

# Final Project - LLVM IR

---

作者

410985013 羅凱威

編譯器支援的文法規則或功能

```
Program = GlobalStatements;

GlobalStatements =
    | GlobalStatement
    | GlobalStatements GlobalStatement
    ;

GlobalStatement =
    | FunctionDeclaration
    | FunctionDefinition ";"
    | DeclarationExpression ";"
    ;

FunctionDefinition =
    | type Identifier "(" FPDeclarationList ")"
    ;

FunctionDeclaration =
    | FunctionDefinition FunctionBlock
    ;

FPDeclarationList =
    | /* empty */
    | FPDeclarationList "," FPDeclaration
    | FPDeclaration
    ;

FPDeclaration =
    | type Identifier
    ;

FunctionBlock =
    | "{" FunctionStatements "}"
    ;

FunctionStatements =
    | FunctionStatement
    | FunctionStatements FunctionStatement
    ;

FunctionStatement =
    | Statement
```

```

    | ReturnStatement
    ;

ReturnStatement =
    | "return" Expression ";"
    | "return" ";"
    ;

FunctionCallExpression =
    | Identifier "(" FCParameterList ")"
    ;

FCParameterList =
    | /* empty */
    | FCParameterList "," Expression
    | Expression
    ;

Statement =
    | DeclarationExpression ";"
    | AssignExpression ";"
    | FunctionCallExpression ";"
    | IfStatement
    | WhileStatement
    | ForStatement
    | ContinueStatement
    | BreakStatement
    | ReturnStatement
    ;

ForStatement =
    | "for" "(" ForInitExpression ";" Condition ";" AssignExpression ")" Block
    ;

ForInitExpression =
    | DeclarationExpression
    | AssignExpression
    ;

ContinueStatement =
    | "continue" ";"
    ;

BreakStatement =
    | "break" ";"
    ;

WhileStatement =
    | "while" "(" Condition ")" Block
    ;

IfStatement =
    | "if" "(" Condition ")" Block
    | "if" "(" Condition ")" Block "else" Block

```

```

;

Block =
| Statement
| "{" Statements "}"
;

Statements =
| /* empty */
| Statements Statement
;

DeclarationExpression =
| type DeclarationList
;

DeclarationList =
| DeclarationList "," Identifier
| DeclarationList "," AssignExpression
| AssignExpression
| Identifier
;

AssignExpression =
| Identifier "=" Expression
;

Condition =
| Expression "<" Expression
| Expression ">" Expression
| Expression "==" Expression
| Expression "<=" Expression
| Expression ">=" Expression
| Expression "!=" Expression
;

Expression =
| Term
| Expression "+" Term
| Expression "-" Term
;

Term =
| Factor
| Term "*" Factor
| Term "/" Factor
;

Factor =
| Identifier
| Numeric
| "(" Expression ")"
| FunctionCallExpression
;

```

```
Identifier =  
    | IDENTIFIER  
    ;  
  
Numeric =  
    | NUMBER  
    | FRAC_NUMBER  
    ;  
  
type =  
    | "int"  
    | "double"  
    | "void"  
    ;
```

## 編譯器製作的過程

### 如何撰寫

利用 flex 進行詞法分析，bison 進行語法分析，並在語法分析的途中，建出語法抽象樹 AST(Abstract Syntax Tree)，接著遍歷 AST 並產生 LLVM IR (Intermediate Representation)，最後利用 LLVM Backends 將 LLVM IR 編譯成目標代碼 (object code)，生成的目標代碼可直接編譯成執行檔，也可以與其他目標代碼一同編譯成執行檔。

### 遭遇的困難

LLVM 算是中大型的專案，在 windows 下載並成功建置就需要一番工夫。

另外 LLVM 歷史悠久，網路上的資源未必適用於新版本，我的參考資料多數都超過五年以上。

### 如何解決問題

仔細閱讀 LLVM 官方文件，並參考其他人的經驗，通常我遇到的問題，都會有其他人遇到。有複雜，難以描述清楚的問題，就透過 GPT 輔助，將問題拆解成小問題後再丟給 Google。

### 編譯器製作的成果：測試程式執行結果的截圖

```
int a = 1.5, b, c; // global  
  
void printInt(int a);  
void printDouble(double a);  
  
int testFunction(int a, int b);  
void nothing();  
int fib(int a);  
  
int main() {  
    int a = 1.5, b; // local
```

```

double z;
double x = 3.14, y = 2.71;

printDouble(x);
/*
multiline comment
*/

int z = testFunction(1, 2);
printInt(z);

a = 5;
b = a + 10;
c = (a + b) * 2 / testFunction(3 + 5, a + b); // global c

if (a == b) {
    c = a + b;
}
else if (a > b) {
    c = a - b;
}
else {
    c = a * b; // c = 75
}
printInt(c);

while (c > 0) {
    c = c / 2;
    if (c > 10) {
        continue;
    }
    printInt(c);
}

for (int i = 0; i < 10; i = i + 1) {
    if (i == 5) {
        break;
    }
    if (i == 3) {
        continue;
    }
    printInt(i);
}

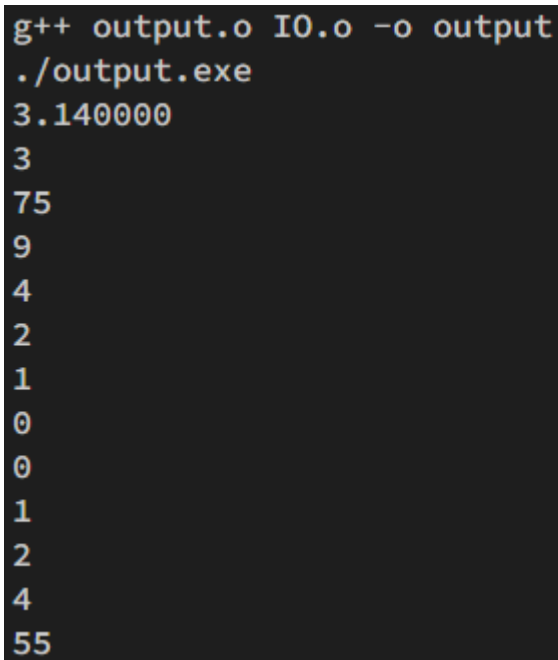
printInt(fib(10));
return 0;
}

int fib(int a) {
    if (a == 0) {
        return 0;
    }
    if (a == 1) {
        return 1;
    }
}

