Programming Languages

Tevfik Koşar

Lecture - II
January 19th, 2006

Roadmap

- Compilation
- Interpretation
- Preprocessing
- Linking
- Assembling
- Phases of Compilation
  - Scanning
  - Parsing
  - Semantic Analysis
Compiler

- Translates high-level program source code (in text) into a target code (generally binary executable)
  
  Source program (text) → Compiler → Target program (binary)

- Generated target program is standalone
  - After compilation the compiler goes away

- Generated target program can be platform-dependant

Interpreter

- Reads and executes the source code line by line
- Stays around during execution
- Does not generate standalone executables

Source program → Interpreter → Output
Compilation vs Interpretation

- **Compilation**
  - Better performance
  - Most decisions are done at compile time (e.g., memory references)
  - Compile once, execute many times

- **Interpretation**
  - More flexible
  - Enables better diagnostics (error messages)
  - After compilation some information is lost
  - Can have source-level debugger

Hybrid Systems

- Example: Java
  - Intermediate binaries are called: “byte codes”
Preprocessors

• Preprocessor: initial translator
  - Removes comments & white space
  - Groups characters into tokens (keywords, identifiers, numbers)
  - Expands macros and abbreviations
  - Produced source can be compiled/interpreted more efficiently
    • In early versions of Basic, you had to remove comments to improve performance (reread everytime a certain part was executed)

Compilation, Interpretation & Preprocessing

• Compilation generally produces a binary; but does NOT have to produce machine language for some sort of hardware
• Compilation is translation from one language into another, with full analysis of the meaning of the input
• Compilation & Interpretation entail semantic understanding of what is being processed; pre-processing does not
• A pre-processor will often let errors through. Compilers and Interpreters will not.
Examples

- Interpreted Languages:
  - Java
  - Scheme
  - Prolog
  - Python
  - Most Scripting Languages
- Compiled Languages
  - C / C++
  - Pascal
  - Fortran
  - Ada

Linking

- Compiler uses a linker program to merge the appropriate library of subroutines (e.g., math functions such as sin, cos, log, etc.) into the final program:
  - eg. Fortran Compiler
Assembling

- Many compilers generate assembly language instead of a machine language

  Source program → Compiler → Assembly language
  Assembly language → Assembler → Machine language

- Facilitates debugging
  - Assembly is easier to read
- Isolates compiler from changes in the format of machine language files
  - only assembler need to be changed, and it is shared by many compilers

C Compiler

- C preprocessor
  - Removes comments & extends macros
  - It can also delete portions of code, which allows several versions of a program to be built from the same source
    - eg. Adding & removing debugging information
Early C++ Compiler

- Early C++ compilers were generating C code
- Complete error check was performed
- If no errors, C compiler was invoked by the C++ compiler
  - Programmers were unaware of this fact

Roadmap

- Compilation
- Interpretation
- Preprocessing
- Linking
- Assembling
- Phases of Compilation
  - Scanning
  - Parsing
  - Semantic Analysis
Phases of Compilation

Example

- Source Code for GCD (in Pascal):

```pascal
program gcd(input, output);
var i, j : integer;
begin
  read(i, j);
  while i <> j do
    if i > j then i := i - j
    else j := j - i;
  writeln(i)
end.
```
Example

- **After Scanning (Lexical Analysis):**
  - Characters are grouped into tokens (smallest meaningful units of the program)
  - Eg. identifiers, variables, punctuation, operators...

```plaintext
program gcd ( input , output ) ;
var i , j : integer ;
read ( i , j ) ;
while i <> j do if i > j then i := i - j ;
else j := j - i ;
writeln ( i )
end .
```

- **Purpose of Scanning (Lexical Analysis):**
  - Simplify the task for parser by reducing the input size

---

**Phases of Compilation**

1. **Character stream**
2. **Token stream**
3. **Parse tree**
4. **Abstract syntax tree or other intermediate form**
5. **Modified intermediate form**
6. **Assembly or machine language, or other target language**
7. **Modified target language**
8. **Symbol table**
9. **Scanning (lexical analysis)**
10. **Parser (syntax analysis)**
11. **Semantic analysis and intermediate code generation**
12. **Machine-independent code improvement (optional)**
13. **Target code generation**
14. **Machine-specific code improvement (optional)**
**Parsing (Syntax Analysis)**

- **Goal**: To check if the source code fits the Grammar of that Particular Language.
  - Eg: for comparison:
    - In C: if (a != b) ....
    - In Pascal: if (a <> b) then ...

