

# GlobalLogic<sup>®</sup>

## GL Basecamp: C/Embedded

# Getting started

## Why to bother

- We're looking for the bright minds
  - Preferably geeks with night watch on their pet-projects
- We want smarter people
  - And to help our education, BTW

## C (Embedded, Linux) Trainee Engineer (IRC48427)

- BS/MS degree in computer science
- Programming experience in C
- Experience with Linux
- Basic understanding of embedded software
- Pre-intermediate level of English
- + Networking
- + Linux Kernel
- + Telecom-Wireless

# Embedded/Automotive Domain

## Embedded clients



TIZEN®

yocto  
PROJECT

freeRTOS

ALTERA  
MEASURABLE ADVANTAGE™

CEVA®

Celeno™  
Wireless CommunicationsSEQUANS  
COMMUNICATIONS

AMIMON

RENESAS



SanDisk®

QUALCOMM®

TEXAS  
INSTRUMENTS

Continental

Panasonic

LATTICE  
SEMICONDUCTOR.inside  
SECURE

NXP



CSR

# Embedded product applications

Telecom NE & CPE



Automotive, Telematics



Digital Entertainment



Handsets



Portable/Wearable Devices



Healthcare Devices



Industrial Control



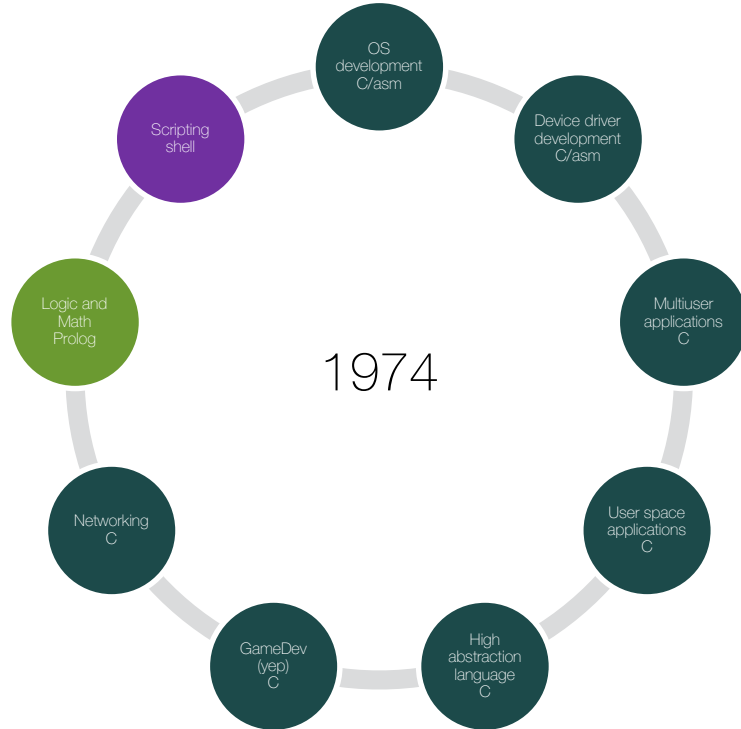
Enterprise Hardware



# C/Embedded

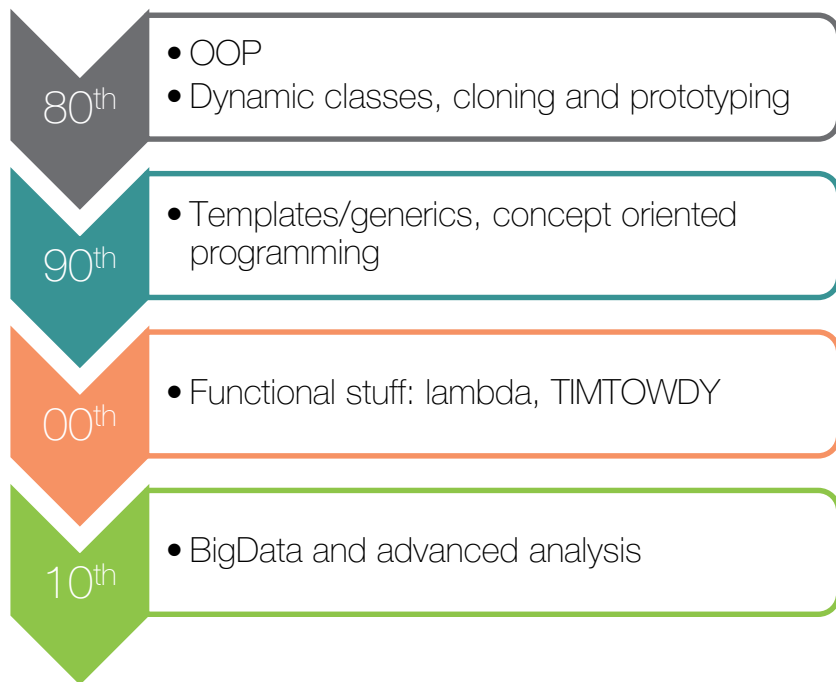
