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:

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

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

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

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 tools needed are: **Bison**, **Flex**, **VsCode**, **git**.

3.3 Modules

This part contain main modules that is to be implemented in this compiler. Including not only basic functionality, but also some bonus functionality. Items with * are bonus modules.

Module Name	Brief Description
Variable store and load	Basic functionality
*Constant store and load	Support constant identifiers
*Multi-type supporting	Support integer, float, string, char and boolean
read and write	Basic input and output, supporting multiple types
Arithmetic operation	Basic arithmetic operation including +, -, *, /, %
Logic operation	Basic logic operation including ==, !=, etc.
Instant number in instruction	Essential modules for multiple types supporting
Expression	Complex, mixed type expression
*Unary operator	Support ++, - -, !
Basic condition statement	If-else statement
Basic loop statement	Do-while, while statement
*Advanced condition statement	Switch-case-default statement
*Advanced loop statement	For statement
*N-dimension array	Support theoretically unlimited dimension array
*Break/Continue	Support break/continue in for, do-while, while, switch, etc.
Error processing	Reporting Syntax and Semantic errors.
Magic identifiers	114514, 1919810, yarimasune, etc.

3.4 Hardware

Item	Model
CPU	Intel Xeon E5-2699v3@2.30GHz(18C36T)
Main Board	ASUS ROG Rampage V Extreme
RAM	Corsair DDR4 2133@15-15-36-50 64GB
GPU	Nvidia Geforce RTX 2080Ti 11GB $\times$ 2
Hard Disk	Intel 750 NVMe SSD 1.2TB $\times$ 2
OS	Ubuntu 18.04 LTS 64-bit

4 Details of modules developing

4.1 ISA

This section mainly describe all technical details of the instruction set, including meaning, usage, etc.

4.2 Variable store and load

4.3 Constant store and load

4.4 Multi-type supporting

4.5 read and write

4.6 Arithmetic operation

4.7 Logic operation

4.8 Expression

4.9 Unary operator

4.10 Basic condition statement

4.11 Basic loop statement

4.12 Advanced condition statement

4.13 Advanced loop statement

4.14 N-dimension array

4.15 Break/Continue

4.16 Error Processing