# CS 50:
# Software Design and Implementation

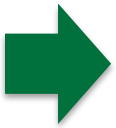
Intro to C

# Let’s C?

**Why use C?**

- Base for other languages: Java, Python, others
- OS
- Networking/Internet
- Speed
- Low level access to machine
- Write once, compile anywhere

# Agenda



1. Hello world
2. From C source to executable
3. Building a C program step by step
4. Activity

# First start with comments

**blank.c**

```
 1 /*
 2  * name.c - describe what the program does
 3  *
 4  * input:
 5  * output:
 6  *
 7  * compile: mygcc -o name name.c ...
 8  * usage: ./name  params
 9  *
10  * Tim Pierson, date
11  * CS 50, term
12  *
13  */
14 #include <stdio.h>
15
16 int main(int argc, char *argv[]) {
17
18
19     return 0;
20 }
```

I like to start with a blank.c file with a template for comments

Edit comments for new program

Almost always want to import stdio.h and return 0 if successful

Copy to new file

```
$ cp blank.c helloworld.c
$ vi helloworld.c
```

# Then add hello world!

**helloworld.c**

```c
/*
 * helloworld.c - traditional first program
 *
 * input:
 * output:
 *
 * compile: mygcc -o helloworld helloworld.c ...
 * usage: ./helloworld
 *
 * Tim Pierson, Fall 2022
 * CS 50, Fall 2022
 *
 */
#include <stdio.h>

int main(int argc, char *argv[]) {
    printf("Hello world!\n");

    return 0;
}
```

Update comments

Write code
Here only printf

We will use gcc as our compiler
mygcc is an alias that adds several flags

```
$ alias mygcc
alias mygcc='gcc -Wall -pedantic -std=c11 -ggdb'

$ mygcc helloworld.c
$ ./a.out
Hello world!
```

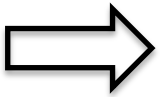
Compile C program
Output is a.out unless we use –o flag

# Agenda

1. Hello world
2. From C source to executable
3. Building a C program step by step
4. Activity

# C is a “high-level language that is compiled into machine-executable code

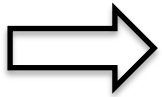
**C code**



```c
#include <stdio.h>

int main()
{
    int a = 1, b = 2;
    printf("Hello world\n");
    return 0;
}
```

- **C code is easily read by humans (but not computers, yet!)**
- **Similar to Java (Java was based on C)**
- **Transformed into executable code via the compilation process**

**Assembly code**



```
        .file   "hello.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello world"
.LC2:
        .string "%f\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $1, -8(%rbp)
        movl    $2, -4(%rbp)
<snip>
```

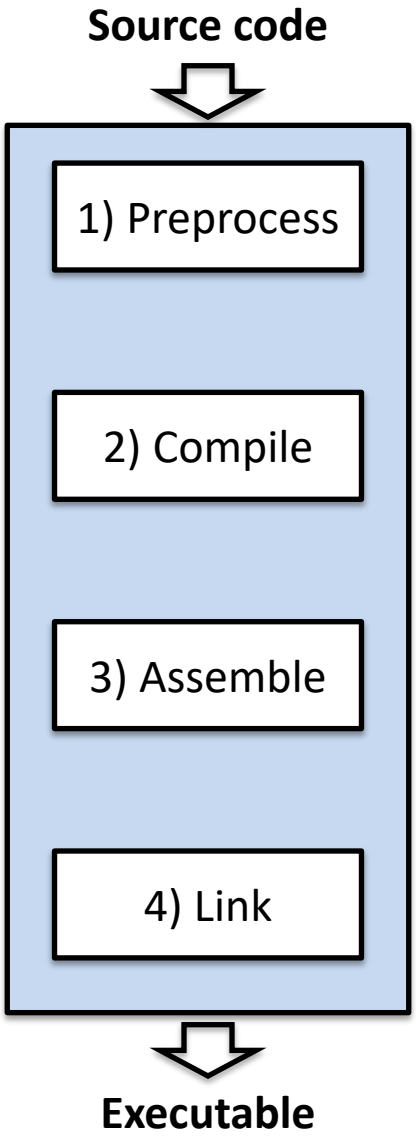
**Executable code**

```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 3e 00 01 00 00 00  00 00 00 00 00 00 00 00  |..>.............|
00000020  00 00 00 00 00 00 00 00  78 03 00 00 00 00 00 00  |........x.......|
00000030  00 00 00 00 40 00 00 00  00 00 40 00 0d 00 0c 00  |....@.....@.....|
00000040  55 48 89 e5 48 83 ec 10  c7 45 f8 01 00 00 00 c7  |UH..H....E......|
00000050  45 fc 02 00 00 00 48 8d  3d 00 00 00 00 e8 00 00  |E.....H.=.......|
00000060  00 00 8b 55 f8 8b 45 fc  01 d0 f2 0f 2a c0 f2 0f  |...U..E.....*...|
00000070  10 0d 00 00 00 00 f2 0f  59 c1 48 8d 3d 00 00 00  |........Y.H.=...|
00000080  00 b8 01 00 00 00 e8 00  00 00 00 b8 00 00 00 00  |................|
```