## Lecture 1: Getting started

# C in low level for 40 years



# Syntax sugar, C style

## Syntax sugar in other languages

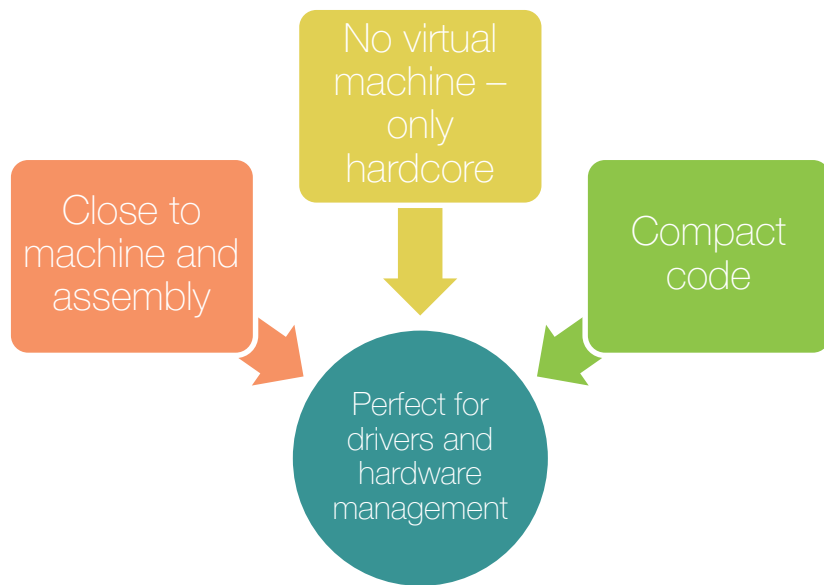


## Syntax sugar, C style

- Registers and direct memory access
- Direct operation with system calls and OS interface
- Embedded ASM
- Manual linkage and code segments control



## Why it's C in low level



- Low level coding is technically easy
- It doesn't requires powerful abstractions
- It's harder to debug
- It's much closer to hardware/OS specification
- Doesn't require OS at all
- Very easily optimizable

## Use cases

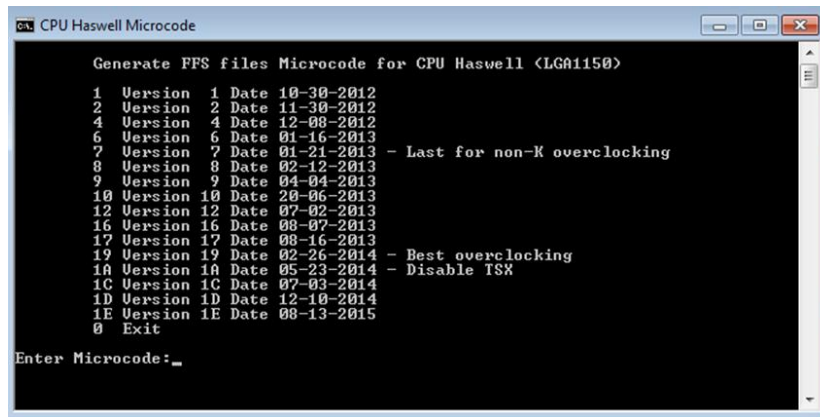
- Microcode running on the microcontroller itself without OS
- Bootloading and secure operations before OS
- OS core
- OS modules and drivers (at least those, that are connected directly to OS)
- Minimalistic userspace
  - Very minimalistic
    - Very-very minimalistic
- Resourceless systems (sensors, chip sets, etc.)
- BTW, Oracle Java machine is written in C:  
[https://asktom.oracle.com/pls/asktom/f?p=100:11:0::NO::P11\\_QUESTION\\_ID:5228516811673](https://asktom.oracle.com/pls/asktom/f?p=100:11:0::NO::P11_QUESTION_ID:5228516811673)
  - It was written in Fortan before.

