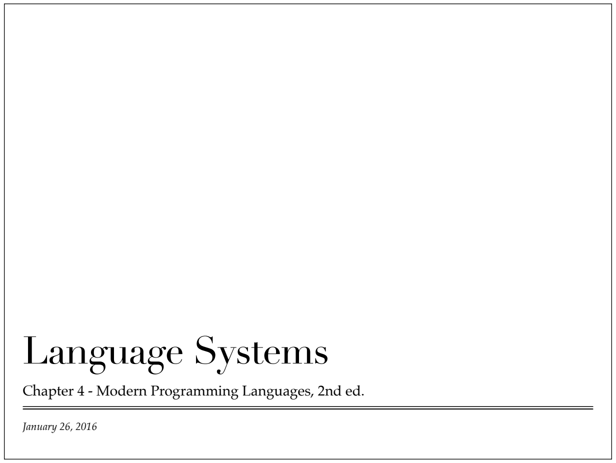
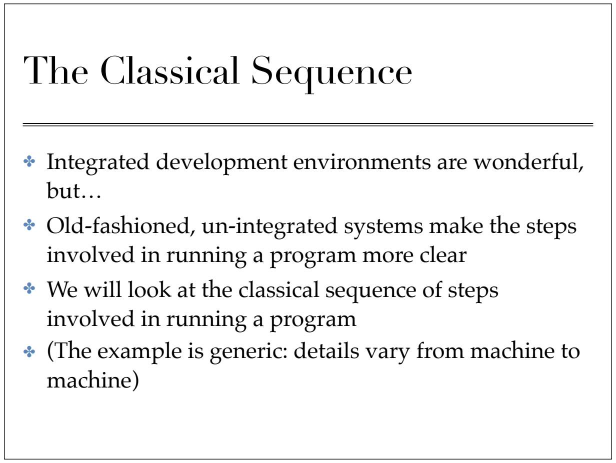


1

Language Systems

Chapter 4 - Modern Programming Languages, 2nd ed.

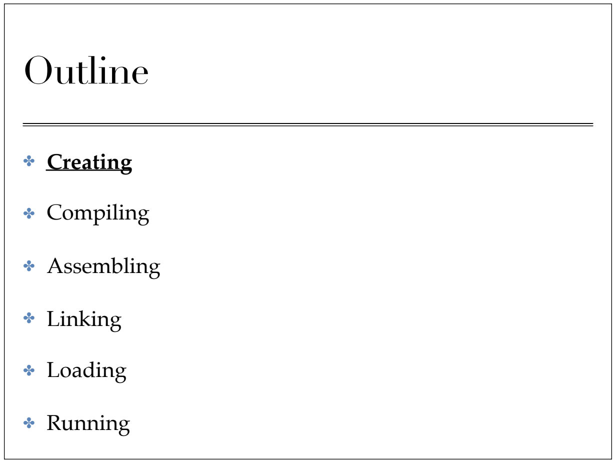
January 26, 2016



2

The Classical Sequence

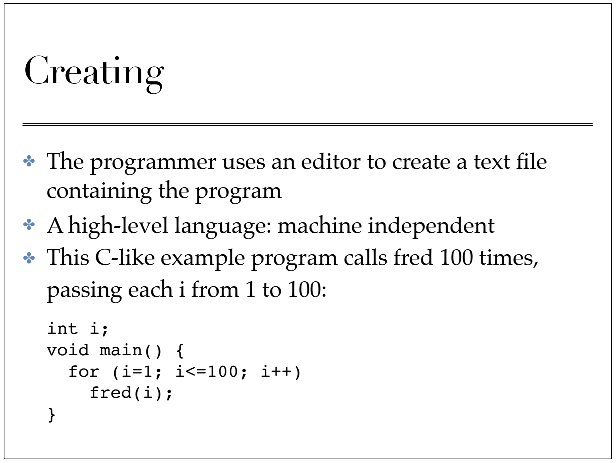
- Integrated development environments are wonderful, but…
- Old-fashioned, un-integrated systems make the steps involved in running a program more clear
- We will look at the classical sequence of steps involved in running a program
- (The example is generic: details vary from machine to machine)



