CMSC 223 Systems Programming

Fall 2023
Bryn Mawr College
Instructor: Deepak Kumar

1

CMSC 223 Systems Programming

C was developed in 1971-72!

2

Go to class web page…

- Go to: https://cs.brynmawr.edu/
  Click on the course CMSC 223

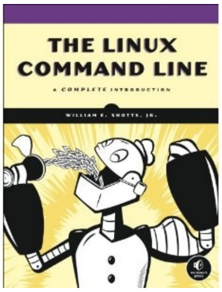
  OR

- Go to: http://www.cs.brynmawr.edu/Courses/cs223/Fall2023/

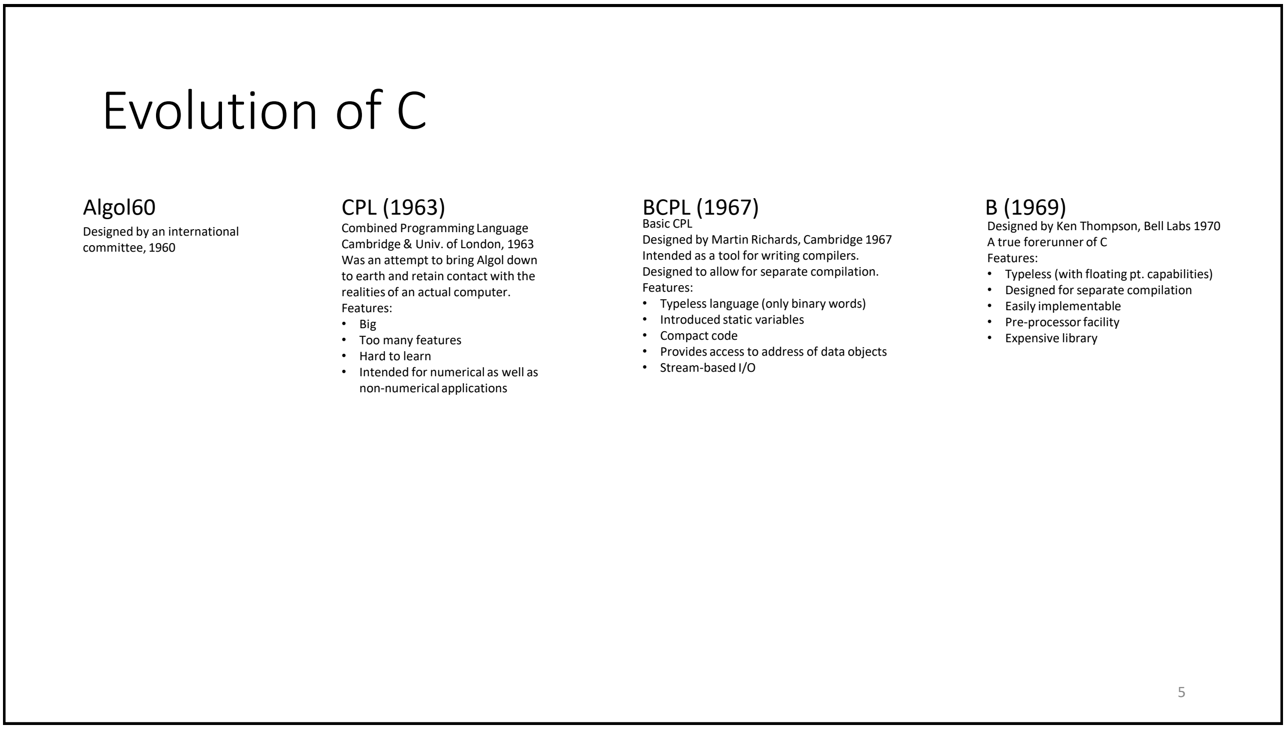
Goals

- Learn Linux (CLI, not WIMP!)
- Learn C
- Learn Linux tools

Linux™



THE C PROGRAMMING LANGUAGE



Evolution of C

Algol60

Designed by an international committee, 1960

CPL (1963)

Combined Programming Language
Cambridge & Univ. of London, 1963
Was an attempt to bring Algol down to earth and retain contact with the realities of an actual computer.
Features:

- Big
- Too many features
- Hard to learn
- Intended for numerical as well as non-numerical applications

BCPL (1967)

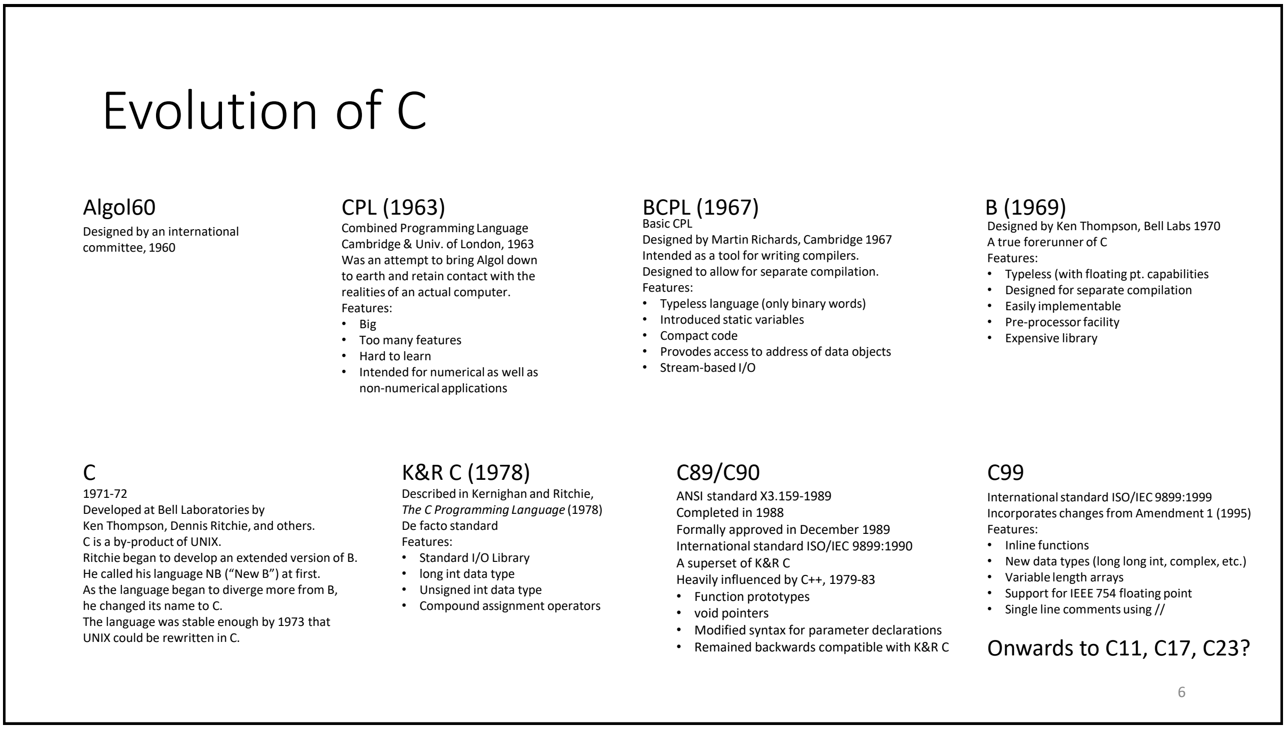
Basic CPL
Designed by Martin Richards, Cambridge 1967
Intended as a tool for writing compilers.
Designed to allow for separate compilation.
Features:

