- ✓ Syntax & Data Types
- ✓ Loops & Functions
- ✓ Arrays & Strings
- ✓ Pointers & Memory
- ✓ struct · union · Linked Lists
- ✓ Preprocessor & Headers
- ✓ gcc -Wall -g -o
- ✓ Files & stdlib

C

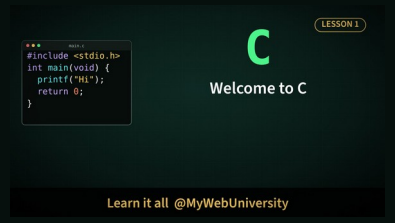
FULL COURSE

17 Lessons · Beginner to Advanced

Free · English & Dari

Learn it all @MyWebUniversity

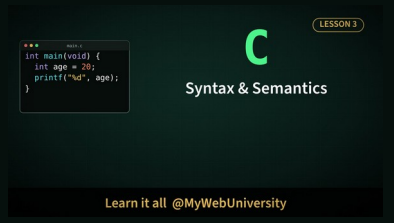
The 17 Lessons



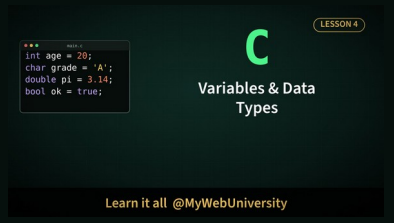
1. Welcome to C Programming



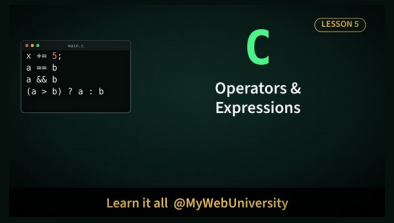
2. Compiling C with gcc



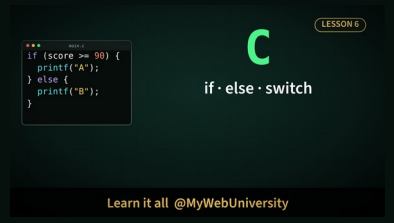
3. Program Structure, Syntax and Semantics



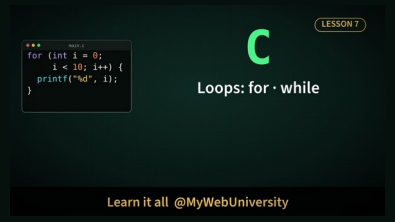
4. Variables and Data Types



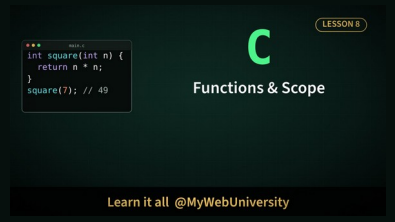
5. Operators and Expressions



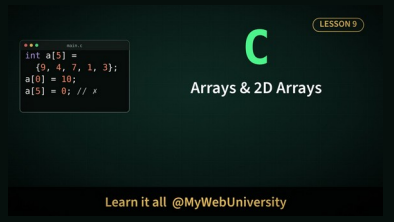
6. Branching: if, else and switch



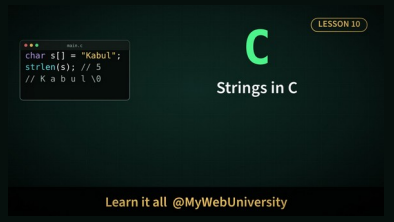
7. Loops: while, do-while and for



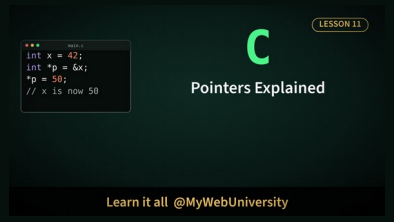
8. Functions, Prototypes and Scope



9. Arrays and Multi-Dimensional Arrays



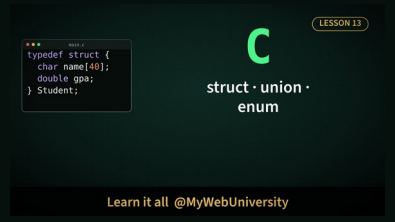
10. Strings in C



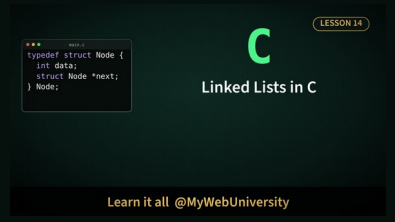
11. Pointers



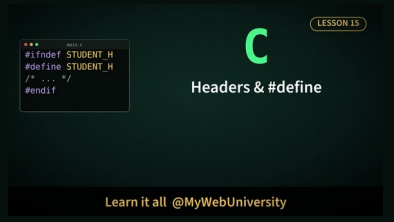
12. Dynamic Memory and Memory Leaks



13. struct, union, enum and typedef



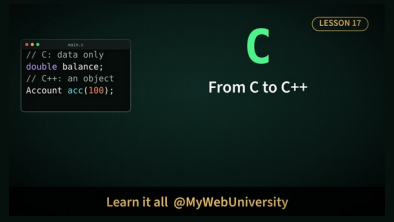
14. Linked Lists



15. The Preprocessor and Header Files



16. The Standard Library and File I/O



17. Modular Design, C's Limits, and the Road to C++



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 1

Welcome to C Programming

Why C still matters, and your first program

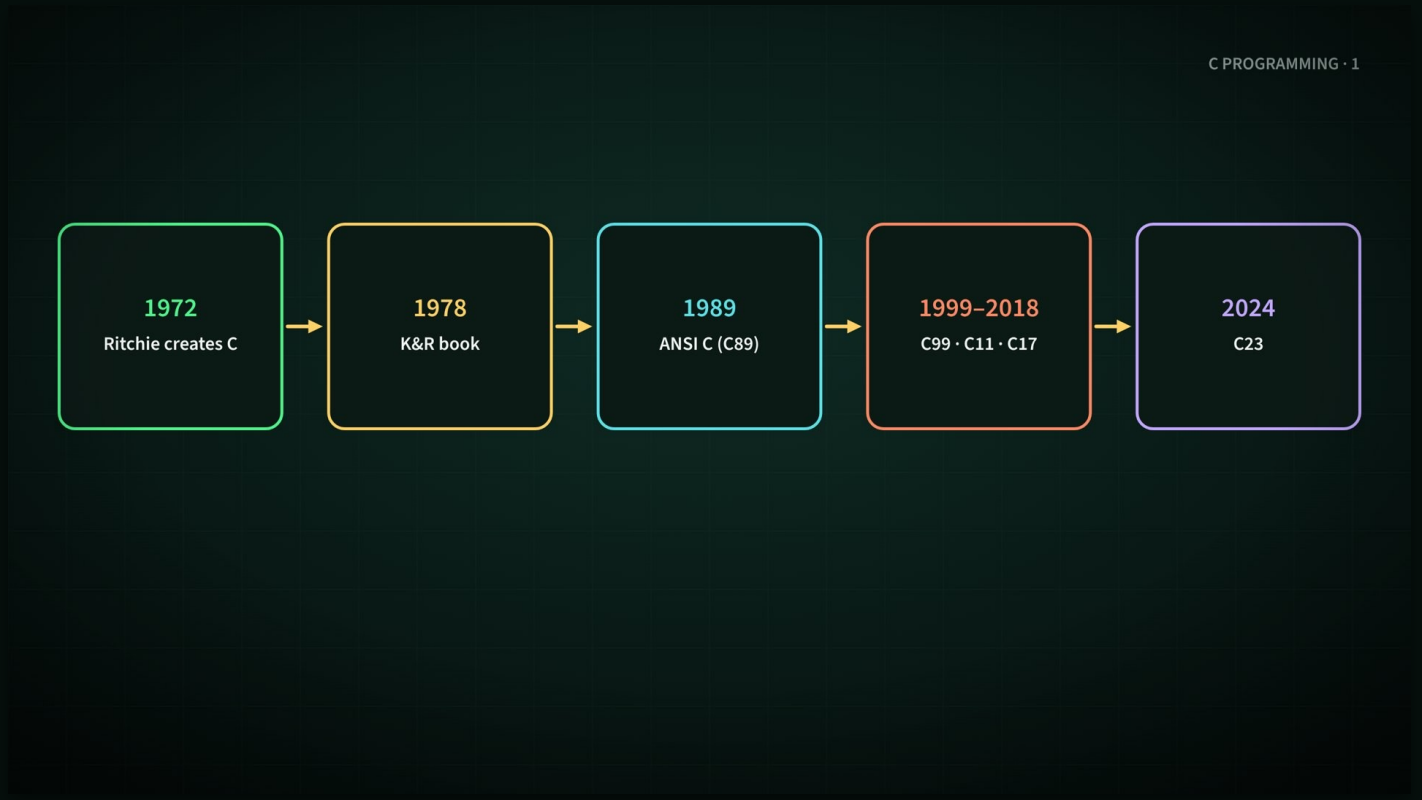
Why Learn C?



- Fast — close to the hardware
- Everywhere — Linux, Windows, databases, chips
- Foundation — C++, Java, Go, Python borrowed from C

1.01

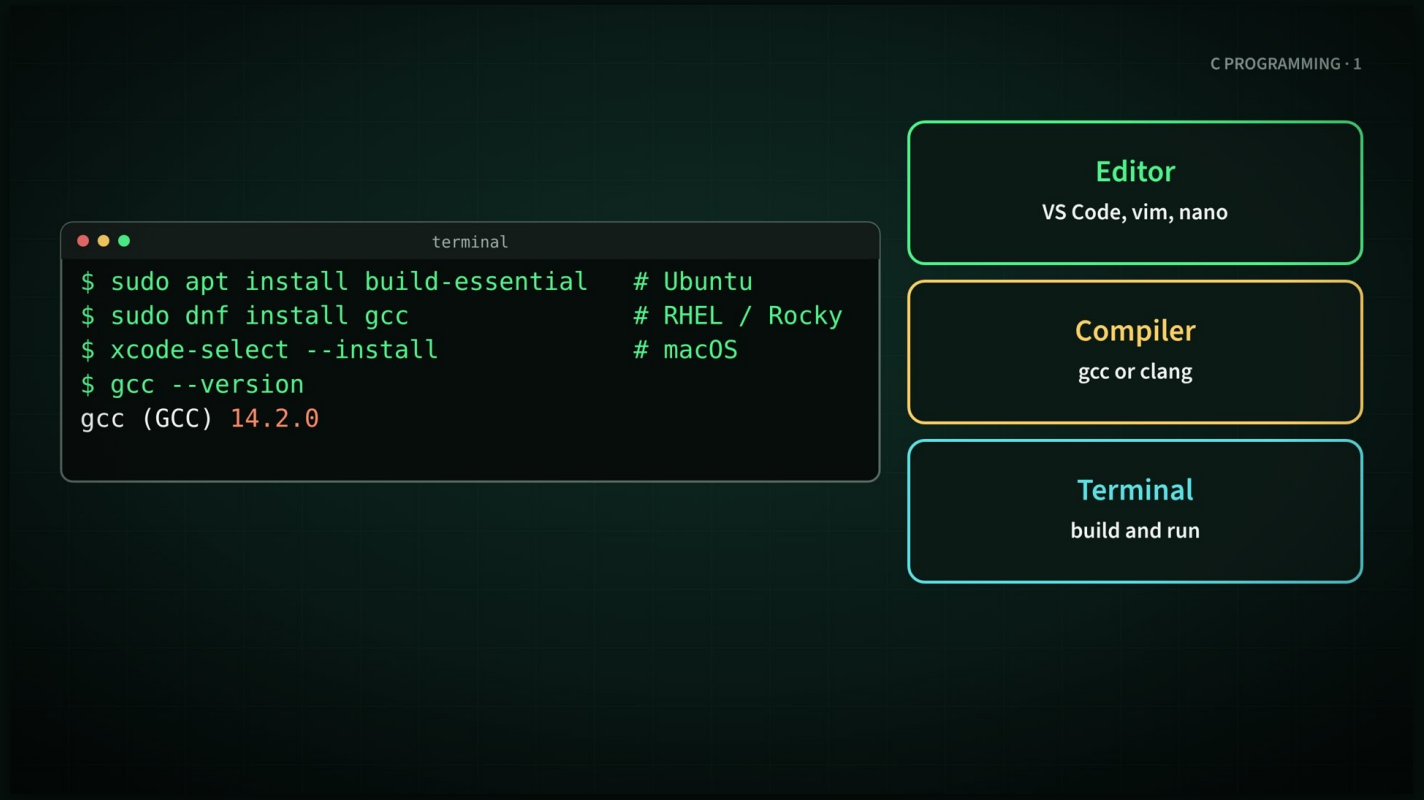
A Short History



- 1972 — Ritchie creates C
- 1978 — K&R book
- 1989 — ANSI C (C89)

1.02

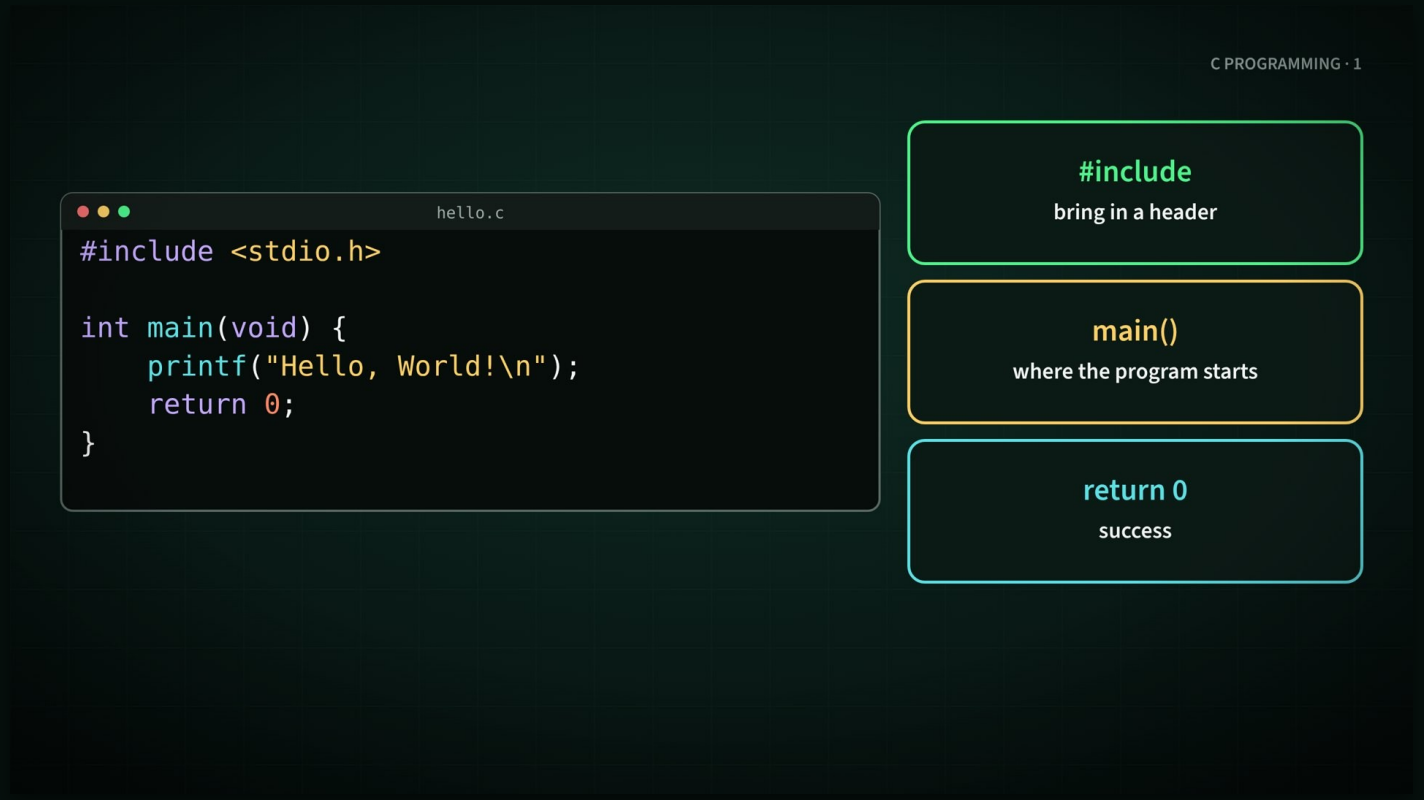
What You Need



- Editor — VS Code, vim, nano
- Compiler — gcc or clang
- Terminal — build and run

1.03

Your First Program



- #include — bring in a header
- main() — where the program starts
- return 0 — success

1.04

Compile and Run

terminal

```
$ gcc hello.c -o hello
$ ./hello
Hello, World!

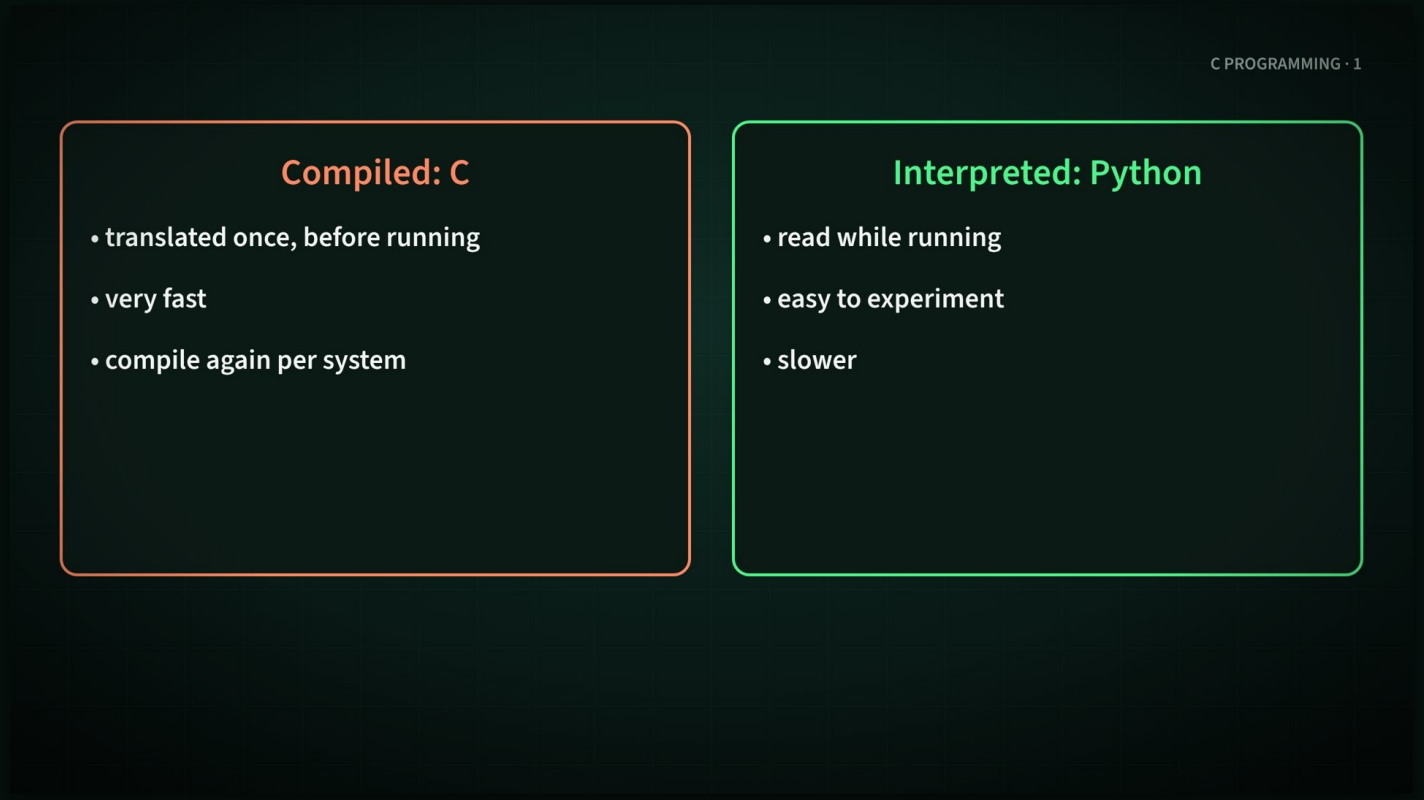
$ gcc bad.c -o bad
bad.c:4:5: error: expected ';' before 'return'
```

source code → compiler → program

- Open a terminal in the same folder and type: gcc hello.c -o hello. gcc reads your source …
- Then run it with dot slash hello. You will see Hello, World! on the screen.
- If you make a typing mistake, the compiler shows the file name, the line number and the …

1.05

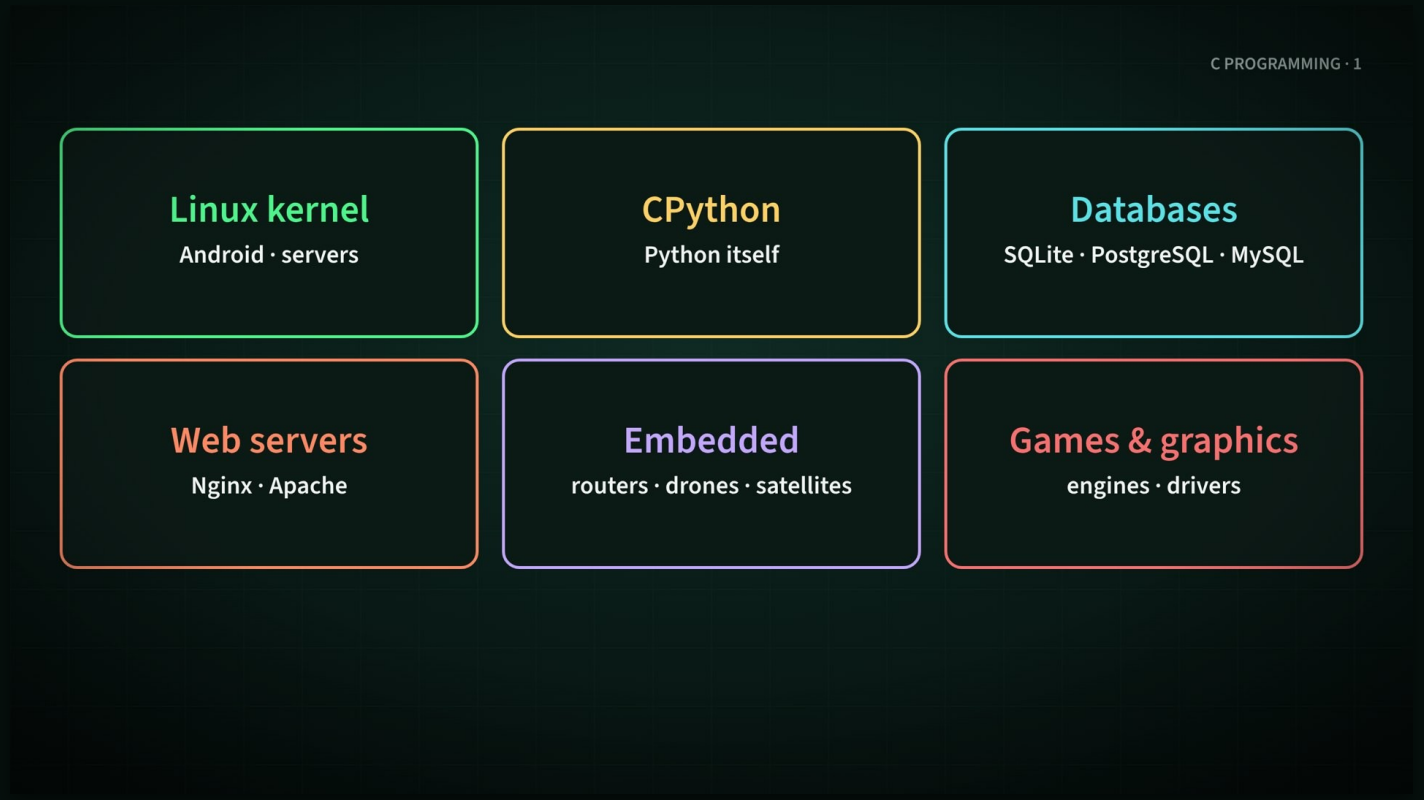
Compiled, Not Interpreted



- C is a compiled language. The whole program is translated to machine code before it runs, …
- Python is usually interpreted: it reads your code while it runs. That is easier to …
- The compiled program only runs on the kind of computer it was built for. To run on another …

1.06

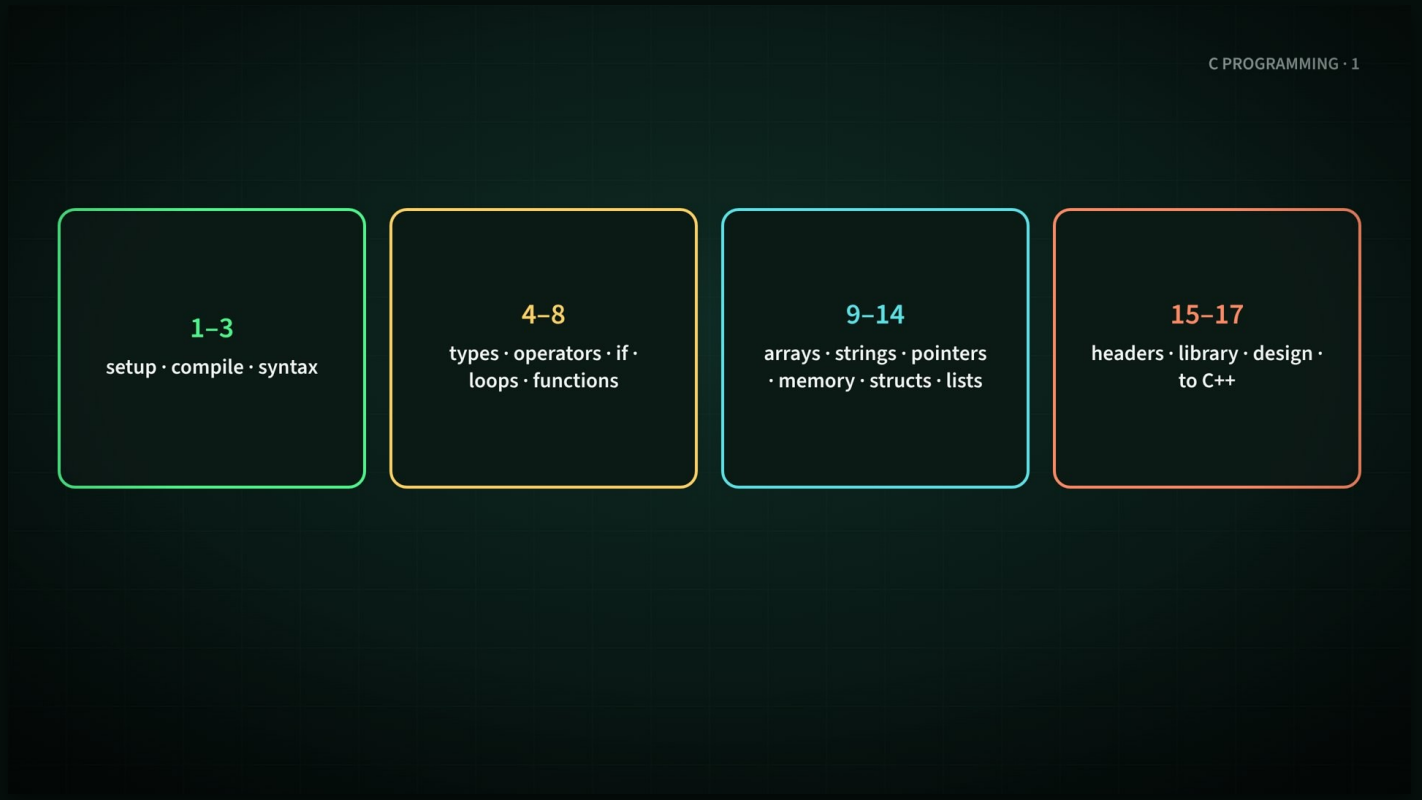
Where C Runs Today



- Linux kernel — Android · servers
- CPython — Python itself
- Databases — SQLite · PostgreSQL · MySQL

1.07

Your Roadmap



- 1–3 — setup · compile · syntax
- 4–8 — types · operators · if · loops · functions
- 9–14 — arrays · strings · pointers · memory · structs · lists

