# Swift Subset Compiler
# Part 3: Complete Compiler

**Name: Anaum Khan** **Course: Compiler Design**
**Faculty Number: 22COB307** **Date: 25 October 2025**

---

## 1. Objective

To design and implement a complete compiler for a subset of the Swift programming language. This compiler performs lexical analysis, syntax analysis, semantic analysis, and code generation. It translates Swift-like source code into executable C code, generating three-address intermediate code in the process.

---

## 2. Tools & Environment

| Tool/Component | Version / Notes |
|---|---|
| Operating System | Windows 10 + WSL Ubuntu 22.04 |
| Editor | VS Code (Remote - WSL Extension) |
| Lexical Analyzer Generator | Flex |
| Parser Generator | Bison |
| Compiler | GCC |
| Build Tool | Make |

### Build Commands

```
root@LAPTOP-B9CHUST6:~/assgnmnt# make clean
rm -f compiler lex.yy.o parser.tab.o symtab.o codegen.o lex.yy.c parse
r.tab.c parser.tab.h output.c
root@LAPTOP-B9CHUST6:~/assgnmnt# make
bison -d parser.y
parser.y: warning: 1 shift/reduce conflict [-Wconflicts-sr]
parser.y: note: rerun with option '-Wcounterexamples' to generate conf
lict counterexamples
flex lexer.l
gcc -Wall -g -c lex.yy.c
lex.yy.c:1533:16: warning: ‘input’ defined but not used [-Wunused-func
tion]
 1533 |     static int input  (void)
      |                ^~~~~
lex.yy.c:1486:17: warning: ‘yyunput’ defined but not used [-Wunused-fu
nctio]
 1486 |     static void yyunput (int c, char * yy_bp )
      |                 ^~~~~~~
gcc -Wall -g -c parser.tab.c
gcc -Wall -g -c symtab.c
gcc -Wall -g -c codegen.c
gcc -Wall -g -o compiler lex.yy.o parser.tab.o symtab.o codegen.o -lfl
```

### Execution Command

**./compiler test_input.swift**

**gcc -o program output.c**

**./program**

---

# 3. Lexical Analyzer Design

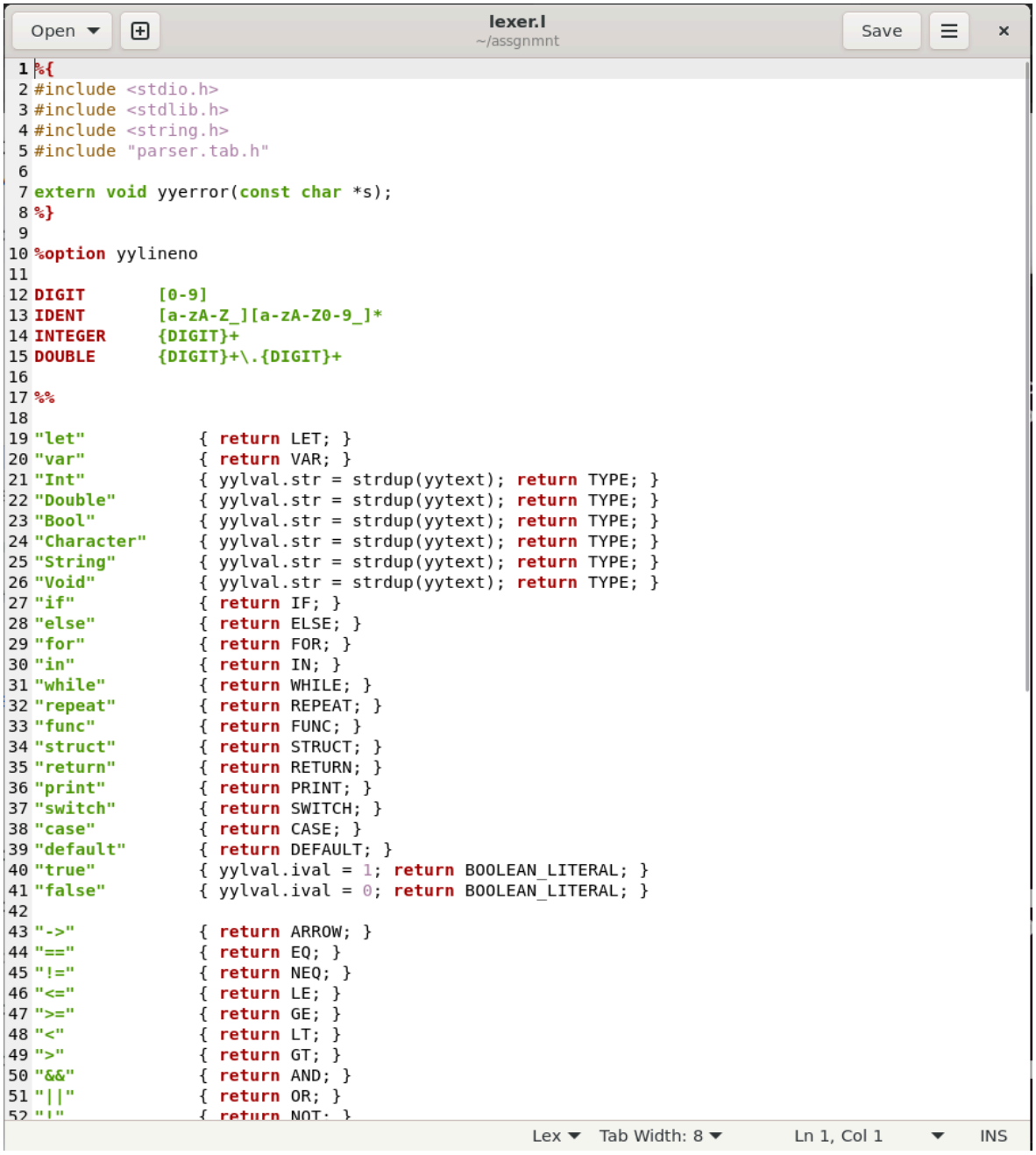
### Token Categories:

