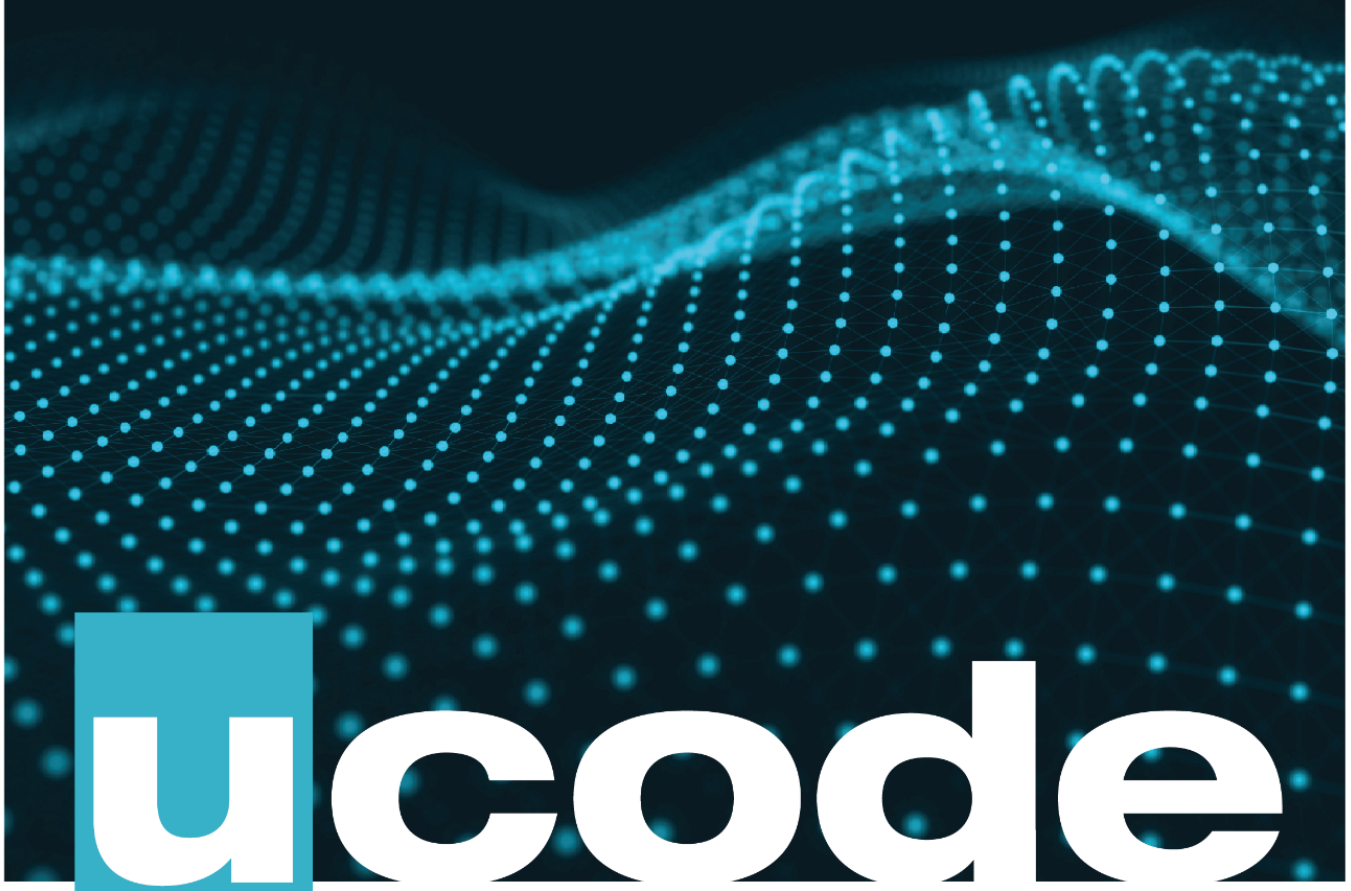
ush

Track C

January 11, 2021



ucode

Contents

Engage

DESCRIPTION

Hello, ucoders!

Have you ever thought what a user interface (UI) is, in general? Did you know that this term doesn't only relate to computers and software design?
There are many physical interfaces that surround us in everyday life, such as water taps, cars, keyboards, etc. Basically, everything can be called as an interface if it helps the user to interact with the object/device. However, the term UI is often used in the sense of software and electronic devices, and provides a layer between the human and the computer.
There are command-line interfaces (CLI), graphical user interfaces (GUI), text user interfaces (TUI), etc. You can read more about it here. That can be used during software design.