1.08



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 2

Compiling C with gcc

The four stages, the flags you need, and gdb

Four Stages



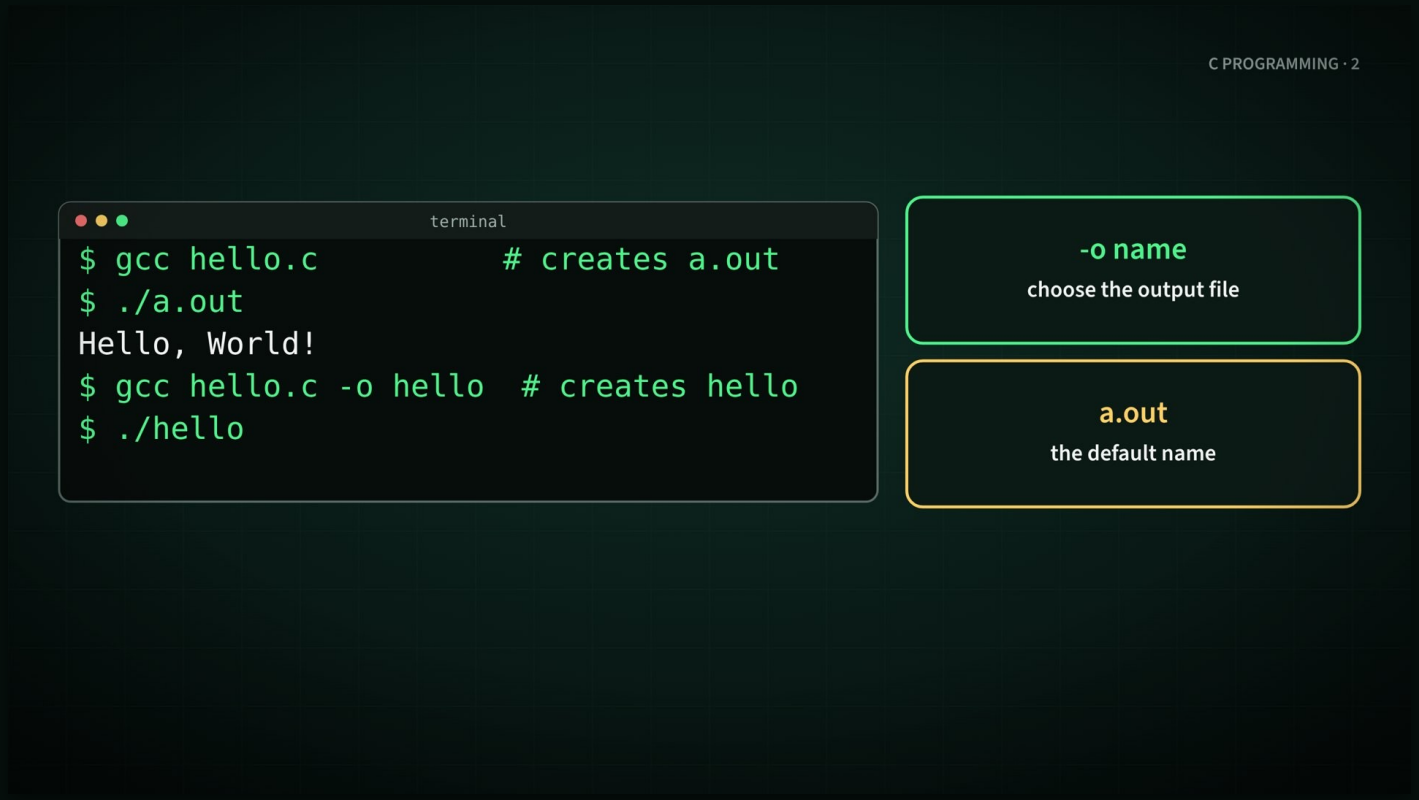
terminal

```
$ gcc -E hello.c -o hello.i   # stop after preprocessing
$ gcc -S hello.c               # stop after compiling: hello.s
$ gcc -c hello.c               # stop after assembling: hello.o
$ gcc hello.o -o hello         # link
```

- Preprocess — #include, #define → .i
- Compile — C → assembly .s
- Assemble — → object file .o

2.01

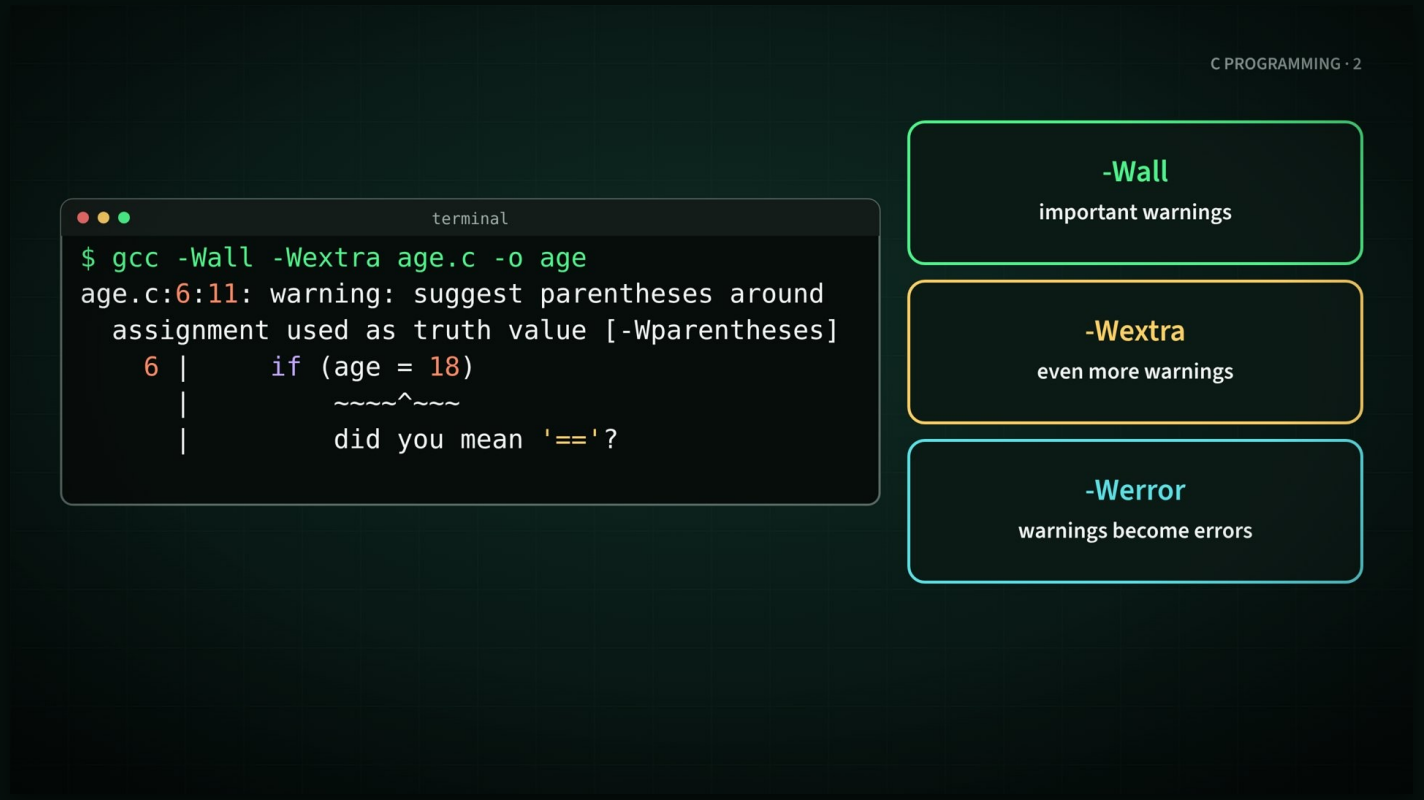
-o: Name Your Program



- -o name — choose the output file
- a.out — the default name

2.02

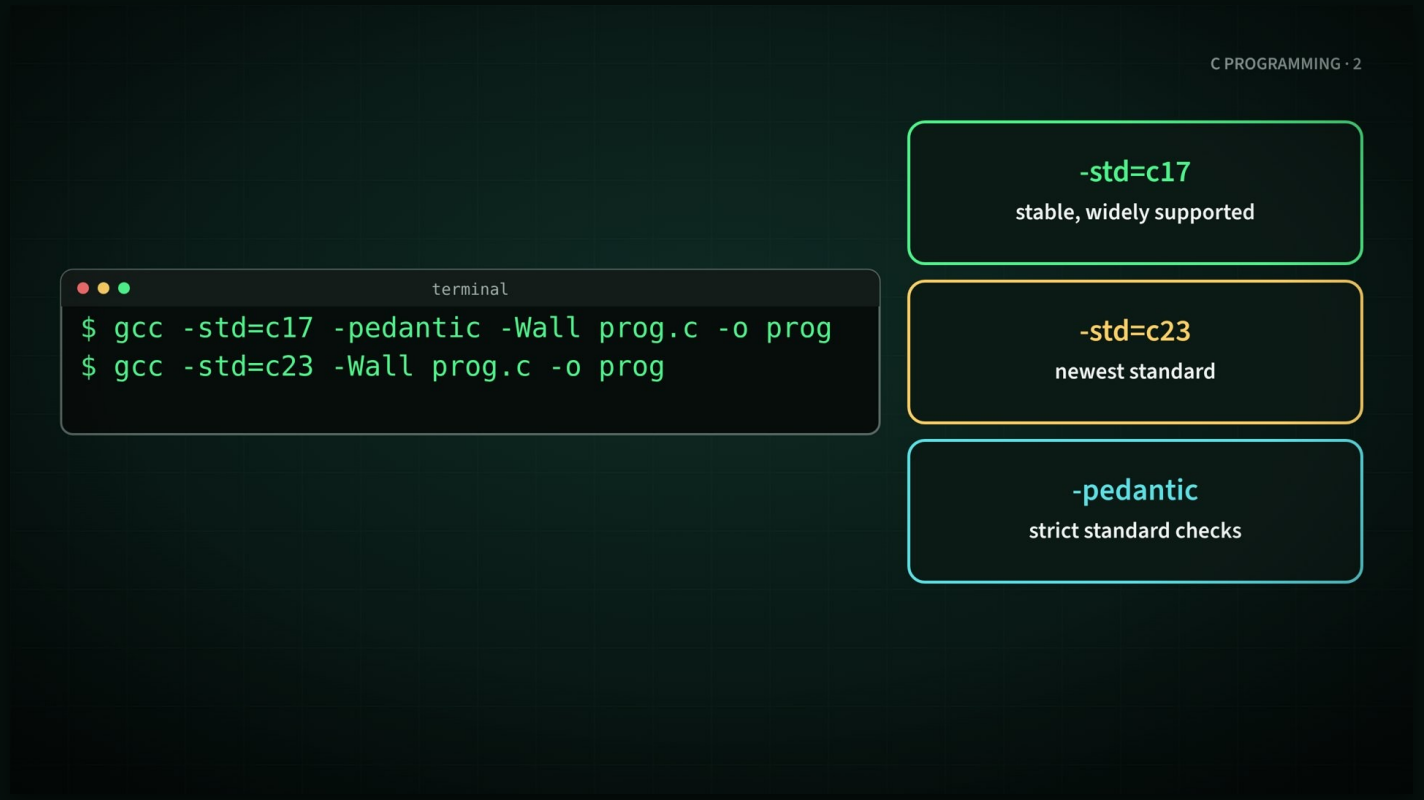
-Wall and -Wextra: Turn On Warnings



- -Wall — important warnings
- -Wextra — even more warnings
- -Werror — warnings become errors

2.03

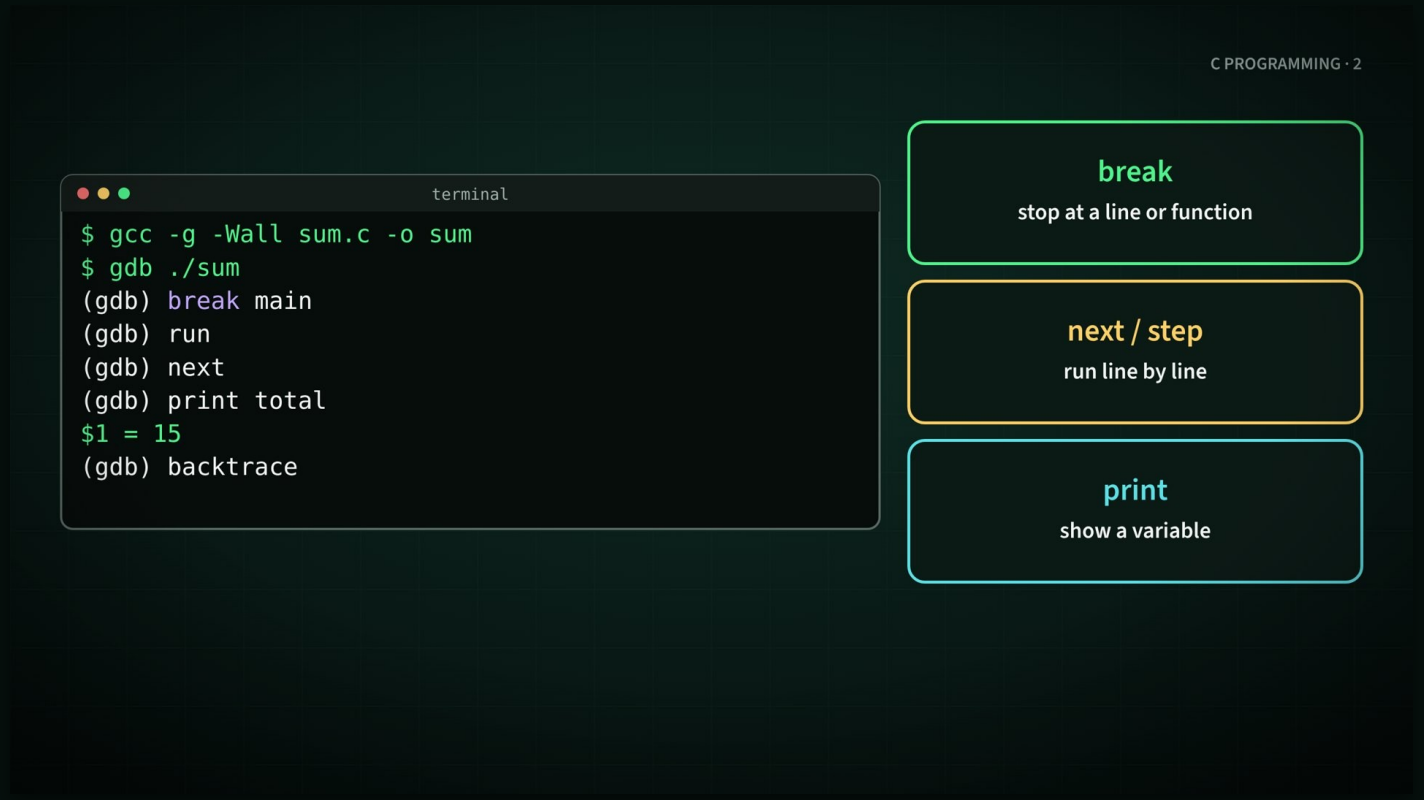
-std: Choose the C Standard



- -std=c17 — stable, widely supported
- -std=c23 — newest standard
- -pedantic — strict standard checks

2.04

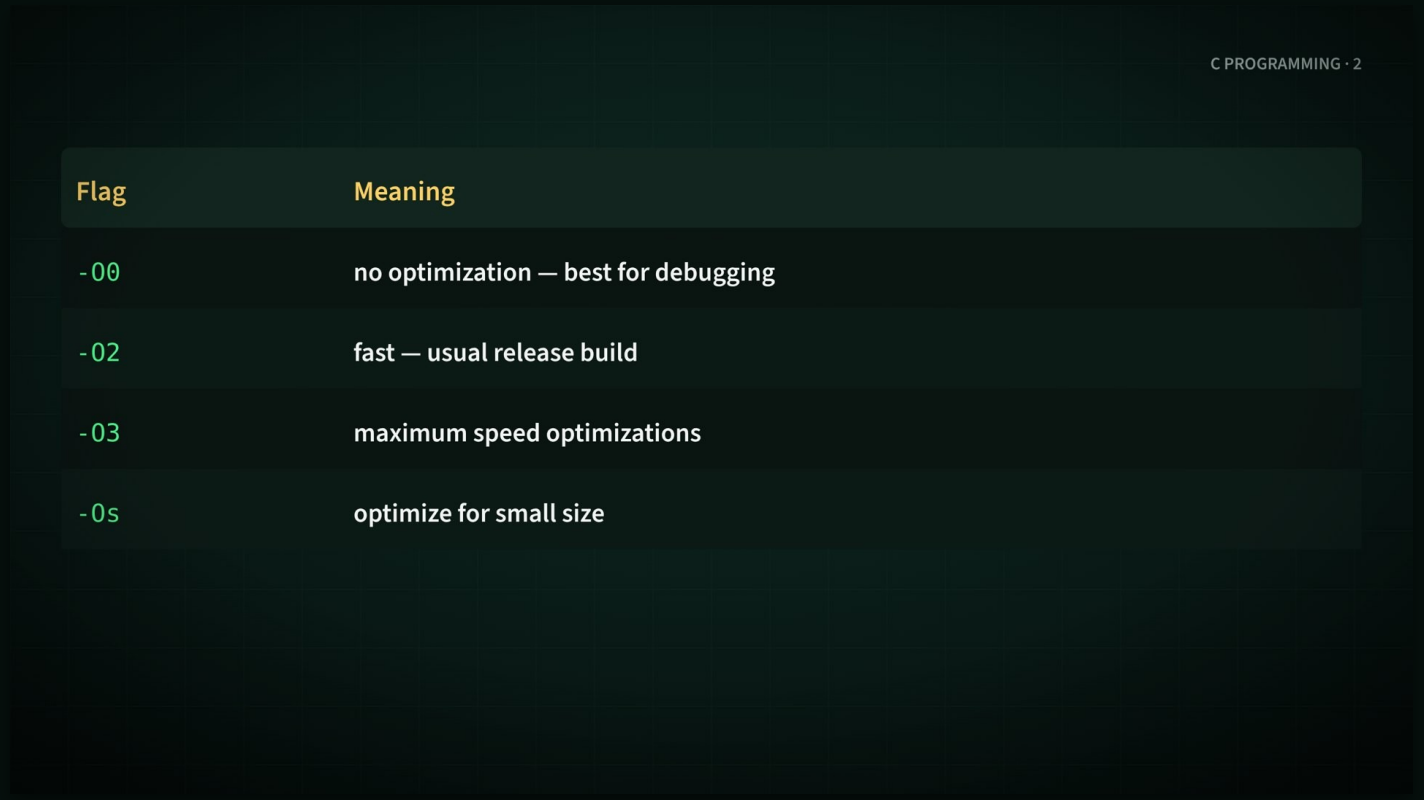
-g and gdb: Debugging



- break — stop at a line or function
- next / step — run line by line
- print — show a variable

2.05

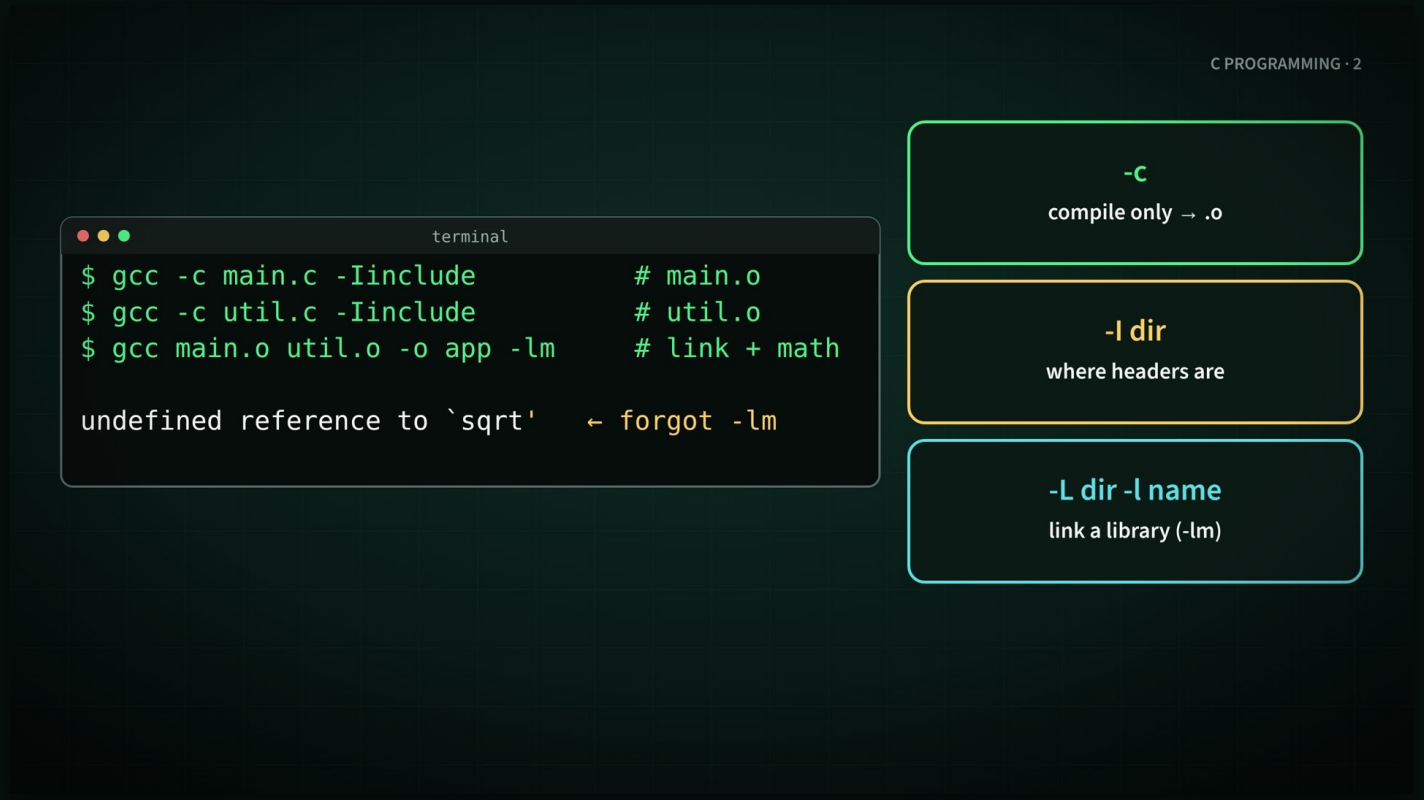
-O2: Optimization



- The -O flags tell the compiler how hard to optimize. -O0 means no optimization, which is …
- -O2 makes the program much faster, and is the usual choice for release builds. -O3 tries …
- A good habit: debug with -g -O0, and release with -O2.

2.06

-c, -I, -L and -l: Bigger Projects



- -c — compile only → .o
- -I dir — where headers are
- -L dir -l name — link a library (-lm)

2.07

A Professional Command

terminal

```
$ gcc -std=c17 -Wall -Wextra -g -fsanitize=address prog.c -o prog
$ ./prog
```

standard · warnings · debug info · memory checks

- Here is a command you can use for every exercise in this course.
- It selects C17, turns on strong warnings, adds debug information, and turns on the address …

2.08



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 3

Program Structure, Syntax and Semantics

The building blocks of every C program

Anatomy of a C Program

area.c

```c
#include <stdio.h>          // 1. preprocessor

#define PI 3.14159          // 2. a constant

double area(double r);      // 3. declaration

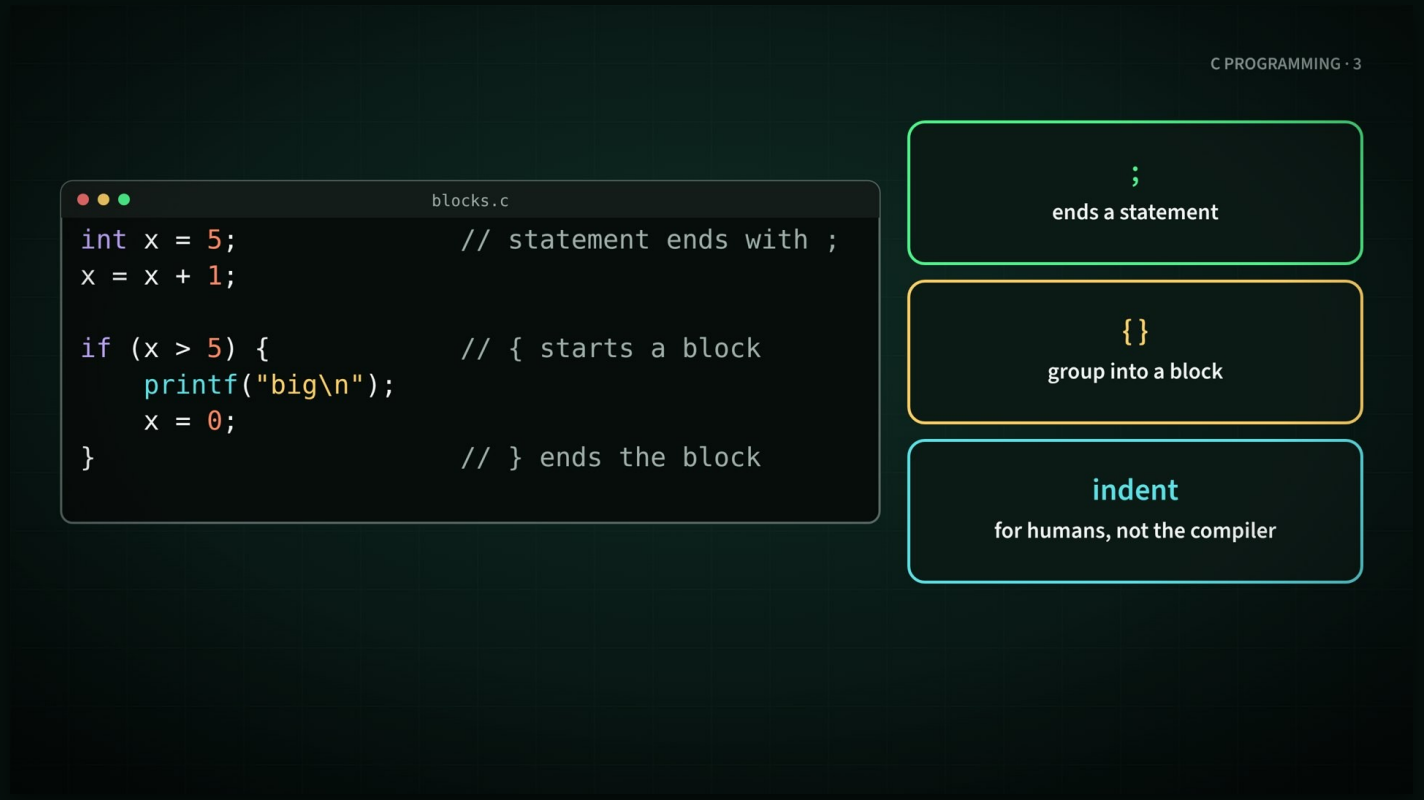
int main(void) {           // 4. main function
    double r = 2.0;
    printf("%.2f\n", area(r));
    return 0;
}

double area(double r) {    // 5. definition
    return PI * r * r;
}
```

- Every C program follows the same shape. At the top are preprocessor lines, like include. …
- One function must be called main. The operating system calls main, and when main returns, …
- Inside a function, statements run from top to bottom, one after another.

3.01

Statements and Blocks



- ; — ends a statement
- { } — group into a block
- indent — for humans, not the compiler

3.02

Tokens: The Words of C

Token type	Examples	Meaning
keyword	`int if while return`	reserved by C
identifier	`total count area`	names you choose
constant	`42 3.14 'A'`	fixed values
string literal	`"Hello"`	text in quotes
operator	`+ - * / = == ;`	actions and punctuation

- The compiler first breaks your code into tokens, the smallest meaningful pieces, like words ...
- There are keywords such as int, if and return; identifiers, the names you choose; constants ...

3.03

Rules for Names

Valid ✓

```
student_count
_temp
total2
maxSpeed
```

Invalid ×

```
2total      // starts with a digit
my-name     // minus sign
int         // a keyword
first name  // space
```

- Identifiers may use letters, digits and the underscore, but must not start with a digit, …
- C is case sensitive: total, Total and TOTAL are three different names.
- Choose meaningful names like student count instead of x. Your future self will thank you.

