CPSC 326: ASTs

Quiz 1 due Tuesday

Generating Abstract Syntax Tree (AST's)

- Second half of parser
- AST's used for a number of steps
- Parser builds AST instance as it parses
- AST's used in
  - Type checking (Semantic Analysis)
  - Optimization
  - Generate assembly code
  - Translate into another language (Transpiler)
  - Cleanly print out computer code
    - Most IDE's impliment a parser & Lexer so it can do type & error checking for you
- AST's very similar to expression trees
  - nodes contain operators or values

simple Examples

<stmt_list> ::= VAR ASSIGN <expr> <stmt_list_tail>

<stmt_list_tail> ::= SEMICOLON <stmt_list> | ε

<expr> ::= VAR <expr_tail>

<expr_tail> ::= PLUS VAR | MINUS VAR | ε

```
Class Parser() {
  Public:
    Parser(&Lexer);
    Void parse;
  Private:
    advance();
    eat();
    error()
    Void stmt_list();
    Void stmt_list_tail();
    Void expr();
    Void expr_tail();
};
```

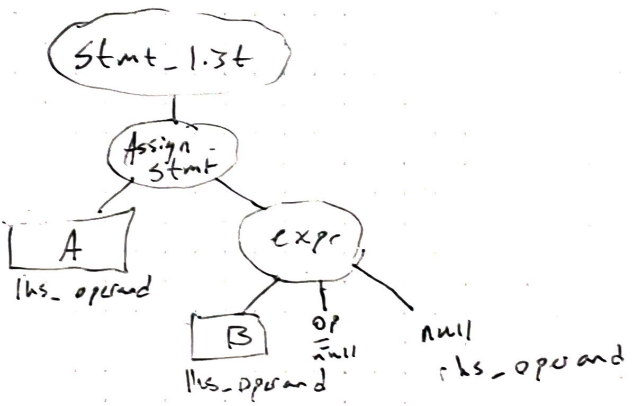
- Our AST nodes would include
  - stmtlist => } class Representation
  - Stmt =>
  - expr => Expr

```
Class Expr
{
  Public:
    Token lhs_operand;
    Token* op=nullptr;
    Token* rhs_operand = nullptr;
    ~Expr() { delete op, delete rhs}
};
```

```
Class AssignStmt {
  Public:
    Token lhs_var;
    Expr* expr;
};

Class StmtList {
  Public:
    std::List <stmt*> stmts;
  ~StmtList() { for (stmt* s : stmts) delete s; }
```

Ex "A = B"



```
void parse (& stmt_list node)
{
  advance();
  stmt_list(node);
  eat('EOS', "Expected End");
}
```

New parse function to write

```
void stmt_list(& stmt_list node)
{
  AssignStatement* s = new AssignStatement;
  s->lhs_var = curr_token;
  eat(VAR, "...");
  eat(ASSIGN, "..."); // isn't stored as part of s
  expr* e = new expr;
  s-> expr = e;
  expr(*e);
  node.stmts.push_back(s);
  stmt_list_tail(node);
}
```

```
void expr (& expr e)
{
  e->lhs_operand = curr_token;
  eat(VAR, "...");
  expr_tail(e);
}
```

```
void expr_tail (& expr e)
{
  if (curr_token == PLUS)
  {
    eat(PLUS, "...")
    e->op = new Token(P
    e->rhs_op = curr_tok
  }
  else minus
```