# We will use several different types of files to create executable code

| File type | Use |
|---|---|
| .c | C code to implement a solution to a problem |
| .h | Header file that contains<br>• Function declarations<br>• Constants<br>• Macros |
| .o | Object file, mostly machine code, but not linked with system or other code |
| Makefile | Instructions on how to compile program (command line becomes tricky with large programs, Makefile saves typing) |
| README | Description of program, expected inputs and expected outputs |
| TESTING | Script to demonstrate that the program behaves as expected |

# C code is converted into an executable in a four-step compilation process

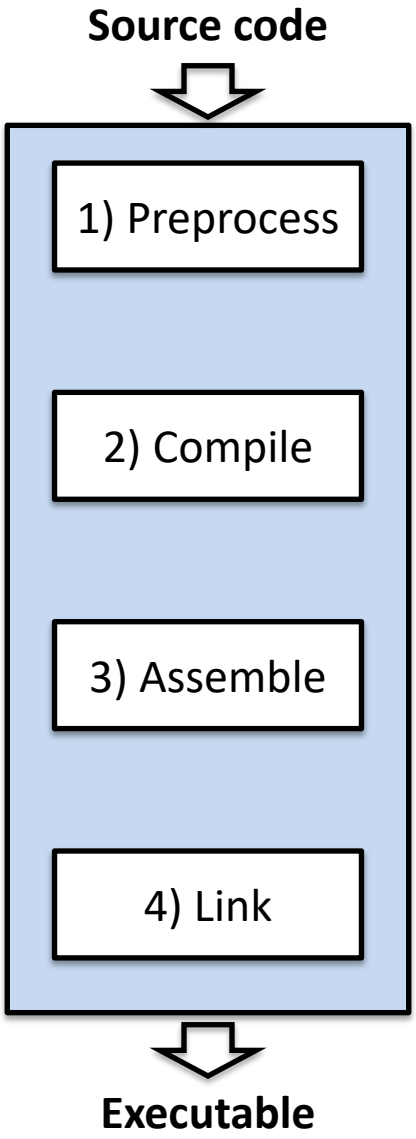



- **Include standard I/O library (printf)**
- **Almost always include this library**
- **.h is a "header" file**
- **<> tells compiler to find stdio.h file in normal location for include files (/usr/include)**

**hello.c**

```
#include <stdio.h>
#define add(a,b) (a+b)
#define PI 3.14159


int main()
{
    int a = 1, b = 2;
    printf("Hello world\n");
    printf("%f\n",add(a,b)*PI);
    return 0;
}
```

# C code is converted into an executable in a four-step compilation process




- **#define tells compiler to find instances of first parameter and replace with second**
- **Find PI and replace with 3.14159**
- **Text replacement**
- **No data types involved with define**

