Prerequisites

- Algorithm complexity: **Big Oh** notation
- Java/c++

Procedural-oriented programming

- Top-down approach
- Procedure (or Function) is building block

Ex: Calculator → scientific

→ non-scientific → float

→ int → add.

→ subs.

→ mult.

→ div.

} Small functions

Ex: C - language

Procedural-oriented programming

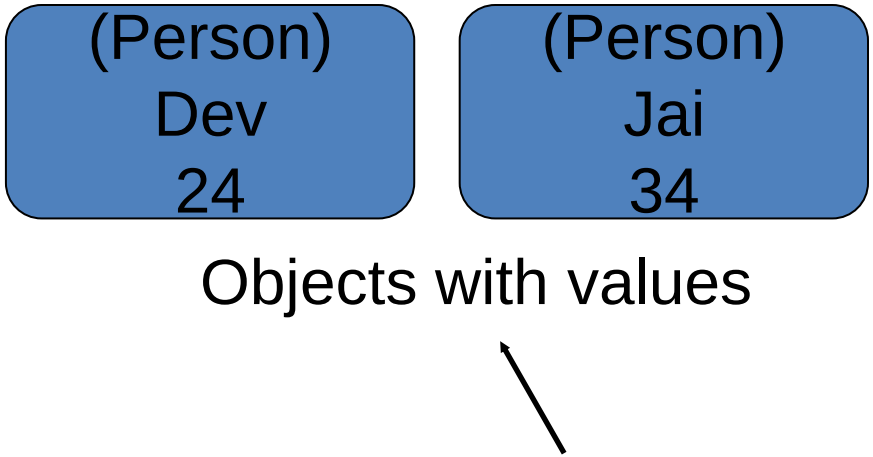
- **Adv:**
  - Re-usability
  - Easy to debug
- **Disadv:**
  - Concentrate on what we want to do, not on who will use it
  - Data does not have a owner (sharing)
  - All functions are global
  - No data security
    - Ex: let there are three functions a(),b() and c(). Data d is used by a() and b() but how to restrict from c() [data will be either global or local]
  - No data integrity [Stack:{push,pop},Queue:{enque, deque}]
    - How to distinguish which fun associated with which data structure

Object-oriented programming

Keep large software projects manageable by human programmers

- Bottom-up approach (user oriented)
- Objects as building block
  - Each contains both methods and variables

An example



- Everything in *OOP* is grouped as self sustainable "*objects*".
- let's take your "*hand*" as another example (class)
  - Your body has two objects of type hand, named left hand and right hand. Their main functions are controlled/ managed by a set of electrical signals sent through your shoulders
  - So the shoulder is an interface which your body uses to interact with your hands
  - The hand class is being re-used to create the left hand and the right hand by slightly changing the properties of it

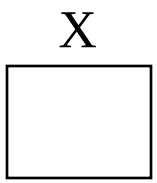
OOP languages

- ***Both Java and C++ are most popular object-oriented programming languages***
- ***C++ was created at AT&T Bell Labs in 1979***
- ***Java was born in Sun Microsystems in 1990***

Variables for built-in types

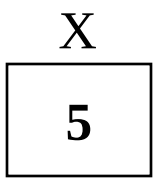
In both C++ and Java

int x;



…

x = 5;



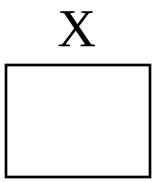
Reference variables (in Java)

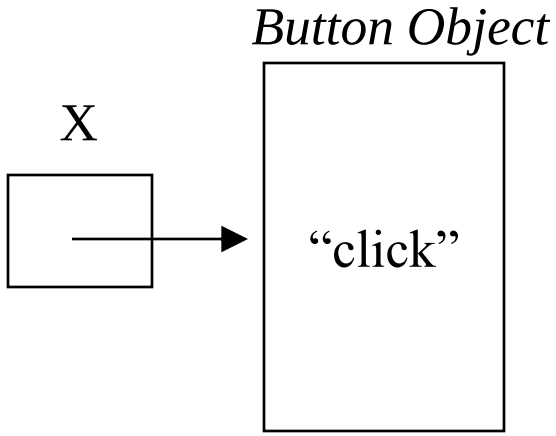
- Reference type variables

Button x;

…

x = new Button(“click”);

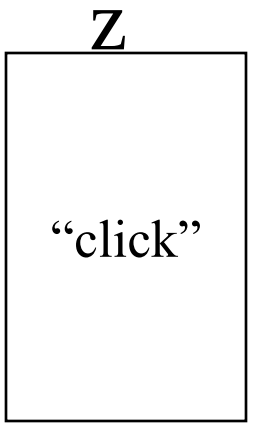




Variables that “hold” objects (in C++)

