

COMPSCI 1XC3

Week 5-2 — Introduction to C

Goal: understand C's compilation model and basic types

What is C?

- A compiled, statically typed programming language
- You write source files like `main.c`
- You compile them into executables (machine code)
- Lower-level than Python — you manage memory and types explicitly

A Typical C Program

```
#include <stdio.h> // Header for input/output functions

int main() {      // Program entry point
    printf("Hello world!\n");
    return 0;     // 0 = success
}
```

Key parts:

- `#include <stdio.h>` — import header for `printf`
- `int main()` — every C program starts here
- `return 0;` — exit code (0 = success, non-zero = error)
- `;` — statement terminator (required!)

Compile + Run with gcc

```
gcc main.c          # Compile → creates a.out
./a.out             # Run the executable

gcc main.c -o myProgram # Specify output name
gcc -Wall main.c       # Show all warnings (recommended!)
```

Always use `-Wall` to catch potential issues early

Variables in C

```
int age = 21;      // Integer type
float grade = 87.5; // Floating point type
char letter = 'A'; // Character type
```

Breaking it down:

- `int` — data type
- `age` — variable name
- `= 21` — initialization (giving it a value)
- `;` — statement terminator

Unlike Python, semicolons are required, not line breaks

Variable Naming Rules

Valid names:

- `x`, `count`, `total_score`, `item1`, `_hidden`

Invalid names:

- `1var` (starts with number)
- `my var` (contains space)
- `@name`, `$price` (special characters)

Avoid reserved keywords:

- `int`, `float`, `if`, `return`, `while`, etc.

Common conventions:

- `camelCase` or `snake_case`

Basic Data Types

```
int        // Integer: 42, -17, 0
float      // Decimal: 3.14, -0.5, 100.0
char       // Single character: 'A', 'z', '7'
double     // Larger/more precise decimals
```

Type specifiers (modify size/range):

```
unsigned int count;    // Only positive (0 and up)
short age;             // Smaller integer
long population;       // Larger integer
```

Always Initialize Variables

```
int x;           // Contains random garbage value!  
int y = 0;       // Safe – explicitly set to 0  
  
// Multiple variables at once  
float price = 19.99, tax = 0.13;
```

Why this matters:

- Uninitialized variables have unpredictable values
- Can cause strange bugs that are hard to find
- Always give variables a starting value

printf Format Specifiers

```
int age = 21;
float gpa = 3.87;
char grade = 'A';

printf("Age: %d\n", age);           // %d for int
printf("GPA: %.2f\n", gpa);        // %.2f for 2 decimals
printf("Grade: %c\n", grade);      // %c for char
```

Common specifiers:

- `%d` — int
- `%f` — float/double
- `%c` — char
- `%lu` — sizeof result (unsigned long)

Format Specifier Details

```
printf("Number: %d\n", 42);           // Basic integer
printf("Decimal: %.2f\n", 3.14159);   // 2 decimal places → 3.14
printf("Width: %5d\n", 42);           // Min width 5 → "   42"
printf("Percent: %d%%\n", 95);        // Print % symbol → "95%"
```

Precision control:

- `%.1f` — 1 decimal place
- `%.3f` — 3 decimal places
- `%5d` — minimum width of 5 characters

Escape sequences:

- `\n` — newline
- `%%` — literal percent sign

sizeof and Type Sizes

```
printf("%lu bytes\n", sizeof(int));    // Typically 4
printf("%lu bytes\n", sizeof(char));   // Always 1
printf("%lu bytes\n", sizeof(float));  // Typically 4
printf("%lu bytes\n", sizeof(double)); // Typically 8
```

Common sizes:

- `char` = 1 byte
- `int` = 4 bytes
- `float` = 4 bytes
- `double` = 8 bytes

Why sizes matter: They determine the range of values you can store

Constants in C (4 Ways)

1. Literals (direct values):

```
42          // int literal
3.14f       // float literal (note the 'f')
'A'         // char literal
```

2. #define (preprocessor macro):

```
#define PI 3.14159 // No semicolon!
#define MAX 100
```

3. const (typed constant):

```
const int SIZE = 500; // Has a type
const float RATE = 0.13;
// SIZE = 600; // Compile error - can't change
```

enum Constants

```
enum Day {MON, TUE, WED, THU, FRI};  
enum Day today = WED;  
printf("%d\n", today); // Prints: 2 (starts at 0)
```

Custom values:

```
enum Grade {FAIL=0, PASS=50, EXCELLENT=90};  
enum Grade myGrade = PASS;  
printf("%d\n", myGrade); // Prints: 50
```

- Values start at 0 by default
- Each increments by 1
- Can specify custom values

Type Casting

Converting between types:

```
int x = 5, y = 2;

// Without cast – integer division
int result1 = x / y;           // 2 (no decimals)

// With cast – float division
float result2 = (float)x / y;  // 2.5 (decimals!)

// Casting floats to ints (truncates)
int rounded = (int)3.7;        // 3 (not 4!)
```

When you need it:

- Getting decimal results from integer division
- Converting between numeric types

Checking Type Limits

```
#include <limits.h> // Required for limit constants

printf("int range: [%d, %d]\n", INT_MIN, INT_MAX);
printf("char range: [%d, %d]\n", CHAR_MIN, CHAR_MAX);
printf("short range: [%d, %d]\n", SHRT_MIN, SHRT_MAX);
```

Useful predefined constants:

- `INT_MIN`, `INT_MAX` — int limits
- `CHAR_MIN`, `CHAR_MAX` — char limits
- `SHRT_MIN`, `SHRT_MAX` — short limits
- `UINT_MAX` — unsigned int max

TA Check-Off — Task 1

Basic Compilation

Create a program that:

- Prints three lines: your name, your program, and your year
- Compiles with no warnings using `-Wall`

Show the TA:

- Source code
- Compilation command
- Program output

TA Check-Off — Task 2

Temperature Converter

Write a program that converts Fahrenheit to Celsius:

- Store a Fahrenheit temperature as an integer (e.g., 68)
- Print both temperatures with proper labels and formatting

Challenge: Make sure you get decimal precision in the result

Hint: Think about integer division vs float division

TA Check-Off — Task 3

Create a program that displays:

1. A constant for your student ID (use `#define`)
2. A constant for the current year (use `const`)
3. The size in bytes of: `char` , `int` , `float` , `double`
4. The range (min and max) of `int` , `char` , and `short` types
5. Print your student ID and the max value of `short` — does your ID fit?

Think about:

- How do you determine if a value fits in a type?
- Compare your student ID number to `SHRT_MAX`

Required: Include `<limits.h>` for type ranges