Introduction to C Programming

Oliver Masters
oliver.masters@ibm.com

15th August 2018

What is C?

- General-purpose, imperative, procedural programming language

What is C?

- General-purpose, imperative, procedural programming language
- Compiled

What is C?

- General-purpose, imperative, procedural programming language
- Compiled
- Extremely portable

What is C?

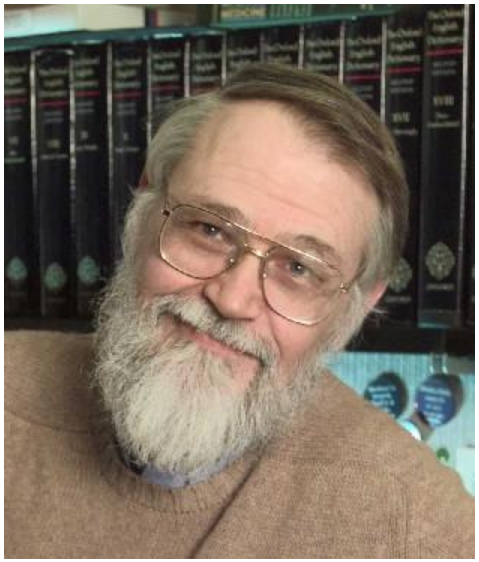
- General-purpose, imperative, procedural programming language
- Compiled
- Extremely portable
- Static, weak typing

What is C?

- General-purpose, imperative, procedural programming language
- Compiled
- Extremely portable
- Static, weak typing
- Higher level than assembly, but low-level memory access available

