Monday, August 30, 2021 12:00 PM

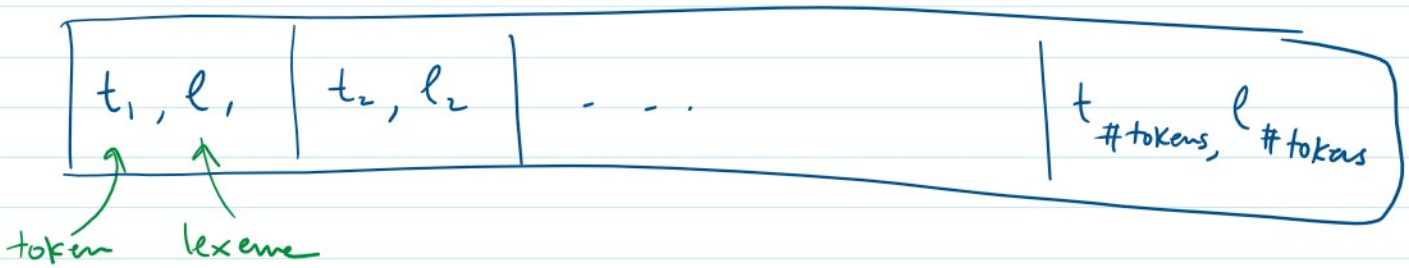
8-30-2021 CSE 340

Last time
- Syntax analysis
- getToken() function and its behavior
- difference between syntax and semantics

Today
- getToken()
- peek()
- recursive descent parsing

getToken() and peek()

Assume that the input is already broken down into a sequence of tokens that are stored in an array



we also have an index that initially points to the first entry (index = 1 initially)
If there are no tokens, the array is empty and # tokens = 0 but index = 1 initially.

if index > # tokens

getToken() : if index > # tokens
return EOF

else
tok = token_array[index]
index = index + 1 // consume the token
return tok;

---

peek(int howfar): if index + howfar - 1 > # tokens
return EOF

howfar to peek. > 0

else
return
token_array[index + howfar - 1]

// does not consume
// tokens and does
// not change index

Example token list IF = {"if"}
ID
NVM

input: if if␣if␣if␣iff␣123

space which is a delimeter

1	2	3	4
ID, "if if if"	IF, ""	ID "iff"	NVM "123"

Initially index = 1

getToken() ⟶ ID, "if1if" index = 2

peek(1) ⟶ IF, "if"

peek(2) ⟶ ID "iff"

peek(4) ⟶ EOF

peek(1) ⟶ IF, "if"

getToken() ⟶ IF, "if"

peek(1) ⟶ ID, "iff"

Parsing by example

Grammar for expressions

expr ⟶ term PLUS expr

expr ⟶ term

term ⟶ factor MULT term

term ⟶ factor

factor ⟶ NUM

factor ⟶ ID

factor ⟶ LPAREN E RPAREN

left and right parentheses

top level symbol

E ⟶ T PLUS E

grammar rule

LHS: left hand side

RHS right hand side

E ⟶ T

T ⟶ F MULT T

T ⟶ F

F → NUM | ID | LPAREN E RPAREN

OR ↑ Example (1+2)

The LHS's of rules are non-terminals

The RHS's of rules are sequence of non-terminals and terminals (tokens)

E → T PLUS E

NT NT Terminal (token) NT

Rule

Example 1 + 2 * 3

NUM PLUS NUM MULT NUM

F F F

T T

T PLUS E

E

E → T + E
E → T
T → F * T
T → F
F → ID | NUM | (E)

Recursive descent parsing:

- One parsing function for each non-terminal
- When parse_X() is called at a given point in the input:
  - an X will be "consumed" no more no less OR
  - a syntax error is raised

- to "parse" terminals, we call gettoken()

```
parse_input()  // input consists of one expr
{
    parse_E();
    t = gettoken();
    if (t.token_type != EOF)
        syntax_error();
}
```

1 + 2 * 3 before

1 + 2 * 3 after (1 + 2 consumed)

≡

```
expect(EOF)
```

```
parse_F()
{
    // F -> NUM
    // F -> ID
    // F -> ( E )

    t = peek(1);
    if (t.token_type == NUM)
        expect(NUM);        // F -> NUM
    else if (t.token_type == ID)
        expect(ID);         // F -> ID
    else if (t.token_type == LPAREN)
    {
        expect(LPAREN);     // (
        parse_E();          // E
```

```
        expect (LPAREN);        // (
        parse_E();              // E
        expect (RPAREN);        // )
    }
    else
        syntax_error();
}

parse_T()
{   // T -> F MULT T
    // T -> F
    parse_F();                          // F
    t = peek(1);
    if (t.token_type == MULT)
    {
        expect (MULT);                  // MULT
        parse_T();                      // T
    } else if ((t.token_type == EOF) |
               (t.token_type == RPAREN) |
               (t.token_type == PLUS))
        return;
    else
        syntax_error();
}
```