**hello.c**

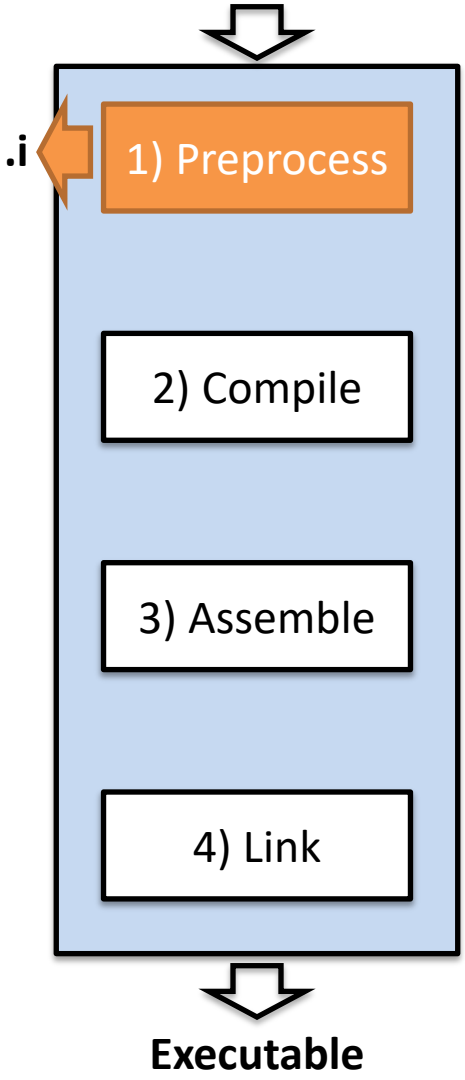
```
#include <stdio.h>
#define add(a,b) (a+b)
#define PI 3.14159


int main()
{
    int a = 1, b = 2;
    printf("Hello world\n");
    printf("%f\n",add(a,b)*PI);
    return 0;
}
```

- **#define not recommended anymore**
- **You might see it in legacy code (it used to be popular, some people still use it)**
- **Instead use `const int PI=314159;`**

# Step 1: preprocess C code to remove comments, expand macros and includes

$ gcc -o hello –save-temps hello.c

hello.i




- Remove comments
- Expand macros
- Expand include files
- Code still in C
- Outputs .i file

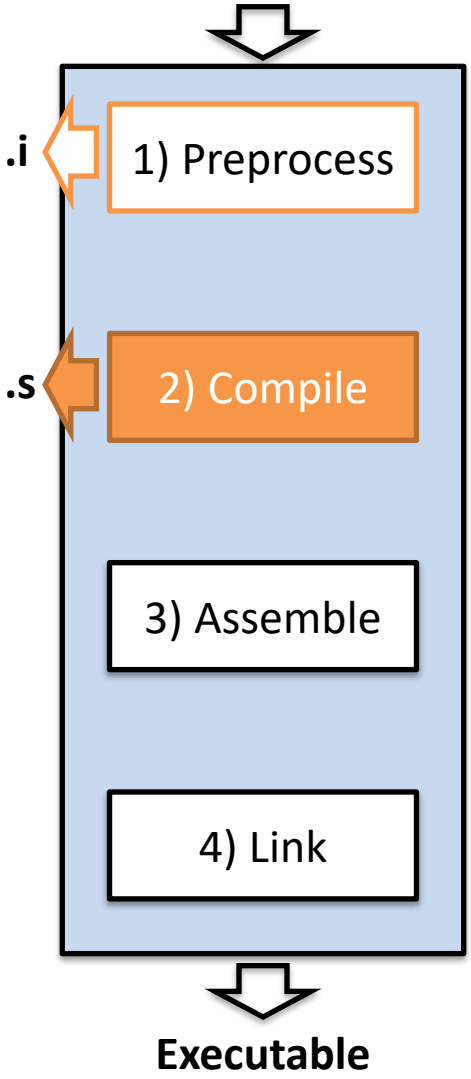
**References to stdio.h code included**

**add(a,b) changed to (a+b)**
**PI changed to 3.14159**

```
<snip>
extern void funlockfile (FILE
*__stream) __attribute__
((__nothrow__ , __leaf__));
# 868 "/usr/include/stdio.h" 3 4


# 2 "hello.c" 2


# 5 "hello.c"
int main()
{
 int a = 1, b = 2;
 printf("Hello world\n");
 printf("%f\n",(a+b)*3.14159);
 return 0;
}
```