- **Keywords**: `let`, `var`, `if`, `else`, `switch`, `case`, `default`, `for`, `in`, `while`, `repeat`, `func`, `struct`, `return`, `print`
- **Type Specifiers**: `Bool`, `Int`, `Double`, `String`, `Character`, `Void`
- **Identifiers**: user-defined names for variables, functions, structs
- **Literals**:
    - Integers (e.g., `42`)
    - Doubles (e.g., `3.14`)
    - Strings (e.g., `"hello"`)
    - Characters (e.g., `'a'`)

    - Booleans (`true`, `false`)
- **Operators**:
    - Arithmetic: `+`, `-`, `*`, `/`, `%`
    - Relational: `==`, `!=`, `<`, `>`, `<=`, `>=`
    - Logical: `&&`, `||`, `!`
    - Assignment: `=`
    - Arrow: `->`
- **Delimiters / Symbols**: `{`, `}`, `(`, `)`, `[`, `]`, `:`, `,`, `;`, `.`

---

**Code- lexer.l file:**




```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "parser.tab.h"

extern void yyerror(const char *s);
%}

%option yylineno

DIGIT       [0-9]
IDENT       [a-zA-Z_][a-zA-Z0-9_]*
INTEGER     {DIGIT}+
DOUBLE      {DIGIT}+\.{DIGIT}+

%%

"let"           { return LET; }
"var"           { return VAR; }
"Int"           { yylval.str = strdup(yytext); return TYPE; }
"Double"        { yylval.str = strdup(yytext); return TYPE; }
"Bool"          { yylval.str = strdup(yytext); return TYPE; }
"Character"     { yylval.str = strdup(yytext); return TYPE; }
"String"        { yylval.str = strdup(yytext); return TYPE; }
"Void"          { yylval.str = strdup(yytext); return TYPE; }
"if"            { return IF; }
"else"          { return ELSE; }
"for"           { return FOR; }
"in"            { return IN; }
"while"         { return WHILE; }
"repeat"        { return REPEAT; }
"func"          { return FUNC; }
"struct"        { return STRUCT; }
"return"        { return RETURN; }
"print"         { return PRINT; }
"switch"        { return SWITCH; }
"case"          { return CASE; }
"default"       { return DEFAULT; }
"true"          { yylval.ival = 1; return BOOLEAN_LITERAL; }
"false"         { yylval.ival = 0; return BOOLEAN_LITERAL; }

"->"            { return ARROW; }
"=="            { return EQ; }
"!="            { return NEQ; }
"<="            { return LE; }
">="            { return GE; }
"<"             { return LT; }
">"             { return GT; }
"&&"            { return AND; }
"||"            { return OR; }
"!"             { return NOT; }
```

```
"||"            { return OR; }
"!"             { return NOT; }
"="             { return ASSIGN; }

"+"             { return PLUS; }
"-"             { return MINUS; }
"*"             { return MUL; }
"/"             { return DIV; }
"%"             { return MOD; }

":"             { return ':'; }
";"             { return ';'; }
","             { return ','; }
"."             { return '.'; }
"("             { return '('; }
")"             { return ')'; }
"{"             { return '{'; }
"}"             { return '}'; }
"["             { return '['; }
"]"             { return ']'; }

{DOUBLE}        { yylval.fval = atof(yytext); return DOUBLE_LITERAL; }
{INTEGER}       { yylval.ival = atoi(yytext); return INT_LITERAL; }
\"([^\\\"]|\\.)*\"   { yylval.str = strdup(yytext); return STRING_LITERAL; }
\'([^\\\']|\\.)\'    { yylval.str = strdup(yytext); return CHARACTER_LITERAL; }
{IDENT}         { yylval.str = strdup(yytext); return IDENTIFIER; }

[ \t\r\n]+      ;

.               { printf("Unknown character: %s\n", yytext); exit(1); }

%%

int yywrap() {
    return 1;
}
```

