

Version: 1.0



Selection

Rosetta_Stone

Summary

Create a program that converts numbers into their written words using a dictionary for customizable value mappings.

#C

#Impativerive

#Unix

42

Intellectual Property Disclaimer

All content presented in this training module, including but not limited to texts, images, graphics, and other materials, is protected by intellectual property rights held by Association 42.

Terms of Use:

- **Personal use:** You are permitted to use the contents of this module solely for personal purpose. Any commercial use, reproduction, distribution, modification, or public display is strictly prohibited without prior written permission from Association 42.
- **Respect for Integrity:** You must not alter, transform, or adapt the content in any way that could harm its integrity.

Protection of Rights:

Any violation of these terms constitutes an infringement of intellectual property rights and may result in legal action. We reserve the right to take all necessary measures to protect our rights, including but not limited to claims for damages.

For any questions regarding the use of the content or to obtain authorization, please contact:
legal@42.fr

Contents

1	Instructions	1
2	Foreword	4
3	Mandatory	5
4	Submission and peer-evaluation	7

Chapter 1

Instructions

- The group WILL be registered to defense. Automatically. If one of you cancels it, there won't be another one.
- Any question regarding the subject will probably complicate the subject.
- You have to follow the submission procedures for all your exercises.
- This subject could change up to an hour before submission.
- The program must compile with the following flags: `-Wall -Wextra -Werror`; and uses `cc`.
- If your program doesn't compile, you'll get 0.
- Your program must be written in accordance with the Norm. If you have bonus files/functions, they are included in the norm check and you will receive a 0 if there is a norm error inside.
- You must turn in a Makefile, which will compile your project using the rules `$NAME`, `clean` and `fclean`
- You must therefore do the project with the imposed team and show up at the defense slot you've assigned, with all of your teammates.
- Each member of your group must be fully aware of the works of the project. Should you choose to split the workload, make sure you all understand what everybody's done. During defense, you'll be asked questions, and the final grade will be based on the worst explanations.
- Gathering the group is your responsibility. You've got all the means to get in contact with your teammates: phone, email, carrier pigeon, spiritism, etc. So don't bother blurping up excuses. Life isn't always fair, that's just the way it is.
- However, if you've really tried everything one of your teammates remains unreachable : do the project anyway, and we'll try and see what we can do about it during defense. Even if the group leader is missing, you still have access to the submission directory.

● Context

The C Piscine is intense. It's your first big challenge at 42 — a deep dive into problem-solving, autonomy, and community.

During this phase, your main objective is to build your foundation — through struggle, repetition, and especially **peer-learning** exchange.

In the AI era, shortcuts are easy to find. However, it's important to consider whether your AI usage is truly helping you grow — or simply getting in the way of developing real skills.

The Piscine is also a human experience — and for now, nothing can replace that. Not even AI.

For a more complete overview of our stance on AI — as a learning tool, as part of the ICT curriculum, and as a growing expectation in the job market — please refer to the dedicated FAQ available on the intranet.

● Main message

- 👉 Build strong foundations without shortcuts.
- 👉 Really develop tech & power skills.
- 👉 Experience real peer-learning, start learning how to learn and solve new problems.
- 👉 The learning journey is more important than the result.
- 👉 Learn about the risks associated with AI, and develop effective control practices and countermeasures to avoid common pitfalls.

● Learner rules:

- You should apply reasoning to your assigned tasks, especially before turning to AI.
- You should not ask for direct answers to the AI.
- You should learn about 42 global approach on AI.

● Phase outcomes:

Within this foundational phase, you will get the following outcomes:

- Get proper tech and coding foundations.
- Know why and how AI can be dangerous during this phase.

● Comments and example:

- Yes, we know AI exists — and yes, it can solve your activities. But you're here to learn, not to prove that AI has learned. Don't waste your time (or ours) just to demonstrate that AI can solve the given problem.
- Learning at 42 isn't about knowing the answer — it's about developing the ability to find one. AI gives you the answer directly, but that prevents you from building your own reasoning. And reasoning takes time, effort, and involves failure. The path to success is not supposed to be easy.
- Keep in mind that during exams, AI is not available — no internet, no smartphones, etc. You'll quickly realise if you've relied too heavily on AI in your learning process.
- Peer learning exposes you to different ideas and approaches, improving your interpersonal skills and your ability to think divergently. That's far more valuable than just chatting with a bot. So don't be shy — talk, ask questions, and learn together!
- Yes, AI will be part of the curriculum — both as a learning tool and as a topic in itself. You'll even have the chance to build your own AI software. In order to learn more about our crescendo approach you'll go through in the documentation available on the intranet.