# Step 2: Compile expanded C code into low-level assembly code

$ gcc -o hello –save-temps hello.c



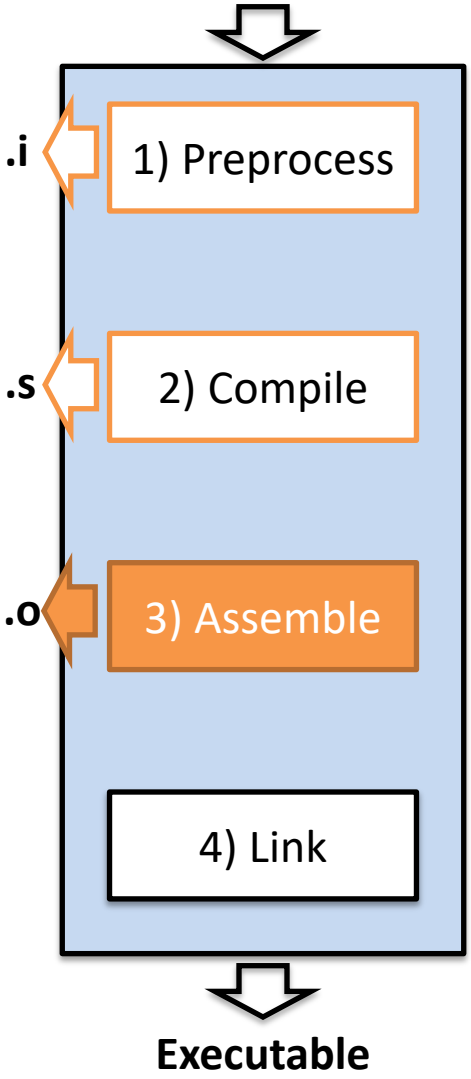

- **String constant**
- Produce assembly code
- Outputs .s file

hello.s

```
        .file   "hello.c"
        .text
        .section        .rodata
.LC0:
        .string "Hello world"
.LC2:
        .string "%f\n"
        .text
        .globl  main
        .type   main, @function
main:
.LFB0:
        .cfi_startproc
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset 6, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register 6
        subq    $16, %rsp
        movl    $1, -8(%rbp)
        movl    $2, -4(%rbp)
      <snip>
```