3.04

Comments

comments.c

```c
// one-line comment

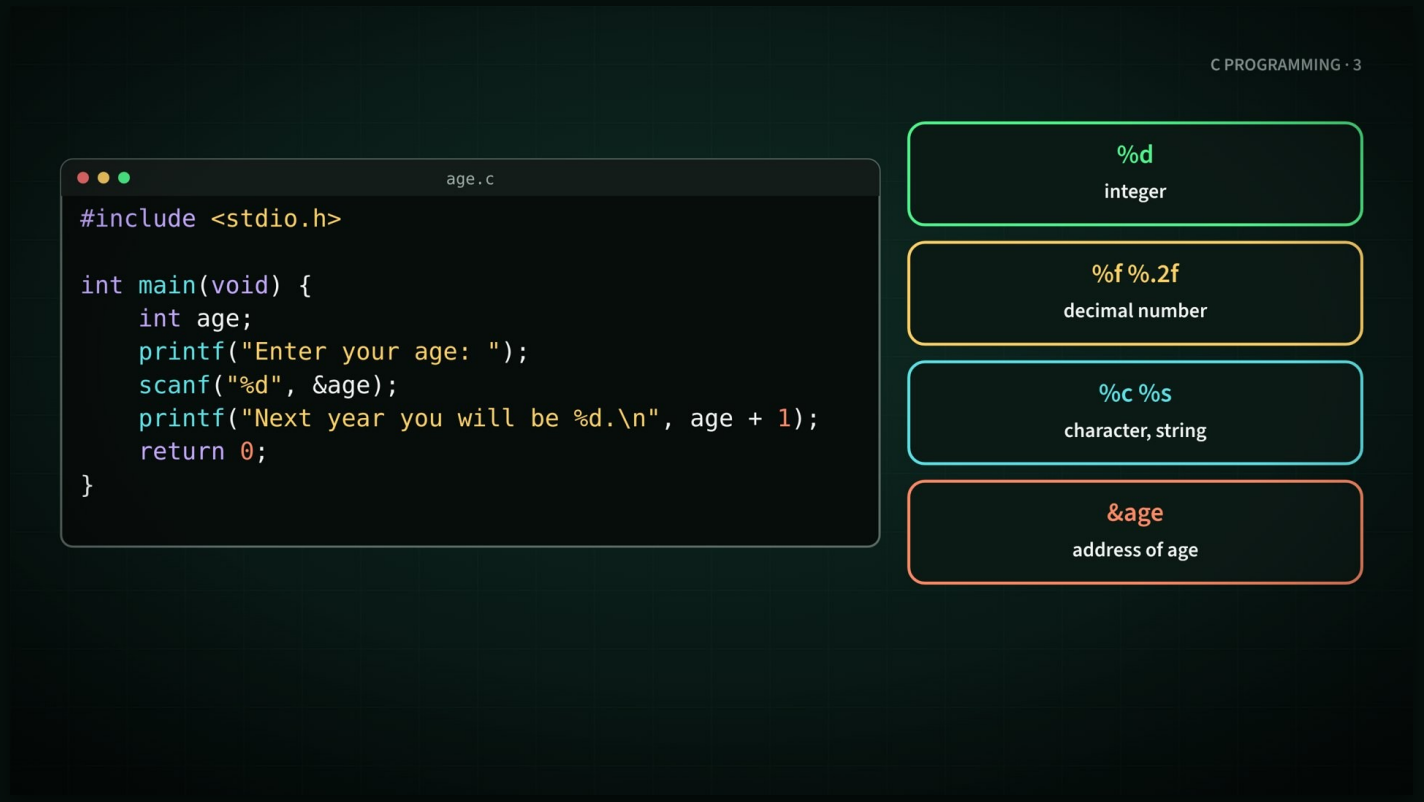
/* a comment that
   spans several lines */

int retries = 3;  // why: the network is slow at night
```

- Comments are notes for people; the compiler ignores them. Two slashes start a comment that …
- Slash star starts a comment that can span many lines, and star slash ends it.
- Good comments explain why the code does something, not just what it does.

3.05

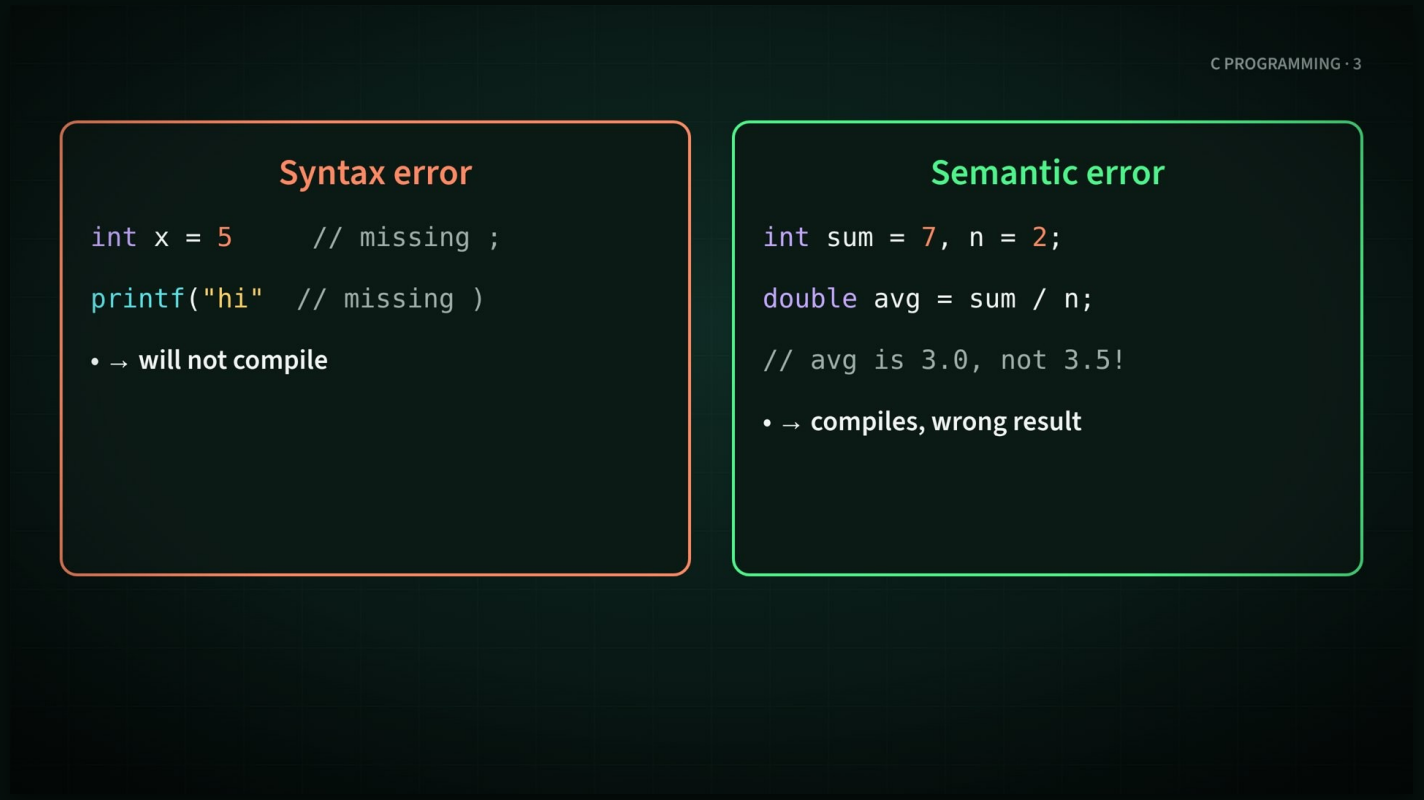
printf and scanf



- %d — integer
- %f %.2f — decimal number
- %c %s — character, string

3.06

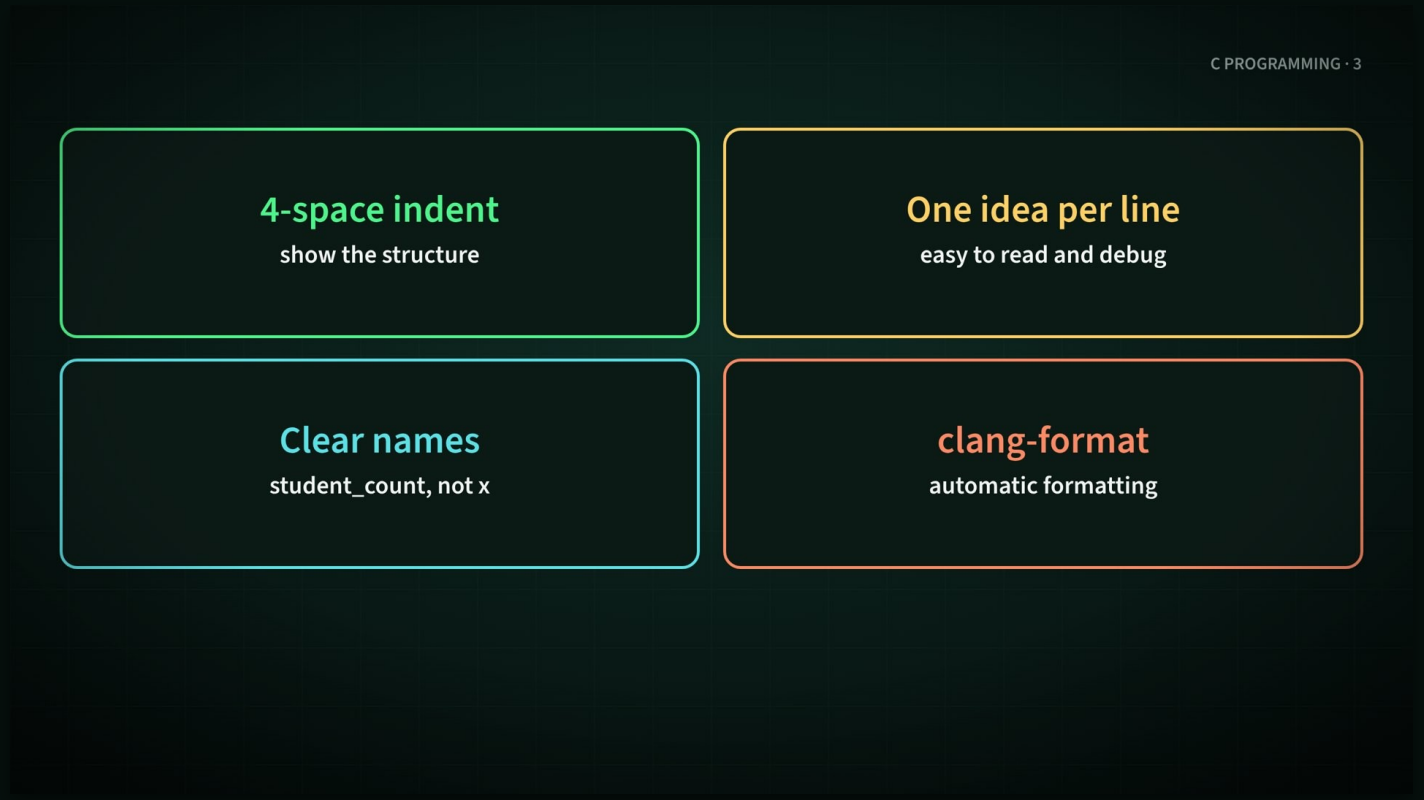
Syntax vs Semantics



- Syntax is the grammar of the language: where semicolons, braces and parentheses go. If the …
- Semantics is the meaning: what the code actually does when it runs. Code can have perfect …
- Here, both lines compile, but dividing two integers drops the decimal part, so the average …

3.07

Write Readable Code



- 4-space indent — show the structure
- One idea per line — easy to read and debug
- Clear names — student_count, not x

3.08



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 4

Variables and Data Types

int, char, float, double, sizeof, const and casting

A Variable Is a Named Box

main.c

```c
int    age   = 20;
char   grade = 'A';
double price = 9.99;
```



- A variable is a named place in memory that holds a value. In C, you must declare its type …
- The type tells the compiler how many bytes to reserve and how to interpret them. Here, age …

4.01

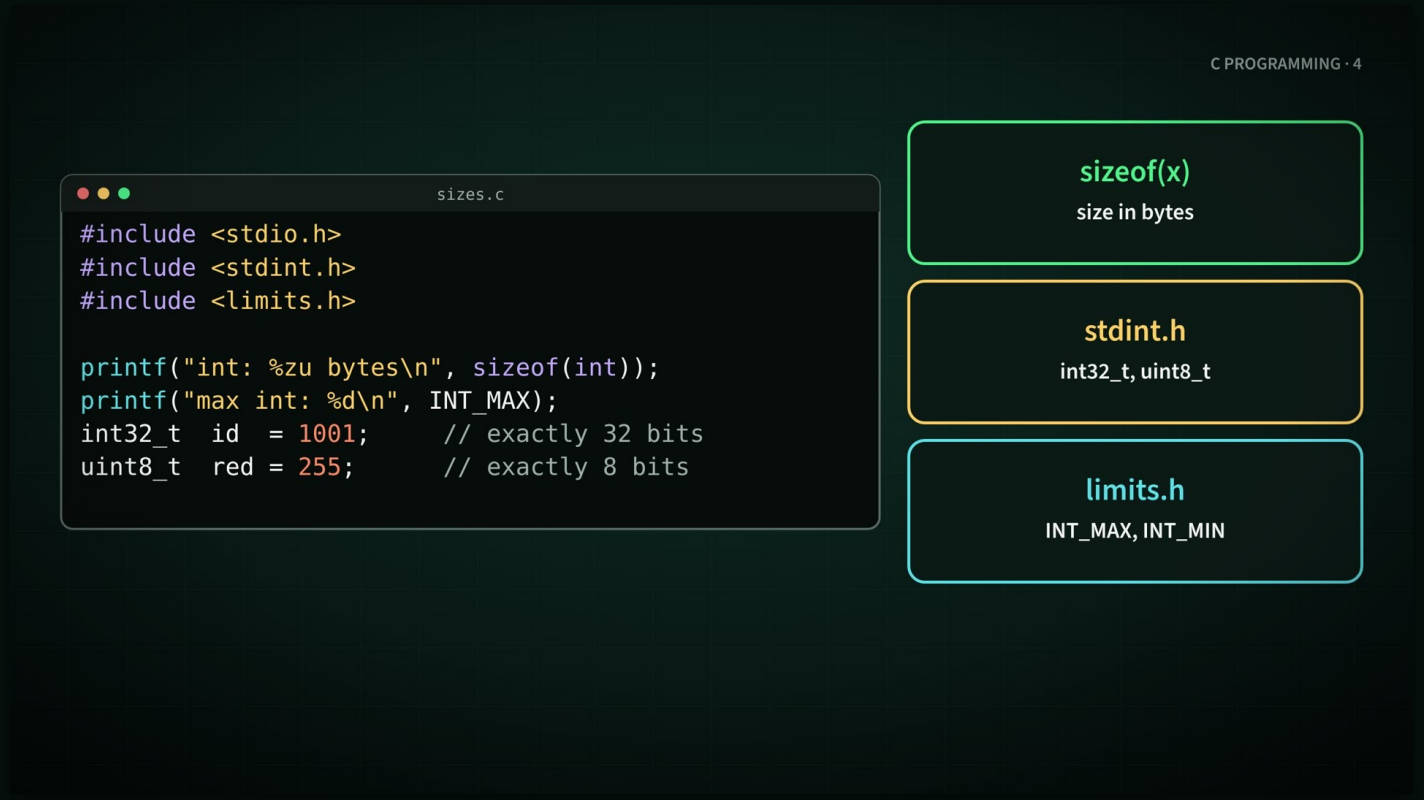
The Basic Types

Type	Typical size	Holds
`char`	1 byte	a character, -128 to 127
`int`	4 bytes	whole numbers, about ±2.1 billion
`long long`	8 bytes	very large whole numbers
`float`	4 bytes	decimals, ~7 digits
`double`	8 bytes	decimals, ~15 digits
`unsigned int`	4 bytes	0 to about 4.2 billion

- C has a few basic types. char holds one character or a small number. int holds whole …
- Modifiers change the range: short and long change the size, and unsigned removes negative …

4.02

sizeof and Exact Sizes



- sizeof(x) — size in bytes
- stdint.h — int32_t, uint8_t
- limits.h — INT_MAX, INT_MIN

4.03

Declare and Initialize



- Declaring a variable reserves memory. Initializing gives it a first value. In C, a local …
- Reading an uninitialized variable is a classic bug, so always give variables a starting value.

4.04

Constants and Literals

const.c

```c
const int DAYS_PER_WEEK = 7;
DAYS_PER_WEEK = 8;   // error: read-only variable

int    a = 42;       // int literal
double b = 3.14;     // double literal
float  c = 3.14f;    // float literal
char   d = 'A';      // char literal
char  *e = "Kabul";  // string literal
```

const
cannot change

literals
42 · 3.14 · 'A' · "text"

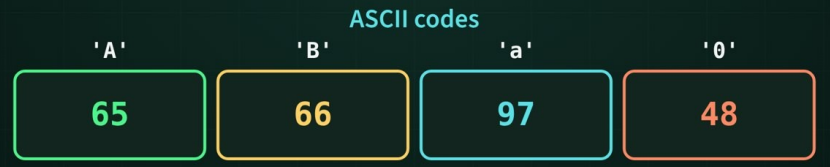
- const — cannot change
- literals — 42 · 3.14 · 'A' · "text"

4.05

Characters Are Numbers

main.c

```c
char c = 'A';
printf("%c = %d\n", c, c);      // A = 65
printf("%c\n", c + 1);          // B
```



- In C, a char is really a small number. The letter A is stored as 65 in the ASCII table, B ...
- That is why you can do math with characters: 'A' plus 1 is 'B'. Text in other languages, ...

4.06

Type Conversion and Casting

cast.c

```c
int a = 7, b = 2;

double x = a / b;           // 3.0  (integer division!)
double y = (double)a / b;   // 3.5  (cast first)

int n = (int)9.99;          // 9    (decimals cut)
unsigned char u = 300;      // 44   (overflow wraps)
```

a / b
int ÷ int → int

(double)a
cast to double

overflow
value too big for type

- a / b — int ÷ int → int
- (double)a — cast to double
- overflow — value too big for type

4.07

True and False

bool.c

```c
#include <stdbool.h>

bool is_adult = age >= 18;
if (is_adult) {
    printf("Welcome!\n");
}

if (5)   { /* runs: 5 is true  */ }
if (0)   { /* never runs       */ }
```

- Classic C has no boolean type: zero means false, and any other value means true.
- Since C99 you can include stdbool.h and write bool, true and false, which makes code …

4.08



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 5

Operators and Expressions

Arithmetic, comparison, logic, bits and precedence

Arithmetic Operators

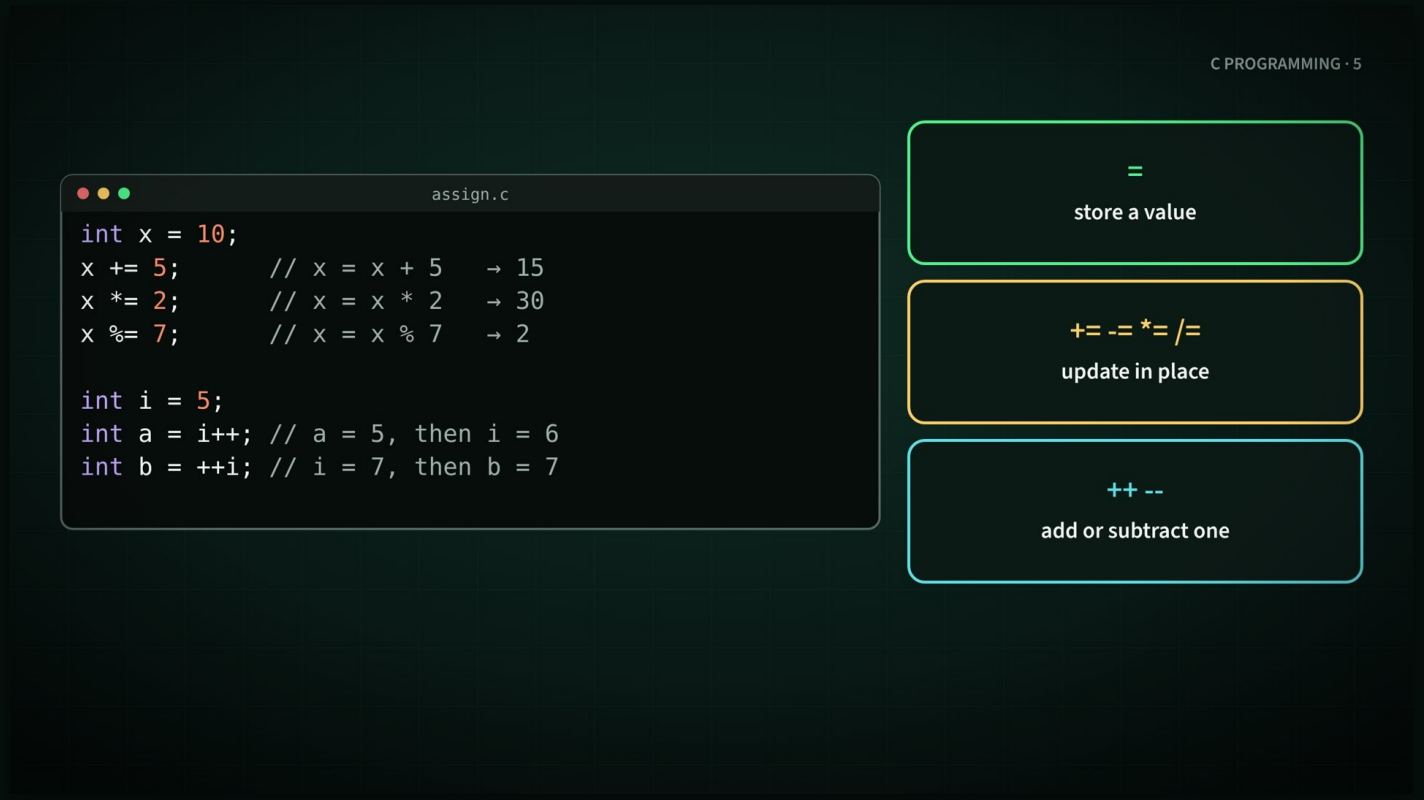
C PROGRAMMING · 5

Op	Example	Result
+	17 + 5	22
-	17 - 5	12
*	17 * 5	85
/	17 / 5	3 (integer division)
%	17 % 5	2 (remainder)

- C has the arithmetic operators you know: plus, minus, star for multiply, and slash for divide.
- The percent sign is the remainder, or modulo, operator. 17 percent 5 is 2. It is perfect …
- Remember: dividing two integers gives an integer. 17 divided by 5 is 3, not 3.4.

5.01

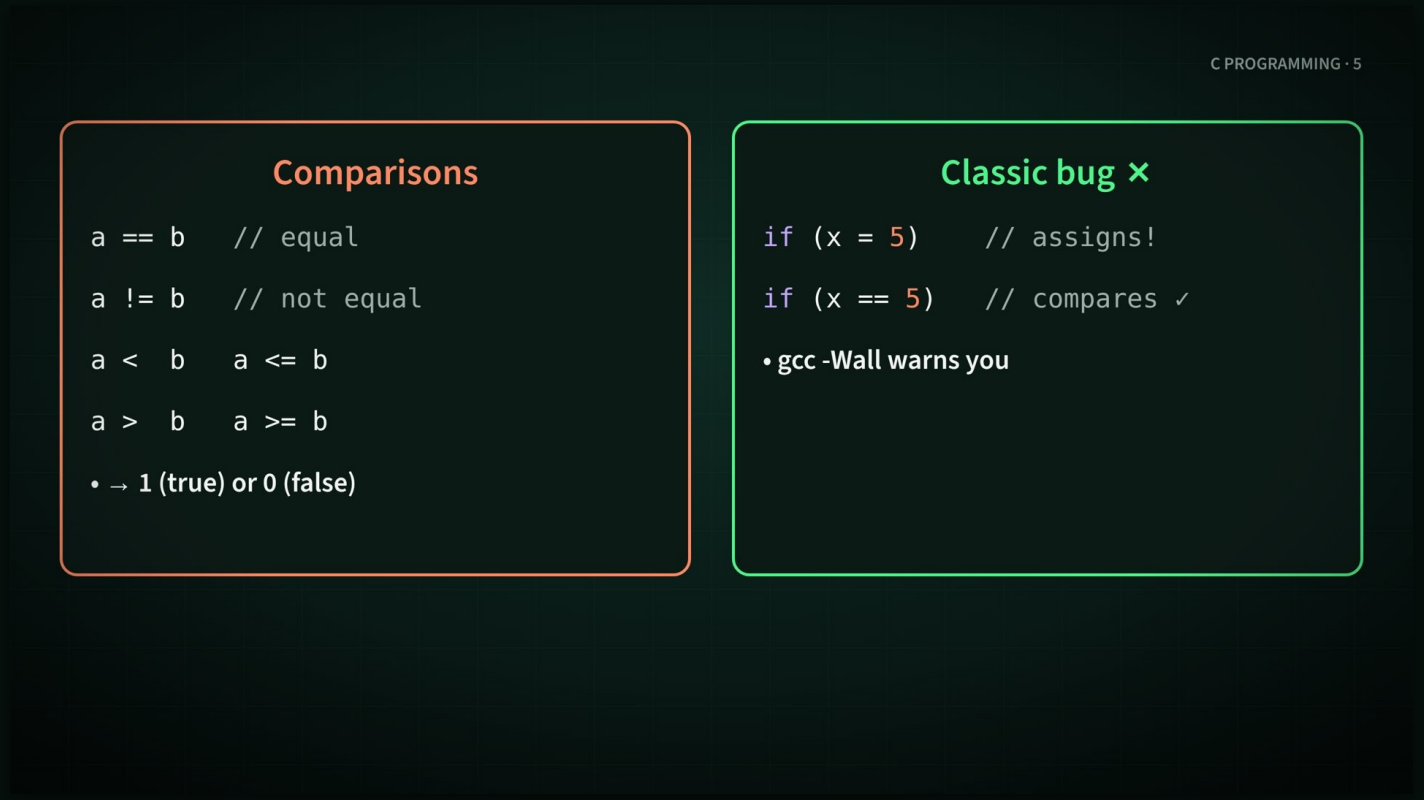
Assignment and Shortcuts



- = — store a value
- += -= *= /= — update in place
- ++ -- — add or subtract one

5.02

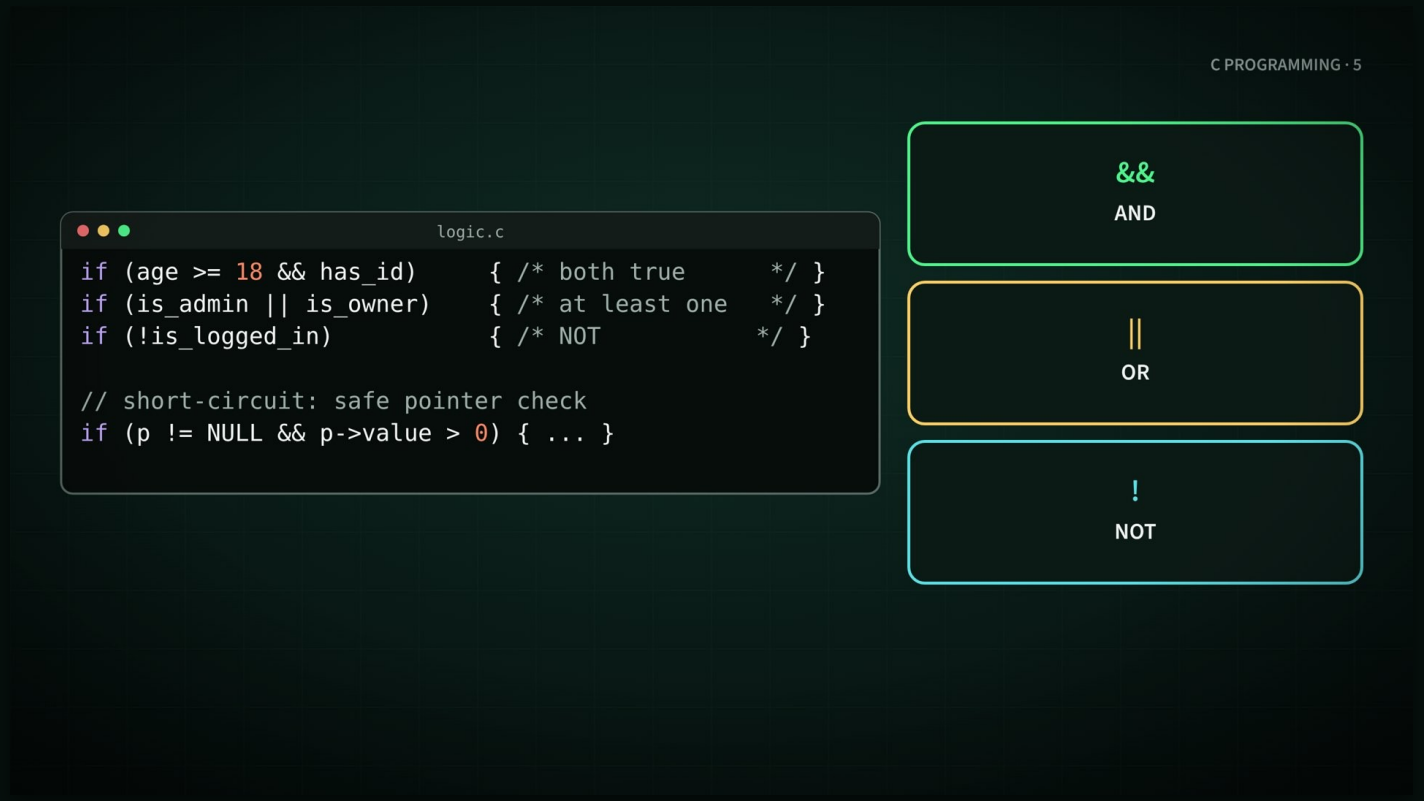
Comparison Operators



- Comparisons give one for true and zero for false. Double equals tests equality, and ...
- There are also less than, greater than, and their or-equal versions.
- The most famous C bug is writing one equals sign instead of two inside an if. That assigns ...

