Bachelor Thesis Exam

Topics and their theory

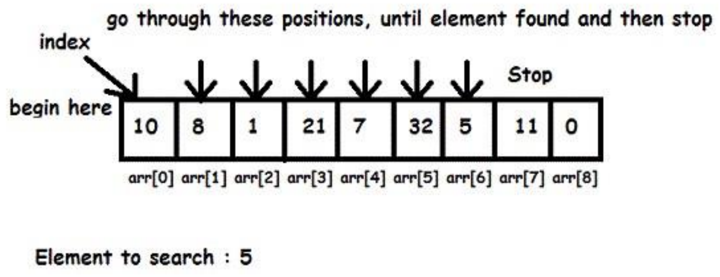
Part 1. Algorithms and Programming

- **Courses: Programming Fundamentals, Object Oriented Programming, Data Structures and Algorithms**

Search

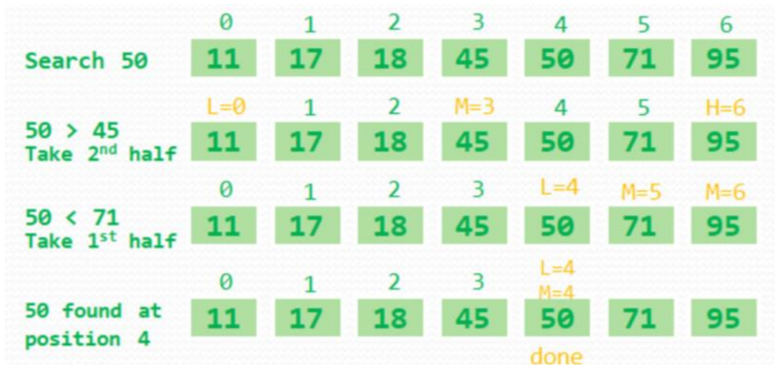
1. Sequential Search

- it is also called a linear search
- the keys are successively examined, starting from the first element until the last until a match is found
- maximum number of searches required: n+1
- average number of searches required: (n+1)/2



2. Binary Search

- more efficient than sequential search
- uses the "divide and conquer" technique
- begin with the element in the middle of the array as a search key, if the value of the search key is equal to the item it returns the index of the search key, otherwise if the value of the search key is less than the item in the middle of the interval, narrow the interval to the lower half (or left half), otherwise, narrow it to the upper half (or right half), check repeatedly from the second point until the value is found or the interval is empty



Merging

- we have 2 data sequences, sorted in increasing (or decreasing) order by the values of the keys
- in order to merge, a new sequence sorted in the same manner needs to be build from the 2 previous data sequences
- there are 2 types of merging:
    - with preserving the duplicates
    - without the duplicates
- a possible solution can be to simply create a third sequence by concatenating the first 2, and then sort the resulted sequence. Thus it is correct, it is inefficient
- it is important to create the third resulted sequence by having a single traversal f the starting 2 sequences

Sorting

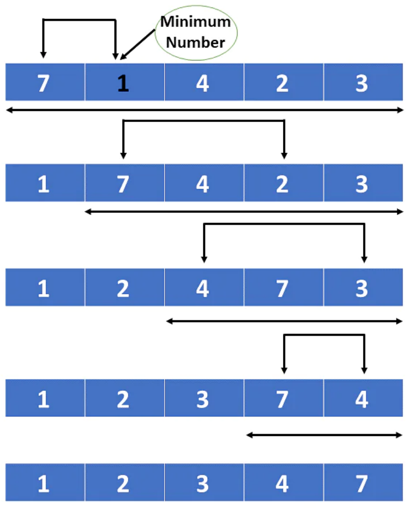
- it is the operation to reorganize the elements in a collection already available in the internal memory, in such a way that the record keys are sorted in increasing (or decreasing) order

r

1. Selection Sort

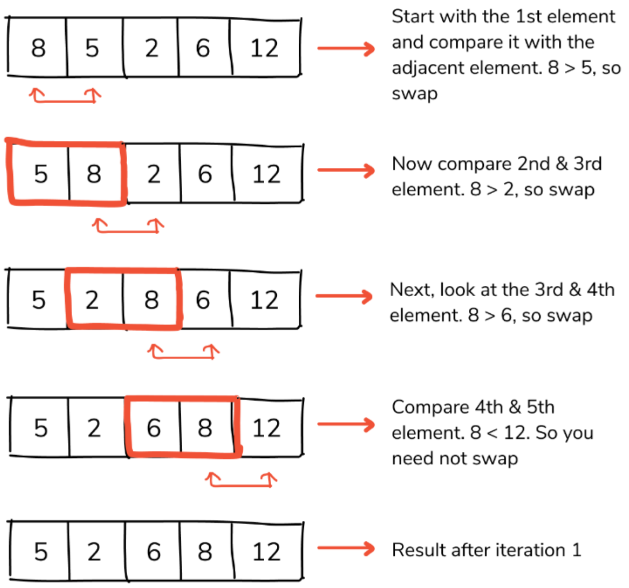
- it works by determining the element having the minimal (or maximal) key and swapping it with the first element, the procedure is resumed for the remaining elements until all elements have been considered

- the total number of comparisons is: **(n-1)+(n-2)+...+2+1=n(n-1)/2**



2. Bubble Sort

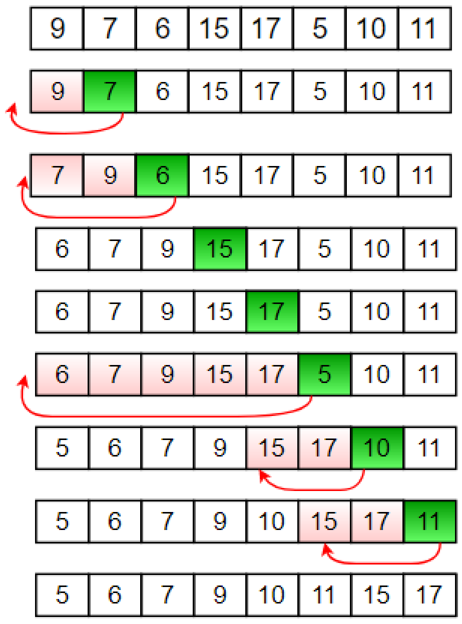
- it compares two consecutive element, and if they are not in the expected relationship, they will be swapped
- the process ends when all pairs o consecutive elements are in the expected relationship


8 5 2 6 12
Start with the 1st element and compare it with the adjacent element. 8 > 5, so swap
5 8 2 6 12
Now compare 2nd & 3rd element. 8 > 2, so swap
5 2 8 6 12
Next, look at the 3rd & 4th element. 8 > 6, so swap
5 2 6 8 12
Compare 4th & 5th element. 8 < 12. So you need not swap
5 2 6 8 12
Result after iteration 1

3. Insertion Sort

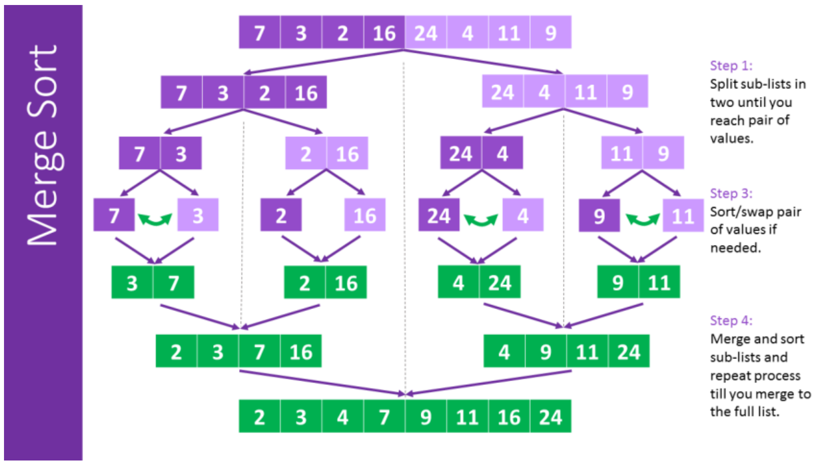
- the idea of this sorting method is that while traversing the elements, we insert the current element at the right position in the subsequence of already sorted elements

- in this way the subsequence containing the already processed elements is kept sorted and at the end of the traversal, the whole sequence will be sorted



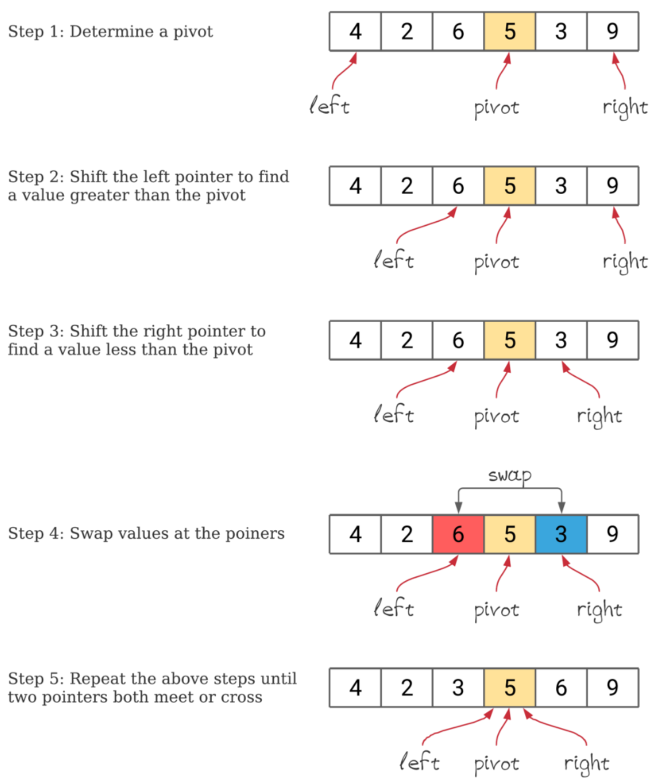
4. Merge Sort

- it is the algorithm for sorting a sequence S of n elements based on a “divide-and-conquer” method
  - if S has at least two elements, let S1 and S2 be the subsequences from S each containing about a half of the elements of S.
  - sort sequences S1 and S2 using merge sort
  - put back the elements into S by merging the sorted sequences S1 and S2 into one sorted sequence


Merge Sort
7 3 2 16 24 4 11 9
7 3 2 16
24 4 11 9
7 3
2 16
24 4
11 9
7 3
2 16
24 4
9 11
3 7
2 16
4 24
9 11
2 3 7 16
4 9 11 24
2 3 4 7 9 11 16 24
Step 1:
Split sub-lists in two until you reach pair of values.
Step 3:
Sort/swap pair of values if needed.
Step 4:
Merge and sort sub-lists and repeat process till you merge to the full list.

