UNIVERSITY of WASHINGTON

LECTURE 2 · CSE 333 · Summer 2026

Style; Pointers

Discussion: What is your favorite summer food?

Reminders

Assessments:

- Pre-quarter survey due tonight at 11:59pm
- Exercise 1 due tomorrow at 11:59pm
  - If you do not **commit** *and* **tag** your code, you won't get any points :(

General:

- Section tomorrow!
- Check your access to Ed, Gradescope, Canvas, and GitLab repository (!)
- Setup your lab environment
- Notify us ASAP if you have conflicts with the midterm (7/27) or final exam (8/18)
- CSE course website is back up!

AI Usage

Allowed:

I'm confused about pointers. Can you generate an example and walk me through it?

In lecture/section, we were given this example. Could you generate a diagram explaining how it works in more detail?

Not Allowed:

Copy and paste the spec into an AI tool.

Copy and paste the starter code into an AI tool (or use one built into the IDE).

What is the Nilakantha series and how would a program that approximates pi using it look like?

What edge cases should I think about for this exercise? Can you generate some tests for me?

Probably Not Allowed:

Can you help me understand this error that I'm running into while trying to compile my exercise?

Um, Actually

(CSE 333-edition)

Round 0

```c
#include <stdio.h>

int main(int argc, char* argv[]) {
  int answer = 42;
  printf("The answer is %d.\n", answer);
  return 0;
}
```

Round 0

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

int main(int argc, char* argv[]) {
  int answer = 42;
  printf("The answer is %d.\n", answer);
  return EXIT_SUCCESS;
}
```

Round 1

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

int main(int argc, char* argv[]) {
  for (int curVal = 5; curVal > 0; --curVal) {
    printf("%d...\n", curVal);
  }
  printf("Liftoff!\n");
  return EXIT_SUCCESS;
}
```

Round 1

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

int main(int argc, char* argv[]) {
  for (int cur_val = 5; cur_val > 0; --cur_val) {
    printf("%d...\n", cur_val);
  }
  printf("Liftoff!\n");
  return EXIT_SUCCESS;
}
```

Round 2

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

// Returns the square of n.
int squareOf(int n) {
  return n * n;
}

int main(int argc, char* argv[]) {
  printf("7 squared is %d.\n", squareOf(7));
  return EXIT_SUCCESS;
}
```

Round 2

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

// Returns the square of n.
int SquareOf(int n) {
  return n * n;
}

int main(int argc, char* argv[]) {
  printf("7 squared is %d.\n", SquareOf(7));
  return EXIT_SUCCESS;
}
```

Round 3

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

// Prints a friendly greeting.
void Greet() {
  char* course = "CSE 333";
  printf("Hello, %s!\n", course);
}

int main(int argc, char* argv[]) {
  Greet();
  return EXIT_SUCCESS;
}
```

Round 3

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

// Prints a friendly greeting.
void Greet(void) {
  char* course = "CSE 333";
  printf("Hello, %s!\n", course);
}

int main(int argc, char* argv[]) {
  Greet();
  return EXIT_SUCCESS;
}
```

QUICK ASIDE

printf + format specifiers

A ***format specifier*** follows this prototype: [see compatibility note below]

`%[flags][width][.precision][length]specifier`

Where the ***specifier character*** at the end is the most significant component, since it defines the type and the interpretation of its corresponding argument:

specifier	Output	Example
d ***or*** i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65

QUICK ASIDE

printf + format specifiers

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

int main(int argc, char* argv[]) {
  int count = 42;
  float pi = 3.14159f;
  double price = 19.5;
  char name[] = "Alice";
  bool is_ready = true;

  printf("Integer: %d\n", count);
  printf("Float: %f\n", pi);
  printf("Money: $%.2f\n", price);
  printf("String: %s\n", name);
  printf("Boolean: %s\n", is_ready ? "true" : "false");

  return 0;
}
```

Round 4

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

#define daysPerWeek 7

int main(int argc, char* argv[]) {
    int weeks = 3;
    printf("%d weeks is %d days.\n", weeks, weeks * daysPerWeek);
    return EXIT_SUCCESS;
}
```

Round 4

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

#define DAYS_PER_WEEK 7

int main(int argc, char* argv[]) {
    int weeks = 3;
    printf("%d weeks is %d days.\n", weeks, weeks * DAYS_PER_WEEK);
    return EXIT_SUCCESS;
}
```

Round 5

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

#define NUM_SCORES 5

// Prints every score in the array.
void PrintScores(int scores[]) {
  for (int i = 0; i < 5; ++i) {
    printf("Score %d: %d\n", i, scores[i]);
  }
}

int main(int argc, char* argv[]) {
  int scores[NUM_SCORES] = {90, 85, 100, 70, 95};
  PrintScores(scores);
  return EXIT_SUCCESS;
}
```

Round 5

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

#define NUM_SCORES 5

// Prints every score in the array.
void PrintScores(int scores[]) {
  for (int i = 0; i < NUM_SCORES; ++i) {
    printf("Score %d: %d\n", i, scores[i]);
  }
}

int main(int argc, char* argv[]) {
  int scores[NUM_SCORES] = {90, 85, 100, 70, 95};
  PrintScores(scores);
  return EXIT_SUCCESS;
}
```

Round 6

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

#define LAST_NUM 100

int main(int argc, char* argv[]) {
  int total = 0;
  for (int i = 1; i <= LAST_NUM; i++) {
    total += i;
  }
  printf("Sum is %d\n", total);
  return EXIT_SUCCESS;
}
```

Round 6

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #define LAST_NUM 100
5
6  int main(int argc, char* argv[]) {
7    int total = 0;
8    for (int i = 1; i <= LAST_NUM; ++i) {
9      total += i;
10   }
11   printf("Sum is %d\n", total);
12   return EXIT_SUCCESS;
13 }
```

Round 7

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

int main(int argc, char* argv[]) {
  int apples = 4;
  int oranges = 5;
  int bananas = 2;
  int total = apples+oranges + bananas;
  printf("You have %d pieces of fruit.\n", total);
  return EXIT_SUCCESS;
}
```

Round 7

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

int main(int argc, char* argv[]) {
  int apples = 4;
  int oranges = 5;
  int bananas = 2;
  int total = apples + oranges + bananas;
  printf("You have %d pieces of fruit.\n", total);
  return EXIT_SUCCESS;
}
```

Round 8

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

// Returns the sum of a and b.
int Add(int a, int);

int main(int argc, char* argv[]) {
  printf("Sum: %d\n", Add(3, 9));
  return EXIT_SUCCESS;
}

int Add(int a, int b) {
  return a + b;
}
```

Round 8

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

// Returns the sum of a and b.
int Add(int a, int b);

int main(int argc, char* argv[]) {
  printf("Sum: %d\n", Add(3, 9));
  return EXIT_SUCCESS;
}

int Add(int a, int b) {
  return a + b;
}
```

