generated, using as input your own source code as an input file that will contain both correct and incorrect expressions of the language. When an expression is recognized it should appear in the output file numbered.

Attention: This step is designed for your own practice without having to submit anything related to eClass or include this step in your final deliverable!

PART B-2: Linking flex code with bison code

In this step you must:

1. **Include in the bison code you prepared in Part B-1 any expressions of the grammar that you have not already included**
2. **Make the appropriate adaptation of the bison code so that your parser can to work with a verbal parsing routine to be produced through flex**
3. **Make the appropriate adaptation of the flex code that you prepared in Part A-3 so that instead of creating an independent parser, a parsing routine is created for the parser that will result from the adaptation of the flex code**
4. **Follow the lab instructions to create a Makefile that will allow your code to automatically compile and run to create a parser in conjunction with a parsing routine**
5. **At the end of the syntactic analysis carried out by the SA, the result of the analysis of the input source code must be displayed (correctly recognized words, correctly recognized expressions and whether it contained verbal and/or syntactical errors and how many), strictly following the syntax:**

CORRECT WORDS: <number of correct words>

CORRECT EXPRESSIONS: <number of correct expressions>

WRONG WORDS: <number of wrong words>

WRONG EXPRESSIONS: <number of incorrect expressions>

Caution: You have not yet been taught all the necessary material for counting incorrect expressions. For part B-2 you can optionally, using the material available in the Open eClass for the bison tool, try to implement this requirement using the panic method. In any case, the counting of incorrect expressions is only required for part B-3 of the paper, which is also the only deliverable for part B of the semester paper.

Thoroughly test the SA that will be generated using as input one or more source code files that will contain both correct and incorrect words and expressions of the language. The completeness of the above control is a key element in the evaluation of the work.

Attention: This step is designed to properly prepare/implement your SA (started from part B-1) by connecting it to the LA of part A-3. Although you do NOT need to submit anything at this stage, all of the materials you prepare for part B-2 will need to be submitted along with the additional part B-3 prompts in the next lab class!

PART B-3: Handling verbal and syntactic warning errors Handling verbal and syntactic warning errors requires the enrichment of grammar rules.

To achieve the above you should:

- **complete your implementation for counting incorrect expressions by now definitely using the panic method in your SA. If a structure in the input file cannot be recognized as a valid expression, then the error handling routine SyntaxError must be called to handle the problem with the panic method and the fatal errors counter incremented by 1, while parsing to continue at the beginning of the next input (command) string.**

- **augment the grammar rules you have created in the bison code with rules for recognizing common syntax warning errors so that it can recover from errors and the parser continues recognition at the beginning of the next input (command) string, displaying in the output file the appropriate error messages and incrementing the corresponding error counters.**

- **enrich the flex code with additional rules for detecting and handling warning errors, appropriately generating corresponding regular expressions, displaying appropriate error messages in the output file, and incrementing corresponding error counters.**

Delivery of final work (part B-3 deliverable)

Submit the following in compressed form, based on additional instructions given to you in the lab:

1. **the issue of your work documentation in Word, Writer or PDF format with the following:**
   - o **cover (with names of team members, team number and section), table contents, introductory text**
   - o **documentation of grammar (for verbal, syntactic, semantic analysis) and of design**
   - o **documentation of the code given as input to flex and that given as input to bison** o **presentation of exhaustive tests with**

     annotated results and final comments on of work
   - o **a detailed report of the responsibilities of all team members as well as a name report of the members who left your team (if any) and in which part of the work this happened**

   - o **explicit reporting if your code compiles ("passes" the bison, flex, and gcc tools) successfully or not, if there are any warnings, and if it finally executes correctly or not**

2. **the (fully commented) codes for flex (.l file) and bison (.y file)**

3. the Makefile that will allow the code to be compiled and run automatically
like the one provided in the template code you downloaded

4. (optional) a separate Word, Writer or PDF file with the detailed corrections (and only these) of the previous assignments (A-2 and A-3). Attention: Complete resubmission of previous deliverables will not be evaluated!

5. your exhaustive tests file (or files) and the corresponding output file (or files). It is recalled that the accompanying input - output files (input.txt - output.txt) should include several recognition cases for all language structures as well as cases of invalid structures that the SA should deal with using the panic method (or if this is not possible, with a syntax error message)

6. (optional) if you consider it necessary, a note (plain text file) with possible observations-problems that the work reviewer should definitely notice before or during the execution of your code

Note: You can optionally -- and after first completing the basic requirements of the work -- proceed with the implementation and documentation of the following: symbol table, abstract syntax tree, intermediate code.

Documentation Guidelines: Write a documentation issue with a cover page, table of contents, introduction, body of documentation, exhaustive test runs, and presentation of results properly annotated for understanding.

Note: Job

submissions with a zip file name that does not begin with a group code WILL NOT BE DONE RECEIVERS.

It is also clarified that where code or results are required:

1. No Greek will be accepted, only Greek (or only proper English). 2. Image snapshots / captures will not be accepted, but only properly formatted text (regarding line wrapping, so that it is visible and readable).

3. Test results should be annotated so they can be understood. It is noted that the tests must be exhaustive to certify the correctness and completeness of the code.

4. In any case, you should explicitly state in the documentation whether or not your code compiles and whether or not it produces correct/acceptable results. Additionally, any problems or deficiencies in execution must be reported.

To better understand the assessment criteria for your work, you should carefully read the assessment rubric found in eClass.

Last update: 2024.05.06