The Terminal is one of the very first tools you've used in ucode. A Unix shell is a command-line interpreter that provides a user interface for Unix-like operating systems. The most generic sense of the term shell means any program that users employ to type commands. A shell hides the details of the underlying operating system and manages the technical details of the operating system kernel interface, which is almost the lowest-level component of most operating systems.

The previous projects have already given you a basic understanding of Unix systems. In this challenge, you will dig deeper. The challenge invites you to find out how shell works internally. Also, you have a chance to develop your ideal shell.

Imagine you are a computer science pioneer in 1960s. GUI has not been invented yet. You need to develop a user-friendly interface to execute commands on the computer.

BIG IDEA

User interface.

ESSENTIAL QUESTION

How users can interact with OS kernel?

CHALLENGE

Develop Unix shell.

ush | Track C **> 2**

Investigate

GUIDING QUESTIONS

We invite you to find answers to the following questions. By researching and answering them, you will gain the knowledge necessary to complete the challenge. To find answers, ask the students around you and search the internet. We encourage you to ask as many questions as possible. Note down your findings and discuss them with your peers.

- Do you prefer `Terminal`, `iTerm` or `cmd.exe` to execute shell?
- What is your favorite shell `zsh`, `bash`, `csh` or `powershell`?
- What does the term `process` mean in computing?
- How does the shell manage commands?
- What is a step-by-step shell algorithm (reading, interpreting, and execution commands)?
- What are shell builtins? Why does the shell require them?
- How to run an external program in a separate process?
- What are the pros and cons of builtins compared to external programs?
- How to create a new process?
- What are daemon processes?
- What shell features are needed first of all?
- What does a prompt stand for?
- What is the difference between terminal, shell and console?
- What are the environment variables and how does the shell interact with them?
- What is `POSIX`?
- What does the standard C library for POSIX systems consist of?

GUIDING ACTIVITIES

Complete the following activities. Don't forget that you have a limited time to overcome the challenge. Use it wisely. Distribute tasks correctly.

- Read the story carefully. Find out what you have to do.
- Read the docs of shells (e.g. sh, zsh, csh, bash) and try to understand the differences.
- Experiment with standard shells. Learn their features. There you'll find out additional information that is not listed in the docs.
- Read about exotic shells.
- Use `zsh` shell as a reference for this challenge.
- Read the documentation about every component of the solution.
- Collaborate with other students to get a deeper understanding of the challenge. Learn from others and share your knowledge.
- Build a great team. A great team is the one whose work is based on clear objectives, clear roles, clear communication, cooperation, and opportunities for personal development.
- Establish communication. Get to know each other closer.

ush | Track C **> 3**

- It is important to set clear milestones and deadlines for the process and products. We strongly recommended using a task manager to organize processes in the team (e.g. Trello, Jira, etc.). Use meetings for periodic monitoring of the deadlines and for the development process.
- Clone your git repository that is issued on the challenge page.
- Distribute tasks among team members.
- Start developing your program. Try to implement your thoughts in code.
- Open Auditor and follow the rules.
- Explore new things.
- Start developing your implementation.
- Test your code.

Building an effective team is one of the most important responsibilities for all of us. It's not something you can instantly achieve. It's an ongoing process, which requires constant attention and evaluation.

ANALYSIS

Analyze your findings. What conclusions have you made after completing guiding questions and activities? In addition to your thoughts and conclusions, here are some more analysis results.

- Challenge has to be carried out by an entire team.
- Each team member must understand the challenge and realization, and be able to reproduce it individually.
- It is your responsibility to assemble the whole team. Phone calls, SMS, messengers are good ways to stay in touch.
- Be attentive to all statements of the story.
- The challenge must be performed in `C`.
- This time, each line of text in your code must be at most 100 characters long.
- Perform only those tasks that are given in the story.
- The challenge must have the following structure:
  - `src` directory contains files with extension `.c`. Or the `src` directory can consist of subdirectories with `.c` files
  - `obj` ddirectory contains files with extension `.o` (you must not push this directory in your repository, only `Makefile` creates it during compilation)
  - `inc` directory contains header files `.h`. Or the `inc` directory can consist of subdirectories with `.h` files
  - `libmx` directory contains source files of your library including its `Makefile`. Recommended, but not required
  - `Makefile` that compiles the library `libmx` firstly and then compiles and builds `ush`