- **Scanner** can be considered **language-blind**
- **Parser** is **language-specific**

**Context-free Grammar**

- **Example** (Pascal):

```
program -> PROGRAM id ( id more_ids ) ; block .

where

block -> labels constants types variables subroutines BEGIN stmt
       more_stmts END

and

more_ids -> , id more_ids
or
more_ids -> ε
```
Parsing Example

```
program program_id ( id more_id ) ; block ;
where
    block ::= labels constants types variables subroutines BEGIN stmt
            more_stmts END
end
more_id ::= , id more_id
or
more_id ::= e
```

```
program gcd ( input , output ) ;
var i , j : integer ; begin
    read ( i , j ) ;
    while i <> j do
        if i > j then
            i := i - j
        else
            j := j - i
        end ;
write ( i )
```

Parse Tree
Parse Tree (cont.)

Phases of Compilation

Character stream
Token stream
Parse tree
Abstract syntax tree or other intermediate form
Modified intermediate form
Assembly or machine language, or other target language
Modified target language

Scanner (lexical analysis)
Parser (syntax analysis)
Semantic analysis and machine-independent code generation
Target code generation
Machine-specific code improvement (optional)
Symbol table
Semantic Analysis

- Discovery of the meaning of the program
- Creates a symbol table which maps each identifier to the information known about it
  - eg. type, scope (the portion of the program it is valid)
- Semantic Analyzer checks to make sure that:
  - Every identifier is declared before it is used
  - No identifier is used in an inappropriate context
    - Assigning incompatible types to each other.
  - Subroutine calls have the correct number and types of arguments

Example

- **Source Code**
  ```plaintext
  program gcd(input, output);
  var i, j : integer;
  begin
    read(i, j);
    while i <> j do
      if i > j then i := i - j
      else j := j - i;
    writeln(i)
  end.
  ```

- **Symbol Table**
<table>
<thead>
<tr>
<th>Index</th>
<th>Symbol</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>INTEGER</td>
<td>type</td>
</tr>
<tr>
<td>2</td>
<td>TEXTFILE</td>
<td>type</td>
</tr>
<tr>
<td>3</td>
<td>INPUT</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>OUTPUT</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>GCD</td>
<td>program</td>
</tr>
<tr>
<td>6</td>
<td>I</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>J</td>
<td>1</td>
</tr>
</tbody>
</table>
Syntax Tree

```
program
    read
      (5)
    read
      (3) (6)
    while
      (3) (7)
    write
      (4) (6)
    writeln
      (4)

Index  Symbol     Type
1      INTEGER    type
2      TEXTFILE   type
3      INPUT      2
4      OUTPUT     2
5      GCD        program
6      I          1
7      J          1
```

Syntax tree and symbol table for the GCD program.

---

Phases of Compilation

Character stream → Token stream → Parse tree → Abstract syntax tree or other intermediate form → Modified intermediate form → Assembly or machine language, or other target language → Modified target language → Symbol table

Scanner (lexical analysis) → Parser (syntax analysis) → Semantic analysis and intermediate code generation → Machine-independent code improvement (optional) → Target code generation → Machine-specific code improvement (optional)

4
Any Questions?

Hmm...

Announcements

- Reading Assignment: Sections 2.1 & 2.2
- HW 1 will be out next Tuesday and will be due 1 week
- Please send your course schedules to me
- Make sure you are in the mailing list