DATA STRUCTURES AND ALGORITHMS

Introduction:

data, data structures,abstract data types, algorithms, arrays

About lecturer

Head of Information Systems Department
prof. dr. Dalius Mažeika
room No. L413
e-mail Dalius.Mazeika@vgtu.lt
Tel. 27449830
http://dma.vgtu.lt

Lectures

- Theory – every Wednesday at 8:30 am
- Practice – every Wednesday at 10:20 am

Structure of the final grade

- Exam – coefficient 0.6
- Practice – coefficient 0.2 (data structures project) + coefficient 0.2 (sorting algorithm project)
- Additional 0.2 point can granted for every well-done homework.

Literature

- Clifford A. Shaffer. **A Practical Introduction to Data Structures and Algorithm Analysis**. Virginia Tech Blacksburg, VA 24061, 2011.
- T.Cormen, etc. **Introduction to Algorithms**. MIT, 2009.
- W.Collins. **Data structures and standart template library**. McGraw Hill, 2003.
- R.Čiegis. **Duomenų struktūros, algoritmai ir jų analizė**. Vilnius, Technika 2007.
- Baniulis, Tamulynas. **Duomenų struktūros**. Kaunas, Technologija, 2003.
- http://www.academictutorials.com/data-structure/
- http://cprogramminglanguage.net/

Content of the lecture

- Data and relations
- Data Structures
- Algorithms
- Relation between Algorithms and Data Structures
- Abstract data types
- Arrays

Introduction

Representing information is fundamental to computer science. The primary purpose of most computer programs is **not to perform calculations, but to store and retrieve information** — usually as fast as possible.

The study of data structures and the algorithms that manipulate them is at the heart of computer science.

This course will help you to understand how to structure information to support efficient processing.

The need for Data Structures

In the most general sense, a data structure is any data representation and its associated operations.

Even an integer or floating point number stored on the computer can be viewed as a simple data structure.

More typically, a data structure is meant to be an **organization or structuring for a collection of data items**.

A sorted list of integers stored in an array is an example of such a structuring.

Abstract Data Types

A type is a collection of values.

For example, the Boolean type consists of the values **true** and **false**. The integers also form a type. An integer is a simple type because its values contain no subparts. A bank account record will typically contain several pieces of information such as name, address, account number, and account balance. Such a record is an example of an **aggregate type or composite type**.

A data type is a type together with a collection of operations to manipulate the type.

For example, an **integer variable** is a member of the integer data type. **Addition** is an example of an operation on the integer data type.

Abstract Data Types

An abstract data type (ADT) is the realization of a data type as a software component.

The interface of the ADT is defined in terms of a **type** and a **set of operations** on that type. The behavior of each operation is determined by its inputs and outputs.

An ADT does not specify how the data type is implemented. These implementation details are hidden from the user of the ADT and protected from outside access, a concept referred to as encapsulation.

Data structures

A data structure is the implementation for an ADT.

In an object-oriented language such as C++, an ADT and its implementation together make up a class.

Each operation associated with the ADT is implemented by a member function or method. The variables that define the space required by a data item are referred to as data members.
An object is an instance of a class, that is, something that is created and takes up storage during the execution of a computer program.

