X0-Compiler Design Document

Altair, Liu @ ilovehanhan1120@hotmail.com

November 22, 2018

1 Introduction

1.1 Purpose

The purpose of conducting as technical proposal to describe the global designing of this project, containing basic functionality of the system, run-time designing and error detecting methods. This document is aimed to provide a schema of designing and implement all functionality, which will be the critical document during the process of developing. This document will be read by developers and testers.

1.2 Background

This project is to develop a **X0 Language Compiler**, which is a C-like language. This project is mainly for research and study purpose.

Item	Detail
Project Name	X0-Compiler(mini-C)
Developing Platform	Ubuntu 18.04 64-bit
Developing Tools	**Flex** and **Bison**
Open Source or not	Yes

All source files can be found at: http://github.com/SubjectNoi/X0-Compiler

1.3 Remarks

Usage:

```
Ubuntu>$ git clone http://github.com/SubjectNoi/X0-Compiler
Ubuntu>$ cd X0-Compiler
Ubuntu>$ make
Ubuntu>$ ./X0 [Your source file]
```

2 Design Summarize

2.1 Main purpose of the project

Following are main purposes of this project:

- Run correctly on target OS: Ubuntu 18.04 64-bit
- Compile X0 language
- Report compile error, including syntax and semantic error

2.2 Primary demand

The X0 compiler should compile these C-like language, detailed grammar definition will be showed in next section.

```
main {
        integer i, j, flag, cnt := 0;
        for (i := 2; i != 101; i++) {
                flag := 0;
                for (j := 2; j != i; j++) {
                        if (i % j == 0) {
                                flag := 1;
                                break;
                        }
                }
                if (flag == 0) {
                        write(i);
                        cnt++;
                }
        }
        write("There're:");
        write(cnt);
        write("Primes.");
}
```

And correct result should be given. If there exists syntax or semantic error, compiler should report them.

2.3 Restrictions of Design

To complete this project, following restrictions should be watched out:

- Project will be only run on Ubuntu 18.04
- Both developing and testing should be finished before 2018-11-26T11:30:00.000Z

2.4 Principles and Rules of Design

Following principles should be followed in the process of developing:

- Complete: implement as many features as possible
- Simple: try best to ensure low coupling between modules
- High Efficiency: try best to ensure the highest execution efficiency of virtual machine code.

When developing, following rules should be obey:

- All files should be named under following rules:

File	Naming rule
Yacc file	X0-Bison.y
Lex file	X0-Lex.l
Constructing file	Makefile
Testing source	/TestingSrc/Test**XX**_[**Testing Content**]
Git ignore file	.gitignore

- Git is used for version control
- Use **git fetch && git pull**
- Use **git rm -r –cached .**

- Use **git add .**
- Use **git commit -am** [**Meaningful Comment**]
- Never **git push -f**

3 Main Design

3.1 Demand

In this sub-section, detailed grammar of X0 Language will be given:

$$
\begin{aligned}
\textbf{program} &\rightarrow {'main', \{, \textbf{statement\_list}, \}} && (1)\\
\textbf{statement\_list} &\rightarrow \textbf{statement\_list}, \textbf{statement} && (2)\\
&\quad | \textbf{statment} && (3)\\
&\quad | \epsilon && (4)\\
\textbf{statement} &\rightarrow \textbf{expression\_list} && (5)\\
&\quad | \textbf{if\_statement} && (6)\\
&\quad | \textbf{while\_statement} && (7)\\
&\quad | \textbf{read\_statement} && (8)\\
&\quad | \textbf{switch\_statement} && (9)\\
&\quad | \textbf{case\_stat} && (10)\\
&\quad | \textbf{write\_statement} && (11)\\
&\quad | \textbf{compound\_statement} && (12)\\
&\quad | \textbf{for\_statement} && (13)\\
&\quad | \textbf{do\_statement} && (14)\\
&\quad | \textbf{declaration\_list} && (15)\\
&\quad | \textbf{continue\_stat} && (16)\\
&\quad | \textbf{break\_stat} && (17)\\
&\quad | \textbf{yarimasu\_stat} && (18)\\
&\quad | \epsilon && (19)\\
\textbf{declaration\_list} &\rightarrow \textbf{declaration\_list}, \textbf{declaration\_stat} && (20)\\
&\quad | \textbf{declaration\_stat} && (21)\\
&\quad | \epsilon && (22)\\
\textbf{declaration\_stat} &\rightarrow \textbf{typeenum}, \textbf{identlist}, ' ; ' && (23)\\
&\quad | \textbf{typeenum}, \textbf{identarraylist} && (24)\\
&\quad | 'const', \textbf{typeenum}, \textbf{identlist}, \textbf{SEMICOLONSTAT} && (25)\\
&\quad | 'const', \textbf{typeenum}, \textbf{identarraylist} && (26)\\
\textbf{identlist} &\rightarrow \textbf{identdef} && (27)\\
&\quad | \textbf{identlist}, ' , ', \textbf{identdef} && (28)\\
&\quad | \epsilon && (29)\\
\textbf{identdef} &\rightarrow IDENT && (30)\\
&\quad | IDENT, ' := ', \textbf{factor} && (31)\\
& && (32)
\end{aligned}
$$

