PSEUDOCODE

CSC100 Introduction to programming in C/C++ (Spring 2024)

README

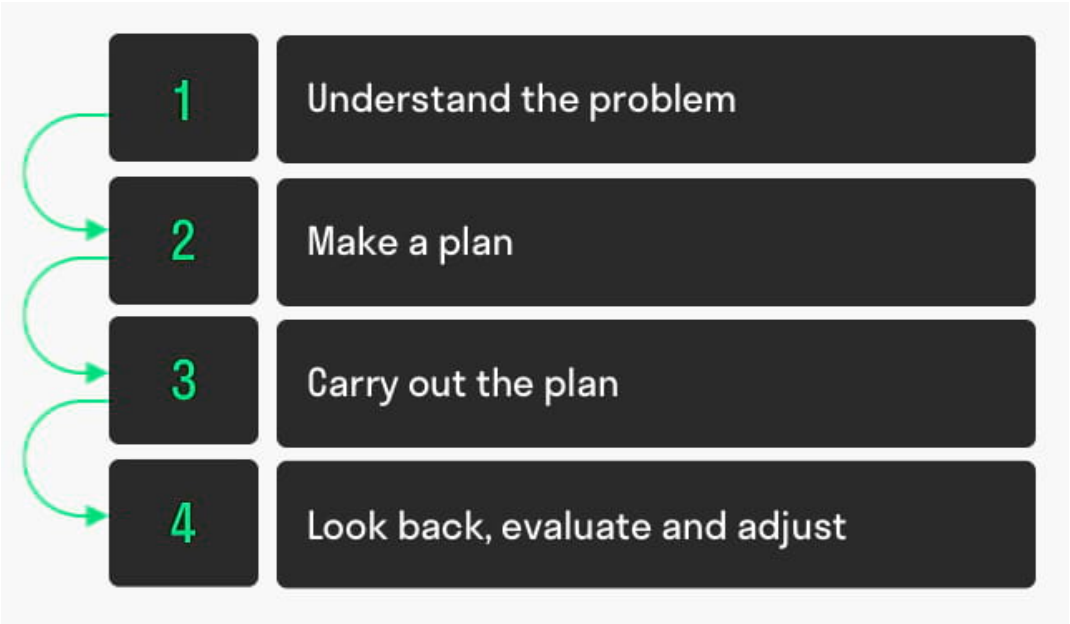
- In this section of the course, we go beyond simple statements and turn to program flow and evaluation of logical conditions
- This section follows chapter 3 in Davenport/Vine (2015) and chapters 4 and 5 in King (2008)
- Practice workbooks, input files and PDF solution files in GitHub

Overview

- **Pseudocode** is a method to write down/analyze an *algorithm* or a *heuristic* without having to bother with *syntax* (like `&i` vs. `i`)
- The prefix pseudo- comes from Ancient Greek , meaning "lying", "false" or "untrue", as in "pseudoscience" or "pseudonym"
- Pseudocode does not need to compile or run so it is closer to a heuristic than to an exact algorithm.
- Code however needs to be exact and is always algorithmic.
- **Always start with pseudocode** before coding, and when you're stuck (not because of syntax ignorance) go back to pseudocode.

Algorithms vs. heuristic - Pólya's problem solving method

The heuristic method à la George Pólya in four steps:



See also "How To Solve It" in Chat for more details:

1. **Understanding the problem:**
   - What is the unknown? What are the data? What is the condition? Is it possible to satisfy the condition? Is the condition sufficient to determine the unknown? Or is it insufficient? Or redundant? Or contradictory?
   - Draw a figure. Introduce suitable notation
   - Separate the various parts of the condition. Can you write them down?
2. **Devising a plan:**
   - Have you seen it before? Have you seen the same problem in a slightly different form? Do you know a related problem? Do you know a theorem or a law or a formula that could be useful?
   - Look at the unknown: Try to think of a familiar problem having the same or a similar unknown.
   - If you have a problem related to yours and solved before: could you use it? Could you use its result? Could you use its method? Should you introduce some auxiliary element in order to make its use possible?

- Could you restate the problem? Could you restate it still differently? Go back to the definitions.
- If you cannot solve the proposed problem try to solve first some related problem. Could you imagine a more accessible related problem? A more general problem? A more special problem? An analogous problem? Could you solve a part of the problem? Keep only part of the condition, drop the other part: how far is the unknown then determined, how can it vary? Could you derive something useful from the data? Could you think of other data appropriate to determine the unknown? Could you change the unknown or the data, or both if necessary, so that the new unknown and the new data are nearer to each other?
- Did you use all the data? Did you use the whole condition? Have you taken into account all essential notions involved in the problem?

3. **Carrying out the plan:** Carrying out your plan of the solution, check each step. Make sure that you understand each step and its consequences. Can you see clearly that the step is correct? Can you prove that it is correct?
4. **Looking back:** Can you check the result? Can you check the argument? Can you derive the result differently? Can you see it at a glance? Can you use the result, or the method, for some other problem?

Pseudocode example

- Example: player problem statement in below:

  "Drink a health potion when a character's health is 100 or less. If health reaches 100 or more, resume battle."

- Given the problem , this is the pseudocode [1]:

```
if health is less than 100
```

[1] In Org mode, you can use the language as an example header argument to enable syntax highlighting. For pseudocode, this will of course not work perfectly, since most syntax elements are not in C.

```
  Drink health potion
else
  Resume battle
end if
```

- The code in would not compile as a C program (you can test yourself: which mistakes would the compiler find?[2])
- The conceptual "trick" with generating pseudocode from a prose description is to identify the **logical condition** so that you can perform a comparison (= apply a **conditional operator**)
- The pseudocode leads to the condition `health < 100`:

```
if health < 100
  Drink health potion
else
  Resume battle
end if
```

- Notice that you could also use another operator: `>=` This operator would have had the same effect but it is not what you were supposed to code. How would the pseudocode change with `health >= 100`?

```
if health >= 100
   Resume battle
else
   Drink health potion
end if
```

- **Rule:** when making models (via **abstraction**), always stay as close to the **problem** description as possible - in terms of **language**, **logic**, **tone**, etc If you're unsure, **ask** (your client/professor/colleague).
- In the pseudocode example above, what relates to 1) language, 2) logic, and 3) tone? And 4) are there other specifications?[3]

[2] Undeclared variable `health`, missing closure semi-colons after the statements, functions `Drink` and `Resume` not known, and more.

[3] 1) Language: words used like `health`, `drink` or `resume`. 2) Logic: IFTTT (If This Then That) Else That. 3) Tone: game language 'drink health potion' as function `drink_health_potion`. 4) The indentation and the use of `if else end if` instead of, e.g. "WHEN health < 100 THEN drink health potion OTHERWISE resume battle".