5.03

Logical Operators



- && — AND
- || — OR
- ! — NOT

5.04

Bitwise Operators

Op	Example	Binary	Result
&	12 & 10	1100 & 1010	8
|	12 | 10	1100 | 1010	14
^	12 ^ 10	1100 ^ 1010	6
<<	3 << 2	0011 → 1100	12
>>	12 >> 2	1100 → 0011	3

- Bitwise operators work on the individual bits of a number. Single ampersand is bitwise AND, …
- Shift left multiplies by powers of two, and shift right divides. These operators are used …

5.05

Precedence and Parentheses

precedence.c

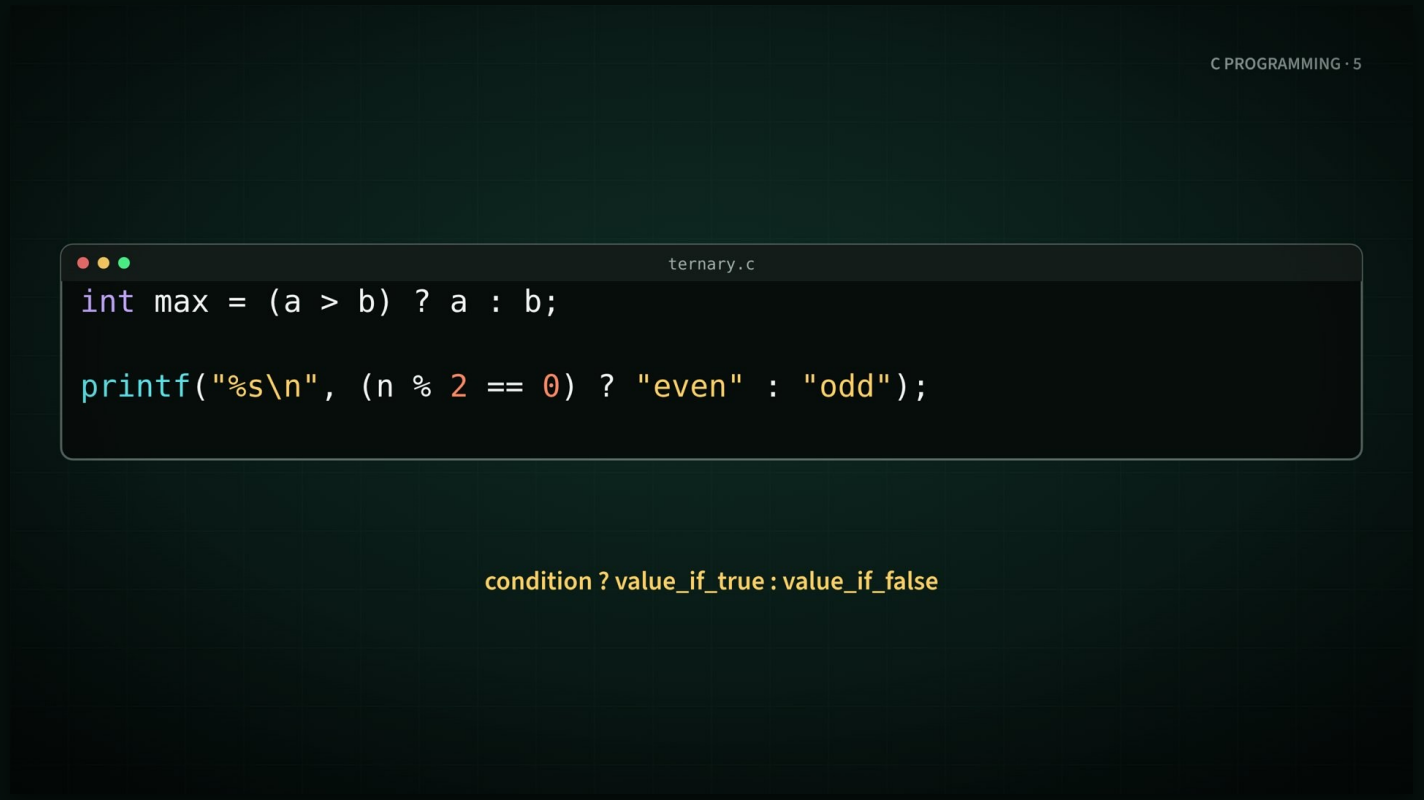
```c
int r1 = 2 + 3 * 4;         // 14, not 20
int r2 = (2 + 3) * 4;       // 20

// highest → lowest (simplified):
//  ()  ++ --  !   * / %   + -   << >>
//  < <= > >=   == !=   &&   ||   = +=
```

- When an expression has several operators, precedence decides which runs first. …
- Comparisons come before AND, and AND comes before OR. Assignment comes last.
- You do not need to memorize the whole table. When in doubt, use parentheses. They make your …

5.06

The Conditional Operator



- The conditional operator, also called the ternary operator, is a short if-else that …
- It is great for small choices like picking the larger of two numbers, but for anything …

5.07



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 6

Branching: if, else and switch

Teaching your program to make decisions

The if Statement



terminal

```
if (temperature > 30) {
    printf("It is hot. Drink water!\n");
}
```

- condition — temperature > 30
- true — run the block
- continue — next statement

6.01

if, else if, else

grade.c

```c
if (score >= 90) {
    printf("A\n");
} else if (score >= 80) {
    printf("B\n");
} else if (score >= 70) {
    printf("C\n");
} else {
    printf("Keep practicing!\n");
}
```

top to bottom
first true wins

else
catches the rest

order matters
specific first

- top to bottom — first true wins
- else — catches the rest
- order matters — specific first

6.02

Nested Conditions

Deeply nested

```
if (user) {
  if (user->active) {
    if (user->age >= 18) {
      allow();
    }
  }
}
```

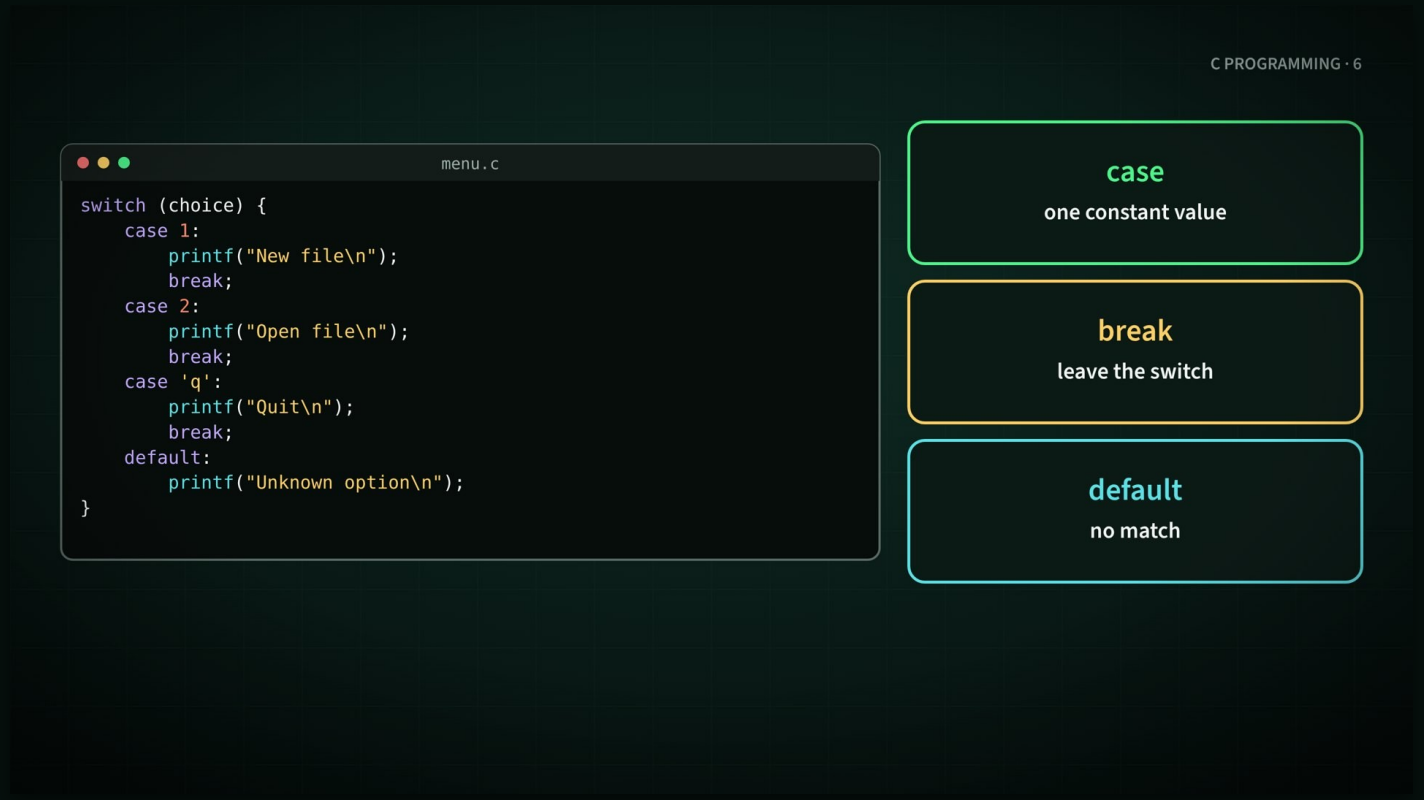
Flat and clear ✓

```
if (user &&
    user->active &&
    user->age >= 18) {
    allow();
}
```

- You can put an if inside another if. This is called nesting.
- But deep nesting quickly becomes hard to read. Often you can combine conditions with AND, ...

6.03

The switch Statement



- case — one constant value
- break — leave the switch
- default — no match

6.04

Fall-Through

days.c

```c
switch (day) {
    case 6:
    case 7:                 // grouped on purpose
        printf("Weekend\n");
        break;
    case 1: case 2: case 3: case 4: case 5:
        printf("Workday\n");
        break;
}
```

- If you forget break, execution falls through into the next case. Sometimes that is a bug, …
- When you fall through on purpose, write a comment, so the next reader knows it was intended.

6.05

Common Mistakes

pitfalls.c

```c
if (x = 0) { }            // ✗ assigns 0, never runs
if (x > 5); {             // ✗ stray ';' ends the if
    run();                 //   this always runs!
}
if (0.1 + 0.2 == 0.3) { } // ✗ false!
if (fabs(a - b) < 1e-9) { } // ✓ compare with tolerance
```

- Watch for these classic mistakes: using one equals sign instead of two; putting a semicolon …
- Decimal numbers are stored approximately, so 0.1 plus 0.2 is not exactly 0.3. Compare them …

6.06

Example: Leap Year

leap.c

```c
#include <stdio.h>

int main(void) {
    int year;
    printf("Year: ");
    scanf("%d", &year);
    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0) {
        printf("%d is a leap year\n", year);
    } else {
        printf("%d is not a leap year\n", year);
    }
    return 0;
}
```

- Let us combine everything. A year is a leap year if it is divisible by 4, except years …
- This one condition uses modulo, comparison, AND, OR and parentheses. Try it with 2024, 1900 …

6.07



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 7

Loops: while, do-while and for

Repeating work without repeating code

The while Loop

countdown.c

```c
int n = 5;
while (n > 0) {          // checked before each round
    printf("%d ", n);
    n--;                 // moves toward the end
}
printf("Liftoff!\n");
// 5 4 3 2 1 Liftoff!
```

condition first
may run zero times

change something
or it never stops

- condition first — may run zero times
- change something — or it never stops

7.01

The do-while Loop

menu.c

```c
int choice;
do {
    printf("1) Start  2) Help  0) Exit: ");
    scanf("%d", &choice);
} while (choice < 0 || choice > 2);   // note the ;
```

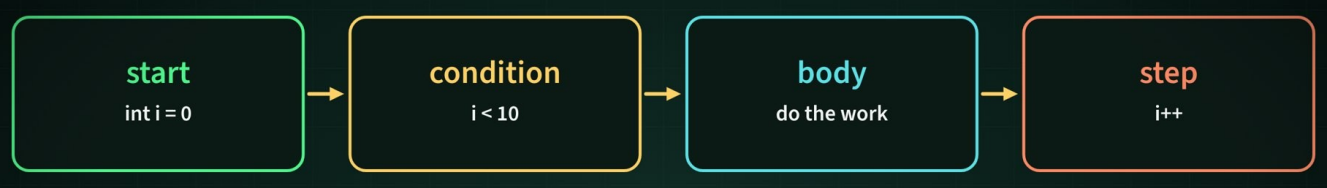
body first
runs at least once

menus
input validation

- body first — runs at least once
- menus — input validation

7.02

The for Loop



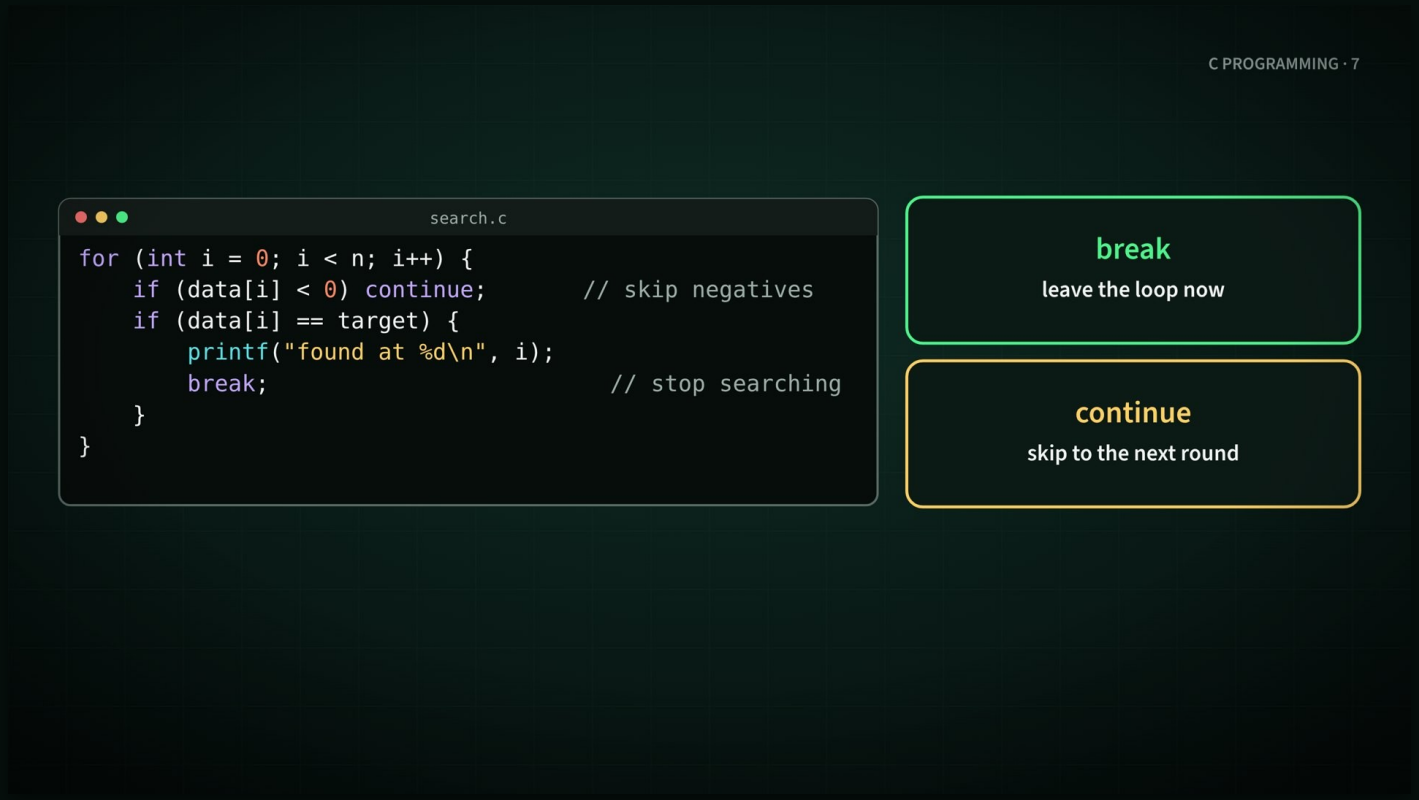
terminal

```
for (int i = 0; i < 10; i++) {
    printf("%d squared is %d\n", i, i * i);
}
```

- start — int i = 0
- condition — i < 10
- body — do the work

7.03

break and continue



- break — leave the loop now
- continue — skip to the next round

7.04

Nested Loops

table.c

```c
for (int row = 1; row <= 3; row++) {
    for (int col = 1; col <= 5; col++) {
        printf("%4d", row * col);
    }
    printf("\n");
}
//    1   2   3   4   5
//    2   4   6   8  10
//    3   6   9  12  15
```

- A loop inside another loop is called a nested loop. The inner loop runs completely for …
- Nested loops draw tables, process grids and images, and compare every item with every other …

7.05

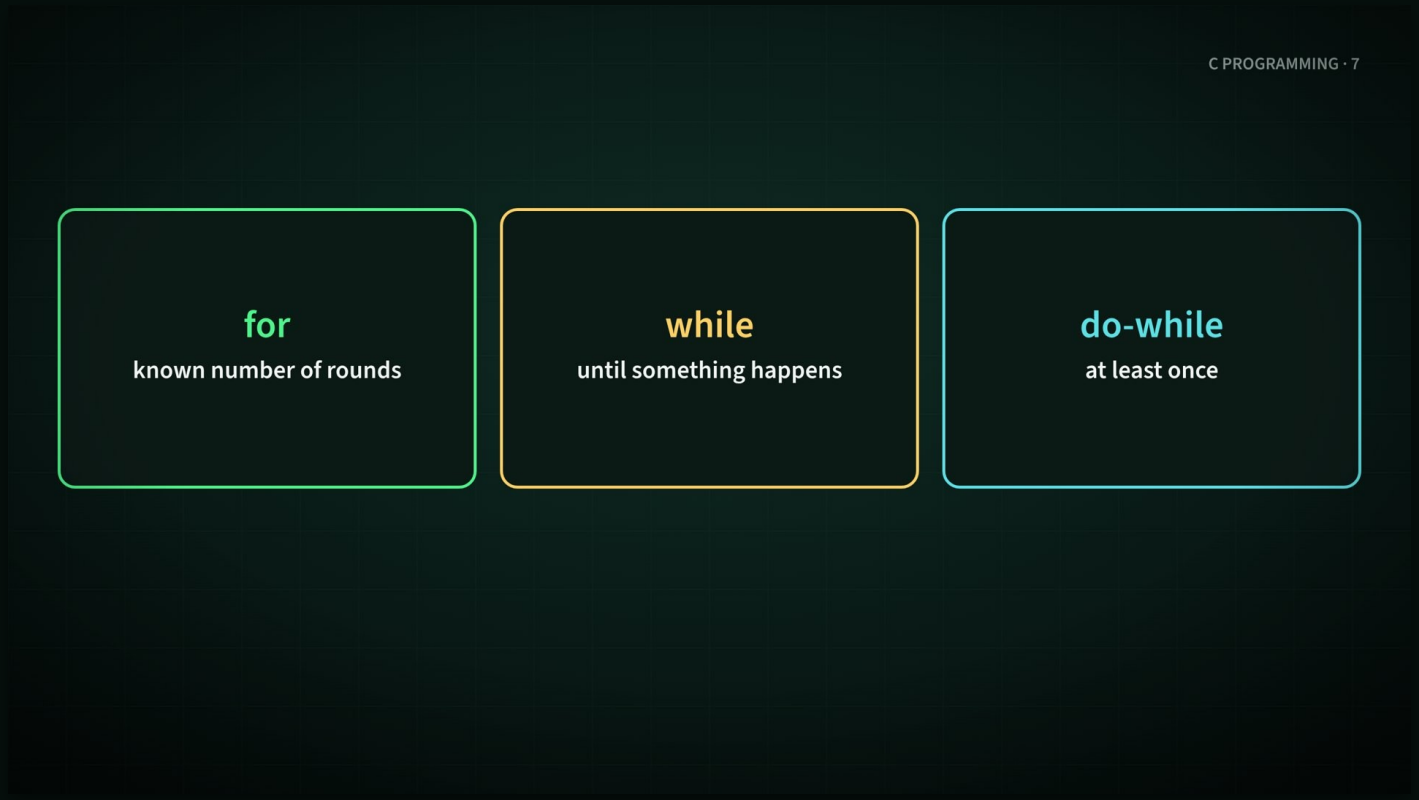
Infinite Loops



- A loop whose condition never becomes false runs forever. Usually that is a bug: you forgot …
- Sometimes it is intended, like a server waiting for requests, written as while one or for …
- If your program hangs in the terminal, press Control C to stop it.

7.06

Which Loop Should I Use?



- for — known number of rounds
- while — until something happens
- do-while — at least once

7.07



A FREE MYWEBUNIVERSITY COURSE

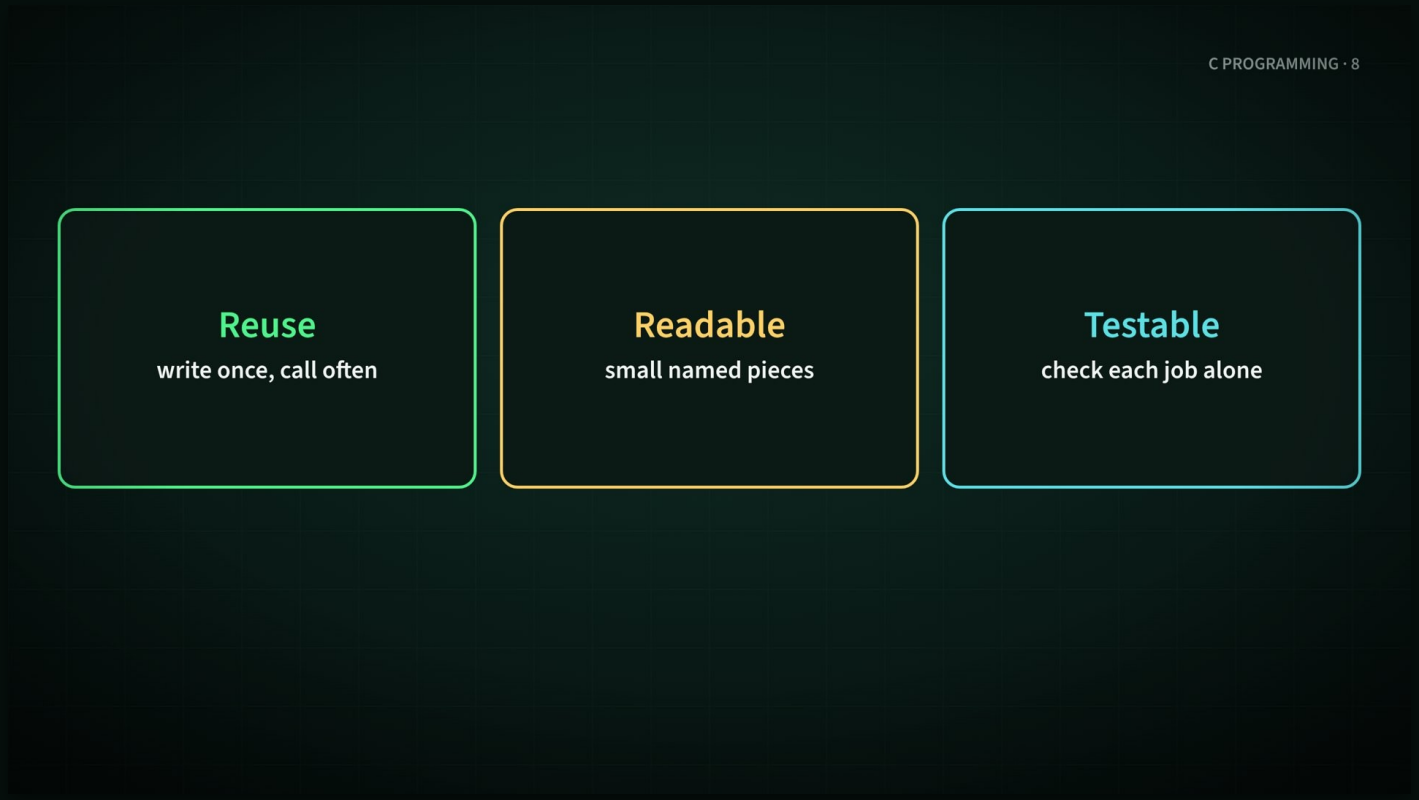
C PROGRAMMING

LESSON 8

Functions, Prototypes and Scope

Top-down programming with user-defined functions

Why Functions?