# Step 3: Produce object code ready to be linked with system or other code

$ gcc -o hello –save-temps hello.c




hello.o

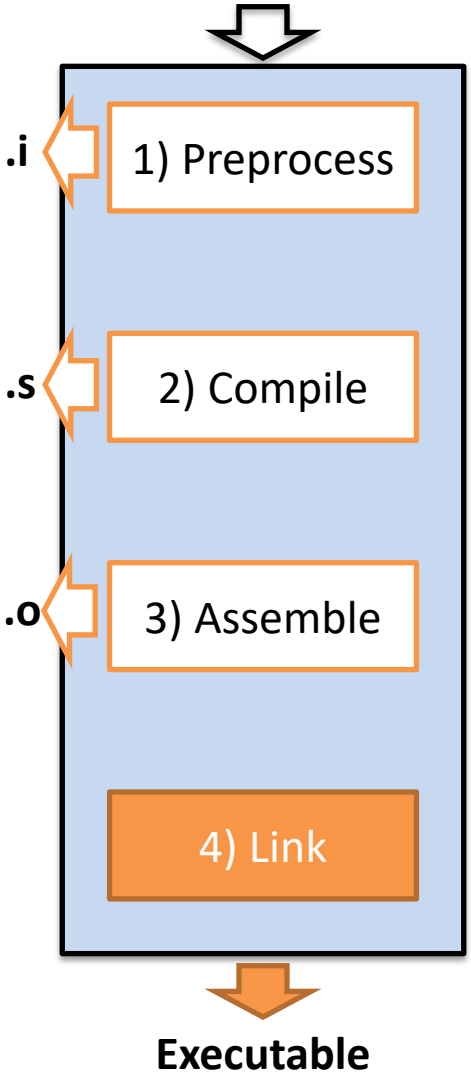
```
00000000  7f 45 4c 46 02 01 01 00  00 00 00 00 00 00 00 00  |.ELF............|
00000010  01 00 3e 00 01 00 00 00  00 00 00 00 00 00 00 00  |..>.............|
00000020  00 00 00 00 00 00 00 00  78 03 00 00 00 00 00 00  |........x.......|
00000030  00 00 00 00 40 00 00 00  00 00 40 00 0d 00 0c 00  |....@.....@.....|
00000040  55 48 89 e5 48 83 ec 10  c7 45 f8 01 00 00 00 c7  |UH..H....E......|
00000050  45 fc 02 00 00 00 48 8d  3d 00 00 00 00 e8 00 00  |E.....H.=.......|
00000060  00 00 8b 55 f8 8b 45 fc  01 d0 f2 0f 2a c0 f2 0f  |...U..E.....*...|
00000070  10 0d 00 00 00 00 f2 0f  59 c1 48 8d 3d 00 00 00  |........Y.H.=...|
00000080  00 b8 01 00 00 00 e8 00  00 00 00 b8 00 00 00 00  |................|
00000090  c9 c3 00 00 00 00 00 00  48 65 6c 6c 6f 20 77 6f  |........Hello wo|
000000a0  72 6c 64 00 25 66 0a 00  6e 86 1b f0 f9 21 09 40  |rld.%f..n....!.@|
000000b0  00 47 43 43 3a 20 28 55  62 75 6e 74 75 20 37 2e  |.GCC: (Ubuntu 7.|
000000c0  35 2e 30 2d 33 75 62 75  6e 74 75 31 7e 31 38 2e  |5.0-3ubuntu1~18.|
```

- Convert assembly into object code
- Mostly machine code, but not yet executable
- Libraries not yet linked
- Outputs .o file

# Step 4: Linked required code into an executable file

$ gcc -o hello –save-temps hello.c

./hello




```
00000370  00 00 00 00 00 00 00 00  00 6c 69 62 63 2e 73 6f  |.........libc.so|
00000380  2e 36 00 70 75 74 73 00  70 72 69 6e 74 66 00 5f  |.6.puts.printf._|
00000390  5f 63 78 61 5f 66 69 6e  61 6c 69 7a 65 00 5f 5f  |_cxa_finalize.__|
000003a0  6c 69 62 63 5f 73 74 61  72 74 5f 6d 61 69 6e 00  |libc_start_main.|
000003b0  47 4c 49 42 43 5f 32 2e  32 2e 35 00 5f 49 54 4d  |GLIBC_2.2.5._ITM|
000003c0  5f 64 65 72 65 67 69 73  74 65 72 54 4d 43 6c 6f  |_deregisterTMClo|
000003d0  6e 65 54 61 62 6c 65 00  5f 5f 67 6d 6f 6e 5f 73  |neTable.__gmon_s|
000003e0  74 61 72 74 5f 5f 00 5f  49 54 4d 5f 72 65 67 69  |tart__._ITM_regi|
000003f0  73 74 65 72 54 4d 43 6c  6f 6e 65 54 61 62 6c 65  |sterTMCloneTable|
```

```
tjp@plank$ ls -l hello*
-rwxr-xr-x 1 d84xxxx thayerusers 17256 Sep 19 10:25 hello*
-rw-r--r-- 1 d84xxxx thayerusers   419 Sep 19 09:54 hello.c
-rw-r--r-- 1 d84xxxx thayerusers 13984 Sep 19 10:25 hello.i
-rw-r--r-- 1 d84xxxx thayerusers  3968 Sep 19 10:25 hello.o
-rw-r--r-- 1 d84xxxx thayerusers  5291 Sep 19 10:25 hello.s
```

