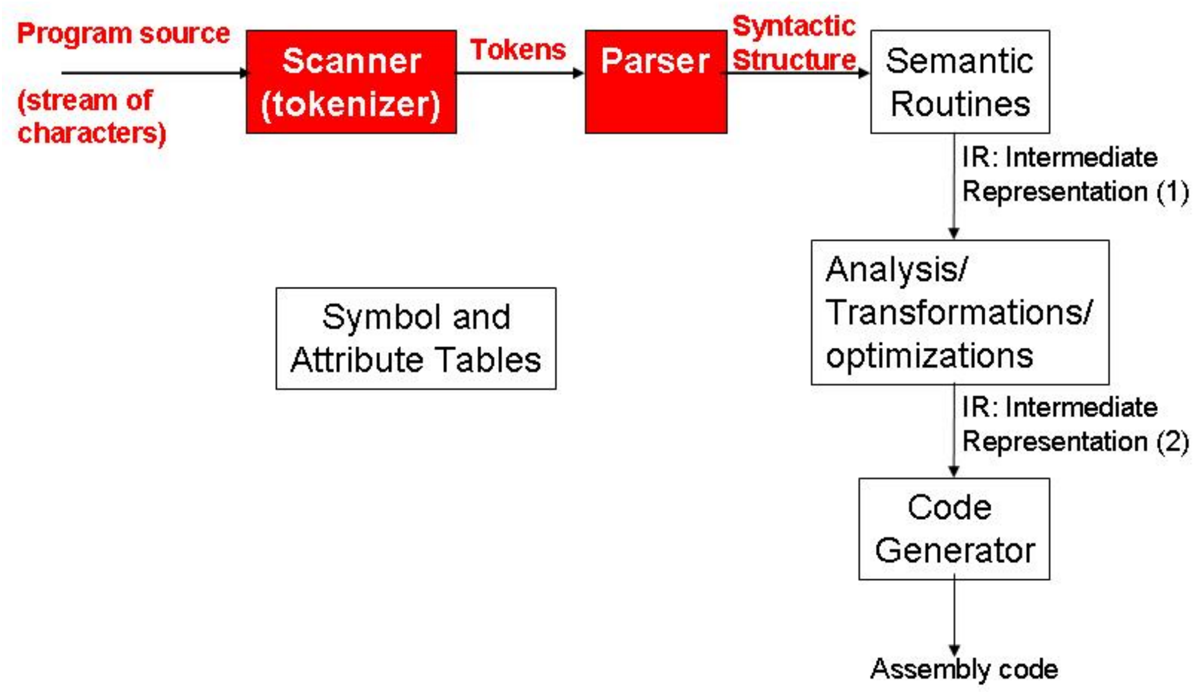
Step 2: Parser -- Due: Friday, February 4th, 11:59pm

Introduction

Your goal in this step is to generate the parser for the project's grammar. By the end of this step your compiler should be able to take a source file as the input and parse the content of that file returning "Accepted" if the file's content is correct according to the grammar or "Not Accepted" if it is not.

Now the scanner created in the first step will be modified to feed the parser. Instead of printing the tokens, the scanner has to return what token is recognized in each step. The following diagram shows the basic structure of a simple compiler and hightlights the steps that will be completed at the end of this step:


Program source
(stream of characters)
Scanner (tokenizer)
Tokens
Parser
Syntactic Structure
Semantic Routines
IR: Intermediate Representation (1)
Symbol and Attribute Tables
Analysis/ Transformations/ optimizations
IR: Intermediate Representation (2)
Code Generator
Assembly code

Parser Generation Tools and Their Tutorials

Similar to the scanner case, there are tools that automatically generate parsers based on context-free grammar descriptions. More precisely, there are parser generators for two main classes of context-free grammars: LR(1) and LL(k). Please, refer to the textbook or one of the references at the end of this handout for the theory of LL(k) and LR(k) parsing.

For this project we recommend two main parser generation tools:

- ANTLR: A LL(k) parser generator for Java.
- Bison: A GNU version of YACC which is a classic LR(1) parser generator for C/C++.

The Task

At this step you have to create the code for the parser of your compiler. We recommend you to do that using one of the tools described above, although the whole parser could be written manually without the help of any tool.

The following basic steps should be followed:

1. Write the parser using one of the tools suggested above;
2. Modify your lexer in the first step to feed tokens to parser instead of printing them;
3. Your parser should print "Accepted" to the standard output if the input file is accepted by the language's grammar or "Not accepted" otherwise.
4. Make sure your parser doesn't produce any shift-reduce errors. You may loose points if you have such errors.

