Lab 6

Requirement

Statement: For the mini DSL, perform parsing (syntactical analysis) using yacc or ANTLR

Requirement 1: define grammar for mini-DSL

Requirement 2: perform parsing using yacc or ANTLR

Input: grammar, PIF

Output: string of productions

Requirement 3: integrate scanning with parsing (lex with yacc or ANTLR)

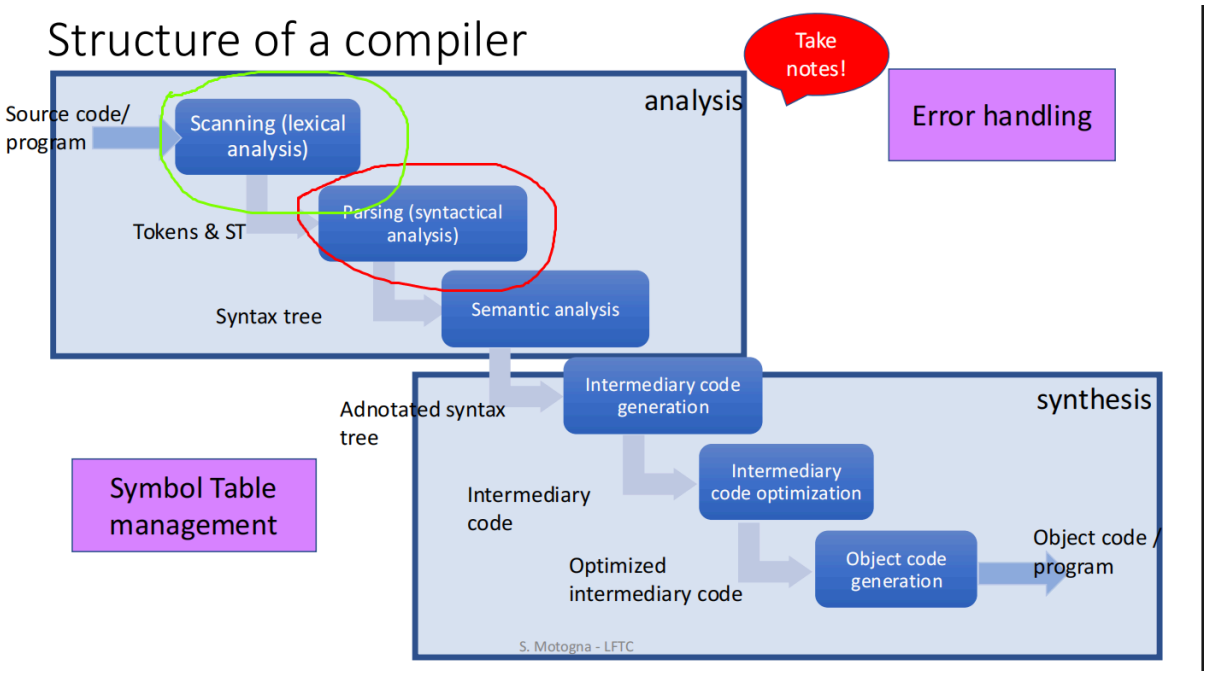
Delivery time: 2 week

Deliverables:

- 1 doc – grammar definition
- Source code

A short view back to the past... to the structure of a compiler from Lecture 1

Lab 3 - Lexical Analysis/Scanner: resulted in PIF (Program Internal Form) and ST (Symbol Table for identifiers and constants)



Lab 6 (this lab) - Syntactical Analysis/Parser: results in a syntax tree, yes/no based on syntactical correctness

---

Choose a way to perform the parsing

ANTLR

Check files from `Teams > Lab Assignments > Appendix Lab 6 > ANTLR`

- recommended if you chose to implement your scanner in Python, using the Scanning Algorithm from Lecture 1

YACC

- recommended if you chose to implement your scanner using Lex (as you can integrate lex for the 3rd requirement)
- https://gnuwin32.sourceforge.net/packages/bison.htm

> **Do not install it in a path containing spaces**

Requirement 1: define grammar for mini-DSL

- Check the BNF rules of your language defined in Lab 1
- Extract grammar rules from there
  - for yacc: in a `.y` file

  ▼ example

In BNF:

<program> ::= `begin` `:` <statement_list> `end`

In yacc: ** contents of *.y file **

```
%{
#include <stdio.h>
#include <stdlib.h>
#define YYDEBUG 1

%}

// tokens go here
%token BEGIN
%token END


%start program
%error-verbose

%%

program : BEGIN ':' statement_list END ;

%%

yyerror(char *s)
{
```

```
  printf("%s\n", s);
}

extern FILE *yyin;

main(int argc, char **argv)
{
  if(argc>1) yyin = fopen(argv[1], "r");
  if((argc>2)&&(!strcmp(argv[2],"-d"))) yydebug = 1;
  if(!yyparse()) fprintf(stderr,"\tO.K.\n");
}

** end of contents of *.y file **


where BEGIN and END are tokens defined in the *.lxi file:

begin       {printf( "Reserved word: %s\n", yytext); return BEGIN;}
end         {printf( "Reserved word: %s\n", yytext); return END;}
```

- for ANTLR: in a `.g4` file

▼ example

See `MyGrammar` from `Teams > Lab Assignments > Appendix Lab 6 > ANTLR > Example 2`

Requirement 2: perform parsing using yacc or ANTLR

- Ensure your grammar specification is free of errors
  - when running yacc commands or antlr commands, there are no errors
- For ANTLR see uploaded files from Teams: `antlr4 <grammarFile>.g4`
- YACC command:

```
bison -d <file_name>.y
```

Requirement 3: integrate scanning with parsing (lex with yacc or ANTLR)

- For ANTLR see `.g4` files from Teams
- For yacc, see example above - adapt the `.lxi` file from the Scanner lab accordingly (no more PIF & ST logic - just print each token as it is + no more Finite Automata logic for identifiers and constants)

▼ Commands for yacc and lex

```
flex <scannerName>.lxi
bison -d <parserName>.y
gcc <parserName>.tab.c lex.yy.c -o <executable_name>
<executable_name>.exe < <input_code>.txt > <output_code>.txt
```

Deliverables

- 1 doc – grammar definition
- Source code
- Parser output for a correct program and an incorrect one
  - for yacc: syntactical correctness (yes/no) + string of productions
  - for antlr: syntactical correctness (yes/no) + parse tree (either png or text)

Resources

Introduction to YACC - GeeksforGeeks

Your All-in-One Learning Portal: GeeksforGeeks is a comprehensive educational platform that empowers learners across domains-spanning computer science and programming,

https://www.geeksforgeeks.org/compiler-design/introduction-to-yacc/