Topic 2:
Lecture 1a
Intro to Data Structures and Algorithms: The Bag

CSCI 240

Data Structures and Algorithms

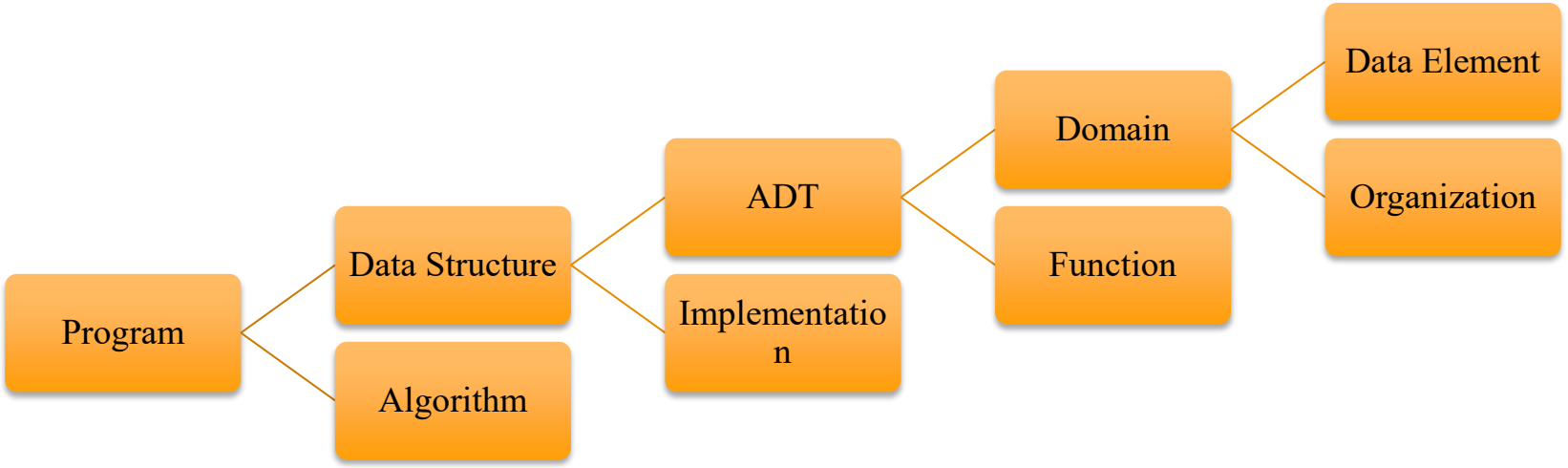
Prof. Dominick Atanasio

Topics

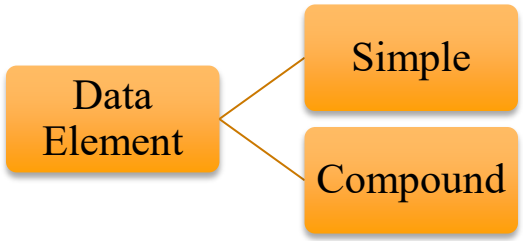
- Introduction
  - Overview
  - Data Structures and Algorithms

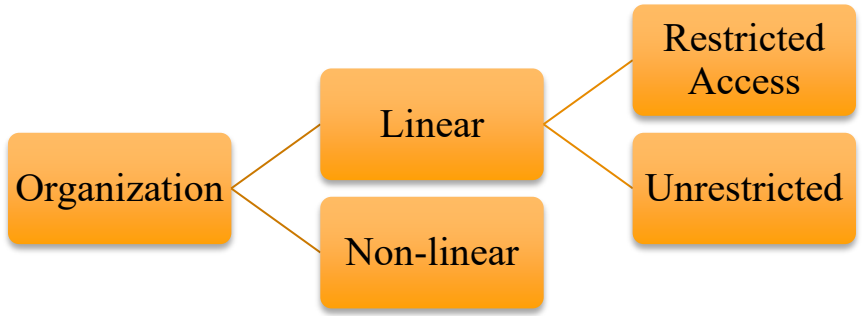
- Phases of software development

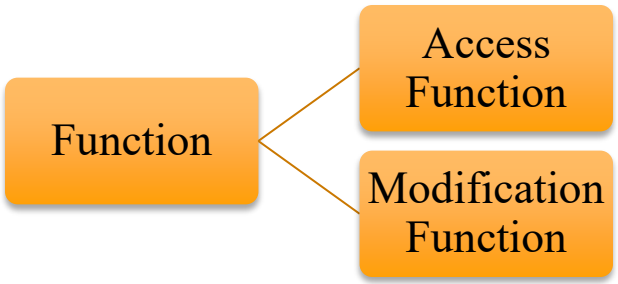
Overview


Program
Data Structure
Algorithm
ADT
Implementation
Domain
Function
Data Element
Organization

Overview


Data Element
Simple
Compound


Organization
Linear
Non-linear
Restricted Access
Unrestricted


Function
Access Function
Modification Function

Data Structures

- Data Structures
  - Why?
    - To write efficient programs
    - Need to organize data to effectively access and manipulate
  - What?
    - A collection of data, organized so that items can be stored and retrieved by some techniques
  - Main types
    - Linear – array, list, stack, queue. Each of them is a collection that stores its entries in a linear sequence, and in which entries may be removed or added at will
    - Non-linear – Trees and graphs. Data entries are not arranged in sequence, but with different rules.

