Problem Solving

Deepak Mitra
CDAC

The Need for Data Structures

- Data structures organize data
  - This gives ***more efficient programs.***
- More powerful computers encourage more complex applications.
- More complex applications demand more calculations.
- Complex computing tasks are unlike our everyday experience.

Organization

- Any organization for a collection of records can be searched, processed in any order, or modified.
  - The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.
- A solution is said to be ***efficient*** if it solves the problem within its ***resource constraints***.
  - Space
  - Time
- The ***cost*** of a solution is the amount of resources that the solution consumes.

Selecting A Data Structure

- Select a data structure as follows
  1. Analyze the problem to determine the resource constraints a solution must meet.
  2. Determine basic operations that must be supported. Quantify resource constraints for each operation.
  3. Select the data structure that best meets these requirements.
- Some questions to ask:
  - Are all the data inserted into the structure at the beginning or are insertions interspersed with other operations?
  - Can data be deleted?
  - Are the data processed in some well-defined order, or is random access allowed?

Data Structure Philosophy

- Each data structure has costs and benefits.
- Rarely is one data structure better than another in all situations.
- A data structure requires:
  - space for each data item it stores,
  - time to perform each basic operation,
  - programming effort.
- Each problem has constraints on available time and space.
- Only after a careful analysis of problem characteristics can we know the best data structure for the task.

Example

- Bank account transactions
  - Open an account: a few minutes
  - Withdraw from the account: a few seconds
  - Add to the account: overnight (or more)
  - Close the account : overnight

Definitions

- A ***type*** is a set of values.
  - E.g., boolean, int
  - Can be an infinite set (but not on a computer)
- A ***data type*** is a type and a collection of operations that manipulate the type.
  - E.g., && || + - * >> <<
- A ***data element*** or ***element*** is a piece of information of a record.
- A data item is said to be a ***member*** of a data type.
- A ***simple data item*** contains no subparts.
- An ***aggregate data item*** may contain several pieces of information.

Abstract Data Types

- ***Abstract Data Type (ADT):*** a definition for a data type solely in terms of a set of values and a set of operations on that data type.
- Each ADT operation is defined by its inputs and outputs.
- The process of hiding implementation details is known as ***encapsulation***.
- C++ implements this concept with the class declaration (.h file or prototypes).

Data Structure

- A ***data structure*** is the physical implementation of an ADT.
  - Each operation associated with the ADT is implemented by one or more subroutines (functions) in the implementation.
- ***Data structure*** usually refers to an organization for data in a computer's main memory.
- ***File structure*** is an organization for data on peripheral storage, such as disk or tape.
- C++ implements this concept with the class implementation (public and private functions and data).

Logical vs. Physical Form

- Data items have both a ***logical*** and a ***physical*** form.
- Logical form: definition of the data item within an ADT.
- Physical form: implementation of the data item within a data structure.
- For example, the concept (logical form) of a list could be implemented (physical form) using an array or a linked list.

Computer programming

- Computers carry out a variety of tasks on our behalf
  - Check our spelling
  - Download a web page
  - Play a music file
- In order to carry out these tasks, a computer needs a set of detailed instructions. These instructions are called a **program**.
- The writing of these instructions is called **computer programming**.

Computer programming – three stages

- Computer programming can be divided into three distinct areas of activity:

  1. Problem definition and problem solving
  2. Creating a structured solution (or algorithm)
  3. Coding (e.g. Java, Python, C++)

- This module focuses on the first two areas of programming: problem solving and algorithm development.

