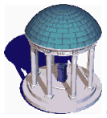


The University of North Carolina at Chapel Hill

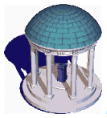
COMP 144 Programming Language Concepts

Spring 2002

Lecture 2: Compilation and Interpretation

Felix Hernandez-Campos

Jan 11

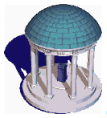


From Source Code to Executable Code

```
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.
```

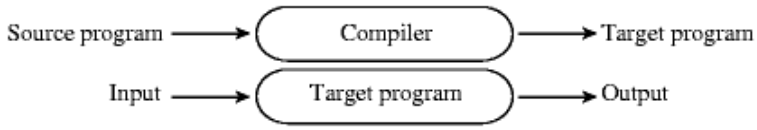
Compilation

```
27bdffd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c
00401825 10820008 0064082a 10200003 00000000 10000002 00832023
00641823 1483fffa 0064082a 0c1002b2 00000000 8fbf0014 27bd0020
03e00008 00001025
```

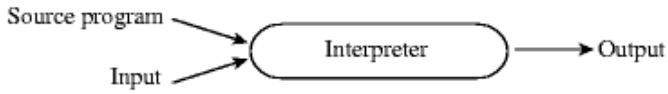


Compilation and Interpretation

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

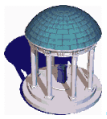


- An interpreter is a program that **executes** another program



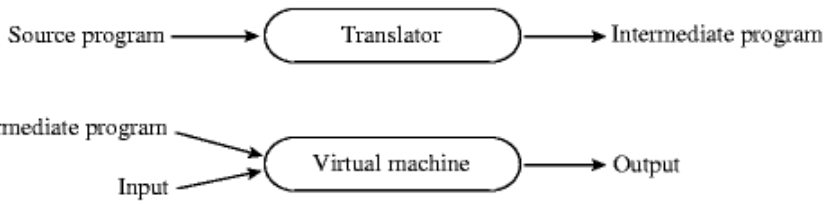
COMP 144 Programming Language Concepts
Felix Hernandez-Campos

3



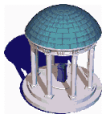
Mixing Compilation and Interpretation

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

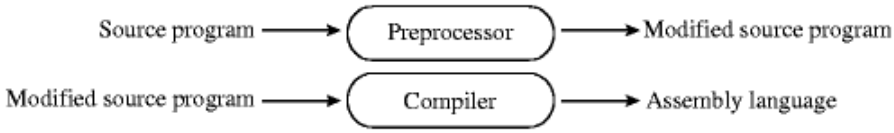


COMP 144 Programming Language Concepts
Felix Hernandez-Campos

4

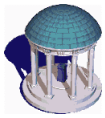


Preprocessing



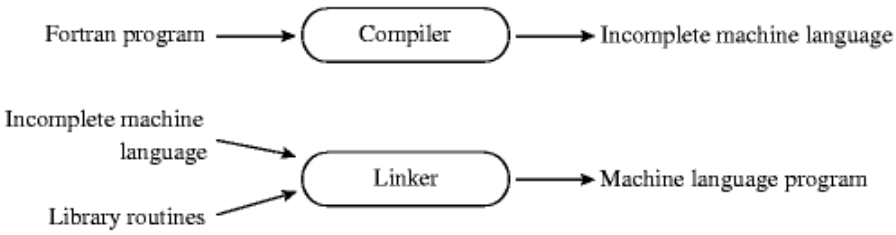
- Macros
  - #define <macro> <replacement name>
  - `#define FALSE 0`
  - `#define max(A,B) ( (A) > (B) ? (A):(B))`

