CSCI 4160 Project4

Due: see class calendar

Goal:

This assignment is used to create the Abstract Syntax Tree for Tiger language.

Description:

This is an individual project that build on the previous team project. What you need to do in this assignment is to define actions for each production of Tiger language defined in tiger.yy. Instructor provides both tiger.ll and tiger.yy files. These actions will create the abstract syntax tree (AST). Before defining production actions, make sure you understand the following classes, which define the AST node types within the namespace. You can find all class definitions in Absyn.h. In this assignment, you only need to understand the constructor of each class.

class Absyn; This abstract base class is the ancestor of all nodes. It contains two member data: lineno and colon, which represents the location information of the construct stored in the node.

//All kinds of expression nodes

class Exp; This abstract base class is the parent of all expression nodes

- ~~class ArrayExp; represents array expression like: score[10] of 1 where score is an integer array type declared earlier, 10 is the size of the array and 1 is the initial value.~~
- class AssignExp; represents assignment expression like: x := 5 + 9;
- class BreakExp; represents the break expression.
- class CallExp; represents the function call expression like: gcd(x, 20);
- class ForExp; represents the for loop.
- class IfExp; represents an if-statement
- class IntExp; represents an integer literal like: 10
- class LetExp; represents a let expression
- ~~class NilExp; represents the constant Nil.~~
- class OpExp; represents the expressions involving a binary operator, for example: x < y
- ~~class RecordExp; represents a record expression like: myrecord{f1=10, f2="a string"};~~
- class SeqExp; represents a sequence of expressions separated by semicolon like: x := 5; y := 8; z := x + y.
- class StringExp; represents a string literal like "This is a string literal".
- class VarExp; represents a lvalue such as simple variable (like x), field variable (like sect.x1 where sect is a record variable and x1 is a field name of the record) or subscript variable (a[3] where a is an array variable).
- class WhileExp; represents a while statement.

//All kinds of type nodes

class Ty; This abstract base class is the parent of all type nodes

- ~~class ArrayTy; represents an array type~~
- class NameTy; represents an alias of an existing type in Tiger language. Such type is defined using type declaration like: type myint = int
- ~~class RecordTy; represents a record type~~

//All kinds of declaration nodes

class Dec; This abstract base class is the parent of all declaration nodes

- class FunctionDec; represents a function declaration
- class TypeDec; represents a type declaration
- class VarDec; represents a variable declaration

//all kinds of variable nodes
class Var; This abstract base class is the parent of all variable nodes

- ~~class FieldVar; represents a field variable such as sect.x1 where sect is a record variable and x1 is a field name of the record~~
- class SimpleVar; represents a simple variable such as x.
- ~~class SubscriptVar; represents an array element such as a[3] where a is an array variable.~~

//all types of list nodes
class DecList; represents a list of declaration list of LetExp.
class ExpList; represents a list of expressions used in SeqExp, or CallExp.
~~class FieldExpList; represents a list of field expression list like: f1=10, f2="strings", f3=30. This class is typically used with RecordExp.~~
class FieldList; represents a sequence of names and their data types. This class is typically used to represents the field names of a record type, or the parameter list of a function declaration.

Set up environment:
I strongly suggest teams to use the sample Visual Studio solution provided by me in the class repository.

1. Use tiger.ll and tiger.yy provided by the instructor.
2. Add actions to each grammar rule in tiger.yy file, which is the only file you need to work on. Please ignore ALL PRODUCTIONS THAT PERFORM ERROR RECOVERY.

How to handle Boolean expression like exp1 AND exp2; exp1 OR exp2?
Treat both expressions as if-else statements. More specifically,
exp1 AND exp2 is equivalent to: if exp1 then exp2 else 0
exp1 OR exp2 is equivalent to: if exp1 then 1 else exp2

Instructor provided files in the class repository

The following files in project4 folder are provided by the instructor:

- AbsynProject folder. This folder contains a sample Visual Studio 2010 project.
  - Skeleton source files provided in the sample project are listed below:
    - tiger.ll: an empty file. Should be replaced.
    - lex.yy.c: generated by Flex when compiling tiger.ll
    - tiger.yy: a skeleton file for tiger CFG. Please follow the instructions to modify it.
    - tiger.tab.hh & tiger.tab.cc: generated by Bison when compiling tiger.yy.
    - tiger.output: debug file for CFG
    - ErrorMsg.h: contains the definition of error handler
    - Print.h and Print.cpp: used to print the abstract syntax tree.
    - Absyn.h: contains class definition for all AST node types.
    - main.cpp: the driver
    - example.tig: test program of tiger language
- Description4.pdf: this file
- Rubric4.doc: the rubric used to grade this assignment.
- example.txt. sample output when parsing example.tig file

How to submit

1. Once you have finished, submit the project in the following way:

- Copy the file projects/project4/rubric4.doc from the class repository to the project4 folder in your local repository of project4. Edit the file to put your name.
- Commit the whole project4 folder to your local repository.
- Push all the changes to master repository on ranger.
- **Any commit of the project after the deadline is considered as cheating. If this happens, the latest version before the deadline will be graded, and you may receive up to 50 points deduction.**

2. You can also check your overall grade by update the rubric4.doc from the repository after the notice from the instructor.