Round 9

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

int main(int argc, char* argv[]) {
  printf("Sum to 5 is %d\n", SumTo(5));
  return EXIT_SUCCESS;
}

// Returns the sum of 1 to max.
int SumTo(int max) {
  int sum = 0;
  for (int i = 1; i <= max; ++i) {
    sum += i;
  }
  return sum;
}
```

Round 9

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

int SumTo(int max);

int main(int argc, char* argv[]) {
  printf("Sum to 5 is %d\n", SumTo(5));
  return EXIT_SUCCESS;
}

// Returns the sum of 1 to max.
int SumTo(int max) {
  int sum = 0;
  for (int i = 1; i <= max; ++i) {
    sum += i;
  }
  return sum;
}
```

Round 10

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

// Returns whether n is even.
bool IsEven(int n) {
  if(n % 2 == 0) {
    return true;
  }
  return false;
}

int main(int argc, char* argv[]) {
  int n = 10;
  printf("%d is even: %s\n", n, IsEven(n) ? "true" : "false");
  return EXIT_SUCCESS;
}
```

Round 10

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

// Returns whether n is even.
bool IsEven(int n) {
  if (n % 2 == 0) {
    return true;
  }
  return false;
}

int main(int argc, char* argv[]) {
  int n = 10;
  printf("%d is even: %s\n", n, IsEven(n) ? "true" : "false");
  return EXIT_SUCCESS;
}
```

Round 11

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

// Prints whether n is even or odd.
void Parity(int n) {
  if (n % 2 == 0) {
    printf("even\n");
  }
  else {
    printf("odd\n");
  }
}

int main(int argc, char* argv[]) {
  Parity(7);
  return EXIT_SUCCESS;
}
```

Round 11

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

// Prints whether n is even or odd.
void Parity(int n) {
  if (n % 2 == 0) {
    printf("even\n");
  } else {
    printf("odd\n");
  }
}

int main(int argc, char* argv[]) {
  Parity(7);
  return EXIT_SUCCESS;
}
```

Round 12

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

int main(int argc, char* argv[]) {
  int score = 100;
  printf("Congratulations, you earned a perfect score of %d points today!\n", score);
  return EXIT_SUCCESS;
}
```

Round 12

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

int main(int argc, char* argv[]) {
  int score = 100;
  printf("Congratulations, you earned a perfect score of %d points "
         "today!\n", score);
  return EXIT_SUCCESS;
}
```

Round 13

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

struct Point {
	int x;
	int y;
};

int main(int argc, char* argv[]) {
	struct Point origin = {0, 0};
	printf("Origin is at (%d, %d).\n", origin.x, origin.y);
	return EXIT_SUCCESS;
}
```

Round 13

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

struct Point {
  int x;
  int y;
};

int main(int argc, char* argv[]) {
  struct Point origin = {0, 0};
  printf("Origin is at (%d, %d).\n", origin.x, origin.y);
  return EXIT_SUCCESS;
}
```

Round 14

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

// Returns the larger of a and b.
int Max(int a, int b) {
  return a > b ? a : b;
}
// Returns the smaller of a and b.
int Min(int a, int b) {
  return a < b ? a : b;
}

int main(int argc, char* argv[]) {
  printf("%d and %d\n", Max(3, 9), Min(3, 9));
  return EXIT_SUCCESS;
}
```

Round 14

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

// Returns the larger of a and b.
int Max(int a, int b) {
  return a > b ? a : b;
}

// Returns the smaller of a and b.
int Min(int a, int b) {
  return a < b ? a : b;
}

int main(int argc, char* argv[]) {
  printf("%d and %d\n", Max(3, 9), Min(3, 9));
  return EXIT_SUCCESS;
}
```

Round 15

```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {
  char name[] = "Husky";
  printf("Go %s! (%zu letters)\n", name, strlen(name));
  return EXIT_SUCCESS;
}
```

Round 15

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char* argv[]) {
  char name[] = "Husky";
  printf("Go %s! (%zu letters)\n", name, strlen(name));
  return EXIT_SUCCESS;
}
```

Round 16

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

int main(int argc, char* argv[]) {
  int lives = 3;
  int *lives_ptr = &lives;
  *lives_ptr = 5;
  printf("Lives: %d\n", lives);
  return EXIT_SUCCESS;
}
```

Round 16

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

int main(int argc, char* argv[]) {
  int lives = 3;
  int* lives_ptr = &lives;
  *lives_ptr = 5;
  printf("Lives: %d\n", lives);
  return EXIT_SUCCESS;
}
```

Round 17

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

// Swaps the values pointed to by a and b.
void Swap(int* a, int* b) {
  // save a's value
  int tmp = *a;
  *a = *b;
  //restore the saved value into b
  *b = tmp;
}

int main(int argc, char* argv[]) {
  int x = 1;
  int y = 2;
  Swap(&x, &y);
  printf("%d %d\n", x, y);
  return EXIT_SUCCESS;
}
```

Round 17

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

// Swaps the values pointed to by a and b.
void Swap(int* a, int* b) {
  // save a's value
  int tmp = *a;
  *a = *b;
  // restore the saved value into b
  *b = tmp;
}

int main(int argc, char* argv[]) {
  int x = 1;
  int y = 2;
  Swap(&x, &y);
  printf("%d %d\n", x, y);
  return EXIT_SUCCESS;
}
```

Round 18

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

int main(int argc, char* argv[]) {
  int lives = 3;  // starting lives
  int score = 0; // current score
  printf("%d lives, %d points\n", lives, score);
  return EXIT_SUCCESS;
}
```

Round 18

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

int main(int argc, char* argv[]) {
  int lives = 3;  // starting lives
  int score = 0;  // current score
  printf("%d lives, %d points\n", lives, score);
  return EXIT_SUCCESS;
}
```

Round 19

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

// Returns the next ticket number, starting from 1.
int NextTicket(void) {
  static int count = 0;
  ++count;  // add one to count
  return count;
}

int main(int argc, char* argv[]) {
  printf("First customer gets ticket %d\n", NextTicket());
  return EXIT_SUCCESS;
}
```

Round 19

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

// Returns the next ticket number, starting from 1.
int NextTicket(void) {
  static int count = 0;
  ++count;
  return count;
}

int main(int argc, char* argv[]) {
  printf("First customer gets ticket %d\n", NextTicket());
  return EXIT_SUCCESS;
}
```

Round 20

