Data Types and Character IO

Class 9

Happy
Programmer's Day!

Happy Programmer's Day!

13 September is the 256th day of a non-leap year

Steve Jobs

Steve Jobs died, 5 October 2011
co-founder and CEO of Apple

there was a huge outpouring of accolades and emotion

there were many statements about how this one man had literally changed the world

Dennis Ritchie

seven days later, 12 October 2011
Dennis Ritchie died

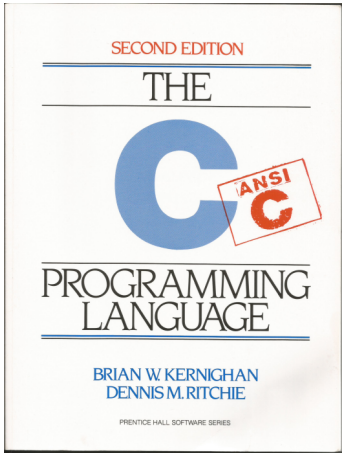
an immense response from computer scientists, but most people didn't even know he existed

but without Dennis Ritchie, there would have been no Steve Jobs and no Bill Gates either

C and Unix

Dennis Ritchie created the C language

with Ken Thompson, he used C to write Unix



I challenge you to find anything in computers or running the web that isn't based on C and Unix — including every Apple product and every Microsoft product

Unix and C Derivatives

Bill Gates' DOS was a dumbed-down ripoff of Unix, written in C

Unix and C Derivatives

Bill Gates' DOS was a dumbed-down ripoff of Unix, written in C

Windows was originally written in C
then re-written in C++ (a C derivative)
then re-re-written in C# (another C derivative)

Unix and C Derivatives

Bill Gates' DOS was a dumbed-down ripoff of Unix, written in C

Windows was originally written in C
then re-written in C++ (a C derivative)
then re-re-written in C# (another C derivative)

Java, Python, Ruby, Perl
- all are C derivatives, directly or indirectly
- all were originally implemented in C

all network switch, router, and hub software is written in C or C++
all firewall software is written in C or C++

Mac OS, Mac iOS, and Android *are* Unix

ok, FORTRAN, COBOL, and Lisp are not C derivatives (but Lisp is normally implemented in C)

From B to C

- Dennis Ritchie created C because he and Ken Thompson wanted a language to write Unix in
- they started writing Unix in assembly language, but they realized they needed a higher level language
- they tried using FORTRAN, but it just wasn't suited
- Ritchie proposed creating a new language similar to an earlier language Thompson had played with called B
- but B was an interpreted language; they needed a compiled language
- they created the language C so they could create Unix

Joke

Ritchie's running joke was that C had "all the power of assembly language and all the convenience of . . . assembly language"