- Reuse — write once, call often
- Readable — small named pieces
- Testable — check each job alone

8.01

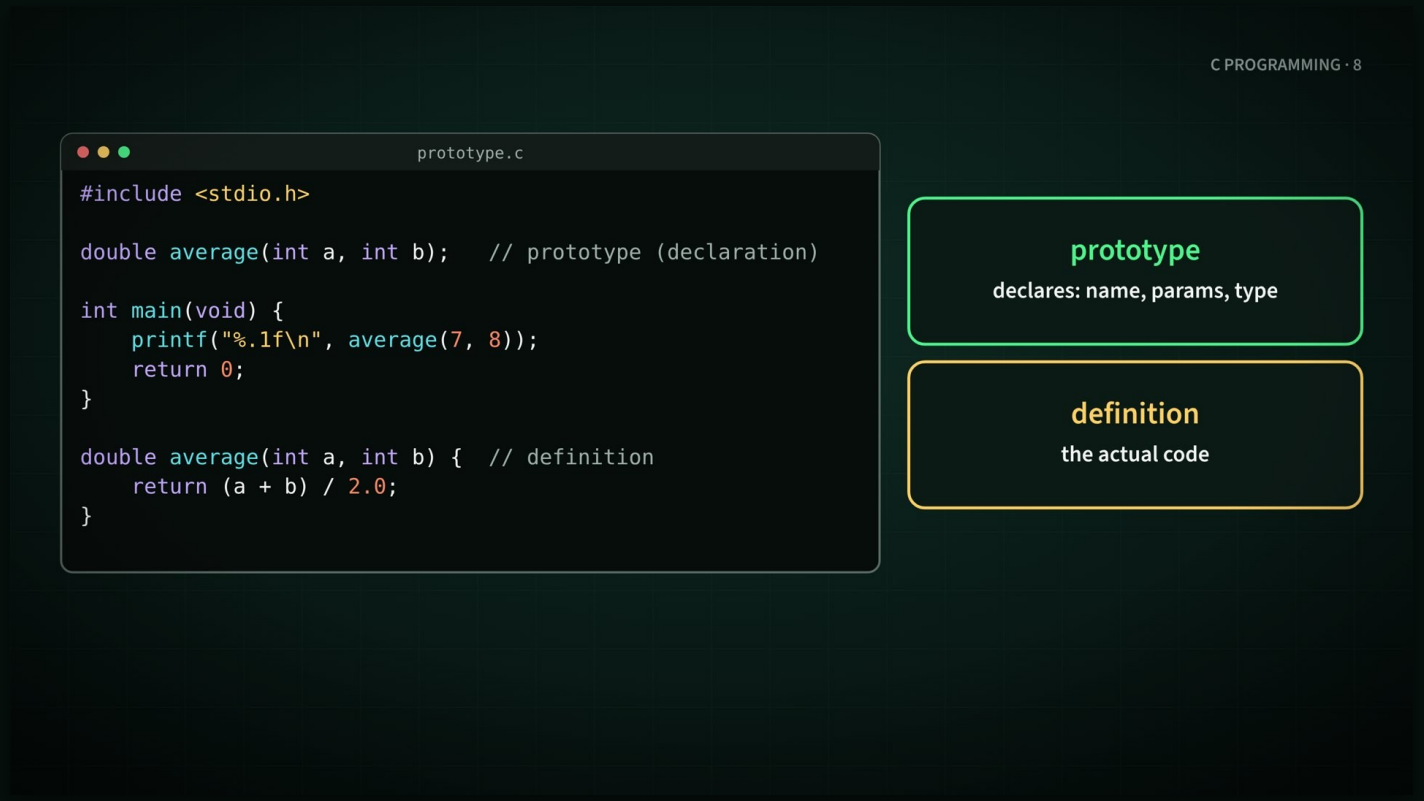
Defining a Function



- return type — int, double, void …
- parameters — inputs
- return — send back the result

8.02

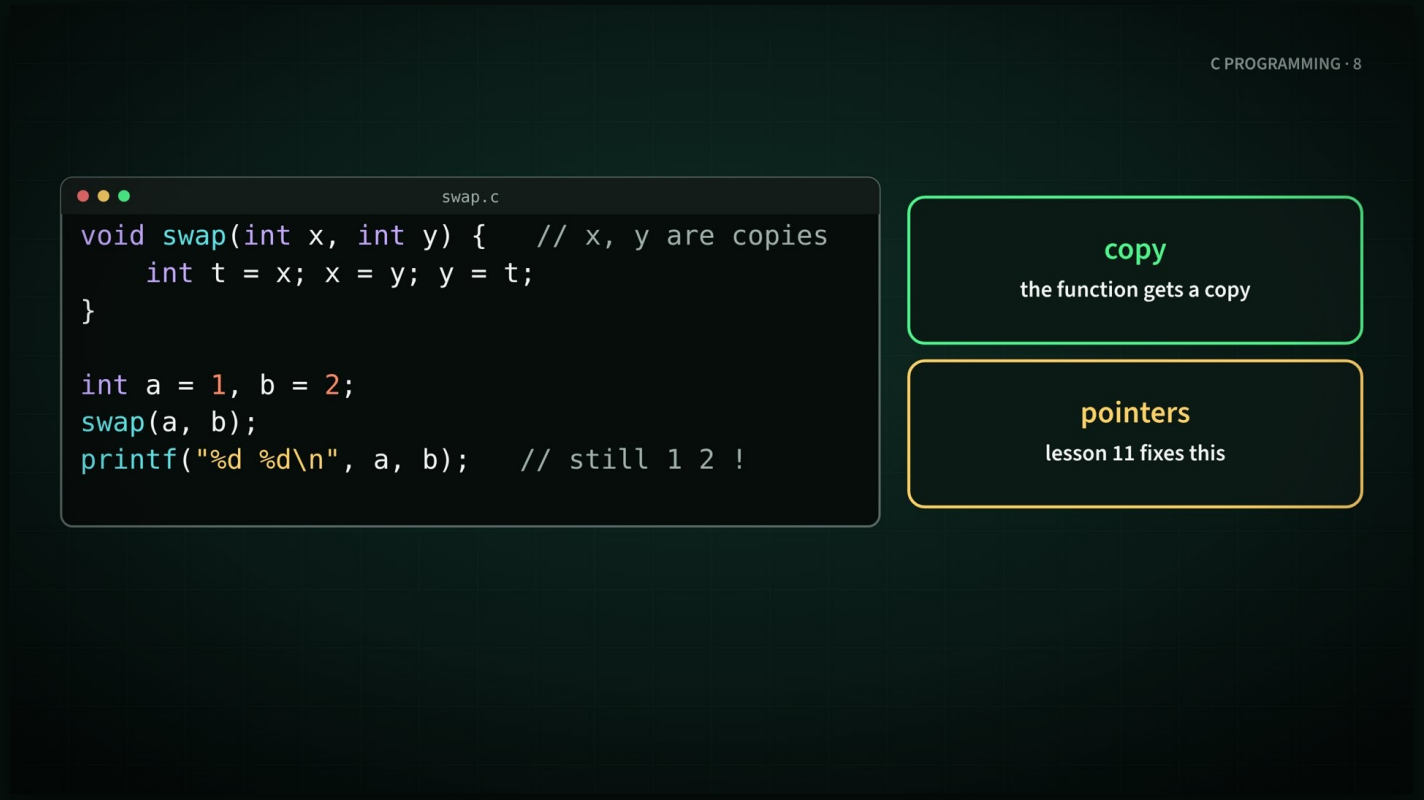
Prototypes



- prototype — declares: name, params, type
- definition — the actual code

8.03

Pass by Value



- copy — the function gets a copy
- pointers — lesson 11 fixes this

8.04

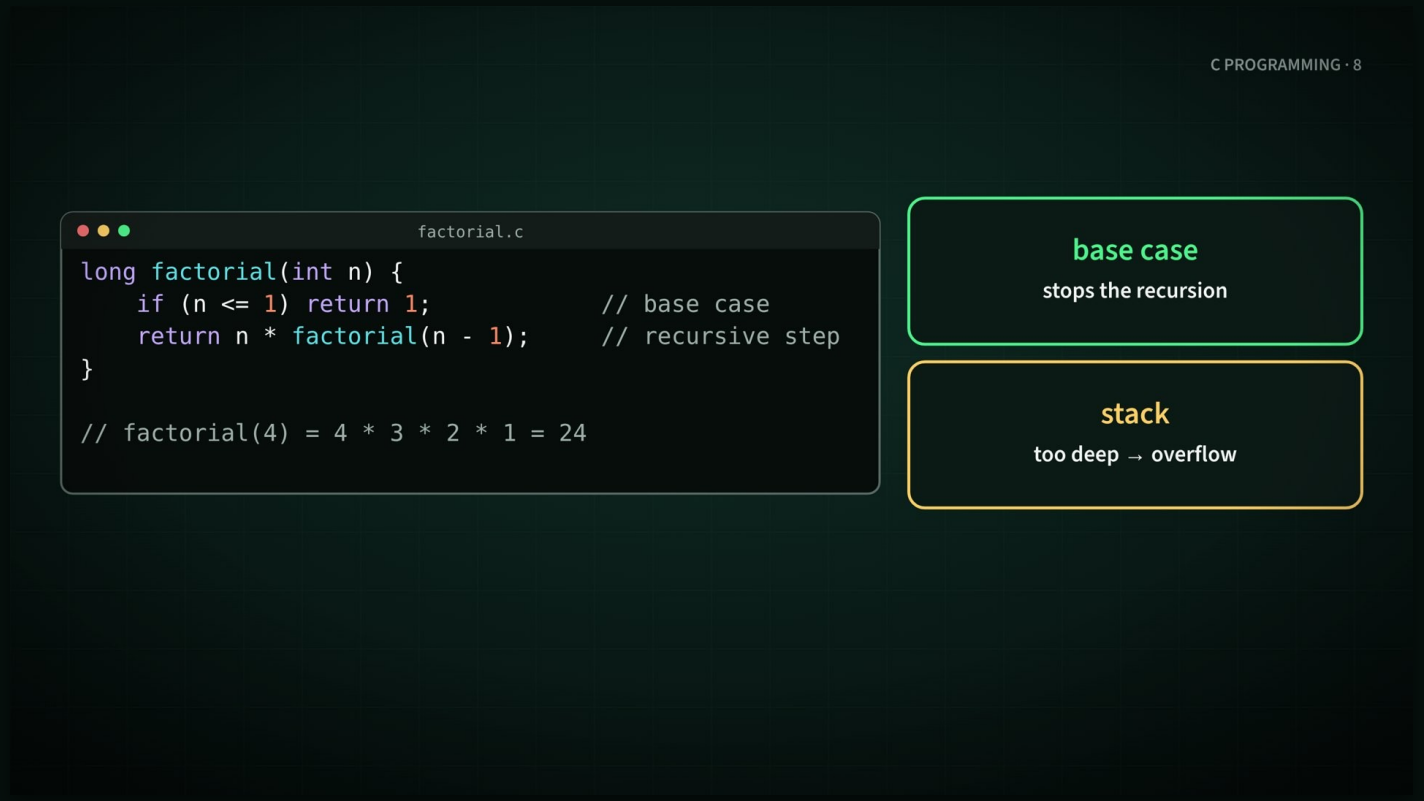
Scope and Lifetime



- local — inside its block
- global — everywhere — avoid
- static local — remembers between calls

8.05

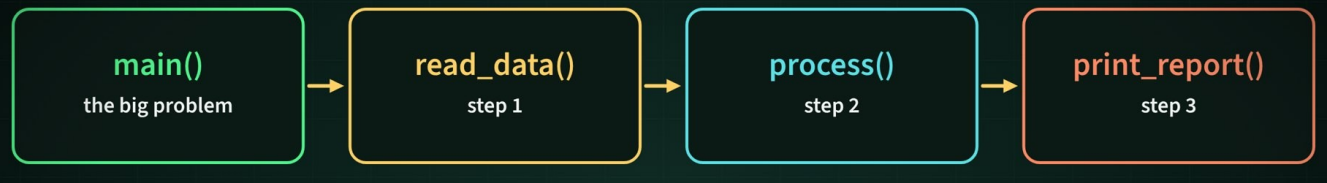
Recursion



- base case — stops the recursion
- stack — too deep → overflow

8.06

Top-Down Design



terminal

```
int main(void) {
    Data d = read_data();
    Result r = process(d);
    print_report(r);
}
```

- main() — the big problem
- read_data() — step 1
- process() — step 2

8.07



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 9

Arrays and Multi-Dimensional Arrays

Storing many values under one name

What Is an Array?

main.c

```c
int scores[5] = {90, 85, 77, 64, 98};
scores[2] = 80;            // change the third element
```



- An array stores many values of the same type, side by side in memory, under one name.
- Instead of five variables for five test scores, you write one array of five ints. Each …
- Indexes start at zero. So an array of five elements has indexes zero to four.

9.01

Arrays and Loops

average.c

```c
int scores[] = {90, 85, 77, 64, 98};
int n = sizeof(scores) / sizeof(scores[0]);   // 5

int sum = 0;
for (int i = 0; i < n; i++) {
    sum += scores[i];
}
printf("average = %.1f\n", (double)sum / n);   // 82.8
```

- Arrays and for loops are best friends. The loop counter walks through every index from zero …
- To get the number of elements, divide the size of the whole array by the size of one …

9.02

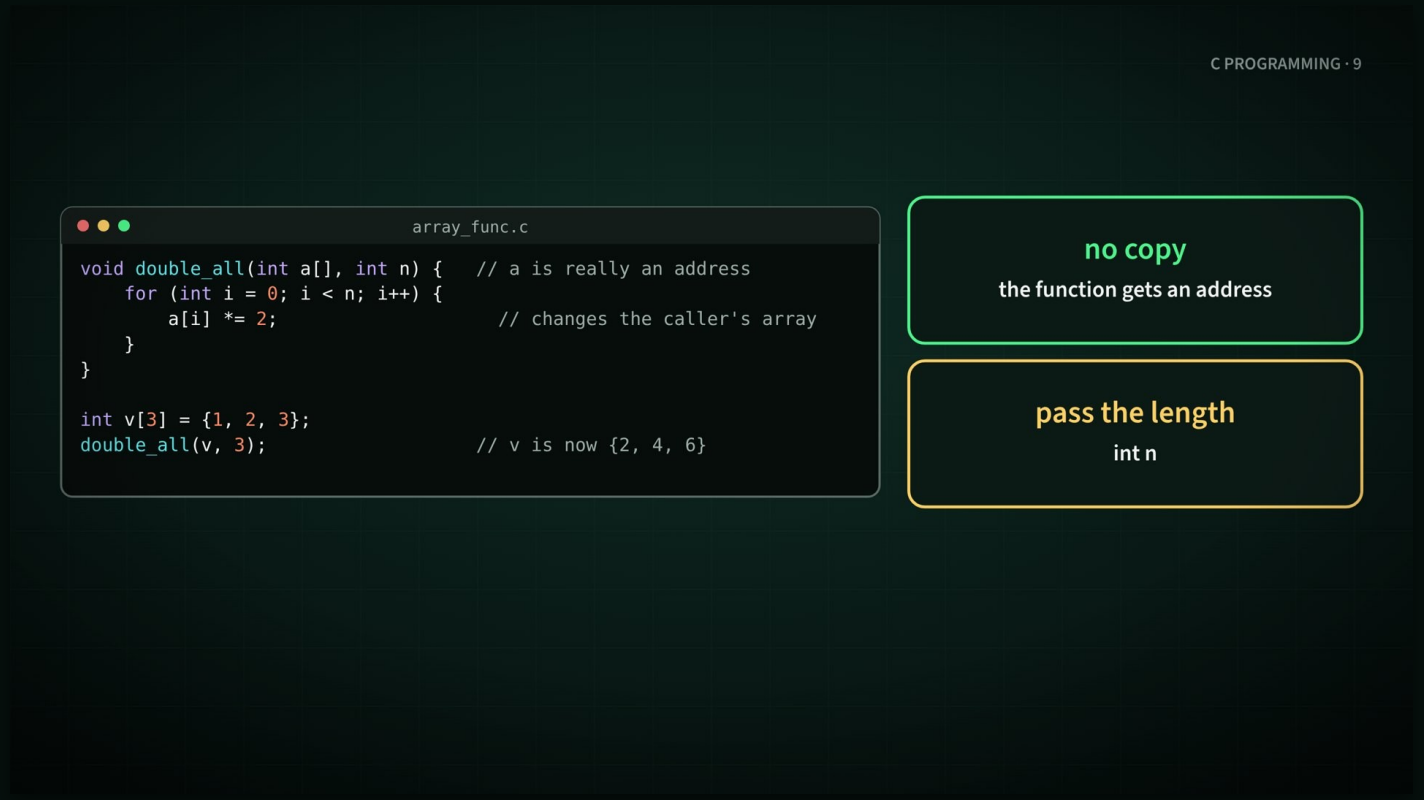
No Bounds Checking!



- valid indexes — 0 … n-1
- i < n — never i <= n
- -fsanitize=address — catches overflows

9.03

Passing Arrays to Functions



- no copy — the function gets an address
- pass the length — int n

9.04

Two-Dimensional Arrays

matrix.c

```c
int m[3][4] = {
    { 1,  2,  3,  4},
    { 5,  6,  7,  8},
    { 9, 10, 11, 12}
};
for (int r = 0; r < 3; r++) {
    for (int c = 0; c < 4; c++) {
        printf("%3d", m[r][c]);
    }
    printf("\n");
}
// m[1][2] is 7
```

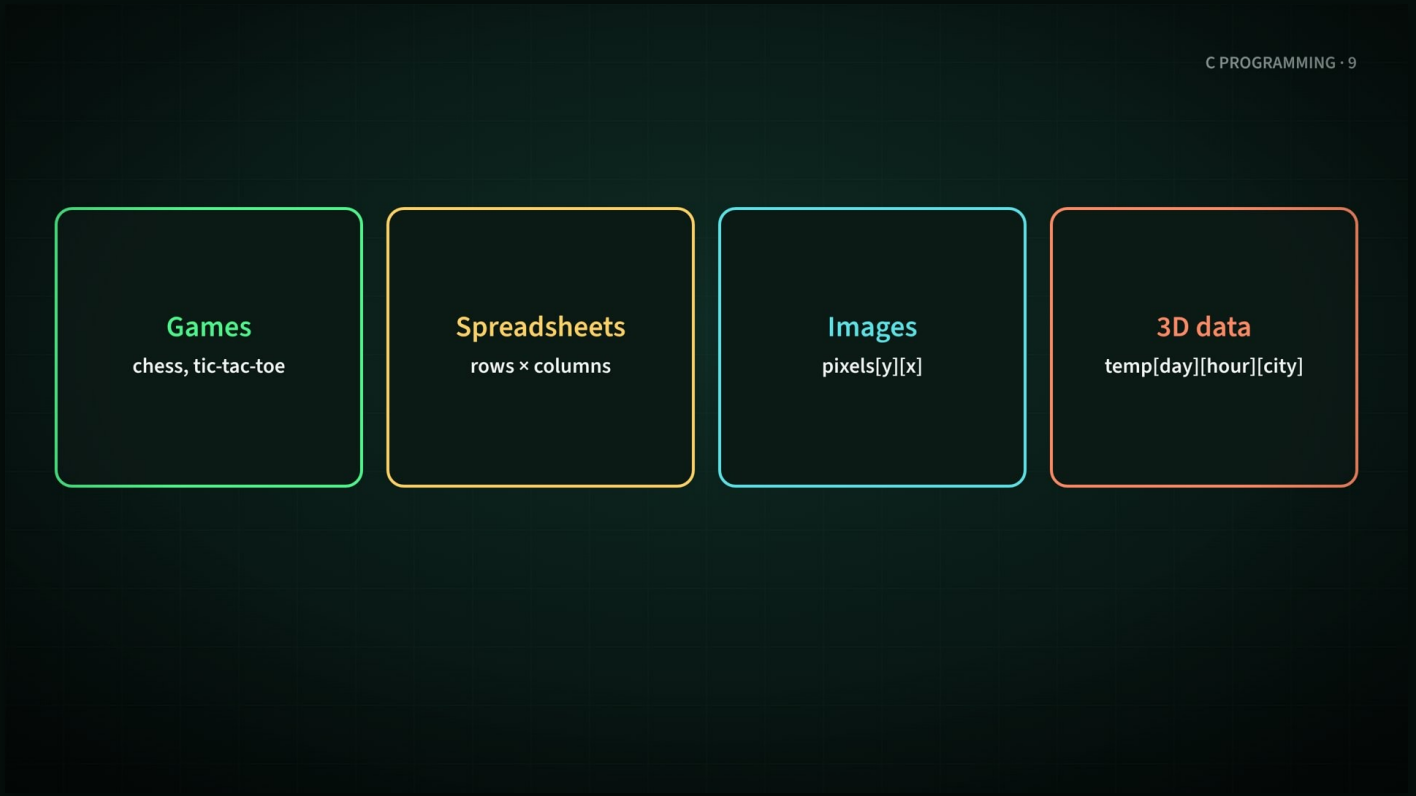
m[row][col]
row first

row-major
rows stored one after another

- m[row][col] — row first
- row-major — rows stored one after another

9.05

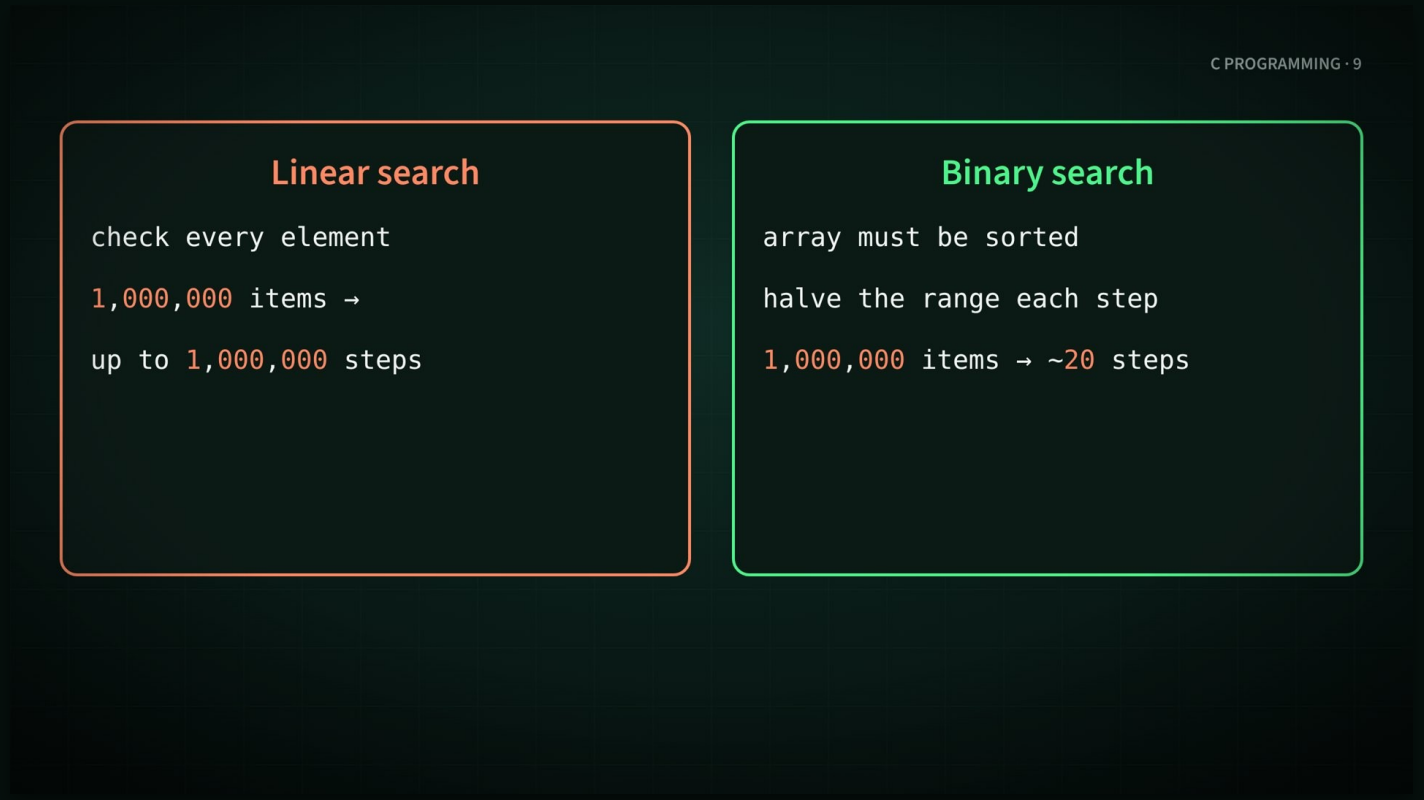
More Dimensions and Uses



- Games — chess, tic-tac-toe
- Spreadsheets — rows × columns
- Images — pixels[y][x]

9.06

Searching and Sorting



- Two classic tasks on arrays are searching and sorting. Linear search checks every element; …
- If the array is sorted, binary search halves the range at each step, so it finds an item …
- The C library includes qsort for sorting and bsearch for binary search. We will use qsort …

9.07



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 10

Strings in C

char arrays, the null terminator and string.h

A String Is a char Array

main.c

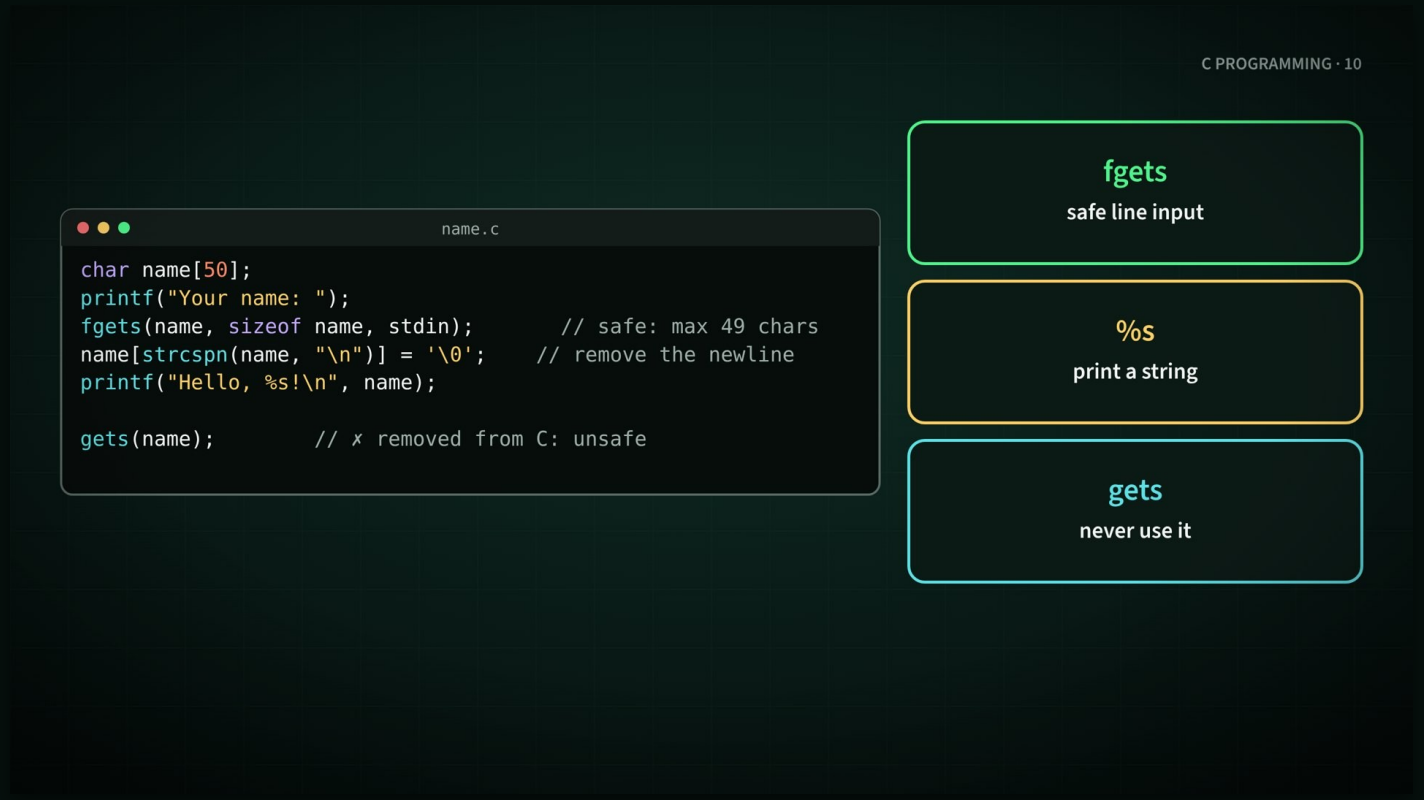
```
char city[] = "Kabul";        // 6 bytes
char c = 'K';                 // one character
```



- C has no special string type. A string is an array of characters that ends with a special …
- The word Kabul needs six bytes: five letters plus the terminator. Every string function …
- Single quotes make one character. Double quotes make a string, and the compiler adds the …

10.01

Reading and Printing Strings



- fgets — safe line input
- %s — print a string
- gets — never use it

10.02

The string.h Functions

Function	Example	Does
`strlen(s)`	`strlen("Kabul") → 5`	length
`strcpy(d, s)`	`strcpy(dst, "Hi")`	copy
`strcat(d, s)`	`strcat(dst, "!")`	append
`strcmp(a, b)`	`strcmp("a","b") → <0`	compare (0 = equal)
`strchr(s, c)`	`strchr(s, '@')`	find a character

- The header string.h provides functions for working with strings. strlen returns the length, …
- strcpy copies one string into another, strcat appends, and strcmp compares two strings, …
- You cannot compare strings with double equals, and you cannot assign them with equals; …

10.03

Safer String Handling



- snprintf — never writes past the size
- size + 1 — room for '\0'

10.04

Working Character by Character

upper.c

```c
#include <ctype.h>

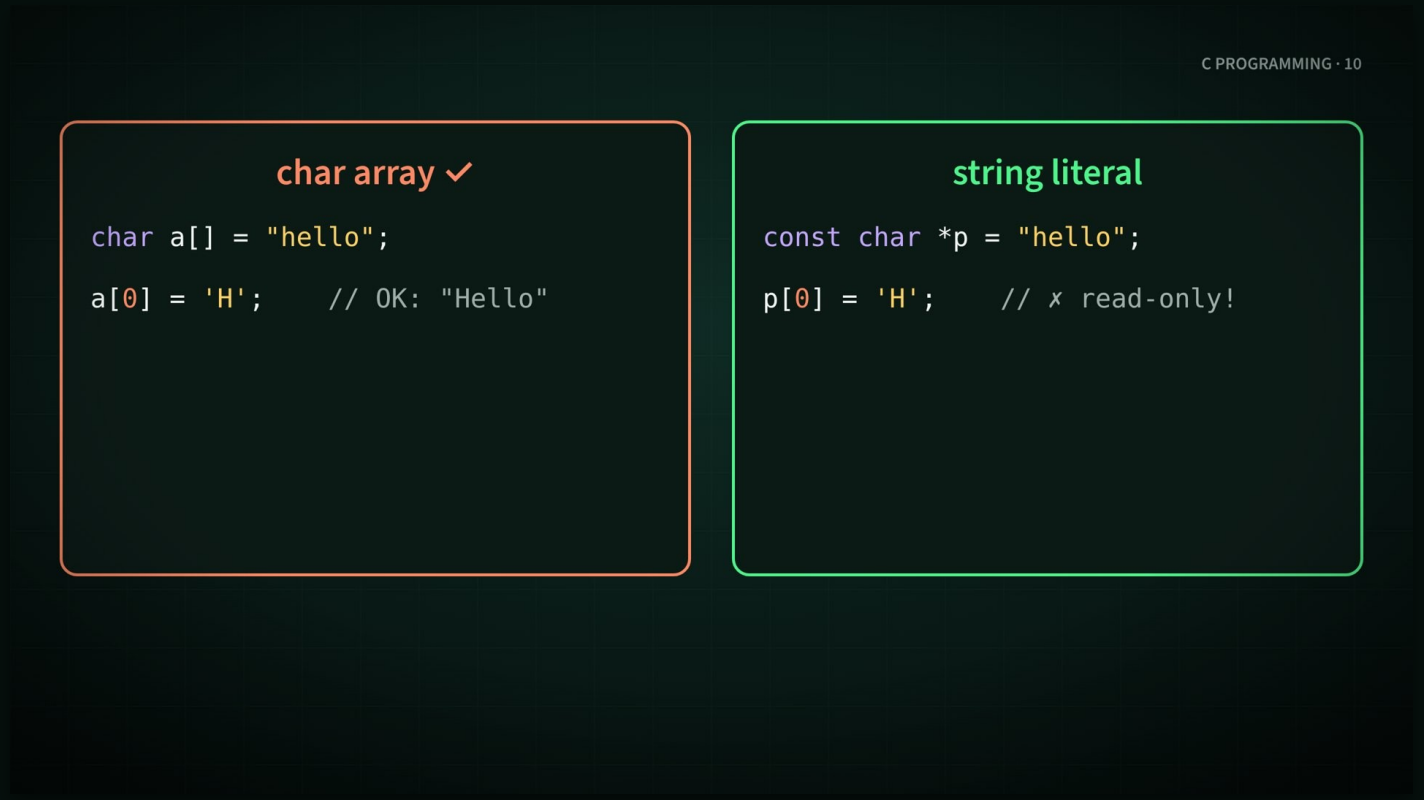
void to_upper(char *s) {
    for (int i = 0; s[i] != '\0'; i++) {
        s[i] = toupper((unsigned char)s[i]);
    }
}

char w[] = "kabul";
to_upper(w);          // w is now "KABUL"
```