- Declaration of an object variable allocates space for the object

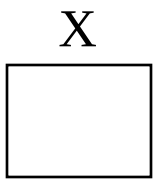
Button z(“Click”);

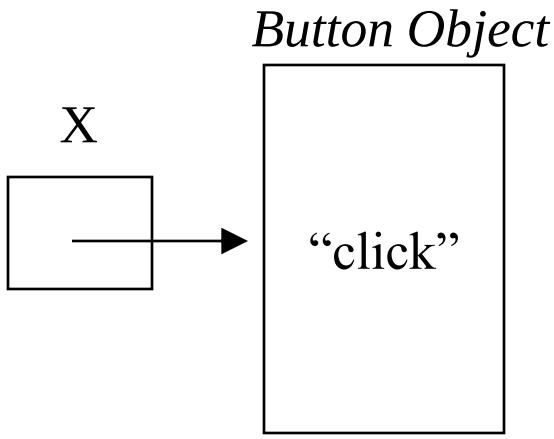


Pointers (in C++)

- Variables can be explicitly declared as pointers to objects

```
Button *x;
…
x = new Button(“click”);
```





Disposing of allocated memory

- In Java, garbage collection is automatic
  - Memory allocated objects are reclaimed when no variables refer to them
  - Need to set reference variables to null when the object is no longer needed
- In C++, object destruction is the programmers responsibility using the **delete** keyword

C++ control over copy and assignment

- In C++, the semantics of “a = b” (assignment) can be specified
  - by defining the copy-assignment operator
  - both (a and b) are diff. objects but same content

  - In java it copies the address

Java basics

- Treat primitive data types and objects differently
  - Int i; // i actually holds the value
  - Account ac; // reference to a account obj

Summary

- ***No pointer***
- ***No multiple inheritance***
- ***Automatic garbage collection***
- ***No operator overloading***
- ***No goto statement and no structure and union data structure***

- ***Pointer***
- ***Multiple inheritance***
- ***Manual garbage collection***
- ***Operator overloading***
- ***Goto statement and structure and union data structure***

Data structures and algorithms

Algorithm, program and data

- **Algorithm**: a sequence of instructions to perform a job

- **Program**: an algorithm written in a programming language
  - manipulates data- stored in main memory
  - needs data representation (for efficiency)

Data type

- **Data type** of a variable in a programming language refers to the set of values that the variable can take.

- Example 1:

  ```
  int x;
  ```

  x takes on values from –INT_MIN to INT_MAX

- Example 2:

  ```
  typedef double height;
  height h;
  ```

  h takes on values from -DBL_MIN to DBL_MAX

Simple vs aggregate data types

- **Simple data type** is scalar data type that *takes values from only one set*
- is either
  - built-in (supplied by the programming language)
    - Examples: int, double

    OR
  - user-defined simple data type
    - Example: height

Simple vs aggregate data types

- **Aggregate data type** is
  - a sequence of one kind of simple data type or
  - n-tuple of many kinds of simple or aggregate data types, or
  - a sequence of aggregate data type.
- Example:

sequence or array: int a[200];

n-tuple: struct point { double x;
double y;
int a[10]; };

sequence or array: struct point p[100];

Data structures

- A **data structure** is data type

  that is

  either simple or aggregate,

  built-in or user-defined, or

  an organized collection of these

- a particular way of organizing data in a computer memory so that it can be accessed efficiently

Data type

- A data type is characterized by:
  - a set of *values*
  - a *data representation,* which is common to all these values, and
  - a set of *operations,* which can be applied uniformly to all these values

Primitive data types in Java

- Java provides eight primitive types:
  - boolean(1), char(16)
  - byte(8), short(16), int(32), long(64)
  - float(32), double(64)
- Each primitive type has
  - a set of values (int: -2^{31}… $2^{31}-1$)
  - a data representation (32-bit)
  - a set of operations (+, -, *, /, etc.)
- These are "**fixed**"—the programmer cannot change anything

Abstract data types

- In OOP,

  data and

  the operations that manipulate that data

  are grouped together in classes

Abstract data types

- An **abstract data type** (ADT) consists of a data structure, together with operations (functions or methods) for accessing and modifying the data structure.
- The individual elements of the data structure of the ADT are called **member data**, and the operations of the ADT are called **member functions**.

ADT and information hiding

- The ADT implements **information hiding**:

  although the programmer (or-creator) of the ADT knows all the details about the member data and member functions,

  the user of the ADT need not know the details of the member data and must use the member functions supplied by the creator to access and modify the data structure.

ADT = Abstract + Data Type

- To Abstract is to leave out information, keeping (hopefully) the more important parts

  What part of a Data Type does an ADT leave out?

- An Abstract Data Type (ADT) is:
  - a set of *values*
  - a set of *operations*, which can be applied uniformly to all these values

It is NOT characterized by its data representation.