5. Quick Sort

- it is a more efficient sorting method
- it is based on the “divide and conquer” technique
- it picks an element as pivot and partitions the given array around the picked pivot



Step 1: Determine a pivot
Step 2: Shift the left pointer to find a value greater than the pivot
Step 3: Shift the right pointer to find a value less than the pivot
Step 4: Swap values at the poiners
Step 5: Repeat the above steps until two pointers both meet or cross
left
pivot
right
swap

- the target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements before x, and all greater elements after x

The Backtracking method

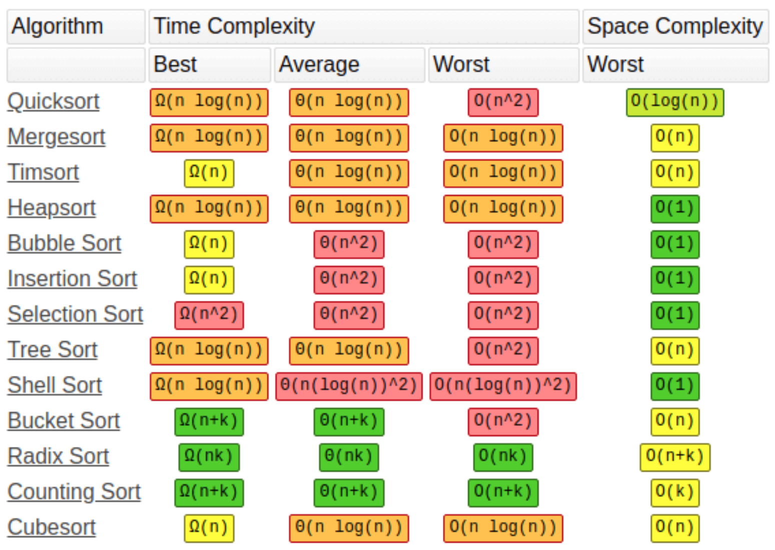
- it is applicable to search problems with more solutions
- its main disadvantage is that it has an exponential run time
- the goal is to get all the solutions for a problem using brute force approach
- it consists of building a set of all solutions incrementally
- since a problem has constraints, the solutions that will fail to satisfy them will be removed
- it uses recursive calling to find a solution set by building a solution step by step, increasing levels with time
- in order to find these solutions, a search tree named state-space tree is used. In a state-space tree, each branch is a variable and each level represents a solution
- a backtracking algorithm uses the depth-first search method, when it starts exploring the solutions, a bounding function is applied so that the algorithm can

check if the so-far built solution satisfies the constraints. If it does, it continues searching, if it does not, the branch would be eliminated and the algorithm goes back to the level before

Algorithms Complexity

- big O describes the upper bound of an algorithm: WC
- big Omega describes the lower bound of an algorithm: BC
- big Theta describes the tight bound of an algorithm, its limit from above and below: AC (or expected)
- for the algorithms above, we have:
    - Sequential Search:
        - WC: O(n)
        - BC: O(1)
        - AC: O(n)
        - Space Complexity: O(1)
    - Binary Search:
        - WC: O(logn)
        - BC: O(1)
        - AC: O(logn)
        - Space Complexity: O(1)
    - Selection Sort:
        - WC: O(n^2)
        - BC: Omega(n^2)
        - AC: Theta(n^2)
        - Space Complexity WC: O(1)
    - Bubble Sort:
        - WC: O(n^2)
        - BC: Omega(n)
        - AC: Theta(n^2)
        - Space Complexity WC: O(1)
    - Insertion Sort:
        - WC: O(n^2)
        - BC: Omega(n)
        - AC: Theta(n^2)
        - Space Complexity WC: O(1)
    - Merge Sort:
        - WC: O(n*log(n))
        - BC: Omega((n*log(n))
        - AC: Theta(n*log(n))
        - Space Complexity WC: O(n)
    - Quick Sort:

- WC: O(n^2)
- BC: Omega(n*log(n))
- AC: Theta(n*log(n))
- Space complexity WC: O(log(n))



Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

OOP Concepts in programming languages (we will use C++)

1. Object
- any entity that has a state and behavior is known as an object

2. Class
- collection of objects is called a class

3. Members of a class & the *this* pointer
- C++ classes have their own members.
- these members include variables (including other structures and classes), functions known as methods, constructors and destructors
- in order to refer to class attributes or methods the dot '.' or '->' arrow operator is used
- however, inside methods, in order to refer to attributes or (other) methods only their name needs to be used, the operators being optional
- the compiler automatically generates a special pointer, the *this* pointer, at each method call and it uses the generated pointer to identify attributes and methods
- the *this* pointer will be declared automatically as a pointer to the current object

4. Access Modifiers
- are keywords in OOP languages that set the accessibility of classes, methods and other member
    - private
    - protected

    - public

5. Constructors and Destructors

- constructor in C++ is a special member function of a class whose task is to initialize the object of the class
- the constructor name has to be the same with the class name
- a class may have multiple constructors. In this case, these methods will have the same name and it is possible due to function overloading. the number & type of formal parameters has to be different otherwise the compiler cannot choose the correct constructor
- constructors do not return any value
- we have 2 special types of constructors:
    - default constructor
        - a constructor without any parameters
    - copy constructor
        - it is used for object initialization given an object of the same type
        - the copy constructor is declared as follows: *class_name(const class_name& object);*
        - the const keyword expresses the fact that the copy constructor's argument is not changed
- a destructor is also a member function of a class that is instantaneously called whenever an object is destroyed
- global object destructor is called automatically at the end of the main function as part of the exit function
- using the exit function in a destructor is not recommend as it leads to an infinite loop
- local object destructor is executed automatically when the block in which these objects were defined is finished
- in case of dynamically allocated objects, the destructor is usually called indirectly via the delete operator (provided that the object has been previously created using the new operator)
- there is also an explicit way of calling the destructor and this is the case where the destructor name needs to be preceded by the class name and the scope resolution operator
- the destructor name starts with the '~' character followed by the class name
- the destructor does not return any value

6. Data Protection in Modular Programming

- in procedural programming, developing programs means using functions and procedures for creating the wanted programs
- in C/C++ programming languages instead of functions and procedures we have functions that return a value and functions that do not return a value

- in case of large applications, it is desirable to have some kind of data protection, meaning that only some functions have access to problem data, specifically functions referring to that data
- in modular programming, data protection can be achieved by using static memory allocation

7. Abstract Data Types

- they enable a tighter bound between the problem data and operations referring to these data
- an abstract data type declaration is similar to a struct declaration, which apart of the data also declares or defines functions referring to these data
- the functions declared within the struct will be called methods and the data will be called attributes
- if a method is declared within the struct, then it is considered an inline method
- if a method is declared outside the struct then the function name will be replaced by the abstract data type name followed by the scope resolution operator '::' and the method name

8. Class declaration

- a class abstract data type is declared as: *class 'name' { }*
- data protection is achieved by using the access modifiers:
    - private
    - protected
    - public
- private, protected represent protected data, while public represents unprotected data

Derived Classes and Inheritance

1. Theoretical Basis

- the use of abstract data types creates an ensamble for managing data and operations on this data
- usually, the protected elements can only be accessed by the methods of the given class. this property of objects is called *encapsulation*
- we do not see separate objects only, but also different relationships among these objects and among classes the objects belong to. In this way a class hierarchy is formed. the result is a second property of objects: *inheritance.* this means that all attributes and methods of the base class are inherited by the derived class, but new members (both attributes and methods) can be added to it. if a derived class has more than one base class, we talk about *multiple inheritance*

2. Declaration of Derived Classes

- a derived class is declared in the following way:
    - *class name_of_derived_class: list_of_base_classes { //new attributes and methods };*

- the public, protected, private keywords are called inheritance access modifiers in this situation too
- if the access modifier is missing it means it is private
- the private members of the base class are inaccessible in the derived class
- protected and private members become protected and private, respectively and remain unchanged if the inheritance access modifier is public
- this is why, generally, attributes and methods are declared protected and inheritance access modifier is public. they can be accessed but are protected in the derived classes too

  3. Virtual Functions
- polymorphism leads naturally to the problem of determining the method that will be called for a given object
- if a method is virtual, than, for every call of it, the implementation corresponding to the class hierarchy will not be determined at compile time, but at execution-time, depending on the type of the object on which the call was made. this property is called *dynamic binding*, and if the method is determined at compile time, it is called *static binding*.

Method Overriding

- another important property of objects belonging to the derived class is that the methods can be *overridden*
- this means that an operation related to objects belonging to the hierarchy has a single signature, but the methods that describe the operation can be different
- so, the name and the list of formal parameters of the method is the same in both the base and the derived class, but the implementation can be different

Polymorphism

- the derived class methods can be specific to that class, although the operation is identified through the same name. this property is called *polymorphism*
- in C++ it means that the same entity (method or object) behaves differently in different scenarios
- as an example, the '+' operator can perform an addition when it comes to numbers, or a concatenation when it comes to strings
- types of polymorphism in C++:
  - compile time polymorphism:
    - in this type of polymorphism, a function is called at the time of program compilation. it is also called 'early binding' or 'static binding'. function overloading and operator overloading is the type of compile time polymorphism
      - function overloading: means one function can perform many tasks. in C++, a single function is used to perform many tasks

        with the same name and different types of arguments. In the function overloading, function will call at the time of program compilation
        - operator overloading: means defining additional tasks to operators without changing its actual meaning
  - run time polymorphism:
    - functions are called at the time of program execution, hence it is called also late binding or dynamic binding
    - function overriding is a part of runtime polymorphism. In function overriding, more than one method has the same name with different types of the parameter list
    - it is achieved by using virtual functions and pointers. It provides slow execution as it is known at the run time, thus it is more flexible as all things executed at run time
      - function overriding: we give a new definition to the base class function in the derived class. at that time, we can say the base function has been overridden. it can be only possible in the 'derived' class. in function overriding, we have 2 definitions of the same function, one in the super class (parent class, base class, etc) and one in the derived class
      - virtual function: it is declared by the keyword 'virtual'. it is a member of the base class and we can redefine it in the derived class
      - pure virtual function: the declaration of this functions happens in the base class but the definition will be written in the derived class

Dynamic Binding

- it is determining the method to invoke at runtime instead of compile time
- it is also referred to as 'late binding'

Abstract Classes and Interfaces

1. Abstract Classes

- In case of a complicated class hierarchy, the base class can have some properties which we know exist, but we can only define them for the derived class
- classes that contain at least one pure virtual method are called *abstract classes*
- since abstract classes contain methods that are not defined, it is not possible to create objects that belong to an abstract class
- if a pure virtual method was not defined in the derived class, then the derived class will also be abstract and it is impossible to define objects belonging to it

2. Interfaces

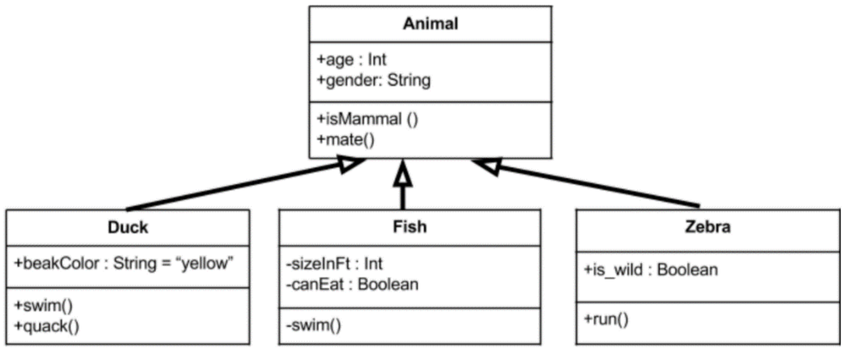
- the C++ programming language has no notion of interfaces

- but any abstract class that contains only pure virtual methods can be considered an interface

