CSC 448: Compiler Design

Lecture 3

Joseph Phillips
De Paul University

2015 April 14

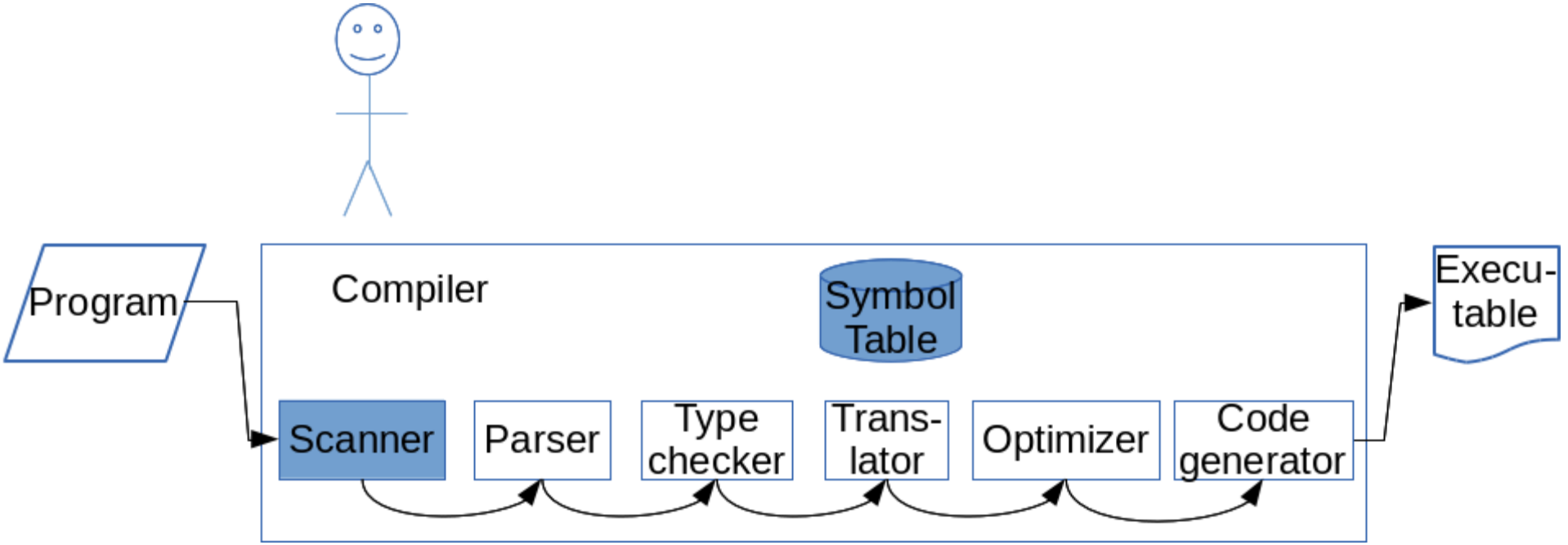
Reading

- Charles Fischer, Ron Cytron, Richard LeBlanc Jr. “Crafting a Compiler” Addison-Wesley. 2010.
  - Chapter 3: Scanning – Theory and Practice

Topics:

- Scanning (Practice)
- Flex

Overview:


Program
Compiler
Symbol Table
Execu-table
Scanner
Parser
Type checker
Trans-lator
Optimizer
Code generator

Remember our hand-coded parser?

```
Symbol*   scanner    ()  throw(const char*)
{
  while  ( isspace(inputCharStream_.peek()) )
    inputCharStream_.advance();

  if  ( inputCharStream_.isAtEnd() )
    return( &endSymbol );

  if  ( isdigit(inputCharStream_.peek()) )
    return( scanDigits() );

  char       ch     = inputCharStream_.peek();
  Symbol*   symbolPtr  = (Symbol*)
          malloc(sizeof(Symbol));

  inputCharStream_.advance();

  switch  (ch)
  {
  case '=' :
    symbolPtr->symbol_    = ASSIGN_SYMBOL;
    break;

  case '+' :
    symbolPtr->symbol_    = ADD_SYMBOL;
    break;

  case '-' :
    symbolPtr->symbol_    = SUBTRACT_SYMBOL;
    break;

  case 'p' :
    symbolPtr->symbol_    = PRINT_SYMBOL;
    break;

  case 'i' :
    symbolPtr->symbol_    = INT_DECLARE_SYMBOL;
    break;

  case 'f' :
    symbolPtr->symbol_    = FLOAT_DECLARE_SYMBOL;
    break;

  default :
    if  ( islower(ch) )
    {
      symbolPtr->symbol_  = ID_SYMBOL;
      symbolPtr->value_.varName_= ch;
      break;
    }

    throw "Unexpected character in input";
  }

  return(symbolPtr);
}
```

