// You will need these forward references.

class Expr;

class Stmt;

// Runtime Global Variables

int pc; // program counter

vector<string> lexemes;

vector<string> tokens;

vector<string>::iterator lexitr;

vector<string>::iterator tokitr;

map<string, int> vartable; // map of variables and their values

vector<Stmt \*> insttable; // table of instructions

map<string, string> symboltable; // map of variables to datatype (i.e. sum t\_integer)

// Runtime Global Methods

void dump(); // prints vartable, instable, symboltable

// You may need a few additional global methods to manipulate the global variables

// Classes Stmt and Expr

// It is assumed some methods in statement and expression objects will change and

// you may need to add a few new ones.

class Stmt{ // statements are executed!

private:

string name;

public:

Stmt(){}

virtual ~Stmt(){};

virtual string toString() = 0;

virtual void execute() = 0;

};

class AssignStmt : public Stmt{

private:

string var;

Expr\* p\_expr;

public:

AssignStmt();

~AssignStmt();

string toString();

void execute();

};

class InputStmt : public Stmt{

private:

string var;

public:

InputStmt();

~InputStmt();

string toString();

void execute();

};

class StrOutStmt : public Stmt{

private:

string value;

public:

StrOutStmt();

~StrOutStmt();

string toString();

void execute();

};

class ExprOutStmt : public Stmt{

private:

Expr\* p\_expr;

public:

ExprOutStmt();

~ExprOutStmt();

string toString();

void execute();

};

class IfStmt : public Stmt{

private:

Expr\* p\_expr;

int elsetarget;

public:

IfStmt();

~IfStmt();

string toString();

void execute();

};

class WhileStmt : public Stmt{

private:

Expr\* p\_expr;

int elsetarget;

public:

WhileStmt();

~WhileStmt();

string toString();

void execute();

};

class Expr{ // expressions are evaluated!

public:

virtual int eval() = 0;

virtual string toString() = 0;

virtual ~Expr(){}

};

class ConstExpr : public Expr{

private:

int value;

public:

ConstExpr(int val);

int eval();

string toString();

};

class IdExpr : public Expr{

private:

string id;

public:

IdExpr(string s);

int eval();

string toString();

};

class InFixExpr : public Expr{

private:

vector<Expr \*> exprs;

vector<string> ops; // tokens of operators

public:

~InFixExpr();

int eval();

string toString();

};

class Compiler{

private:

void buildIf();

void buildWhile();

void buildStmt();

void buildAssign();

void buildInput();

void buildOutput();

// use one of the following buildExpr methods

void buildExpr(Expr\*&); Expr\* buildExpr();

void populateTokenLexemes(istream& infile);

void populateSymbolTable(istream& infile);

public:

Compiler(istream& source, istream& symbols){

populateTokenLexemes(source);

populateSymbolTable(symbols);

}

bool compile(); // builds the instruction table and declaration check

void run(); // executes the instruction table

};