- Typeless language (only binary words)
- Introduced static variables
- Compact code
- Provides access to address of data objects
- Stream-based I/O

B (1969)

Designed by Ken Thompson, Bell Labs 1970
A true forerunner of C
Features:

- Typeless (with floating pt. capabilities)
- Designed for separate compilation
- Easily implementable
- Pre-processor facility
- Expensive library



Evolution of C

Algol60

Designed by an international committee, 1960

CPL (1963)

Combined Programming Language
Cambridge & Univ. of London, 1963
Was an attempt to bring Algol down to earth and retain contact with the realities of an actual computer.
Features:

- Big
- Too many features
- Hard to learn
- Intended for numerical as well as non-numerical applications

BCPL (1967)

Basic CPL
Designed by Martin Richards, Cambridge 1967
Intended as a tool for writing compilers.
Designed to allow for separate compilation.
Features:

- Typeless language (only binary words)
- Introduced static variables
- Compact code
- Provides access to address of data objects
- Stream-based I/O

B (1969)

Designed by Ken Thompson, Bell Labs 1970
A true forerunner of C
Features:

- Typeless (with floating pt. capabilities
- Designed for separate compilation
- Easily implementable
- Pre-processor facility
- Expensive library

C

1971-72
Developed at Bell Laboratories by
Ken Thompson, Dennis Ritchie, and others.
C is a by-product of UNIX.
Ritchie began to develop an extended version of B.
He called his language NB ("New B") at first.
As the language began to diverge more from B,
he changed its name to C.
The language was stable enough by 1973 that
UNIX could be rewritten in C.

K&R C (1978)

Described in Kernighan and Ritchie, *The C Programming Language* (1978)
De facto standard
Features:

- Standard I/O Library
- long int data type
- Unsigned int data type
- Compound assignment operators

C89/C90

ANSI standard X3.159-1989
Completed in 1988
Formally approved in December 1989
International standard ISO/IEC 9899:1990
A superset of K&R C
Heavily influenced by C++, 1979-83

- Function prototypes
- void pointers
- Modified syntax for parameter declarations
- Remained backwards compatible with K&R C

C99

International standard ISO/IEC 9899:1999
Incorporates changes from Amendment 1 (1995)
Features:

- Inline functions
- New data types (long long int, complex, etc.)
- Variable length arrays
- Support for IEEE 754 floating point
- Single line comments using //

Onwards to C11, C17, C23?

First C Program: Hello, World!

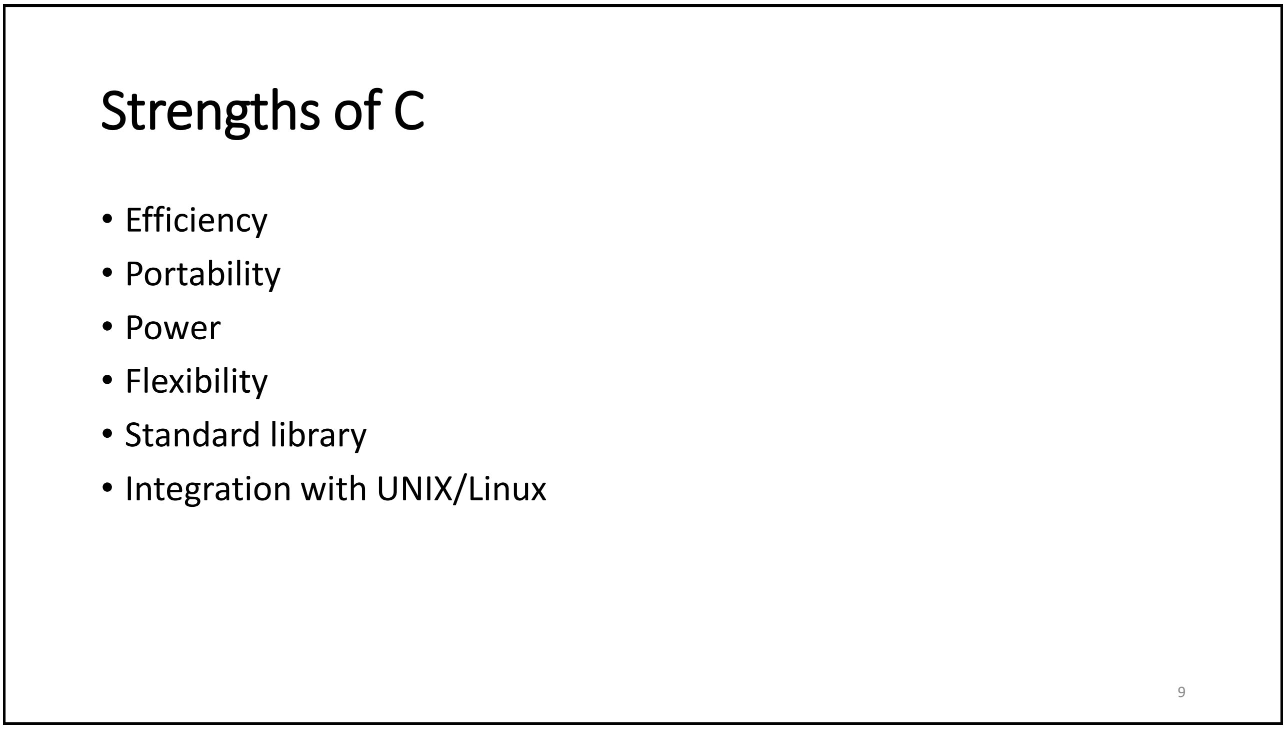
```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
}
```

- This program might be stored in a file named `hello.c`.
- The file name doesn’t matter, but the `.c` extension is often required.

