Yale-NUS College - 2018-2019

Bruno Bodin

YSC2227: INTRO TO C

week 01.1.intro (auto-generated)

IN A NUTSHELL

- **Bruno Bodin, Assistant Professor Yale-NUS**
- Previously Research Associate at The University of Edinburgh, Scotland,
- and Software Engineer at Kalray, Paris.
- Research interests: Language, Compilation, Parallelism, and Robotics.
- Hobbies: Climbing, Weiqi
- Contact: **bruno.bodin@yale-nus.edu.sg**

WHAT IS THE C PROGRAMMING LANGUAGE

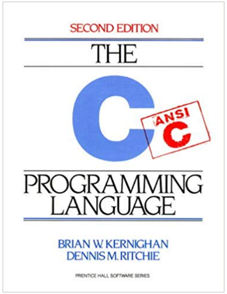
- C is one of the most commonly used programming languages, especially in science, engineering, and electronics. Many O.S. and microcontrollers are at least partly coded in C.
- C is lightweight and fast, while offering a complex memory management system.
- To learn and master C, you will require a deep understanding of how memory works and how data is represented.

EXPECTED OUTCOMES

- **You master the C language syntax and semantic.**
- You can **write**, **compile**, **test**, and **debug** a C program.
- You **understand** the memory management system used in C, as well as the way data is represented in memory.
- You can **explain** and **implement** the core functions of the standard library.

TEXTBOOKS

- **The C Programming Language** by Brian Kernighan and Dennis Ritchie
- **An Introduction to the C Programming Language and Software Design** by Tim Bailey
- **Digital Design and Computer Architecture - Appendix C** by D. M. Harris and S. L. Harris



EXERCICES

https://www.codeabbey.com/

http://rosettacode.org/wiki/Category:Programming_Tasks

https://www.codingame.com/

ACKNOWLEDGEMENT

With their permission I use teaching material from Simon Perrault (Yale-NUS College).
You can find more informations about sources in the last slide.

TOPICS THAT WILL BE COVERED

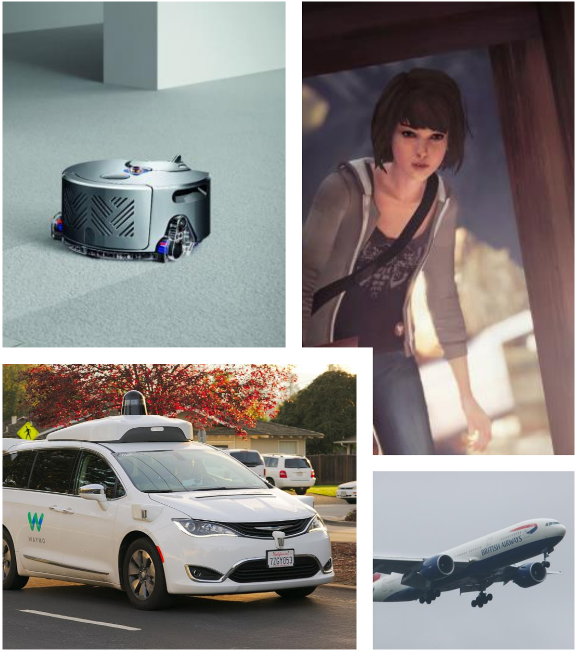
- Variables and Assignement Operators
- Numeric Data Types and Conversion
- Arrays
- Arithhmetic and bitwise operators
- Compilation, flags, and command-line arguments
- Pointers
- C functions
- Files and I/O
- Control structures, logic operators, and loops
- Scopes
- Structures and Unions
- Memory management and segmentation
- Basic libraries
- Makefile
- Debugging

FINAL GRADE

- 6 Assignments 60%
- 2 Quizzes 20%
- Class participation 20%
- **Each work must be your own, and original.**

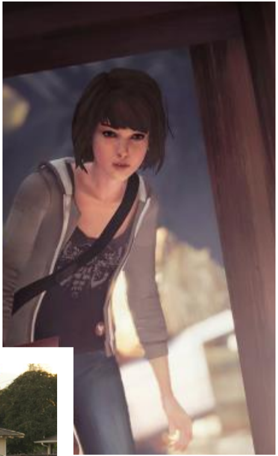
WAYMO
7ZGY053

BRITISH AIRWAYS



- Common point? *Computers* !

