Department of Computer Science
Ashoka University

Programming Language Design and Implementation (PLDI): CS-1319-1

Assignment - 3: `Lexer for` $\mathcal{F}_{-15}$ *Marks: 100*
Assign Date: *September 18, 2024* Submit Date: *23:55, October 1 , 2024*

---

1 Specification of femtoFORTRAN or $\mathcal{F}_{-15}$

Follow the specification of $\mathcal{F}_{-15}$ as provided in Assignment 2.

2 The Assignment

In this assignment use the prefix `name = FirstName_LastName` for all files.

1. Write a Bison specification for the language of $\mathcal{F}_{-15}$ using the syntax specification (grammar) given in Assignment 2 and generate `name_A3.tab.h` (`y.tab.h`) and `name_A3.tab.c` (`y.tab.c`).

   Use the Flex specification that you had developed for Assignment 2 and generate `name_A3.yy.c` (`lex.yy.c`). If required, you may fix your Flex specification.

   If you encounter shift-reduce or reduce-reduce error/s in Bison, you may modify the grammar (without changing the language). Document and justify the changes made.
2. Names of your `.l` and `.y` files should be `name_A3.l` and `name_A3.y` respectively. *The* `.y` *or the* `.l` *file should not contain the function* `main()`. Write your `main()` (in a separate file `name_A3.c`) to test your lexer and parser.
3. Prepare a Makefile to compile the specifications and generate the lexer and the parser. Your Makefile must have a build rule such that when we run `make build`, the output is an executable named `parser`. Your `build` rule should have these commands. The `gcc` command without the `-Werror` will count as the Makefile being incorrect.
4. Prepare a test input file `name_A3.f15` that will test all the rules that you have coded.
5. Prepare a `name_A2.pdf` file explaining the working of your lexer and parser, and your design choices.
6. Prepare a compressed-archive with the name `name_A3.x` , where `x` is one of `zip`, `tar` or `rar`, containing all the above files and upload it to Classroom.

3 Credits

1. Correctness of Implementation: **70**
2. Main file: **5**
3. Makefile[1]: **5**
4. Test file: **5**
5. Explanation of Program: **15**

4 Grading Specifics

1. We will first try to execute `make build`. If this succeeds **and** if after running there is an executable file named `parser` in the directory, we move to point 3, else 2.

[1] If upon make build your Makefile overwrites the wrong file, it will be considered incorrect and if this causes a compilation error, it will be graded as such.

2. We will run the following commands[2], [3] in order,

   ```
   bison name_A3.y --defines=name_A3.tab.h -o name_A3.tab.c
   flex -o name_A3.yy.c name_A3.l
   gcc -o parser name_A3.yy.c name_A3.tab.c name_A3.c -lfl -Werror
   ```

   If this succeeds **and** if after running there is an executable file named `parser` in the directory, we move to point 3 else reports `Compilation Error` and halts. The version of bison being run here is **(GNU Bison) 3.8.2**, flex is **flex 2.6.4** and gcc is **13.2.0**.

3. We will be running your parser on each test case and match the output of your program with the correct output. The command it uses is,

   ```
           ./parser < path/to/test_case/case_name.f15
   ```

   If there is a `Runtime Error` error on any test-case, it throws an exception which is handled by flagging your program and marking that case as failed.

4. Please ensure you have a function, `yyerror()`, defined as follows,

   ```
           void yyerror(char *s) {
               printf("Error: %s on '%s'\n", s, yytext);
           }
   ```

   at the end of your `name_A3.y` file. We will be testing erroneous cases and expecting this format in the output.

5. Please follow the suggested output format as provided with some samples outputs.

[2]The -Werror flag treats all warnings as errors. Thus, if your program compiles with a warning on your system, it will fail on ours. So please test your programs with this flag enabled.

[3]The -lfl flag can be substituted with -ll