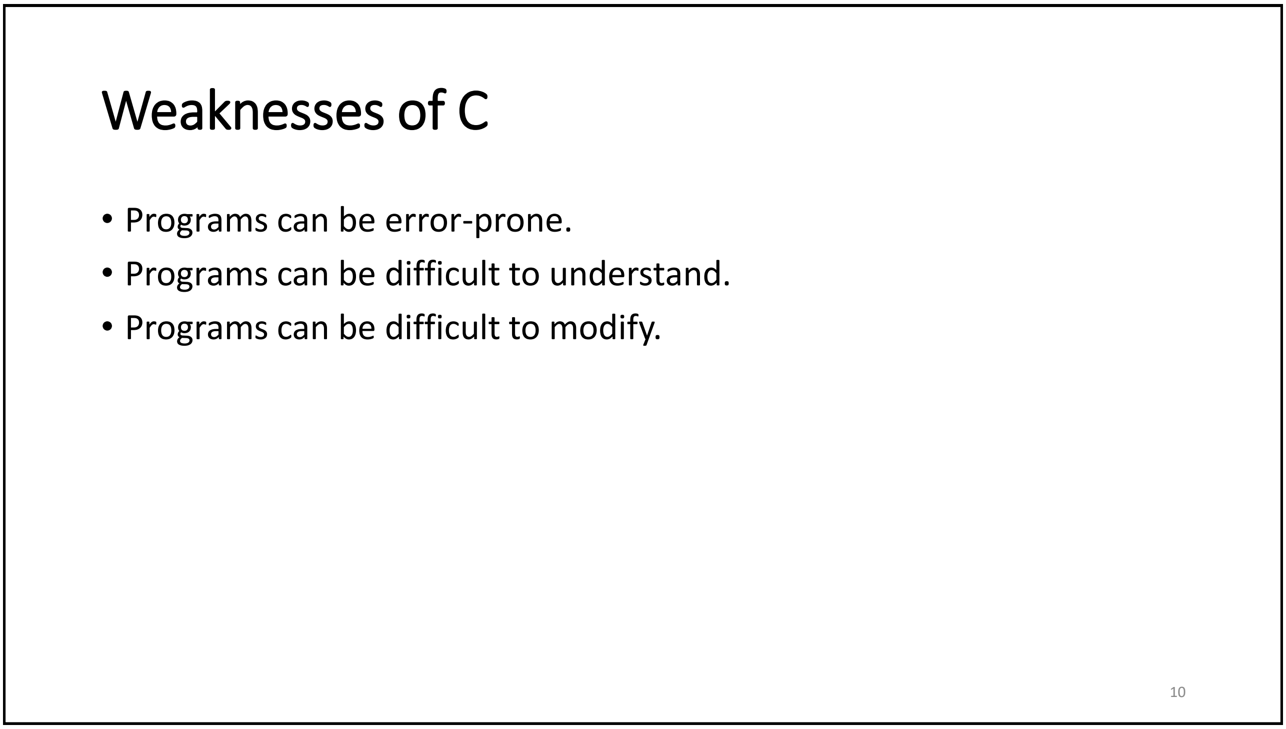
Properties of C

- Low-level
- Small
- Permissive



Strengths of C

- Efficiency
- Portability
- Power
- Flexibility
- Standard library
- Integration with UNIX/Linux



Weaknesses of C

- Programs can be error-prone.
- Programs can be difficult to understand.
- Programs can be difficult to modify.

Effective Use of C

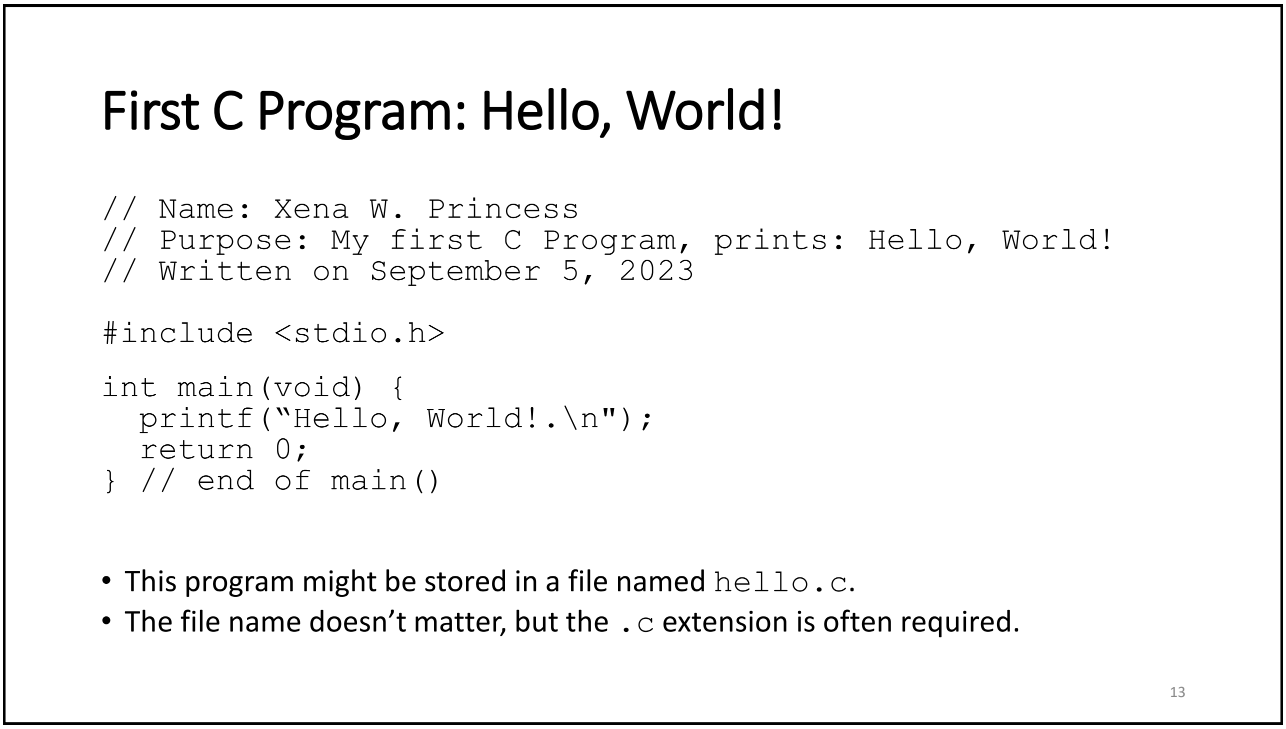
- Learn how to avoid pitfalls.
- Use software tools to make programs more reliable.
- Take advantage of existing code libraries.
- Adopt a sensible set of coding conventions.
- Avoid “tricks” and overly complex code.
- Stick to the standard.
- Try and adapt the good habits from programming in Java!

First C Program: Hello, World!

```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
}
```

- This program might be stored in a file named `hello.c`.
- The file name doesn’t matter, but the `.c` extension is often required.



