

# C-Compile **README FOR ENGLISH**

---

**This is an English version for Non-Chinese Programmers.**

C-Compile **README FOR ENGLISH**

To See the Statistic of my code

How To Run it and See the result

**Environment**

**prerequisite**

**You Can Do This**

**FrontEnd**

**LexicalParse**

**Syntaxparse**

**SymbolItem**

**SymbolTable**

**BackEnd**

**BasicBlock**

**FlowGraph**

**optimizer**

**MipsGenerator**

**TmpCode Structure**

**ErrorParse**

My **ErrorParser** is able to identify the error in C below:

How To See The File Structure

**Show you The Case**

**input**

**TmpCode :: Before Optimize**

**TmpCode :: After optimize**

**Mips Code :: For Assemble**

**Running On Mars**

## To See the Statistic of my code

```
cloc-1.64.exe ./
```

```
http://cloc.sourceforge.net v 1.64 T=2.15 s (22.8 files/s, 4151.9 lines/s)
-----
Language             files            blank          comment           code
-----
C++                   17              413            445            5104
C/C++ Header         15              228            136            732
C                     2              111             52            530
make                  1              179             84            334
CMake                 10              58              25            287
XML                   4               0               0            203
-----
SUM:                  49             989            742            7190
-----
```

## How To Run it and See the result

### Environment

- clang++

### prerequisite

- Visual Studio 2019 or Clion
- MARS Simulator (I pack it in my project)

### You Can Do This

1. Add all the code files including headers and source code into Visual studio workplace and then just click "run" To See the result in mips.txt
2. copy mips.txt to "mars simulator" and run for the final result of your input file.

## FrontEnd

Include Syntax-Parse and Lexical-Parse

### LexicalParse

`LexicalParser` is to identify the words, including variable name, constant name, function name and C's Key-word.

### Syntaxparse

`Syntaxparse` is to identify syntax composition by using recursion method.

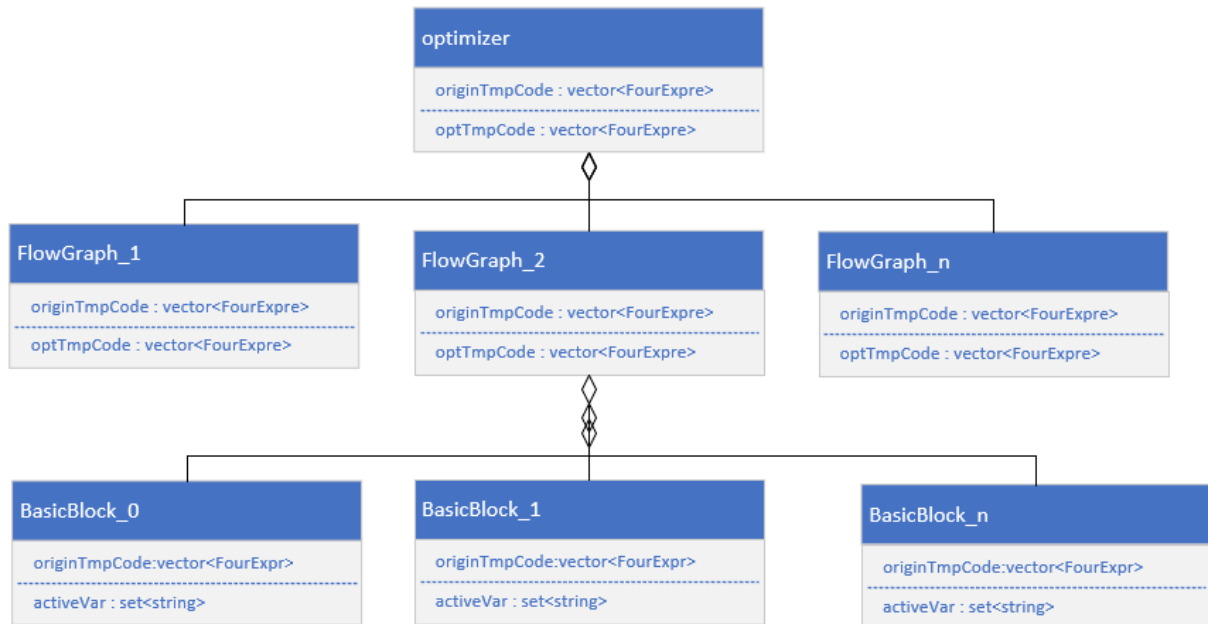
### SymbolItem

`SymbolItem` is used for keep some crucial information of a symbol.

