VE280 Lab6

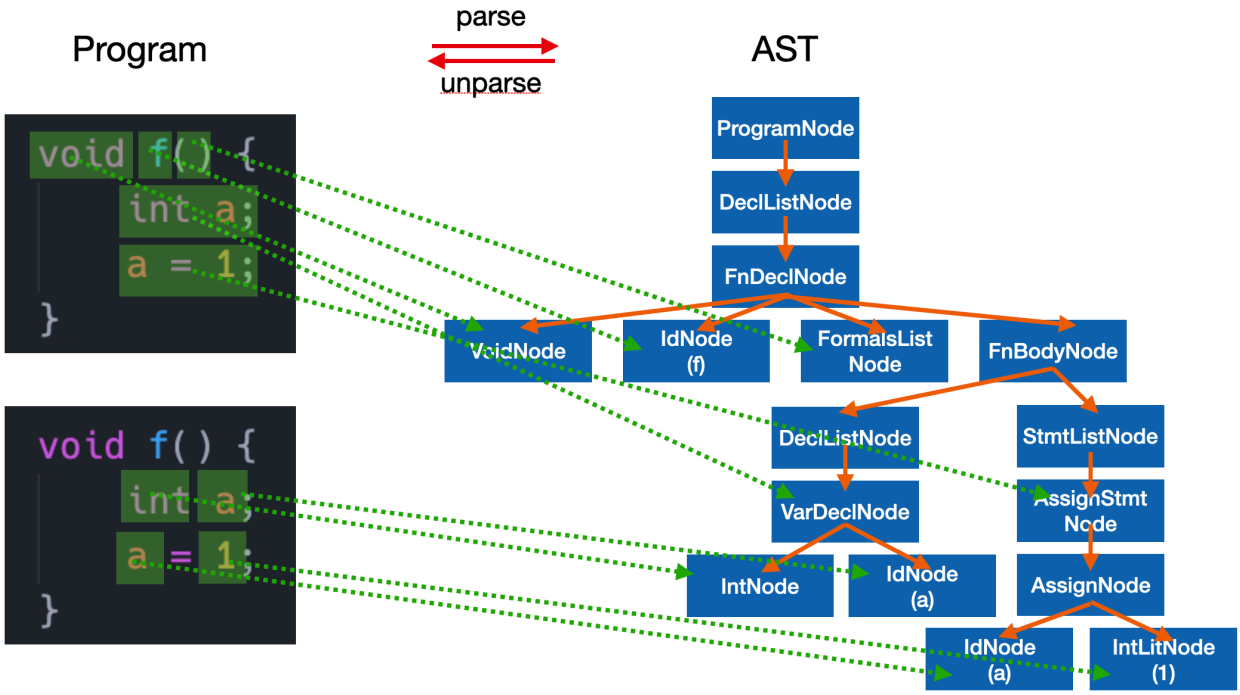
Out: 00:00, Jun 23, 2020; **Due**: 23:59, Jun 30, 2020

This lab contains only one exercise. Lucky you!

Ex1. Abstract Syntax Tree Unparser

Problem: In compilers a **program** is usually converted into a tree called **Abstract Syntax Tree (AST)** and this process is called parsing. Imagine you are implementing a compiler, and you want to know whether the AST is correct. You can write a tool to convert the AST back to a program to see if it is correct or not, and this process is called unparsing. In this exercise, you will implement this AST unparser.

So, what is a program and what is an AST? The following is a simple example.



Input Format: An integer `n (0 < n < 100)` that stands for `n` nodes, followed by `n` lines of AST nodes with format <NodeType> <ChildNum> [OptionalParam]

```
# sample input corresponding to above picture
16
ProgramNode 1 # ProgramNode with one child
DeclListNode 1
FnDeclNode 4
VoidNode 0
IdNode 0 f # IdNode with 0 child and name f
FormalsListNode 0
FnBodyNode 2
DeclListNode 1
StmtListNode 1
```

```
VarDeclNode 2
AssignStmtNode 1
IntNode 0
IdNode 0 a
AssignNode 2
IdNode 0 a
IntLitNode 0 1 # IntLitNode with 0 child and value 1
```

Output Format: The program corresponding to that AST. Additional or less space will not affect correctness.

```
# sample output
void f() {
    int a;
    a = 1;
}
```

Tasks: Here are the general steps to complete this exercise (see below for hints):

1. Get familiar with the classes in the starter file.
2. Implement the `unparse()` method of each node (inside classes).
3. Save each line of input as a node into an array (inside main).
4. Traverse the array and construct the AST (inside main).
5. Call the `unparse()` method of the AST root to print the whole program (inside main).
6. Free the allocated memory of nodes in the array (inside main).