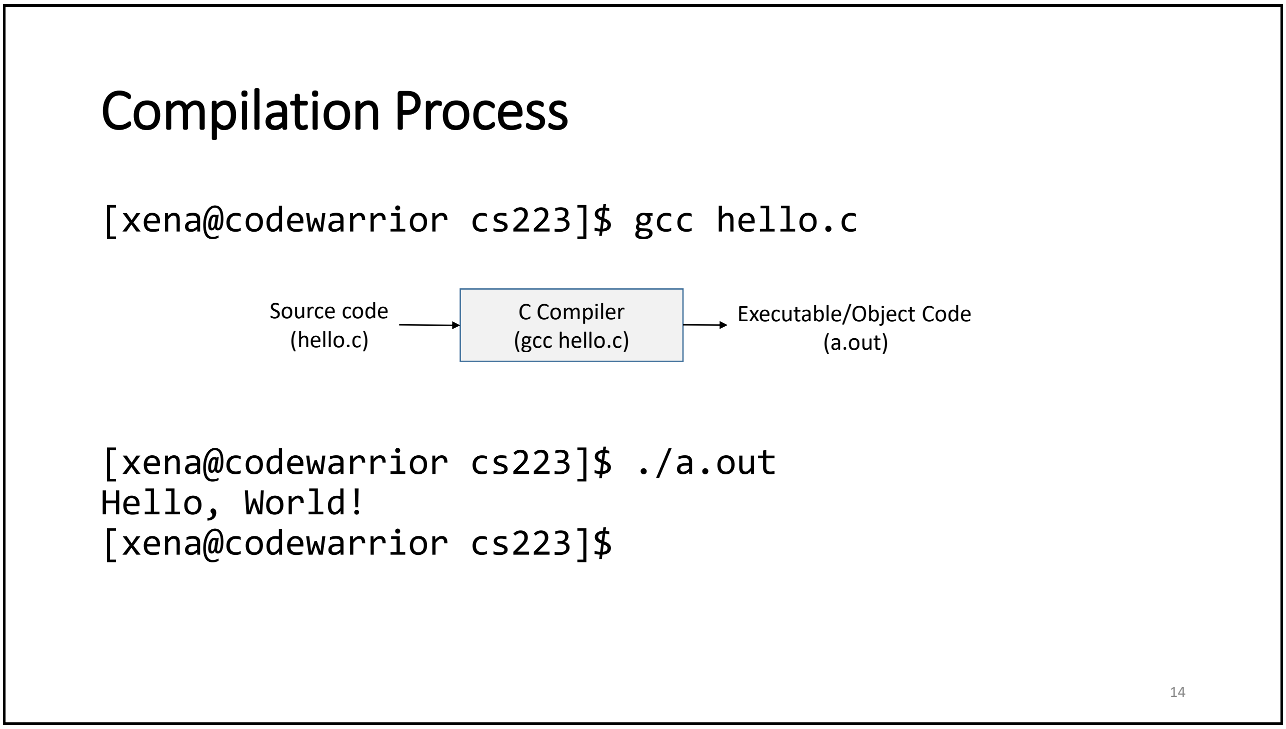
First C Program: Hello, World!

```
// Name: Xena W. Princess
// Purpose: My first C Program, prints: Hello, World!
// Written on September 5, 2023

#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
} // end of main()
```

- This program might be stored in a file named `hello.c`.
- The file name doesn’t matter, but the `.c` extension is often required.



Compilation Process

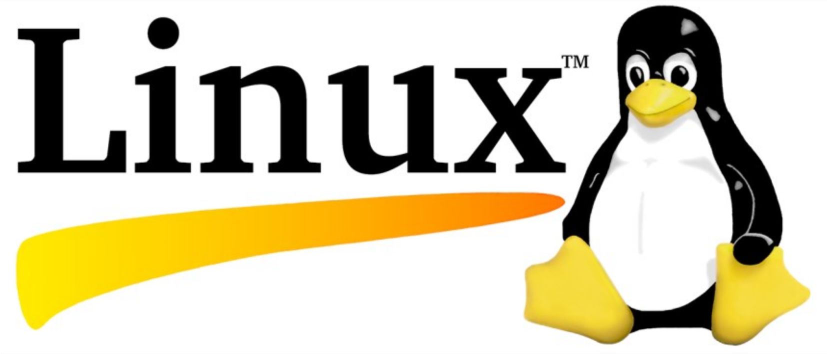
```
[xena@codewarrior cs223]$ gcc hello.c
```

Source code
(hello.c)
C Compiler
(gcc hello.c)
Executable/Object Code
(a.out)

```
[xena@codewarrior cs223]$ ./a.out
Hello, World!
[xena@codewarrior cs223]$
```

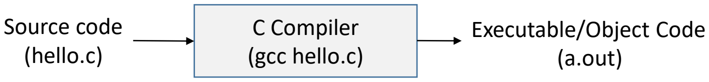
Excursion to Linux

How to create, edit, compile, and run C programs.



Compilation Process – GNU C Compiler

```
[xena@codewarrior cs223]$ gcc –o hello hello.c
```



```
[xena@codewarrior cs223]$ ./hello
Hello, World!
[xena@codewarrior cs223]$
```

Compilation Process

```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
}
```

Compilation is a 3-step process

1. **Preprocessing**
   Source code commands that begin with a # are preprocessed. E.g.,

   ```
   #include <stdio.h>
   ```

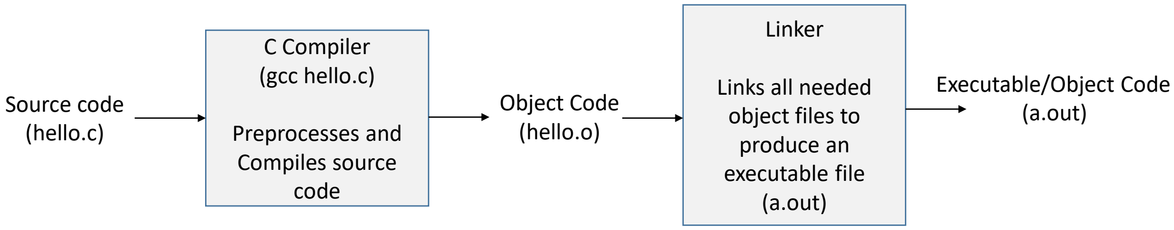
2. **Compiling**
   Source code is translated into object code (m/c language)

3. **Linking**
   All libraries/modules used by the program are linked to produce an executable object code

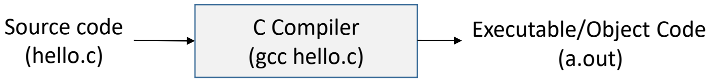
Preprocessing is normally integrated into the compiler. Linking is done by a separate program/command.