```c
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

// Divides a by b, writing the quotient to *quotient.
// Returns false if b is zero, true on success.
bool SafeDivide(int* quotient, int a, int b) {
  if (b == 0) {
    return false;
  }
  *quotient = a / b;
  return true;
}

int main(int argc, char* argv[]) {
  int result;
  if (SafeDivide(&result, 20, 4)) {
    printf("Result: %d\n", result);
  }
  return EXIT_SUCCESS;
}
```

Round 20

```
1  #include <stdbool.h>
2  #include <stdio.h>
3  #include <stdlib.h>
4
5  // Divides a by b, writing the quotient to *quotient.
6  // Returns false if b is zero, true on success.
7  bool SafeDivide(int a, int b, int* quotient) {
8    if (b == 0) {
9      return false;
10   }
11   *quotient = a / b;
12   return true;
13 }
14
15 int main(int argc, char* argv[]) {
16   int result;
17   if (SafeDivide(20, 4, &result)) {
18     printf("Result: %d\n", result);
19   }
20   return EXIT_SUCCESS;
21 }
```

Compilation

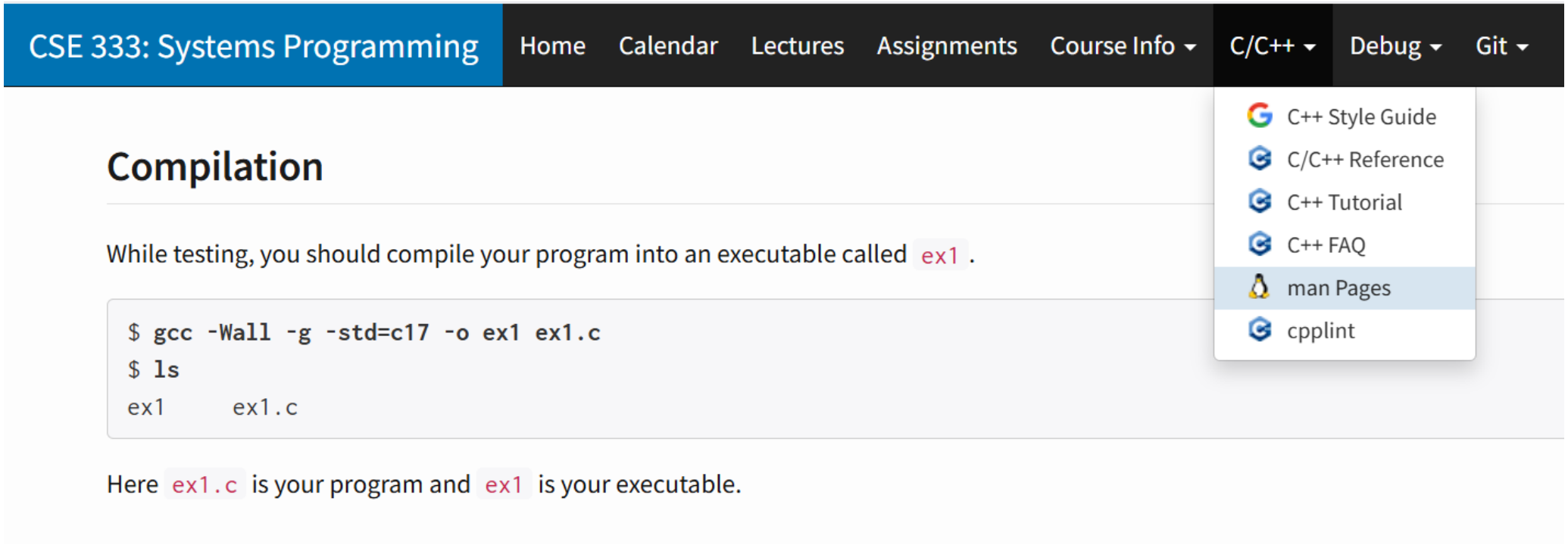
Using gcc

In the homework, we ask you to run this:

```
$ gcc -Wall -g -std=c17 -o ex1 ex1.c
```

But what does it do? There are a few ways to find out…

(1) man pages


CSE 333: Systems Programming
Home
Calendar
Lectures
Assignments
Course Info
C/C++
Debug
Git
C++ Style Guide
C/C++ Reference
C++ Tutorial
C++ FAQ
man Pages
cpplint
Compilation
While testing, you should compile your program into an executable called ex1 .
$ gcc -Wall -g -std=c17 -o ex1 ex1.c
$ ls
ex1 ex1.c
Here ex1.c is your program and ex1 is your executable.

(1) man pages

-g Produce debugging information in the operating system's native format (stabs, COFF, XCOFF, or DWARF). GDB can work with this debugging information.

On most systems that use stabs format, **-g** enables use of extra debugging information that only GDB can use; this extra information makes debugging work better in GDB but probably makes other debuggers crash or refuse to read the program. If you want to control for certain whether to generate the extra information, use **-gstabs+**, **-gstabs**, **-gxcoff+**, **-gxcoff**, or **-gvms** (see below).

-Wall

This enables all the warnings about constructions that some users consider questionable, and that are easy to avoid (or modify to prevent the warning), even in conjunction with macros. This also enables some language-specific warnings described in **C++ Dialect Options** and **Objective-C and Objective-C++ Dialect Options**.

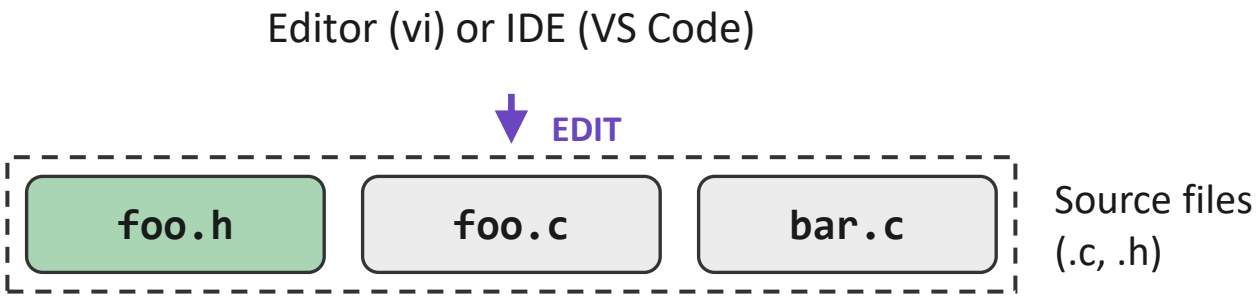
-Wall turns on the following warning flags:

-Waddress -Warray-bounds=1 (only with **-O2**) **-Wbool-compare -Wbool-operation -Wc++11-compat -Wc++14-compat -Wcatch-value**

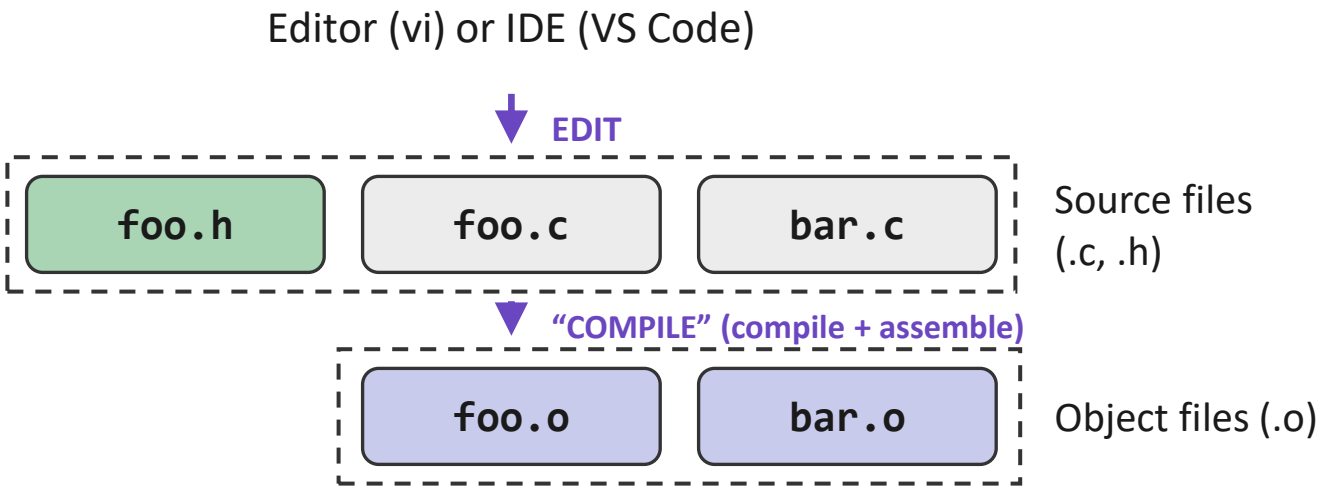
(2) Just try it!