Step 1 Hints: First get familiar with the classes given in the starter file, especially the ASTNode base class. Think about what data members and function members each subclass have access to. Here is the AST node class description. You can also find it at the top of the starter file `ex1.cpp`.

```
//     Subclass           Children                                    Description
//     --------           --------                                    -----------
//   ASTNode:                                                        the base class
//     ProgramNode        DeclListNode                                the whole program
//     DeclListNode       a list of DeclNode                          a list of declarations, e.g. int a; bool b;
//
//     DeclNode:                                                      a declaration
//       VarDeclNode      TypeNode, IdNode                            a variable declaration, e.g. int a;
//       FnDeclNode       TypeNode, IdNode, FormalsListNode, FnBodyNode  a function declaration, e.g. void f(...){...}
//       FormalDeclNode   TypeNode, IdNode                            a funtion parameter, e.g. int a
//
//     FormalsListNode    a list of FormalDeclNode                    a list of function parameters, e.g. (int a, bool b)
//     FnBodyNode         DeclListNode, StmtListNode                  the function body
//     StmtListNode       a list of StmtNode                          a list of statements
//
//     TypeNode:                                                      a type keyword
//       IntNode          -- none --                                  the keyword int
//       BoolNode         -- none --                                  the keyword bool
//       VoidNode         -- none --                                  the keyword void
//
//     StmtNode:                                                      a statement end with semicolon ;
//       AssignStmtNode     AssignNode                                an assign statement, e.g. a=1;
//       PostIncStmtNode    ExpNode                                   a post increase statement, e.g. a++;
//       PostDecStmtNode    ExpNode                                   a pose decrease statement, e.g. a--;
//
//     ExpNode:                                                       an expression
//       IntLitNode         -- none --                                an int literal, e.g. 1
//       TrueNode           -- none --                                the keyword true
//       FalseNode          -- none --                                the keyword false
//       IdNode             -- none --                                an identifier, e.g. a, f, main
//       AssignNode         ExpNode, ExpNode                          an assignemnt expression, e.g. a=1 (notice no ";")
```

- Indentation means subclass.
  e.g. *VarDeclNode* is a subclass of *DeclNode*.
- Each class contains a data member `ASTNode* children[]`. The order in the array should be the same as the children in the above description.
  e.g. for *FnDeclNode* Class, `children[0]` points to *TypeNode*, `children[1]` points to *IdNode*, `children[2]` points to *FormalsListNode*, `children[3]` points to *FnBodyNode*.
- The child number of most classes are fixed. Only DeclListNode, FormalsListNode, and StmtListNode has variable number of child. The input provides child number for all classes just for consistency.
- The above figure is a complete list of classes, which means the printed program will not contain statements that cannot be represented by the above classes.
  e.g. doesn't contain control statements `if(){}`, arithmetic expression `a+b`, ...

Step 2 Hints: After you get familiar with the classes, you can start to think what's the job of the `unparse()` method in each class. Notice that the `unparse()` method of each node is to simply **print the corresponding code snippet of that node and its children**. And that means it might have to call the `unparse()` method of its children. The method is not complicated and most of them are within `5` lines!

Notice although additional/less space will not affect the correctness, it's a good habit to have the code well indented. This is achieved by the `addIndent(int indent)` method of each node. The **indent value** should be decided by its parent. Whether or not the `unparse()` calls `addIndent()` is determined by the node, here *VarDeclNode*, *FnDeclNode* and *subclass of StmtNode* should call `addIndent()`.
e.g. If *FnBodyNode* has indent value=0, then its children, *DeclListNode*, *StmtListNode* should have indent value=0+4=4

You should really try to understand what each class stands for in a program with the help of provided class description before you start writing the `unparse()` method. It would be very helpful to start with simple examples: write a simple program and try to write its AST by hand, just like the first example in the figure. If you get lost, draw the tree, point to the nodes by your finger, and see whose `unparse()` method is called first and

whose is called next. **The unparse() method of ProgramNode, DeclListNode, and FnDeclNode are provided to help you understand.**

Step 3 Hints: This part is easy, simply read each line from the input, allocate a new node based on the read information. However, you may write a bunch of if statements to find out which class it belongs to.
e.g. `if(!read_type.compare("ProgramNode")) {nodes[0] = new ProgramNode(num_child);}`.

Step 4 Hints: This part is a little bit tricky. If you compare the sample text input and the AST figure, you can find that the text input is actually reading the AST from top to down, from left to right. This is actually a Bread First Search of that AST. Think creatively how you can construct the AST from the array you get in **step 2**. It can be implemented within 6 lines with a nested loop.

Step 5 Hints: This step is already done by the starter file.

Step 6 Hints: This part is also easy, simply free the space you allocated in **step 3**.
e.g. `delete(nodes[0])`

Takeaway: In this exercise, you wrote an unparser that converts an AST into a program and got familiar with the knowledge of class inheritance.

Tags: `#ADT` `#subtype` `#inheritance` `#virtual function` `#interface/abstract base class`

Files

2. *lab6.pdf*: description in pdf
3. *lab6.zip*: starter files

Submission

Compress each **.cpp** file into a **.tar** file and submit to JOJ.
Do not change the name of the **.cpp** file.

Created by: Yiqing Fan
Last update: Jun 22, 2020
@ UM-SJTU Joint Institute