## Use case #1

Microcode running on the microcontroller itself without OS

- Most of the modern processors have some microcode flashed at the factory
- It usually serves only one aim: bridge control to the next chain software
- In powerful systems it's responsible for configuration and effective resources management
- Or it restricts user from configuring something he can spoil
  - or hack

## i7 Hashwell microcode update



```
Generate FFS files Microcode for CPU Haswell (LGA1150)

1 Version 1 Date 10-30-2012
2 Version 2 Date 11-30-2012
4 Version 4 Date 12-08-2012
6 Version 6 Date 01-16-2013
7 Version 7 Date 01-21-2013 - Last for non-K overclocking
8 Version 8 Date 02-12-2013
9 Version 9 Date 04-04-2013
10 Version 10 Date 20-06-2013
12 Version 12 Date 07-02-2013
16 Version 16 Date 08-07-2013
17 Version 17 Date 08-16-2013
19 Version 19 Date 02-26-2014 - Best overclocking
1A Version 1A Date 05-23-2014 - Disable ISX
1C Version 1C Date 07-03-2014
1D Version 1D Date 12-10-2014
1E Version 1E Date 08-13-2015
0 Exit

Enter Microcode: _
```

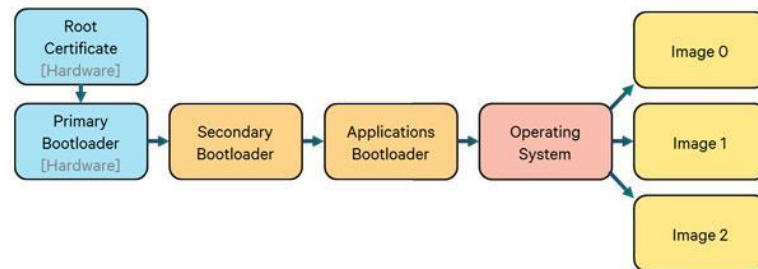
credits: [www.techinferno.com](http://www.techinferno.com)

## Use case #2

### Bootloading

- Microcode jumps to the bootloader
- Bootloader is used for booting OS, sometimes two OSes
- Sometimes it verify OS and restores it from the damage
- Or it can blink leds and initialize hardware
- And usually bootloader simplify OS/firmware replacement

### Android bootloaders chain



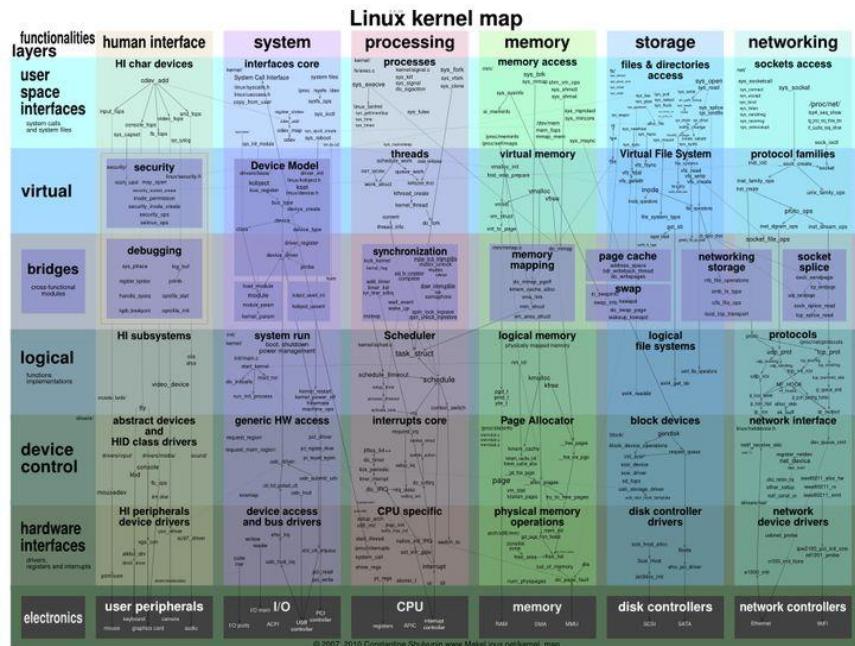
credits: [www.qualcomm.com](http://www.qualcomm.com)

