Brief Reasoning and History of the C Programming Language

Dmitry V. Luciv

Chair of Software Engineering

Contents

Dennis Ritchie: The Development of the C Language

https://www.bell-labs.com/usr/dmr/www/chist.html

- Setting
- Origins in Other Languages
- Later Usage

We will address to it

Computer Generations

1. 1940s–1950s. Relays and vacuum tubes: 10^5 watts, many rooms; available for military purposes (and then for other physical computations)
2. 1950s–1960s. Semiconductors (transistors, diodes): 10^4 watts, several racks; available for large institutions, banks
3. 1960s–1970s. Integrated circuits: $10^2 - 10^3$ watts, one or several racks; available for smaller institutions and laboratories
4. 1970s–1980s–now. Microporcessors in single integrated circuit: $10 - 10^2$ watts, one box, available for any organization and later for personal use

Setting

Common approach of 1960s

- Mainframes like IBM/360 or GE-645
- Programming languages like PL/I
- Operating systemc like OS/360 or Multics
- Batch control approach like JCL
- No powerful interactive shell

Everything is complicated and heavy-weight

Setting

Common approach of 1960s

- Mainframes like IBM/360 or GE-645
- Programming languages like PL/I
- Operating systemc like OS/360 or Multics
- Batch control approach like JCL
- No powerful interactive shell

Everything is complicated and heavy-weight

New approach of 1970s

- Simpler *and cheaper* mini-computers like DEC PDP-7
- More universal use of them
- Many computer families

Birth of Unix

Killer-features of Unix:

- Hierarchical file system with single tree of file names
- Agnostic approach to file data: before, data was usually stored in formatted files, which offered good throughput but were complicated for software developers
- Interactive powerful shell running in user space

Birth of Unix

Killer-features of Unix:

- Hierarchical file system with single tree of file names
- Agnostic approach to file data: before, data was usually stored in formatted files, which offered good throughput but were complicated for software developers
- Interactive powerful shell running in user space
- And one more feature that we will describe later...

Before and After Unix

Note:

- Multics already offered many of above features, but still was too complicated; minimalist design was desired
- Above approaches were very good finding and they are still actual after 50 years: we see elements of such a design in such OSs as DOS and then Windows

See a pretty nice AT&T documentary on this:

- https://youtu.be/tc4ROCJYbm0

Popular languages of 1960s and before

- Fortran: one of the first, high-level, computational
- COBOL: business-oriented language
- PL/I: general purpose complicated language, suited better for systems programming than above two
- Assembly languages for many computer architectures, not portable

Popular languages of 1960s and before

- Fortran: one of the first, high-level, computational
- COBOL: business-oriented language
- PL/I: general purpose complicated language, suited better for systems programming than above two
- Assembly languages for many computer architectures, not portable
- *All above — not only assembly* — were not very portable
- They were not structural languages, which lead to poor quality code which was difficult to maintain due to spaghetti-code

Birth of C

- Compiled language
- Structural language
- Good for systems programming
- Simple enough and portable

What is structural programming?

Program consists of:

- Sequential blocks of operators
- Loops
- Branching (**`if`** — **`else`** — ...)
- All above can be used withing each other

What is structural programming?

Program consists of:

- Sequential blocks of operators
- Loops
- Branching (**`if`** – **`else`** – ...)
- All above can be used withing each other

Böhm-Jecopini theorem: above are enough to express any algorithm in sense of Turing-completeness

What is structural programming?

Program consists of:

- Sequential blocks of operators
- Loops
- Branching (**`if`** – **`else`** – ...)
- All above can be used withing each other

Böhm-Jecopini theorem: above are enough to express any algorithm in sense of Turing-completeness

Additionally:

- **`goto`** is available but not welcome
- Procedures!
- Clean variable scopes (not as in Basic or Python!)

More links to look at

- Notes on Structured Programming. By Prof. Dr. Edsger W. Dijkstra — T. H. Report 70-WSK-03 Second Edition April 1970
- Dijkstra: EWD 215: A Case against the GO TO Statement (PDF).

C Predecessors

- 1960: Algol-60
- 1963: CPL
- 1967: BCPL
- 1969: B
- 1972: C

C and Unix evolved together

- C was portable (not referring any particular architecture properties)
- C was simple enough to create new compiler targets quickly
- C suited well for systems programming

C and Unix evolved together

- C was portable (not referring any particular architecture properties)
- C was simple enough to create new compiler targets quickly
- C suited well for systems programming

In beginning of 1970s the majority of Unix code was re-implemented in C, which was one of the reasons of its popularity till now. Now we have it in servers, networking hardware, PCs, mobiles etc.

1980s–1990s

- Cheap PCs
- Internet

1990s–2000s–now

- Many mobile and embedded architectures
- Parallel architectures

Questions please!

EDU.DLUCIV.NAME