Remember our hand-coded parser?

```
Symbol*       scanDigits ()  throw()
{
  std::string   lex("");

  while  ( isdigit(inputCharStream_.peek()) )
  {
    lex += inputCharStream_.peek();
    inputCharStream_.advance();
  }

  Symbol* symbolPtr;

  if  (inputCharStream_.peek() != '.')
  {
    symbolPtr= (Symbol*)
      malloc(sizeof(Symbol));
    symbolPtr->symbol_    = INT_SYMBOL;
    symbolPtr->value_.integer_
= strtol(lex.c_str(),NULL,10);
  }
  else
  {
    lex += inputCharStream_.peek();
    inputCharStream_.advance();
    while  ( isdigit(inputCharStream_.peek()) )
    {
      lex += inputCharStream_.peek();
      inputCharStream_.advance();
    }

    symbolPtr= (Symbol*)
      malloc(sizeof(Symbol));
    symbolPtr->symbol_    = FLOAT_SYMBOL;
    symbolPtr->value_.floatPt_
= strtod(lex.c_str(),NULL);
  }

  return(symbolPtr);
}
```

Eeww! What a pain-in-the-ass!
Ain’t there a better way?

A: Indeed there is! It’s called ***flex***

History of flex

- lex:
  - “Lexical Analyzer”
  - Written by Mike Lesk and Eric Schmidt for ATT Unix in 1970s
  - Uses regular expression to define lexemes
  - C code actions do-able for each
- flex:
  - “fast lexical analyzer”
  - Written by Vern Paxson circa 1987

Our first lex/flex program (1)

```
%{
  //  Compile with:
  //  $ flex -o ex1_echoer.c ex1_echoer.lex
  //  $ gcc ex1_echoer.c -o ex1_echoer
%}
%%
\n      printf("\n");
.       printf("%c",yytext[0]);
%%

intyywrap() { return(1); }

intmain ()
{
  yylex();
  return(0);
}
```

Our first lex/flex program (2)

- I know! I know! It looks funny . . . just compile and run it!

```
$ flex -o ex1_echoer.c ex1_echoer.lex
$ gcc ex1_echoer.c -o ex1_echoer
$ ./ex1_echoer
I type something
I type something
I type something else
I type something else
Hey! Are *YOU* copying me?!?!
Hey! Are *YOU* copying me?!?!
^C    # I typed Ctrl-C
```

Our first lex/flex program (3)

- So, it just copies what I type:

```
$ ./ex1_echoer < ex1_echoer.lex
%{
  //  Compile with:
  //  $ flex -o ex1_echoer.c ex1_echoer.lex
  //  $ gcc ex1_echoer.c -o ex1_echoer
%}
%%
\n      printf("\n");
.       printf("%c",yytext[0]);
%%

int yywrap() { return(1); }

int main ()
{
  yylex();
  return(0);
}
```

Our first lex/flex program (4)

- Now let's try to understand the bad-boy:
- In general:

```
%{
  // #includes and C-externs go here
%}
%%
/* regular expressions and their code
goes here */
%%
C functions go here
```

Our first lex/flex program (5)

```
%{
// Compile with:
// $ flex -o ex1_echoer.c ex1_echoer.lex
// $ gcc ex1_echoer.c -o ex1_echoer
%}
```

- **Not much going on here . . . just storing some comments**

Our first lex/flex program (6)

```
%%
\n        printf("\n");
.         printf("%c",yytext[0]);
%%
```