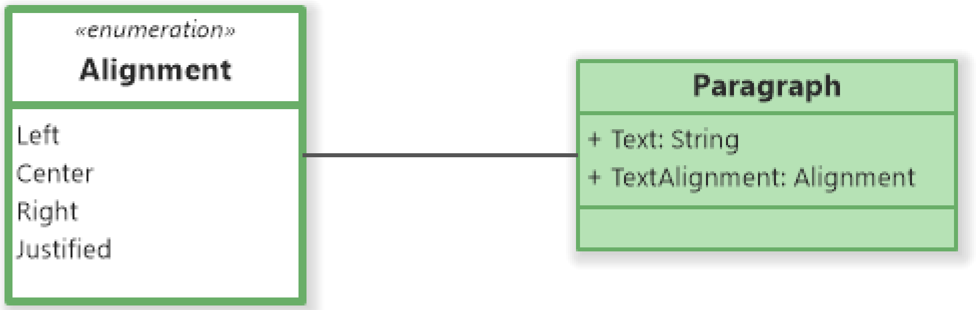
UML

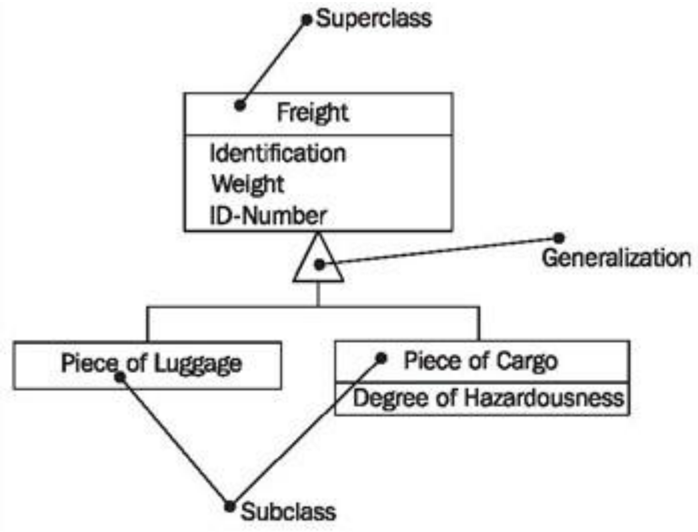
- UML (Unified Modelling Language) defines a set of modelling elements and graphical notations associated to these elements. the modelling elements can be used for describing any software system. particularly, the UML language contains elements which can be used for the Object Oriented Systems

1. Class Diagram



- a diagram is a graphical representation of the elements from a model
- the class diagrams represent the types of objects used in a system, as well as the relationships between them
- an *UML Class* represents a set of objects with the same structural elements (properties) and behavioral elements (operations). The UML classes are data types and correspond to application classes from Java, C++ and C# languages.
- an UML Class may be derived from many other classes. The use of multiple inheritance in the model does not lead to a direct correspondence between the model and the code in the case of Java or C# languages
- an UML class can implement many interfaces as in Java or C#. the correspondence between the models which contain classes that implement multiple interfaces and C++ is made through purely abstract C++ classes and multiple inheritance
- the *substation principle* is applicable for all instances having a class or interface type as in Java, C#, C++. This means that the instances of a class may be replaced with instanced of the derived types, without semantically altering the program
    - Interfaces
        - it is a data type which declares a set of operations
        - a class that implements the interface needs to provide the declared operations

- Enumerations
  - they describe a set of symbols which have no associated values
  - the structured types are modelled using the datatype stereotype and correspond to C++/C# structures and primitive types from Java
  - the instances of these data types are identified by their values
  - they are used to describe the classes properties and correspond to value objects, but they have no identity
  - 

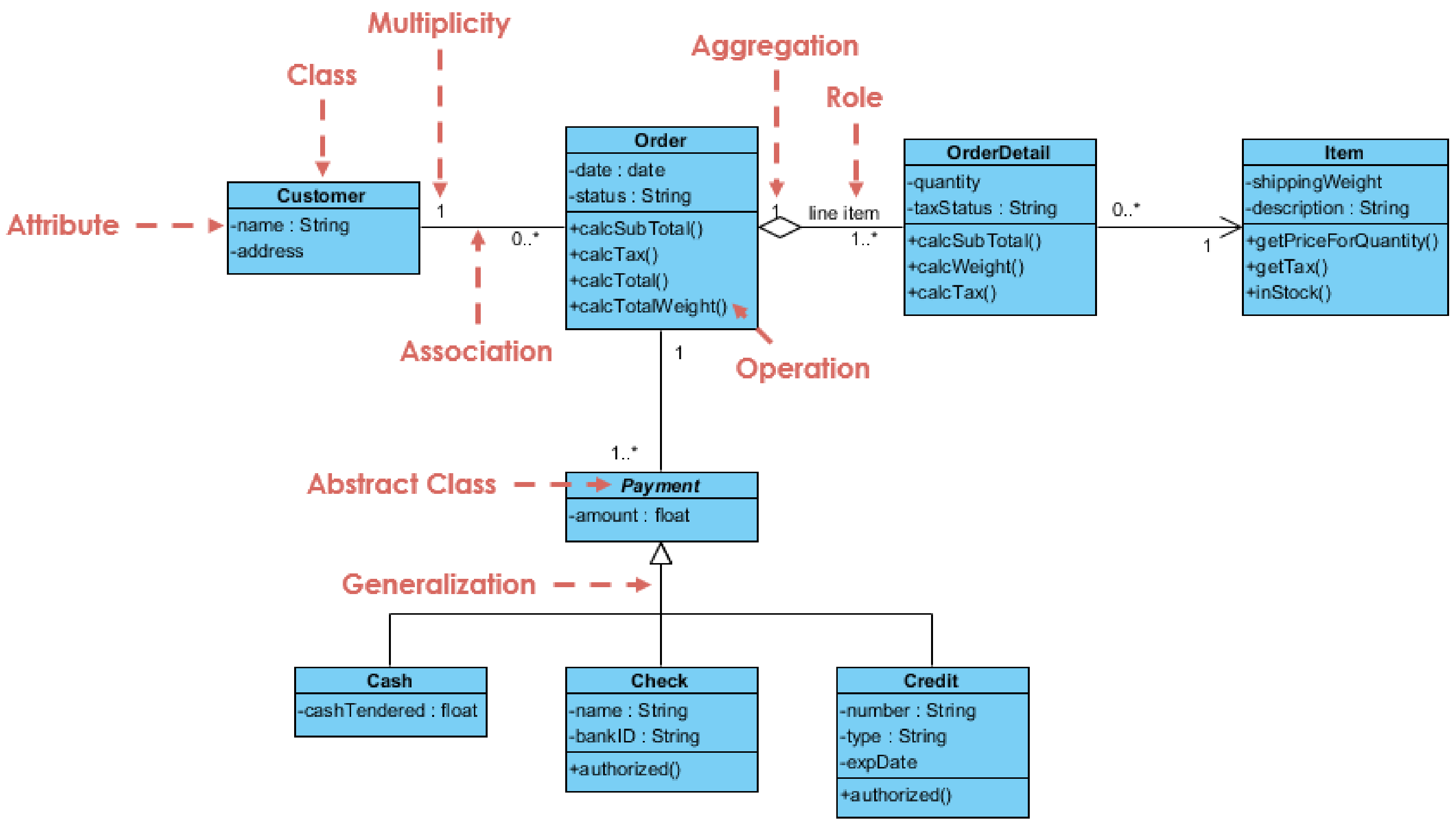
- Generalization and Interface Realization
  - the generalization is a relation between a more general data type and one more specialized
  - this relation can be applied between two classes or two interfaces, corresponding to the inheritance relations between classes
  - the realization of an interface represents a relation between a class and an interface and indicates that the class is conform to the contract specified by the interface
  - 

- Properties
  - represent structural aspects of a data type
  - the properties of a class are introduced through attributes and associations
  - an attribute describes a property of the class in its second compartment as: *visibility name: type multiplicity= value {properties}*

    - the name is compulsory, as well as the visibility which can be:
        - + public
        - - private
        - # protected
    - the other elements for declaring a property are optional
    - the type of a property can be any of:
        - class
        - interface
        - enumeration
        - structured type
        - primitive type: String, Integer, Boolean, UnlimitedNatural
    - the multiplicity can be 1 (implicit value when the multiplicity is not specified)
        - 0..1: optional
        - 0..*: zero or more values
        - 1..*: one or more values
        - m..n: between m and and values
        - m..* we can additionally specify if:
            - the values can be repetitive or not
            - the values can be referred to as indices
- Associations
    - represent a set of tuples, each tuple linking two instances of some data types
    - an association is a data type which links properties of other data types
    - the unidirectional association introduce properties in the source class, having the type of the destination class
    - the bidirectional associations link two properties from two different classes or from the same class
    - a compulsory step that has to be made during the detailed analysis is the association refinement, firstly the transformation of the bidirectional associations into unidirectional ones
    - an aggregation is an association which indicates that an object is a part of another object
    - a containment is an aggregation which additionally indicates that the contained objects can be part of a single whole
    - the associations' refinement also includes the setting of the aggregation and containment relationships
    - we can define static or class type properties these being represented through underlining
- Dependencies

- between 2 software elements, client and supplier, a dependency exists if changing the definition of the supplier leads to the change of the client

- Operations
    - define the behavior of the objects and correspond to the methods in the OOP programming languages
    - the operations specify the behavior (represent the signature) and the implementation is defined by behavioral elements such as interactions, state machines and activities
    - the implementations are known as methods in UML
    - the syntax for specifying an operation is: *visibility name (list-of-parameters): returned type {properties}*
    - the parameters from the list-of-parameters are separated by commas, a parameter having the following form: *direction name: type = implicit-value*
    - direction can be: in, out and in-out, implicitly being in



Relations between Classes

- described and defined above

Lists

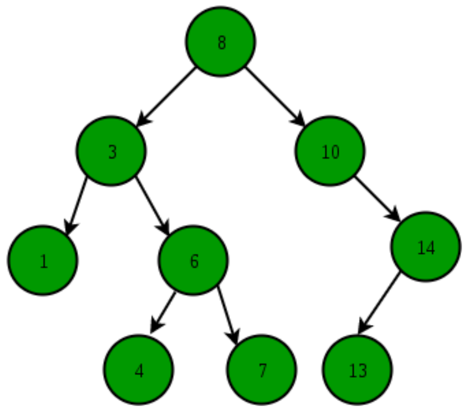
- a list can be seen as a sequence of elements <l1,..ln> of the same type that are in a certain order
- each element has a well established position in the list, as a result the position of the elements in the list is essential: access, deletion and addition can be made based on a position in the list
- list- dynamic collection of elements in which the order of the elements is essential. the number n of elements in the list is called the length
    - list of length 0 is an empty list
    - the dynamic character of the list is given by the fact that the list may change over time because of additions and deletions
- a linear list is a structure that is either empty or:
    - has an unique first element
    - has an unique last element
    - each item in the list (except the last element) has one successor
    - each item in the list (except the first element) has one predecessor
- in a linear list we can insert items, delete items, determine the position of the successor (or predecessor) of an element, you can access an element based on its position, etc.
- a list is called circular if:
    - the predecessor of the first node is the last node and the last node's successor is the first node
- position of an element in a list can be viewed in different ways:
    - given by rank (order number) item in the list. the position of an item in the list is its index in the list
    - given by reference to the location where the element is stored (eg: pointer to the location where the element is stored)
- we say that a position p is valid if it is the position of a list element

Maps

- are containers that contain pairs (key->value)
- they store items so they can be easily located using keys
- in a map, keys are unique and a key has an unique associated value

Binary Search Trees

- is a node-based binary tree data structure which has the following properties:
    - the left subtree of the node contains only nodes with keys lesser than the node's key
    - the right subtree of the node contains only nodes with keys greater than the node's key
    - the left and right subtree must also be a binary search tree



Hash Tables

- a data structure which stores data in associative manner
- data is stored in array format, where each data value has its own unique index value
- access of data becomes very fast if we known the index of the desired data
- it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data
- hash tables uses an array as storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from
- hashing is a technique to convert a range of key values into a range of indexes of an array
- to resolve conflicts, we can use
    - linear probing: searches the next empty location by looking into the next cell until we find an empty one

Identify Data Structures and Data Types suitable for solving problems. The use of existing libraries for these structures (C++)

Part 2. Databases

- **Courses: Databases**

The Relational Model

- consider A1, A2,.., An a set of attributes (or columns, fields, data names, etc) and Di = Dom(Ai) U {?} the domain of possible values of attribute Ai, where "?" is denoted "undefined" (null) value.
- "undefined" value is used to specify if an attribute has a value assigned or it has no value. this value does not belong to a specific data type and it could be used in expression together with other attributes values having different types (number, text, data, etc..)