Lex ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

---

# 4. Sample Programs & Token Output

## Test 1: Variables & Types

Open ▾ | test1.swift ~/assgnmnt | Save | ☰ | ×

```
let a: Int = 10;
let b: Int = 20;
var sum = a + b;
var diff = a - b;
var product = a * b;
var quotient = b / a;

print(sum);
print(diff);
print(product);
print(quotient);

if (a < b) {
    print(1);
} else {
    print(0);
}
```

**OUTPUT:**

```
root@LAPTOP-B9CHUST6:~/assgnmnt# ./compiler test1.swift
╔═══════════════════════════════════════╗
║        SWIFT SUBSET COMPILER - PART 3         ║
╚═══════════════════════════════════════╝

=== COMPILATION STARTED ===

=== LEXICAL ANALYSIS ===
Tokenizing input file...
Identifying keywords, operators, and identifiers...
Status: IN PROGRESS

=== SYNTAX ANALYSIS ===
Building parse tree...
Status: IN PROGRESS


=== PARSING COMPLETE ===
Abstract Syntax Tree built successfully

=== SEMANTIC ANALYSIS ===
Checking variable declarations...
Checking type compatibility...
Checking constant assignments...
Status: PASSED

=== SYMBOL TABLE ===

Symbol Table Contents:
----------------------
  Name: quotient        Type: Int        Kind: variable
  Name: product         Type: Int        Kind: variable
  Name: diff            Type: Int        Kind: variable
  Name: sum             Type: Int        Kind: variable
  Name: b               Type: Int        Kind: constant
  Name: a               Type: Int        Kind: constant

=== CODE GENERATION ===
```

```
=== CODE GENERATION ===
Generating intermediate code...
Translating to C...
Status: COMPLETE

=== THREE ADDRESS CODE ===
==== Three-Address Code ====
1: a = 10
2: b = 20
3: t0 = a + b
4: sum = t0
5: t1 = a - b
6: diff = t1
7: t2 = a * b
8: product = t2
9: t3 = b / a
10: quotient = t3
11: print sum
12: print diff
13: print product
14: print quotient
15: t4 = a < b
16: if t4 == false goto L0
17: goto L1
18: L0:
19: L1:
20: print 1
21: print 0

╔═══════════════════════════════════════╗
║          COMPILATION SUCCESSFUL       ║
╚═══════════════════════════════════════╝

Output file: output.c

To run the compiled program:
```

```
To run the compiled program:
  gcc -o program output.c
  ./program
root@LAPTOP-B9CHUST6:~/assgnmnt# gcc -o program output.c
root@LAPTOP-B9CHUST6:~/assgnmnt# ./program
30
-10
200
2
1
```

## Test 2:

Open | test2.swift ~/assgnmnt | Save

```swift
var numbers: [Int] = [5, 10, 15, 20, 25];

print(numbers[0]);
print(numbers[2]);
print(numbers[4]);

numbers[1] = 100;
print(numbers[1]);

var counter = 0;
while (counter < 5) {
    print(counter);
    counter = counter + 1;
}

var limit = 3;
for idx in limit {
    print(idx);
}
```

