Compiler Construction 2012/2013: Exercises

Manuel Geffken

October 25, 2012

Outline

Organization

Organization

Who, where, when?

- Manuel Geffken, geffken@informatik.uni-freiburg.de
- Office hours: Thu 14-15 in Building 079, Room 014
- Lab session/Exercises: Thu 11-12

Exercises

The Project

A compiler from MiniJava to MIPS.

- Each sheet focuses on one part: Parser, Typechecker, ...
- Approximately six sheets
- Points per sheet and due dates: varies by difficulty
- Sheet 1: warmup exercises, due 2012-11-1, 5% of total points

Exam and final grade

- Oral examination
- The exercises are no admission criterion for the final exam
- Best out of
  - Grade from final exam
  - Grade based on average of project (50 % corresponding to 4.0) and final exam.

Motivation

```c
#include <stdio.h>

int main(void)
{
    puts("Welcome to Compiler Construction 2012/2013!");
}
```

Does this compile?

```
1 #include <stdio.h>
2 int main(){puts("Welcome to Compiler Construction 2012/2013!");}
```

Does this compile?

```
1 #include <stdio.h>
2 int main(){puts("Welcome to Compiler Construction 2012/2013!");}
```

Compiles.

And this?

```
1 #include <stdio.h>
2 intmain(){puts("Welcome to Compiler Construction 2012/2013!");}
```

Does this compile?

```
1 #include <stdio.h>
2 int main(){puts("Welcome to Compiler Construction 2012/2013!");}
```

Compiles.

And this?

```
1 #include <stdio.h>
2 intmain(){puts("Welcome to Compiler Construction 2012/2013!");}
```

Does not compile.

- What about the next one?

```c
#include "stdio.h"
#define e 3
#define g (e/e)
#define h ((g+e)/2)
#define f (e-g-h)
#define j (e*e-g)
#define k (j-h)
#define l(x) tab2[x]/h
#define m(n,a) ((n&(a))==(a))
long tab1[]={ 989L,5L,26L,0L,88319L,123L,0L,9367L };
int tab2[]={ 4,6,10,14,22,26,34,38,46,58,62,74,82,86 };
main(m1,s) char *s;{
int a,b,c,d,o[k],n=(int)s;
if(m1==1){ char b[2*j+f-g]; main(l(h+e)+h+e,b); printf(b); }
else switch(m1-=h){
case f:a=(b=(c=(d=g)<<g)<<g)<<g;
return(m(n,a|c)|m(n,b)|m(n,a|d)|m(n,c|d));
case h:
for(a=f;a<j;++a)if(tab1[a]&&!(tab1[a]%((long)l(n))))return(a);
case g:if(n<h)return(g);
if(n<j){n-=g;c='D';o[f]=h;o[g]=f;}
else{c='\r'-'\b';n-=j-g;o[f]=o[g]=g;}
if((b=n)>=e)for(b=g<<g;b<n;++b)o[b]=o[b-h]+o[b-g]+c;
return(o[b-g]%n+k-h);
default:if(m1-=e) main(m1-g+e+h,s+g); else *(s+g)=f;
for(*s=a=f;a<e;) *s=(*s<<e)|main(h+a++,(char *)m1);}}
```


JavaScript
Perl
PHP
Markup
Scala
Functional
C++
SQL
NET
XML
Pascal
VisualBasic
DataStructured
Erlang
Declarative
Ruby
Ada
Scripting
Squeak
Delphi
Multiparadigm
Logic-based
LabVIEW
Occam
Python
Cobol
Prototype-based
Yorick
Assembly
Java
Scheme
Clojure
Lisp
Haskell
Distributed
Imperative
Interpreted
MATLAB
Object-Oriented
AppleScript
BASIC
HyperCard
Procedural
PostScript
Fortran
Obfuscated
Visual
C-sharp
Smalltalk
Prolog
Reflective
HTML

Tools

Tools

Tools you need to know/learn

- Java $\geq$ 1.6
- Eclipse $\geq$ 4.2 (other IDEs: you're on your own)
- SableCC 3.6
- LaTeX (or anything else for high-quality type-system typesetting)

Tools you will use without knowing

- Ant
- Checkstyle
- See tools page for installation instructions.