Brief history of C

- First released 1972

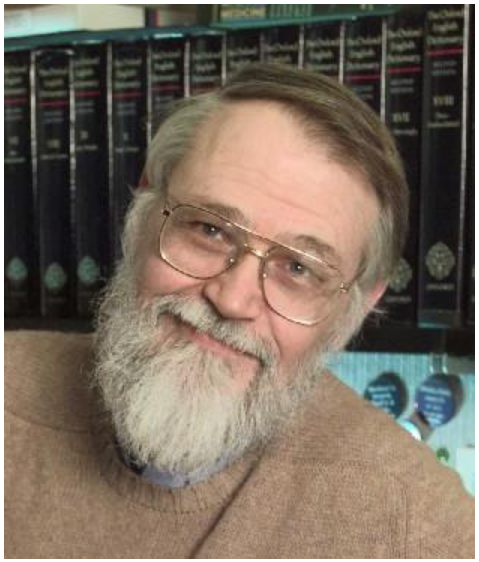


Brian Kernighan

Dennis Ritchie

Brief history of C

- First released 1972
- Bell Laboratories

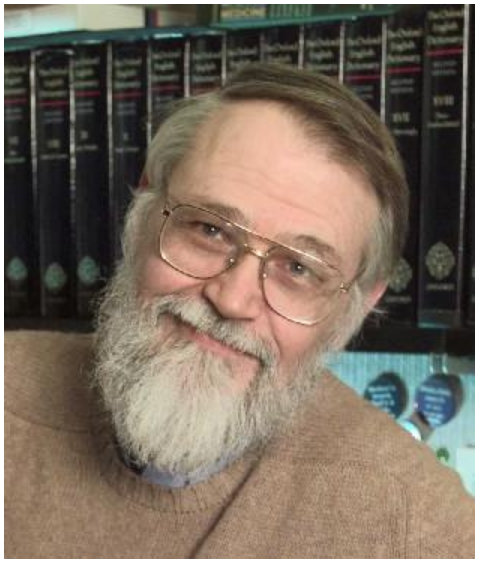


Brian Kernighan

Dennis Ritchie

Brief history of C

- First released 1972
- Bell Laboratories
- Developed alongside Unix by K&R

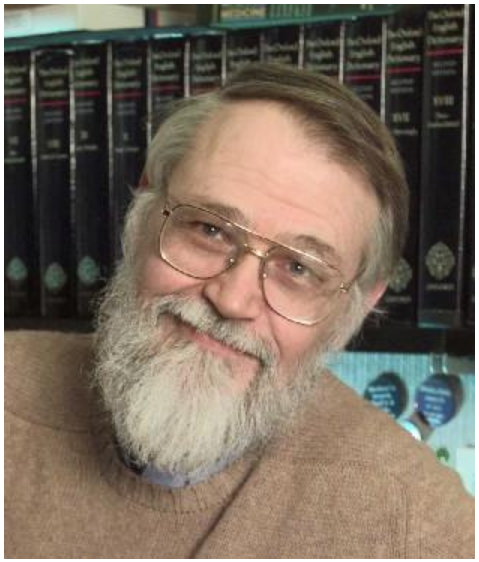


Brian Kernighan

Dennis Ritchie

Brief history of C

- First released 1972
- Bell Laboratories
- Developed alongside Unix by K&R
- 5 standards in total, latest: C11

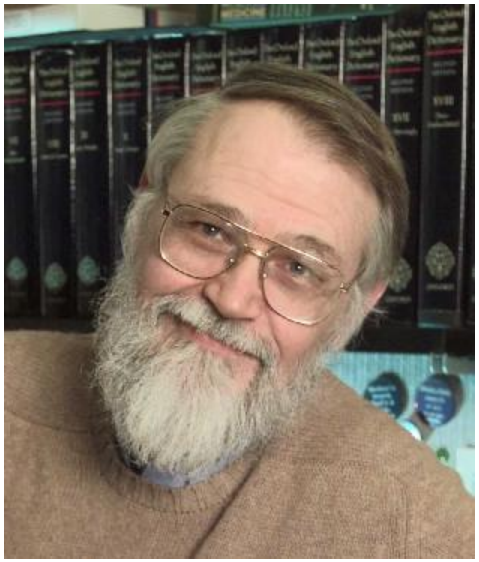


Brian Kernighan

Dennis Ritchie

Brief history of C

- First released 1972
- Bell Laboratories
- Developed alongside Unix by K&R
- 5 standards in total, latest: C11
- Influenced almost every major language since



Brian Kernighan

Dennis Ritchie

Why learn or use C?

- Extremely fast to execute

Why learn or use C?

- Extremely fast to execute
- Unbeatable portability

Why learn or use C?

- Extremely fast to execute
- Unbeatable portability
- Very mature – systems critical to safety

Why learn or use C?

- Extremely fast to execute
- Unbeatable portability
- Very mature – systems critical to safety
- Microcontrollers/embedded systems

Why learn or use C?

- Extremely fast to execute
- Unbeatable portability
- Very mature – systems critical to safety
- Microcontrollers/embedded systems
- Kernel and driver development

Why learn or use C?

- Extremely fast to execute
- Unbeatable portability
- Very mature – systems critical to safety
- Microcontrollers/embedded systems
- Kernel and driver development
- Deeper understanding of algorithms and data structures

Why learn or use C?

- Extremely fast to execute
- Unbeatable portability
- Very mature – systems critical to safety
- Microcontrollers/embedded systems
- Kernel and driver development
- Deeper understanding of algorithms and data structures
- Low-level control

Hello World

```c
#include <stdio.h>

/*This is a comment, which may be
multi-lined*/

int main(int argc, char *argv[])
{
    printf("Hello, world!\n");
    return 0;
}
```

Notes on syntax

- Whitespace ignored
- Semicolons at ends of statements
- Braces
- Identifiers may not override keywords

auto	extern	short	while
break	float	signed	_Alignas
case	for	sizeof	_Alignof
char	goto	static	_Atomic
const	if	struct	_Bool
continue	inline	switch	_Complex
default	int	typedef	_Generic
do	long	union	_Imaginary
double	register	unsigned	_Noreturn
else	restrict	void	_Static_assert
enum	return	volatile	_Thread_local

Table: Keywords in C11

Variables

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int distanceInKm = 500;
    float timeInSeconds = 13231.5;
    const int hourInSeconds = 3600;
    float timeInHours;
    double averageSpeed;