Data Structures

- An Abstract Data Type (ADT) is a model of a data structure that specifies:
  - The characteristics of the collection of data
  - The operations that can be performed on the collection
- It’s abstract because it doesn’t specify how the ADT will be implemented.
- A given ADT can have multiple implementations.

Data Structures

- To implement an ADT, you need to choose:
  - a data representation
    - must be able to represent all possible values of the ADT
    - should be private
  - an algorithm for each of the possible operations
    - must be consistent with the chosen representation
    - all auxiliary (helper) operations that are not in the contract should be private

Data Structures

- Algorithms
  - An algorithm is a procedure, a finite set of well-defined instructions, for solving a problem which, given an initial state, will terminate in a defined end-state.
  - Different ways to express an algorithm:
    - Natural language (NOT used)
    - Pseudo code
    - Flowcharts
    - Programming languages (intended for expressing algorithms in a form that can be executed by a computer)

- Structured ways to express algorithms
- Independent of a particular implementation language

Phases of Software Development

- The development process involves the following steps:
  - Specification of the task
  - Design of a solution
  - Implementation (coding) of the solution
  - Analysis of the solution
  - Testing and debugging
  - Maintenance

Review: What is an Object?

- An object groups together:
  - One or more data values(the object's data members–also known as instance variables)
  - A set of operations that the object can perform (the object's member functions)
- In C++, we use a class to define a new type of object.
  - Serves as a "blueprint" for objects of that type

Review: What is an Object?

Simple example:

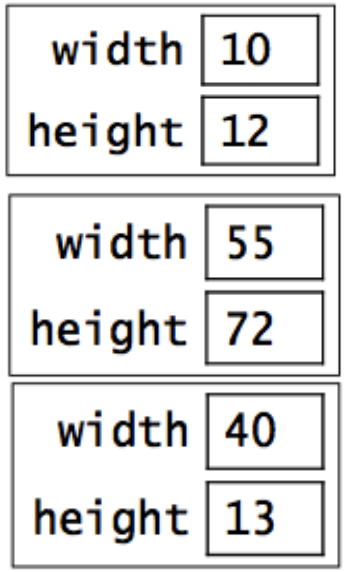
```
class Rectangle
{
  private:
       int width;
       int height;
  public:
       Rectangle(int,int);
       int area();
       ...
};
```

Class vs. Object

- Objects of Class Rectangle

The `Rectangle` class is a blueprint

```
class Rectangle
{
private:
     int width;
     int height;
public:
     Rectangle(int,int);
     int area();
     ...
};
```



Creating and Using an Object

- We create an object two ways, a standard declaration (stack-bound), or by using the new operator :
  - Rectangle rect1{10,30};
  - or Rectangle *rect1 = new Rectangle(10, 30);
- initialization arguments, {10,30} are passed into the object's constructor function.
- Once an object is created, we can call one of its functions by using dot or arrow (for pointers) notation:
  - int area1 = rect1.area(); or int area1 = rect1->area();
- The object on which the function is invoked is known as the called object or the current object.

Two Types of Member Functions

- Member functions that belong to an object are referred to as instance member functions or non-static member functions.
  - They are invoked on an object
  - They have access to the data members of the called object
- Static member functions do not belong to an object – they belong to the class as a whole.
  - They have the keyword static in their header:
  - static int max(int, int);
  - They do not have access to the data members of the class.
  - Outside the class, they are invoked using the classname:
  - int result = Math::max(5, 10);

A Simple ADT: A Bag

- A bag is just a container for a group of data items.
  - analogy: a bag of candy
- The positions of the data items don't matter (unlike a list).
  - {3, 2, 10, 6} is equivalent to {2, 3, 6, 10}
- The items do not need to be unique (unlike a set
  - {7,2,10,7,5} isn't a set, but it is a bag

A Simple ADT: a bag (cont.)

- The operations supported by our Bag ADT:
  - add(item): add item to the Bag
  - remove(item): remove one occurrence of item (if any) from the Bag
  - contains(item): check if item is in the Bag
  - size(): get the number of items in the Bag
  - grab(): get an item at random, without removing it
    - reflects the fact that the items don’t have a position (and thus we can’t say "get the 5th item in the Bag")
- Note that we don’t specify how the bag will be implemented.

Specifying an ADT Using an Interface

- In C++, we can use an interface to specify an ADT, that is, a header file:

```
class Bag
{
  public:
      bool add(int item);
      bool remove(int item);
      bool contains(int item);
      int size();
      int grab();
};
```

- A header file specifies the declarations of the class.
  - Includes only the data members and prototypes (member function headers)
  - If using templates, it includes definition as well

Example: Implement the Bag ADT in C++

In Class Exercise: implement:

- ArrayBag: implemented using an array (auto resizing)
- LinkedBag: implemented using a linked list

Encapsulation

- Our implementation provides proper encapsulation.
  - A key principle of object-oriented programming
  - Also known as information hiding or protection
- We prevent direct access to the internals of an object by making its data members private.
- We provide limited indirect access through public member functions
- In the case of the LinkedBag, we use a Node object which is encapsulated by the owner object so the data members of the Node class do not have to be encapsulated by the node class itself.