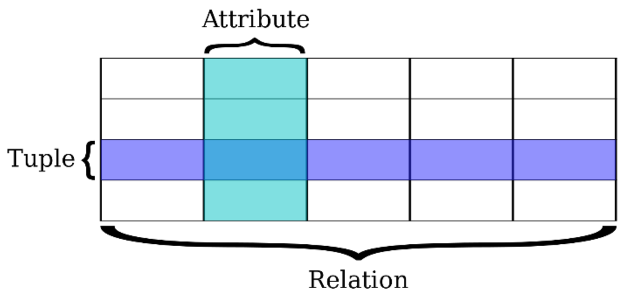
1. Relations

- using the domains defined above, we could define a relation of degree (arity) n as: $R \subseteq D1 \times D2 \times \ldots \times Dn$ and it could be considered as a set of vectors with n values, one value for each attribute Ai.

R	*A1*	…	*Ai*	…	*An*
r1	*a11*	…	*a1j*	...	*a1n*
…	…	…	…	…	...
ri	*ai1*	…	*aij*	…	*ain*
…	…	…	…	...	...
rn	*am1*	…	*amj*	…	*amn*

- lines represent relation elements, or tuples, or records which are distinct
- because the way of representing the elements of a relation R is similar with a table, the relationship is called *table*
- a relation instance ([R]) is a subset of D1 X D2 X .. X Dn
- a relational database is a set of relations
- a database schema is the set of schemas of the relations in the database
- a database instance (state) is the set of instances) of the relations in the database
- a relational database has 3 parts:
  - data (relations content) and its description
  - integrity constraints (to preserve database consistency)
  - data management operators

- a field is a data structure for a single piece of data, fields are organized into records, which contain all the information within the table relevant to a specific entity
- a field or a set of fields is a key for a relation. if there are not, 2 tuples can have the same values for all fields and this is not true for any subset of the key



- a relation may have more keys: one of them is chosen as a primary key, all the others being considered secondary keys (or key candidates)

- relational databases do not allow two or more rows of a relation having the same value for any key

2. Integrity Constraints
    - a key is an integrity constraint for a database
    - integrity constraints are conditions that must be true for any instance of the database
    - integrity constraints are specified when schema is defined and are checked when relations are modified
    - a legal instance of a relation is one that satisfies all specified integrity constraints
    - using foreign keys we can store 1:n relations between entities
    - a foreign key could be used to store m:n relations too
    - a m:n relation is stored using a so called cross table

    Domain Constraints

    - are user-defined columns that help the user enter the value according to the data type
    - if it encounters a wrong input it gives the message that the column is not fulfilled properly
        - not null
        - check: it defines a condition that each row must satisfy which means it restricts the value of a column between ranges or we can say that it just like a condition or filter checking before saving data into a column

Constraints for attributes:

- NOT NULL- the attribute cannot have “undefined” value assigned
- PRIMARY KEY- the current attribute is the primary key of the relation
- UNIQUE- the current attribute is a key candidate
- CHECK(CONDITION)- simple logical conditions which should be true for a consistent database
- FOREIGN KEY f REFERENCES PARENT_TABLE [(ATTRIBUTE_NAME)] [ON UPDATE/ DELETE action]- the current attribute is a reference to a record stored in another relation

Constraints for relations:

- PRIMARY KEY (list of attributes)- definition of a composed primary key
- UNIQUE (list of attributes)- definition of a composed candidate key
- CHECK (condition)- condition that involves more than one attribute of a relation
- FOREIGN KEY f (list of attributes) REFERENCES PARENT_TABLE [(list of attributes)] [ON UPDATE/DELETE action]- defines a composed reference to a record stored in other table

SQL

1. Create
    - CREATE DATABASE is used to create a new SQL database
    - CREATE TABLE is used to create a new table in the database
    - CREATE INDEX is used to create indexes in the table (allows duplicates)
    - CREATE UNIQUE INDEX is used to create a unique index in the table
    - CREATE VIEW- is used to create a view
    - CREATE PROCEDURE- is used to create a stored procedure
2. Alter
    - ALTER TABLE command adds, deletes or modifies columns in a table
    - ALTER COLUMN is used to change the data type of a column in a table
3. Drop
    - DROP COLUMN is used to delete a column in an existing table
    - DROP a UNIQUE constraint
    - DROP a PK constraint
    - DROP a FK constraint
    - DROP a CHECK constraint
    - DROP DEFAULT is used to delete a default constraint
    - DROP INDEX
    - DROP DATABASE
    - DROP TABLE
    - DROP VIEW

4. Primary Key
    - PK constraint uniquely identifies each record in a table
5. Foreign Key
    - FK constraint is a field (or a collection of fields) in one table that refers to the PK of another table
6. Unique
    - the UNIQUE constraint ensures that all values in a column are unique
7. Check
    - The CHECK constraint limits the value that can be placed in a column
8. Null
    - NOT NULL: enforces a column to not accept NULL values
9. Default
    - DEFAULT constraint provides a default value for a column. It will be added to all new records if no other value is specified
10. Select
    - it is used to select data from a database, the data returned is stored in a result table called the result set
11. Insert
    - INSERT INTO table VALUES: is used to insert new rows in a table
12. Update
    - UPDATE command is used to update existing rows in a table
13. Delete
    - DELETE command is used to delete existing rows in a table
14. 3-valued logic
    - besides TRUE and FALSE, the result of a logical expression can also be UNKNOWN
    - SQL's three valued logic is a consequence of supporting null to mark absent data. If a null value affects the result of a logical expression, the result is neither true nor false but unknown.
15. Distinct
    - the SELET DISTINCT command returns only distinct values in the result set
16. From
    - the FROM command is used to specify which table to select or delete data from
17. Where
    - the WHERE command filters a result set to include only records that fulfill a specified condition
18. Group By
    - the GROUP BY command is used to group the result set (used with aggregate functions: COUNT, MAX, MIN, SUM, AVG).
19. Having
    - the HAVING command is used instead of WHERE with aggregate functions

20. Order By
- the ORDER BY command is used to sort the result set in ascending or descending order.
- the ORDER BY command sorts the result set in ascending order by default. To sort the records in descending order, use the DESC keyword.

21. Top
- the TOP command is used to specify the number of records to return
- eg: for SELECT TOP 3 * FROM table it returns the first 3 records

22. In
- the IN command allows you to specify multiple values in a WHERE clause

23. Exists
- the EXISTS command tests for the existence of any record in a subquery, and returns true if the subquery returns one or more records

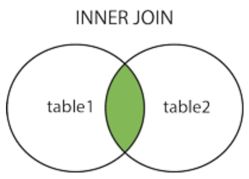
24. Any
- the ANY command returns true if any of the subquery values meet the condition

25. All
- ALL command returns true if all of the subquery values meet the condition

26. Joins

   a. Inner Join

      i. selects records that have matching values in both tables



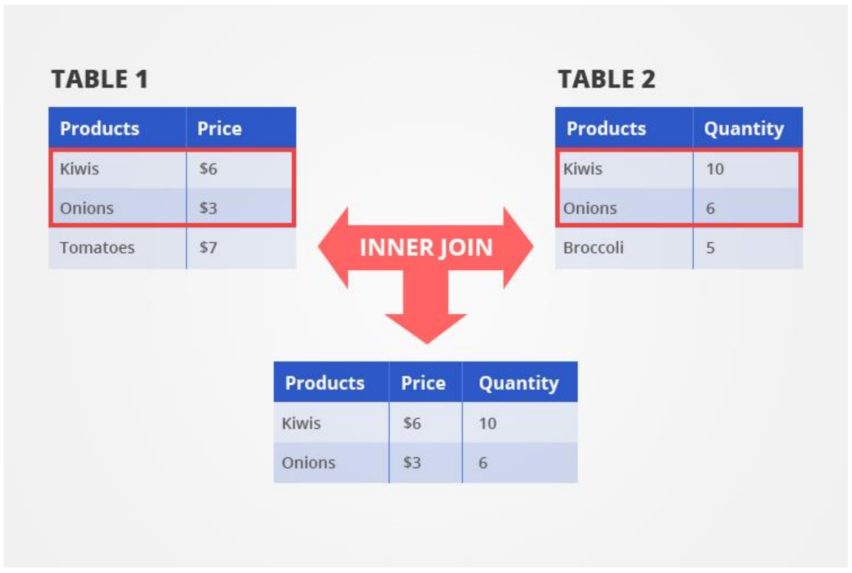


TABLE 1

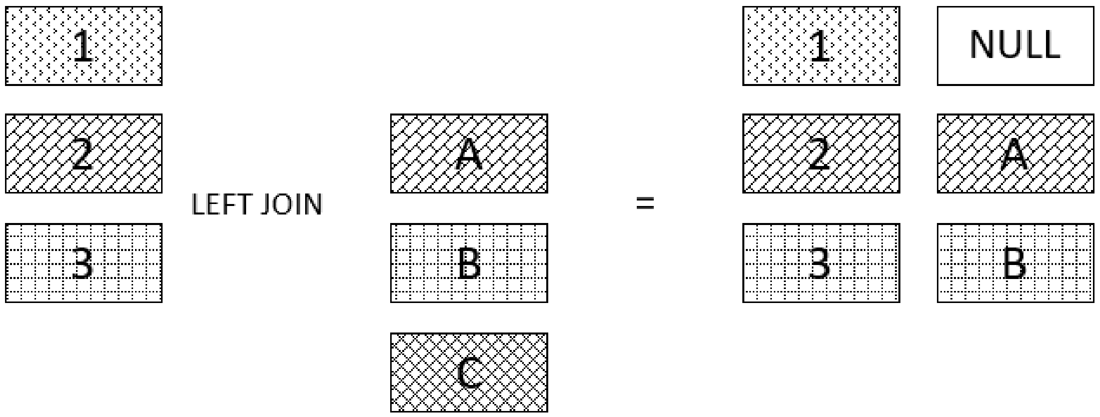
Products	Price
Kiwis	$6
Onions	$3
Tomatoes	$7

TABLE 2

Products	Quantity
Kiwis	10
Onions	6
Broccoli	5

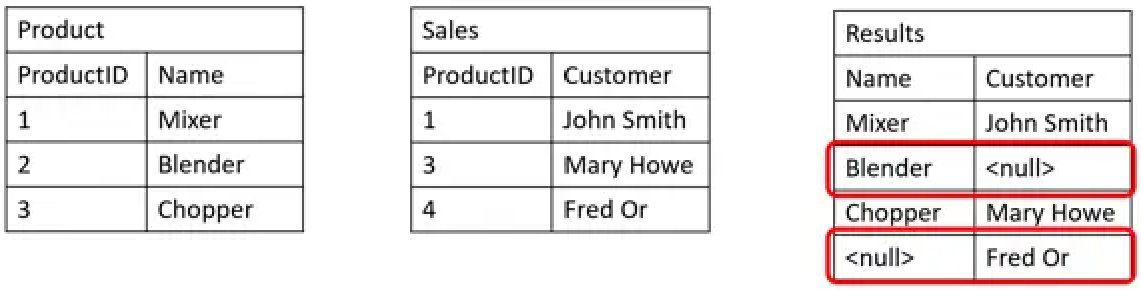
Products	Price	Quantity
Kiwis	$6	10
Onions	$3	6

b. Left Join
   i. returns all rows from the left table and the matching rows from the right table. If no matching rows are found in the right table, NULL are used



c. Right Join
   i. Returns all rows from the right table and the matching rows from the left table. If no matching rows are found in the left table, NULL are used

d. Full Join
   i. combines the results of both left and right outer joins.
   ii. The joined table will contain all records from both the tables and fill in NULLs for missing matches on either side



Product

ProductID	Name
1	Mixer
2	Blender
3	Chopper

Sales

ProductID	Customer
1	John Smith
3	Mary Howe
4	Fred Or

Results

Name	Customer
Mixer	John Smith
Blender	<null>
Chopper	Mary Howe
<null>	Fred Or

27. Union [ALL]

- UNION ALL command combines the result set of two or more SELECT statements (allows duplicate values)