- You can proceed to `Act: Creative` only after you have completed all requirements in `Act: Basic`. But before you begin to complete the challenge, pay attention to the program's architecture. Take into account the fact that many features indicated in the `Act: Creative` require special architecture. And in order not to rewrite all the code somewhen, we recommend you initially determine what exactly you will do in the future. Note that the `Act: Basic` part gives the minimum points to validate the challenge.
- Complete the challenge according to the rules specified in the `Auditor`.
- Submit your files using the layout described in the story. Only useful files allowed, garbage shall not pass!
- Compile C-files with clang compiler and use these flags: `clang -std=c11 -Wall -Wextra -Werror -Wpedantic`.
- Pay attention to what is allowed. Use of forbidden stuff is considered a cheat and your challenge will be failed.
- Your program must manage memory allocations correctly. A memory that is no longer needed must be freed, otherwise, the challenge is considered incomplete.
- Check the memory leaks with the `leaks tool` on the macOS.
- Check the memory leaks with the `valgrind tool` on the AWS Cloud9.
- The solution will be checked and graded by students like you. `Peer-to-Peer learning`.
- Also, the challenge will pass automatic evaluation which is called `Oracle`.
- If you have any questions or don't understand something, ask other students or just Google it.

ush | Track C **> 5**

Act: Basic

ALLOWED

C POSIX library, `errno` and `environ` global variables

BINARY

ush

DESCRIPTION

Create a basic command-line interpreter with features without which there can be no shell. As a reference, you must take zsh.

Your shell must:

- have the default prompt must look like `u$h>`, followed by a space character
- deal only with one line user input. In other cases, the appropriate descriptive error message must be displayed
- implement builtin commands without flags: `export, unset, exit, fg`
- implement the following builtin commands with flags:
  - `env` with `-i`, `-P`, `-u`
  - `cd` with `-s`, `-P` and `-` argument
  - `pwd` with `-L`, `-P`
  - `which` with `-a`, `-s`
  - `echo` with `-n`, `-e`, `-E`
- find builtins or flags in manuals to other shells if zsh hasn't got them
- call the builtin command instead of the binary program if there is a name match among them
- correctly manage errors
- manage user environment correctly
- run programs located in the directories listed in the `PATH` variable
- implement the command separator `;`
- implement management of the following expansions correctly:
  - the tilde expansion `~` with the following tilde-prefixes: `~`, `~/dir_name`
  - the basic form of parameter expansion `${parameter}`

The purpose of this challenge is to learn the system APIs, so you are allowed to use any functions of the C POSIX library. This implies that you are forbidden to use any third-party libraries, except as expressly indicated in the story. If you need it, you must develop it and consider adding to your `libmx` for future use.

ush | Track C **> 6**

CONSOLE OUTPUT

```
>echo "cd /tmp ; pwd" | ./ush
/tmp
>./ush
u$h> unknown
ush: command not found: unknown
u$h> echo $SHLVL
2
u$h> cd ~xlogin
u$h> pwd
/Users/xlogin
u$h> cd ~/Desktop
u$h> pwd
/Users/xlogin/Desktop
u$h> echo "${PWD}"
/Users/xlogin/Desktop
u$h> mkdir tricky\ dir
u$h> touch tricky\ dir/more\ tricky\`file
u$h> ls tricky\ dir/
more tricky`file
u$h> rm -rf tricky\ dir
u$h> cd -
u$h> rm -rf ush
u$h> echo "Bye!
Odd number of quotes.
u$h> echo 'F*ck. Bye!'
F*ck. Bye!
u$h> exit
>
```

FOLLOW THE WHITE RABBIT

- man zsh
- man builtin
- man zshbuiltins
- man env
- man environ
- man fork
- man execve
- man posix_spawn
- man 2 wait
- man 2 stat
- man 3 exit
- man 3 signal
- man 2 kill