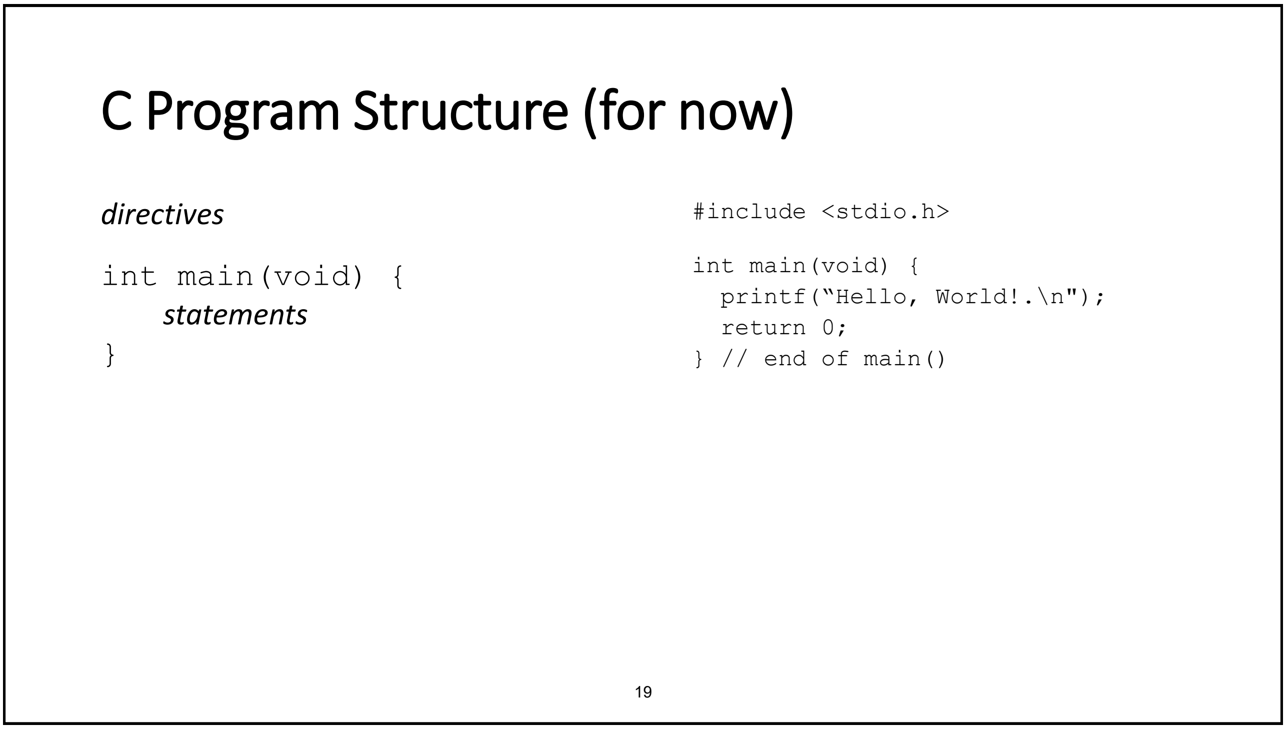
Compilation Process

Compilation is a 3-step process


Source code
(hello.c)
C Compiler
(gcc hello.c)
Preprocesses and Compiles source code
Object Code
(hello.o)
Linker
Links all needed object files to produce an executable file (a.out)
Executable/Object Code
(a.out)

The **gcc** command, in its simplest form, integrates all three steps.





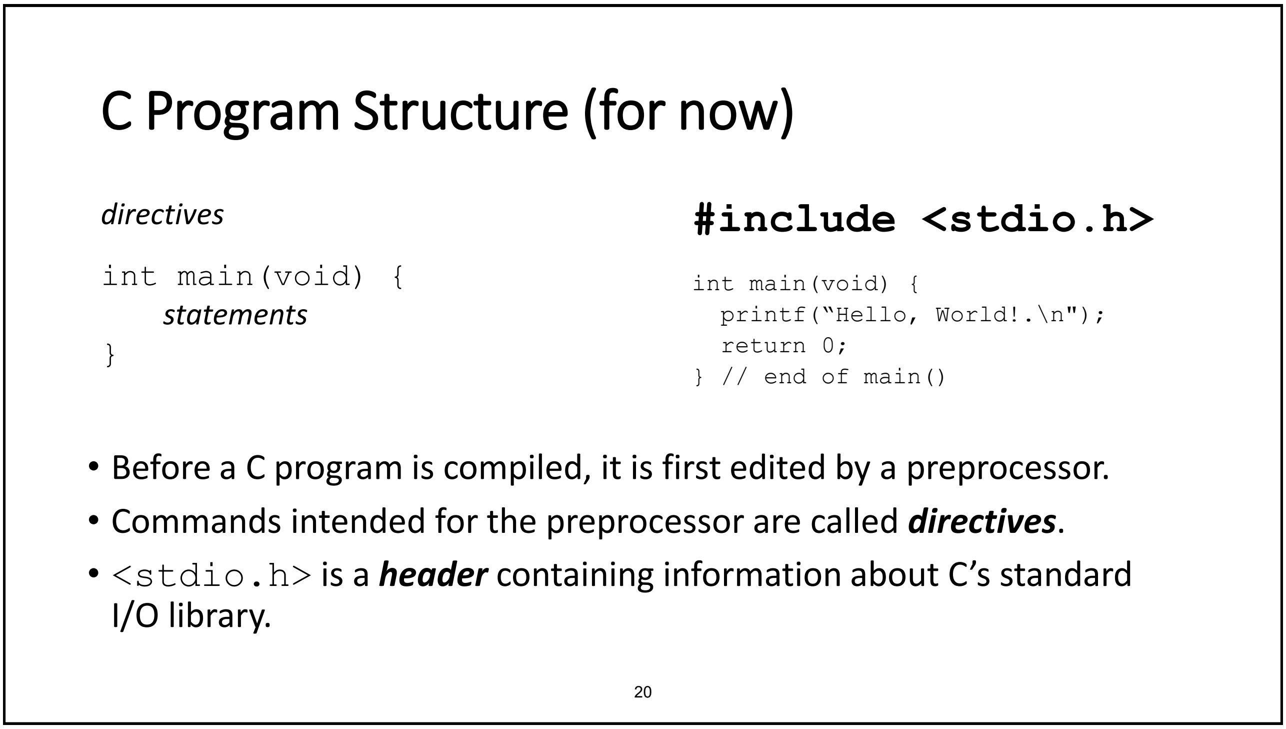
C Program Structure (for now)

directives

```
int main(void) {
```
    statements
```
}
```

```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
} // end of main()
```



C Program Structure (for now)

directives

```
int main(void) {
```
    statements
```
}
```

```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
} // end of main()
```

- Before a C program is compiled, it is first edited by a preprocessor.
- Commands intended for the preprocessor are called ***directives***.
- `<stdio.h>` is a ***header*** containing information about C’s standard I/O library.

```
#include <stdio.h>

int main(void) {
  printf(“Hello, World!.\n");
  return 0;
} // end of main()
```

`main()`

- The `main()` function is mandatory.
- `main()` is special: it gets called automatically when the program is executed.
- `main` returns a status code; the value 0 indicates normal program termination.
- If there’s no `return` statement at the end of the `main` function, many compilers will produce a warning message.