```
$ cd ./ex1/
$ ls
ex1.c
$ gcc -Wall -g -std=c17 -o ex1 ex1.c
$ ls
ex ex1.c
$ ./ex1
some output...
```

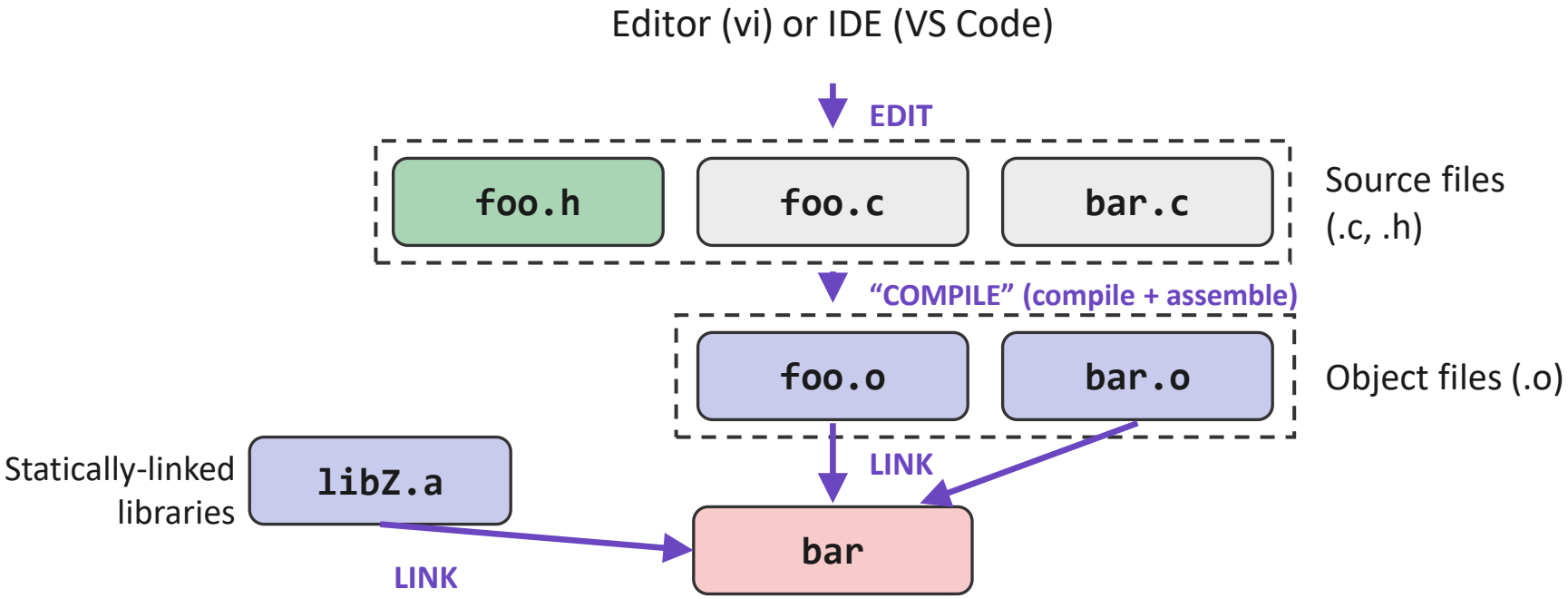
(3) Read about it



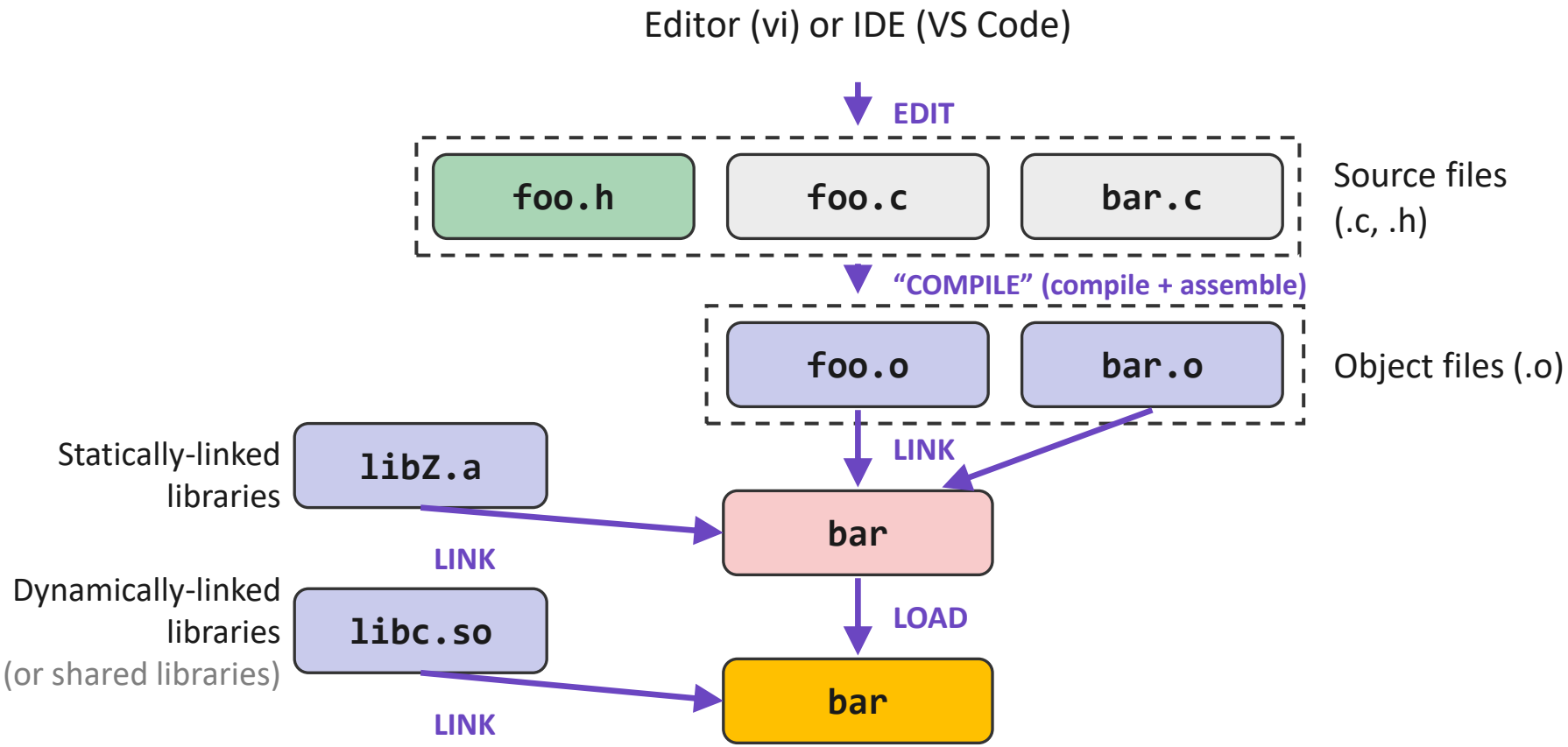
(3) Read about it



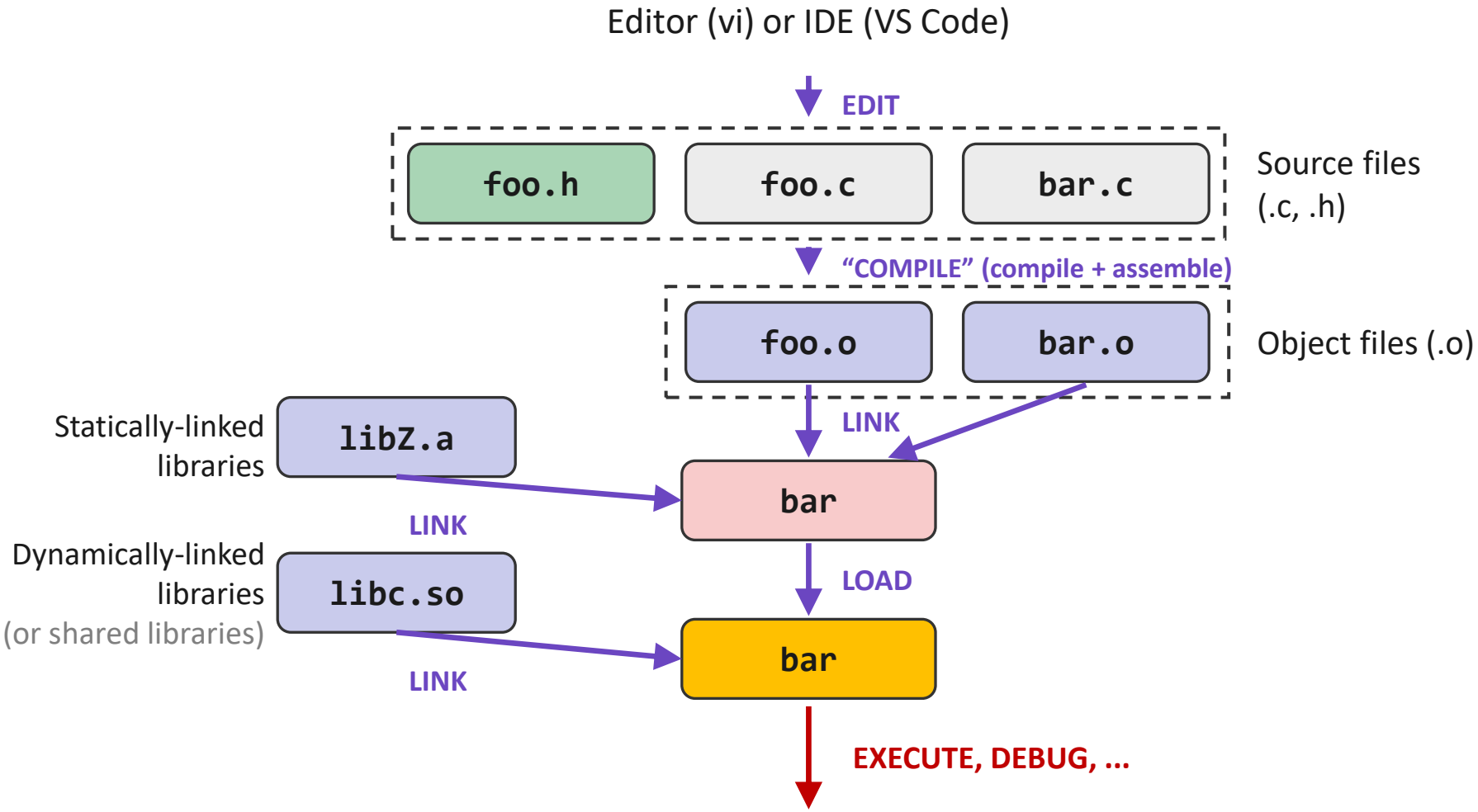
(3) Read about it



(3) Read about it



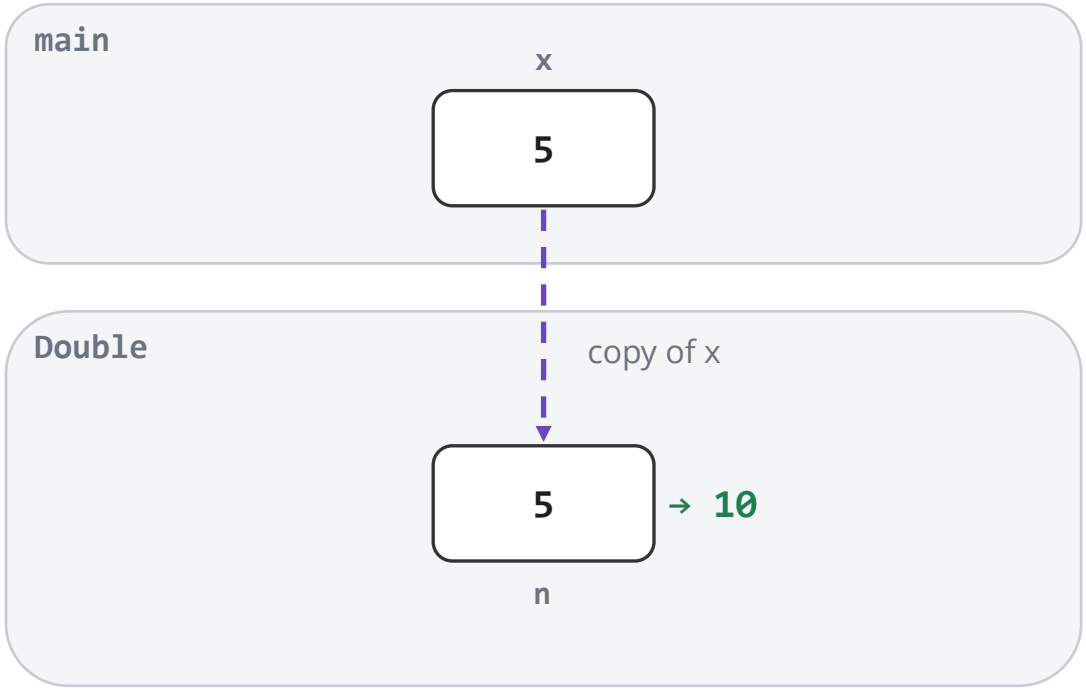
(3) Read about it



Reference vs. Value

Pass-by-Value

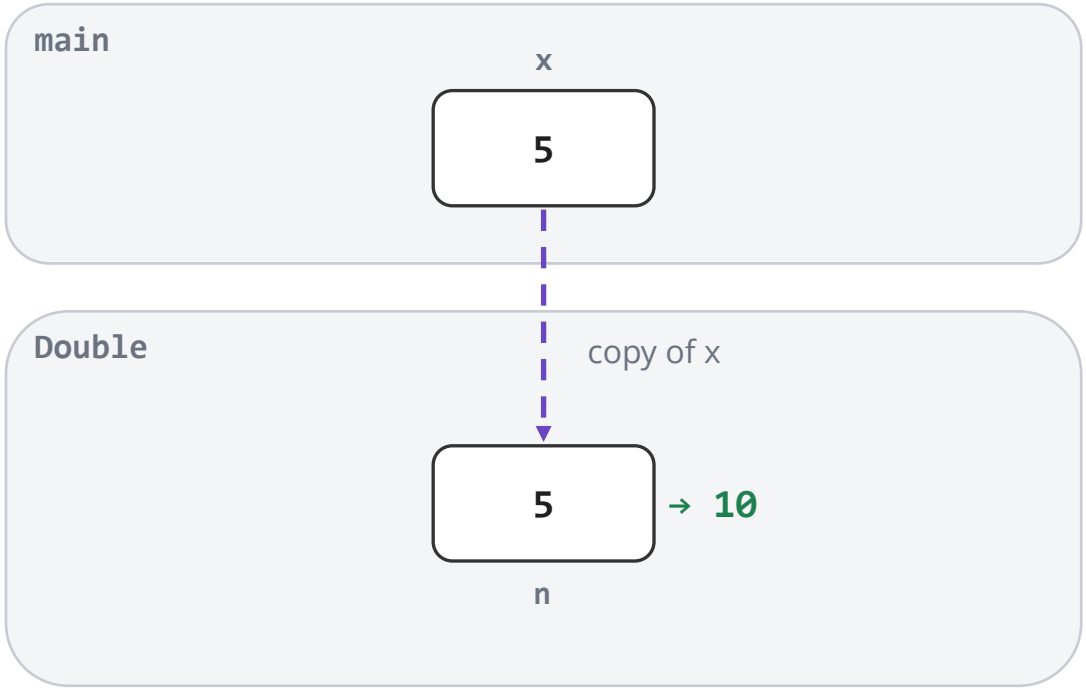
```
1  // Tries to double n.
2  void Double(int n) {
3    n = n * 2;
4  }
5
6  int main(int argc, char* argv[]) {
7    int x = 5;
8    Double(x);
9    printf("x = %d\n", x);  // 5, not 10!
10   return EXIT_SUCCESS;
11 }
```



Takeaway: C passes arguments by value (i.e. the function gets its own copy). Assigning to a parameter never changes the caller's variable.