- Data representation is private, and changeable, with no effect on application code.

Example ADT

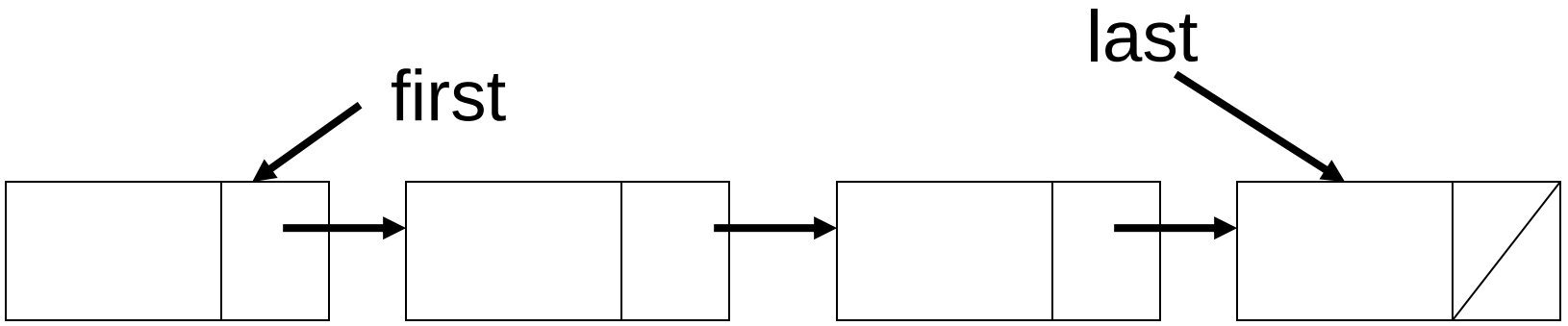
- Simplest ADT is a Bag
  - no implied order to the items
  - duplicates allowed
  - items can be added, removed, accessed
- Another ADT is a Set
  - same as a bag, except duplicate elements not allowed
  - union, intersection, difference, subset

Lists ADT

- Items have a position in this Collection
- Operations: add, delete

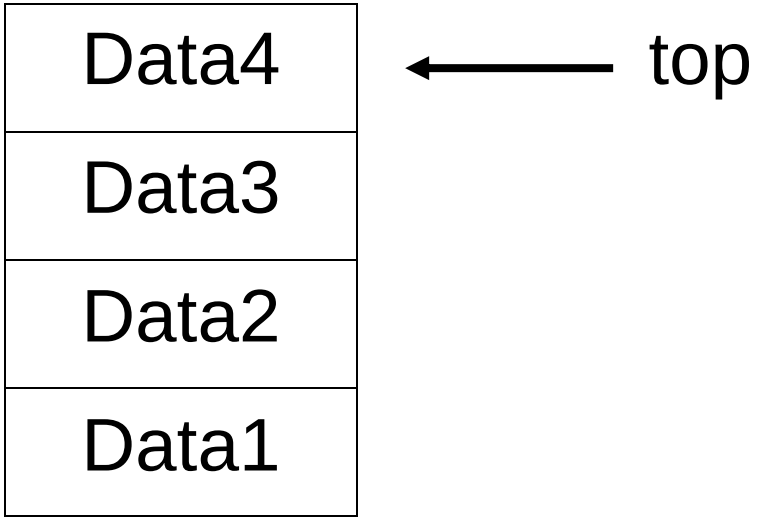
Implementation of List ADT

- Array List
  - internal storage container is native array
- Linked List



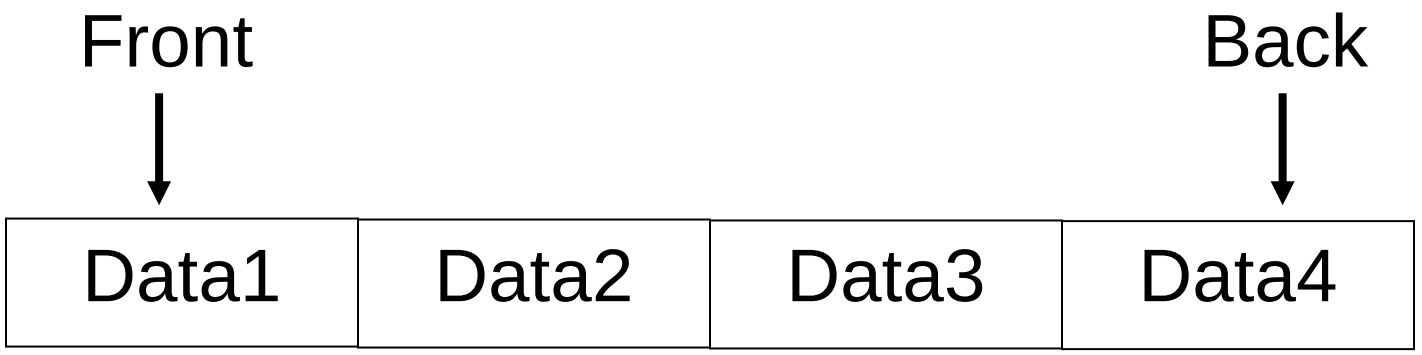
Stacks

- Collection with access only to the last element inserted
- Last in first out
- Operations
  - Push (insert)
  - Pop (remove)
  - top
  - make empty