28. Intersect

- INTERSECT clause/operator is used to combine two SELECT statements, but returns rows only from the first SELECT statement that are identical to a row in the second SELECT statement. This means INTERSECT returns only common rows returned by the two SELECT statements

29. Except
    - SQL EXCEPT statement returns those records from the left SELECT query, that are not present in the results returned by the SELECT query on the right side of the EXCEPT statement
30. Count
    - COUNT() function returns the number of rows that matches a specified criterion
31. Sum
    - SUM() function returns the total sum of a numeric column
32. Avg
    - AVG() function returns the average value of a numeric column
33. Min
    - MIN() function returns the smallest value of the selected column
34. Max
    - MAX() function returns the greatest value of the selected column
35. Nested Queries
    - a query is written inside a query. The result of inner query is used in execution of outer query
36. Between
    - the BETWEEN operator selects values within a given range. The values can be numbers, text, or dates.
    - the BETWEEN operator is inclusive: begin and end values are included
37. Like
    - LIKE operator is used in a WHERE clause to search for a specified pattern in a column.
    - there are two wildcards often used in conjunction with the LIKE operator:
        - the percent sign (%) represents zero, one, or multiple characters
        - the underscore sign (_) represents one, single character
        - ‘a%’ finds any value that starts with a
        - ‘%a’ finds any value that ends with a
        - ‘_r%’ on the second position
        - ‘a%o’ starts with a, ends with o
        - ‘a_’ starts with a and has 2 characters

Functional Dependencies. Normal Forms

1. Definition
- a functional dependency is a relation between attributes
- it typically exists between the primary key and non-key attribute within a table
- the left side of FD is known as a determinant and the right side of the production as a dependent

2. Basic properties
    a. Reflexivity

i. if alpha is a set of attributes and beta is a subset of alpha, then alpha holds beta

b. Transitivity

i. if a->b holds b->c holds, then a->c holds

c. Augmentation

i. if a->b holds and y is attribute set, then ay->by also holds

d. Union

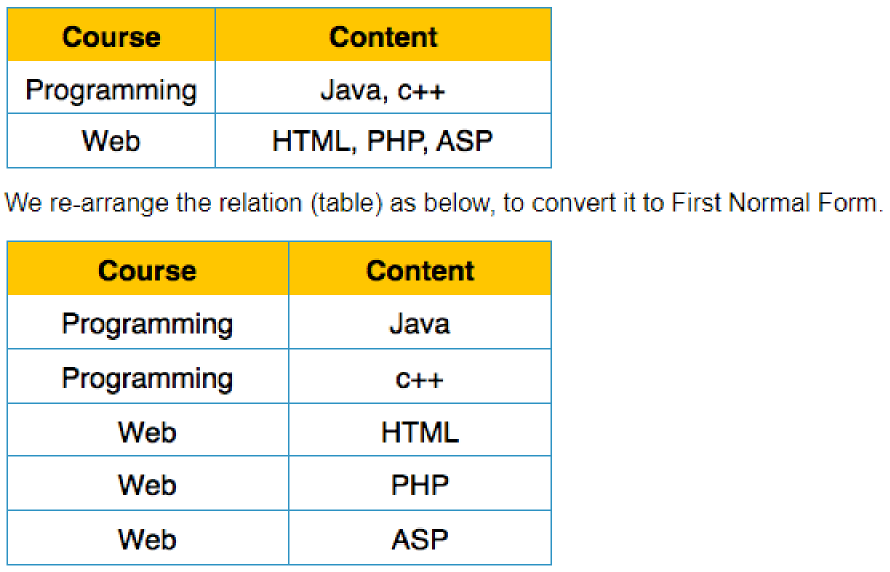
i. if two tables are separate and the PK is the same, you should consider putting them together

e. Decomposition

i. it is a rule that suggests if you have a table that appers to contain 2 entities which are determined by the same PK then you should consider breaking them up in 2 different tables

3. 1NF

- it is defined in the definition of relations (tables) itself
- this rule defines that all the attributes in a relation must have atomic domains
- the values in an atomic domain are indivisible units



Course	Content
Programming	Java, c++
Web	HTML, PHP, ASP

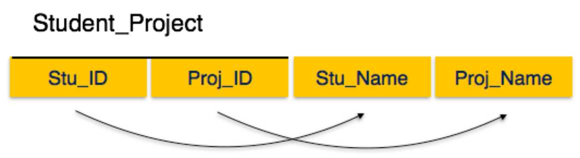
We re-arrange the relation (table) as below, to convert it to First Normal Form.

Course	Content
Programming	Java
Programming	c++
Web	HTML
Web	PHP
Web	ASP

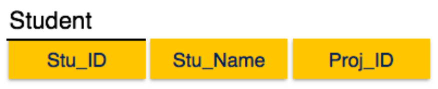
4. 2NF

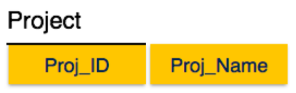
- before defining the 2NF we need to understand the following:
  - prime attribute: an attribute, which is part of the candidate-key, is known as a prime attribute
  - non-prime attribute: an attribute, which is not part of the prime-key, is known as a non-prime attribute

- if we follow 2NF form, then every non-prime attribute should be fully functionally dependent on prime key attribute
- that is X -> A holds, then there should not be any proper subset Y of X for which Y->A holds true



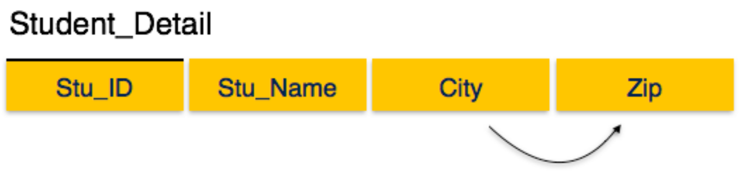
We see here in Student_Project relation that the prime key attributes are Stu_ID and Proj_ID. According to the rule, non-key attributes, i.e. Stu_Name and Proj_Name must be dependent upon both and not on any of the prime key attribute individually. But we find that Stu_Name can be identified by Stu_ID and Proj_Name can be identified by Proj_ID independently. This is called **partial dependency**, which is not allowed in Second Normal Form.





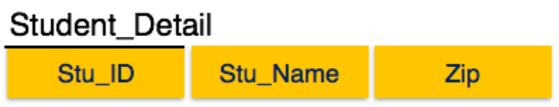
5. 3NF

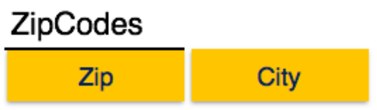
- for a relation to be in 3NF, it must be in 2NF and the following must satisfy:
  - no non-prime attribute is transitively dependent on prime key attribute
  - for any non-trivial functional dependency, X->A, then either:
    - X is a superkey or A is a prime attribute



We find that in the above Student_detail relation, Stu_ID is the key and only prime key attribute. We find that City can be identified by Stu_ID as well as Zip itself. Neither Zip is a superkey nor is City a prime attribute. Additionally, Stu_ID → Zip → City, so there exists **transitive dependency**.

To bring this relation into third normal form, we break the relation into two relations as follows −



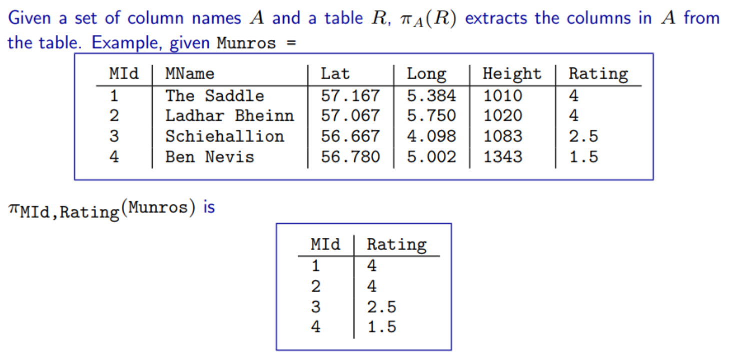


6. BCNF

- Boyce-Codd NF is an extension of the 3NF on strict items
- BCNF states that:
    - for any non-trivial dependency, X->A, X must be a superkey
- in the above images, Stu_ID is a super key in the Student_Detail table and Zip is a super key in the ZipCodes table, then the relations are in BCNF

Relational Algebra on Sets

1. Selection
    - SELECT (symbol: σ)
    - the SELECT operation is used for selecting a subset of tuples according to a given selected condition
    - σ topic = "Database" (Tutorials) selects tuples from Tutorials where topic = “Database”
2. Projection
    - PROJECT (symbol: π)
    - it eliminates all attributes of the input relation but those mentioned in the projection list
    - the projection method defines a relation that contains a vertical subset of relation
    - this helps to extract the values of specified attributes to eliminate duplicate values


Given a set of column names A and a table R, $\pi_A(R)$ extracts the columns in A from the table. Example, given Munros =

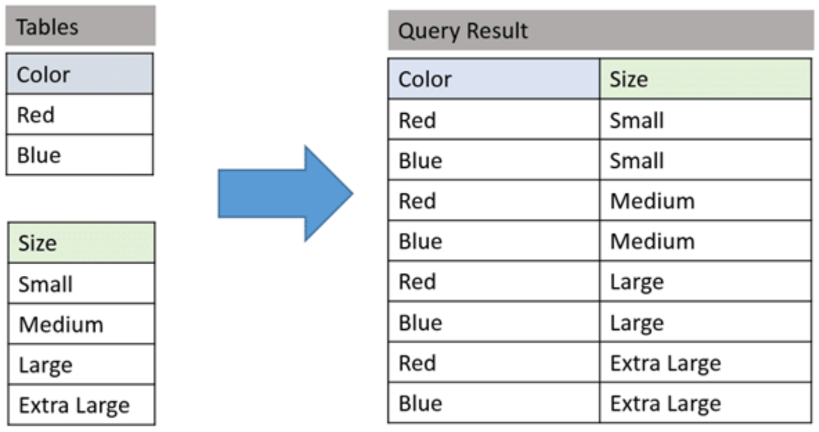
MId	MName	Lat	Long	Height	Rating
1	The Saddle	57.167	5.384	1010	4
2	Ladhar Bheinn	57.067	5.750	1020	4
3	Schiehallion	56.667	4.098	1083	2.5
4	Ben Nevis	56.780	5.002	1343	1.5

$\pi_{\texttt{MId,Rating}}(\texttt{Munros})$ is

MId	Rating
1	4
2	4
3	2.5
4	1.5

3. Cross-Product
    - CARTESIAN PRODUCT, CROSS JOIN (symbol: X)

- is an operation used to merge columns from 2 relations



Tables

Color
Red
Blue

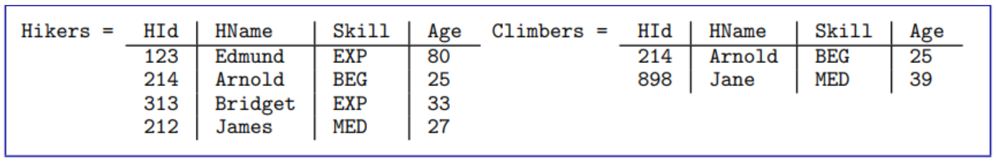
Size
Small
Medium
Large
Extra Large

Query Result

Color	Size
Red	Small
Blue	Small
Red	Medium
Blue	Medium
Red	Large
Blue	Large
Red	Extra Large
Blue	Extra Large

4. Union

- UNION (symbol: ∪)
- it includes all tuples that are in tables A or in B
- it also eliminates duplicate tuples