- Link libraries and system files
- Outputs executable

CS 57 for more info

**hello executable has a larger size than hello.o due to linked system code for printf from stdio library**

# Run executable via command line

**hello.c**

```c
#include <stdio.h>
#define add(a,b) (a+b)
#define PI 3.14159


int main()
{
    int a = 1, b = 2;
    printf("Hello world\n");
    printf("%f\n",add(a,b)*PI);
    return 0;
}
```

**Use \n for new line, otherwise output is all on same line!**

**gcc is the compiler we will use in CS50**

**--save-temps tells compiler to save all files used in compilation (for demonstration, this flag is normally not included)**

```
tjp@plank$ mygcc --save-temps -o hello hello.c
tjp@plank$ ./hello
Hello world
9.424770
```

**-o tells the compiler what to name the executable**

**Defaults to a.out if –o not provided**

**Code to compile**

# Agenda

1. Hello world
2. From C source to executable
3. Building a C program step by step
4. Activity

# First start with comments

**blank.c**

```
 1 /*
 2  * name.c - describe what the program does
 3  *
 4  * input:
 5  * output:
 6  *
 7  * compile: mygcc -o name name.c ...
 8  * usage: ./name  params
 9  *
10  * Tim Pierson, date
11  * CS 50, term
12  *
13  */
14 #include <stdio.h>
15
16 int main(int argc, char *argv[]) {
17
18
19     return 0;
20 }
```

I like to start with a blank.c file with a template for comments

Edit comments for new program

Almost always want to import stdio.h and return 0 if successful

Copy to new file

```
$ cp blank.c myprog.c
$ vi myprog.c
```

# guessprime1.c does the same things as our guessprime.sh bash script from last class

**guessprime1.c**

```
   <comments above>

20 #include <stdio.h>
21
22 int main(int argc, char *argv[]) {
23   const int answer = 31;
24   int guess;
25
26   printf("Enter a prime between 1-100: ");
27   scanf("%d", &guess);
28
29   // compare guess number to answer
30   while (guess != answer) {
31     printf("Wrong! try again\n");
32     printf("Enter a prime between 1-100: ");
33     scanf("%d", &guess);
34   }
35
36   return 0;  // exit status
37 }
```

Include C's standard input/output routines
You will almost always need to include this!

Execution begins at `main` function (like Java)

`const` means constant (will not change)

To go to new line, end printf with '\n'
Here we want the input to appear on the same line so we do not use '\n'

Read from stdin

Loop until user enters 31

Return exit code of 0 if successful

# guessprime1.c does the same things as our guessprime.sh bash script from last class

**guessprime1.c**

```
    <comments above>

 20 #include <stdio.h>
 21
 22 int main(int argc, char *
 23   const int answer = 31;
 24   int guess;
 25
 26   printf("Enter a prime between 1-100: ");
 27   scanf("%d", &guess);
 28
 29   // compare guess number to answer
 30   while (guess != answer) {
 31     printf("Wrong! try again\n");
 32     printf("Enter a prime between 1-100: ");
 33     scanf("%d", &guess);
 34   }
 35
 36   return 0;  // exit status
 37 }
```

```
$ mygcc -o guessprime guessprime1.c
$ ./guessprime
Enter a prime between 1-100: 20
Wrong! try again
Enter a prime between 1-100: 5
Wrong! try again
Enter a prime between 1-100: 31
$ echo $?
0
```