- Because a string is an array, you can loop through it until you reach the terminator.
- The header ctype.h helps classify characters: isdigit, isalpha, isspace, and toupper and …

10.05

Arrays vs String Literals



- There is an important difference between a char array and a pointer to a string literal.
- A char array holds its own copy of the text, so you can change it. A pointer to a literal …

10.06



A FREE MYWEBUNIVERSITY COURSE

C PROGRAMMING

LESSON 11

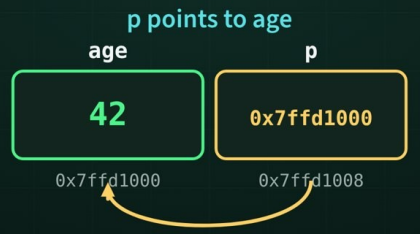
Pointers

Addresses, dereferencing, pointer arithmetic and function pointers

Every Variable Has an Address

main.c

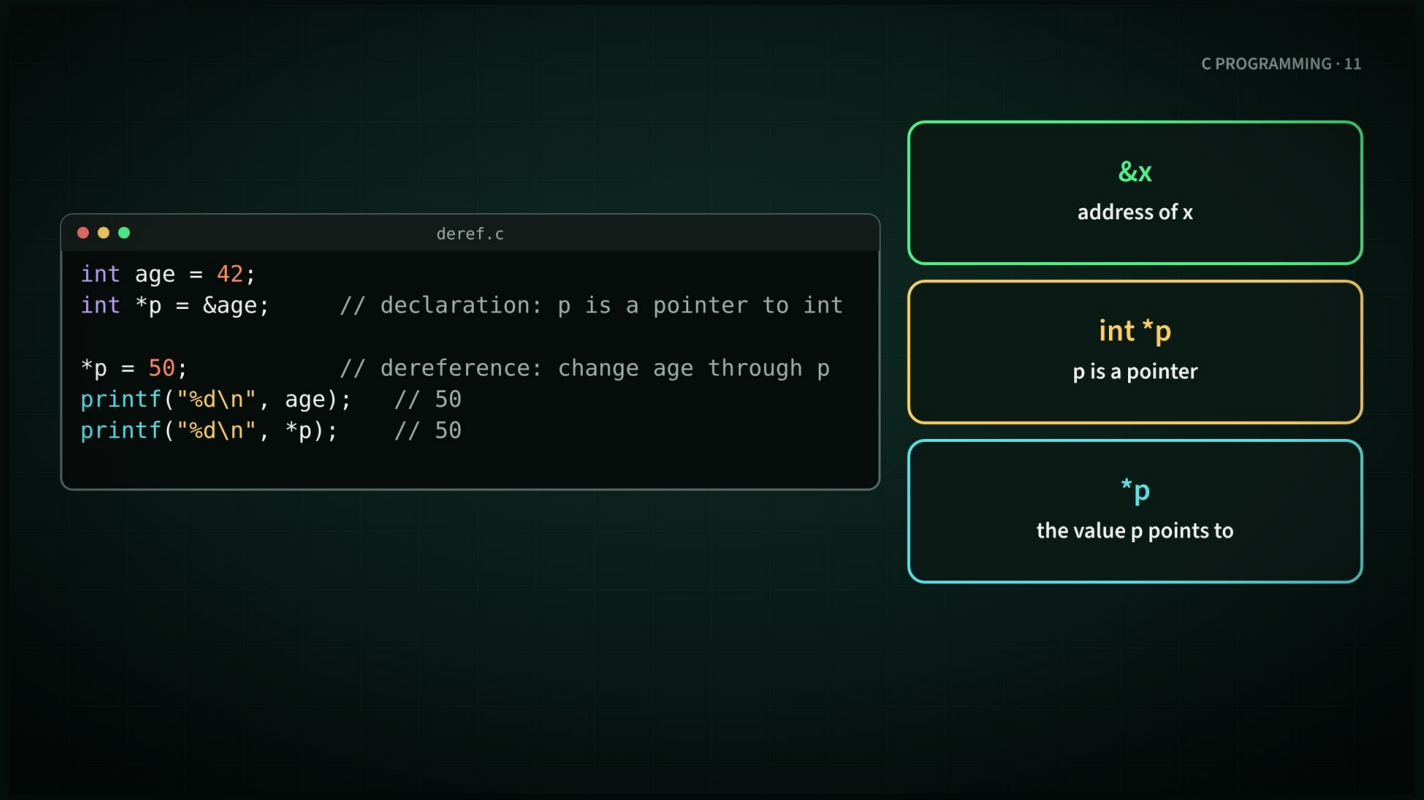
```c
int age = 42;
int *p = &age;            // p holds the address of age
printf("%p\n", (void*)p);  // 0x7ffd1000
```



- Memory is a long row of numbered bytes. The number of a byte is its address, just like a …
- The ampersand operator gives you the address of a variable. You can print it with percent p.
- A pointer is simply a variable that stores an address.

11.01

Declaring and Dereferencing



- &x — address of x
- int *p — p is a pointer
- *p — the value p points to

11.02

Pass by Reference: swap Works!

swap.c

```c
void swap(int *x, int *y) {
    int t = *x;
    *x = *y;
    *y = t;
}

int a = 1, b = 2;
swap(&a, &b);
printf("%d %d\n", a, b);   // 2 1  ✓
```

&a, &b
send addresses

***x = *y**
change the originals

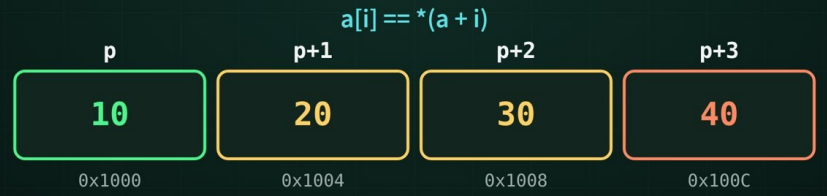
- &a, &b — send addresses
- *x = *y — change the originals

11.03

Pointers and Arrays

main.c

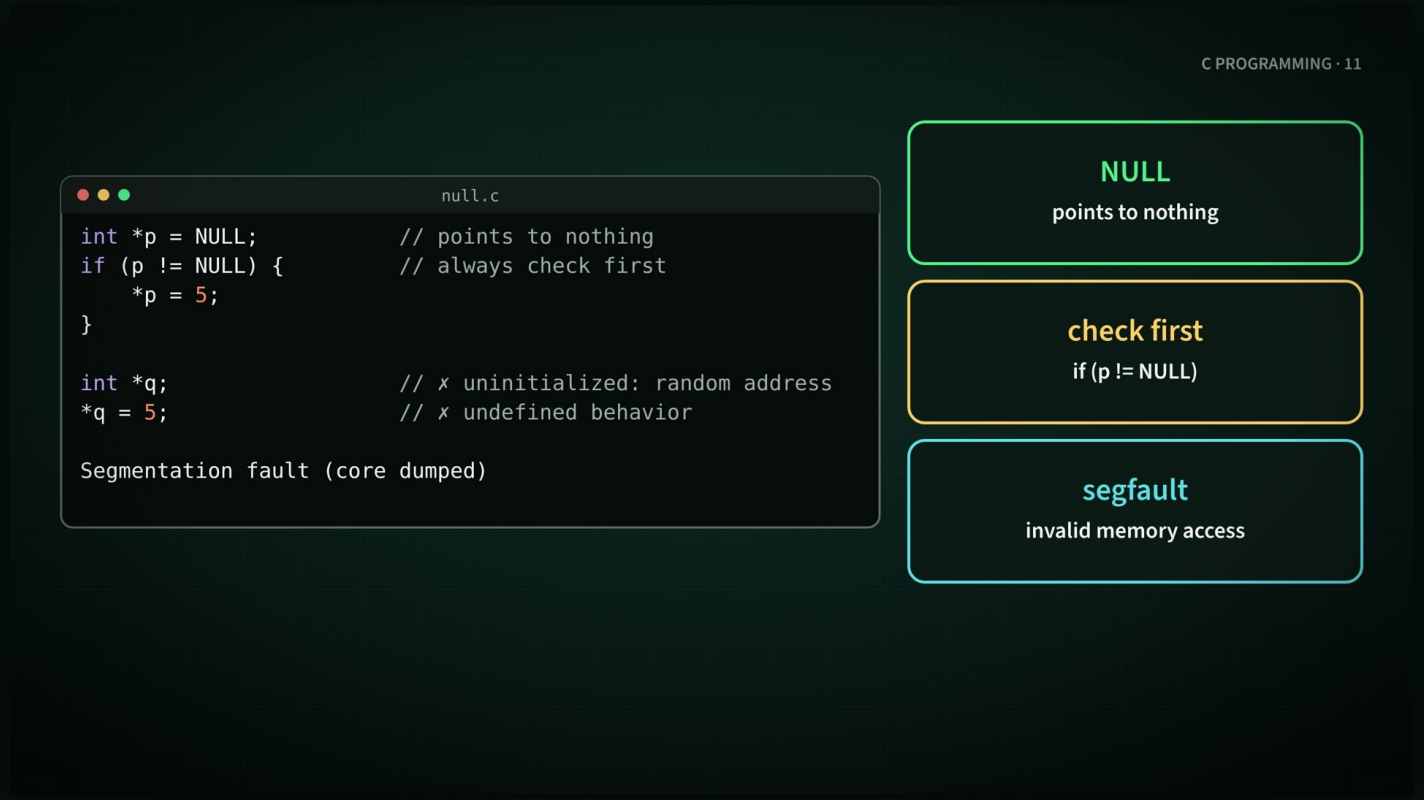
```c
int a[4] = {10, 20, 30, 40};
int *p = a;                  // same as &a[0]
printf("%d\n", *(p + 2));    // 30, same as a[2]
```



- An array name turns into a pointer to its first element. So a and the address of a zero are …
- Pointer arithmetic moves by whole elements, not bytes: p plus one points to the next int, …
- In fact, a sub i is defined as star of a plus i. That is why arrays and pointers work so …

11.04

NULL and Dangerous Pointers



- NULL — points to nothing
- check first — if (p != NULL)
- segfault — invalid memory access

11.05

const with Pointers

constptr.c

```c
const int *p = &x;    // pointer to const int: *p = 1 ✗, p = &y ✓
int *const q = &x;    // const pointer:        *q = 1 ✓, q = &y ✗

size_t length(const char *s);   // promise: I only read s
```

- const and pointers combine in two ways. A pointer to const means you cannot change the …
- A const pointer means the pointer itself cannot point anywhere else. Read the declaration …

11.06

Pointers to Pointers and Function Pointers

fptr.c

```c
int main(int argc, char **argv);    // argv: pointer to pointers

int add(int a, int b) { return a + b; }
int mul(int a, int b) { return a * b; }

int (*op)(int, int) = add;          // function pointer
printf("%d\n", op(3, 4));   // 7
op = mul;
printf("%d\n", op(3, 4));   // 12
```

- A pointer can point to another pointer: int star star. You see this with argv, an array of …
- A function pointer stores the address of a function, so you can pass behavior as an …

11.07



A FREE MYWEBUNIVERSITY COURSE

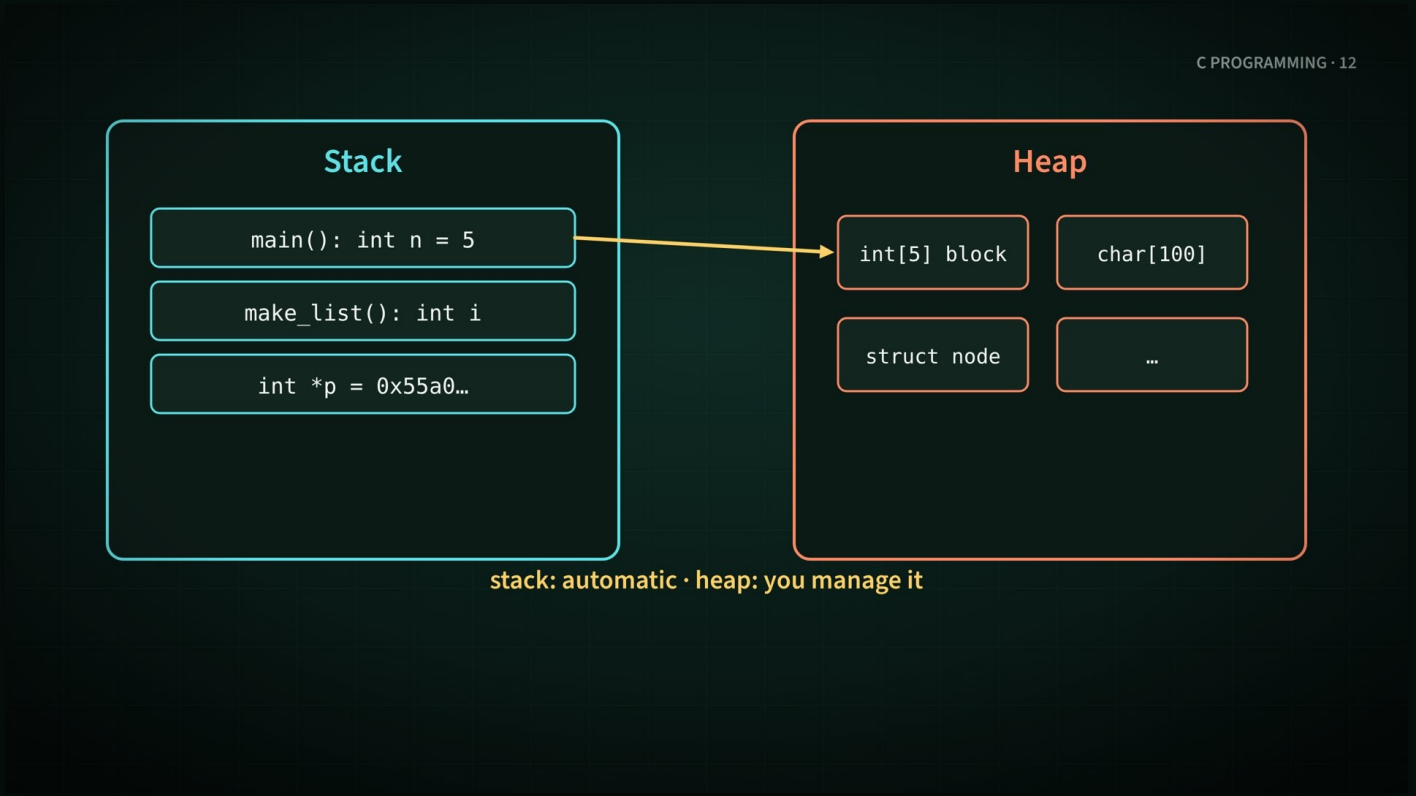
C PROGRAMMING

LESSON 12

Dynamic Memory and Memory Leaks

Stack vs heap, malloc, free, and finding leaks

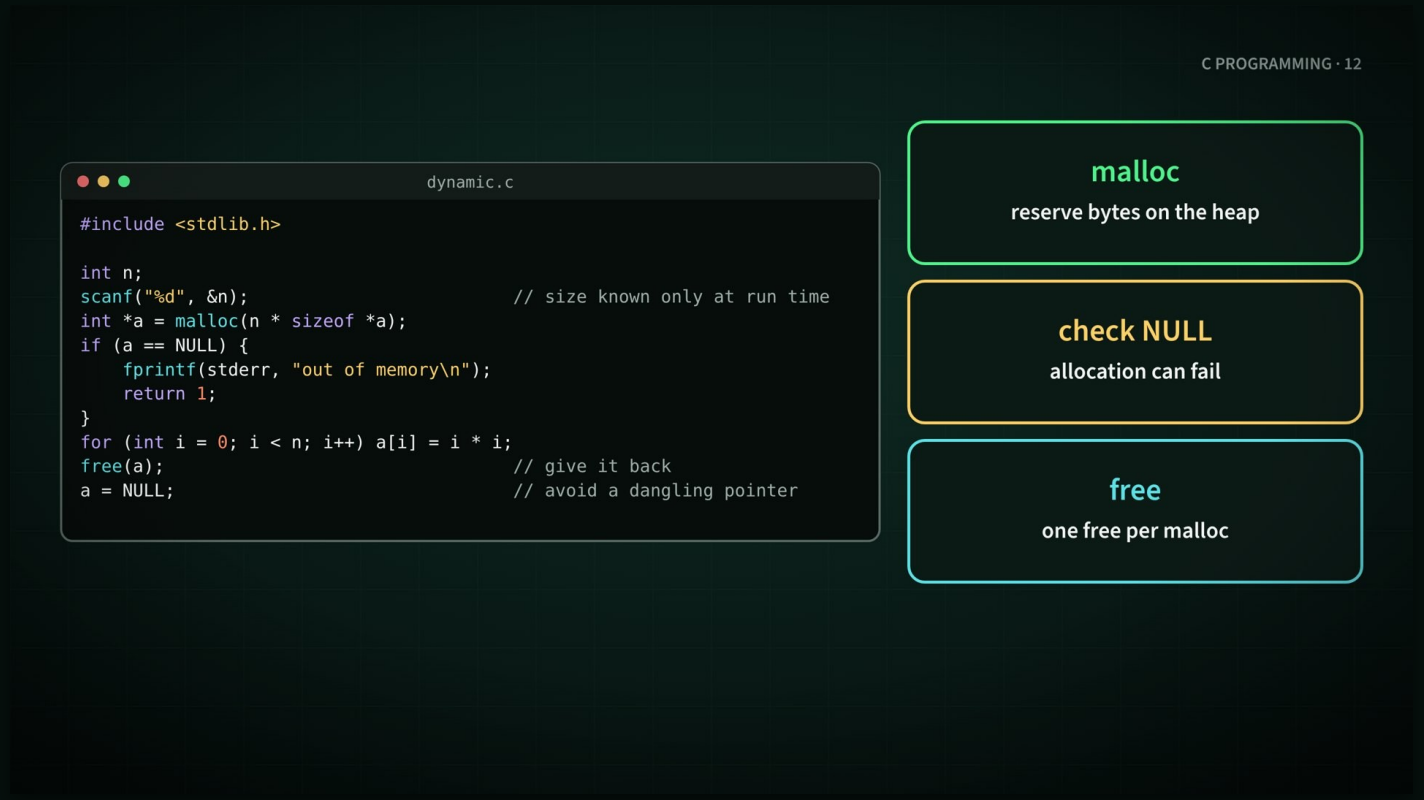
The Stack and the Heap



- A running C program uses two main areas of memory. The stack holds local variables. It is …
- The heap is a large pool of memory that you request and release yourself. Use it when you …

12.01

malloc and free



- malloc — reserve bytes on the heap
- check NULL — allocation can fail
- free — one free per malloc

12.02

calloc and realloc

grow.c

```c
int *a = calloc(10, sizeof *a);          // 10 zeros

int *bigger = realloc(a, 20 * sizeof *a); // grow to 20
if (bigger == NULL) {
    free(a);                             // old block still valid
    return 1;
}
a = bigger;
free(a);
```

- calloc is like malloc, but it fills the memory with zeros. Use it when you need a clean …
- realloc changes the size of a block, for example to grow an array when more data arrives. …

12.03

Memory Leaks

leak.c

```c
void handle_request(void) {
    char *buf = malloc(1024);
    if (!buf) return;
    /* ... use buf ... */
    if (error) return;          // ✗ leak: returned without free
    free(buf);
}

// 1 KB × 1,000,000 requests = 1 GB lost
```

- A memory leak happens when you allocate memory and lose the last pointer to it without …
- In a short program that may not matter. But a server that runs for months and leaks a …

12.04

Dangling Pointers and Double Free



- After free, the pointer still holds the old address, but the memory no longer belongs to …
- Calling free twice on the same pointer is a double free, which can corrupt the heap. …

12.05

Finding Memory Bugs

terminal

```
$ gcc -g -fsanitize=address leak.c -o leak && ./leak
ERROR: LeakSanitizer: detected memory leaks
Direct leak of 1024 byte(s) in 1 object(s) allocated from:
    #1 handle_request leak.c:2

$ valgrind --leak-check=full ./leak
definitely lost: 1,024 bytes in 1 blocks
```

- You do not have to find these bugs by hand. Compile with -fsanitize=address, and the …
- On Linux, Valgrind runs your program and reports every byte that was never freed. …

12.06



A FREE MYWEBUNIVERSITY COURSE

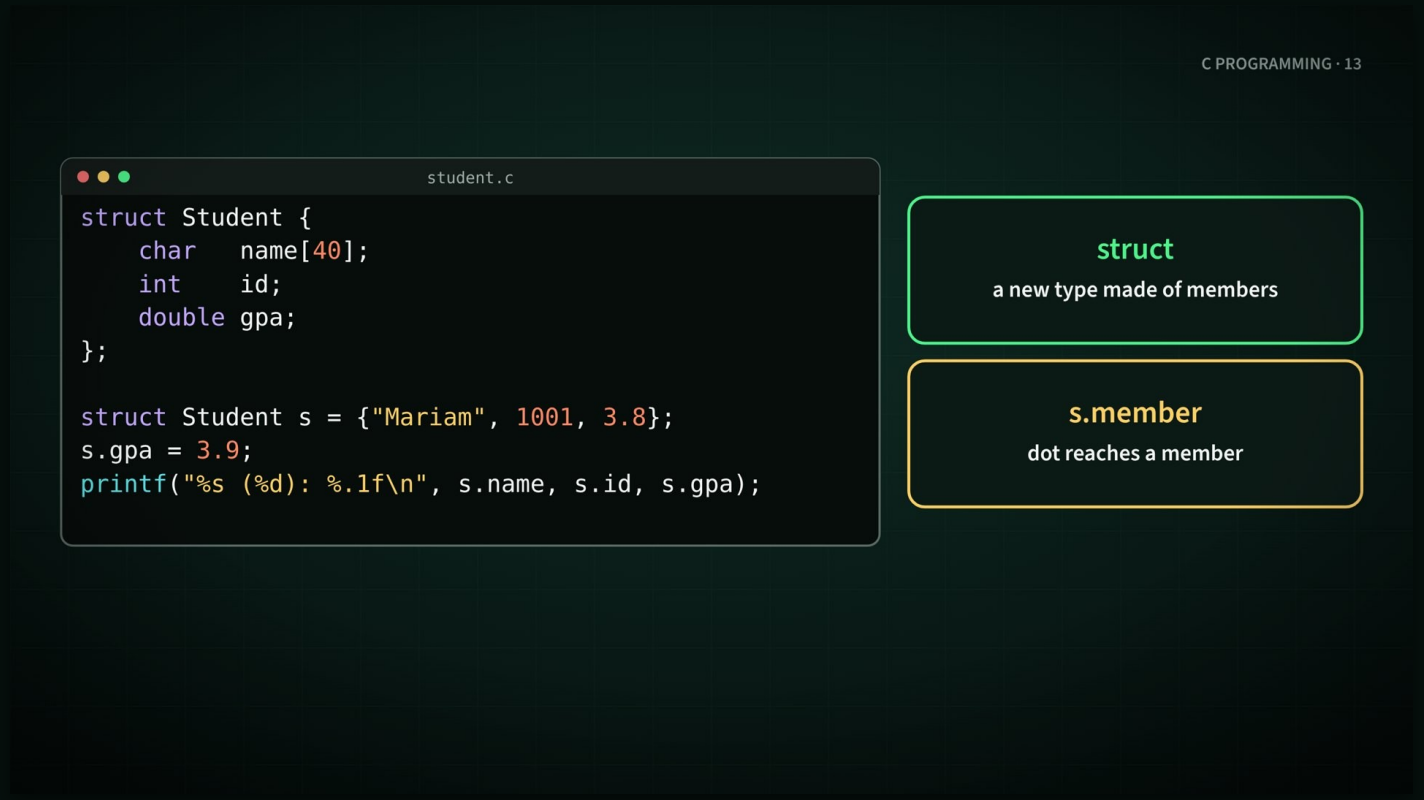
C PROGRAMMING

LESSON 13

struct, union, enum and typedef

Building your own data types

struct: Grouping Related Data



- struct — a new type made of members
- s.member — dot reaches a member

13.01

typedef: Shorter Names

typedef.c

```c
typedef struct {
    double x, y;
} Point;

Point a = {0, 0}, b = {3, 4};

typedef unsigned char byte;
byte pixel = 255;
```

- Writing struct Student every time is long. typedef gives a type a new name, so you can …
- typedef is also used to give meaningful names to basic types, like typedef unsigned char byte.