Testcases

Testcases can be downloaded here.

How to handle "Not accepted"

- Java and ANTLR

  In Antlr, a parse error throws an exception. So you would have to write an exception handler that catch "ANTLRException" which is the base class of all exceptions in antlr. Also antlr has default error handling that generates error messages and you need to remove those messages. To do that set the option "defaultErrorHandler=false;" in the options section of the parser.

- C/C++ and Flex/Bison

  When an error occurs in Bison/YACC, the error handler "yyerror" will be invocated. You should customize this handler to print the message and terminate current program.

Submission

You will lose 10% of total credit if you failed to follow the following rules.

Expected Directory Structure of Your Project

Assuming the career account name is "sun47"

- If Java and ANTLR are used, the directory structure of your project is as follows:

```
Some_Directory
`-- sun47
    |-- Makefile
    |-- generated
    |   |-- ...
    |   |-- Files generated by ANTLR from src/*.g
    |   `-- ...
    |-- classes
    |   |-- ...
    |   |-- *.class files compiled from src/*.java and generated/*.java
    |   |-- Micro.class (The class file of your compiler's main class)
    |   `-- ...
    |-- lib
    |   `-- antlr.jar (Please remember to update the Makefile if you use another name for this file.)
    `-- src
        |-- ...
        |-- Your program's source code (*.java) and ANTLR's input file (*.g)
        `-- ...
```

- If C/C++ and Flex/Bison are used, the directory structure of your project is as follows:

```
Some_Directory
`-- sun47
    |-- Makefile
    |-- generated
    |   |-- ...
    |   |-- Files generated by Flex/Bison from src/*.l and src/*.y
    |   `-- ...
    |-- build
    |   |-- ...
    |   |-- *.o files compiled from src/*.c (or src/*.cpp) and generated/*.c (or generated/*.cpp)
    |   |-- micro (The executable file of your compiler)
    |   `-- ...
    |-- lib (This directory may be empty because library files of Flex/Bison have been installed on ECN machines.)
    |   |-- ...
    |   |-- (Please remember to update the Makefile if you add some other library files here.)
    |   `-- ...
    `-- src
        |-- ...
        |-- Your program's source code (*.h and *.c/*.cpp) and Flex/Bison's input file (*.l and *.y)
        `-- ...
```

After ensuring that your project files are organized according to the above directory structure, compress (either with tar and gzip, or with zip) the whole directory structure and email it to the TA and professor.

Expected Behavior of Your Makefile

Assuming the above directory structure is used

- Printing your (and your partner's) ID

  Sample Output:

```
$ cd <Some_Directory>/sun47
$ make group
sun47
$
```

- Cleaning all generated files

  Sample Output:

```
$ cd <Some_Directory>/sun47
$ make clean
$
```

After execution, the directory structure will be:

```
Some_Directory
`-- sun47
    |-- Makefile
    |-- lib
    |   `-- ...
    `-- src
        `-- ...
```

- Building your compiler

  Sample Output:

  ```
  $ cd <Some_Directory>/sun47
  $ make all
  $
  ```

  After execution, the class files or executable files of your compiler should be in the sub-directory mentioned above.

Expected Behavior of Your Compiler

Assuming the above directory structure is used

- If Java and ANTLR are used,

  ```
  $ java -cp <Some_Directory>/sun47/antlr.jar:<Some_Directory>/sun47/classes Micro path_to_your_testcases/test.micro
  Accepted
  $
  ```

- If C/C++ and Flex/Bison are used,

  ```
  $ <Some_Directory>/sun47/build/micro path_to_your_testcases/test.micro
  Accepted
  $
  ```

References

- **Alfred v. Aho, Ravi Sethi, Jeffrey D. Ullman, "Compilers: Principles, Techniques, and Tools", Addison-Wesley, 1986.**
  Syntax Analysis - Chapter 4
- **Andrew W. Appel, "Modern Compiler Implementation in Java", Cambridge Univ. Press, 1998.**
  Parsing - Chapter 3
- **Charles Fischer, Richard LeBlanc, "Crafting a Compiler with C", The Benjamin/Commings Publishing Company, Inc, 1991.**
  Grammars and Parsing - Chapter 4
  LL(1) Grammars and Parsers - Chapter 5
  LR Parsing - Chapter 6