

# Introduction to C: History

- In **1970**, **Ken Thompson** of Bell Labs designed the “**B**” language for the first Unix system on the DEC PDP-7.
- Soon after, the “**C**” language was designed by **Dennis Ritchie** at Bell Labs for implementing Unix on the PDP-11.
- In **1978**, **Brian Kernighan** and **Dennis Ritchie** published the first edition of “*The C Programming Language*” (K&R).
- About **1981**, **Bjarne Stroustrup** of Bell Labs defined an extended C language that eventually evolved into **C++**.
- In **1983** ANSI established a committee to define a standard C.
- In late **1988** the **ANSI C** standard was submitted for approval. The language includes some of Stroustrup’s extensions.
- **Today** C is no longer changing, but C++ continues to evolve separately from ANSI C.

# ANSI C versus K&R C

There are two main dialects of C that you might encounter: *K&R* and *ANSI*.

- *K&R* (Kernighan and Ritchie) C is the old version of the C language, defined about 1978.
- *ANSI* C is the new standard, defined in 1988.
- ANSI C includes significant extensions, primarily w.r.t type checking, that make it an improvement over K&R C.
- Most new C programming is now done in ANSI C.

We will focus exclusively on ANSI C. K&R C is only important if you work on legacy code.

# C and Java

C and Java look superficially alike, because Java adopted C's syntax. But ...

*C and Java are very different!*

C and Java have much less in common than you might expect from a cursory glance.

# Java-to-C: Forget Everything!

To move from Java to C, you first have to forget everything you learned from Java – really!

- Forget *objects*, forget *classes*, forget *methods*.
- Forget *packages*, forget *interfaces*, forget *exceptions*.
- Forget `new`, forget *garbage collection*.
- Forget array *bounds-checking*.
- Forget the *JVM* and on-demand *class loading*.
- Forget *platform independence*.

# Java-to-C: Add Low-level Features

C adds low-level, potentially dangerous features that Java does not have:

- Add *macros*.
- Add *pointers*.
- Add *structures*.
- Add `malloc` and `free`.
- Add “manual” *linking*.

C requires a disciplined programming style to avoid pitfalls and simulate facilities that Java provides for free.

# C is “High-level Machine Language”

- C began as “high-level machine language” for the DEC PDP-11 computer.
- Many C constructs reflect closely the PDP-11 instruction set.
- Even today, the underlying machine “shows through” to a great extent in C.
- The programmer must understand the underlying machine and pay close attention to low-level considerations.

C philosophy: “Programmers know what they are doing. They can do whatever they want, as long as they say it explicitly.”

*This can lead to trouble...*

# A Simple C Program

Here is the classic “Hello world!” program in C:

```
#include <stdio.h>

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

# A Simple C Program: Notes

- C programs have *functions*, not methods.
- Every C program must have a `main()` function.
- `printf()` is a *C library function*.  
(it is not a C language construct)
- `#include` is a *preprocessor directive*.  
(handled by the *C preprocessor*)

# Compiling a C Program

The C compiler is traditionally invoked as a “command-line” tool:

```
$ gcc hello_world.c
```

(the \$ is the “command shell” prompt).

- On Unix-like systems, this produces the executable file “a.out” .  
On Windows, it produces “a.exe” .
- The executable is launched by giving its name as a command:

```
$ a.out  
Hello world!
```