- Common point? *Computers* !
- Computers process **input** data and generate new **output** data. This is computation.



- Common point? *Computers* !
- Computers process **input** data and generate new **output** data. This is computation.
- Computer *programs* **describe** specific computation **tasks** executed by the computer.

HOW TO DESCRIBE COMPUTATION TASKS

- "Clean the floor", "Drive", "Flight", "Entertain"
- What language will you speak with your computer?

[1] notation for computation

[2] notion of computation

HOW TO DESCRIBE COMPUTATION TASKS

- In computer science we use *programming languages* to express computation tasks.[1]
- We define languages by their *syntax* and *semantic* .

programming languages, syntax, semantic, imperative, paradigms

[1] notation for computation

[2] notion of computation

HOW TO DESCRIBE COMPUTATION TASKS

- In computer science we use *programming languages* to express computation tasks.[1]
- We define languages by their *syntax* and *semantic* .
- Syntax defines **rules** to produce programs in a specific language.

[1] notation for computation

[2] notion of computation

programming languages, syntax, semantic, imperative, paradigms

HOW TO DESCRIBE COMPUTATION TASKS

- In computer science we use *programming languages* to express computation tasks.[1]
- We define languages by their *syntax* and *semantic* .
- Syntax defines **rules** to produce programs in a specific language.
- Semantic conveys **meaning**.

[1] notation for computation

[2] notion of computation

HOW TO DESCRIBE COMPUTATION TASKS

- In computer science we use *programming languages* to express computation tasks.[1]
- We define languages by their *syntax* and *semantic* .
- Syntax defines **rules** to produce programs in a specific language.
- Semantic conveys **meaning**.
- *Imperative* languages in computer science **formally** specify a set of operations used to produce outputs given specific inputs. Like a recipe.

[1] notation for computation

[2] notation of computation

programming languages, syntax, semantic, imperative, paradigms

HOW TO DESCRIBE COMPUTATION TASKS

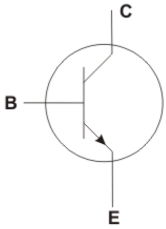
- In computer science we use *programming languages* to express computation tasks.[1]
- We define languages by their *syntax* and *semantic* .
- Syntax defines **rules** to produce programs in a specific language.
- Semantic conveys **meaning**.
- *Imperative* languages in computer science **formally** specify a set of operations used to produce outputs given specific inputs. Like a recipe.
- But there is many more *paradigms* [2] , such as declarative, functional (OCaml), reactive...

programming languages, syntax, semantic, imperative, paradigms

[1] notation for computation

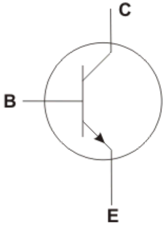
[2] notion of computation

WHAT'S THE DEAL WITH COMPUTERS?

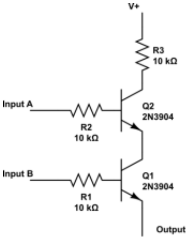


B = Base
C = Collector
E = Emitter

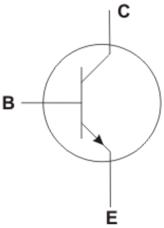
WHAT'S THE DEAL WITH COMPUTERS?



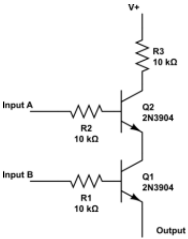
B = Base
C = Collector
E = Emitter

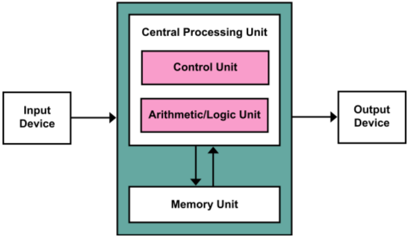


WHAT'S THE DEAL WITH COMPUTERS?



B = Base
C = Collector
E = Emitter

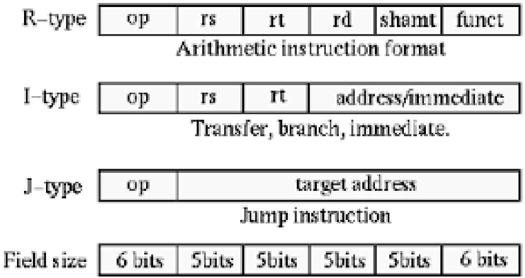




Von Neumann Architecture

MACHINE LANGUAGE (0 AND 1)

- *Instruction Set Architecture (ISA)* defines the syntax and semantic of a *machine language*
- Example of the MIPS instruction set :



R–type	op	rs	rt	rd	shamt	funct

Arithmetic instruction format

I–type	op	rs	rt	address/immediate

Transfer, branch, immediate.