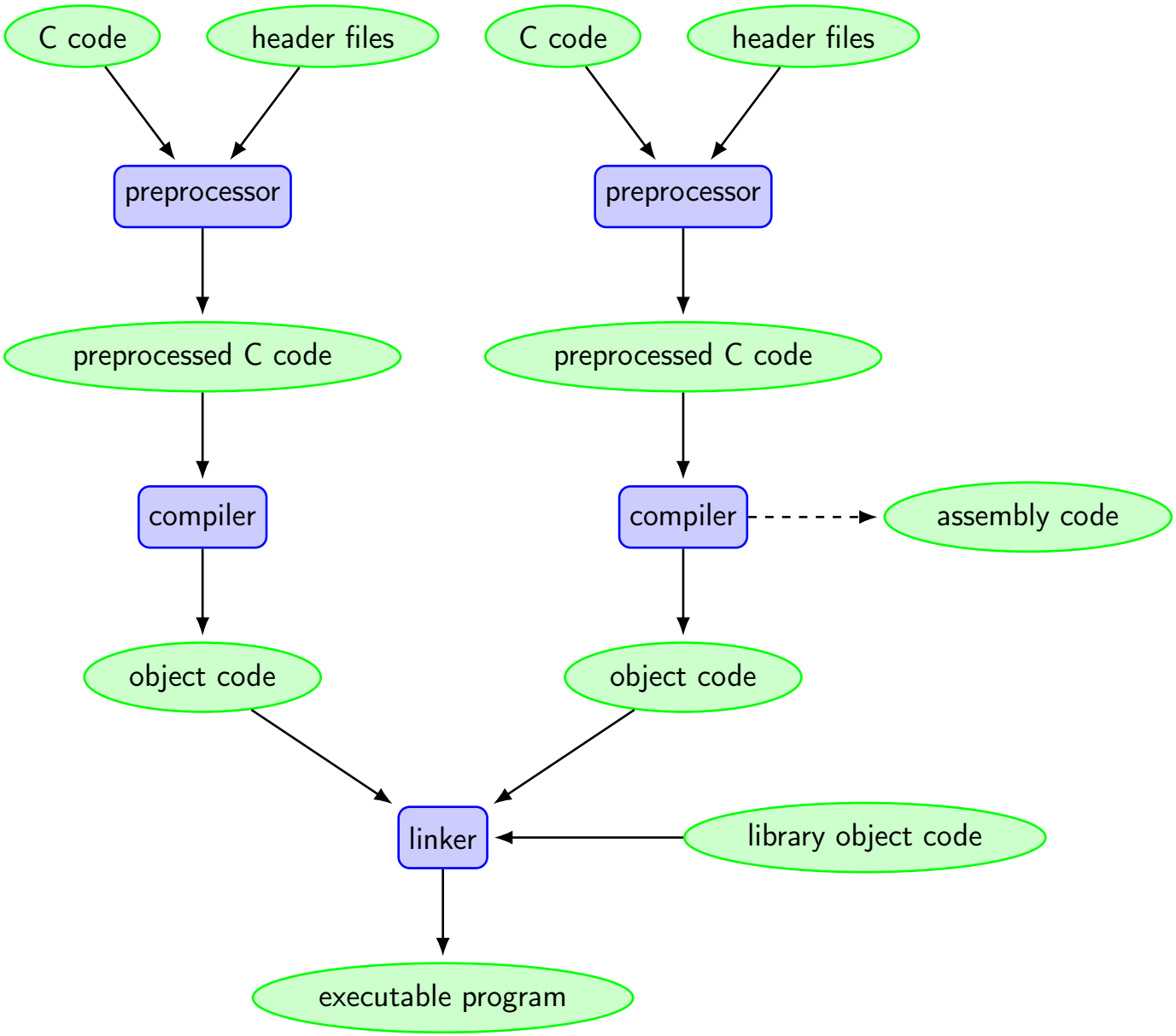
Ritchie Awards and Recognition

- 1983 ACM Turing Award
- 1990 IEEE Hamming Medal
- 1999 National Medal of Technology

Sir Isaac Newton: "If I have seen farther it is by standing on the shoulders of giants"

all of us are standing on Dennis Ritchie's shoulders

C Compilation Process



Hello World

- edit, compile, run

Compiler

- the C language standard for this course is C89
- later versions of C have borrowed many features from C++, but they're not "pure" C
- the required compiler for this course is clang
- we will use a standard set of options for clang
- and other options from time to time
- the standard compiler command we will use:

```
$ clang -pedantic-errors -Weverything \
      -std=c89 -o filename filename.c
```

  `-pedantic-errors` enforce ANSI C standards
  `-Weverything` turn on all warnings
  `-std=c89` use the C89 standard
  `-o filename` specify executable file name
  `filename.c` the source code file containing main

- use the command line to run the program:

```
$ ./filename
```

Types

- a data type is
  - a set of values
  - a set of operations defined on those values
- in C (and most other languages) there are two types of types
  - built-in primitive or fundamental types, part of the language
  - programmer-defined aggregate types, defined in code

Primitive Types

the main primitive types in C are

- void
- char
- short, int, long
- float, double
- char and the integer types can also have modifiers
  - signed
  - unsigned

- there is no Boolean data type in C

C Types

- because of historical decisions, the integer type system of C is a huge hot mess
- the two big problems are
  - unspecified sizes of the different types (e.g., int and long int may be the same size, or may be different sizes — it's up to the compile
  - totally screwy default decisions about signedness
    - rand() is guaranteed to return a non-negative result, but its return type is a defined as a signed integer
    - char is for characters
      toupper('a') returns a signed int, not an unsigned char!
- therefore, safe and robust programming dictates very careful choices about which types to use

Our Integer Types

we will use these types:

- unsigned for "normal" values that:
  - cannot be negative, which means counting and loop control
  - we don't care what size they are
- int for "normal" integer values that:
  - might be negative
  - are required because of signedness compatibility
  - we don't care what size they are
- size_t, defined in stdlib.h for any variable that stores a size:
  - sizeof
  - array indices
  - number of characters in a string (strlen)

Exact-Sized Types

- we are using C for systems programming
- very frequently we care about the size of a variable
- sometimes we need to save space
- or we need a variable big enough so it won't overflow
- the header file ⟨stdint.h⟩ defines the following:
  - `uint8_t`
  - `uint16_t`
  - `uint32_t`
  - `uint64_t`
  - `int8_t`
  - `int16_t`
  - `int32_t`
  - `int64_t`

Integer Type Rules

- use char only for character data, not for numbers
- do not use signed or unsigned with char
- use signed integers only when you have to
- most of the time we'll use unsigned integers
- use one of the exact-sized types when you need to control how big the variable is
- do not ever use long or short — you have no idea how big they are

printf

- the primary screen output mechanism in C is the printf function
- defined in ⟨stdio.h⟩ header file
- mnemonic for "print a formatted string"
- the function prototype is
  `printf(char* format, arg1, arg2, ...)`
- format is a string that contains zero or more conversion codes, each starting with %
- the number of args must match the number of codes
- the type of each arg must match the corresponding code

Example:

```
printf("Name:  %s account total:  %f\n", name, total);
```

Man Pages

- if you do `$ man printf` you get the man page for the Unix command printf in manual section 1
- if you do `$ man 3 printf` you get the man page for the C printf function in manual section 3

- man pages are divided into nine sections:
  1. Executable programs or shell commands
  2. System calls (functions provided by the kernel)
  3. Library calls (functions within program libraries)
  4. Special files (usually found in /dev)
  5. File formats and conventions, e.g. /etc/passwd
  6. Games
  7. Miscellaneous
  8. System administration commands (usually only for root)
  9. Kernel routines (Not standard)