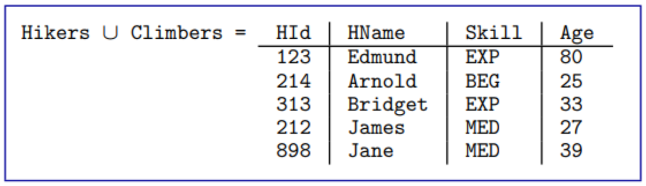


Hikers =

HId	HName	Skill	Age
123	Edmund	EXP	80
214	Arnold	BEG	25
313	Bridget	EXP	33
212	James	MED	27

Climbers =

HId	HName	Skill	Age
214	Arnold	BEG	25
898	Jane	MED	39

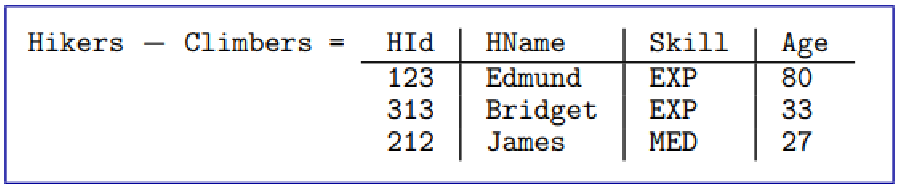


Hikers ∪ Climbers =

HId	HName	Skill	Age
123	Edmund	EXP	80
214	Arnold	BEG	25
313	Bridget	EXP	33
212	James	MED	27
898	Jane	MED	39

5. Set-Difference

- DIFFERENCE (symbol: -)
- it includes all tuples that are in A but not in B
- the attribute name of A has to match with the attribute name in B
- the two-operand relations A and B should be either compatible or union compatible
- it should be defined relation consisting of the tuples that are in relation A but not in B



Hikers — Climbers =

HId	HName	Skill	Age
123	Edmund	EXP	80
313	Bridget	EXP	33
212	James	MED	27

6. Intersection

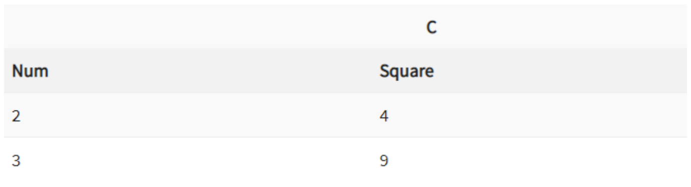
- INTERSECTION (symbol: ∩)
- defines a relation consisting of a set of all tuples that are in both A and B, however A and B must be union-compatible
-

7. Conditional Join (Theta Join)

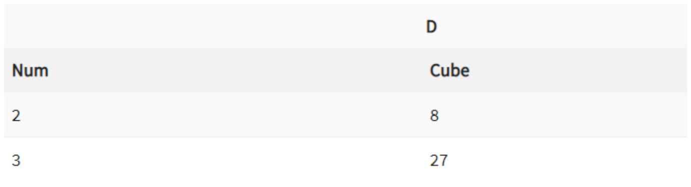
- THETA JOIN (symbol: θ)
- THETA join can use any conditions in the selection criteria

8. Natural Join

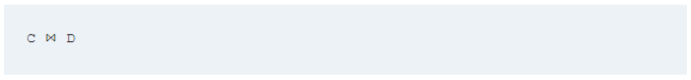
- NATURAL JOIN (symbol: ⋈)
- can only be performed if there is a common attribute (column) between the relations
- the name and type of the attribute must be the same



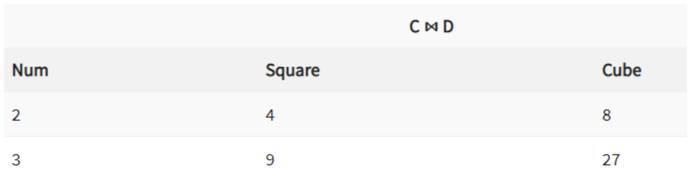
C	
Num	Square
2	4
3	9



D	
Num	Cube
2	8
3	27



```
C ⋈ D
```

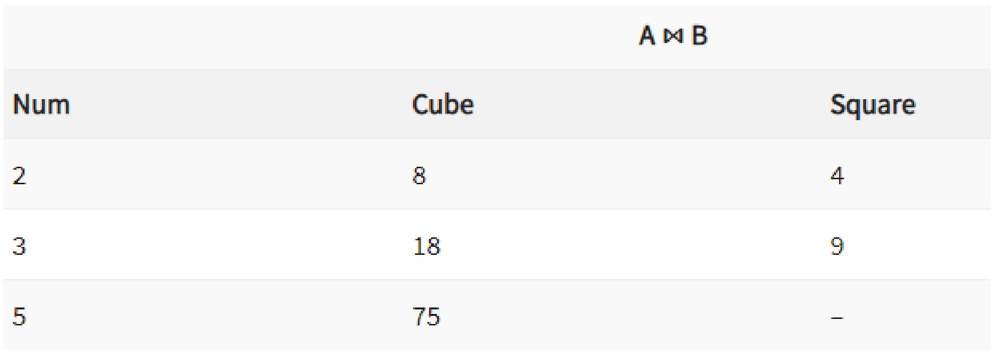


C ⋈ D		
Num	Square	Cube
2	4	8
3	9	27

9. Left Outer Join

- LEFT OUTER JOIN (symbol: ⟕)

- in the left outer join, operation allows keeping all tuple in the left relation, however, if there is no matching tuple found in the right relation, then the attributes of right relation in the JOIN result are filled with null values



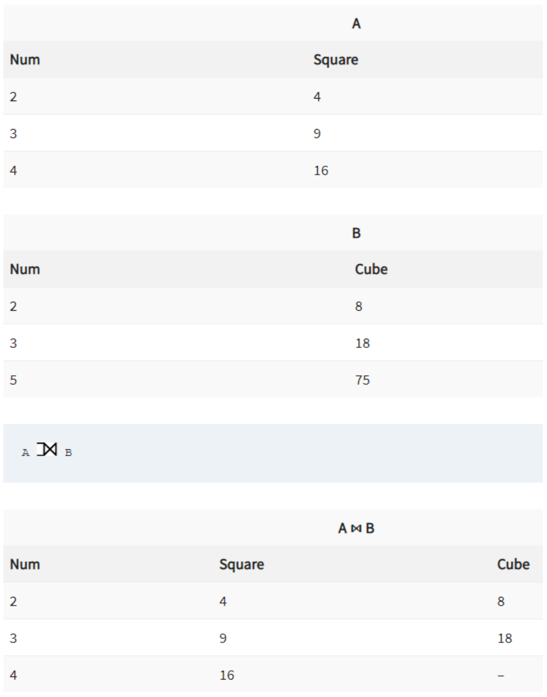
A ⋈ B		
Num	Cube	Square
2	8	4
3	18	9
5	75	–

10. Right Outer Join

- RIGHT OUTER JOIN (symbol: ⟖)
- operation allows keeping all tuple in the right relation
- however, if there is no matching tuple in the left relation, then the attributes of the left relation in the join result are filled with null values

11. Full Outer Join

- FULL OUTER JOIN(symbol: ⟗)
- all tuples from both relations are included in the result, irrespective of the



A	
Num	Square
2	4
3	9
4	16

B	
Num	Cube
2	8
3	18
5	75

A ⟗ B

A ⋈ B		
Num	Square	Cube
2	4	8
3	9	18
4	16	–

matching condition

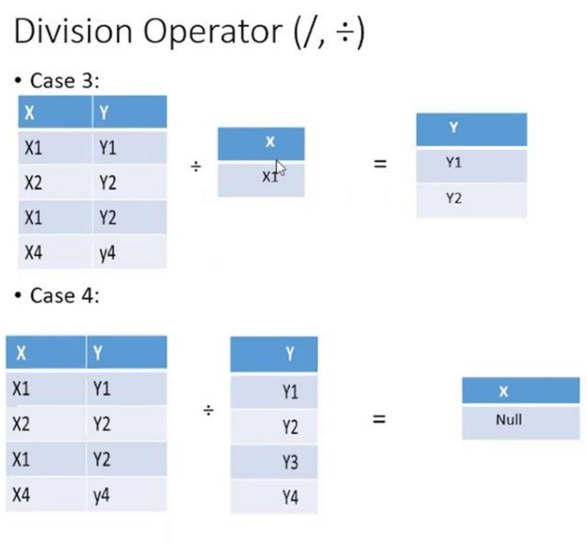
- the values which don’t match have ‘null’ as value

12. Division

- DIVISION (symbol: /)
- example: suppose we have 2 tables with schemas R(A, B) and S(B)
- R/S is defined to be the set of A values in R which are paired (in R) with all B values in S
- that is the set of all x for which πB(S) ⊆ πB(σA=x(R)).
- the relation returned by division operator will return those tuples from relation A which are associated to every B’s tuple

13. Assignment

- ASSIGNMENT (symbol: ←)
- similar to assignment operator in programming languages
- is useful in the situation where it is required to write relational algebra expressions by using temporary relation variables



Division Operator (/, ÷)

- Case 3:

X	Y
X1	Y1
X2	Y2
X1	Y2
X4	y4

÷

X
X1

=

Y
Y1
Y2

- Case 4:

X	Y
X1	Y1
X2	Y2
X1	Y2
X4	y4

÷

Y
Y1
Y2
Y3
Y4

=

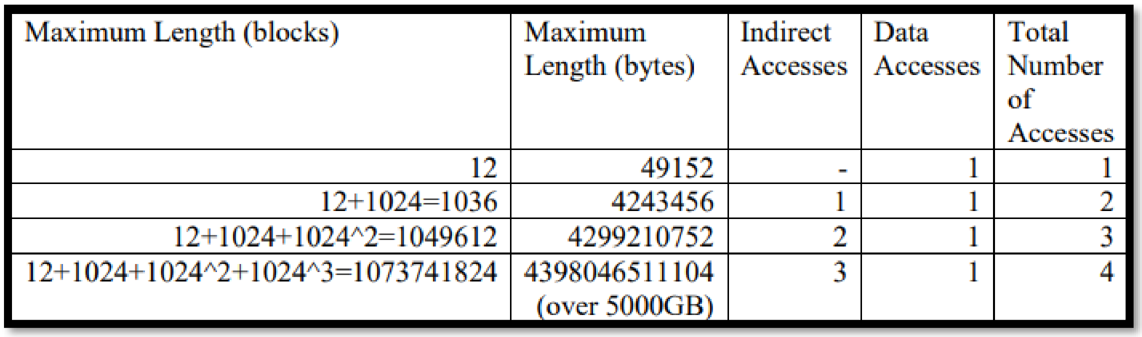
X
Null

Part 3. Operating Systems

- **Courses: Operating Systems**

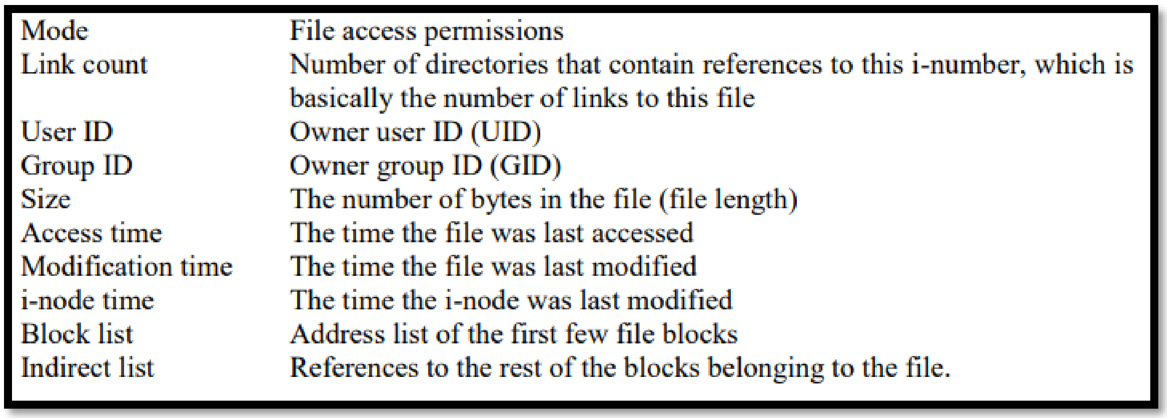
The structure of UNIX file systems

- a UNIX file system is stored either on a device such as a hard-drive or CD or on a hard-drive partition
- the hard-drive partitioning is an operation relatively independent of the operating system stored there
- a UNIX file is a sequence of bytes, each byte being individually addressable
- a byte can be accessed both sequentially as well as directly
- the data exchange between memory and the disk is done in block
- a UNIX file system is a structure stored on disk, and structured in four block types:
  - block 0 contains the OS boot loader, this is a program dependent on the machine architecture on which it is installed
  - block 1, the so called super-block contains information that defines the file systems layout on the disk. Such information is:
    - the number of i-nodes
    - the number of disk zones
    - pointers to the i-node allocation map
    - pointers to the free space
    - disk zone dimensions
  - blocks 2..n
- an i-node is UNIX name for the file descriptor
- the i-nodes are stored on disk as a list named i-list
- the order number of an i-node in the i-list is represented on two bytes and is called i-number
- this i-number is the link between the file and the user programs
- the largest part of the disk is reserved for file data