ucode

ush | Track C **> 7**

Act: Creative

ALLOWED FUNCTIONS

C standard library, termcap library

DESCRIPTION

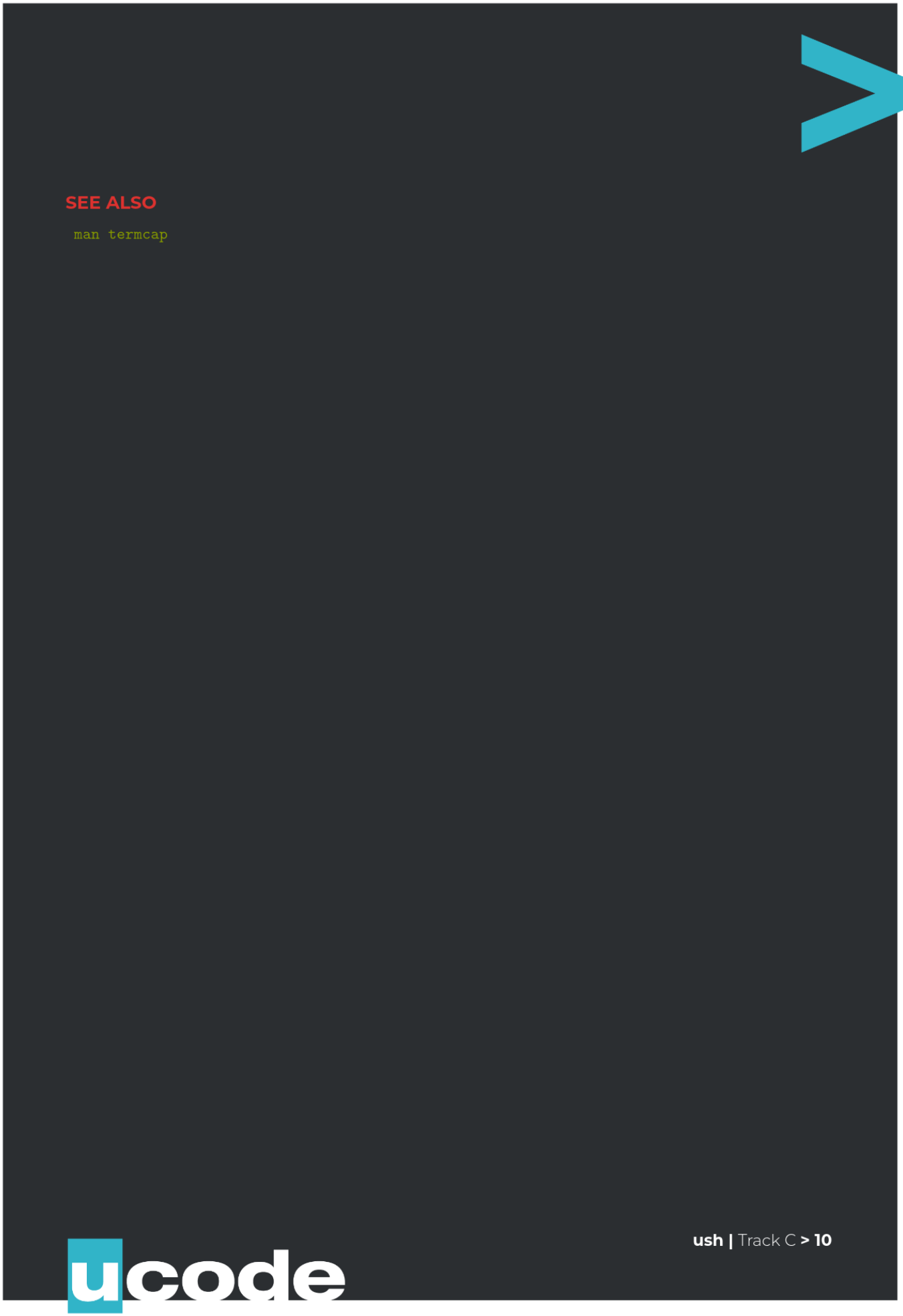
It is the place where your imagination and creativity plays a major role. Implement additional features to make the program better and more unique. Listed below are a few ideas you can add to your program. You can come up with everything you want to improve your program. Creative features:

