The University of North Carolina at Chapel Hill

COMP 144 Programming Language Concepts
Spring 2002

Lecture 2: Compilation and Interpretation

Felix Hernandez-Campos
Jan 11

From Source Code to Executable Code

```
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.
```

Compilation
Compilation and Interpretation

- A compiler is a program that translates high-level source programs into target program.

```
Source program → Compiler → Target program
Input               Output
```

- An interpreter is a program that executes another program.

```
Source program → Interpreter → Output
Input
```

Mixing Compilation and Interpretation

- Fuzzy difference:
  - A language is interpreted when the initial translation is simple.
  - A language is compiled when the translation process is complicated.

```
Source program → Translator → Intermediate program
Intermediate program → Virtual machine → Output
Input
```
Preprocessing

- Macros
  - `#define <macro> <replacement name>`
  - `#define FALSE 0`
  - `#define max(A,B) ( (A) > (B) ? (A):(B))`

Linking

- Libraries of subroutines
Portability

- Assembly language instead of machine language
  
  Source program \(\rightarrow\) Compiler \(\rightarrow\) Assembly language
  
  Assembly language \(\rightarrow\) Assembler \(\rightarrow\) Machine language

- Intermediate source code
  
  Source program \(\rightarrow\) Preprocessor \(\rightarrow\) Modified source program
  
  Modified source program \(\rightarrow\) C++ compiler \(\rightarrow\) C code
  
  C code \(\rightarrow\) C compiler \(\rightarrow\) Assembly language

Programming Environments

- Much more than compilers and interpreters
  - Assemblers, debuggers, preprocessors and linkers
  - Editors
  - Pretty printers
  - Style Checkers
  - Version management
  - Profilers

- Integrated environments
  - Beyond a simple *bus error*
  - *Emacs*
Overview of Compilation

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.

Phases of Compilation

Character stream → Scanner (lexical analysis)
Token stream → Parser (syntax analysis)
Parse tree → Semantic analysis and intermediate code generation
Abstract syntax tree or other intermediate form → Machine-independent code improvement (optional)
Modified intermediate form → Target code generation
Assembly or machine language, or other target language → Machine-specific code improvement (optional)
Modified target language → Symbol table
Example

- From Scott’s class notes
- Desk calculator language
- Example program:
  ```plaintext
  read A
  read B
  sum := A + B
  write sum
  write sum / 2
  ```

Lexical Analysis

- Tokens:
  ```plaintext
  id = letter ( letter | digit ) * [ except "read" and "write" ]
  literal = digit digit *
  ":=", "+", "-", "*", "/", "("", ")"
  $$$ [end of file]
  ```
Syntax Analysis

• Grammar in EBNF

```ebnf
<pgm>        ->  <statement list> $$$
<stmt list>  ->  <stmt list> <stmt> | E
<stmt>       ->  id := <expr> | read <id> | write <expr>
<expr>       ->  <term> | <expr> <add op> <term>
<term>       ->  <factor | <term> <mult op> <factor
<factor>     ->  ( <expr> ) | id | literal
<add op>     ->  + | -
<mult op>    ->  * | /
```

Code Generation

• Intermediate code:

```
read
pop A
read
pop B
push A
push B
add
pop sum
push sum
write
push sum
write
push 2
div
write
```
**Code Generation**

- Target code:

```
.data
A: .long 0
B: .long 0
sum: .long 0
.text
main: jsr read
      movl d0,d1
      movl d1,A
      jsr read
      movl d0,d1
      movl d1,B
      movl A,d1

movl B,d2
addl d1,d2
movl d1,sum
movl sum,d1
movl d1,d0
jsr write
movl sum,d1
movl $2,d2
divsl d1,d2
movl d1,d0
jsr write
```