CS50 Week 1 - Introduction to Computer Science and Programming

Animesh Garg and ChatGPT

1 Introduction

This document contains a comprehensive set of notes from **Week 1 of CS50**, covering the transition from Scratch to the C programming language, the command-line environment, essential syntax, functions, conditionals, loops, variables, and considerations of correctness, design, and style. It also highlights important pitfalls such as integer overflow and floating-point imprecision.

Contents

2 Transition from Scratch to C

2.1 Key Ideas

- In Week 0, you used **Scratch** blocks (functions, loops, conditionals, variables) to build programs.
- These same concepts exist in **C** (and most programming languages); only the *syntax* differs.
- A **compiler** converts human-readable **source code** into **machine code** (zeros and ones).

2.2 Environment: Visual Studio Code (VS Code) for CS50

- Access via `cs50.dev` (GitHub Codespaces).
- Preconfigured with necessary software (*e.g.*, `make`, `gcc`, `clang`).
- Provides:
  - A **CLI** (command-line interface) at the bottom.
  - A file explorer on the left.
  - A text editor in the center.
- Essential CLI commands:

```
ls           // list files
cd <dir>     // change directory
mkdir <dir>  // create new directory
mv <old> <new>   // move/rename
cp <old> <new>   // copy
rm <file>        // remove/delete
rmdir <dir>      // remove empty directory
code <filename>  // open file in VS Code
```

- `Control-C` breaks or interrupts a running program in an infinite loop.

3 `Hello, world` in C

3.1 Basic Steps

1. **Create** a file:

```
code hello.c
```

2. **Write the source code** (in `hello.c`):

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
}
```

3. **Compile** the program:

```
make hello
```

4. **Run** the program:

```
./hello
```

Key points:

- `#include <stdio.h>` gives access to `printf`.
- `\n` is a newline escape sequence.
- Statements end with semicolons; `main(void)` is the entry point.

4 Header Files and Libraries

- **Header Files** (like `stdio.h`, `cs50.h`) include function declarations.
- `#include <stdio.h>` → you can use `printf`, `scanf`, etc.
- `#include <cs50.h>` → use `get_string`, `get_int`, etc.
- The **CS50 Library** provides training-wheel functions for simplified input.

5 Printing, Placeholders, and Escape Sequences

5.1 Using `printf` with Format Codes

- `printf("hello, %s\n", name);` → `%s` for a string.
- `printf("%i\n", x);` → `%i` for an integer.
- `printf("%.2f\n", f);` → `%.2f` for floating-point with 2 decimals.

5.2 Common Escape Sequences

- `\n` → newline
- `\r` → carriage return
- `\"` → print double quote
- `\\` → print backslash

6 Data Types in C

- `int`: Integers (32 bits).
- `long`: Larger integers (64 bits).
- `float`: Real numbers (32 bits).
- `double`: Real numbers (64 bits).
- `char`: Single characters (e.g., `'y'`).
- `string`: Text (CS50-specific alias).
- `bool`: Boolean (`true`, `false`) in CS50.

7 Variables and Operators

7.1 Declaring Variables

```
int x;      // declares an int x
x = 5;      // assigns 5 to x
int y = 10; // declares & assigns 10
```

7.2 Arithmetic Operators

- `+` addition
- `-` subtraction
- `*` multiplication
- `/` division
- `%` remainder (modulus)

7.3 Increment / Decrement

- `i++` → `i = i + 1`
- `i--` → `i = i - 1`
- `i += 2` → `i = i + 2`
- `i *= 2` → `i = i * 2`

8 Conditionals

- **If / Else If / Else:**

```
if (x < y)
{
    ...
}
else if (x > y)
{
    ...
}
else
{
    ...
}
```

- **Comparison Operators**:

```
==  // equals
!=  // not equals
<   // less than
>   // greater than
<=  // less/equal
>=  // greater/equal
```

- **Logical Operators**:

```
&&  // AND
||  // OR
!   // NOT
```

- Example (`agree.c`):

```
char c = get_char("Do you agree? ");
if (c == 'y' || c == 'Y')
{
    printf("Agreed.\n");
}
else if (c == 'n' || c == 'N')
{
    printf("Not agreed.\n");
}
```

9 Loops

9.1 `while` Loop

```
while (condition)
```

```
{
    // repeatedly do something
}
```

Example (counting down):