Visitor Pattern

Motivation
(Palsberg and Jay, The Essence of the Visitor Pattern, 1998)

Summing the elements of a list

```
1 interface List {}
2
3 class Nil implements List {}
4 class Cons implements List {
5   int head;
6   List tail;
7 }
```

1. Approach: InstanceOf and Type Casts

```
1  List l;
2  int sum = 0;
3  boolean proceed = true;
4  while(proceed){
5    if (l instanceof Nil)
6      proceed = false;
7    else if (l instanceof Cons){
8      sum += ((Cons) l).head;
9      l = ((Cons) l).tail;
10   }
11 }
```

- Classes are not touched.
- ... but frequent type casts and instanceof! :(

2. Approach: Dedicated Methods

```
1  interface List {
2    public int sum();
3  }
4  class Nil implements List {
5    public int sum() { return 0; }
6  }
7  class Cons implements List {
8    int head;
9    List tail;
10   public int sum() { return head + tail.sum(); }
11 }
```

- No type casts, systematic and object-oriented.
- ... but frequent re-compilation and changing of classes! :(

3. Approach: Visitor Pattern (Gamma et al., Design Patterns, 1995)

Intent

Represent an operation to be performed on the elements of an object structure. The Visitor pattern lets you define a new operation *without changing the classes* of the elements on which it operates.

Idea

- Distinguish between object structure and the visitor.
- Insert an `accept` method in each class of the object structure.
- For each of these classes, a visitor contains a `visitXXX` method.

Visitor Pattern

```
1  interface List {
2    void accept(Visitor v);
3  }
4
5  class Nil implements List {
6    public void accept(Visitor v) {
7      v.visitNil(this);
8    }
9  }
10 class Cons implements List {
11   int head;
12   List tail;
13   public void accept(Visitor v) {
14     v.visitCons(this);
15   }
16 }
```

Visitor Pattern

```
1  interface Visitor {
2    void visitNil(Nil x);
3    void visitCons(Cons x);
4  }
5  class SumVisitor implements Visitor {
6    int sum;
7    public void visitNil(Nil x) {}
8    public void visitCons(Cons x) {
9      sum += x.head;
10     x.tail.accept(this);
11   }
12 }
13 ...
14 SumVisitor sv = new SumVisitor();
15 l.accept(sv);
16 System.out.println(sv.sum);
```

Visitor Pattern - Summary

The visitor pattern gives you..

- New methods/functionality without recompiling the object structure!
- Related operations are structured together.
- Visitors can accumulate (and also encapsulate) state.

But...

- All classes must have an accept method.
- Adding new classes to the object structure is nasty.

Careful!

The visit methods describe actions **and** access to subobjects.

SableCC

SableCC

What is SableCC?

- open-source parser generator for Java
- `http://sablecc.org`
- generates LALR(1) parsers
- featuring: lexer, parser, nodes/ast, analysis/visitors

A specification for SableCC

Parts

- Package *package-name*;
- Helpers *id* = *regexp*;
- Tokens *id* = *regexp*;
- Ignored Tokens *token1*,...,*tokenN*;
- Productions (simplified)
  id = *{altname} elem** | ... ;
  with *elem* = *[id]: id* (+|*|?)

A specification for SableCC

Example

```
1  Package simpleAdder;
2
3  Tokens
4    l_par  = '(';
5    r_par  = ')';
6    plus   = '+';
7    number = ['0'..'9'];
8
9  Productions
10   exp = {constant} number
11       | {add} addition;
12   addition = l_par [left]:exp plus [right]:exp r_par;
```

A specification for SableCC

Generated files

```
1  /*     exp = {constant} number | {add} addition;
2    addition = l_par [left]:exp plus [right]:exp r_par; */
3  abstract class Node {}
4  /** Superclass of all exp-> right-hand sides  */
5  abstract class PExp extends Node{}
6  /** One exp->number right-hand side */
7  class AConstantExp extends PExp {
8    TNumber getNumber(){...}  ...
9  }
10 /** One exp->{add}addition right-hand side  */
11 class AAddExp extends PExp {
12   PAddition getAddition(){...}  ...
13 }
14 /** one addition->l_par... subtree */
15 class AAddition extends PAddition {
16   TLPar getLPar() {...} // corresponds to l_par
17   PExp getLeft() {...} // corresponds to [left]:exp
18   ...
19 }
```

Visitor Pattern in SableCC

Generated files

```
1 class DepthFirstAdapter extends AnalysisAdapter {
2   void caseXxx(Xxx node) {
3     inXxx(node);
4     node.getYyy.apply(this); // first child of Xxx
5     node.getZzz.apply(this); // second child of Xxx
6     outXxx(node);
7   }
8   ...
9 }
```

Important

Do **not**...

- modify any generated files!
- submit any homework late!
- copy anyone's homework!
- panic! Ask for help!

Do ...

- comment your submissions!
- start early on the assignments!
- consult manuals, tutorials, our forum and the homepage!
- have fun!