### SymbolTable

`SymbolTable` is one of the most important part of this project which is to store all the symbol information globally and locally.

## Backend



### BasicBlock

In compiler construction, a basic block is a straight-line code sequence with no branches in except to the entry and no branches out except at the exit. This restricted form makes a basic block highly amenable to analysis. Compilers usually decompose programs into their basic blocks as a first step in the analysis process. Basic blocks form the vertices or nodes in a control flow graph.

## reference on Google wikipedia  
:: [https://en.wikipedia.org/wiki/Basic\\_block](https://en.wikipedia.org/wiki/Basic_block)

### FlowGraph

*Generally known as flow-diagram*

A data-flow diagram (DFD) is a way of representing a flow of a data of a process or a system. It can describe the data flow between each basicblock. And every flowGraph contains the code sequence in a function.

The utilization of flowGraph is for global register distribution.

### optimizer

Top-Layer for optimize the code for `mips` running.

Optimizer performs as a switch to start the optimizing process in each flowGraph.

But before we get start, we need to divide the tmpCode sequence by function-scope.

## MipsGenerator

We assume that all the TmpCode sequence have been optimized and store in tmpCodeVector. All we should do is to translate the structure version into mips code.

## TmpCode Structure

I think you may speculate the meaning of each composition...

```
struct FourComExpr {
    TmpCodeType type;
    RetType valueType;
    string target;
    string index;
    string left;
    string right;
    string index1 = "";
    string index2 = "";
    string op;
    int arrayOrVar{};
    vector<SymbolItem> paraSet;
    vector<ExpRet> valueParaTab;
    string varScope;
};
```

