



Vietnam National University of HCMC
International University
School of Computer Science and Engineering



Overview of Algorithms, Data Structures, and Object-Oriented Programming

Dr Vi Chi Thanh - vcthanh@hcmiu.edu.vn

<https://vichithanh.github.io>



SCAN ME

Week by week topics (*)

1. Overview, DSA, OOP and Java

2. Arrays

3. Sorting

4. Queue, Stack

5. List

6. Recursion

Mid-Term

7. Advanced Sorting

8. Binary Tree

9. Hash Table

10. Graphs

11. Graphs Adv.

Final-Exam

10 LABS

What are they?

- +What is a data structure?
- +What is an algorithm?
- +Not yet to answer but let's see some examples

Examples

- + You have a book-shelf.
 - + How to arrange books into the book-shelf?
 - + Why ?
- + To travel to all famous tourist sites in HCM city.
 - + What route and schedule should you follow?
 - + Why should you do like that?

What are they?

- + A data structure is
 - + an arrangement of data
 - + in a computer's memory (or sometimes on a disk).

What is data structure?

- + In **computer science**, simply speaking a ***data structure is a way of storing data*** in a computer so that ***it can be used efficiently***.
- + A data structure is a way of organizing data that considers ***not only the items stored, but also their relationship*** to each other. Advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data.
- + Often a ***carefully chosen data structure will allow a more efficient algorithm*** to be used. The choice of the data structure often begins from the choice of an abstract data structure. A well-designed data structure allows a variety of critical operations to be performed on using as little resources, both execution time and memory space, as possible.
- + Different kinds of data structures are suited to different kinds of applications, and some are highly specialized to certain tasks. For example, **B-trees** are particularly well-suited for implementation of databases, while **compiler** implementations usually use **hash tables** to look up identifiers.

Why data structure is important in computer science?

- + Data structures are used in almost every program or software system.
- + Data structures provide a means to manage huge amounts of data efficiently, such as large **databases** and **internet indexing services**.
- + Usually, efficient data structures are a key to designing efficient **algorithms**. Some formal design methods and **programming languages** emphasize data structures, rather than algorithms, as the key organizing factor in software design.
- + Niklaus Emil Wirth (born February 15, 1934) is a Swiss computer scientist said:

Algorithms + Data Structures = Programs

Basic types of data structure

- + In **programming**, the term **data structure** refers to a scheme for organizing related pieces of information. The basic types of data structures include:
 - + files, lists, arrays
 - + records, trees, tables
- + Each of these basic structures has many variations and allows different operations to be performed on the **data**.

What are they? Algorithms

- + Are sequences of instructions
 - + To manipulate the data in these data structures
 - + In a variety of ways.
- + Such as
 - + Insert a new data item
 - + Delete a specified item
 - + Iterate through all the items in a data structure
 - + Sorting in-order all items in a data structure

What is algorithm?

- + A **formula or set of steps for solving a particular problem**. To be an algorithm, a set of rules must be **unambiguous** and have a **clear stopping point**. Algorithms can be expressed in any language, from natural languages like English or French to programming languages like FORTRAN.
- + We use algorithms every day. For example, a recipe for baking a cake is an algorithm.
- + Most programs, with the exception of some artificial intelligence applications, consist of algorithms.
- + Inventing elegant algorithms -- algorithms that are simple and require the fewest steps possible -- is one of the principal challenges in programming.