Pass-by-Value

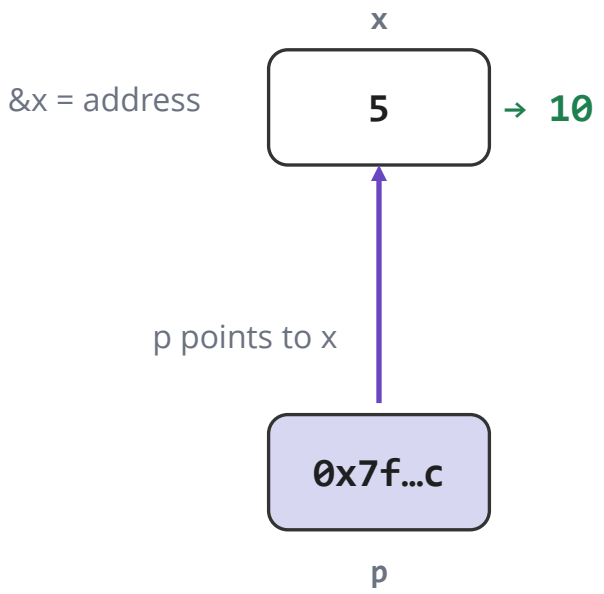
```
1  // Tries to double n.
2  void Double(int n) {
3      n = n * 2;
4  }
5
6  int main(int argc, char* argv[]) {
7      int x = 5;
8      Double(x);
9      printf("x = %d\n", x);
10     return EXIT_SUCCESS;
11 }
```



Takeaway: C passes arguments by value (i.e. the function gets its own copy). Assigning to a parameter never changes the caller's variable.

Pointers

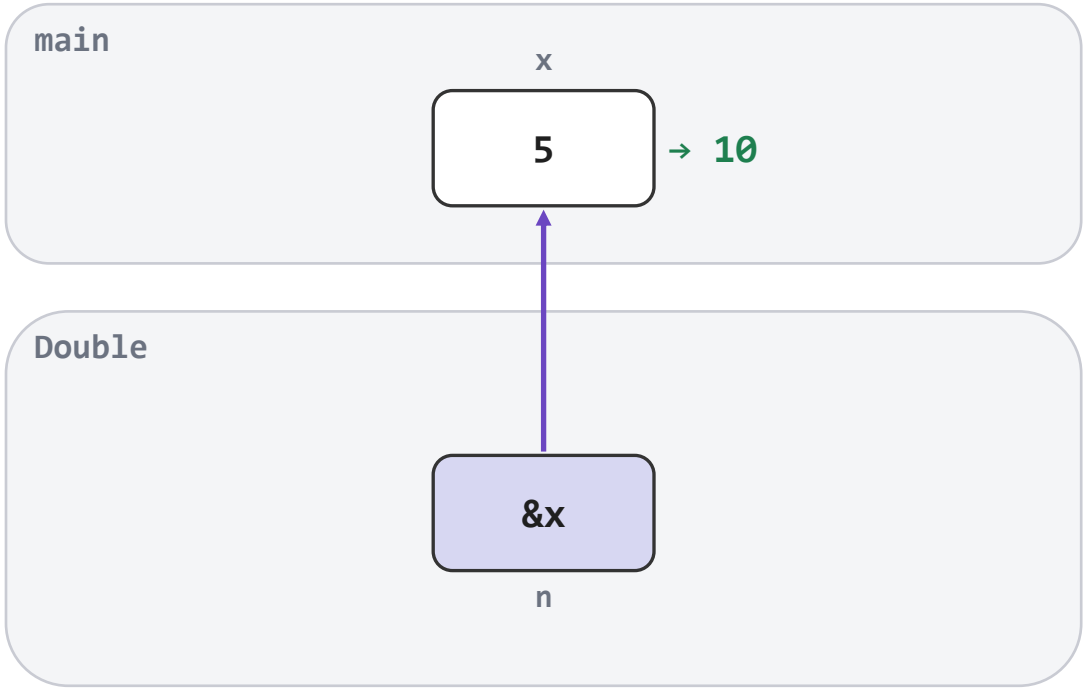
```
1  int main(int argc, char* argv[]) {
2    int x = 5;
3    int* p = &x;    // p holds x's address
4    *p = 10;        // write through p
5    printf("x = %d\n", x);  // 10
6    return EXIT_SUCCESS;
7  }
```



Takeaway: &x is the address of x. *p is the value p points to (dereference). A pointer just holds an address.