Printing Strings

- The statement

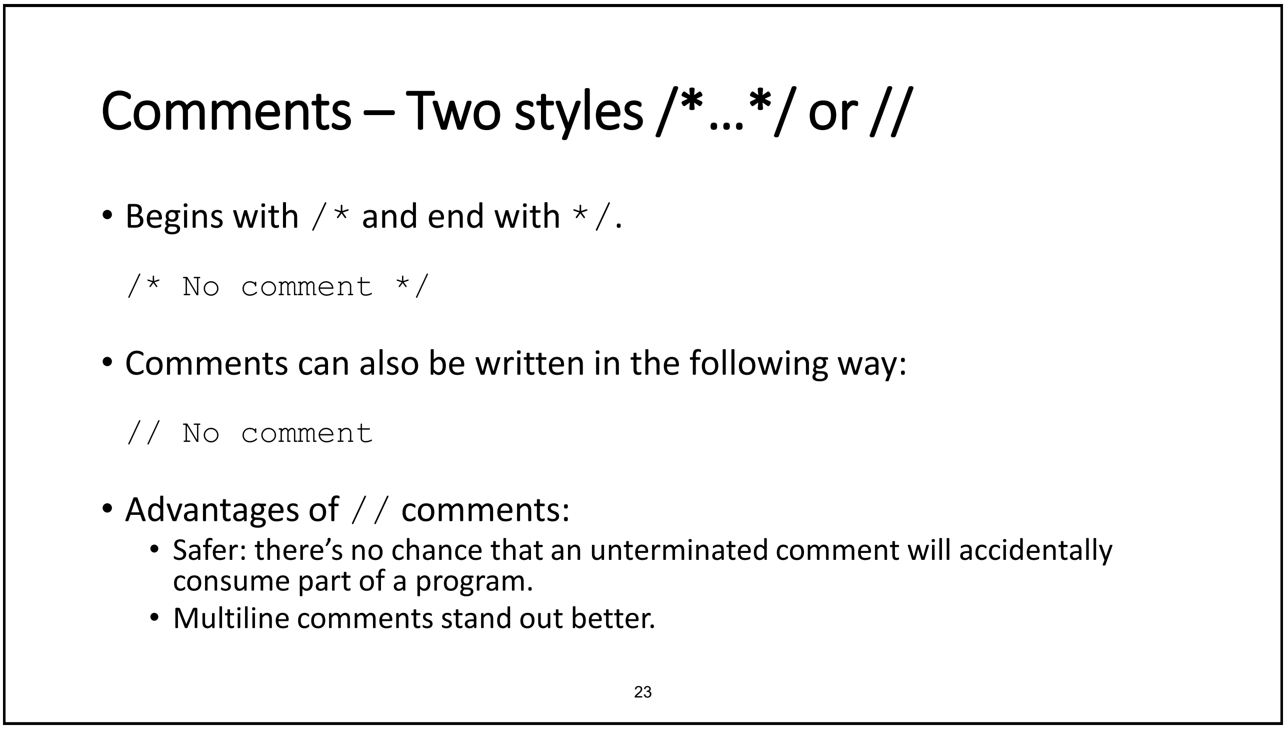
  ```
  printf("To C, or not to C: that is the question.\n");
  ```

  could be replaced by two calls of `printf`:

  ```
  printf("To C, or not to C: ");
  printf("that is the question.\n");
  ```

- The new-line character can appear more than once in a string literal:

  ```
  printf("Brevity is the soul of wit.\n  --Shakespeare\n");
  ```



Comments – Two styles /*…*/ or //

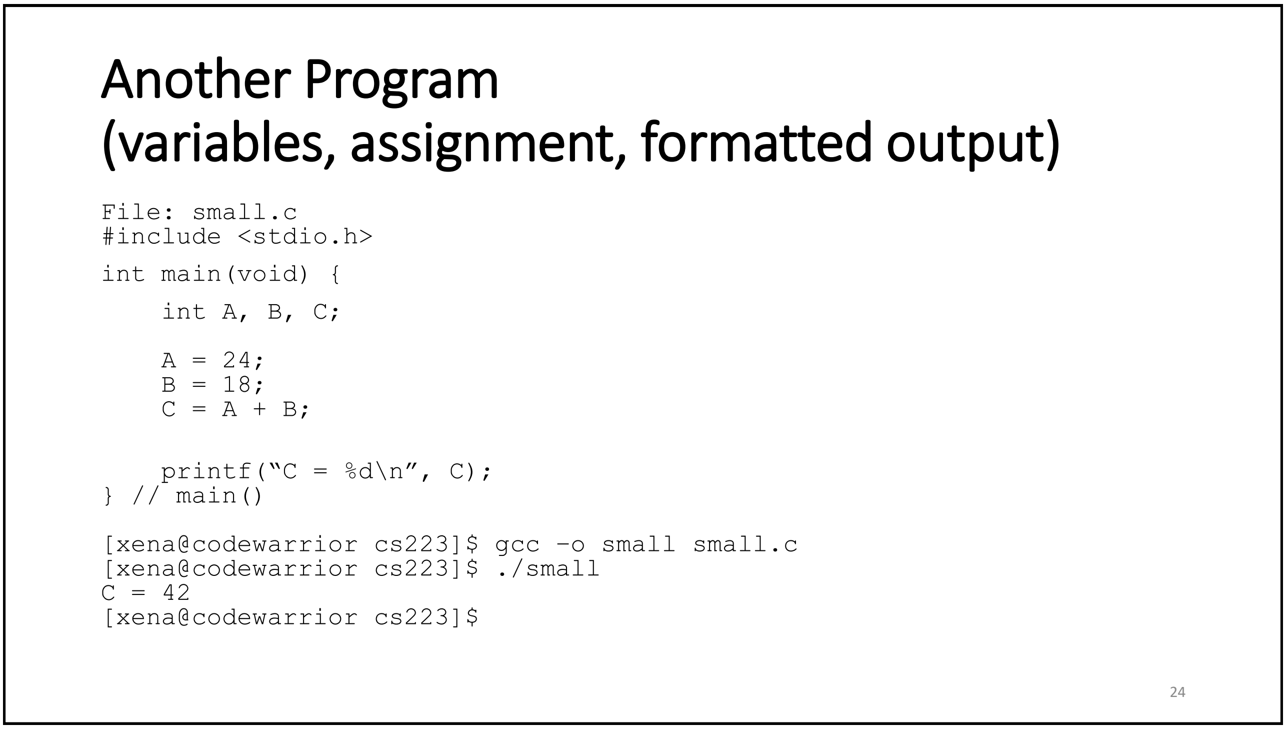
- Begins with `/*` and end with `*/`.

```
/* No comment */
```

- Comments can also be written in the following way:

```
// No comment
```

- Advantages of `//` comments:
  - Safer: there's no chance that an unterminated comment will accidentally consume part of a program.
  - Multiline comments stand out better.



Another Program (variables, assignment, formatted output)