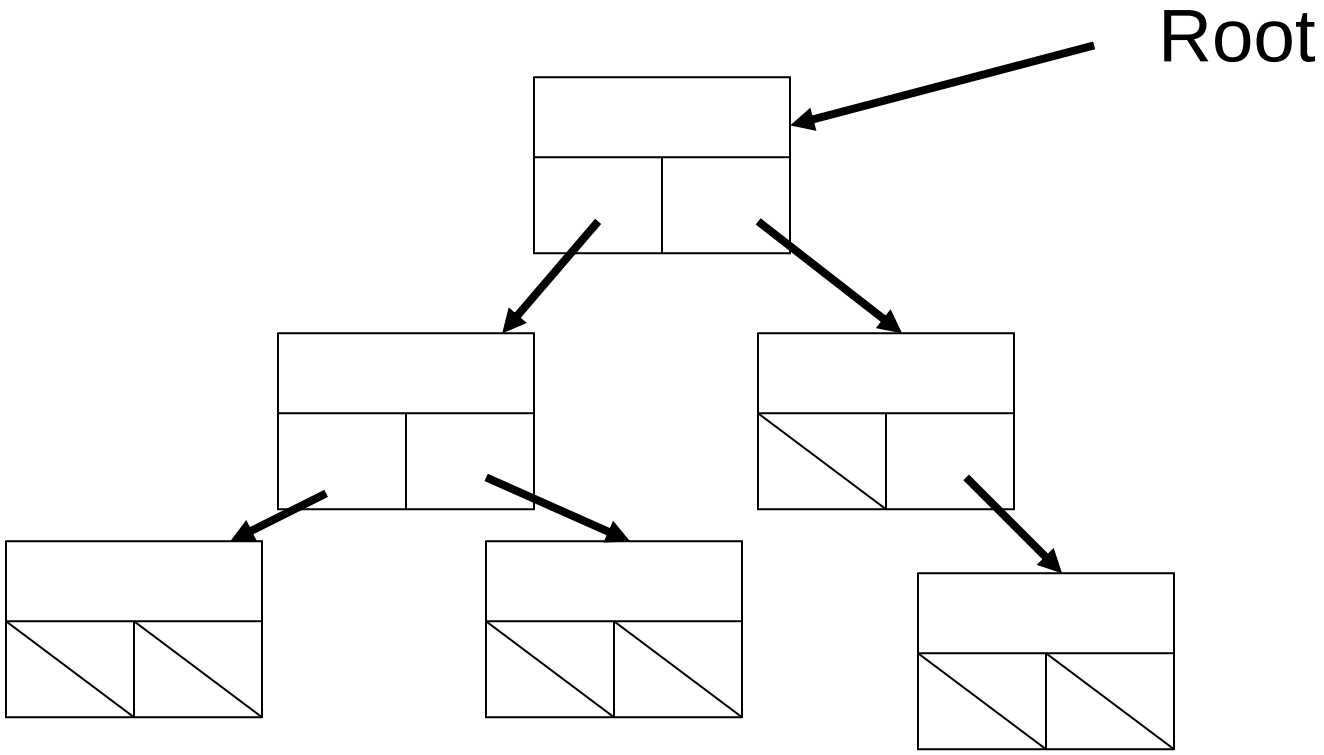
Queues

- Collection with access only to the item that has been present the longest
- Last in last out or first in first out
- Operations: enqueue, dequeue, front



Trees

- Similar to a linked list



Other Types of Trees

- Binary Search Trees
  - sorted values
- Heaps
  - sorted via a different algorithm
- AVL and Red-Black Trees
  - binary search trees that stay balanced
- B Trees

Some more

- Hash Tables
- Maps
  - a.k.a. Dictionary
  - Collection of items with a key and associated values
  - similar to hash tables, and hash tables often used to implement Maps
- Graphs
  - Nodes with unlimited connections between other nodes
- Sparse vectors and sparse matrices

How to pick a ADT

- Many, many different ADTs
  - picking the right one for the job is an important step in design
- High level languages often provide built in ADTs,
  - the Java standard library

Implementation of ADTs

- **Class**es and **struct**s:

  ADTs are implemented

  in Java using classes, and

  in C++ using classes and structures

  in C using structures