Computer Science 1510

Lecture 3

Lecture Outline

- Problem solving
- Flowcharts

Programming and Problem solving

- When presented with a problem that you wish to use a computer to solve, there are four main steps that should be followed:

  1. Problem analysis and specification.
  2. Data organization and algorithm design.
  3. Program coding. (Implementation)
  4. Execution and testing.

Problem analysis and specification

- The main step in the analysis of a problem is to formulate a precise specification, for which we have to identify and describe the input and output required.
  - **Input** - the information you already know, or need to know, to start solving the problem.
  - **Output** - what you want to know or determine.

- To allow a program to be used to solve related problems, it should be designed in general terms.

- If a problem is large or complex, it is best to break it into smaller problems and solve the individual simpler problems first.

Data organization and algorithm design

- To organize the data for a particular problem, we must determine the type of data involved (integer numbers, real numbers, characters) and assign labels or variables to the quantities.

- An algorithm is an ordered list of steps, that describes in a complete and detailed manner, a method by which we can solve a particular problem.

- To ensure that an algorithm is correct, it should be tested using sample input to confirm that the correct solution is obtained.

- To help with algorithm design, a combination of natural language and programming language constructs, referred to as *pseudocode*, is often used.

- Pseudocode has no fixed rules.

- Once you have developed and tested an algorithm, the next stage is to translate that algorithm into a program.

Example: Radioactive decay

Suppose you are conducting research on a radioactive element. The half-life of the element is 140 days, which means that because of radioactive decay, the amount of the element remaining after 140 days is one-half of the original amount. You would like to know how much of the element will remain after running an experiment for 180 days if 10 milligrams are present initially.

1. **Problem Analysis and Specification**
   First we have to identify the important pieces of information, and extract the input and output.

   Input: Initial amount, Half-life, Time period.
   Output: Amount remaining.

   Note that although we know the amounts of the input quantities in this case, it is best to develop an algorithm for the problem in general so that it can be used for further instances of the same problem.

2. **Data organization and Algorithm design**
   To organize the data, we can assign names to each of the input and output quantities. For example:

   - `init` for the Initial amount.
   - `halflife` for the Half-life.
   - `time` for the Time period.
   - `amt_remaining` for the Amount remaining.

   To design an algorithm we first begin with a very general breakdown of what needs to be done, and then refine any steps that need to be considered in more detail.

For example, we could start with the following steps:

1. Obtain values for `init`, `halflife`, and `time`.
2. Compute the value of `amt_remaining` for the given `time`.
3. Display `amt_remaining`.

Clearly, step 2 requires further refinement to explain how `amt_remaining` is computed.

If we start with 10 milligrams, then after 140 days we have

$$10(0.5)$$

milligrams remaining. After 280 days we have

$$10(0.5)(0.5) = 10(0.5)^2$$

remaining. After 420 days we have

$$10(0.5)(0.5)(0.5) = 10(0.5)^3$$

remaining. In general, after `time` days we have

$$10(0.5)^{\texttt{time/half-life}}$$

milligrams remaining.

In pseudocode we could write this algorithm as:

1. Get values for `init`, `halflife`, and `time`.
2. Compute `amt_remaining=init*(0.5)**(time/halflife)`.
3. Print `amt_remaining`.

When writing an algorithm, it is best to include a description of the problem that is being solved, a list of the input and output variables, followed by the algorithm steps.

For the above example we would have:

This algorithm calculates the amount of a radioactive substance that remains after a specified time for a given initial amount and a given half-life.

Input: An initial amount of the radioactive substance (`init`), the half-life of the substance (`halflife`), and the time period in days (`time`).

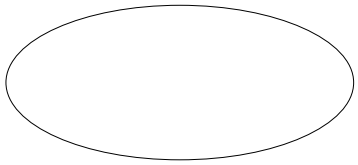
Output: The amount remaining (`amt_remaining`).

1. Get values for `init`, `halflife`, and `time`.
2. Compute `amt_remaining=init*(0.5)**(time/halflife)`.
3. Print `amt_remaining`.