13.02

Arrays of Structs and the Arrow

class.c

```c
Student class[30];

void raise_gpa(Student *p) {   // pointer: no copy
    p->gpa += 0.1;              // same as (*p).gpa
}

for (int i = 0; i < 30; i++) {
    raise_gpa(&class[i]);
}
```

p->member
through a pointer

by pointer
no big copies

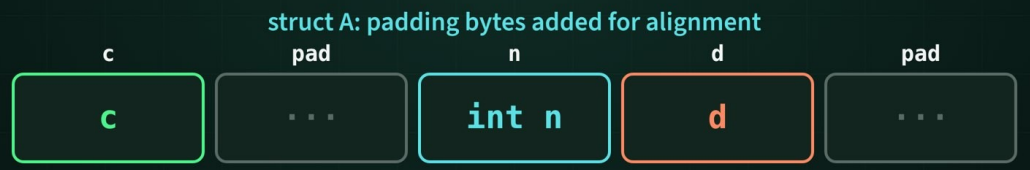
- p->member — through a pointer
- by pointer — no big copies

13.03

Memory Layout and Padding

main.c

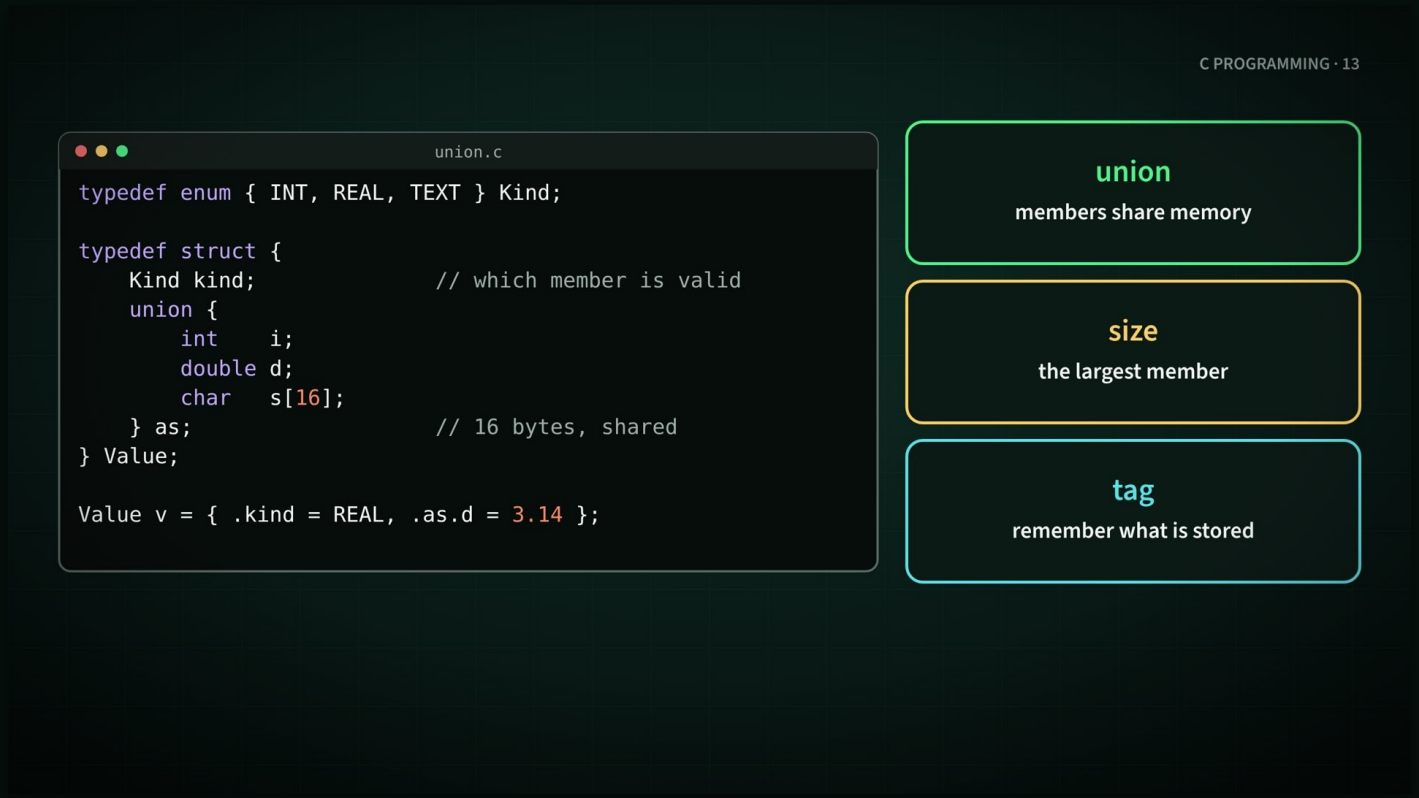
```
struct A { char c; int n; char d; };   // sizeof = 12
struct B { int n; char c; char d; };   // sizeof = 8
```



- The members of a struct are stored in order, but the compiler may add empty padding bytes …
- That is why sizeof a struct can be bigger than the sum of its members. Ordering members …

13.04

union: Sharing Memory



- union — members share memory
- size — the largest member
- tag — remember what is stored

13.05

enum: Named Constants

enum.c

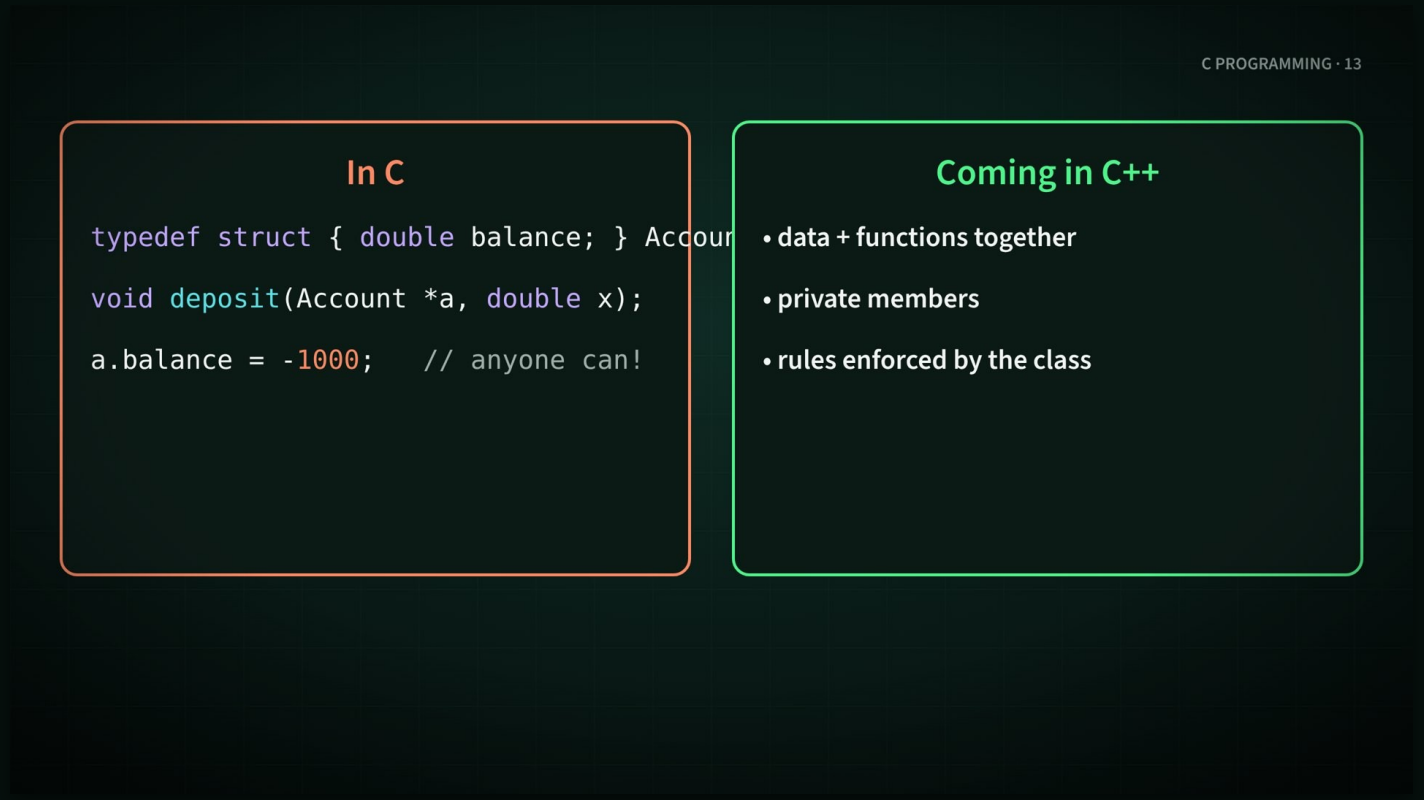
```c
enum Light { RED, GREEN, YELLOW };   // 0, 1, 2
enum Http  { OK = 200, NOT_FOUND = 404 };

enum Light l = GREEN;
switch (l) {
    case RED:    printf("Stop\n"); break;
    case GREEN:  printf("Go\n");   break;
    case YELLOW: printf("Slow\n"); break;
}
```

- An enum defines a set of named integer constants. Instead of remembering that 2 means …
- By default the values start at zero and count up, but you can choose them. enums make …

13.06

Modeling the Real World



- With struct, union, enum and typedef you can model almost anything: a bank account, a game …
- But notice what is missing: the functions that work on a struct live separately from it, …

13.07



A FREE MYWEBUNIVERSITY COURSE

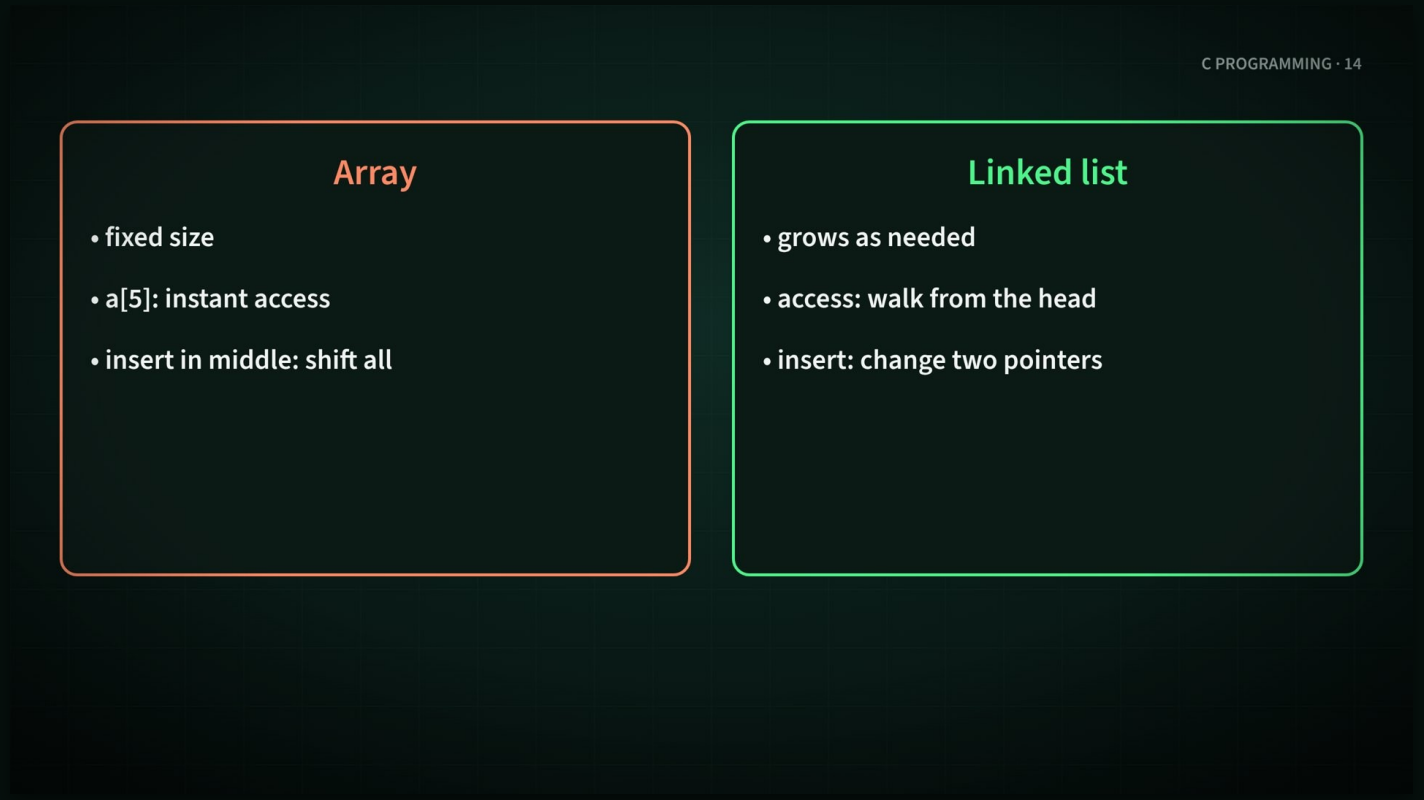
C PROGRAMMING

LESSON 14

Linked Lists

Your first dynamic data structure

Why Not Just an Array?



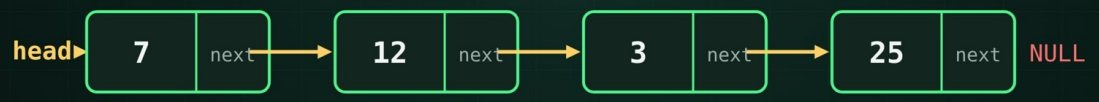
- Arrays have a fixed size, and inserting an element in the middle means shifting everything …
- A linked list grows one node at a time, and inserting or removing a node only changes a …

14.01

The Node

list.c

```c
typedef struct Node {
    int data;
    struct Node *next;
} Node;
```



- A linked list is a chain of nodes. Each node is a struct holding some data and a pointer to …
- The first node is called the head. The last node's next pointer is NULL, which marks the …

14.02

Adding a Node at the Front

push.c

```c
int push(Node **head, int value) {
    Node *n = malloc(sizeof *n);
    if (n == NULL) return -1;
    n->data = value;
    n->next = *head;      // new node points to old head
    *head   = n;          // new node becomes the head
    return 0;
}

Node *list = NULL;
push(&list, 3); push(&list, 12); push(&list, 7);
```

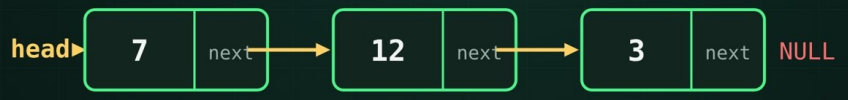
- To add a node at the front, allocate a new node with malloc, store the data, point its next …
- Because the function must change the caller's head pointer, it receives a pointer to the …

14.03

Walking the List

list.c

```c
for (Node *p = head; p != NULL; p = p->next) {
    printf("%d -> ", p->data);
}
printf("NULL\n");    // 7 -> 12 -> 3 -> NULL
```



- To visit every node, start at the head and follow the next pointers until you reach NULL.
- This same loop is the base of printing, counting, searching and summing. It is the most …

14.04

Removing a Node

remove.c

```c
void remove_value(Node **link, int value) {
    while (*link != NULL) {
        if ((*link)->data == value) {
            Node *dead = *link;
            *link = dead->next;     // skip over it
            free(dead);             // give memory back
            return;
        }
        link = &(*link)->next;
    }
}
```

- To remove a node, make the previous node skip over it, then free it.
- Use a pointer to the link you might change. Then removing the head and removing a middle …

14.05

Freeing the Whole List

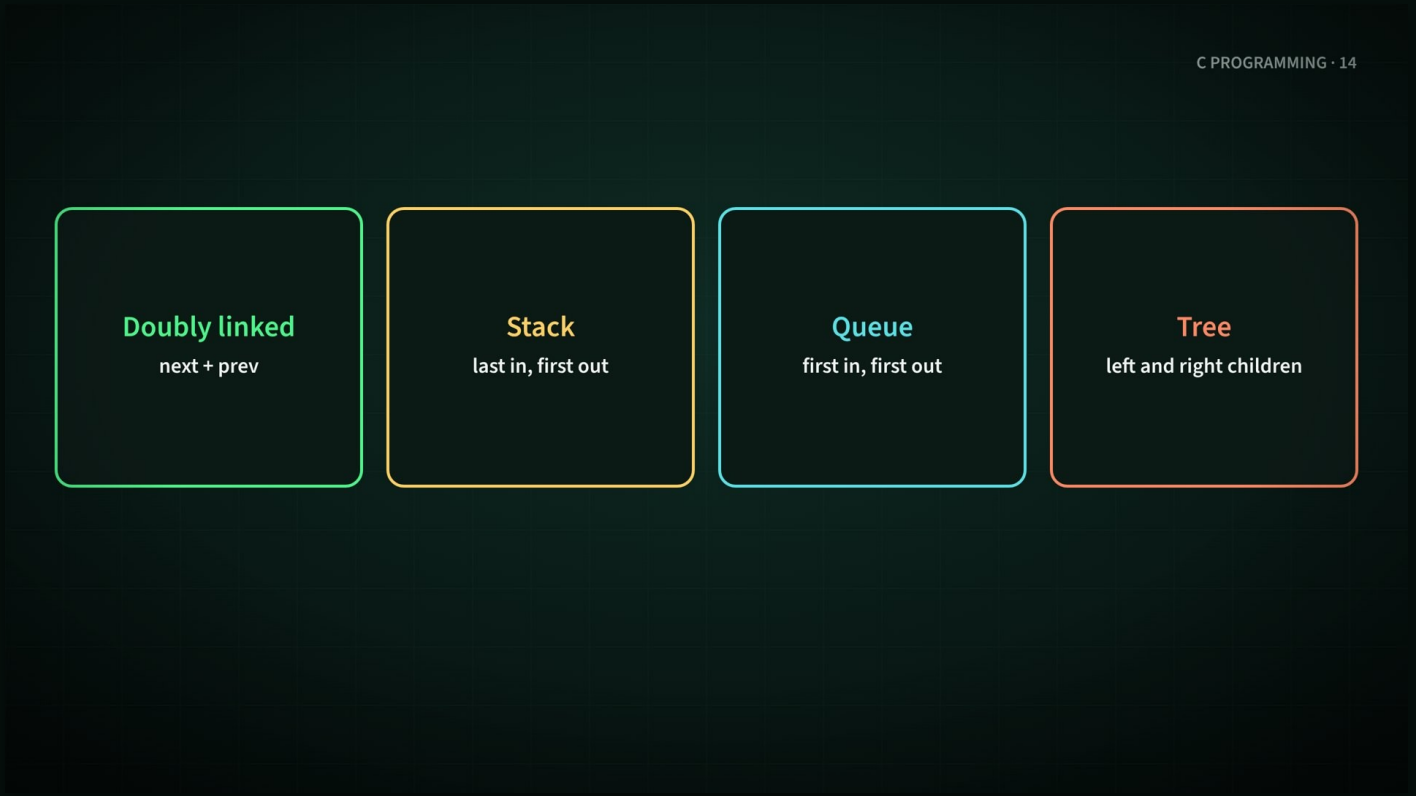
free_list.c

```c
void free_list(Node *head) {
    while (head != NULL) {
        Node *next = head->next;   // save first!
        free(head);
        head = next;
    }
}
```

- Every node was created with malloc, so every node must be freed. Walk the list, but save …
- If you free first and then read next, you are using freed memory, the use-after-free bug …

14.06

The Linked Structure Family



- Doubly linked — next + prev
- Stack — last in, first out
- Queue — first in, first out

14.07



A FREE MYWEBUNIVERSITY COURSE

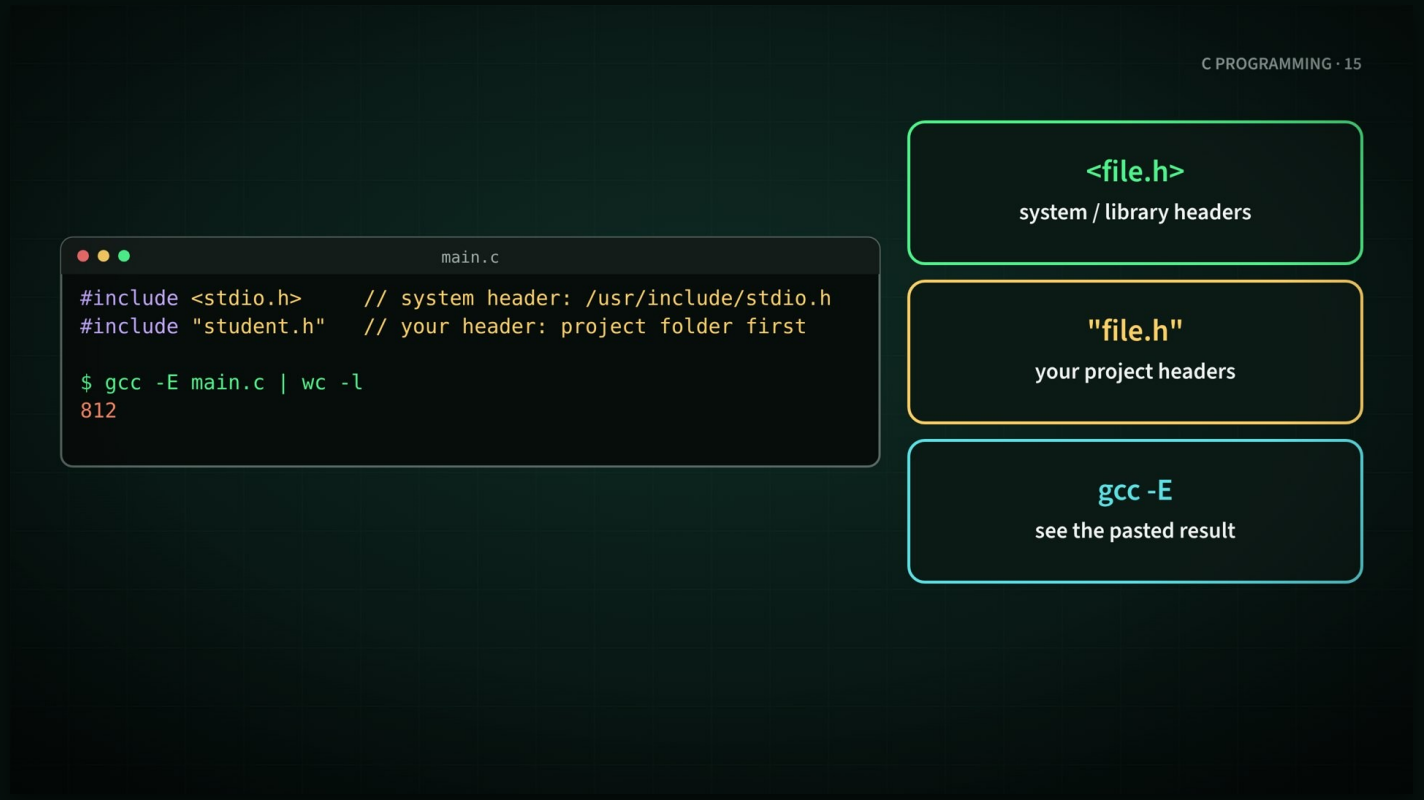
C PROGRAMMING

LESSON 15

The Preprocessor and Header Files

#include, #define, macros, #ifndef guards, multi-file programs and Makefiles

What #include Really Does



- <file.h> — system / library headers
- "file.h" — your project headers
- gcc -E — see the pasted result

15.01

What Is Inside stdio.h?



- A header like stdio.h does not contain the code of printf. It contains declarations: …
- The actual compiled code lives in the C library, libc, which the linker connects …

15.02

Common Standard Headers

Header	Gives you
stdio.h	printf scanf fgets fopen fprintf FILE EOF
stdlib.h	malloc free atoi strtol qsort rand exit
string.h	strlen strcpy strcmp memcpy memset
math.h	sqrt pow sin fabs (link with -lm)
ctype.h · time.h	isdigit toupper · time clock strftime
stdint.h · stdbool.h	int32_t uint8_t · bool true false

- Here are the headers you will use most. stdio.h for input and output, stdlib.h for memory, …
- math.h for sqrt, pow and sin; on Linux, link it with -lm. ctype.h for characters, time.h …

15.03

#define: Constants and Macros

macros.c

```c
#define MAX_STUDENTS 100
#define SQUARE_BAD(x)  x * x
#define SQUARE(x)      ((x) * (x))

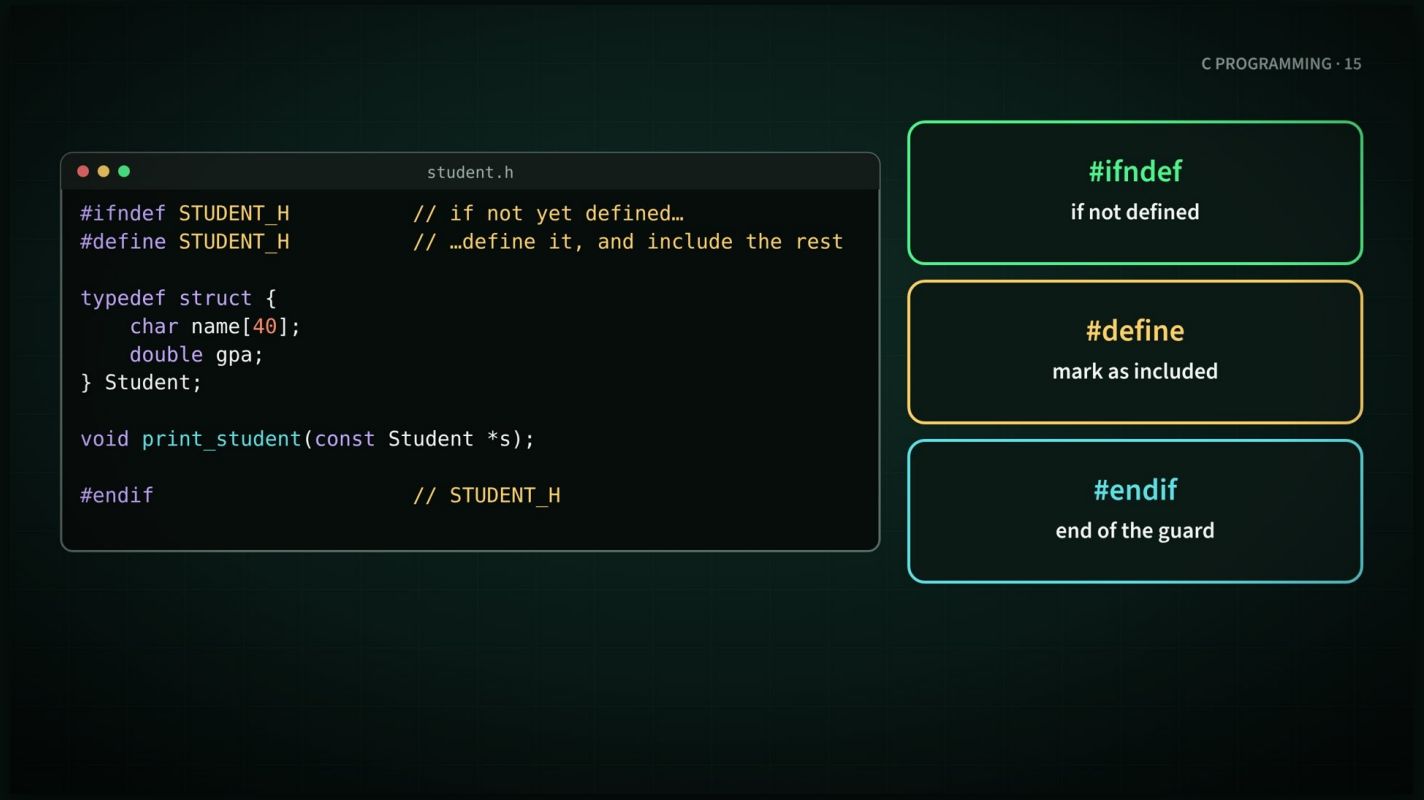
SQUARE_BAD(2 + 3)   // 2 + 3 * 2 + 3 = 11  ✗
SQUARE(2 + 3)       // ((2+3) * (2+3)) = 25 ✓
SQUARE(i++)         // i++ runs twice!      ✗

static inline int square(int x) { return x * x; }   // safer
```

- #define creates a name that the preprocessor replaces with text before compiling. It is …
- A macro with parameters looks like a function, but it is plain text replacement, and that …
- Even with parentheses, a macro may evaluate its argument twice. For most jobs, a const …

15.04

Include Guards: #ifndef



- #ifndef — if not defined
- #define — mark as included
- #endif — end of the guard

15.05

Conditional Compilation

debug.c

```
#ifdef DEBUG
    #define LOG(msg) fprintf(stderr, "[debug] %s\n", msg)
#else
    #define LOG(msg) ((void)0)
#endif

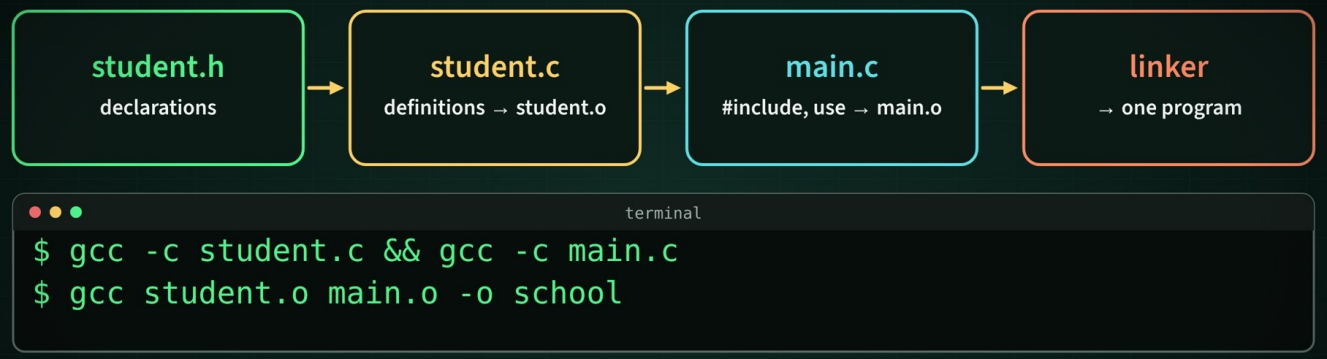
#ifdef _WIN32
    const char *os = "Windows";
#else
    const char *os = "Linux / macOS";
#endif

$ gcc -DDEBUG -g app.c -o app
```

- #ifdef, #if and #else let you compile different code for different situations, like extra …
- You can define a name from the command line with -D, so the same source builds a debug …

15.06

Multi-File Programs



- student.h — declarations
- student.c — definitions → student.o
- main.c — #include, use → main.o

15.07

Makefiles

Makefile

```
CC     = gcc
CFLAGS = -std=c17 -Wall -Wextra -g

school: main.o student.o
 $(CC) main.o student.o -o school

%.o: %.c student.h
 $(CC) $(CFLAGS) -c $<

clean:
 rm -f *.o school

$ make        # builds only what changed
```

- Typing compile commands gets tiring. make reads a Makefile that lists each target, the …
- make only rebuilds what changed, which saves a lot of time in big projects. Note: command …

