

CS 3721: Programming Languages Lab

Lab #11: Subtype Polymorphism in C++

Due date: April 22, 3:30pm. At the beginning of the next recitation.

Goals of this lab:

- Be able to use eclipse to write C++ programs
- Be able to read/write C++ programs using subclassing and overridden virtual functions to obtain subtype polymorphism.
- Learn a Design Pattern

1. *Base Class.* Today, we will implement a simple AST, and then apply the visitor design pattern to print out this AST in preorder. First of all, we need a base *class ASTNode*. So define a base *class ASTNode* having *leftChild* and *rightChild* pointing to left and right subtrees respectively. Initialize your left and right child pointers to 0 (NULL) in constructor, and implement set/get methods according to the following prototypes:

```
ASTNode *getLeft();
ASTNode *getRight();
void setLeft(ASTNode *);
void setRight(ASTNode *);
```

Please, use exactly the same names for the names of the methods because you may need to use the function *ConstructAST* written for you to construct an AST to save time.

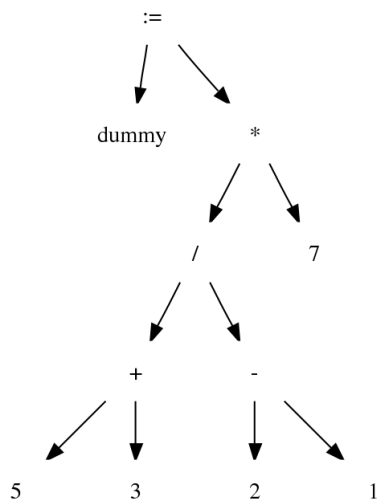
2. *Subclasses.* Our simple AST will be composed of identifiers, numbers, operators, and assignments. So, we need the subclasses IdentifierNode, LiteralNode, OperNode, and AssignmentNode derived from the base class ASTNode. In addition to the left and right pointers, the IdentifierNode class will have the member, *name*, which is type of *string*; the LiteralNode class will have the member, *number*, which is type of *int*; the OperNode class will have the member, *oper*, which is type of *char*. We don't need any additional member for the subclass AssignmentNode. You will also need get methods while printing the values of the nodes, so implement the get method for the subclasses. Moreover, implement the constructors with the following prototype:

```
IdentifierNode(string) // initialize the member, name
LiteralNode(int) // initialize the member, number
OperNode(char) // initialize the member, oper
```

Hint: Use public inheritance to make the derived class a *subtype* of the base class.

3. *Visitor Design Pattern.* Now, we will apply Visitor Design Pattern to print out our AST. In order to achieve this, you should define a pure abstract base class *Visitor*. Don't forget to make the methods virtual to dispatch them dynamically. Then, you should define and implement a subclass PrintVisitor. Moreover, you have to add virtual *accept* method to each of your node classes in order to call back the *visit()* method for its class. The resulting output should be in preorder, so implement the *accept()* methods accordingly. *Hint:* Have a look at the *Car* example that we discussed.

Testing your code: The function, *ConstructAST*, creates the following AST. Expression:
 dummy = ((5 + 3) / (2 - 1)) * 7



And, the function, *ConstructAST*:

```

void ConstructAST(ASTNode *&root)
{
    root = new AssignmentNode();
    ASTNode *temp = root;
    temp->setLeft(new IdentifierNode("dummy"));
    temp->setRight(new OperNode('*'));

    temp = temp->getRight();
    temp->setLeft(new OperNode('/'));
    temp->setRight(new LiteralNode(7));

    temp = temp->getLeft();

    temp->setLeft(new OperNode('+'));
    temp->setRight(new OperNode('-'));
    ASTNode *tempLeft = temp->getLeft();
    ASTNode *tempRight = temp->getRight();

    tempLeft->setLeft(new LiteralNode(5));
    tempLeft->setRight(new LiteralNode(3));
    tempRight->setLeft(new LiteralNode(2));
    tempRight->setRight(new LiteralNode(1));
}
  
```

Then, you can test your code with the following:

```
int main()
{
    ASTNode *root = 0;
    ConstructAST(root);
    PrintVisitor visitor;
    if(root)
        root->accept(visitor);
    return 0;
}
```

The output should be:

:= dummy */ + 5 3 - 2 1 7

Save your code in a file named lab11.cpp

Submit your code electronically via email to btas@cs.utsa.edu