Algorithm example

Recipe CHOCOLATE CAKE

4 oz. chocolate
1 cup butter
2 cups sugar

3 eggs
1 tsp. vanilla
1 cup flour

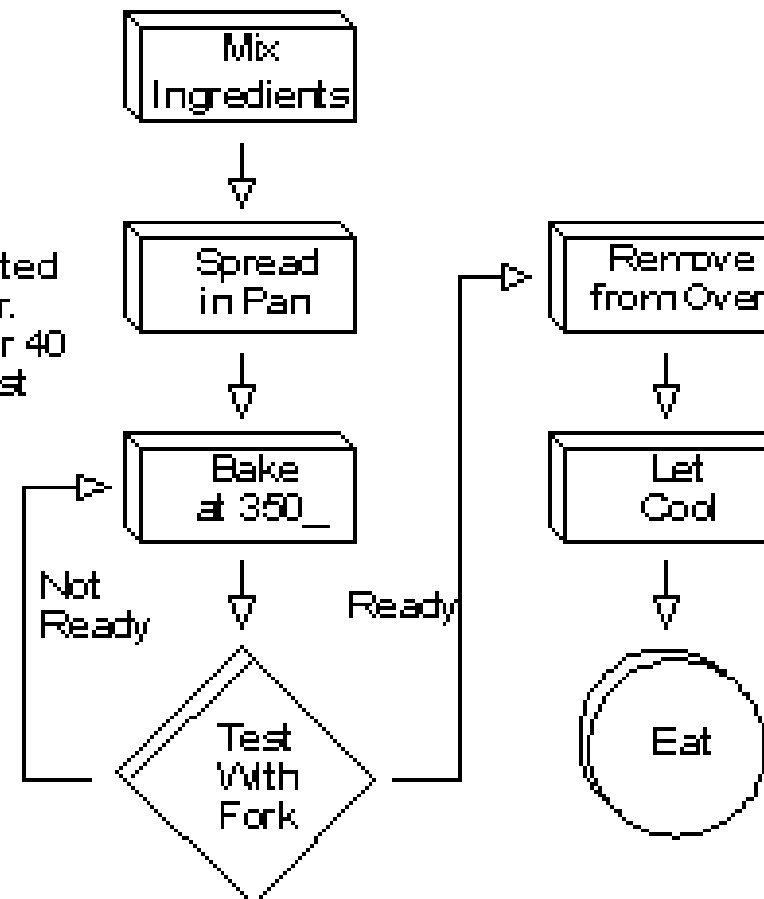
Melt chocolate and butter. Stir sugar into melted chocolate. Stir in eggs and vanilla. Mix in flour. Spread mix in greased pan. Bake at 350_ for 40 minutes or until inserted fork comes out almost clean. Cool in pan before eating.

Program Code

Declare variables:

| | | |
|-----------|---------|-----|
| chocolate | eggs | mix |
| butter | vanilla | |
| sugar | flour | |

```
mix = melted ((4*chocolate) + butter)
mix = stir (mix + (2*sugar))
mix = stir (mix + (3*eggs) + vanilla)
mix = mix + flour
spread (mix)
While not clean (fork)
  bake (mix, 350)
```



What are the properties of an algorithm?

- + **Input** - an algorithm accepts zero or more inputs
- + **Output** - it produces at least one output.
- + **Finiteness** - an algorithm terminates after a finite numbers of steps.
- + **Definiteness** - each step in an algorithm is unambiguous. This means that the action specified by the step cannot be interpreted (explain the meaning of) in multiple ways & can be performed without any confusion.
- + **Effectiveness** - it consists of basic instructions that are realizable. This means that the instructions can be performed by using the given inputs in a finite amount of time.
- + **Generality** - it must work for a general set of inputs.

How to represent algorithms?

- + Use **natural languages**
 - + too verbose
 - + too "context-sensitive"- relies on experience of reader
- + Use **formal programming languages**
 - + too low level
 - + requires us to deal with complicated syntax of programming language
- + **Pseudo-Code** (alias programming language) - natural language constructs modeled to look like statements available in many programming languages.
- + **Flowchart** (diagram) - A **flowchart** is a common type of chart, that represents an algorithm or process, showing the steps as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are used in analysing, designing, documenting or managing a process or program in various fields.

Algorithm representation examples

+ Problem: find the greatest of three numbers

Natural language:

find max(a,b,c)

- Assign $x = a$
- If b great than x then assign $x=b$
- If c great than x then assign $x=c$;
- Result: $x \Rightarrow \max(a,b,c)$

Pseudo-code:

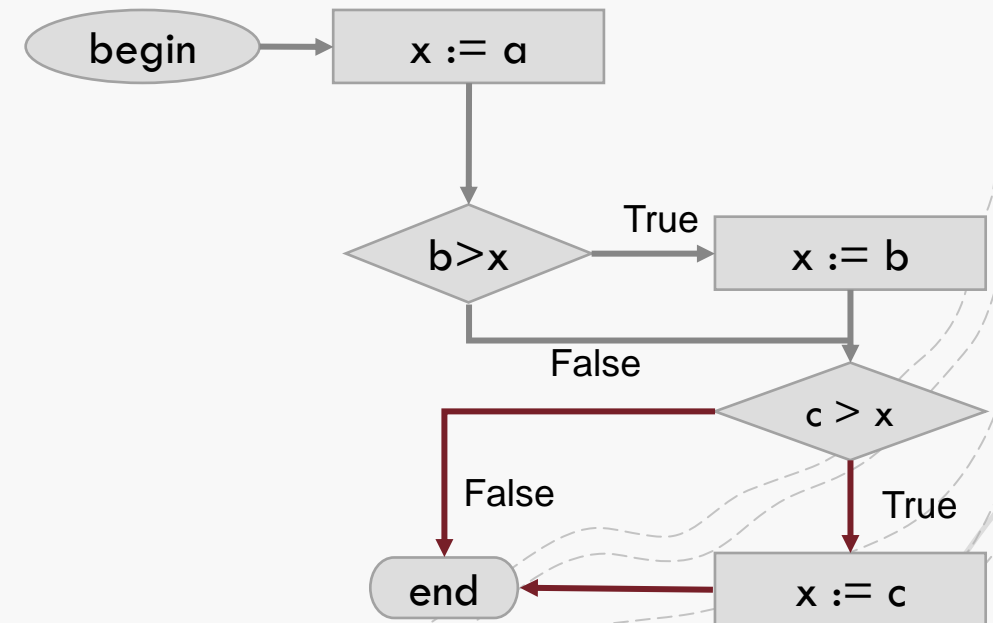
function max(a,b,c)