    // Implicit type conversions
    timeInHours = timeInSeconds / hourInSeconds;
    averageSpeed = distanceInKm / timeInHours;

    printf("Average speed: %.2f km/h\n", averageSpeed);

    return 0;
}
```

Conditional execution

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int age = 22;
    if(20 <= age && age <= 29)
    {
        printf("I am in my twenties\n");
    }
    else
    {
        printf("I am not in my twenties\n");
    }
    return 0;
}
```

While loops

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int a = 0;
    printf("Even natural numbers below 20:\n");
    while(a < 20)
    {
        printf("%d\n", a);
        a += 2;
    }
    return 0;
}
```

For loops

```c
#include <stdio.h>
#include <math.h>

int main(int argc, char *argv[])
{
    printf("Square numbers below 100:\n");
    for(int i = 0; i < 100; i++)
    {
        int squareRoot = sqrt(i);
        if(squareRoot * squareRoot == i)
        {
            printf("%d\n", i);
        }
    }
    return 0;
}
```

Functions

```c
#include <stdio.h>
#include <math.h>

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

int main(int argc, char *argv[])
{
    printf("Max of 10 and 20: %d\n", max(10,20));
    return 0;
}
```

Pointers

```c
#include <stdio.h>

void triple(int *number)
{
    *number *= 3;
}

int main(int argc, char *argv[])
{
    int salary = 30000;
    int *salaryPointer = &salary
    triple(salaryPointer);
    printf("%d\n", salary);
    return 0;
}
```

Arrays

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    float numbers[5] = {1.2, 53.2, 0.7, -10, 100.4};
    for(int i = 0; i < 5; i++)
    {
        printf("%.2f\n", numbers[i]);
    }
    return 0;
}
```

Passing Arrays to Functions

```c
#include <stdio.h>

void printNumbers(float *array, int arraySize)
{
    for(int i = 0; i < arraySize; i++)
    {
        printf("%.2f\n", array[i]);
    }
}

int main(int argc, char *argv[])
{
    float numbers[5] = {1.2, 53.2, 0.7, -10, 100.4};
    printNumbers(numbers, 5);
    return 0;
}
```

Strings

```c
#include <stdio.h>
#include <string.h>
#include <stdbool.h>

bool isPalindrome(char *word)
{
    int length = strlen(word);
    for(int i = 0; i < length/2; i++)
    {
        if(word[i] != word[length-1-i])
            return false;
    }
    return true;
}

int main(int argc, char *argv[])
{
    int number = 100 + 1;
    char string[50];
    sprintf(string, "abc%dcba", number);
    if(isPalindrome(string))
        printf("%s is a palindrome!\n", string);
    else
        printf("%s is not a palindrome!\n", string);
}
```

Structs

```c
#include <stdio.h>
#include <math.h>

struct point
{
    double x;
    double y;
};

double size(struct point *vector)
{
    return sqrt(vector->x * vector->x + vector->y * vector->y );
}

double distance(struct point *a, struct point *b)
{
    struct point difference;
    difference.x = a->x - b->x;
    difference.y = a->y - b->y;
    return size(&difference);
}

int main(int argc, char *argv[])
{
    struct point a = {1, 5};
    struct point b = {4, 1};
    printf("%.2f\n", distance(&a, &b));
    return 0;
}
```

Memory Management

- Can also be done dynamically

Memory Management

- Can also be done dynamically
- No garbage collection – `malloc` and `free`

Memory Management

- Can also be done dynamically
- No garbage collection – `malloc` and `free`
- See example code in `Code/Memory/memory.c`

Further reading

- General tutorials
  - https://www.tutorialspoint.com/cprogramming/index.htm
  - https://www.cprogramming.com/tutorial/c-tutorial.html
- Algorithms and data structures in C
  - https://www.sitesbay.com/data-structure/c-data-structure
  - https://www.geeksforgeeks.org/data-structures/
  - https://www.cprogramming.com/algorithms-and-data-structures.html
- Useful tools, frameworks and libraries
  - https://cmake.org/
  - https://libcheck.github.io/check/
  - http://cutest.sourceforge.net/
  - https://www.gnu.org/software/gsl/
  - http://www.netlib.org/lapack/
  - https://developer.gnome.org/glib/
  - https://curl.haxx.se/libcurl/
  - https://www.openmp.org/

Questions

- Any questions?