J–type	op	target address

Jump instruction

Field size	6 bits	5bits	5bits	5bits	5bits	6 bits

00100000101001010000000000000001

instruction set architecture (isa), machine language

ASSEMBLY LANGUAGE

- Writing programs in series of 0 and 1 was **not an option**
- The *assembly language* is almost a one-to-one correspondence with the machine one.

Example for the **addi** instruction in MIPS

Description: Adds a register and an immediate value and stores the result in a register

Operation: \$t = \$s + imm; advance_pc(4);

Syntax: addi \$t, \$s, imm

Encoding: 0010 00ss ssst tttt iiii iiii iiii iiii

addi r5, r5, 1 ⇒ [001000] [00101] [00101] [0000000000000001]

NEED FOR HIGH-LEVEL LANGUAGES

- Assembly is **not *modular*** and **not *portable***

modular, portable, high-level languages, compilers

NEED FOR HIGH-LEVEL LANGUAGES

- Assembly is **not *modular*** and **not *portable***
- Comes the idea of *high-Level languages* , and of *compilers* .

modular, portable, high-level languages, compilers

NEED FOR HIGH-LEVEL LANGUAGES

- Assembly is **not *modular*** and **not *portable***
- Comes the idea of *high-Level languages* , and of *compilers* .
- For example FORTRAN and ALGOL58, two early imperative languages.
- 1957: The Fortran Optimizing Compiler: first demonstration that it is possible to automatically generate good machine code from high-level languages.
- 1960: LISP, first functionnal language.

modular, portable, high-level languages, compilers

REASONS FOR THE C LANGUAGE

- How to program an *Operating System* ?

REASONS FOR THE C LANGUAGE

- How to program an *Operating System* ?
- ASSEMBLY was too hard to maintain and not portable

REASONS FOR THE C LANGUAGE

- How to program an *Operating System* ?
- ASSEMBLY was too hard to maintain and not portable
- High-level languages such as FORTRAN were too far from the machine

REASONS FOR THE C LANGUAGE

- How to program an *Operating System* ?
- ASSEMBLY was too hard to maintain and not portable
- High-level languages such as FORTRAN were too far from the machine
- 1969-1972: The C language combines low-level and high-level features
- Trade-off between computer and application point of views

REASONS FOR THE C LANGUAGE

- How to program an *Operating System* ?
- ASSEMBLY was too hard to maintain and not portable
- High-level languages such as FORTRAN were too far from the machine
- 1969-1972: The C language combines low-level and high-level features
- Trade-off between computer and application point of views
- Low-level features: Direct access to memory (pointers)

REASONS FOR THE C LANGUAGE

- How to program an *Operating System* ?
- ASSEMBLY was too hard to maintain and not portable
- High-level languages such as FORTRAN were too far from the machine
- 1969-1972: The C language combines low-level and high-level features
- Trade-off between computer and application point of views
- Low-level features: Direct access to memory (pointers)
- High-level features: condition and loop statements, type checking, and a standard library.

REASONS FOR THE C LANGUAGE

- How to program an *Operating System* ?
- ASSEMBLY was too hard to maintain and not portable
- High-level languages such as FORTRAN were too far from the machine
- 1969-1972: The C language combines low-level and high-level features
- Trade-off between computer and application point of views
- Low-level features: Direct access to memory (pointers)
- High-level features: condition and loop statements, type checking, and a standard library.
- Used to be the fastest …

KEY POINTS

- *computers, programs*
- *programming languages, syntax, semantic, imperative, paradigms*
- *instruction set architecture (isa), machine language*
- *assembly language*
- *modular, portable, high-level languages, compilers*
- *operating system*

REFERENCES

- cook and magician: *https://pixabay.com/en/users/graphicmama-team-2641041/*
- simon perrault (yale-nus college): *https://www.yale-nus.edu.sg/about/faculty/simon-perrault/*