15.08



A FREE MYWEBUNIVERSITY COURSE

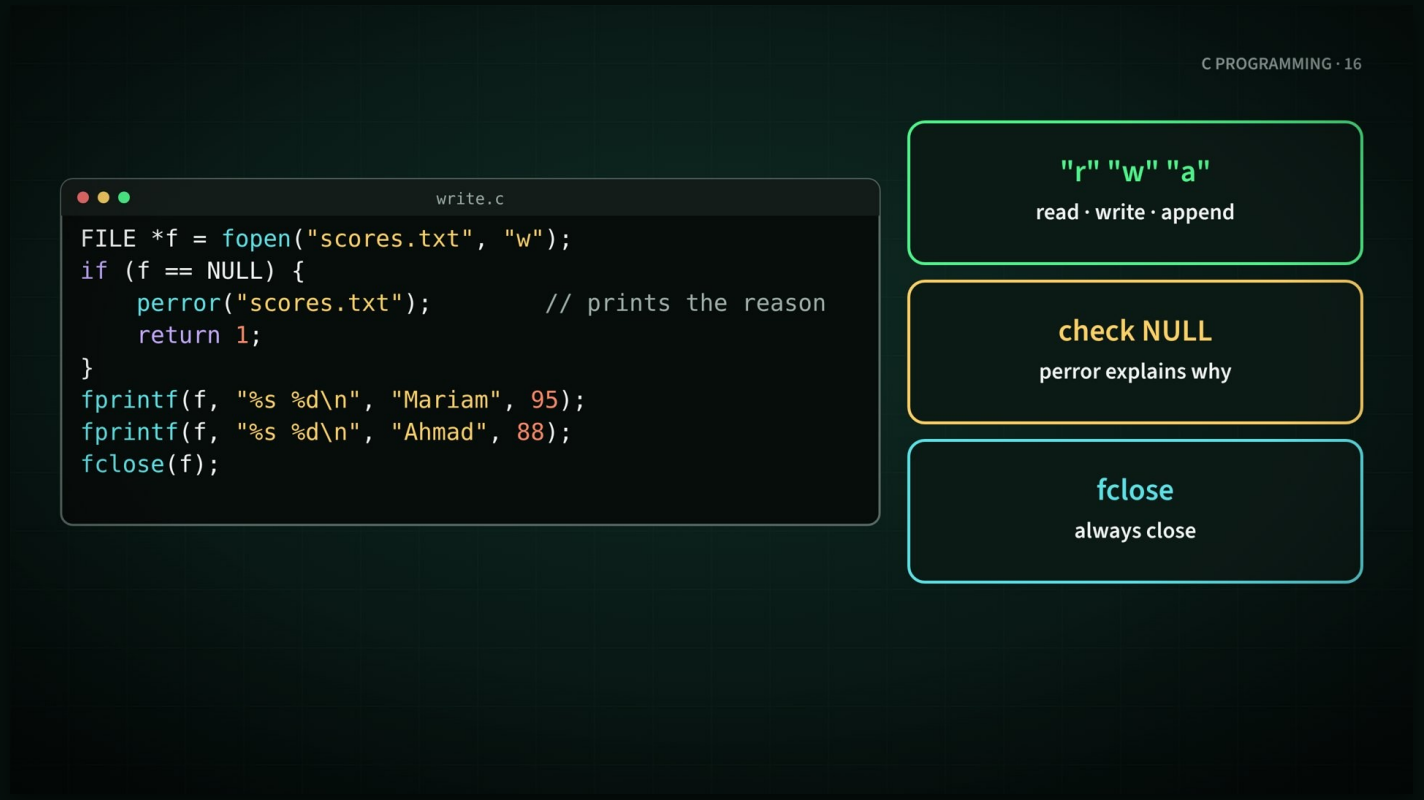
C PROGRAMMING

LESSON 16

The Standard Library and File I/O

Files, command-line arguments, qsort, random numbers and errors

Writing a File



- "r" "w" "a" — read · write · append
- check NULL — perror explains why
- fclose — always close

16.01

Reading a File

read.c

```c
FILE *f = fopen("scores.txt", "r");
if (!f) { perror("scores.txt"); return 1; }
char line[128], name[40];
int score;
while (fgets(line, sizeof line, f) != NULL) {
    if (sscanf(line, "%39s %d", name, &score) == 2) {
        printf("%-10s %3d\n", name, score);
    }
}
fclose(f);
```

- To read, open with mode r, and read one line at a time with fgets in a loop. fgets returns …
- To pull values out of a line, use sscanf, which works like scanf but reads from a string, …

16.02

Binary Files

binary.c

```c
Student list[30];

FILE *out = fopen("class.dat", "wb");
fwrite(list, sizeof(Student), 30, out);
fclose(out);

FILE *in = fopen("class.dat", "rb");
size_t n = fread(list, sizeof(Student), 30, in);
fclose(in);
```

- Text files are easy for people to read. Binary files store the raw bytes of your data, …
- Open with rb or wb, and use fwrite and fread to save and load whole structs or arrays in …

16.03

Command-Line Arguments

greet.c

```c
int main(int argc, char *argv[]) {
    if (argc < 2) {
        fprintf(stderr, "usage: %s NAME\n", argv[0]);
        return 1;
    }
    printf("Hello, %s!\n", argv[1]);
    return 0;
}
```

```
$ ./greet Wahid
Hello, Wahid!
```

- main can receive the words typed after the program name. argc is the number of arguments, …
- This is how tools like gcc and grep receive file names and options. Always check argc …

16.04

Useful stdlib.h Functions

sort.c

```c
int cmp_int(const void *a, const void *b) {
    int x = *(const int *)a, y = *(const int *)b;
    return (x > y) - (x < y);
}

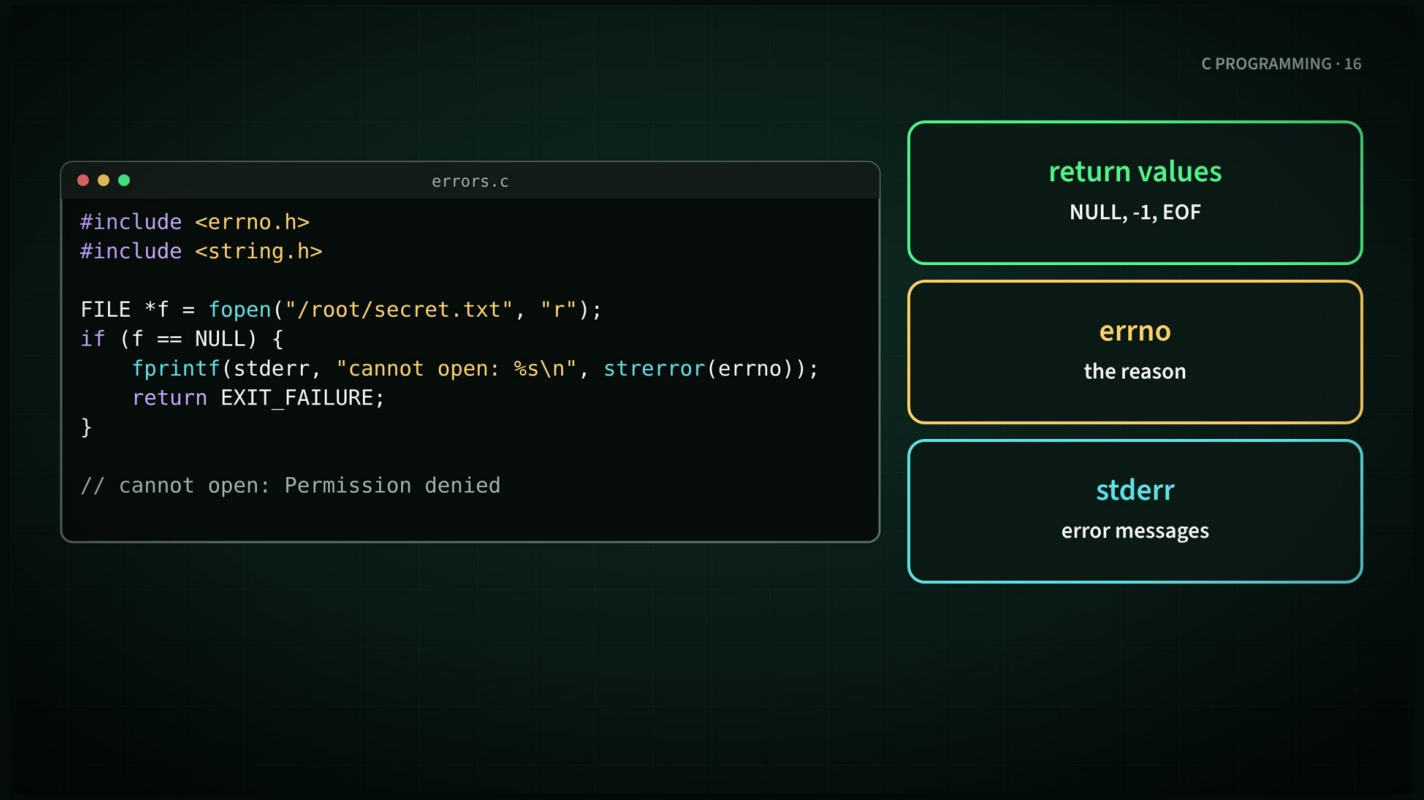
int v[] = {42, 7, 19, 3};
qsort(v, 4, sizeof v[0], cmp_int);     // 3 7 19 42

srand((unsigned)time(NULL));            // seed once
int dice = rand() % 6 + 1;              // 1..6
long n = strtol("123abc", &end, 10);   // 123, end → "abc"
```

- strtol converts text to a number safely and tells you where the number ended. atoi is …
- qsort sorts any array; you give it a comparison function, using the function pointers from …
- exit ends the program from anywhere, and EXIT_SUCCESS and EXIT_FAILURE are clear return codes.

16.05

Handling Errors



- return values — NULL, -1, EOF
- errno — the reason
- stderr — error messages

16.06



A FREE MYWEBUNIVERSITY COURSE

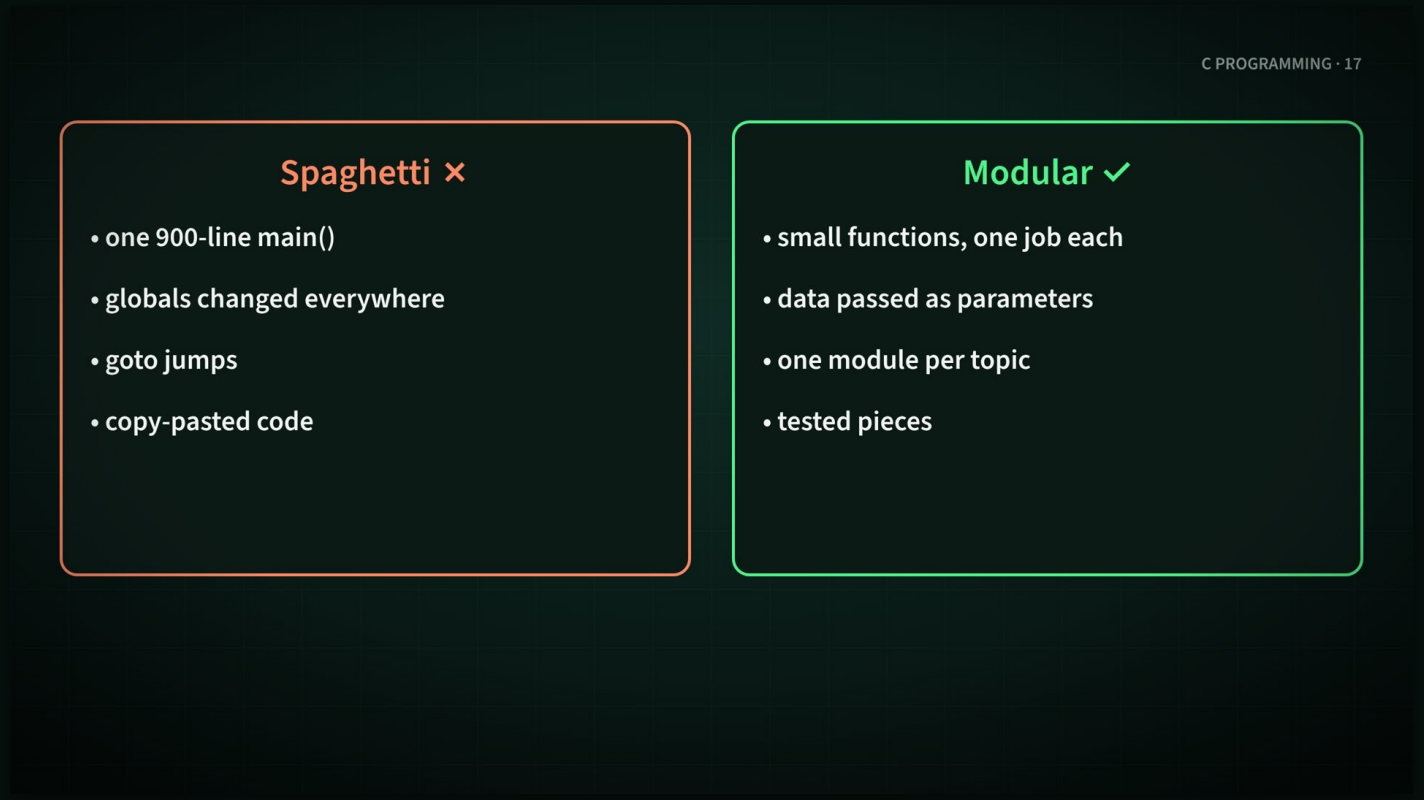
C PROGRAMMING

LESSON 17

Modular Design, C's Limits, and the Road to C++

Spaghetti code vs clean modules, and why C++ was born

Spaghetti Code



- Spaghetti code is code where everything is tangled together: one giant main, global ...
- It may work at first, but every change breaks something else, and nobody, including you ...

17.01

Modules in C

stack.h · stack.c

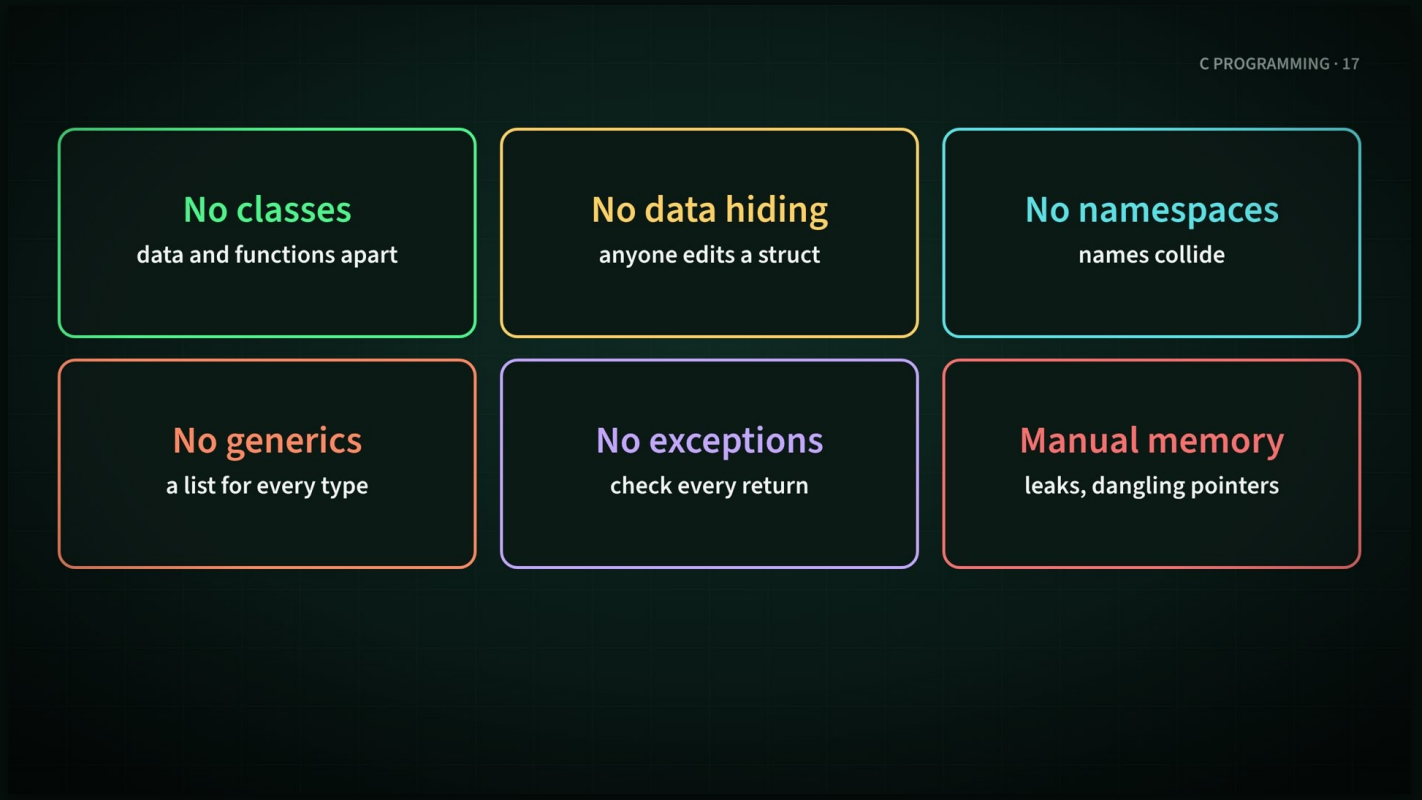
```c
// stack.h — the public interface
typedef struct Stack Stack;          // opaque: details hidden
Stack *stack_new(void);
void   stack_push(Stack *s, int v);
int    stack_pop(Stack *s);
void   stack_free(Stack *s);

// stack.c — the private implementation
struct Stack { int *items; int top, cap; };
static void grow(Stack *s) { /* private helper */ }
```

- A module in C is a pair of files: a header that is the public interface, and a .c file that …
- static functions and variables stay private to the module. Users only see the header, so …

17.02

What Standard C Is Missing



- No classes — data and functions apart
- No data hiding — anyone edits a struct
- No namespaces — names collide

17.03

Variables vs Objects

C: a variable

```
double balance = 100;

balance = -1000;   // nobody stops this
```

C++: an object

```
Account acc(100);

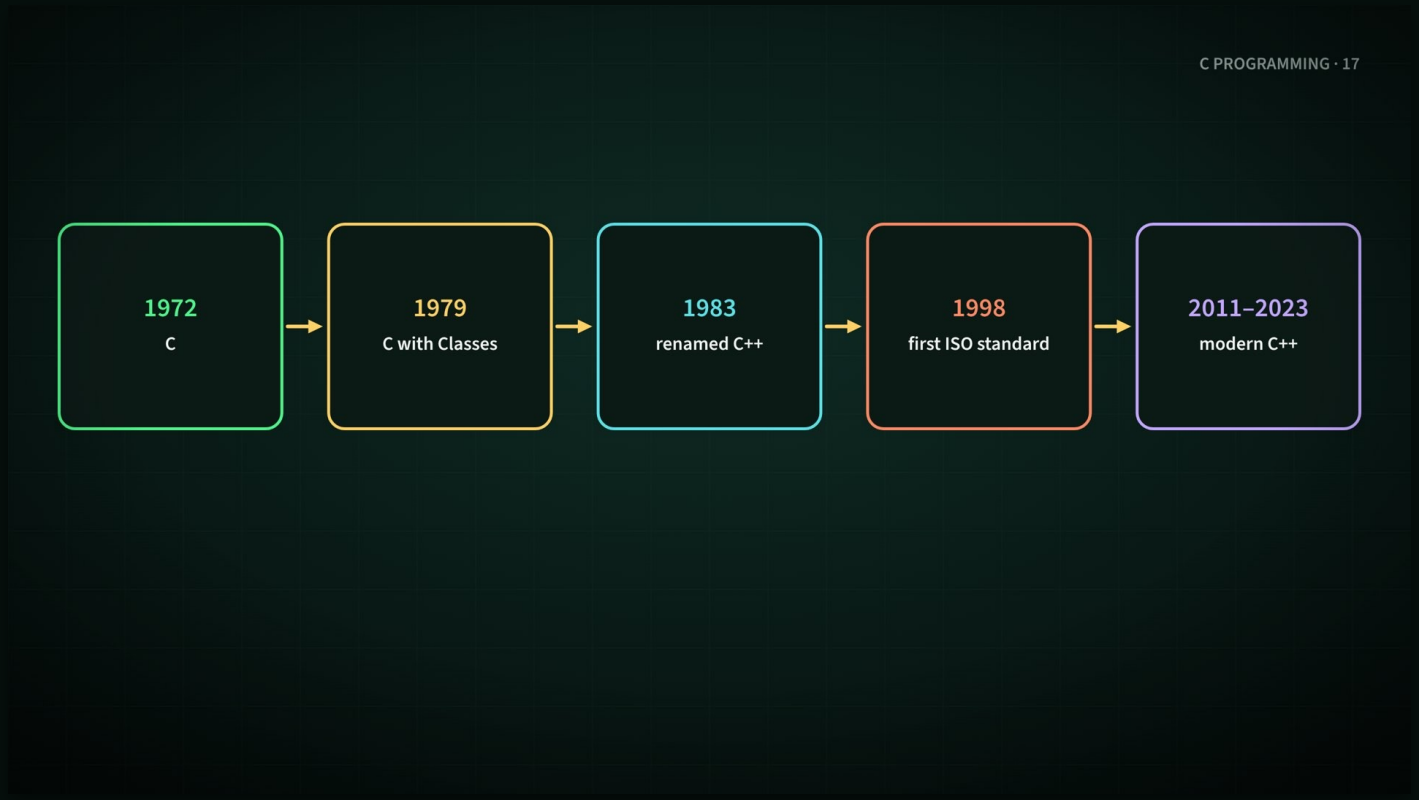
acc.deposit(-50);  // refused

acc.balance = 0;   // compile error
```

- In C, a variable is just data. A bank balance is a double that any code can set to minus …
- An object combines data with the functions that are allowed to change it, and it protects …

17.04

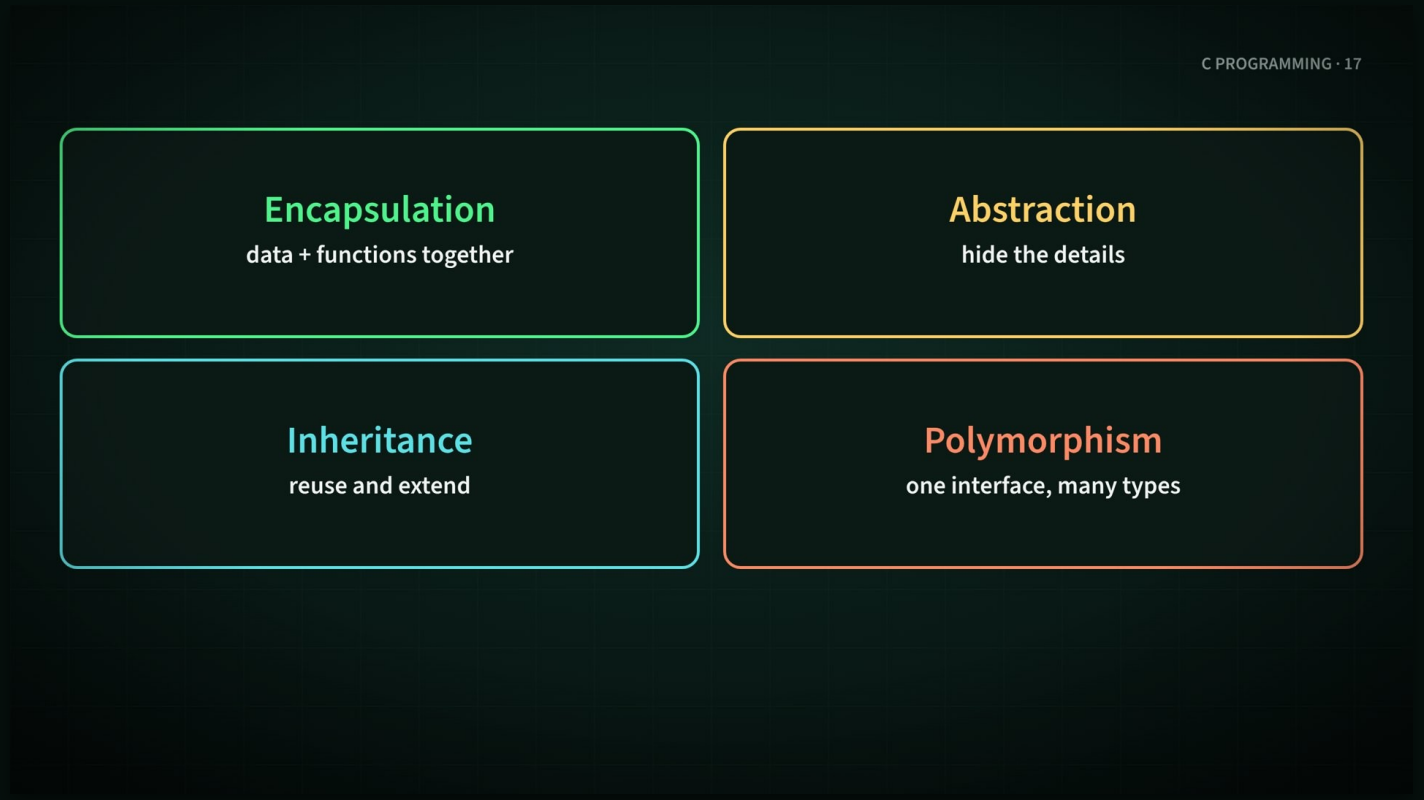
Why C++ Was Born



- 1972 — C
- 1979 — C with Classes
- 1983 — renamed C++

17.05

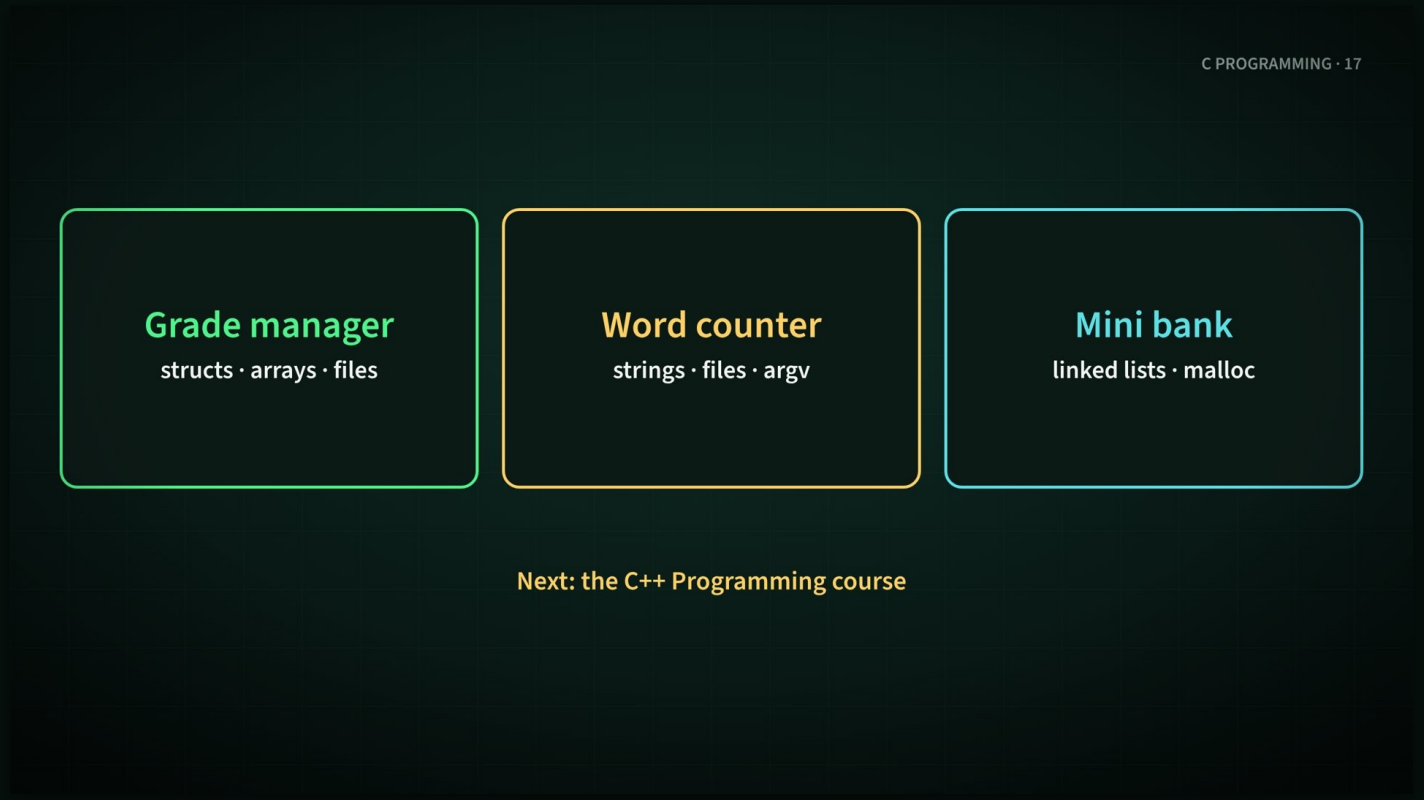
The Four Pillars Preview



- Encapsulation — data + functions together
- Abstraction — hide the details
- Inheritance — reuse and extend

17.06

Congratulations, and What Next



- Grade manager — structs · arrays · files
- Word counter — strings · files · argv
- Mini bank — linked lists · malloc

17.07

Sources and Further Reading (1/1)

1. Kernighan & Ritchie, The C Programming Language, 2nd ed., 1988
2. ISO/IEC 9899:2018 (C17) and 9899:2024 (C23) — the C standard
3. cppreference.com — C language and standard library reference
4. GCC manual — gcc.gnu.org/onlinedocs · GDB manual — sourceware.org/gdb
5. MyWebUniversity — free C and Linux courses

Thank you for learning with us

MyWebUniversity.com

Learn it all @MyWebUniversity

Free technical education since 2000 — English and Dari