Computer Programming – 3 Stages

- Example
  - **Specify the problem**

    Customers paying by cash often do not have the correct money. If they pay more than the price of the item, then they need to be given the correct amount of change.
  - **Solve the problem**
    The amount of change returned to the customer will be the amount paid by the customer, minus the price of the goods.
  - **Specify the solution in a structured (algorithm) format**

```
Get Price ;
Get Amount Paid ;
Calculate Change Due (Amount paid – Price) ;
Return Change to Customer;
```

Computer Programming

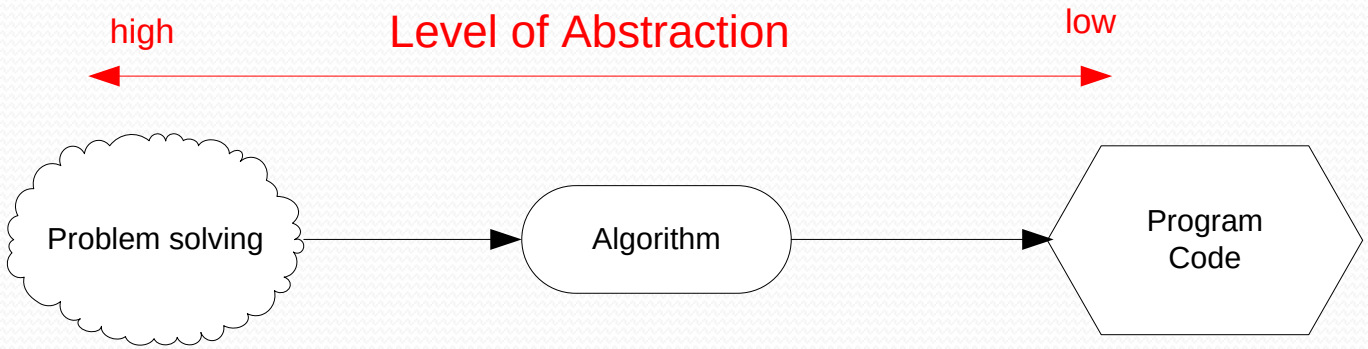
- **Code the program** (java)

```
public class Vendor
{
   double price;
   double paid;

    public Vendor(double pr, double pa) {
       this.price = pr;
       this.paid = pa;
    }

  double getChange(){
      return pa - pr;
    }

    public static void main(String[] args){
    Vendor v = new Vendor(400, 1000);
    double cashToReturn = v.getChange();
    v.returnCash(cashToReturn);
    }
}
```

Problem Solving, Algorithm Development and Coding

- In problem solving, algorithm development and coding we tend to view problems at different levels of **abstraction**.
  - **Problem solving** is done at a **high** level of abstraction. Much of the detail is omitted at this stage in order that the problem can be more easily specified and solved.
  - **Algorithm development** works at a **lower** level of abstraction than problem solving. It contains much of the detail that will eventually become the program. However, it still omits the finer detail required in the final computer program.
  - **Coding** works at a very **low** level of abstraction. Programming code must be written with every single detail a computer needs to carry out a task included.

Problem Solving, Algorithm Development and Coding



The amount of change returned to the customer will be the amount paid by the customer, minus the price of the goods.

```
Get Price ;
Get Amount Paid ;
Calculate Change Due
(Amount paid – Price) ;
Return Change to
Customer ;
```

```
public static void main(String[]
 args){
Vendor v = new Vendor(400, 10000);
double cashToReturn = v.getChange();
v.returnCash(cashToReturn);
}
. . . . Etc
 . . . . Etc
```

Problem Solving, Algorithm Development and Coding

- Problem solving, algorithm development and coding differ not only in terms of level of abstraction at which the problem is viewed, but are also different in terms of the language that is
  used to complete the activity:
  - Problem solving is generally done using use natural, everyday language
  - Algorithm development is done using a specialist, semi-structured language (**pseudo code**)
  - Coding is done using a highly-structured (programming) language that is readable by machines

What is a problem?

- There are many different kinds of problems:
  - Why did I break up with my girlfriend?
  - How much tax should my employees be paying?
  - How do I get to Oxford from London?
  - Why is it wrong to lie?
- However, not all kinds of problems can be solved using computers (Which of the above do you think can and cannot be solved with a computer?).
- For a problem to be soluble through the use of a computer, it must generally:
  - Be technical in nature with **objective** answers that can be arrived at using **rational** methods
  - Be **well-structured**
  - Contain sufficient information for a solution to be found
  - Contain little, if any, **ambiguity**.

