Algorithm Discovery and Design

Topics:

Representing Algorithms
Sequential Algorithms

Prof. Dr. Slim Abdennadher

1.11.2007

Objectives

By the end of this lecture, you will be able to:

- **Express simple solutions using sequential operations**
- **Represent sequential algorithms using pseudo-code**

What is an Algorithm?

Informally:

An **algorithm** is a step by step method for solving a problem

Why are Algorithms important?

- If we can specify an algorithm to perform a task, then we can instruct a **computing agent** to execute it and solve the problem for us.
- A **computing agent** is an entity capable of performing the steps described in the algorithm, that is, execute the algorithm
- In our case, typically a computer.

What is an Algorithm?

Formal Definition:

An algorithm is a **well-ordered** *collection of* **unambiguous** *and* **effectively computable operations** *that, when executed,* **produces a result** *and* **halts in a finite amount of time**.

- An algorithm is **well-ordered**: each step of the algorithm is executed in the order in which it is written, or else the order is clearly stated.
- An algorithm is **unambiguous**: The algorithm must be clearly stated, in terms that the computing agent (e.g computer) understands
- An algorithm is **effectively computable**: It must be possible for the computing agent to perform the operation and produce a result
- An algorithm must **halt in a finite amount of time**: must even if it would take centuries to finish

Representing Algorithms

- What language to use?
  - **Expressive**
  - **Clear, precise and unambiguous**
- For example, we could use:
  - Natural Languages (e.g. English)
  - Formal Programming Languages (e.g. Java, C++)
  - Something else?

Representing Algorithms: Natural Languages

Example:

Given is a natural number n. Compute the sum of numbers from 1 to n.

Representation with Natural Language

Initially, set the value of the variable `result` to `0` and the value of the variable `i` to `1`. When these initializations have been completed, begin looping until the value of the variable `i` becomes greater than `n`. First, add `i` to `result`. Then add `1` to `i` and begin the loop all over again.

Disadvantages:

- too verbose
- unstructured
- too rich in interpretation (ambiguous)
- imprecise

Representing Algorithms: Formal Programming Language

Example:

Given is a natural number n. Compute the sum of numbers from 1 to n.

Representation with Formal Programming Language (Java)

```
public class Sum {
  public static void main(String[] args)
  {  int result = 0;
     int n = Integer.parseInt(args[0]);
     int i = 1;
     while (i <= n) {
        result = result + i;
        i =   i + 1;}
  System.outp.println(result);}}
```

Disadvantages:

- Too many implementation details to worry about
- Too rigid syntax

Representing Algorithms: Pseudo-code

- We need a compromise between the two:
  $\Rightarrow$ **Pseudo-code**
- Computer scientist use Pseudo-code to express algorithms:
  - English like constructs (or other natural language) but
  - modelled to look like statements in typical programming languages.
  - Not actually executed on computers
  - Allows us to think out a program before writing the code for it

Pseudo-Code: Input/Output and Computation

- **Input operations:** allow the computing agent to receive from the outside world data values to use in subsequent computations.

  General Format:

  ```
  get value for <variable>
  ```

- **Output Operations:** allow the computing agent to communicate results of the computations to the outside world.

  General Format:

  ```
  print  value for <variable>
  print the message ''<text>''
  ```

- **Computation:** performs a computation and stores the result.

  General format: `set  <variable> to <expression>`

Pseudo-code: What Kind of Operations do We Need?

Example: Given the radius of circle, determine the area

- Decide on names for the objects in the problem and use them consistently, e.g. `radius`, `area` (we call them **variables**)
- Use the following primitive operations:
  - Get a value (input: e.g. `get the value of radius`)
  - Print a value or message (output: e.g. `print the value of area` or `print ‘‘Hello’’`)
  - Set the value of an object (e.g. `set the value of Pi to 3.14`)
    General format: `set the value of <variable> to <expression>`
    Performs a computation and stores the result.
  - Arithmetic operations: e.g. $+, -, \times, \ldots$

Pseudo-code: Variables

A **variable** is a named storage

- A value can be stored into it, overwriting the previous value
- Its value can be copied

Example:

- `Set the value of A to 3`

  The variable `A` holds the value three after its execution

- `Set the value of A to A+1`

  Same as: add 1 to the value of `A` (`A` is now 4)

Pseudo-Code: Not too Strict on Syntax

- Pseudo-code is a kind of programming language without a rigid syntax for example we can write:
  - `set the value of A to B+C`

    as

    `set A to B+C`
  - or even:

    ```
    set the value of A to 0
    set the value of B to 0
    ```

    as

    `set A and B to 0`

Algorithms: Operations

Algorithms can be constructed by the following operations:

- **Sequential Operation**
- **Conditional Operation**
- **Iterative Operation**

Sequential Operations

Each step is performed in the order in which it is written

Example 1: Given the radius of circle, determine the area and circumference

- Names for the objects:
  - Input: `radius`
  - Outputs: `area`, `circumference`
- Algorithm in Pseudo-code:

```
get radius
set area to (radius * radius * 3.14)
print area
set circumference to (2 * radius * 3.14)
print circumference
```

Sequential Operations

- **Example 2:** Algorithm for finding the average of three numbers.

```
get A, B, C
set Sum to A+B+C
set Average to Sum/3
print Average
```

- Let `A=12`, `B=10`, `C=8`

  `Sum = 30`

  `Average = 10`

- **Example 3:** The distance between two points (x_1, y_1) and (x_2, y_2) can be calculated using the following equation:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Write an algorithm that calculates the value of d.

```
get    x1,y1,x2,y2
set    d to SQRT((x2-x1)*(x2-x1) + (y2-y1)*(y2-y1))
print  d
```