Data structures

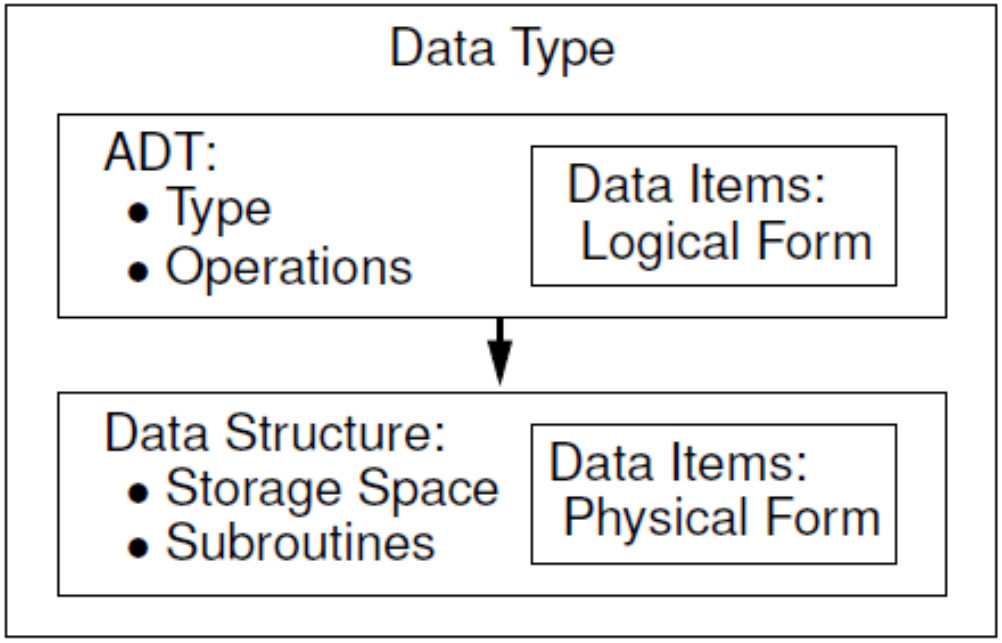
The term “data structure” often refers to data stored in a computer’s main memory.

Example:

- class -> object
- Data types C/C++
  (int, short int, float, double, long double, char, bool).

The related term **file structure** often refers to the organization of data on peripheral storage, such as a disk drive or CD-ROM.

Design patterns



The relationship between data items, abstract data types, and data structures. The ADT defines the logical form of the data type. The data structure implements the physical form of the data type.

Problems, Algorithms, and Programs

Programmers commonly deal with **problems, algorithms**, and **computer programs**.

- A **problem is a function** or a mapping of inputs to outputs. (problem is a task to be performed). Problem definition should not include any constraints on **how** the problem is to be solved.
- An **algorithm is a recipe for solving a problem** whose steps are concrete and unambiguous. The algorithm must be correct, of finite length, and must terminate for all inputs.
- A **program is an instantiation of an algorithm** in a computer programming language.

Algorithm

An algorithm is a method or a process followed to solve a problem.

If the problem is viewed as a function, then an algorithm is an implementation for the function that transforms an input to the corresponding output.

A problem can be solved by many different algorithms. A given algorithm solves only one problem (i.e., computes a particular function).

Algorithm

Algorithm can be represented as:

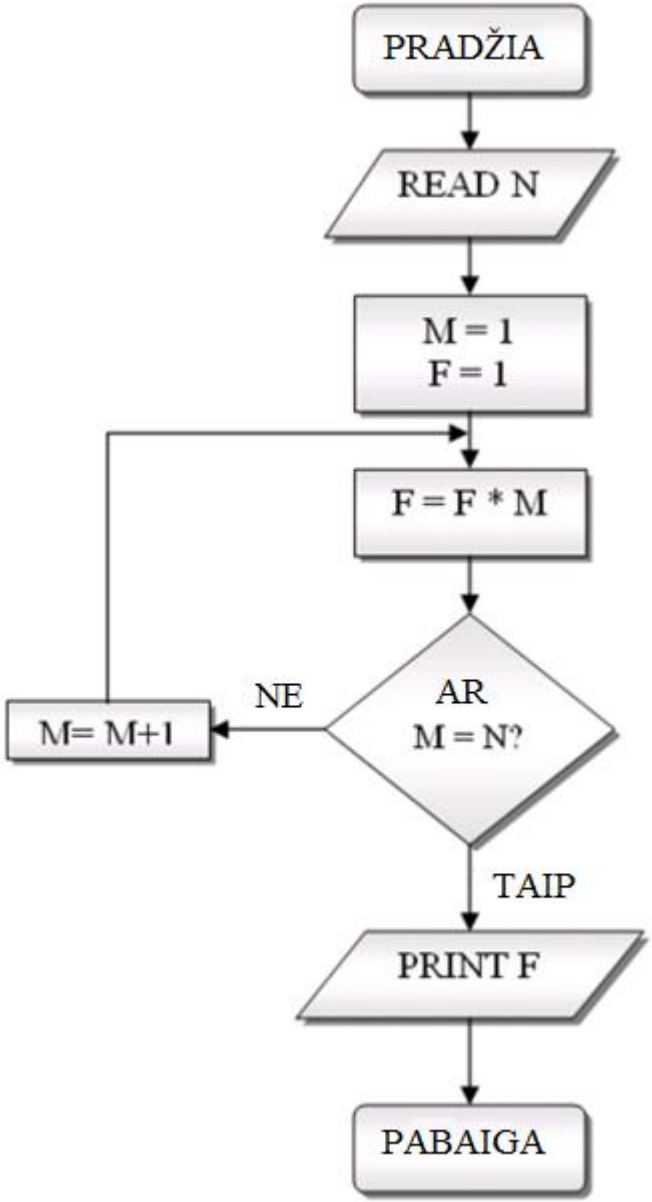
- Flowchart
- Abstract language
- Human language

Algorithm is coded using one of the computer programming language.

Algorithm

Example of the **Factorial** calculation.

```
begin int Factorial (N)
   F = 1;
  for ( M = 1; M != N ; M++ )
            F = F * M;
  return ( F );
end Factorial
```



Properties of the Algorithm

Algorithm must have the following properties:

- It must be correct
- There can be no ambiguity as to which step will be performed next
- It must be composed of a finite number of steps
- It must terminate for all inputs (does not fall into an infinite loop).

Array

Array is a data structure that has the following properties:

- All elements of the array has the same data type;
- Number of elements is fixed and can’t be changed;
- Each element has index number;
- Each element can be accessed by referring it index number.

Arrays has dimension and way how elements are stored depends on array dimension.

Arrays can have one dimension, two dimensions and etc.

Array

Dimension of the array is used to understand the way how elements are virtually stored in the memory.

Elements of the array are stored in sequencial way in the physical memory.

Dimension of the array help us to understand array as multidimensional structure.

Two-dimensional array can be defined as one-dimensional array, but elements of that array are one-dimentional arrays.

Array

Elements of the array are defined using indexes:

- One-dimensional A[0]; A[1]; A[2]
- Two-dimensional B[0] [0]; B[0][1]; B[1][2];
- Three-dimensional C[0] [0] [0]; C[0][0][1]; C[1][2][1];

Indexes are integers. Lets assume that indexes are within interval [L;U]. Then number of the elements are calculated using the following formula:

U - L + 1

Indexes

Lets assume:

- Size of the element is **S** bytes**;**
- Address of the first byte of the first element is **B.**

Then:

Address of the element ***i*** of the one-dimentional array can be calculated as follows:

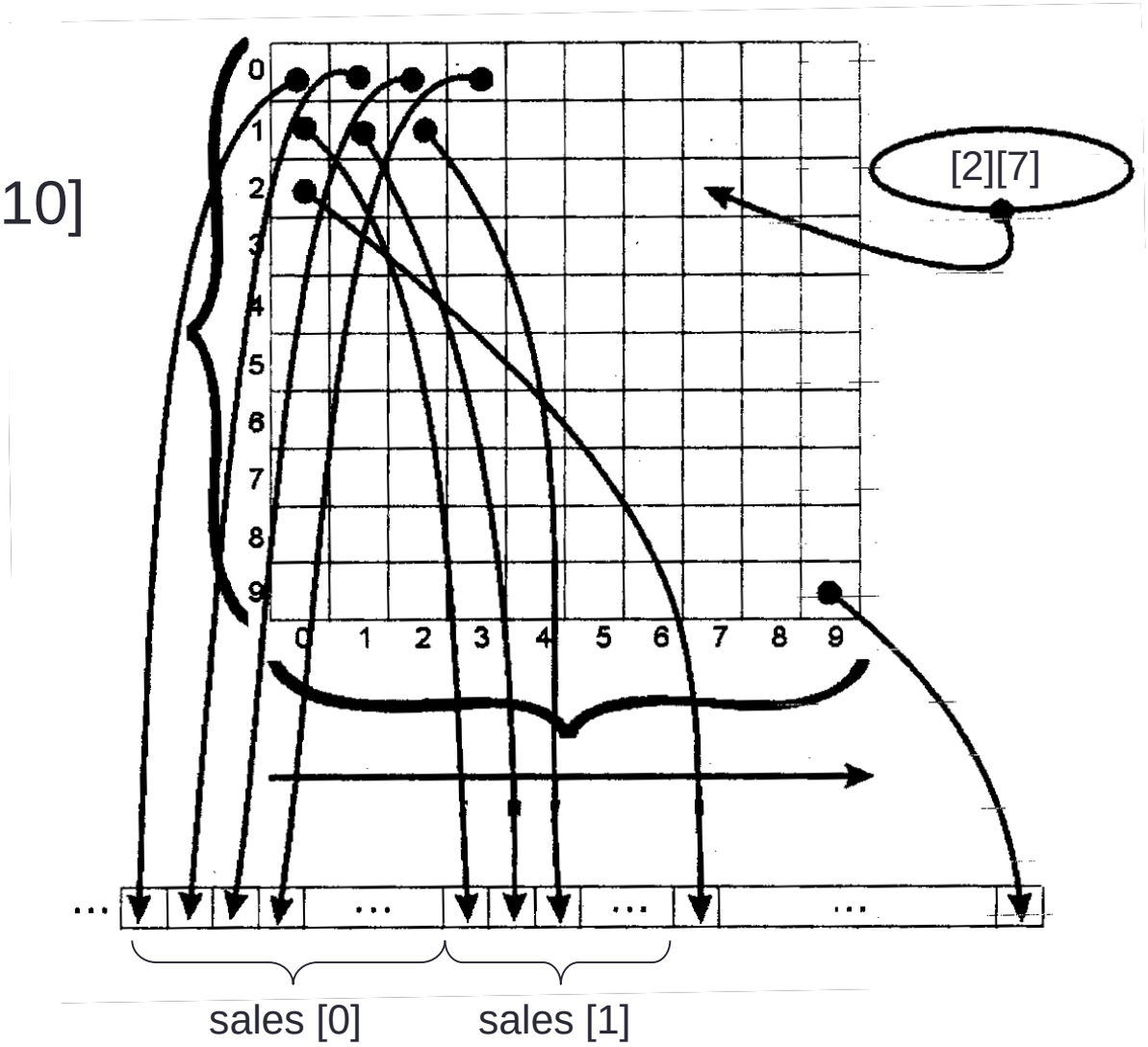
$$A[i] = B + (i - L)*S$$

Address of the element **A[i] [j]** of the two-dimentional array can be calculated as follows:

$$A[i]\,[j] = B + (i - L_1)(U_2 - L_2 + 1)*S + (j - L_2)*S$$

Two dimentional array

Structure of the array: sales[10][10]



Adding and deleting elements

New element is added to the end of array. Index number of a newly added element is equal to **i+1**, where i is the index number of the last element (before adding procedure).

Important. Array must be not full if you want to add new element.

Any element can be deleted from the array. If deleted element isn't the last element, then all elements located at the right side must be shifted to the left side by one position.

Practice

- Microsoft Visual C++ Exress
- First programme “Hello world”
- Solving quadratic equation
- How to find even and odd numbers
- Primary numbers