

# CS100 Lecture 1

Warmup for C

# Contents

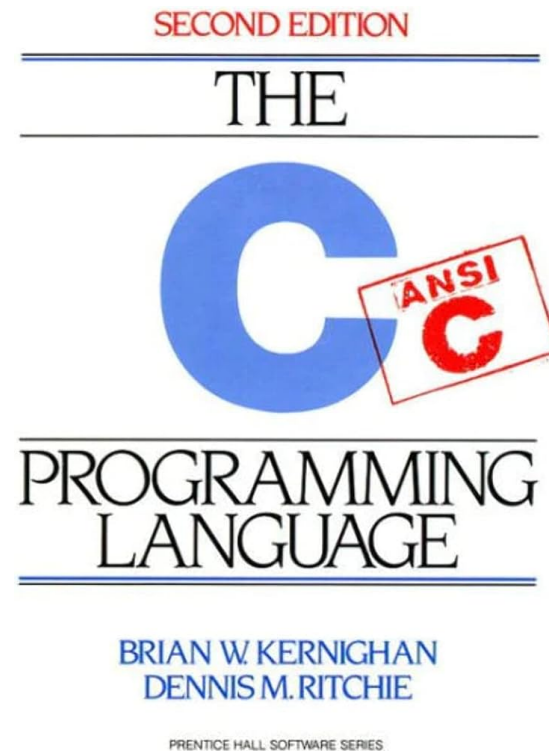
- Brief history of C
- The first C program
  - Functions (basic) and the `main` function
  - `scanf` and `printf`

# Brief history of C

# The UNIX operating system and C

- In 1969, a small group of AT&T Bell Labs led by **Ken Thompson** and **Dennis Ritchie** began to develop UNIX.
- In 1973, UNIX kernel was rewritten in C.
- From 1969 to 1973, Dennis Ritchie developed C in Bell Labs.
- In 1978, Kernighan and Ritchie published **the K&R book**: *The C Programming Language*.

# Dennis Ritchie and the K&R book



- Dennis M. Ritchie (1941 - 2011)
  - The inventor of C
  - Co-inventor of UNIX
  - ACM Turing Award (1983) with Ken Thompson for UNIX

# Standardization of C

- "K&R C": Informal specification (the K&R book)
- ANSI C: Known as "C89"
  - American National Standards Institute
  - Came out in 1989
- ISO C standard: ISO/IEC 9899
  - International Organization for Standardization
  - First version: "C90" in 1990, the same standard as C89 with only formatting changes
  - C99 (1999), C11 (2011), C17 (2017/2018), C23 (2023)

# The first C program

# Hello World

```
#include <stdio.h>

int main(void) {
    printf("hello world\n");
    return 0;
}
```

- Save the code as `hello.c` .
- `gcc hello.c -o hello.exe`  $\Rightarrow$  generates `hello.exe`
- `.\hello.exe`  $\Rightarrow$  prints `hello world` , with a newline at the end.



# The `main` function

Every C program coded to run in a hosted execution environment contains the definition of a **function** named `main`, which is the designated start of the program.

```
// Other things (functions, structures, ...), if any ...  
  
int main(void) {  
    // The program starts here.  
    statement_1;  
    statement_2;  
    // ...  
    statement_n;  
}
```

\* What is a function?

## A function in C

A function in mathematics:  $f : S \mapsto T$ , accepts some **arguments** and **returns** some value.

Example:  $f(x) = x^2, x \in \mathbb{Z}$  accepts an integer argument, and **returns** its square.

Write it in C:

```
int f(int x) {  
    return x * x;  
}
```

## A function in C

Example:  $f(x, y) = x + y, x, y \in \mathbb{R}$

Write it in C:

```
double f(double x, double y) { // Two arguments
    return x + y;
}
```

`double` : double-precision floating-point number  $\Rightarrow$  will be covered in later lectures.

# A function in C

Syntax: `ReturnType FunctionName(Parameters) { FunctionBody }`

`FunctionBody` can also contain more complex statements:

```
int max(int a, int b) {  
    if (a < b)  
        return b;  
    else  
        return a;  
}
```

`if` statement  $\Rightarrow$  will be covered in later lectures.

# A function in C

Syntax: `ReturnType FunctionName(Parameters) { FunctionBody }`

A function can have no arguments. To define such a function, write `void` in

Parameters :

```
int always42(void) {  
    return 42;  
}
```

We will introduce more on **functions** in later lectures.

# The `main` function

Every C program coded to run in a hosted execution environment contains the definition of a **function** named `main`, which is the designated start of the program.

According to the standard, the `main` function **must** has one of the following signatures:

1. `int main(void) { ... }`
2. `int main(int argc, char *argv[]) { ... }`
3. `/* another implementation-defined signature */`

For now, we only use the first one: `int main(void) { ... }`.

"signature": consisting of the return-type, parameters, and some other possible information

## The `main` function

```
int main(void) {  
    printf("hello world\n");  
    return 0;  
}
```

The return value of `main`: Indicates whether the program exits successfully.

A program exits successfully if and only if its `main` function returns `0`.

# The `main` function

A program exits successfully if and only if its `main` function returns `0`.