✓ Good practice:

I'm stuck on a new concept. I ask someone nearby how they approached it. We talk for 10 minutes — and suddenly it clicks. I get it.

✗ Bad practice:

I secretly use AI, copy some code that looks right. During peer evaluation, I can't explain anything. I fail. During the exam — no AI — I'm stuck again. I fail.

Chapter 2

Foreword

Here is a old-fashioned pecan pie recipe for you :

Ingredients

- Pastry dough
- 3/4 stick unsalted butter
- 1 1/4 cups packed light brown sugar
- 3/4 cup light corn syrup
- 2 teaspoon pure vanilla extract
- 1/2 teaspoon grated orange zest
- 1/4 teaspoon salt
- 3 large eggs
- 2 cups pecan halves (1/2 pound)

Accompaniment: whipped cream or vanilla ice cream

Preparation:

Preheat oven to 350°F with a baking sheet on middle rack.

Roll out dough on a lightly floured surface with a lightly floured rolling pin into a 12 inch round and fit into a 9 inch pie plate.

Trim edge, leaving a 1/2-inch overhang.

Fold overhang under and lightly press against rim of pie plate, then crimp decoratively.

Lightly prick bottom all over with a fork.

Chill until firm, at least 30 minutes (or freeze 10 minutes).

Meanwhile, melt butter in a small heavy saucepan over medium heat.

Add brown sugar, whisking until smooth.

Remove from heat and whisk in corn syrup, vanilla, zest, and salt.

Lightly beat eggs in a medium bowl, then whisk in corn syrup mixture.

Put pecans in pie shell and pour corn syrup mixture evenly over them.

Bake on hot baking sheet until filling is set, 50 minutes to 1 hour.

Cool completely.

Cooks notes:

Pie can be baked 1 day ahead and chilled. Bring to room temperature before serving.

Chapter 3

Mandatory

	Exercise: 00	
rush-02		
Directory: ex00/		
Files to Submit: Makefile and all the necessary files		
Authorized: write, malloc, free, open, read, close		

- Create a program that takes a number as argument and converts it to its written letters value.
- Executable name: `rush-02`
- Your source code will be compiled as follows :

```
make fclean
make
```

- Your program can take up to 2 arguments:
 - If there is only one argument, it is the value you need to convert
 - If there are two arguments, the first argument is the new reference dictionary and the second argument is the value you need to convert.
- If the argument isn't a positive number, your program must return "Error", followed by a newline.



Your program should go beyond unsigned int.

- Your program must parse the dictionary given as resource to the project. The values inside it must be used to print the result. These values can be modified.

- Any memory allocated on the heap (with `malloc(3)`) must be freed correctly. This will be verified during evaluation.
 - The dictionary will have the following rules:

Dictionary rules:

```
[a number][0 to n spaces]:[0 to n spaces][any printable characters]
```

- Numbers are to be handled the same way `atoi` handles them.
- You will trim the spaces before and after the value in the dictionary.
- The dictionary will always have at least the keys as in the reference dictionary. Their value can be modified, more entries can be added, but the initial keys can't be removed.
- You only need to use the initial entries (For instance, if we add 54: `fifty-four`, you still have to use 50: `fifty` and 4: `four`)
- The entries of the dictionary can be stored in any order.
- There can be empty lines in the dictionary.
- If you have any errors from the dictionary parsing, your program must output `"Dict Error\n"`
- If the dictionary doesn't allow you to resolve the asked value, your program must output `"Dict Error\n"`.

Example terminal Output:

```
$> ./rush-02 42 | cat -e
forty two$
$> ./rush-02 0 | cat -e
zero$
$> ./rush-02 10.4 | cat -e
Error$
$> ./rush-02 100000 | cat -e
one hundred thousand$
$> grep "20" numbers.dict | cat -e
20      :      hey      everybody !$
$> ./rush-02 20 | cat -e
hey everybody !$
```

Chapter 4

Submission and peer-evaluation

Turn in your assignment in your `Git` repository as usual. Only the work inside your repository will be evaluated during the defense. Don't hesitate to double check the names of your files to ensure they are correct.

As these assignments are not verified by a program, feel free to organize your files as you wish, as long as you turn in the mandatory files and comply with the requirements.



You need to return only the files requested by the subject of this project.