```
int i = 3;
while (i > 0)
{
    printf("meow\n");
    i--;
}
```

9.2 `for` Loop

```
for (initialization; condition; update)
{
    // repeat
}
```

Example (three meows):

```
for (int i = 0; i < 3; i++)
{
    printf("meow\n");
}
```

9.3 `do while` Loop

```
int n;
do
{
    n = get_int("Number: ");
}
while (n < 1);
```

Ensures at least one execution before checking the condition.

9.4 `break`

- Exits the nearest loop.
- `Control-C` forcibly interrupts a running program in the terminal.

10 Functions in C

10.1 Creating Your Own Function

```
void meow(void)
{
    printf("meow\n");
}
```

Prototype at the top:

```
void meow(void);
```

Definition (body) at the bottom. Then **call** from `main`:

```
int main(void)
{
    meow();
}
```

10.2 Return Types and Inputs

- If a function should return an integer, you write `int` instead of `void`.
- If it takes parameters, put them in parentheses:

  ```
  void meow(int n)
  {
      // ...
  }
  ```

- Then call `meow(3)`, for example, to meow 3 times.

11 Overflow and Floating-Point Imprecision

11.1 Integer Overflow

- `int` uses 32 bits: max $\approx$ 2 billion if signed.
- Exceed that, it wraps around (negative or zero).
- `long` = 64 bits, but still finite.
- Real-world examples:
  - Old video games (Pac-Man #256 glitch).
  - Boeing 787 bug after a certain number of days.
  - Year 2038 problem (32-bit UNIX time).

11.2 Floating-Point Imprecision

- `float` and `double` also have finite bits.
- 1/3 might show up as 0.33333334326... not infinite `0.3333...`.
- Must be mindful of rounding errors in real-number arithmetic.

12 Correctness, Design, and Style

- **Correctness** $\rightarrow$ Does it work as intended? Use `check50` for checks.
- **Design** $\rightarrow$ Is the code efficient, not repetitive, logically neat? `design50` can help.
- **Style** $\rightarrow$ Aesthetics (indentation, naming, clarity). Use `style50`.

13 Mario Examples (Nested Loops)

13.1 Single Row of Question Marks

```
for (int i = 0; i < 4; i++)
{
    printf("?");
}
printf("\n");
```

13.2 Single Column of Bricks

```
for (int i = 0; i < 3; i++)
{
    printf("#\n");
}
```

13.3 3-by-3 Grid of Bricks

```
for (int row = 0; row < 3; row++)
{
    for (int col = 0; col < 3; col++)
    {
        printf("#");
    }
    printf("\n");
}
```

Alternatively:

```
const int n = 3;
for (int i = 0; i < n; i++)
{
    for (int j = 0; j < n; j++)
    {
        printf("#");
    }
    printf("\n");
}
```

14 Comments

- Use `//` for single-line remarks.
- Provide clarity on *why* or *how* something is done.

15 Summary of Key Takeaways

- **Write, Compile, Run**:

1. `code filename.c`
2. `make filename`
3. `./filename`

- **Building Blocks**: Functions, Conditionals, Loops, Variables.
- **Avoid Common Mistakes**:
  - Missing semicolons/braces.
  - Wrong format specifiers.
  - Overflow and imprecision.
- **Good Code**:
  - **Correctness**: functionally correct.
  - **Design**: efficient, not repetitive.
  - **Style**: readable, commented, consistent.
- **Practice** with problem set tasks (e.g., printing pyramids in Mario).
- Tools: `check50`, `design50`, `style50`.

16 Further Reading and Practice

- **CS50 Manual Pages**: `https://manual.cs50.io/`
- For formatting: `Control-L` to clear terminal screen visually.
- Real-world documentation for `printf`, `scanf`, etc. (e.g., `man printf` on Linux).

17 Final Words

By the end of Week 1, you should feel comfortable with:

- Navigating VS Code and the command line.
- Translating Scratch concepts (loops, conditionals, variables, functions) into C.
- Compiling and running programs (`make`, `./program`).
- Recognizing pitfalls like **integer overflow** and **floating-point imprecision**.
- Evaluating code's **correctness**, **design**, and **style**.

Use these notes, the examples, and the companion tools to build out your first official assignments (like Mario) and continue learning fundamental C programming skills.