Input: a, b, c

Output: max(a,b,c)

$x = a$;
if $b > x$ then $x = b$;
if $c > x$ then $x = c$;
return x;

Flowchart:



Data Structures and Algorithms

- + Most algorithms operate on data collections
- + Collection Abstract Data Type (ADT)
 - + Methods
 - + Constructor / Destructor
 - + Add / Edit / Delete
 - + Find
 - + Sort
 - +

Who are you? And why?

- +Familiar with OOP in Java/.NET
- +Basic programming skill in Java/.NET
- +Why you learn this course?
 - +DSA is one of the most fundamental course in CS and IT.
 - +Provide necessary knowledge to learn further: Database, Operating System,....

Program = Data Structure + Algorithms

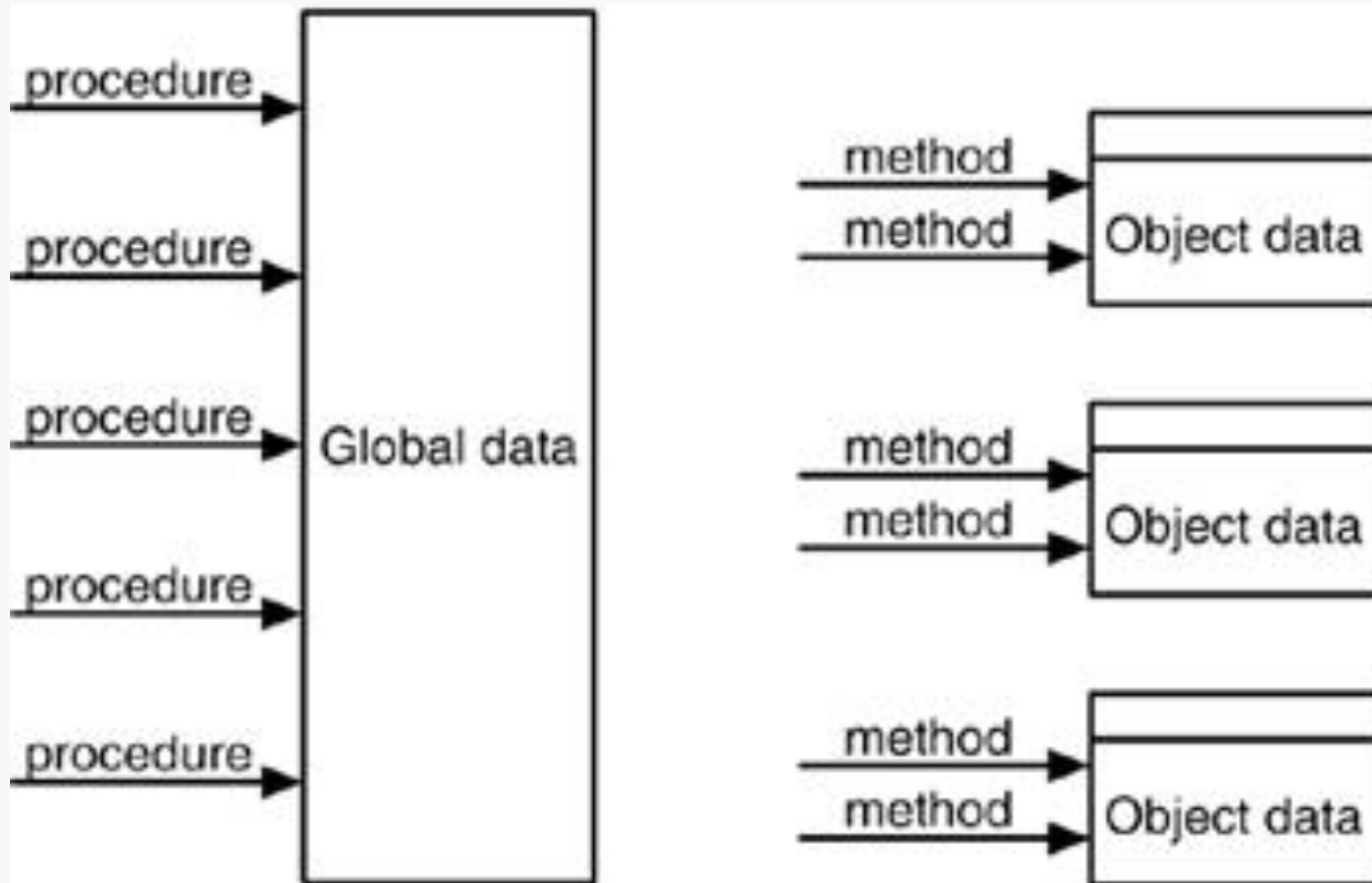
Objectives

- + Understand general concepts of analysing algorithms.
- + Can use basic data structures to solve practical problems.
- + Know to decide which data structures and/or algorithms should be used in practical problems.
- + Implement all of them in Object Oriented Programming (OOP)

Procedural vs. OO programming

+OOP was invented because procedural languages, such as C, Pascal, and early versions of BASIC, were found to be inadequate for large and complex programs.

Procedural vs. OO programming



Procedural vs. OO programming

- + Two problems of Procedural programming:

- + **Prob 1: Poor Modeling of the Real World**

- + A thermostat control program (furnaceON, furnaceOFF, currentTemp, desiredTemp)

- + How about hundreds of thermostats

- + **Prob 2: Crude Organizational Units**

- + Inflexibility of accessing data: there was no way (at least not a flexible way) to specify that some methods could access a variable and others couldn't

Object Class vs. Object

Object Class

- +Template, specification, blueprint for creating objects
- +Don't have states

Object

- +Have is own state (set of objects attributes)
- +Operations

Basic of OOP/ Java Programming

Class

```
class thermostat
{
    private float currentTemp();
    private float desiredTemp();
    public void furnace_on()
    { // method body goes here }
    public void furnace_off()
    { // method body goes here }
} // end class thermostat
```

Object

- +Specifying a class doesn't create any objects of that class
- +To create Object: use the keyword **new**