- **Regular expression rules to match**
  - Period (`.`) matches any char,
  - `\n` means newline
- **`printf(..)` is the C-code to do when regular expression matches**
  - `yytext[ ]` holds the input read so far

Our first lex/flex program (7)

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


int main ()
{
  yylex();
  return(0);
}
```

- `yylex()` is the C function that lex/flex produces
- `yywrap()` is a function/macro that yylex() uses:
  - Return value 0: keep on reading when reach EOF
  - Return value 1: return the zero-token to report EOF

Pressing Ctrl-C was annoying! Another solution? (1)

```
%{
  //  Compile with:
  //  $ flex -o ex2_echoer.c ex2_echoer.lex
  //  $ gcc ex2_echoer.c -o ex2_echoer
%}
%%
quit    return(0);  // This is our new rule!
\n      printf("\n");
.       printf("%c",yytext[0]);
%%
intyywrap() { return(0); }
intmain ()
{
  printf("Type \"quit\" to quit:\n");
  yylex();
  return(0);
}
```

Pressing Ctrl-C was annoying! Another solution? (2)

```
$ flex -o ex2_echoer.c ex2_echoer.lex
$ gcc ex2_echoer.c -o ex2_echoer
$ ./ex2_echoer
Type "quit" to quit:
Hello
Hello
Goodbye
Goodbye
quit
$
```

Your Turn!

Any problems with that rule for reading files?

Your Turn Again!

Write a Lex program that counts the number of newline characters, and the total number of chars.

(Similar to Unix’s wc)

And Again!

Write a program to count the number of vowels in a file

Lex regular expressions (1):

- Rules
  - Period (.) Matches any char
  - | The thing before or after
  - [ ] Defines character class (all included)
    - [0123456789] and [0-9] both define digits
  - [^] Defines character class
    - Everything but all included
    - [^0-9] defines all chars other than digits

Lex regular expressions (2):

- Rules
  - ( ) Just used for grouping
  - * Matches 0 or more of previous thing
  - + Matches 1 or more of previous thing
  - ? Matches 0 or 1 of previous thing
  - { } How many times the previous thing should match
    - A{1,3} matches ‘A’ between 1 to 3 times
  - / Match preceding thing only if followed by following thing
    - 0/1 matches 0 only when followed by 1 (1 still in input)

Lex regular expressions (3):

- Rules:
  - ^ Match at beginning of line
    - **^begin** only matches “**begin**” at beginning of line
  - $ Match at end of line
    - **end$** only matches “**end**” at end of line
  - \ C-escape sequence
  - "..." Interpret all enclosed characters literally (other than C escape chars)

Regular expression rule example (1)

```
%{
  //  Compile with:
  //  $ flex -o ex4_echoer.c ex4_echoer.lex
  //  $ gcc ex4_echoer.c -o ex4_echoer
%}
%%
\n                      // Ignore newlines
("+"|-)?[0-9]+          { int i=atoi(yytext); printf("Integer: %d\n",i); }
[A-Za-z_][A-Za-z_0-9]*  { printf("Identifier: %s\n",yytext); }
.                       // Ignore other chars
%%
int     yywrap  ()      { return(1); }
int     main    ()
{
  yylex();
  return(0);
}
```

Regular expression rule example (2)

```
$ ./ex4_echoer < ex4_echoer.lex
Identifier: Compile
Identifier: with
Identifier: flex
Identifier: o
Identifier: ex4_echoer
Identifier: c
Identifier: ex4_echoer
Identifier: lex
Identifier: gcc
Identifier: ex4_echoer
Identifier: c
Identifier: o
Identifier: ex4_echoer
Identifier: n
Identifier: ignore
Identifier: newline
Integer: 0
Integer: -9
. . .
```

Your Turn!

Revise the previous program to handle floating point numbers
(including scientific notation)

Your Turn Again!

Re-write the program that counts the number of vowels in a file

Functions yylex uses:

- yywrap()
- YY_INPUT
  - For block read
- input()/yyinput()
- unput()

Other helper variables and functions