## OUTPUT:

```
root@LAPTOP-B9CHUST6:~/assgnmnt# ./compiler test2.swift
╔═══════════════════════════════════════╗
        SWIFT SUBSET COMPILER - PART 3
╚═══════════════════════════════════════╝

=== COMPILATION STARTED ===

=== LEXICAL ANALYSIS ===
Tokenizing input file...
Identifying keywords, operators, and identifiers...
Status: IN PROGRESS

=== SYNTAX ANALYSIS ===
Building parse tree...
Status: IN PROGRESS


=== PARSING COMPLETE ===
Abstract Syntax Tree built successfully

=== SEMANTIC ANALYSIS ===
Checking variable declarations...
Checking type compatibility...
Checking constant assignments...
Status: PASSED

=== SYMBOL TABLE ===

Symbol Table Contents:
----------------------
  Name: idx             Type: Int        Kind: variable
  Name: limit           Type: Int        Kind: variable
  Name: counter         Type: Int        Kind: variable
  Name: numbers         Type: [Int]      Kind: variable

=== CODE GENERATION ===
Generating intermediate code...
```

```
=== CODE GENERATION ===
Generating intermediate code...
Translating to C...
Status: COMPLETE

=== THREE ADDRESS CODE ===
==== Three-Address Code ====
1: numbers = {...}
2: t0 = numbers[0]
3: print t0
4: t1 = numbers[2]
5: print t1
6: t2 = numbers[4]
7: print t2
8: numbers[1] = 100
9: t3 = numbers[1]
10: print t3
11: counter = 0
12: L0:
13: t4 = counter < 5
14: if t4 == false goto L1
15: print counter
16: t5 = counter + 1
17: counter = t5
18: goto L0
19: L1:
20: limit = 3
21: idx = 0
22: L2:
23: t6 = idx < limit
24: if t6 == false goto L3
25: print idx
26: idx = idx + 1
27: goto L2
28: L3:
```

```
╔═══════════════════════════════════════╗
║          COMPILATION SUCCESSFUL       ║
╚═══════════════════════════════════════╝

Output file: output.c

To run the compiled program:
  gcc -o program output.c
  ./program
root@LAPTOP-B9CHUST6:~/assgnmnt# gcc -o program output.c
^[[Aroot@LAPTOP-B9CHUST6:~/assgnmnt# ./program
25
15
5
100
0
1
2
3
4
0
1
2
```

## Test 3:

Open | test3.swift ~/assgnmnt | Save

```swift
let grade: Int = 85;

if (grade >= 90) {
    print(4);
} else {
    if (grade >= 80) {
        print(3);
    } else {
        if (grade >= 70) {
            print(2);
        } else {
            print(1);
        }
    }
}

let day: Int = 3;
switch day {
    case 1:
        print(100);
    case 2:
        print(200);
    case 3:
        print(300);
    case 4:
        print(400);
    default:
        print(999);
}

var n = 5;
var factorial = 1;
while (n > 0) {
    factorial = factorial * n;
    n = n - 1;
}
print(factorial);
```

## OUTPUT:

```
root@LAPTOP-B9CHUST6:~/assgnmnt# ./compiler test3.swift
╔═══════════════════════════════════════╗
║        SWIFT SUBSET COMPILER - PART 3         ║
╚═══════════════════════════════════════╝

=== COMPILATION STARTED ===

=== LEXICAL ANALYSIS ===
Tokenizing input file...
Identifying keywords, operators, and identifiers...
Status: IN PROGRESS

=== SYNTAX ANALYSIS ===
Building parse tree...
Status: IN PROGRESS


=== PARSING COMPLETE ===
Abstract Syntax Tree built successfully

=== SEMANTIC ANALYSIS ===
Checking variable declarations...
Checking type compatibility...
Checking constant assignments...
Status: PASSED

=== SYMBOL TABLE ===

Symbol Table Contents:
----------------------
  Name: factorial       Type: Int        Kind: variable
  Name: n               Type: Int        Kind: variable
  Name: day             Type: Int        Kind: constant
  Name: grade           Type: Int        Kind: constant
```

```
=== CODE GENERATION ===
Generating intermediate code...
Translating to C...
Status: COMPLETE

=== THREE ADDRESS CODE ===
==== Three-Address Code ====
1: grade = 85
2: t0 = grade >= 90
3: if t0 == false goto L0
4: goto L1
5: L0:
6: L1:
7: print 4
8: t1 = grade >= 80
9: if t1 == false goto L2
10: goto L3
11: L2:
12: L3:
13: print 3
14: t2 = grade >= 70
15: if t2 == false goto L4
16: goto L5
17: L4:
18: L5:
19: print 2
20: print 1
21: day = 3
22: case 1:
23: print 100
24: case 2:
25: print 200
26: case 3:
27: print 300
28: case 4:
29: print 400
```

```
30: default:
31: print 999
32: n = 5
33: factorial = 1
34: L6:
35: t3 = n > 0
36: if t3 == false goto L7
37: t4 = factorial * n
38: factorial = t4
39: t5 = n - 1
40: n = t5
41: goto L6
42: L7:
43: print factorial

╔═══════════════════════════════════════╗
║          COMPILATION SUCCESSFUL       ║
╚═══════════════════════════════════════╝

Output file: output.c

To run the compiled program:
  gcc -o program output.c
  ./program
root@LAPTOP-B9CHUST6:~/assgnmnt# gcc -o program output.c
root@LAPTOP-B9CHUST6:~/assgnmnt# ./program
3
300
120
```