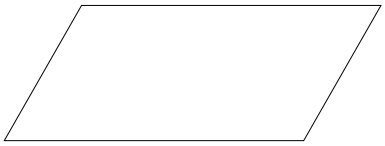
Flowcharts

- A flowchart is a graphical representation of an algorithm.
- Special symbols (blocks) are used for each type of instruction.
- Arrows between blocks indicate the order of execution.
- Each block must have at least one arrow pointing to it and at least one arrow pointing away from it. Except Start (no input) and Stop (no output) blocks.

Flowchart symbols

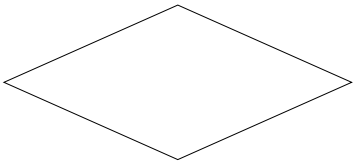


Start / Stop

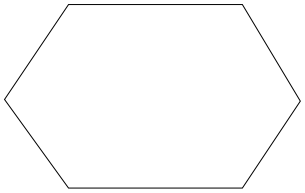


Input / Output

Assignment (calculation)

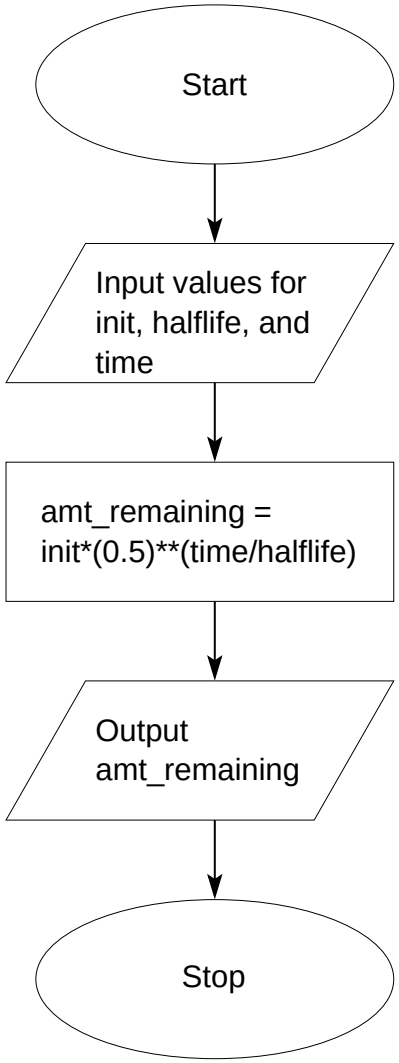


Conditional



Beginning of loop

Flowchart Example: Radioactive Decay



Example: Radioactive decay

3. Program coding (Implementation)

```
PROGRAM Radioactive_decay

!-------------------------------------------------------------
! This program calculates the amount of a radioactive
! substance that remains after a specified time, given
! an initial amount and its half-life.
!  Input:
!    init - initial amount of the substance (mg)
!    time - time at which amt_remaining is calculated (days)
!    halflife - half-life of the substance (days)
!  Output:
!    amt_remaining - amount of the substance remaining
!-------------------------------------------------------------

    IMPLICIT NONE
    REAL::init,time,halflife,amt_remaining

! Prompt the user for the values of init, halflife, and time
    WRITE(*,*) 'Enter the the amount of the substance (mg),'
    WRITE(*,*) 'its halflife (days), and the time at which'
    WRITE(*,*) 'to find the amount remaining (days)'

! Get the values for init, halflife, and time
    READ(*,*) init, halflife, time

! Compute the amount remaining at the specified time
    amt_remaining = init*0.5**(time/halflife)

! Display the amount remaining
    WRITE(*,*) 'There are ',amt_remaining,' mg remaining'

END PROGRAM Radioactive_decay
```

Example: Radioactive decay

4. Execution and Testing

- One way to test an algorithm is to *trace* through the steps with input values for which the correct output values are known.

- Note that it is important to test an algorithm (and program) on several different sets of input data, especially cases that may represent uncommon input (ex. negative values, zero, etc.).

- Once you are confident that your algorithm is correct, and have translated it into source code, then the same test procedure can be used on the executable file created by compiling the code.

Example: Product of x and y

Given two positive integers, x and y, compute the product $x \times y$ by adding $x + x + x + \cdots$ (y times).

The following algorithm could be used:

```
1  get values for x and y
2  set counter=0
3  set sum=0
4  add x to sum (sum=sum+x)
5  add 1 to counter (counter=counter+1)
6  if(counter!=y) jump to step 4
7  print sum
8  stop
```

Note that line numbers have been added for reference purposes.

Flowchart example: Product of x and y