# guessprime2.c add a function to get user input

**guessprime2.c**

```
    <comments above>

18 #include <stdio.h>
19
20 // Ask for and read a guess
21 int readGuess() {
22   int guess;
23
24   printf("Enter a prime between 1-100: ");
25   scanf("%d", &guess);
26
27   return guess;
28 }
29
30
31 // Main function - ask for a guess, quit when it matches the answer and keep asking otherwise
32 int main(int argc, char *argv[]) {
33   const int answer = 31;
34   int guess;
35
36   guess = readGuess();
37   while (guess != answer) {
38     printf("Wrong! try again\n");
39     guess = readGuess();
40   }
41
42   return 0;  // exit status
43 }
```

**Functions return one value (like Java)**
**Here we prompt the user for input and return their guess as an integer**

**Call readGuess when we want the user's input**

# guessprime2.c add a function to get user input

**guessprime2.c**

```
    <comments above>

18 #include <stdio.h>
19
20 // Ask for and read a guess
21 int readGuess() {
22   int guess;
23
24   printf("Enter a prime between 1-1
25   scanf("%d", &guess);
26
27   return guess;
28 }
29
30
31 // Main function - ask for a guess, quit when it matches the answer and keep asking otherwise
32 int main(int argc, char *argv[]) {
33   const int answer = 31;
34   int guess;
35
36   guess = readGuess();
37   while (guess != answer) {
38     printf("Wrong! try again\n");
39     guess = readGuess();
40   }
41
42   return 0;  // exit status
43 }
```

```
$ mygcc -o guessprime guessprime2.c
$ ./guessprime
Enter a prime between 1-100: 6
Wrong! try again
Enter a prime between 1-100: 15
Wrong! try again
Enter a prime between 1-100: 31
$ echo $?
0
```

# guessprime3.c moves `readGuess` function after `main`

**guessprime3.c**

```
18 #include <stdio.h>
19
20 int readGuess(); // declaration of a function prototype
21
22 // Main function - ask for a guess, quit if it matches the answer
23 int main(int argc, char *argv[]) {
24   const int answer = 31;
25   int guess;
26
27   guess = readGuess();
28   while (guess != answer) {
29     printf("Wrong! try again\n");
30     guess = readGuess();
31   }
32
33   return 0;  // exit status
34 }
35
36
37 // Ask for and read a guess
38 // function actual *definition*
39 int readGuess() {
40   int guess;
41
42   printf("Enter a prime between 1-100: ");
43   scanf("%d", &guess);
44
45   return guess;
46 }
```

**Must declare function prototype if function definition occurs after `main`**

**In C a function's *declaration* tells the compiler the return type plus the number and type of parameters**

**A function's *definition* provides the code that implements the function**

**This will be important when we discuss header files**

**Pierson's controversial view: put main last!**

**Here `readGuess` is defined *after* `main`**

# guessprime4.c adds some defensive programming

**guessprime4.c**

```
18 #include <stdio.h>
19 #include <stdbool.h>
20
21 // function prototype *declarations*
22 int readGuess(void);
23 bool isPrime(const int p);
24
25 // Main function - ask for a guess, quit if it matches the answer and keep asking otherwise
26 int main(int argc, char *argv[]) {
27   const int answer = 31;
28   int guess;
29
30   guess = readGuess();
31   while (guess != answer) {
32     printf("Wrong! try again\n");
33     guess = readGuess();
34   }
35
36   return 0;  // exit status
37 }
38
39
```