## Test 4:

Open | test4.swift ~/assgnmnt | Save

```swift
let x: Int = 15;
let y: Int = 25;
var z = x + y;

print(x);
print(y);
print(z);

var max = 0;
if (x > y) {
    max = x;
} else {
    max = y;
}
print(max);

var data: [Int] = [100, 200, 300, 400, 500];
print(data[0]);
data[2] = 350;
print(data[2]);

var i = 1;
while (i <= 5) {
    print(i);
    i = i + 1;
}

let choice: Int = 2;
switch choice {
    case 1:
        print(1000);
    case 2:
        print(2000);
    case 3:
        print(3000);
    default:
        print(9999);
}

var loopvar = 4;
for item in loopvar {
    print(item);
}
```

**OUTPUT:**

```
root@LAPTOP-B9CHUST6:~/assgnmnt# ./compiler test4.swift
╔═══════════════════════════════════════╗
║        SWIFT SUBSET COMPILER - PART 3         ║
╚═══════════════════════════════════════╝

=== COMPILATION STARTED ===

=== LEXICAL ANALYSIS ===
Tokenizing input file...
Identifying keywords, operators, and identifiers...
Status: IN PROGRESS

=== SYNTAX ANALYSIS ===
Building parse tree...
Status: IN PROGRESS


=== PARSING COMPLETE ===
Abstract Syntax Tree built successfully

=== SEMANTIC ANALYSIS ===
Checking variable declarations...
Checking type compatibility...
Checking constant assignments...
Status: PASSED

=== SYMBOL TABLE ===

Symbol Table Contents:
----------------------
  Name: item            Type: Int        Kind: variable
  Name: loopvar         Type: Int        Kind: variable
  Name: choice          Type: Int        Kind: constant
  Name: i               Type: Int        Kind: variable
  Name: data            Type: [Int]      Kind: variable
  Name: max             Type: Int        Kind: variable
  Name: z               Type: Int        Kind: variable
```

```
  Name: y               Type: Int        Kind: constant
  Name: x               Type: Int        Kind: constant

=== CODE GENERATION ===
Generating intermediate code...
Translating to C...
Status: COMPLETE

=== THREE ADDRESS CODE ===
==== Three-Address Code ====
1: x = 15
2: y = 25
3: t0 = x + y
4: z = t0
5: print x
6: print y
7: print z
8: max = 0
9: t1 = x > y
10: if t1 == false goto L0
11: goto L1
12: L0:
13: L1:
14: max = x
15: max = y
16: print max
17: data = {...}
18: t2 = data[0]
19: print t2
20: data[2] = 350
21: t3 = data[2]
22: print t3
23: i = 1
24: L2:
25: t4 = i <= 5
26: if t4 == false goto L3
27: print i
```

```
28: t5 = i + 1
29: i = t5
30: goto L2
31: L3:
32: choice = 2
33: case 1:
34: print 1000
35: case 2:
36: print 2000
37: case 3:
38: print 3000
39: default:
40: print 9999
41: loopvar = 4
42: item = 0
43: L4:
44: t6 = item < loopvar
45: if t6 == false goto L5
46: print item
47: item = item + 1
48: goto L4
49: L5:

╔═══════════════════════════════════════╗
║          COMPILATION SUCCESSFUL       ║
╚═══════════════════════════════════════╝
```

```
root@LAPTOP-B9CHUST6:~/assgnmnt# gcc -o program output.c
root@LAPTOP-B9CHUST6:~/assgnmnt# ./program
15
25
40
25
500
350
1
2
3
4
5
2000
0
1
2
3
```

---

# 5. Conclusion

This compiler successfully translates a subset of Swift to executable C code through multiple compilation phases. It demonstrates understanding of compiler design principles including lexical analysis, parsing, semantic analysis, intermediate code generation, and target code generation. The implementation provides a foundation that could be extended to support more Swift language features.