3

Outline

- **Creating**
- Compiling
- Assembling
- Linking
- Loading
- Running

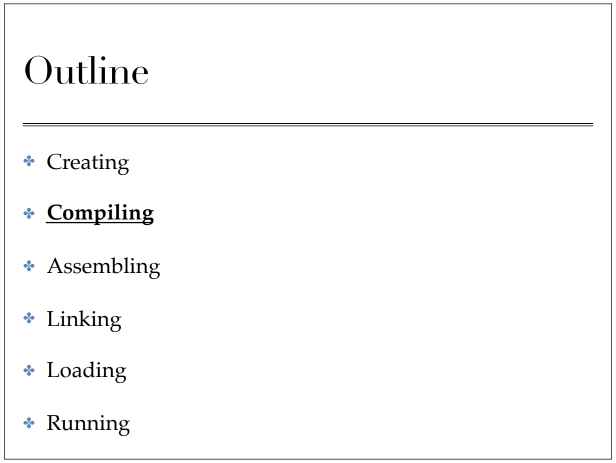


4

Creating

- The programmer uses an editor to create a text file containing the program
- A high-level language: machine independent
- This C-like example program calls fred 100 times, passing each i from 1 to 100:

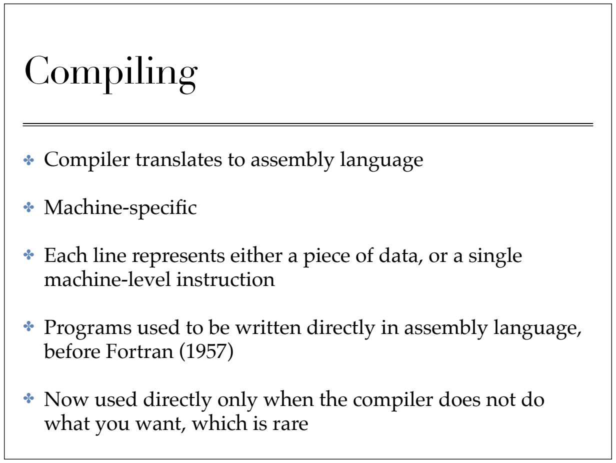
```
int i;
void main() {
  for (i=1; i<=100; i++)
    fred(i);
}
```



5

Outline

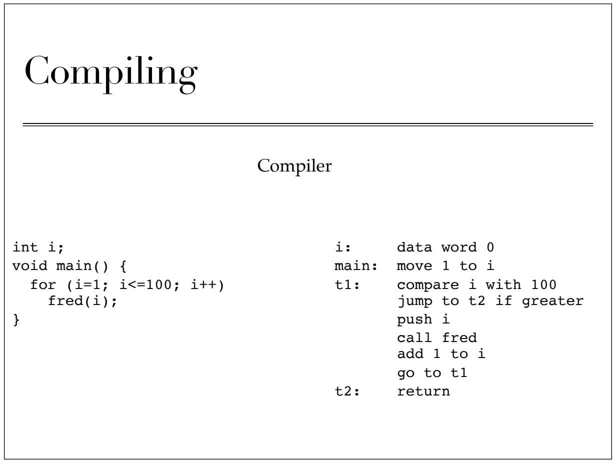
- Creating
- **Compiling**
- Assembling
- Linking
- Loading
- Running



6

Compiling

- Compiler translates to assembly language
- Machine-specific
- Each line represents either a piece of data, or a single machine-level instruction
- Programs used to be written directly in assembly language, before Fortran (1957)
- Now used directly only when the compiler does not do what you want, which is rare