How do we Solve Problems?

- We need to THINK!
- We need to engage in one or more of the following types of thinking:
  - **Logical reasoning** (e.g. *Dave and Anne have two daughters. Each daughter has a brother. How many people are there in the family?).*
  - **Mathematical reasoning** (e.g. *A car has a fuel capacity of 10 gallons and a fuel consumption of 45 miles per gallon. How far can the car travel on three quarters of a tank?).*
  - **Lateral thinking** (e.g. *I throw a heavy weight to arrive at a district in North London?).*
- Problem solving is easier if we employ a **problem solving framework** and an appropriate **problem solving strategy to** aid us.

A Problem Solving Framework

- An outline framework for problem solving:
  1. Understand the problem
  2. Devise a plan to solve the problem
  3. Carry out the plan
  4. Assess the result
  5. Describe what has been learned from the process
  6. Document the solution.
- In Vickers, this framework is called the **How to Think Like a Programmer** (HTTLAP) approach to problem solving.
- We will use this approach throughout this module.

(1) Understanding the Problem

- To understand a problem
  - We need to read and reread it till we understand every detail
  - We need to dissect the problem into its component parts (e.g. problems and **sub-problems**)
  - We need to remove any **ambiguity**
  - We need to remove any information that is **extraneous** to the problem
  - We need to determine our **knowns** and our **unknowns**
  - We need to be aware of any **assumptions** we are making.

(1) Understanding the problem

A car has a fuel capacity of 12 gallons and a fuel consumption of 45 miles per gallon. Gas costs 2.56 a gallon. How far can the car travel on 16.64 worth of petrol?).

- In the above example,
  - What are the knowns and the unknowns?
  - Where, if anywhere, is there ambiguity?
  - What are the component parts of the problem?
  - What information is extraneous to the problem solution?
  - What assumptions have you made about the problem?

(2) Devise a plan to the solve the problem

- If a problem contains a set of sub-problems, in what order are you going to solve them?
- How are you going to represent the problem:
  - Numerically?
  - Graphically?
  - Tabular data?
  - Natural language?
- Does the problem lend itself to a particular problem solving strategy or strategies:
  - Working backwards?
  - Solving a simpler analogous problem?
  - Logical reasoning?
  - Finding a pattern?
  - Accounting for all possibilities?

(3) Carry out the plan

- Consider the following problem:

  In a room with ten people, everyone shakes hands with everyone else exactly once. In total, how many handshakes are there?

- How would you represent and solve the problem?
- To solve it quickly and correctly without guessing we could use one of the strategies mentioned earlier. However, which strategy is best suited to solving this problem?
- To answer this, we need to familiarize ourselves with the strategies and practice their use.
- We will look in more details at problem-solving strategies in the next session. In the meantime, a good introduction to problem solving strategies is:

  Crossing the River with Dogs

(3) Carry out the plan

- To answer the handshakes problem we could begin by **accounting for all the possibilities** and graphically represent those possibilities in a table.
- This clearly shows us who shakes hand with who (A to J shake hands with everybody except themselves (X)).
- We can then use some basic **mathematical reasoning** to find our solution:

number of people = 10

grid size = 10 x 10 = 100

handshakes = (100 - 10)/2 = 45

- From here we can abstract a generalisable solution that can easily be turned into an algorithm and eventually into computer code:

$h = ((p^2) – p))/2$

	A	B	C	D	E	F	G	H	I	J
A	X									
B		X								
C			X							
D				X						
E					X					
F						X				
G							X			
H								X		
I									X	
J										X

(4) Assessing the results

- It is very unusual when solving complex problems to achieve the correct result first time round. We often need several attempts to get it right.
- To verify our solutions are correct, we need to take a few steps backwards:
  - Was our understanding of the problem correct?
  - Did we overlook anything?
  - Did we choose the correct strategy?
  - Did we employ that strategy correctly?
  - Have we made any incorrect or unwitting assumptions?
- However, it is often very difficult to spot our own mistakes. It is often better, therefore, to have somebody else verify our solutions for us.