# Use case #3

## OS core

- A virtual machine for everything
- OS core is that close to machine codes, it's CPU dependent
- Writing OS core in high-level languages increases it size exponentially
- All of the most popular OS cores are written in C
  - Most of the non-popular are written in C as well

## Linux kernel map



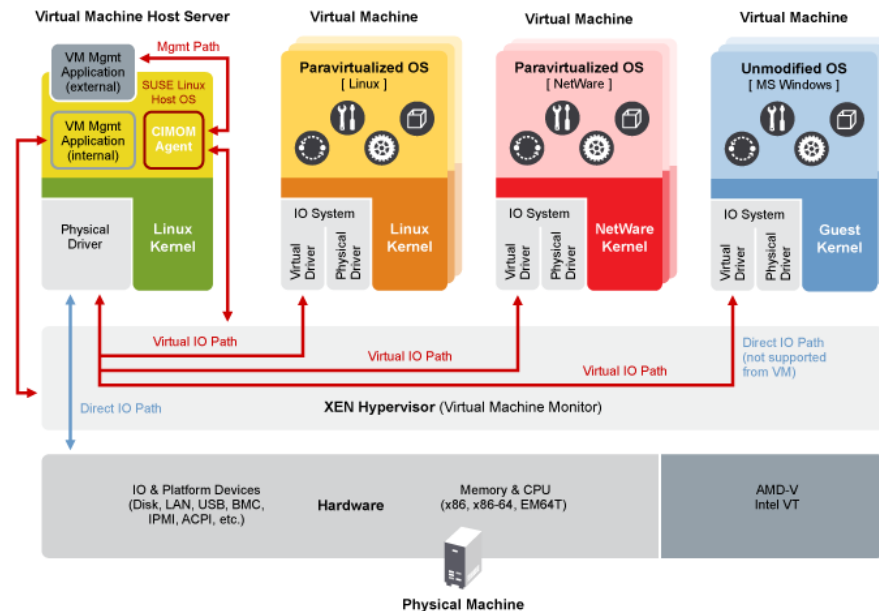
credits: [www.makelinux.com](http://www.makelinux.com)

## Use case #3a

### Paravirtualization

- A virtualization for other OSes
  - A virtual machine for virtual machines
- Naturally a layer below other OSes have to be written in C as well

### XEN hypervisor



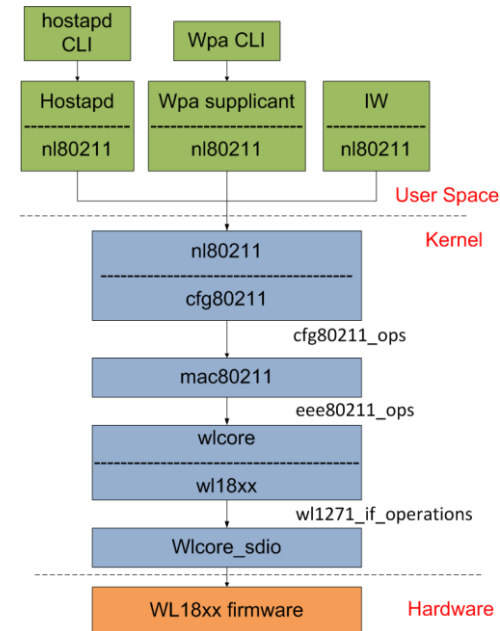
credits: [www.suse.com](http://www.suse.com)

# Use case #4

## Drivers

- An extension for OS is naturally written in the same language as OS
  - or at least the interfacing part
- There millions of drivers for everything, because every hardware differs from each other
- Most standard drivers migrate to the kernel, but new ones are still written manually

## TI WiLink Wi-Fi driver layers



credits: processors.wiki.ti.com

## Use case #5

### Minimalistic userspace

- In case you need to put a user-friendly (well, kind of friendly environment) in 2Mb of flash, you have to write it in C
  - or at least in C++, but hush!
- In fact embedded systems are usually baremetal or have at least minimal userspace

