The C Programming Language

Lecture 2

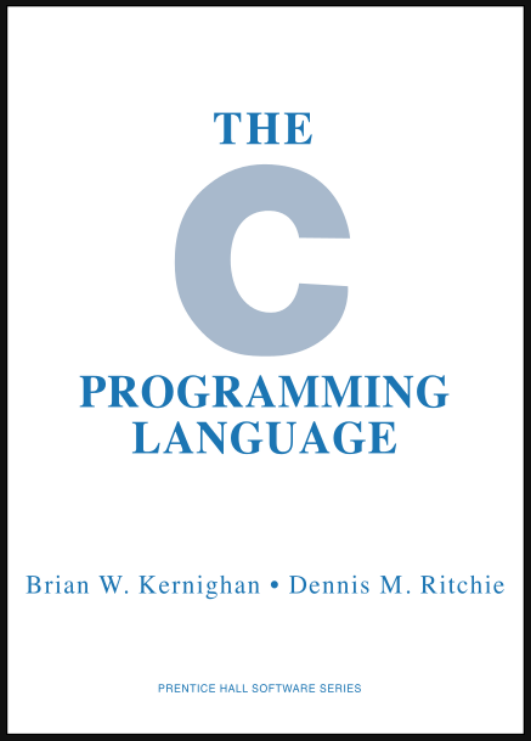
Christian A. Pagot

Universidade Federal da Paraíba
Centro de Informática

What is C?

C is a **imperative, general-purpose** programming language, developed in 1973 at AT&T Bell Labs by **Ken Thompson** (left) and **Dennis Ritchie** (right):

unknown

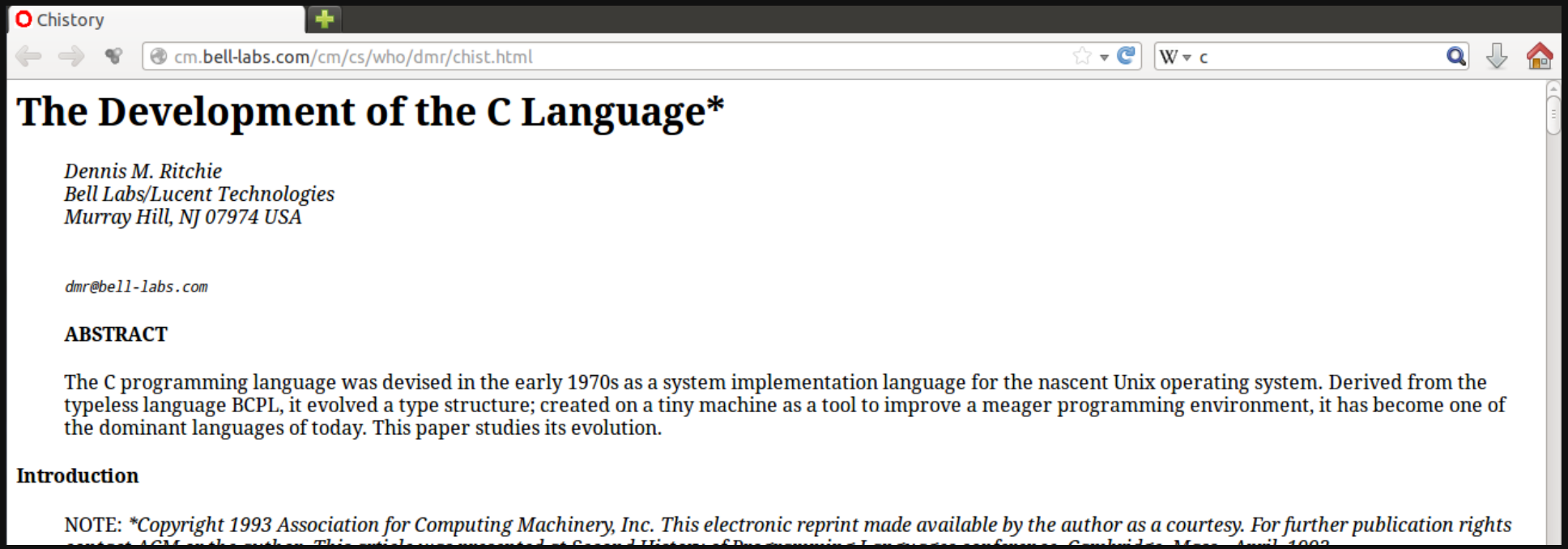


Julesmazur (Wikipedia)

The Development of the C Language

"*The Development of the C Language*", by Dennis Ritchie:

- http://cm.bell-labs.com/cm/cs/who/dmr/chist.html



Chistory

cm.bell-labs.com/cm/cs/who/dmr/chist.html

The Development of the C Language*

Dennis M. Ritchie
Bell Labs/Lucent Technologies
Murray Hill, NJ 07974 USA

dmr@bell-labs.com

ABSTRACT

The C programming language was devised in the early 1970s as a system implementation language for the nascent Unix operating system. Derived from the typeless language BCPL, it evolved a type structure; created on a tiny machine as a tool to improve a meager programming environment, it has become one of the dominant languages of today. This paper studies its evolution.