## ErrorParse

During the whole compiling process, we need to detect some errors

My `ErrorParser` is able to identify the error in C below:

```
enum SyntaxError {
    LexError, // lexical error
    EmptyFile, //
    RedantCont, // There's redundant context after main function
    LackMain, //
    ArrayDefError, //
    LackSemicn, //
    LackRparent, //
    LackRbrack,
    Lackwhile, // do-while with no 'while'
    ConstDefError,
};

enum SemanticError {
```

```
Redeclare, //
Nodeclare, // variable is used before defination
IndexError, // Array index out of bound
ParaNumError, // The nums of parameters of a function is not matched
ParaTypeError, // The type of parameters is not matched
ConstAssError, // assigned to a CONSTANT
AssTypeError, // Assigned with wrong value type
ZeroError, // divide zero
CondParseError, //

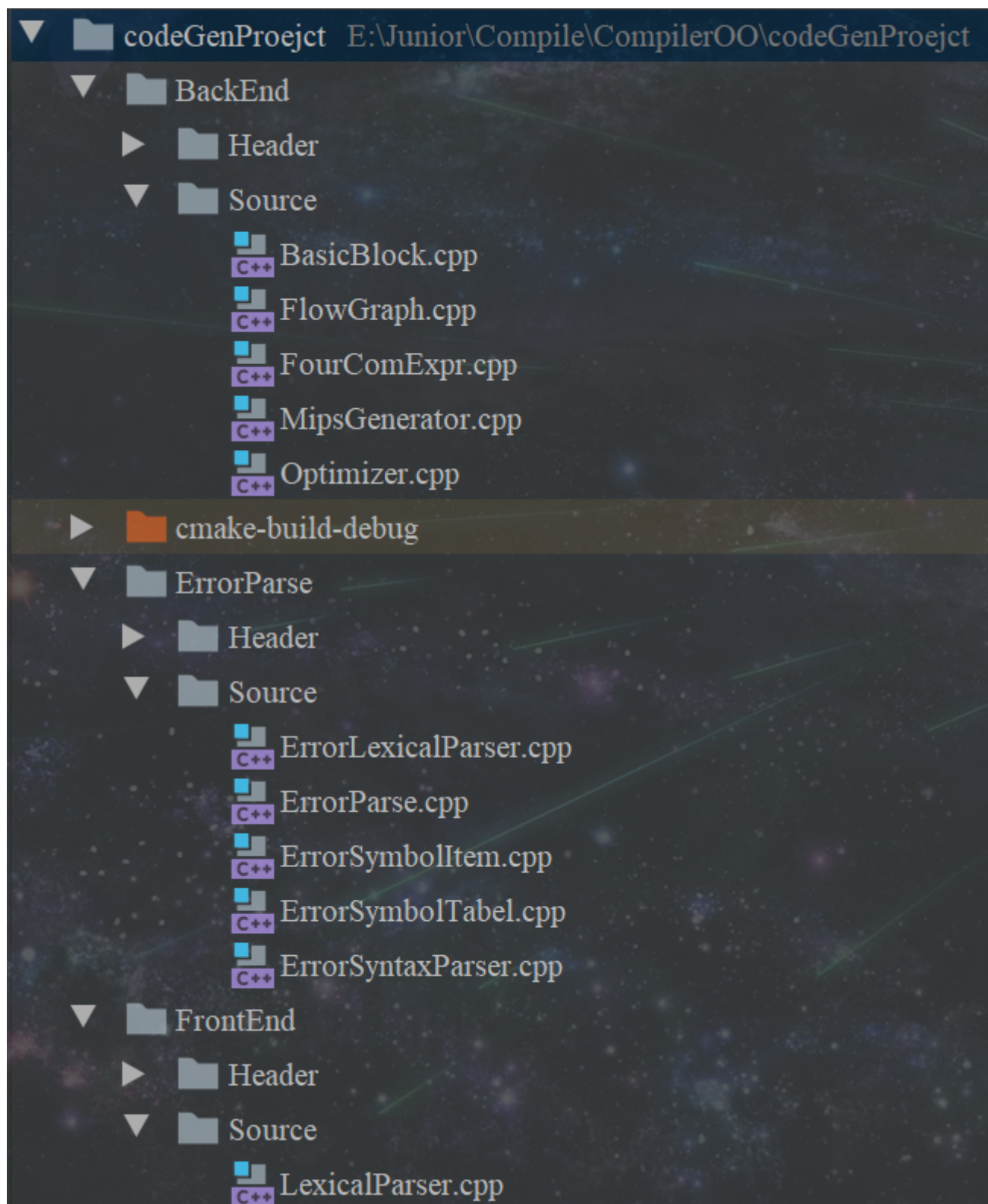
};

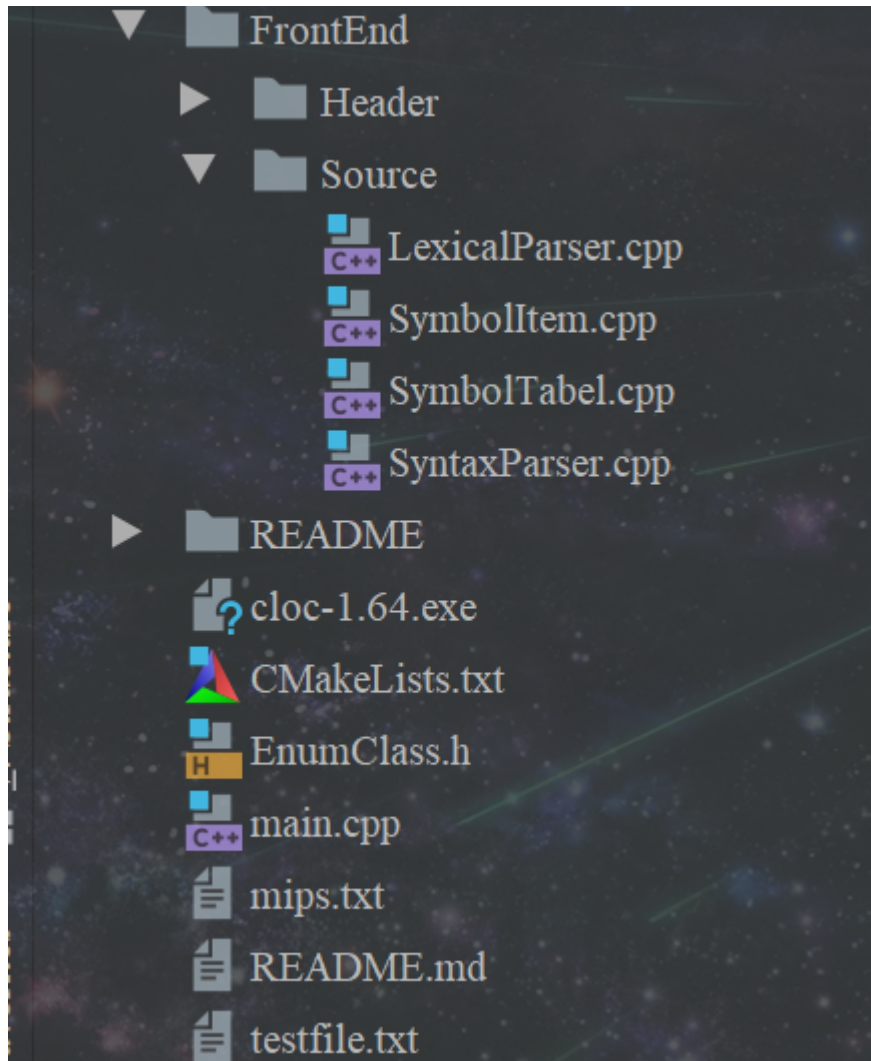
enum ReturnError {
    voidReturn, // void func with unmatched return state
    lackReturn, // in function with return value but lacking valid return
    retTypeError, // with wrong return value
};
```