### Busybox

```
BusyBox v1.15.3 (2010-05-14 01:28:52 CEST) multi-call binary
Copyright (C) 1998-2008 Erik Andersen, Rob Landley, Denys Vlasenko
and others. Licensed under GPLv2.
See source distribution for full notice.

Usage: busybox [function] [arguments]...
or: function [arguments]...

BusyBox is a multi-call binary that combines many common Unix
utilities into a single executable. Most people will create a
link to busybox for each function they wish to use and BusyBox
will act like whatever it was invoked as!

Currently defined functions:
[, [[, acpid, addgroup, adduser, adjtimex, ar, arp, arping, ash, awk,
basename, bb, bbconfig, bbsh, beep, blkid, brctl, bunzip2, bzip2,
bzip2, cal, cat, catv, chat, chattr, chgrp, chmod, chown, chpasswd,
chpst, chroot, chrt, chvt, cksum, clear, cmp, comm, cp, cpio, crond,
cryptpw, cttyhack, cut, date, dd, deallocvt, delgroup, deluser, depmod,
devmem, df, dhcprelay, diff, dirname, dmesg, dnsdomainname, dos2unix,
dpkg-deb, du, dumpkmap, dumpleases, echo, ed, egrep, eject, env,
envdir, envuidgid, ether-wake, expand, expr, false, fbset, fdflush,
fdformat, fdisk, fgrep, find, findfs, flash_eraseall, flash_lock,
flash_unlock, free, freeramdisk, fsck, fsck.minix, fsync, ftpd, fuser,
getopt, getty, grep, gunzip, gzip, halt, hd, hdparm, head, hexdump,
hostname, httpd, hwclock, id, ifconfig, ifdown, ifenslave, ifplugd,
ifup, init, insmod, install, ionice, ip, ipaddr, ipcrm, ipcs, iplink,
```

credits: busybox.net



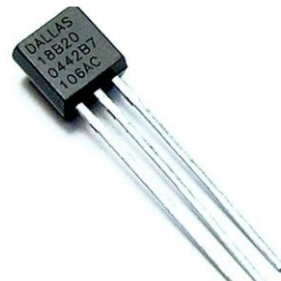
## Use case #6

### Resourceless systems

- Most of the sensors are true hardware
- However IoT trend brings new world of connectivity even to the simplest temperature sensors for low price
- Any connectivity chip has microcode and often some RToS on top



GNSS  
module



Temperature  
sensor

## Some myths

- “C is for small convenient applications”
  - True, but Linux kernel has 15 millions lines of code
  - It requires exceptional skills of working in unknown environment
- “C is primitive language”
  - True, it contains only 32 key words
  - It's harder to maintain readability and clean code
    - See previous note about Linux kernel
- “C is much faster than other languages”
  - True, but C++ is also compiled in the machine codes in the same way as C
  - C is still more optimization friendly and has lower overhead in real world tasks
- “C is all about pointers and bits”
  - Sacred true

## Bookshelf

- Standards
  - *C90* – ISO/IEC 9899:1990
  - *C99* – ISO/IEC 9899:1999
  - POSIX.1-2008 IEEE Std 1003.1™-2008
- Books
  - K&R *The C Programming Language*
  - Paul Deitel, Harvey Deitel *C: How to Program*
  - David Griffiths, Dawn Griffiths *Head First C*
  - and your favorite CPU assembly guide

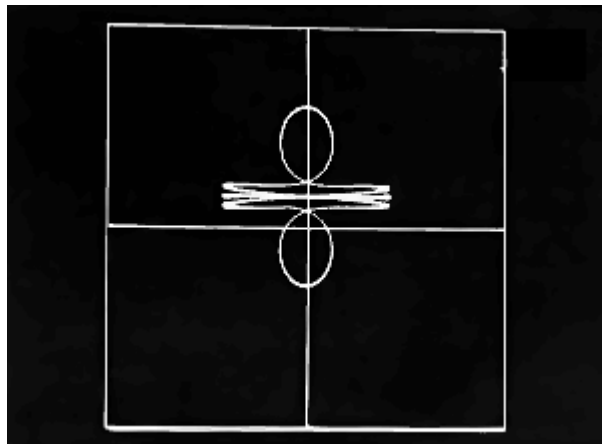
# Hello World

“hello, world\n”



- First written in 1974 in C by Brian Kernighan
- Everybody knows, how hello world looks like, right?
- Then why to invent a whole language?