COMP 144 Programming Language Concepts
Felix Hernandez-Campos
5

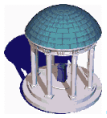


Linking

- Libraries of subroutines

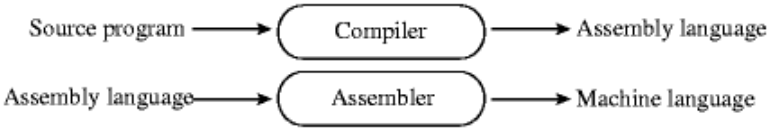


COMP 144 Programming Language Concepts
Felix Hernandez-Campos
6

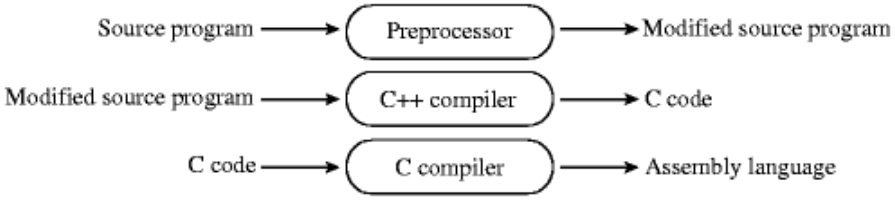


Portability

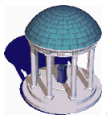
- Assembly language instead of machine language



- Intermediate source code



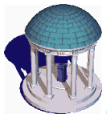
COMP 144 Programming Language Concepts
Felix Hernandez-Campos
7



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*

COMP 144 Programming Language Concepts
Felix Hernandez-Campos
8

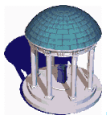


Overview of Compilation

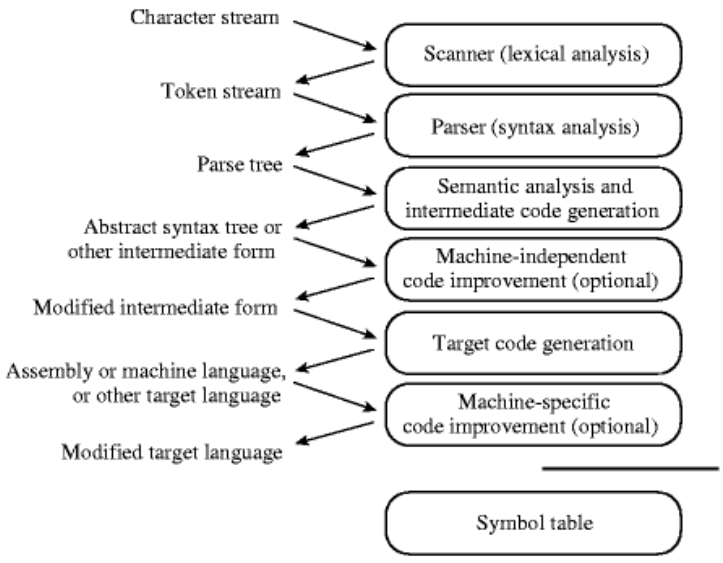
```
program gcd(input, output);
var i, j: integer;
begin
        read(i, j);
        while i <> j do
                if i > j then i := i - j;
                else j := j - i;
        writeln(i)
end.
```

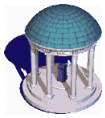
Compilation

```
27bdffd0 afbf0014 0c1002a8 00000000 0c1002a8 afa2001c 8fa4001c
00401825 10820008 0064082a 10200003 00000000 10000002 00832023
00641823 1483fffa 0064082a 0c1002b2 00000000 8fbf0014 27bd0020
03e00008 00001025
```

Phases of Compilation

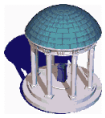




Example

- From Scott's class notes
- Desk calculator language
- Example program:

```
read A
read B
sum := A + B
write sum
write sum / 2
```

Lexical Analysis

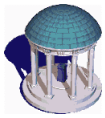
- Tokens:

id = letter (letter | digit) * [except "read" and "write"]

literal = digit digit *

":=", "+", "-", "*", "/", "(", ")"

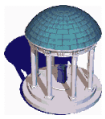
$$$ [end of file]



Syntax Analysis

- Grammar in EBNF

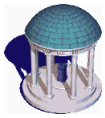
```
<pgm>         ->  <statement list> $$$
<stmt list>   ->  <stmt list> <stmt> | E
<stmt>        ->  id := <expr> | read <id> | write <expr>
<expr>        ->  <term> | <expr> <add op> <term>
<term>        ->  <factor | <term> <mult op> <factor
<factor>      ->  ( <expr> ) | id | literal
<add op>   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

Code Generation

- Intermediate code:

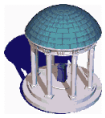
```
read
pop A
read
pop B
push A
push B
add
pop sum
push sum
write
push sum
push 2
div
write
```



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
```

Code Generation

```
        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
```