## How To See The File Structure

---

```
cmd :: tree -f
```





## Show you The Case

### input

see in testfile.txt

### TmpCode :: Before Optimize

see in tmpcode.txt

### TmpCode :: After optimize

see in optTmpCode.txt

### Mips Code :: For Assemble

see in mips.txt

# Running On Mars

E:\Junior\Compile\CompilerOO\codeGenProejct\mips1.asm - MARS 4.5

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Registers Coproc 1 Coproc 0

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$t0	2	0x00000004
\$t1	3	0x00000004
\$t2	4	0x10010002
\$t3	5	0x00000000
\$t4	6	0x00000000
\$t5	7	0x00000000
\$t6	8	0x00000006
\$t7	9	0x00000246
\$t8	10	0x00000246
\$t9	11	0x00000078
\$t10	12	0x00000008
\$t11	13	0x00000006
\$t12	14	0x00000002
\$t13	15	0x00000004
\$t14	16	0x00000000
\$t15	17	0x00000000
\$t16	18	0x00000000
\$t17	19	0x00000000
\$t18	20	0x00000000
\$t19	21	0x00000000
\$t20	22	0x00000000
\$t21	23	0x00000000
\$t22	24	0x10010180
\$t23	25	0x00000018
\$t24	26	0x00000246
\$t25	27	0x00000004
\$gp	28	0x10080000
\$sp	29	0x7fff4ff8
\$fp	30	0x7fff4ff8
\$ra	31	0x00400004
pc		0x00400004
lr		0x00000000

```
692 sw $t2, -28($fp)
693 sw $t3, -32($fp)
694 sw $a1, -4($fp)
695 sw $a2, -8($fp)
696 sw $a3, -12($fp)
697 li $a1, 3
698 jal Factorial
699 move $t0, $v0
700 sll $t9, $t0, 2
701 la $t8, testnum
702 add $k0, $t8, $t9
703 lw $k0, 0($k0)
704 addi $t1, $k0, 0
705 move $a0, $t1
706 li $v0, 1
707 syscall
708 la $a0, newline
709 li $v0, 4
710 syscall
711
```

Line: 708 Column: 16 Show Line Numbers

Mars Messages Run IO

The testCh is : \*\*\*\* user input : a  
\*\*\*\* user input : 1  
\*\*\*\* user input : 1  
Global a \* b is 2  
Before assignment value to local a/b in function a \* b is 300  
After assignment value to local a/b in function a \* b is 600  
Before assignment value to global a/b in function a \* b is 2

Then You can see the result in the console...