Final Handout
Compiler

Interpretation and Compilation
3-DEC-2019

Luis Caires

Goal

Implement a compiler for the basic imperative-functional language specified

Use the approach developed in the lectures

- Define a compile method in interface ASTNode to transverse the AST and generate code
- Use type information (from the typechecker) as needed to generate proper code
- code generation for the JVM (assemble with Jasmin)

Fully understanding the handout statement is part of the handout as well. Contact me if you need help.

Concrete Syntax (Typed Language)

Ty -> **int**	ASTIntType()
| **bool**	ASTBoolType()
| **ref** Ty	ASTRefType(Ty)
| (Ty,...,Ty)Ty	ASTFunType(List<Ty>,Ty)

Concrete Syntax (Typed Language)

EM -> E(<;>EM)*	ASTSeq(E1,E2)
E -> EA(< == > EA)?	ASTEq(EA,EA)
EA -> T(<+>EA)*	ASTAdd(E1,E2)
T -> F ((<*>T)*	ASTMul(F,T)
| (<(>AL<)>)*	ASTApply(F,AL)
| <:=> E)	ASTAssign(F,E)

AL -> (EM(<,>EM)*)?

PL -> (id:Type(<,>id:Type)*)?

F -> **num** | **id** | **bool** | **let** (**id** : **Type** = EM)+ **in** EM **end**

| **fun** PL -> EM **end** | <(> EM <)>	
| **new** F | <!> F	
| **if EM then EM else EM end**	ASTIf(EM,EM,EM)
| **while EM do EM end**	ASTWhile(EM,EM)

Examples

```
(new 3) := 6;;

let a : ref int = new 5 in a := !a + 1; !a end;;

let x : ref int = new 10
    s :ref  int = new 0  in
while !x>0 do
    s := !s + !x ; x := !x – 1
end; !s
end;;
```

Examples

```
let  f :( int,int)int = fun n:int, b:int->
            let
              x : ref int = new n
              s : ref int = new b
            in
               while !x>0 do
                  s := !s + !x ; x := !x – 1
               end;
               !s
            end
          end
in f(10,0)+f(100,20)
end;;
```

Levels of Acomplishment

Implement a compiler for the basic imperative-functional language specified

0 – Interpreter for the full language

1 – Compiler for the basic imperative language

2 – Compiler for the language with functions

3 – Compiler for the full language with structures

The 3 languages are described in the next slides

Level 1

EM -> E(<;>EM)*	ASTSeq(E1,E2)
E -> EA(< == > EA)?	ASTEq(EA,EA)
EA -> T(<+>EA)*	ASTAdd(E1,E2)
T -> F ((<*>T)*	ASTMul(F,T)
| <:=> E)	ASTAssign(F,E)

AL -> (EM(<,>EM)*)?

PL -> (id:Type(<,>id:Type)*)?

F -> **num** | **id** | **bool** | **let** (**id** : **Type** = EM)+ **in** EM **end**

| **new** F | <!> F	
| <(> EM <)>	
| **if EM then EM else EM end**	ASTIf(EM,EM,EM)
| **while EM do EM end**	ASTWhile(EM,EM)
| **println** E	ASTPrint(E)

Level 2

EM -> E(<;>EM)*	ASTSeq(E1,E2)
E -> EA(< == > EA)?	ASTEq(EA,EA)
EA -> T(<+>EA)*	ASTAdd(E1,E2)
T -> F ((<*>T)*	ASTMul(F,T)
| (<(>AL<)>)*	ASTApply(F,AL)
| <:=> E)	ASTAssign(F,E)

AL -> (EM(<,>EM)*)?

PL -> (id:Type(<,>id:Type)*)?

F -> **num** | **id** | **bool** | **let** (**id : Type** = EM)+ **in** EM **end**

| **new** F | <!> F	
| **fun** PL -> EM **end** | <(> EM <)>	
| if EM then EM else EM end	ASTIf(EM,EM,EM)
| while EM do EM end	ASTWhile(EM,EM)
| **println** E	ASTPrint(E)

Level 3

Level 3 language introduces a data type of structures and a data type of strings

The syntax for structure expressions is

```
{ id = E; id = E; id = E }   // structure construction
E.id                         // structure field selection
E1+E2                        // structure concatenation
```

Level 3 - Example

Example

```
let
   person1 = { name = “joe”; age = 22 }
   person2 = { name = “mary”; age = 5}
  person3 = person1 + { tag = -2}
in
  println  person1.age + person2.age;
  println  person3.tag + person3.age;
end
```

NOTE: this program prints out
27
20

Levels of Acomplishment

0 – Interpreter for the full language
worth 15/20 points in final handout grading

1 – Compiler for the basic imperative language
worth 16/20 points in final handout grading

2 – Compiler for the language with functions
worth 19/20 points in final handout grading

3 – Compiler for the full language with structures
worth 20/20 points in final handout grading

Due date for final handout:
21 December 2018