## C/UNIX origins



Denni Ritchie about Kenn Thompson:

“... the DEC PDP-7 on which he started in 1968 was a machine with 8K 18-bit words of memory and no software useful to him ...

While wanting to use a higher-level language, he wrote the original Unix system in PDP-7 assembler. ...

... the entire source of a program was presented to the assembler, and the output file—with a fixed name—that emerged was directly executable [a.out]

... Unix ... needed a system programming language. After a rapidly scuttled attempt at Fortran, he created instead a language of his own, which he called B. B can be thought of as C without types

... By early 1973, the essentials of modern C were complete”

## Reading the code

- Here should be a more advanced hello world code,
- but there will be a part of Linux kernel initialization routine,
- because there is no sense to show hello world to the guys, who code already
- Now let's read the code

```
static int __ref kernel_init(void *unused)
{
    int ret;

    kernel_init_freeable();
    /* need to finish all async __init code before freeing the memory */
    async_synchronize_full();
    free_initmem();
    mark_readonly();
    system_state = SYSTEM_RUNNING;
    numa_default_policy();

    rcu_end_inkernel_boot();

    if (ramdisk_execute_command) {
        ret = run_init_process(ramdisk_execute_command);
        if (!ret)
            return 0;
        pr_err("Failed to execute %s (error %d)\n",
               ramdisk_execute_command, ret);
    }

    /*
    * We try each of these until one succeeds.
    *
    * The Bourne shell can be used instead of init if we are
    * trying to recover a really broken machine.
    */
    if (execute_command) {
        ret = run_init_process(execute_command);
        if (!ret)
            return 0;
        panic("Requested init %s failed (error %d).",
              execute_command, ret);
    }
    if (!try_to_run_init_process("/sbin/init") ||
        !try_to_run_init_process("/etc/init") ||
        !try_to_run_init_process("/bin/init") ||
        !try_to_run_init_process("/bin/sh"))
        return 0;

    panic("No working init found. Try passing init= option to kernel. "
          "See Linux Documentation/admin-guide/init.rst for guidance.");
}
```

# Program structure

- Statements
- Scope
- All of the functions have their own scope
- There is a global scope
- Scope unites several statements
- Statement contains of expressions finished with a semicolon (which is optional after a scope)

```
static int __ref kernel_init(void *unused)
{
    int ret;

    kernel_init_freeable();
    /* need to finish all async __init code before freeing the memory */
    async_synchronize_full();
    free_initmem();
    mark_readonly();
    system_state = SYSTEM_RUNNING;
    numa_default_policy();

    rcu_end_inkernel_boot();

    if (ramdisk_execute_command) {
        ret = run_init_process(ramdisk_execute_command);
        if (!ret)
            return 0;
        pr_err("Failed to execute %s (error %d)\n",
               ramdisk_execute_command, ret);
    }

    /*
     * We try each of these until one succeeds.
     *
     * The Bourne shell can be used instead of init if we are
     * trying to recover a really broken machine.
     */
    if (execute_command) {
        ret = run_init_process(execute_command);
        if (!ret)
            return 0;
        panic("Requested init %s failed (error %d).",
              execute_command, ret);
    }
    if (!try_to_run_init_process("/sbin/init") ||
        !try_to_run_init_process("/etc/init") ||
        !try_to_run_init_process("/bin/init") ||
        !try_to_run_init_process("/bin/sh"))
        return 0;

    panic("No working init found. Try passing init= option to kernel. "
          "See Linux Documentation/admin-guide/init.rst for guidance.");
}
```