Maximum Length (blocks)	Maximum Length (bytes)	Indirect Accesses	Data Accesses	Total Number of Accesses
12	49152	-	1	1
12+1024=1036	4243456	1	1	2
12+1024+1024^2=1049612	4299210752	2	1	3
12+1024+1024^2+1024^3=1073741824	4398046511104 (over 5000GB)	3	1	4

- an i-node usually occupies between 64 and 128 bytes and contains the information in the table below:



Mode	File access permissions
Link count	Number of directories that contain references to this i-number, which is basically the number of links to this file
User ID	Owner user ID (UID)
Group ID	Owner group ID (GID)
Size	The number of bytes in the file (file length)
Access time	The time the file was last accessed
Modification time	The time the file was last modified
i-node time	The time the i-node was last modified
Block list	Address list of the first few file blocks
Indirect list	References to the rest of the blocks belonging to the file.

File Types and File Systems

1. Normal files are sequences of bytes accessible either sequentially or directly using the byte's number order
2. Directories: a directory file differs from a normal file only through the information it contains. A directory contains the list of names and addresses of the files it contains. Usually every user has a directory of its own which points to its normal files and other subdirectories
3. Special files: in this category, we can include:
   a. Hard links
   b. Symbolic links
   c. Sockets
   d. FIFO
   e. Peripheral character devices
   f. Peripheral block devices

- UNIX regards any I/O device as a special file
- every directory includes the two special entries below:
  - '.' Dot points to the directory itself
  - '..' points to the parent directory
- every file system contains a main directory named root or '/.'
- usually, every user is using a current directory attached to the user upon entering the system
  - the user can change the directory: cd
  - create a new directory inside the current directory: mkdir
  - remove a directory: rmdir
  - display access path from root to the current directory: pwd
- the tree structure principle of a file system states that any file or directory has a single parent, implicitly each file or directory and its parent directory will be called *natural link*

- *hard links:*
    - is identical to the natural links and can be created exclusively by the system administrator
    - is an entry in a directory that points to the same substructure to which its natural link is pointing already
    - it makes the substructure appear to have two parent directories
    - basically, a hard link gives a second name to the same file
- *symbolic links:*
    - are special entries in a directory that references some file or directory in the directory structure
    - it behaves as a subdirectory of the directory where the entry was created
    - they can also be created by regular users

UNIX Processes

1. Creation

- a process is created using the fork() system call

2. Fork

- when functioning normally, the effect of calling it is the following:
    - the parent process memory image is copied in a free memory area, this copy being the newly created process, which in the first phase is identical to the initial process
    - the 2 processes continue their execution concurrently with the instruction following the fork() call
    - the newly created process is called child process, and the process that called fork() is called the parent process
    - with the exception of the different address spaces, the child process differs from the parent process only through its PID (process identifier), its parent process (PPID) and the value returned by fork() call
    - when functioning normally, fork() returns in:
        - the parent process: the PID of the new child process
        - the child process: returns 0
- if fork fails, the call fork() will return the value -1

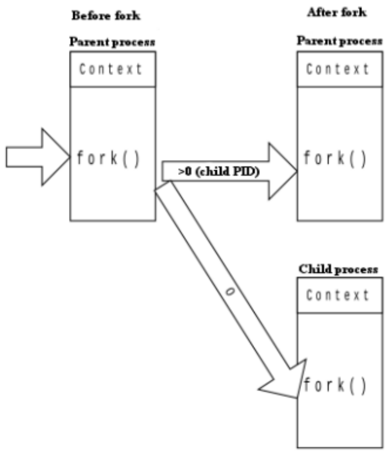


Figure 3.1 The fork mechanism

3. Exec

- almost all OSs and programming environments offer, one way or another, mechanisms for executing a program from inside another program. UNIX offers this mechanism through the exec* system calls
- the exec* system call starts a new program inside the same process
- the exec* system call gets the name of an executable file, and the content of this file overwrites the existing current process
- after calling exec*, the instructions in the current program are not executed any longer, instead, the the instructions of the new program are executed
- UNIX offers six exec* system calls categorized by 3 criteria:
  - the path to the executable: absolute or relative
  - environment inheritance from/ or creation of a new environment for the new program
  - command line arguments specification: specific list or pointer array
- the meaning of the exec parameters is:
  - file
    - the name of the executable file that will replace the current process
    - this name must coincide with argv[0] or arg0
  - argv
    - is an array of pointers, ending in NULL
    - contains the command line arguments for the new program to be executed
  - arg0, arg1, .., argN, NULL
    - the command line arguments of the program about to be executed, given explicitly as strings
    - the last must be NULL as well
  - envp
    - an array of pointers
    - also ending in NULL
    - contains the string corresponding to the new environment variables, given in the format `name=value`

```
int execv (char* file, char* argv[]);
int execl (char* file, char* arg0, …, char* argn, NULL);
int execve(char* file, char* argv[], char* envp[]);
int execle(char* file, char* arg0, …, char* argn, NULL, char*
envp[]);
int execvp(char* file, char* argv[]);
int execlp(char* file, char* arg0, …, char* argn, NULL);
```

4. Exit

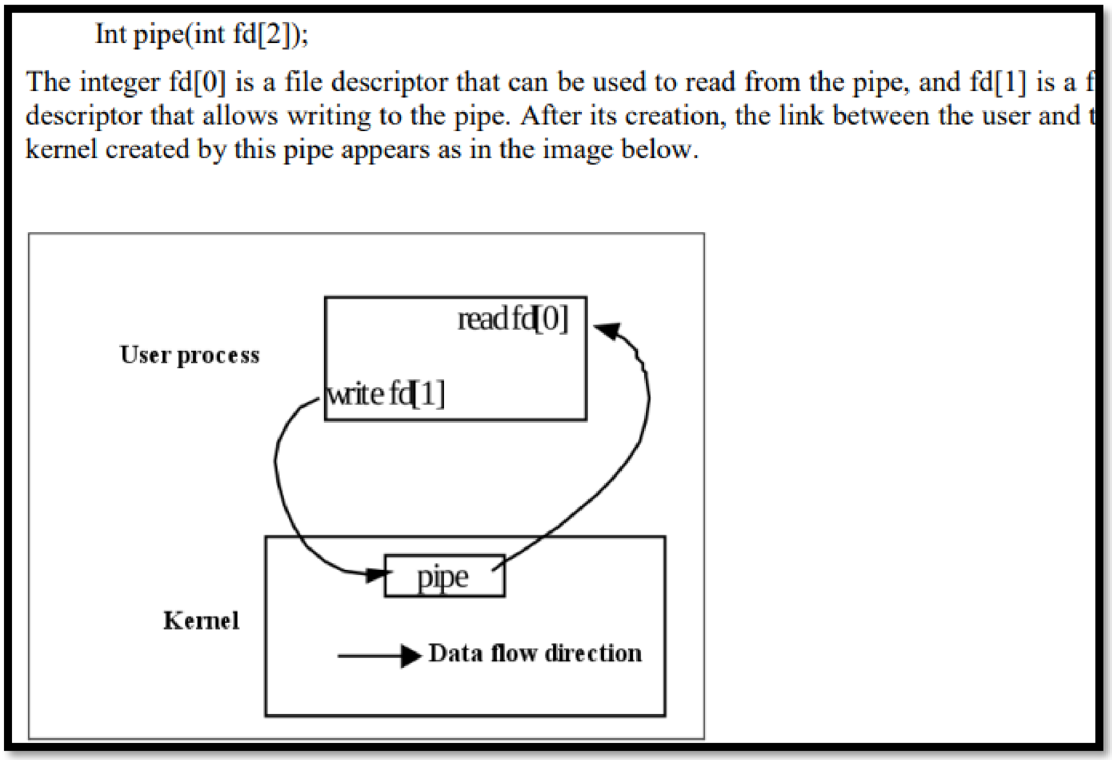
- exit (int n)
    - causes the current process to finish and to return to the parent process (the one that created it using fork)
    - integer n is the exit code of the process
    - if the parent process does not exist any longer, the process is moved to a zombie state and associated with the init process (PID 1)

5. Wait

- the system calls wait(int *status), waitpid
    - are used to wait for the termination of a process
    - the call to wait() suspends the execution of the current process until a child process ends
    - if the child ended before the wait() call, the call ends immediately
    - upon the completion of the wait() call, the child's resources are all freed

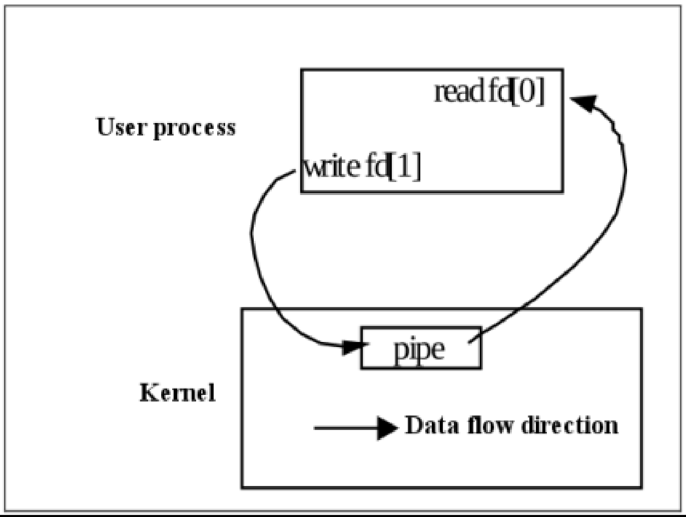
6. Pipe communication

- it is used in order to allow a child process communicate with its parent process
- usually, the parent process redirect its standard output towards the pipe, and the child process redirect its standard input to come from the pipe
- most OSs use the operator '|' to mark this type of connection between the OS commands
- a UNIX pipe is a unidirectional flux of data, managed by the OS kernel. The kernel allocates a buffer of 4096 bytes that are managed as presented below:



Int pipe(int fd[2]);

The integer fd[0] is a file descriptor that can be used to read from the pipe, and fd[1] is a f descriptor that allows writing to the pipe. After its creation, the link between the user and t kernel created by this pipe appears as in the image below.