Pass-by-Reference

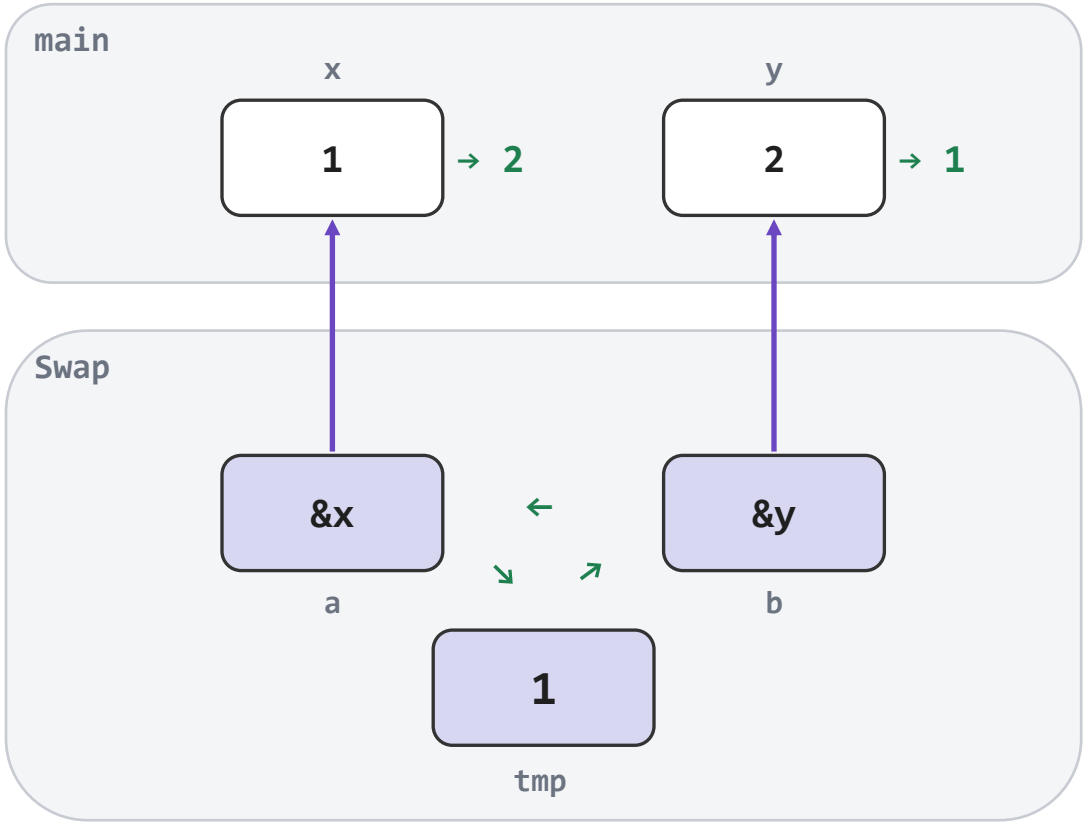
```
1  // Doubles the value n points to.
2  void Double(int* n) {
3    *n = *n * 2;
4  }
5
6  int main(int argc, char* argv[]) {
7    int x = 5;
8    Double(&x);
9    printf("x = %d\n", x);  // 10
10   return EXIT_SUCCESS;
11 }
```



Takeaway: Pass &x so the function can reach back and modify the caller's variable. (NOTE: this is still pass-by-value the value copied is now an address.)

Swap

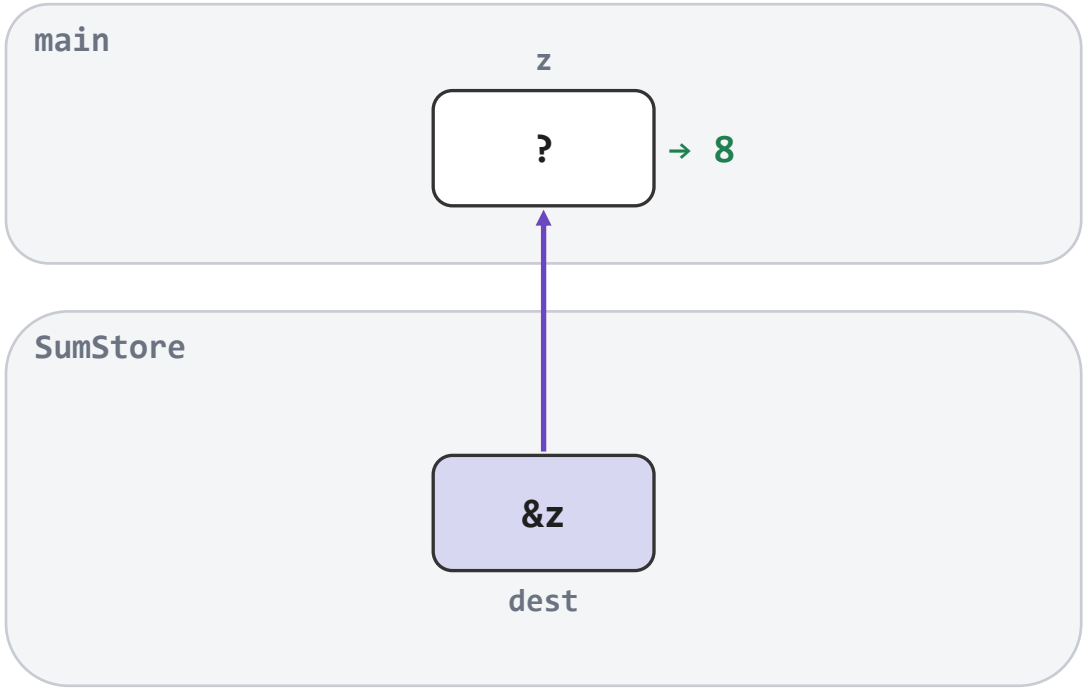
```
1  void Swap(int* a, int* b) {
2    int tmp = *a;
3    *a = *b;
4    *b = tmp;
5  }
6
7  int main(int argc, char* argv[]) {
8    int x = 1, y = 2;
9    Swap(&x, &y);
10   printf("%d %d\n", x, y);  // 2 1
11   return EXIT_SUCCESS;
12 }
```