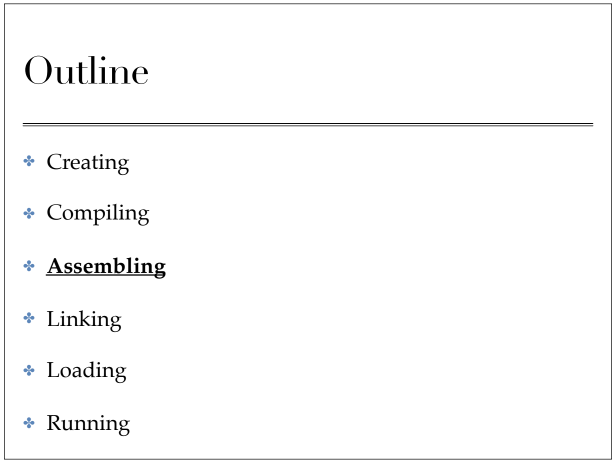


7

Compiling

Compiler

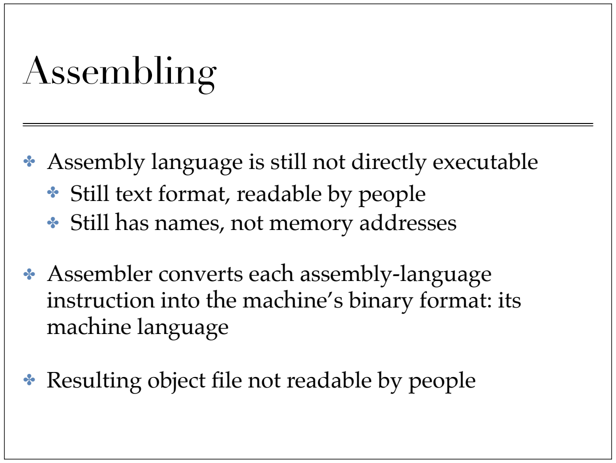
```
int i;                                  i:     data word 0
void main() {                           main:  move 1 to i
  for (i=1; i<=100; i++)                t1:    compare i with 100
    fred(i);                                   jump to t2 if greater
}                                              push i
                                               call fred
                                               add 1 to i
                                               go to t1
                                        t2:    return
```



8

Outline

- Creating
- Compiling
- **Assembling**
- Linking
- Loading
- Running



9

Assembling

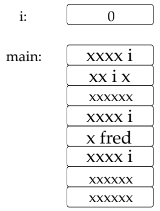
- Assembly language is still not directly executable
  - Still text format, readable by people
  - Still has names, not memory addresses
- Assembler converts each assembly-language instruction into the machine's binary format: its machine language
- Resulting object file not readable by people

Assembling

10

Assembler

```
i:      data word 0
main:   move 1 to i
t1:     compare i with 100
        jump to t2 if greater
        push i
        call fred
        add 1 to i
        go to t1
t2:     return
```



Outline

11

- Creating
- Compiling
- Assembling
- **Linking**
- Loading
- Running

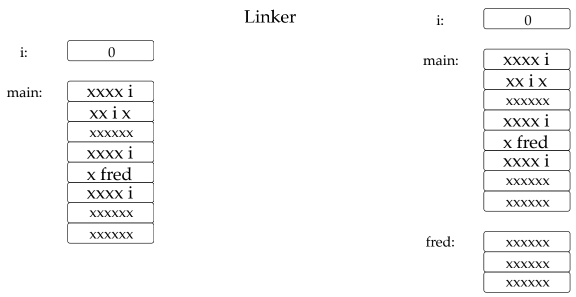
Linking

12

- Object file still not directly executable
  - Missing some parts
  - Still has some names
  - Mostly machine language, but not entirely
- Linker collects and combines all the different parts
- In our example, fred was compiled separately, and may even have been written in a different high-level language
- Result is the executable file

13

Linking



14

Outline

- Creating
- Compiling
- Assembling
- Linking
- **Loading**
- Running

15

Loading

- "Executable" file still not directly executable
  - Still has some names
  - Mostly machine language, but not entirely
- Final step: when the program is run, the loader loads it into memory and replaces names with addresses

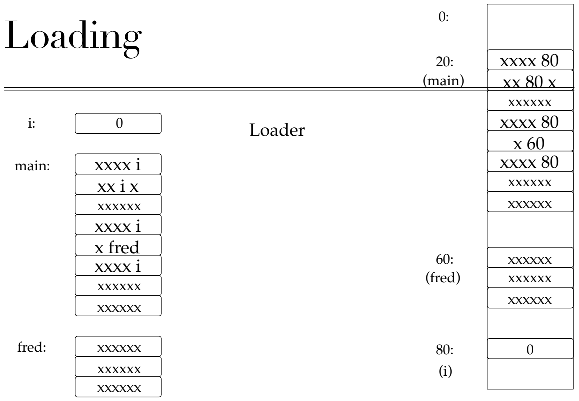
16

A Word About Memory

- For our example, we are assuming a very simple kind of memory architecture
- Memory organized as an array of bytes
- Index of each byte in this array is its address
- Before loading, language system does not know where in this array the program will be placed
- Loader finds an address for every piece and replaces names with addresses

17

Loading



18

Outline

- Creating
- Compiling
- Assembling
- Linking
- Loading
- **Running**

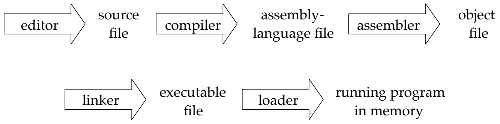
19

Running

- After loading, the program is entirely machine language
  - All names have been replaced with memory addresses
- Processor begins executing its instructions, and the program runs

20

The Classical Sequence



editor → source file → compiler → assembly-language file → assembler → object file

linker → executable file → loader → running program in memory

21

Makefiles

- Automate the compilation process
- https://gist.github.com/jrob8577/df8c93252113129d7508