$$
\begin{aligned}
\textbf{typeenum} &\rightarrow 'integer' && (33)\\
&\mid 'string' && (34)\\
&\mid 'bool' && (35)\\
&\mid 'real' && (36)\\
&\mid 'char' && (37)\\
\textbf{identarraylist} &\rightarrow \textbf{identarraydef} && (38)\\
&\mid \textbf{identarraylist}, ' , ', \textbf{identarraydef} && (39)\\
\textbf{identarraydef} &\rightarrow IDENT, ' [', \textbf{dimensionlist}, ']' && (40)\\
\textbf{dimensionlist} &\rightarrow \textbf{dimension} && (41)\\
&\mid \textbf{dimensionlist}, ' , ', \textbf{dimension} && (42)\\
\textbf{dimension} &\rightarrow INTEGER && (43)\\
\textbf{switch\_statement} &\rightarrow 'switch', ' (', \textbf{expression}, ')', ' \{', \textbf{case\_list}, \textbf{default\_statement}, ' \}' && (44)\\
\textbf{case\_list} &\rightarrow \textbf{case\_list}, \textbf{case\_stat} && (45)\\
&\mid \textbf{case\_stat} && (46)\\
&\mid \epsilon && (47)\\
\textbf{case\_stat} &\rightarrow 'case', \textbf{expression}, ' :', \textbf{compound\_statement} && (48)\\
&\mid \epsilon && (49)\\
\textbf{default\_statement} &\rightarrow 'default', ' :', \textbf{compound\_statement} && (50)\\
\textbf{continue\_stat} &\rightarrow 'continue', ' ;' && (51)\\
\textbf{break\_stat} &\rightarrow 'break', ' ;' && (52)\\
\textbf{if\_statement} &\rightarrow 'if', ' (', \textbf{expression}, ')', \textbf{compound\_statement}, \textbf{else\_list} && (53)\\
\textbf{else\_list} &\rightarrow 'else', \textbf{compound\_statement} && (54)\\
&\mid \epsilon && (55)\\
\textbf{while\_statement} &\rightarrow 'while', ' (', \textbf{expression}, ')', \textbf{compound\_statement} && (56)\\
\textbf{write\_statement} &\rightarrow 'write', ' (', \textbf{expression}, ')' && (57)\\
\textbf{read\_statement} &\rightarrow 'read', ' (', \textbf{var}, ')' && (58)\\
\textbf{compound\_statement} &\rightarrow ' \{', \textbf{statement\_list}, ' \}' && (59)\\
\textbf{for\_statement} &\rightarrow 'for', ' (', \textbf{expression}, ' ;', \textbf{expression}, ' ;', \textbf{expression}, ')', && (60)\\
&\quad \textbf{compound\_statement} && (61)\\
\textbf{do\_statement} &\rightarrow 'do', \textbf{compound\_statement}, ' while', ' (', \textbf{expression}, ')', ' ;' && (62)\\
\textbf{var} &\rightarrow IDENT && (63)\\
&\mid IDENT, ' [', \textbf{expression\_list}, ']' && (64)\\
\textbf{expression\_list} &\rightarrow \textbf{expression} && (65)\\
&\mid \textbf{expression\_list}, ' , ', \textbf{expression} && (66)\\
\textbf{expression} &\rightarrow \textbf{var}, ' :=', \textbf{expression} && (67)\\
&\mid \textbf{simple\_expr} && (68)\\
\textbf{simple\_expr} &\rightarrow \textbf{additive\_expr} && (69)\\
&\mid \textbf{additive\_expr}, \textbf{OPR}, \textbf{additive\_expr} && (70)\\
&\mid \textbf{additive\_expr}, \textbf{SINGLEOPR} && (71)\\
&\mid \textbf{SINGLEOPR}, \textbf{additive\_expr} && (72)\\
\textbf{SINGLEOPR} &\rightarrow ' + +' \mid ' - -' \mid '!' && (73)\\
\textbf{OPR} &\rightarrow ' ==' \mid '! =' \mid ' <' \mid ' <=' \mid ' >' \mid ' >=' \mid '\&\&' \mid '||' \mid ' \wedge ' \mid ' <<' \mid ' >>' && (74)\\
& && (75)
\end{aligned}
$$

$$
\begin{aligned}
\textbf{additive\_expr} &\rightarrow \textbf{term} && (76)\\
&\quad | \textbf{additive\_expr}, \textbf{PLUSMINUS}, \textbf{term} && (77)\\
\textbf{PLUSMINUS} &\rightarrow' +' \,|' -' && (78)\\
\textbf{term} &\rightarrow \textbf{factor} && (79)\\
&\quad | \textbf{term}, \textbf{TIMESDIVIDE}, \textbf{factor} && (80)\\
\textbf{TIMESDEVIDE} &\rightarrow' *' \,|'/'|'\%' && (81)\\
\textbf{factor} &\rightarrow'(', \textbf{expression}, ')' && (82)\\
&\quad | \textbf{var} && (83)\\
&\quad | INTEGER && (84)\\
&\quad | REAL && (85)\\
&\quad | STRING && (86)\\
&\quad | BOOL && (87)\\
&\quad | CHAR && (88)\\
&\quad | YAJU && (89)\\
\textbf{yarimasu\_stat} &\rightarrow' yarimasune', ';' && (90)\\
& && (91)
\end{aligned}
$$

This language should follow this grammar, detailed development of every modules will be mentioned below.

3.2 Environment

This project is developed on Ubuntu 18.08 64-bit, using **make** and corresponding **Makefile** to construct. External tool needed are: **Bison**, **Flex**, **VsCode**, **git**.

3.3 Modules

3.4 Description of Modules

3.5 Hardware

4 Details of modules developing