Takeaway: Swap must take pointers. By value it would only swap copies, and the caller's x and y would be unchanged.

Output Parameters

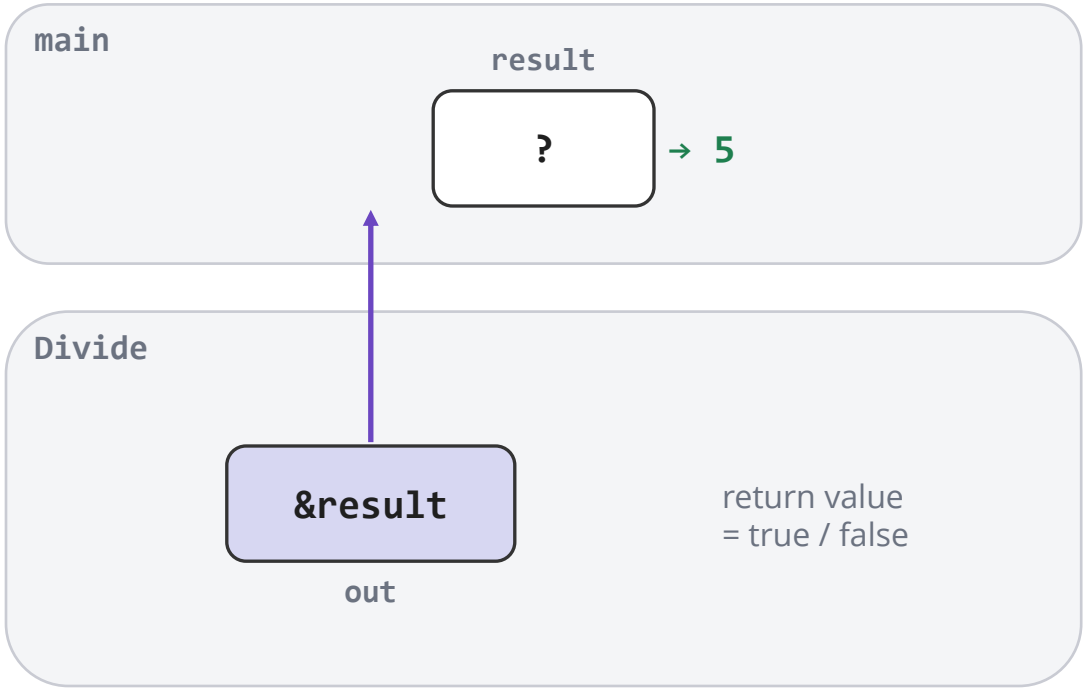
```
1  // Returns x + y through *dest.
2  void SumStore(int x, int y, int* dest) {
3    *dest = x + y;
4  }
5
6  int main(int argc, char* argv[]) {
7    int z;
8    SumStore(3, 5, &z);
9    printf("z = %d\n", z);  // 8
10   return EXIT_SUCCESS;
11 }
```



Takeaway: An output parameter is a pointer the function writes through to "hand back" a result. Allows us to return more than one value.

Output Parameters & Error Codes

```
1  // false if b == 0; else writes a/b to *out.
2  bool Divide(int a, int b, int* out) {
3    if (b == 0) {
4      return false;
5    }
6    *out = a / b;
7    return true;
8  }
9
10 int main(int argc, char* argv[]) {
11   int result;
12   if (Divide(10, 2, &result)) {
13     printf("%d\n", result);  // 5
14   }
15 }
```



Takeaway: A function returns one value. Use the return value for status / an error code and an output parameter for the result — just like strtol and sscanf.

Reminders

Assessments:

- Pre-quarter survey due tonight at 11:59pm
- Exercise 1 due tomorrow at 11:59pm
  - If you do not **commit** *and* **tag** your code, you won't get any points :(

General:

- Section tomorrow!
- Check your access to Ed, Gradescope, Canvas, and GitLab repository (!)
- Setup your lab environment
- Notify us ASAP if you have conflicts with the midterm (7/27) or final exam (8/18)
- CSE course website is back up!

Questions?

Thanks for coming - see you next lecture!