You may also see this somewhere else:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

For help, type "help".
Type "apropos word" to search for commands related to "word".
Warning: Debuggee TargetArchitecture not detected, assuming x86_64.
=cmd-param-changed,param="pagination",value="off"
Stopped due to shared library event (no libraries added or removed)
Loaded '/lib64/ld-linux-x86-64.so.2'. Symbols loaded.
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.so.1".

Breakpoint 1, main () at /home/gkxx/Courses/CS100/tmp/a.c:4
4      printf("hello world");
Loaded '/lib/x86_64-linux-gnu/libc.so.6'. Symbols loaded.
[Inferior 1 (process 6380) exited normally]
The program '/home/gkxx/Courses/CS100/tmp/a' has exited with code 0 (0x00000000).
```



## The `main` function

A program exits successfully if and only if its `main` function returns `0`.

You may also see this somewhere else:

```
Build finished with error(s).
```

```
* The terminal process failed to launch (exit code: -1).  
* Terminal will be reused by tasks, press any key to close it.
```

## The `main` function

It is ok to omit `return 0;` in `main` (*but not in other functions*):

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

According to the standard:

If the return type is compatible with `int` and control reaches the terminating `}`, the value returned to the environment is the same as if executing `return 0;`.

## printf

Declared in the standard library header file `<stdio.h>` .

- That's why we need `#include <stdio.h>` in the beginning.

Writes something to **the standard output**.

```
printf("hello world\n");
```

- Prints `hello world` , with a newline `\n` at the end.
- Try this out: `printf("hello\nworld\n");`

# Output vs return

```
int main(void) {  
    printf("hello world\n");  
    return 0;  
}
```

- What is the output of the program?
- What is the return value of `main` ?

# Output vs return

```
int main(void) {  
    printf("hello world\n");  
    return 0;  
}
```

- What is the output of the program?  $\Rightarrow$  `hello world` with an ending newline.
- What is the return value of `main` ?  $\Rightarrow$  `0` .

# The "A+B" problem

Reads two integers from input (separated by whitespaces), and prints the sum of them.

```
#include <stdio.h>

int main(void) {
    int a, b; // declares two variables of type "int", named "a" and "b".
    scanf("%d%d", &a, &b);
    printf("%d\n", a + b);
    return 0;
}
```

# The "A+B" problem

Reads two integers from input (separated by whitespaces), and prints the sum of them.



The image shows a code editor window with a dark theme. At the top, there's a tab labeled 'a.c' with a close button. Below the tab, the code is written in C. It starts with a preprocessor directive to include 'stdio.h', followed by the 'main' function. Inside 'main', two integers 'a' and 'b' are declared. The 'scanf' function is used to read two integers from standard input, separated by a space. Then, the 'printf' function is used to print the sum of 'a' and 'b' followed by a newline. Finally, 'return 0;' is used to exit the program. Below the code editor, there's a terminal window with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', 'PORTS', and 'GITLENS'. The 'TERMINAL' tab is active, showing the compilation and execution of the program. The first command is 'gcc a.c -o a', which compiles the program into an executable named 'a'. The second command is './a', which runs the program. The output of the program is shown as '30 42' on the first line and '72' on the second line.

```
C a.c x
tmp > C a.c > main(void)
1  #include <stdio.h>
2
3  int main(void) {
4      int a, b;
5      scanf("%d%d", &a, &b);
6      printf("%d\n", a + b);
7      return 0;
8  }

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS

● gkxx@nodiscard-gkxx ~/C/C/tmp> gcc a.c -o a
● gkxx@nodiscard-gkxx ~/C/C/tmp> ./a
30 42
72
```

## scanf

Also declared in `<stdio.h>` .

Reads something from **the standard input**.

Example: Reads two integers from the standard input, separated by whitespaces.

```
scanf("%d%d", &a, &b);
```

- `%d` : Indicates that an integer is expected, and will be stored into an `int` variable.
- `&` : The **address-of** operator  $\Rightarrow$  will be covered in later lectures.

For now, just remember to add `&` when passing things to `scanf` .



## scanf

```
scanf("%d%d", &a, &b);
```

How should these two integers be separated? Try it out:

```
● gkxx@nodiscard-gkxx ~/C/C/tmp> ./a
30 42
72
● gkxx@nodiscard-gkxx ~/C/C/tmp> ./a
30      42
72
● gkxx@nodiscard-gkxx ~/C/C/tmp> ./a
30
42
72
```

## scanf

```
scanf("%d%d", &a, &b);
```

`%d` will skip any leading whitespaces.

- "whitespace" refers to the character that looks "blank": space `' '`, newline `'\n'`, tab `'\t'`, etc.

More on the rules related to `scanf` will be covered in recitations.

## `printf` printing an integer

Given `a = 30`, `b = 42` as input:

- `printf("%d\n", a + b);`

⇒ prints `72`, with a newline in the end.

- `printf("%d + %d equals %d\n", a, b, a + b);`

What is the output?

## `printf` printing an integer

Given `a = 30`, `b = 42` as input:

- `printf("%d\n", a + b);`

⇒ prints `72`, with a newline in the end.

- `printf("%d + %d equals %d\n", a, b, a + b);`

⇒ prints `30 + 42 equals 72`, with a newline in the end.