Compilation

Dr. J.P.E. Hodgson

February 3, 2010
Outline

1. Compilation
2. Phases
3. Syntactic Analysis
4. Contextual Analysis
5. Code Generation
6. Passes
7. Design
Compilation

Dr. J.P.E. Hodgson

Compilation
Phases
Syntactic Analysis
Contextual Analysis
Code Generation
Passes
Design

Compilation

- Phases
- Passes
- The Triangle Compiler
<table>
<thead>
<tr>
<th></th>
<th>Outline</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Compilation</td>
</tr>
<tr>
<td>2</td>
<td>Phases</td>
</tr>
<tr>
<td>3</td>
<td>Syntactic Analysis</td>
</tr>
<tr>
<td>4</td>
<td>Contextual Analysis</td>
</tr>
<tr>
<td>5</td>
<td>Code Generation</td>
</tr>
<tr>
<td>6</td>
<td>Passes</td>
</tr>
<tr>
<td>7</td>
<td>Design</td>
</tr>
</tbody>
</table>
## Phases

1. **Syntactic analysis**
   - 1. **Lexing**
   - 2. **Parsing**

2. **Contextual Analysis**

3. **Code generation**

4. **Code optimization**
Compilation

Phases

Syntactic Analysis

Contextual Analysis

Code Generation

Design

The Data flow
Outline

1. Compilation
2. Phases
3. Syntactic Analysis
4. Contextual Analysis
5. Code Generation
6. Passes
7. Design
Syntactic analysis

```plaintext
! This program is useless
! except for illustration.

let

   var n: Integer;
   var c: Char

in

   begin
   c := '&';
   n := n + 1
   end

Figure 3.2 A Triangle source program.
```
Result: Abstract Syntax tree

Figure 3.3 AST after syntactic analysis of the source program of Figure 3.2.
Outline

1. Compilation
2. Phases
3. Syntactic Analysis
4. Contextual Analysis
5. Code Generation
6. Passes
7. Design
Contextual Analysis

Result: Decorated AST

- Each *applied occurrence* is linked to its corresponding *declaration*;
- Each expression is decorated by its type.
Compilation

Dr. J.P.E. Hodgson

Compilation Phases

Syntactic Analysis

Contextual Analysis

Code Generation

Passes

Design

Figure 3.4 Decorated AST after contextual analysis of the AST of Figure 3.3.
let
  var n: Integer
in ! ill-formed program
  while n/2 do
    m := 'n' > 1

Figure 3.5 An ill-formed Triangle source program.

Figure 3.6 Discovering errors during contextual analysis of the Triangle program of Figure 3.5.
Outline

1. Compilation
2. Phases
3. Syntactic Analysis
4. Contextual Analysis
5. Code Generation
6. Passes
7. Design
Code Generation

- A constant declaration binds an identifier to a value
- A variable declaration binds an identifier to a storage location and a type.

Figure 3.7 Object program after code generation from Figure 3.6
Outline

1. Compilation
2. Phases
3. Syntactic Analysis
4. Contextual Analysis
5. Code Generation
6. Passes
7. Design
Passes

- Multipass compilers
- One pass compilers
Outline

1. Compilation
2. Phases
3. Syntactic Analysis
4. Contextual Analysis
5. Code Generation
6. Passes
7. Design
Compiler Design Issues

- Speed
- Space
- Modularity
- Flexibility
- Semantics preserving transformations – note that these are much easier to do for declarative languages
- Restrictions imposed by the source language.
The structure of the Triangle Compiler

```java
public class Compiler{
    public static void compileProgram(...){
        Parser parser = new Parser(...);
        Checker checker = new Checker(...);
        Encoder generator = new Encoder(...);
        Program theAST = parser.parse();
        checker.check(theAST);
        generator.encode(theAST);
    }
    public static void main(String[] args){
        ... compileProgram(...);
    }
}
```