Course Overview

PART I: overview material
1 Introduction
2 Language processors (tombstone diagrams, bootstrapping)
3 Architecture of a compiler

PART II: inside a compiler
4 Syntax analysis
5 Contextual analysis
6 Runtime organization
7 Code generation

PART III: conclusion
8 Interpretation
9 Review

Chapter 3 Compilation

So far we have treated language processors (including compilers) as “black boxes”

GOAL this lecture:
– A first look "inside the box": how to build compilers.
– Different “phases” and their relationships.

The Major “Phases” of a Compiler

Source Program
→ Syntax Analysis
→ Abstract Syntax Tree
→ Contextual Analysis
→ Decorated Abstract Syntax Tree
→ Code Generation
→ Object Code

Different Phases of a Compiler

The different phases can be seen as different transformation steps to transform source code into object code.

The different phases correspond roughly to the different parts of the language specification:
• Syntax analysis <-> Syntax
• Contextual analysis <-> Contextual constraints
• Code generation <-> Semantics

Example Program

We now look at each of the three different phases in a little more detail. We look at each of the steps in transforming an example Triangle program into TAM code.

```
! This program is useless except for ! illustration
let var n: integer;
var c: char
in begin
  c := '4';
  n := n+1
end
```

1) Syntax Analysis

Source Program
→ Syntax Analysis
→ Abstract Syntax Tree

Note: Not all compilers construct an explicit representation of an AST. (e.g. on a “single pass compiler” generally no need to construct an AST)
1) Syntax Analysis --> AST

2) Contextual Analysis --> Decorated AST

3) Code Generation

3) Code Generation

• Assumes that program has been thoroughly checked and is well formed (scope & type rules)
• Takes into account semantics of the source language as well as the target language.
• Transforms source program into target code.
Compiler Passes

- A “pass” is a complete traversal of the source program, or a complete traversal of some internal representation of the source program (such as an AST).
- A pass can correspond to a “phase” but it does not have to!
- Sometimes a single pass corresponds to several phases that are interleaved in time.
- What and how many passes a compiler does over the source program is an important design decision.

Single Pass Compiler

A single pass compiler makes a single pass over the source text, parsing, analyzing, and generating code all at once.

Dependency diagram of a typical Single Pass Compiler:

Multi Pass Compiler

A multi pass compiler makes several passes over the program. The output of a preceding phase is stored in a data structure and used by subsequent phases.

Dependency diagram of a typical Multi Pass Compiler:

Compiler Design Issues

<table>
<thead>
<tr>
<th></th>
<th>Single-Pass</th>
<th>Multi-Pass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>better</td>
<td>worse</td>
</tr>
<tr>
<td>Memory</td>
<td>better for large programs</td>
<td>(potentially) better for small programs</td>
</tr>
<tr>
<td>Modularity</td>
<td>worse</td>
<td>better</td>
</tr>
<tr>
<td>Flexibility</td>
<td>worse</td>
<td>better</td>
</tr>
<tr>
<td>“Global” optimization</td>
<td>impossible</td>
<td>possible</td>
</tr>
<tr>
<td>Source Language</td>
<td>single pass compilers are not possible for many programming languages</td>
<td></td>
</tr>
</tbody>
</table>

Language Issues

Example Pascal:
Pascal was explicitly designed to be easy to implement with a single pass compiler:

- Every identifier must be declared before its first use.

```
var n: integer;  procedure inc;
begin
  n := n+1
end
```

```
var n: integer;  procedure inc;
begin
  end
```

Undeclared Variable!
Language Issues

Example Pascal:
- Every identifier must be declared before it is used.
- How to handle mutual recursion then?

\[
\begin{align*}
\text{procedure ping(x:integer)} \\
&\text{begin} \\
&\quad \ldots \text{pong}(x-1); \ldots \\
&\text{end;}
\end{align*}
\]

\[
\begin{align*}
\text{procedure pong(x:integer)} \\
&\text{begin} \\
&\quad \ldots \text{ping}(x-1); \ldots \\
&\text{end;}
\end{align*}
\]

Language Issues

Example Pascal:
- Every identifier must be declared before it is used.
- How to handle mutual recursion then?

\[
\begin{align*}
\text{forward procedure pong(x:integer)} \\
\text{procedure ping(x:integer)} \\
&\text{begin} \\
&\quad \ldots \text{pong}(x-1); \ldots \\
&\text{end;}
\end{align*}
\]

OK!

Example: The Triangle Compiler Driver

```java
public class Compiler {
    public static void compileProgram(...) { 
        Parser parser = new Parser(...);
        Checker checker = new Checker(...);
        Encoder generator = new Encoder(...);

        Program theAST = parser.parse(); // first pass
        checker.check(theAST); // second pass
        generator.encode(theAST); // third pass
    }

    public static void main(String[] args) {
        ... compileProgram(...); ...
    }
}
```

Compilation (Chapter 3)