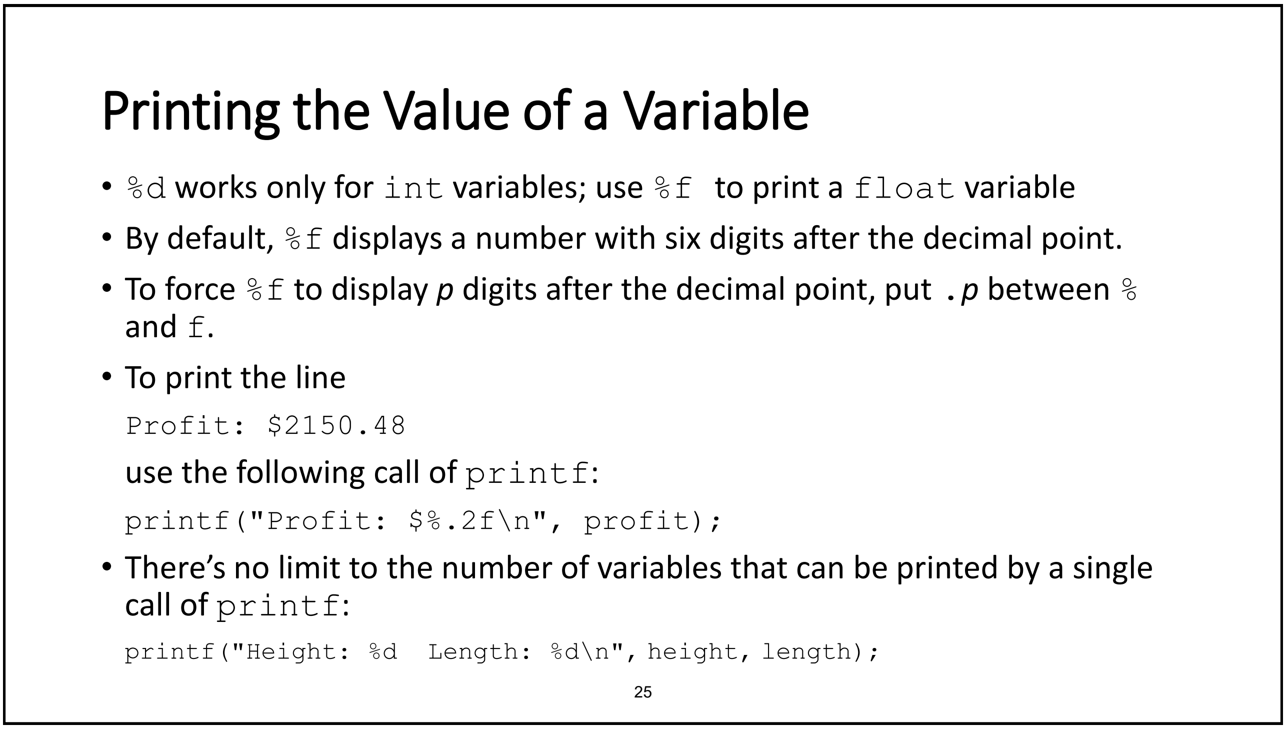
```
File: small.c
#include <stdio.h>

int main(void) {
    int A, B, C;

    A = 24;
    B = 18;
    C = A + B;

    printf(“C = %d\n”, C);
} // main()

[xena@codewarrior cs223]$ gcc -o small small.c
[xena@codewarrior cs223]$ ./small
C = 42
[xena@codewarrior cs223]$
```



Printing the Value of a Variable

- `%d` works only for `int` variables; use `%f` to print a `float` variable
- By default, `%f` displays a number with six digits after the decimal point.
- To force `%f` to display *p* digits after the decimal point, put **.***p* between `%` and `f`.
- To print the line

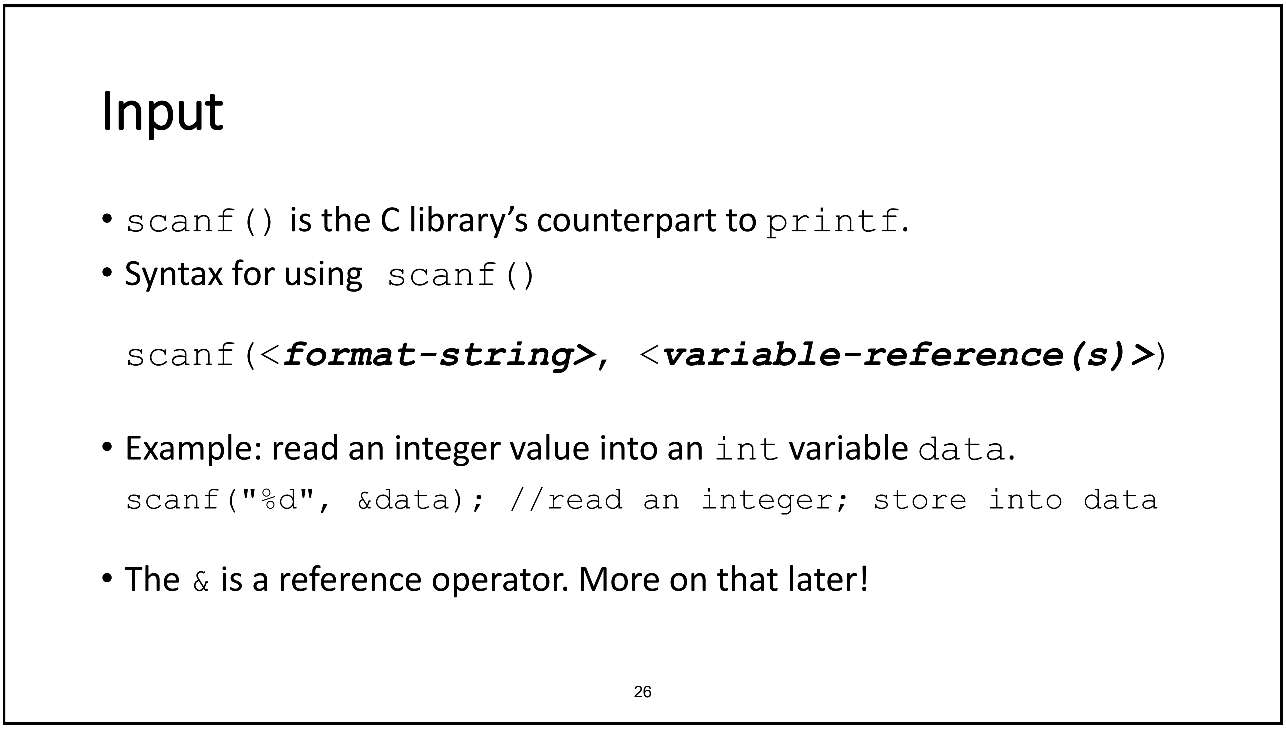
  ```
  Profit: $2150.48
  ```

  use the following call of `printf`:

  ```
  printf("Profit: $%.2f\n", profit);
  ```

- There's no limit to the number of variables that can be printed by a single call of `printf`:

  ```
  printf("Height: %d  Length: %d\n", height, length);
  ```