```

```
    }  
    return fib(a - 1) + fib(a - 2);  
}  
  
int testFunction(int a, int b) {  
    int c = a + b;  
    return c;  
}  
  
void nothing() {  
    return;  
}
```

測試程式執行結果截圖



```
g++ output.o IO.o -o output  
./output.exe  
3.140000  
3  
75  
9  
4  
2  
1  
0  
0  
1  
2  
4  
55
```

## 編譯器製作的心得

這次的專案讓我對編譯器的運作有了更深入的了解，也讓我對 LLVM 有了初步的認識。

## 使用方法

編譯出 compiler.exe

```
make
```

附圖

- 掃描到的 token

```
PS C:\Users\A2320\Desktop\school\4down\compiler\compiler_hw\final_project> make run
./compiler.exe < correct_code.c
TOKEN: KW_INT
TOKEN: IDENTIFIER a
TOKEN: '='
TOKEN: DOUBLE 1.5
TOKEN: ','
TOKEN: IDENTIFIER b
TOKEN: ','
TOKEN: IDENTIFIER c
TOKEN: ';'
TOKEN: KW_VOID
TOKEN: IDENTIFIER printInt
TOKEN: '('
TOKEN: KW_INT
TOKEN: IDENTIFIER a
TOKEN: ')'
TOKEN: ';'
TOKEN: KW_VOID
TOKEN: IDENTIFIER printDouble
TOKEN: '('
TOKEN: KW_DOUBLE
TOKEN: IDENTIFIER a
```

- 語法分析建 AST 的過程

```
Generating code...
Generating code for NVariableDeclarationList
Creating variable declaration list
Creating global variable declaration of a
Creating double: 1.5
Creating global variable declaration of b
Creating global variable declaration of c
Creating function declaration for printInt
Creating function declaration for printDouble
Creating function declaration for testFunction
Creating function declaration for nothing
Creating function declaration for fib
Creating function definition for main
Generating code for 15 statements
Generating code for NVariableDeclarationList
Creating variable declaration list
Creating variable declaration of a
Creating double: 1.5
Creating variable declaration of b
Generating code for NVariableDeclarationList
Creating variable declaration list
Creating variable declaration of z
Generating code for NVariableDeclarationList
Creating variable declaration list
Creating variable declaration of x
Creating double: 2.14
```

- 生成的 LLVM IR ( 可以在 output.ll 查看 )

```
; ModuleID = 'main'
source_filename = "main"
target datalayout = "e-m:w-p270:32:32-p271:32:32-p272:64:64-i64:64-i128:128-f80:128-n8:16:32:64-S128"
target triple = "x86_64-w64-windows-gnu"

@a = common global i64 1.500000e+00
@b = common global i64 0
@c = common global i64 0

declare void @printInt(i64)

declare void @printDouble(double)

define i64 @testFunction(i64 %a, i64 %b) {
entry:
    %a1 = alloca i64, align 8
    store i64 %a, ptr %a1, align 4
    %b2 = alloca i64, align 8
    store i64 %b, ptr %b2, align 4
    %c = alloca i64, align 8
    %a3 = load i64, ptr %a1, align 4
    %b4 = load i64, ptr %b2, align 4
    %addtmp = add i64 %a3, %b4
    store i64 %addtmp, ptr %c, align 4
    %c5 = load i64, ptr %c, align 4
    ret i64 %c5
}

define void @nothing() {
entry:
    ret void
    ret void <badref>
}
```

使用 compiler.exe 編譯程式

會生成 output.o, output.ll · 其中 output.ll 是 LLVM IR, output.o 是目標代碼。

```
compiler.exe < test.c
```

使用 gcc 將目標語言編譯成執行檔

```
g++ -o test.exe output.o
```

我自己寫的測試

```
make run
```

其中會執行以下指令，使用了 IO.cpp 中的 printInt 和 printDouble 函數，來進行輸出。

```
./compiler.exe < correct_code.c
g++ IO.cpp -c -o IO.o
```



```
g++ output.o IO.o -o output  
./output.exe
```

## 參考文獻

<https://llvm.org/docs/GettingStarted.html#tools> <https://bayareanotes.com/llvm-ir-intro/>  
<https://blog.csdn.net/zzhongcy/article/details/93753017> <https://medium.com/codex/building-a-c-compiler-using-lex-and-yacc-446262056aaa>  
[https://www.gnu.org/software/bison/manual/html\\_node/Shift\\_002fReduce.html](https://www.gnu.org/software/bison/manual/html_node/Shift_002fReduce.html)  
<https://stackoverflow.com/questions/12731922/reforming-the-grammar-to-remove-shift-reduce-conflict-in-if-then-else> <https://ithelp.ithome.com.tw/users/20157613/ironman/6494>  
<https://gnu.org/2009/09/18/writing-your-own-toy-compiler/>  
<https://llvm.org/docs/tutorial/MyFirstLanguageFrontend/index.html>