## Comparing to Python

- C is strictly typed language
- C doesn't care about whitespaces:  
only statement separator is semicolon  
(and “invisible” semicolon after scopes)
- Again, ; is mandatory, compiler will get lost, if you skip ;
- C uses “” for strings and ‘’ for characters. String literal in C
- Formatted output in C
- Parenthesis in if is mandatory
- Comments in C

```
static int __ref kernel_init(void *unused)
{
    int ret;

    kernel_init_freeable();
    /* need to finish all async __init code before freeing the memory */
    async_synchronize_full();
    free_initmem();
    mark_readonly();
    system_state = SYSTEM_RUNNING;
    numa_default_policy();

    rcu_end_inkernel_boot();

    if (ramdisk_execute_command) {
        ret = run_init_process(ramdisk_execute_command);
        if (!ret)
            return 0;
        pr_err("Failed to execute %s (error %d)\n",
               ramdisk_execute_command, ret);
    }

    /*
     * We try each of these until one succeeds.
     *
     * The Bourne shell can be used instead of init if we are
     * trying to recover a really broken machine.
     */
    if (execute_command) {
        ret = run_init_process(execute_command);
        if (!ret)
            return 0;
        panic("Requested init %s failed (error %d).",
              execute_command, ret);
    }
    if (!try_to_run_init_process("/sbin/init") ||
        !try_to_run_init_process("/etc/init") ||
        !try_to_run_init_process("/bin/init") ||
        !try_to_run_init_process("/bin/sh"))
        return 0;

    panic("No working init found. Try passing init= option to kernel. "
          "See Linux Documentation/admin-guide/init.rst for guidance.");
}
```

# Comparing to Java

- C doesn't have classes and inheritance at all. Object-oriented principles and design here uses other patterns and approaches
- C has pointers. They're like Java references (strictly speaking, it's Java reference is unlucky renaming), but pointers allow arithmetic operations
- A pointer to void exists
- C doesn't have strict language convention. It also doesn't have recommendation on identifiers names
- C is lesser strict to types, it allows implicit conversions between integer types and cast of pointers.
- There is a global scope in C, when developer can define variables and use them. Strictly speaking the functions defined in the file belongs to global scope

```
static int __ref kernel_init(void *unused)
{
    int ret;

    kernel_init_freeable();
    /* need to finish all async __init code before freeing the memory */
    async_synchronize_full();
    free_initmem();
    mark_readonly();
    system_state = SYSTEM_RUNNING;
    numa_default_policy();

    rcu_end_inkernel_boot();

    if (ramdisk_execute_command) {
        ret = run_init_process(ramdisk_execute_command);
        if (!ret)
            return 0;
        pr_err("Failed to execute %s (error %d)\n",
               ramdisk_execute_command, ret);
    }

    /*
     * We try each of these until one succeeds.
     *
     * The Bourne shell can be used instead of init if we are
     * trying to recover a really broken machine.
     */
    if (execute_command) {
        ret = run_init_process(execute_command);
        if (!ret)
            return 0;
        panic("Requested init %s failed (error %d).",
              execute_command, ret);
    }
    if (!try_to_run_init_process("/sbin/init") ||
        !try_to_run_init_process("/etc/init") ||
        !try_to_run_init_process("/bin/init") ||
        !try_to_run_init_process("/bin/sh"))
        return 0;

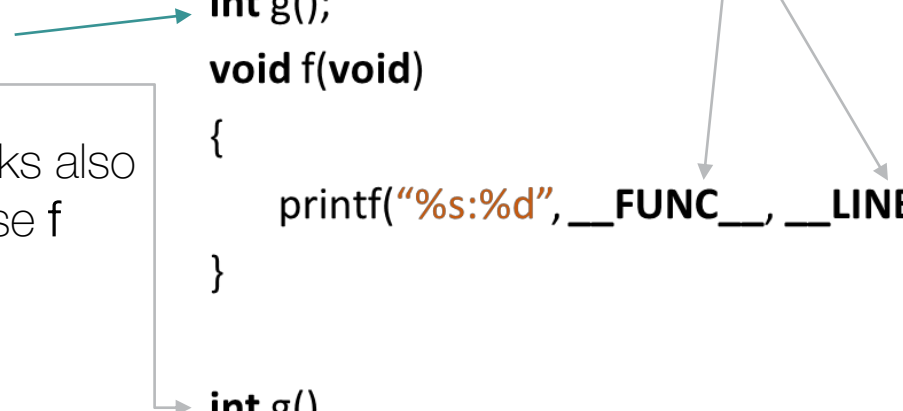
    panic("No working init found. Try passing init= option to kernel. "
          "See Linux Documentation/admin-guide/init.rst for guidance.");
}
```

# Functions

- C have function declarations and definitions
- Definition of the function works also as a declaration, so **g** can use **f** without declaration
- Function name + argument types + return type = semantic
- Functions can't be overloaded in C

```
int g();  
void f(void)  
{  
    printf("%s:%d", __FUNC__, __LINE__);  
}  
  
int g()  
{  
    printf("g");  
    f();  
    return 0;  
}
```

built-in variables for current function and line (useful for debugging)



# Clean code is important

C doesn't restricts you on the way you put spaces and the way you names the identifiers.