- allow users to customize prompt and to make it unique and useful e.g. show current directory, git info, etc.
- implement command editing. Move the cursor using the `Arrow` keys or the `HOME` and `END` keys
- add support of command history. Sequential search e.g. with `Page Up` and `Page Down` keys. Or query search with `CTRL+R`
- implement management of signals `CTRL+D`, `CTRL+C` and `CTRL+Z`
- implement advanced management of the following expansions correctly:
  - the tilde expansion `~` with the following tilde-prefixes: `~username/dir_name`, `~+/dir_name`, `~-/dir_name`
  - the two-level-nested command substitution `$(command)`
- implement the `fg` builtin command without arguments and with arguments, for example, `%n`, `%str`
- implement more builtins of your choice
- implement multiline user input
- add support of the nested command substitution `` `command` `` and `$(command)`
- add support of shell functions `tripple_ls() { ls; ls; ls; }`
- add support of performing arithmetic operations `$((expression))` and their practical usage `echo $(( 10 + 5 / 5 * 121))`
- implement auto-completion using the `TAB` key
- add support of pipes `|`
- add support of redirecting output `>, <, >>, <<`
- add support of logical operators `&&` and `||`
- add support of aliases
- any other useful features you like

Remember, you do not need to create a full-featured shell. Focus on the approximate time of challenge accomplishment on the challenge web page.

ucode

ush | Track C **> 9**


SEE ALSO
man termcap
ucode
ush | Track C > 10

Document

DOCUMENTATION

One of the attributes of Challenge Based Learning is documentation of the learning experience from challenge to solution. Throughout the challenge, you document your work using text and images, and reflect on the process. These artifacts are useful for ongoing reflection, informative assessment, evidence of learning, portfolios, and telling the story of challenge. The end of each phase (Engage, Investigate, Act) of the challenge offers an opportunity to document the process.

Much of the deepest learning takes place by considering the process, thinking about one's own learning, analyzing ongoing relationships with the content and between concepts, interacting with other people, and developing a solution. During learning, documentation of all processes will help you analyze your work, approaches, thoughts, implementation options, code, etc. In the future, this will help you understand your mistakes, improve your work, and read the code.

At the learning stage, it is important to understand and do this, as this is one of the skills that you will need in your future job. Naturally, the documentation should not be voluminous, it should be drawn up in an accessible, logical, and connected form.

So, what must be done?

- a nice-looking and helpful README file. In order for people to want to use your product, their first introduction must be through the README on the project's git page. Your README file must contain:
  - Short description. This means, that there must be some info about what your project actually is. For example, what your program does.
  - Screenshots of your solution. This point is about screenshots of your project "in use".
  - Requirements and dependencies. List of any stuff that must be installed on any machine to build your project.
  - How to run your solution. Describe the steps from cloning your repository to the first launch of your program.
- a full-fledged documentation in any forms convenient for you. By writing this, you will get some benefits:
  - you have an opportunity to think through implementation without the overhead of changing code every time you change your mind about how something should be organized. You will have very good documentation available for you to know what you need to implement
  - if you work with a development team and they have access to this information before you complete the project, they can confidently start working on another part of projects that will interact with your code
  - everyone can find how your project works
- your documentation must contain:
  - Description of progress after every competed CBL stage.
  - Description of the algotithm of your whole program.

ush | Track C **> 11**

Keep in mind that the implementation of this stage will be checked by peers at the assessment!

Also, there are several links that can help you:

- Make a README
- How to write a readme.md file?
- A Beginners Guide to writing a README
- Google Tools - a good way to journal your phases and processes:
  - Google Docs
  - Google Sheets
- Dropbox Paper - a tool for internally developing and creating documentation
- Git Wiki - a section for hosting documentation on Git-repository
- Haroopad - a markdown enabled document processor for creating web-friendly documents
- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio
- code commenting - source code clarification method. The syntax of comments is determined by the programming language
- and others to your taste

ush | Track C **> 12**

Share

PUBLISHING

Last but not least, the final stage of your work is to publish it. This allows you to share your challenges, solutions, and reflections with local and global audiences. During this stage, you will discover ways of getting external evaluation and feedback on your work. As a result, you will get the most out of the challenge, and get a better understanding of both your achievements and missteps.

To share your work, you can create:

- a text post, as a summary of your reflection
- charts, infographics or other ways to visualize your information
- a video, either of your work, or a reflection video
- an audio podcast. Record a story about your experience
- a photo report with a small post

Helpful tools:

- Canva - a good way to visualize your data
- QuickTime - an easy way to capture your screen, record video or audio

Examples of ways to share your experience:

- Facebook - create and share a post that will inspire your friends
- YouTube - upload an exciting video
- GitHub - share and describe your solution
- Telegraph - create a post that you can easily share on Telegram
- Instagram - share photos and stories from ucode. Don't forget to tag us :)

Share what you've learned and accomplished with your local community and the world. Use #ucode and #CBLWorld on social media.