15-110: Principles of Computing

Lecture 3: Simplifying instructions and abstraction

August 07, 2022
Clear algorithms

• **Why is it important for algorithms to be clear?**

• Writing a sequence of steps in English (or in any natural language) can quickly become cumbersome

• More “formal” and less ambiguous language

  ➔ Simplifications
Variables

• In math we commonly use **names** to refer to values
Variables

• In math we commonly use names to refer to values

• **Variables**: provide a way to name information and access and modify the information by using the name

• A named *container* of information

  - What can we do with a variable (e.g., $x$)?
    - **Assign** its value  
      - $x := 2$
    - **Read / use** its value  
      - $y := x + 2$
    - **Modify** its value  
      - $x := 4.5$
Indices

• Sometimes we would like to give names to sequences of objects

Card Deck

$c_0, c_1, c_2, c_3, ..., c_{n-1}$

$C_i$  Card at the ith position in the pile
Finding the maximum card

1. Pick up the first card from the deck pile ($n$ cards)
2. Record down the number $v$ and remove the card from the deck (put it in done pile)
3. Assign the number $v$ to max value

4. Pick up the next card from the deck
5. Look at the number, $v$, and remove the card from the deck
6. If the number is higher than current max value: max value becomes $v$

7. Repeat 4-6 $n - 1$ times (i.e., until no more cards in deck)
8. Output max value
9. Stop
Variables and indices at work

1. Pick up the first card from the deck pile \( n \) cards
2. Record down the number \( v \) and remove the card from the deck (put it in done pile)
3. Assign the number \( v \) to max value
4. Pick up the next card from the deck
5. Look at the number, \( v \), and remove the card from the deck
6. If the number is higher that current max value: max value becomes \( v \)
7. Repeat 4-6 \( n - 1 \) times (i.e., until no more cards in deck)
8. Output max value
9. Stop

1. Define a variable to hold the number of cards: \( n \), e.g., \( n = 52 \)
2. Label the cards values with a set of indices: \( c_0, c_1, c_2, \ldots, c_{n-1} \)
3. Define a variable \( m \) to hold the best value so far
4. \( m := c_0 \)
5. Card index variable, initialized to 1: \( i := 1 \)
6. Check if \( i < n \):
    7. if yes: Check if \( c_i > m \)
    8. if yes: \( m := c_i; i := i + 1; \) go back to step 6
    9. if no: \( i := i + 1; \) go back to step 6
10. If no: highest card is \( m \)
Using a *Repeat for* directive

1. **Input:** Let \( n \) be the number of cards
2. **Input:** Let \( c_0, c_1, c_2, \ldots, c_{n-1} \) be the card values
3. Let \( \text{max} \) be the highest card we have seen
4. \( \text{max} := c_0 \)
5. Let \( i \) be an *index variable*
6. **Repeat for** \( i := 1, \ldots, n-1 \)
7. if yes: Check if \( c_i > \text{max} \)
8. if yes: \( \text{max} := c_i \)
9. **Output:** highest card is \( \text{max} \)

---

**Before:**

1. Define a *variable* to hold the number of cards: \( n \), e.g., \( n = 52 \)
2. Label the cards values with a *set of indices*: \( r_0, c_1, c_2, \ldots, c_{n-1} \)
3. Define a *variable* \( \text{max} \) to hold the best value so far
4. \( \text{max} := c_0 \)
5. Card *index variable*, initialized to 1: \( i := 1 \)
6. Check if \( i < n \):
7. if yes: Check if \( c_i > \text{max} \)
8. if yes: \( \text{max} := c_i; \ i := i + 1; \text{go back to step 6} \)
9. if no: \( i := i + 1; \text{go back to step 6} \)
10. If no: highest card is \( \text{max} \)

---

**What the *Repeat for* directive does?**
Find another problem that can be solved using the same algorithm. Be creative!

First you take your ______________ then add a layer of ______________
before you pour on a hearty dose of ______________.
Next, press some ______________ down into the ______________ before
covering with a sprinkle of ______________.
That’s how I make a ______________!