Formal Languages & Compilers 2020/21

Compiler project

Authors: Sultan Balawal, Sosa Ulises Emiliano, Dalvai Samuel

Simplified Basic-C Compiler

The project developed parses and calculates statements written in a C-like fashion. It can initialize variables of type bool and int, compute the 4 basic arithmetic operations plus the power operation, and perform comparison of variables. Lastly, the parser does type-checking on its primitive data-types.

Although scoping and control-flow structures have not been fully implemented, (a stack structure is necessary in order to compute the correct execution branch at the right moment) it can parse them correctly.

Instead of declaring precedence rules we have developed our grammar in an unambiguous fashion, so that the operations with a higher precedence are computed closer to the leaf level in the parse tree. By doing so we have that a multiplication is computed before the addition but after a unary operation such as the negative value of a variable.

The grammar of Basic-C

```
program -> program stmt | ε

stmt -> varDeclInit ;
        | simpleExp ;
        | if (simpleExp) {stmt}
        | if (simpleExp) {stmt} else {stmt}
        | while (simpleExp) do  {stmt}
        | return ;
        | return simpleExp ;
        | print simpleExp ;

varDeclInit -> typeSpec varDeclId simpleExp | varDeclId : simpleExp

varDeclId -> ID

typeSpec -> int | bool

simpleExp -> simpleExp or andExp | andExp

andExp -> andExp and unaryRelExp | unaryRelExp

unaryRelExp -> not unaryRelExp | relExp

relExp -> sumExp relOp sumExp | sumExp

relOp -> <= | < | >| >= | == | !=

sumExp ->  sumExp sumOp mulExp | mulExp

sumOp -> + | -

mulExp -> mulExp mulOp unaryExp | unaryExp

mulOp -> * | / | **

unaryExp -> - unaryExp | NUM | FALSE | TRUE | ID | (simpleExp)
```

Compiling and running a program

This is how a program in our Basic-C Compiler can start. In this example the *input.txt* file will contain the program statements to be parsed:

```
$user:flex Analyzer.l
$user:yacc -vd Parser.y
$user:gcc y.tab.c -ly -ll -o compiler.out
$user:./compiler.out<input.txt
```

- Or if make is installed:

```
$user:make
$user:./compiler.out<input.txt
```

- It can also receive input interactively from the command line if run in this way:

```
$user:./compiler.out
```

Valid input example

This is an example of a valid program parsed by our compiler:

```
bool t : true;
bool f : false;
print f or t;

if(t){print f;}

if(t == f){
    print true;
} else {
    print t;
}

while (t) do { print f; }

return;
```

With the above input the compiler will produce the following output.

```
1
0
The condition is true
1
1
This is the else branch executed;
0
The while loop should execute here
Exiting program
```

As explained above, the **if**, **then**, **else** and **while** statements are only parsed by the compiler, the statements contained in the conditional branches are executed regardless of the condition being satisfied or not. Moreover, the statements contained in the body between { } are executed before the actual evaluation of the condition, this happens because the parser executes the code associated to a given rule immediately when that rule is recognized.

Invalid input examples

- Invalid variable declaration and initialization:

```
bool x : 100;
int x;
```

It is not possible to assign a value which is not equal to **1**, **0**, **true** or **false** to a boolean variable . Moreover, a variable when declared has to be initialized to some value. The statement `bool x : 100;` will issue the following error:

```
Error: invalid assignment of int value to bool variable!
```

- The following are invalid operations:

```
x : 5 + true;
bool result : (5 or 5);
int res : (true + true);
print (12 and true;)
```

Operations between different types of values (eg. integer and boolean) are not allowed, moreover, operations like logical and, or, not, are not allowed between integers. The same holds for typical integer operations like *sum* between boolean values. The statement `int res : (true + true);` will issue the following error:

```
Error: you cannot sum two bool!
```

- Variable redeclaration

```
int x : 2;
int x : 3;
```

Though it is possible to assign a new value to a variable, it is not possible to redeclare a variable with the same name. The statement above will issue the following error:

```
Error, variable x already defined...
```