Input

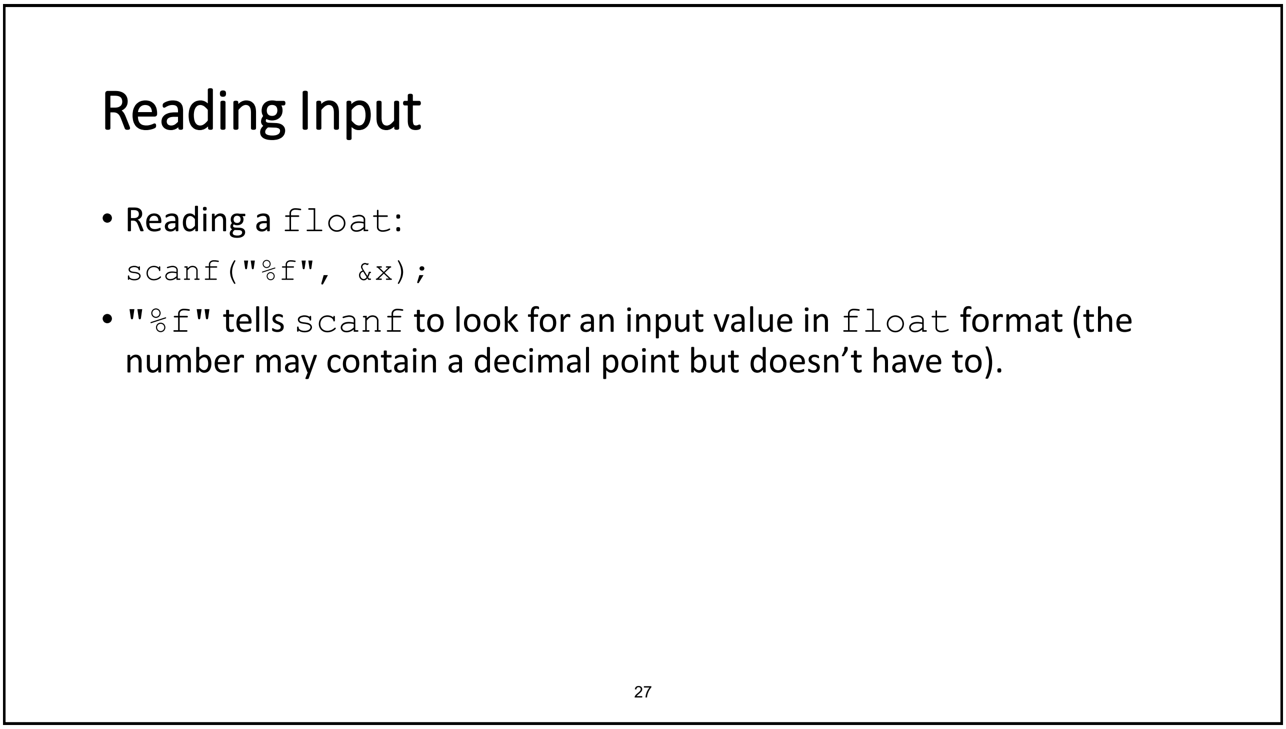
- `scanf()` is the C library's counterpart to `printf`.
- Syntax for using `scanf()`

  ```
  scanf(<format-string>, <variable-reference(s)>)
  ```

- Example: read an integer value into an `int` variable `data`.

  ```
  scanf("%d", &data); //read an integer; store into data
  ```

- The `&` is a reference operator. More on that later!



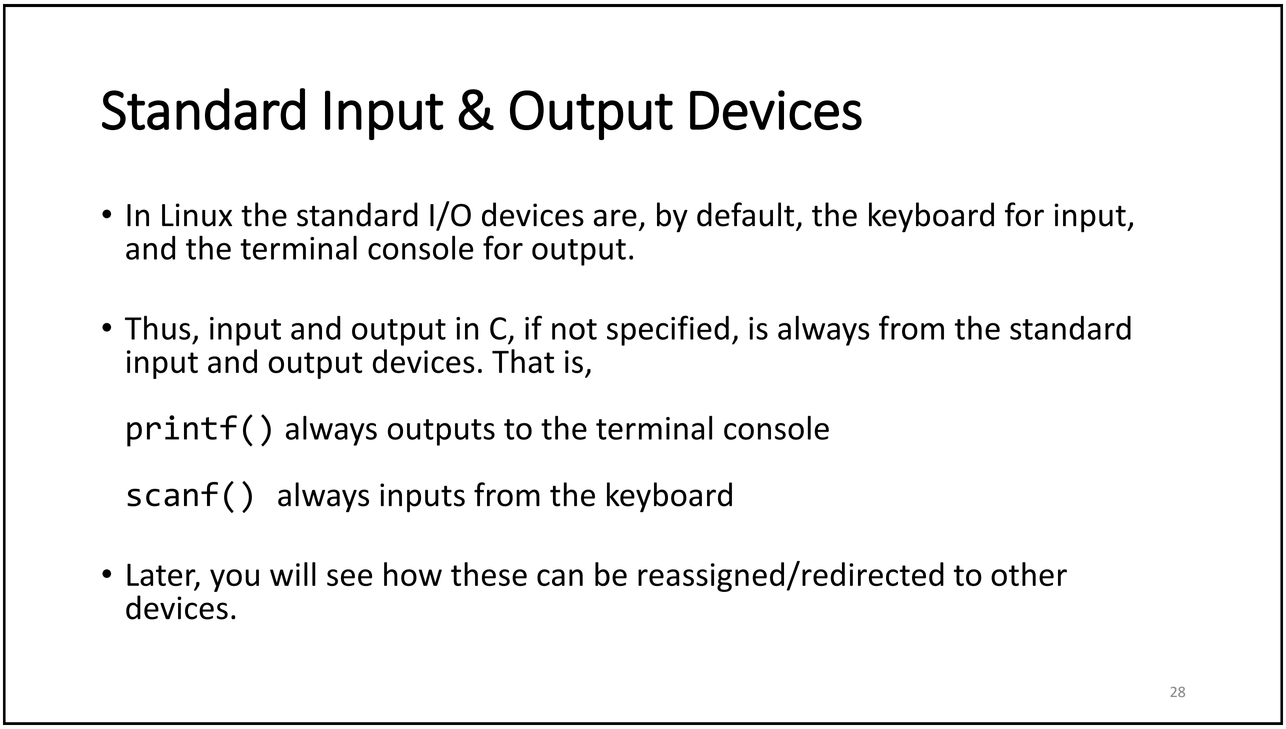
Reading Input

- Reading a `float`:

  `scanf("%f", &x);`
- `"%f"` tells `scanf` to look for an input value in `float` format (the number may contain a decimal point but doesn't have to).

27

27



Standard Input & Output Devices

- In Linux the standard I/O devices are, by default, the keyboard for input, and the terminal console for output.
- Thus, input and output in C, if not specified, is always from the standard input and output devices. That is,

  `printf()` always outputs to the terminal console

  `scanf()` always inputs from the keyboard
- Later, you will see how these can be reassigned/redirected to other devices.

28

28

Program: Convert Fahrenheit to Celsius

- The `celsius.c` program prompts the user to enter a Fahrenheit temperature; it then prints the equivalent Celsius temperature.
- Sample program output:

```
Enter Fahrenheit temperature: 212
Celsius equivalent: 100.0
```

- The program will allow temperatures that aren't integers.

Take a few minutes to write the program.

Program: Convert Fahrenheit to Celsius **ctof.c**

```
#include <stdio.h>

int main(void)
{
  float f, c;

  printf("Enter Fahrenheit temperature: ");
  scanf("%f", &f);

  c = (f - 32) * 5.0/9.0;

  printf("Celsius equivalent: %.1f\n", c);

  return 0;
} // main()
```

Sample program output:

```
Enter Fahrenheit temperature: 212
Celsius equivalent: 100.0
```

Acknowledgements

Some content from these slides is based on the book, C Programming – A Modern Approach, By K. N. King, 2nd Edition, W. W. Norton 2008.

Materials are also included from the lecture slides provided by Prof. K. N. King. Thank You!