**In C to use Booleans we must include stdbool.h :-|**

**Adding new function to check if user's input was a prime number**

**Notice parameter declared to be const, we will not change its value**

**Const not required**

# guessprime4.c adds some defensive programming

```
40 // Ask for and read a guess
41 // function actual *definition*
42 int readGuess(void) {
43   int guess;
44
45   printf("Enter a prime between 1-100: ");
46   scanf("%d", &guess);
47
48   if (guess < 1 || guess > 100) {
49     printf("Hey! %d is out of range [1..100].\n", guess);
50   }
51
52   if (!isPrime(guess)) {
53     printf("Hey! %d is not even a prime number.\n", guess);
54   }
55
56   return guess;
57 }
58
59
60 // Check whether the number is prime
61 // (assuming n <= 100)
62 bool isPrime(const int n) {
63   if (n < 2) return false;
64   if (n > 2 && n%2 == 0) return false;
65   if (n > 3 && n%3 == 0) return false;
66   if (n > 5 && n%5 == 0) return false;
67   if (n > 7 && n%7 == 0) return false;
68   return true;
69 }
```

**Now check to see if input was between 1 and 100 as prompted**

**Also check to see if input was prime**

**`isPrime` function checks to see if input was prime, return boolean**

# guessprime4.c adds some defensive programming

```
40 // Ask for and read a guess
41 // function actual *definition*
42 int readGuess(void) {
43   int guess;
44
45   printf("Enter a prime between 1-100: ");
46   scanf("%d", &guess);
47
48   if (guess < 1 || guess > 100) {
49     printf("Hey! %d is out of range [1..100].\n", guess);
50   }
51
52   if (!isPrime(guess)) {
53     printf("Hey! %d is not even a prime number.\n", guess);
54   }
55
56   return guess;
57 }
58
59
60 // Check whether the number is prime
61 // (assuming n <= 100)
62 bool isPrime(const int n) {
63   if (n < 2) return false;
64   if (n > 2 && n%2 == 0) return false;
65   if (n > 3 && n%3 == 0) return false;
66   if (n > 5 && n%5 == 0) return false;
67   if (n > 7 && n%7 == 0) return false;
68   return true;
69 }
```

```
$ mygcc -o guessprime guessprime4.c
$ ./guessprime
Enter a prime between 1-100: 4
Hey! 4 is not even a prime number.
Wrong! try again
Enter a prime between 1-100: 23
Wrong! try again
Enter a prime between 1-100: 31
```

# guessprime5.c picks a random number (instead of hard coded 31)

**guessprime5.c**

Include stdlib for random functions

New method to pick a prime number

New prompt in readGuess tells user to enter 0 to quit (not shown)

```
17 #include <stdio.h>
18 #include <stdlib.h>
19 #include <stdbool.h>
20 #include <time.h>
21
22 // function prototype *declarations*
23 int pickPrime(const int max);
24 int readGuess(void);
25 bool isPrime(const int p);
26
27 // Main loop - ask for a guess, quit when it matches the answer
28 int main(int argc, char *argv[]) {
29   const int answer = pickPrime(100);
30   int guess;
31
32   guess = readGuess();
33   while (guess != 0 && guess != answer) {
34     printf("Wrong! try again\n");
35     guess = readGuess();
36   }
37
38   if (guess == 0) {
39     printf("Hah! you gave up; the answer was %d\n", answer);
40   }
41   if (guess == answer) {
42     printf("You win!\n");
43   }
44
45   return 0;  // exit status
```

# guessprime5.c picks a random number (instead of hard coded 31)

**guessprime5.c**

```
48 // pick a random prime between 1..max
49 int pickPrime(const int max) {
50   int p;
51
52   srand(time(NULL)); // seed the random-number sequence
53
54   // keep picking random numbers until we find a prime
55   for (p = 0; !isPrime(p); p = (rand() % max + 1))
56     ;
57
58   return p;
59 }
```

**New method to pick a prime number**

**Choose random seed for rand function**

**Choice based on the time the program run, so the random seed is different every time**

**rand returns integer between 0 and RAND_MAX**

**Constrain to 0… max with modulo (like CS10 hash tables)**

**To find value of RAND_MAX**

**`less /usr/include/stdlib.h`**

**Search for RAND_MAX (e.g., /RAND_MAX):**

/* The largest number rand will return (same as INT_MAX). */
#define RAND_MAX 2147483647

# Agenda

1. Hello world
2. From C source to executable
3. Building a C program step by step
4. Activity