Introduction

NOTE: **Copyright 1993 Association for Computing Machinery, Inc. This electronic reprint made available by the author as a courtesy. For further publication rights*

Features of the C Language

- C is **imperative** (procedural).

  Instructions define the actions of the processor.

- C instructions **map easily** to **machine instructions**.

- C is meant to be **cross-platform**.

  Source code can be compiled to different hardware with minimal modifications.

- C uses **lexical scoping**.

  Variable scope defined by its position in the source code.

Features of the C Language

- C uses **static** type system.

  Types are checked during compile time.

- C supports **recursion**.

  A function can call itself.

- In C, function parameters are **passed by value**.

  Copies.

- C is **weakly** typed.

  Casting.

C Compilers

- There are several **C compilers** around. Some examples are:
  - Open Watcom C/C++.
  - CodeWarrior.
  - Clang (LLVM).
  - GCC.
- More C compilers can be found on:

  https://en.wikipedia.org/wiki/List_of_compilers#C_compilers

Versions of C

- Since its creation, C has undergone several **improvements**.
- Resulting **versions** have been **published** as **standards**:
  - ANSI C (1990), ratified as ISO/IEC 9899:1990.

    `gcc` option: **`-ansi`** or **`-std=c90`**
  - ISO/IEC 9899:1999.

    `gcc` option: **`-std=c99`**
  - ISO/IEC 9899:2011.

    `gcc` option: **`-std=c11`**

At the time of this writing, **my gcc defaults** to **`-std=gnu89`**, which stands for the **GNU-extended** version of the **ISO/IEC 9899:1990**.

C Program Example

The program below computes the summation of all integers within a given closed interval:

summation.c

```c
#include <stdio.h>

int Sum( int begin, int end ) {
    int i;
    int acc = 0;
    for ( i = begin; i <= end; i++ )
        acc += i;

    return acc;
}

int main ( void ) {
    int a = 1;
    int b = 5;
    int sum = Sum( a, b );
    printf( "Sum: %i\n", sum );
    return 0;
}
```

Compile and run

```
~$ gcc summation.c
~$ ./a.out
```

or

```
~$ gcc summation.c -o summation
~$ ./summation
```

DIY!

C Data Types

- Basic types.
- Structured types.
- User defined types.

  They are, basically, combinations of the above data types identified by a user−defined name.

Basic Data Types

- `char`
- `int`
- `float`
- `double`
- `pointers`

Optional specifiers*:

- `signed`
- `unsigned`
- `short`
- `long`

*may not apply to all numeric types.

`char`	`signed int`	`long long int`
`signed char`	`unsigned`	`signed long long`
`unsigned char`	`unsigned int`	`signed long long int`
`short`	`long`	`unsigned long long`
`short int`	`long int`	`unsigned long long int`
`signed short`	`signed long`	`float`
`signed short int`	`signed long int`	`double`
`unsigned short`	`unsigned long`	`long double`
`unsigned short int`	`unsigned long int`	
`int`	`long long`	

Aggregate Data Types

C offers 2 types of aggregate data types:

- Arrays.
- Structs.

Structs

- Is an **aggregate** that might contain **members** of **different types**.
- Given a variable of type **`struct`**, its members can be **accessed** through the '`.`' operator.
- Example

structs.c

```
struct Date {
    int day;
    int month;
    int year;
};

int main( void ) {
    struct Date x;
    x.day = 5;
    x.month = 2;
    x.year = 2016;

    return 0;
}
```

Arrays

*"(...) an **array data structure**, or simply an **array**, is a data structure consisting of a **collection** of elements (values or variables), each identified by at least one array **index** or key. An array is stored so that the **position** of each element **can be computed from its index tuple** by a mathematical formula."*

Array data structure, Wikipedia

Arrays in C

- Traditionally of **fixed**, **static** size.
- Usually, **all elements** are of the **same type**.
- **Does not** carry information about its **size**!
- May be **multidimensional**.
- Example:

example_26.c

```c
#include <stdio.h>

int x[4] = { 10, 20, 30, 40 };

int main( void ) {
    int i;
    for ( i = 0; i < 4; i++ )
        printf( "%i ", x[i] );

    return 0;
}
```

Multidimensional Arrays in C

- **Multidimensional** arrays can be obtained by applying the array concept **recursively**.
- Example:

example_27.c

```c
#include <stdio.h>

int x[2][3] = { { 10, 20, 30}, { 40, 50, 60 } };

int main( void ) {
    int i, j;
    for ( i = 0; i < 2; i++ )
        for ( j = 0; j < 3; j++ )
            printf( "%i ", x[i][j] );

    return 0;
}
```

Logical representation of the 2D array.

	0	1	2
0	10	20	30
1	40	50	60

Actual distribution of the array elements in memory.

addr + 0	10
addr + 1	20
addr + 2	30
addr + 3	40
addr + 4	50
addr + 5	60