If the code is supposed to be read at least twice, you have to write it clean, especially if the code will be shared

## Two more C hello worlds

### Arduino

```
void setup () {  
    Serial.begin (9600);  
    Serial.println ("Hello World!");  
}  
  
void loop () {  
    while (1) ;  
}
```

### Linux kernel module

```
#include <linux/init.h>  
#include <linux/module.h>  
static int __init hello_init(void)  
{  
    printk("Hello, world!\n");  
    return 0;  
}  
  
module_init(hello_init);  
static void __exit hello_exit(void)  
{  
}  
  
module_exit(hello_exit);  
MODULE_LICENSE("GPL");  
MODULE_AUTHOR("Valerie Henson <val@nmt.edu>");  
MODULE_DESCRIPTION("\"Hello, world!\" minimal module");  
MODULE_VERSION("printk");
```

# Compilation vs scripting

- We'll take a look on compilation very soon
- For now, you can use  
`gcc --std=c90 -Wall file1.c file2.c file3.c -o a.out`
- Pay attention to any warning emitted, probably compiler suggests proper stuff
- Note for python coders:
  - C is compiled language.  
It doesn't have an interpreter
  - The result of the compilation is an executable (normally), but soon we'll work with firmwares (which are also a segmented bunch of code)
- Note for Java coders:
  - C doesn't have an reference compiler, we'll use gcc, as most cross-platform one
- Note for C++ coders:
  - gcc is for C, g++ is for C++

Here our courses start

## Quiz (light)

Fill in the blanks in each of the following.

- a) Every C program begins execution at the function \_\_\_\_\_.
- b) The \_\_\_\_\_ begins the body of every function and the \_\_\_\_\_ ends the body of every function.
- c) Every statement ends with a(n) \_\_\_\_\_.
- d) The \_\_\_\_\_ standard library function displays information on the screen.
- e) The \_\_\_\_\_ statement is used to execute one action when a condition is true and another action when that condition is false. "
- f) The \_\_\_\_\_ statement specifies that a statement or group of statements is to be executed repeatedly while some condition remains true. "

State whether each of the following is true or false. If false, explain why.

- b) Comments cause the computer to print the text enclosed between /\* and \*/ on the screen when the program is executed.
- d) All variables must be defined before they're used.
- e) All variables must be given a type when they're defined.
- f) C considers the variables number and NuMbEr to be identical.
- g) Definitions can appear anywhere in the body of a function.

Identify and correct the errors in each of the following statements.  
(Note: There may be more than one error per statement.)

- a) scanf( "d", value );
- b) printf( "The product of %d and %d is %d"\n, x, y );
- c) firstNumber + secondNumber = sumOfNumbers
- d) if ( number => largest )  
largest == number;
- e) /\* Program to determine the largest of three integers /\*
- f) scanf( "%d", anInteger );

Write a single C statement to accomplish each of the following:

- a) Read an integer from the keyboard and store the value entered in integer variable a.
- b) Print "The product is" followed by the value of the integer variable result.
- c) If number is not equal to 7, print "The variable number is not equal to 7."



## Home task (light)

(Arithmetic) Write a program that asks the user to enter two numbers, obtains them from the user and prints their sum, product, difference, quotient and remainder.

(Savings Calculator) Create an application that calculates your daily driving cost, so that you can estimate how much money could be saved by car pooling, which also has other advantages such as reducing carbon emissions and reducing traffic congestion. The application should input the following information and display the user's cost per day of driving to work:

- a) Total miles driven per day.
- b) Cost per gallon of gasoline.
- c) Average miles per gallon.
- d) Parking fees per day.
- e) Tolls per day.

# Harder tasks

## Quiz

This is just an introduction, there are no harder questions for now

## Home task

There are no harder homework for now

But it will be soon