(4) Assessing the results

- Sometimes solutions appear correct, but are in fact wrong, due to an initial misunderstanding of a problem (What Vickers calls, **errors of the third kind**).
- If you have misunderstood a problem, it does not matter how good a coder you are, your program will not work as it is supposed to.
- Therefore getting the problem-solving part of programming right is absolutely essential if we are to build programs that work as they are supposed to work.

(5) Describing what you have learned

- You can only become a good problem solver by reflecting on your experiences of problem solving.
- Keeping a record of problems you have attempted, your success, failures, the approaches you have used, etc. will:
  - Broaden your problem solving repertoire
  - Help you to recognize similarities/patterns in problems
  - Help you identify and fix logical or implementation errors
  - Help you to solve problems faster and more effectively

(6) Documenting the solution

- Documenting a solution will help you to **generalize** your approach to other similar problems.
- Do you recognize the following type of problem? How would you solve it?

  Carlos and his friends have a fantasy football league in which each team will play each other three times. The teams are Medellin, Cali, Antigua, Leon, Juarez, Quito and Lima. How many games will be played in all?

- It will also prevent you forgetting how you arrived at your solutions.
- In the long run, it will make you a better problem solver and eventually a better programmer.

What is a problem?

- A problem is a discrepancy (a "gap") between an actual state and a desired state
  - Implies that the desired state is one that is not currently enjoyed
    - Desired state can be seeking an enjoyable experience
    - Desired state can be seeking to avoid pain
  - Some people argue that all human activity is problem solving directed towards achieving a desired state(s)

Desired states, emotions and logic

- Desired states are based in emotions that may be common to all
  - Core values
    - Security (based on desire to survive)
    - Attraction, companionship, procreation
    - ?????? Other ??????
- Logic is a tool we use to acquire desired states.

How do we solve problems?

- Determine the desired state
  - Decide on what is valued
  - Individual perceptions are different, (which is why teams often fall apart)
  - Determine the criteria to use in deciding whether or not the desired state has (or has not) been achieved.
- Identify a course of action to take that realistically will change the existing state to the desired state
  - Often referred to as a solution
  - Should use decision criteria developed in determining the desired state, (but often we don't)
- Implement
  - Includes dealing with contingencies, (often chaotically)

Formal Steps in Problem Solving

- Define the desired state
  - In terms of decision criteria
- Analyze the existing situation
- Develop alternative solutions
- Evaluate and choose alternative solutions
  - Using decision criteria
- Develop a plan to implement the chosen solution
- Implement, monitor, and revise

What skills are needed?

- Bloom's Taxonomy
  - Knowledge:
    - arrange, define, duplicate, label, list, memorize, name, order, recognize, relate, recall, repeat, reproduce state.
  - Comprehension:
    - classify, describe, discuss, explain, express, identify, indicate, locate, recognize, report, restate, review, select, translate,
  - **Application:**
    - **apply, choose, demonstrate, dramatize, employ, illustrate, interpret, operate, practice, schedule, sketch, solve, use, write.**
  - **Analysis:**
    - **analyze, appraise, calculate, categorize, compare, contrast, criticize, differentiate, discriminate, distinguish, examine, experiment, question, test.**
  - **Synthesis:**
    - **arrange, assemble, collect, compose, construct, create, design, develop, formulate, manage, organize, plan, prepare, propose, set up, write.**
  - **Evaluation:**
    - **appraise, argue, assess, attach, choose compare, defend estimate, judge, predict, rate, core, select, support, value, evaluate.**
- More important skills are more difficult