```
thermostat therm1, therm2; // create two references
```

```
therm1 = new thermostat(); // create two objects and
```

```
therm2 = new thermostat(); // store references to them
```

Accessing Object Methods

- +Use the dot operator (.)

- +Example:

```
therm1.furnace_on();
```

```
therm2.furnace_on();
```

Constructors

- + A special method that's called automatically whenever a new object is created
- + Allows a new object to be initialized in a convenient way
- + Always has exactly the same name as the class
- + Example: `BankApp.java`


```
class BankAccount
{
    private double balance; // account balance
    // constructor
    public BankAccount(double openingBalance)
    { balance = openingBalance; }
    // makes deposit
    public void deposit(double amount)
    { balance = balance + amount; }
    // makes withdrawal
    public void withdraw(double amount)
    { balance = balance - amount; }
    // displays balance
    public void display()
    { System.out.println("balance=" + balance); }
} // end class BankAccount
```

```
class BankApp
{
    public static void main(String[] args)
    {
        BankAccount bal = new BankAccount(100.00); //
        create acct

        System.out.print("Before transactions, ");
        bal.display(); // display balance

        bal.deposit(74.35); // make deposit
        bal.withdraw(20.00); // make withdrawal

        System.out.print("After transactions, ");
        bal.display(); // display balance
    } // end main()
} // end class BankApp
```

JAVA - Access modifiers and access level

| Modifier | Class | Package | Subclass | World |
|-------------|-------|---------|----------|-------|
| public | Y | Y | Y | Y |
| protected | Y | Y | Y | N |
| no modifier | Y | Y | N | N |
| private | Y | N | N | N |

Feature of OOP

+ **Inheritance**

- + Extend existing class


+ **Data Encapsulation**

- + Hide data access from outside

+ **Data Abstraction**

- + Hide implementation from outside

+ **Polymorphism** (Overriding)



Vietnam National University of HCMC
International University
School of Computer Science and Engineering



THANK YOU

Dr Vi Chi Thanh - vcthanh@hcmiu.edu.vn

<https://vichithanh.github.io>



SCAN ME