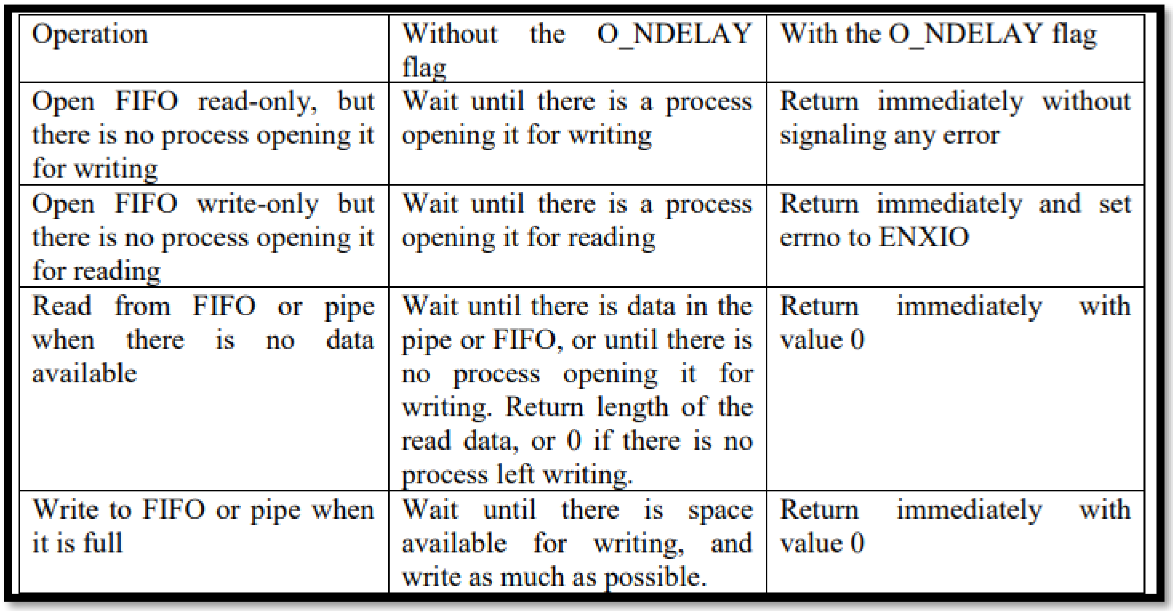
- a PIP within a single process does not make much sense

- it is essential that the pipe and fork work together
- after the pipe is created, fork should be called
- the unidirectional communication through the pipe is left entirely to be insured by the develeoper
- before starting the communication
    - the parent should call close(fd[0])
    - the child should call close(fd[1])

7. FIFO communication

- the main disadvantage of using pipe in UNIX is that it can only be used by related processes
- the processes communicating through the pipe must be descendants of the process creating the pipe
- this is necessary so that the pipe descriptors are inherited by the child process created with fork
- it is a unidirectional flux of data accessed through a file stored on disk in the file system
- the difference between pipe and FIFO is that FIFO has a name and is stored in the file system
- because of that, FIFO can be accessed by any processes, not necessarily having a common parent
- even if the FIFO is stored on disk, no data is stored on disk at all, the data being handled in system kernel buffers
- the essential differences between pipe and FIFO are:
    - the pipe is managed in the RAM memory part managed by the kernel, while for FIFO all of this is done on disk
    - all the processes communicating through pipe must be descendants of the creating process, but for FIFO this is not necessary
- the creation of a FIFO is done using the system calls:
    - int mknod(char* name, int mode, 0);
    - int mkfifo(char* name, int mode);
    - $ mknod name p - shell
    - $ mkfifo name – shell
- the ‘name’ parameter is the name of the file of type FIFO
- the ‘mode’ argument gives the access permissions to the FIFO file
- when using mknod, mode must specify the flag S_IFIFO beside the access permission (<sys.stat.h>
- the last parameter of the mknod system call is ignored, and that is why we put there a 0
- when using the command mknod, the last parameter must be ‘p’ to tell the command to create a named pipe

- to remove a FIFO
    - ‘rm’ – shell
    - ‘unlink’ – C
- once a FIFO is created it must be open for read and write, using the system call “open”



Operation	Without the O_NDELAY flag	With the O_NDELAY flag
Open FIFO read-only, but there is no process opening it for writing	Wait until there is a process opening it for writing	Return immediately without signaling any error
Open FIFO write-only but there is no process opening it for reading	Wait until there is a process opening it for reading	Return immediately and set errno to ENXIO
Read from FIFO or pipe when there is no data available	Wait until there is data in the pipe or FIFO, or until there is no process opening it for writing. Return length of the read data, or 0 if there is no process left writing.	Return immediately with value 0
Write to FIFO or pipe when it is full	Wait until there is space available for writing, and write as much as possible.	Return immediately with value 0

UNIX Shell Programming

A shell interpreter can be regarded in 2 ways:

- Command language that acts as an interface between the user and the UNIX OS (kernel)
    - a shell is started implicitly and acts as a command interpreter
    - the shell displays to the standard out (usually a terminal) offering the user means for running commands
- Programming language that has as base the UNIX commands
    - the primitive element for condition is the exit code of the last command
    - ZERO means true and any other value means FALSE
    - shells have the notions of variable, constant, expression, control structure and subprogram
    - the expressions used by shell are mostly character strings
    - the syntactical aspects were reduced to a minimum

1. Basic concepts:
    a. Variables
        i. a shell variable is a character string to which we assign a value
        ii. a variable is nothing more than a pointer to the actual data
        iii. the Shell enables you to create, assign and delete variables
    b. Control Structures

   i. If/ then/ elif/ else/ fi
   ii. For/ done
   iii. While/ do/ done
   iv. Shift
   v. Break
   vi. Continue

c. Predefined variables
   i. $0
      1. the filename of the current script
   ii. $*
      1. all the arguments are double quoted
      2. if shell receives 2 arguments, $* is equivalent to:
         a. $1, $2
   iii. $@
      1. all the arguments are individually double quoted
   iv. $?
      1. the exit status of the last command executed
   v. $$
      1. the process number of the current shell
      2. the PID under which they are executing
   vi. $!
      1. the process number of the last command

d. I/O Redirections
   i. |
   ii. >
      1. Overwrites an already existing file or a new file is created providing the mentioned file name isn't there in the file name
      2. This means that while making changes in a file you need to overwrite any existing data, use '>' operator
   iii. >>
      1. Appends an already present file or creates a new file if the name doesn't exist in the directory
   iv. <
      1. to be opened for reading
   v. 2>
   vi. 2>>
   vii. 2>&1
      1. Standard output is represented in bash with number 1 and standard error is represented with number 2
      2. The command redirects the standard error to the standard output so they appear together and can be jointly redirected to a file

viii. The/ dev/ null file
  1. It's a special file that's present in every single Linux system
  2. Instead of reading, it is used to write
  3. Whatever you write into the dev/null will be discared into the void 😉

ix. Back-quotes ``
  1. Are used for command expansion
  2. When putting back quotes around some text, the shell replaces the quoted text with the output of running the enclosed command

2. Extended Regular Expressions
   a. POSIX ERE

- when using -E we will not use the '\' anymore for special characters
  - i. "grep -E"
  - ii. "sed -E"

3. Basic commands, functioning and the effect of the specified arguments
   a. cat
      i. short for concatenate
      ii. general syntax: cat [OPTION] [FILE]
      iii. can be used to display the contents of a file: cat file
      iv. view contents of multiple files: cat test test1
      v. create a file with the cat command: cat >test2
      vi. use cat command with more & less options:
         1. cat file.txt | more
         2. cat file.txt | less
      vii. display line numbers in file
         1. cat -n file.txt
      viii. display $ at the end of the file
         1. cat -e file
            a. '$' is added ad the end of lines
      ix. display tab separated lines in file
         1. cat -T test
      x. use standard output with redirection operator: cat test > test1
      xi. appending using redirection operator: cat test >> test1
      xii. cat < test2
         1. it uses file name test 2 as input for the command and output will be shown in a terminal
      xiii. can sort with cat
   b. chmod -R
      i. used to recursively operate on all files and directories under the given directory
      ii. recursively change the file's permissions

c. cp -r
    i. cp is used to copy
    ii. cp -r: copying the directory structure

d. cut
    i. -d
        1. Cut is used to extract specific portion of text in a file
        2. -d by delimiters
    ii. -f
        1. by fields

e. echo
    i. one of the most used commands to print text or string data into the terminal or another command as input or file

f. expr
    i. evaluates a given expression and displays its corresponding output
    ii. it is used for: basic operations like addition, subtraction, multiplication, division, etc or evaluating regular expressions, string operators like substring, length of substrings etc

g. file
    i. it is used to determine the type of a file

h. find
    i. is a command line utility for walking a file hierarchy
    ii. it can be used to find files and directories and perform subsequent operations on them
    iii. -name ‘name’
        1. Searches for the files with the name ‘name’
    iv. -type
        1. Searches for a file with a specific type

i. grep
    i. the grep filter searches a file for a particular pattern of characters and displays all lines that contain that pattern
    ii. the pattern is searched in the file is referred to as a regular expression
    iii. -E
    iv. -i
        1. case insensitive
    v. -q
    vi. -v
        1. Print all the lines that do not match the pattern

j. head -n
    i. as the name implies, it prints the top n numbers of data at the given input
    ii. by default, it prints the first 10 lines of the specified files

k. ls -l
    i. lists directory contents of files and directories
    ii. ls -l displays all information about files and directories
l. mkdir -p
    i. allows user to create or make a new directory
    ii. you can also set permissions with mkdir
m. mv
    i. moves files and folders
n. ps
    i. report a snapshot about the current processes
    ii. -e
        1. View all running processes
    iii. -f
        1. Do full format listing
o. pwd
    i. prints the name of the current working directory
p. read -r
    i. built in command that reads a line from the standard input
    ii. can be used to read line by line
q. rm
    i. it stands for remove
    ii. -f
        1. Force deletion
    iii. -r
        1. Recursive deletion
r. sed
    i. sed command in UNIX stands for stream editor and it can perform lots of functions on file like searching, find and replace, insertion or deletion
    ii. though most common use of SED command in Linux is for substitution or for find and replace
    iii. by using sed you can edit files even without opening them, which is a much quicker way to find and replace something in a file
    iv. -E
    v. Command d
        1. Used to delete
    vi. Command s
        1. Replacing or substituting a string
            a. If at the end you add /2, the second occurrence of that element will be replaced
            b. If ‘g’ is added it is global
            c. If ‘3g’ is added -> all occurrences in the third line.. etc
    vii. Command y

        1. Whenever any character from the left hand side occurs in the input, it replaces it with the corresponding from the right hand side

s. sleep
    i. allows you to suspend the calling process for a specified time
    ii. it is useful when used within a bash shell script, for example, when retrying a failed operation inside a loop

t. sort
    i. -n
        1. Used to sort a file numerically
    ii. -r
        1. Sorting in reverse order

u. tail
    i. -n
        1. Prints the last n number of data of the given input

v. test
    i. is used to test the validity of a command
    ii. checks whether the command/ expression is true or false
    iii. numerical
    iv. string
    v. file operators

w. true
    i. “do nothing, successfully”
    ii. The true command’s sole purpose is to return a successful exit status

x. uniq -c
    i. is used to read a text file by filtering or removing adjacent duplicate lines from the text file
    ii. uniq command is used to detect the adjacent lines from a file and write the content of the file by filtering the duplicate values or write only the duplicate values into another file
    iii. uniq -c: count option

y. wc
    i. wc stands for word count
    ii. it is mainly used for counting purpose
    iii. used to find out the number of lines, word count, byte and characters count in the files specified in the file arguments
    iv. by default it displays four columnar output
    v. -c
        1. Count of bytes present in a file
        2. Two columnar output
        3. 1st column displays the number of bytes present in a file and the 2nd column displays the file name

    vi. -l
        1. is used to print out the length of longest line in a file
    vii. -w
        1. Display the number of word count only of a file

z. who
    i. prints a list of currently logged in users
    ii. it can also show the current run level, time of the last system boot and more

aa. ‘[]?’
    i. the expression between the parentheses can appear at most once

bb. ‘[]+’
    i. the expression between the parentheses can appear at least once

cc. ‘[]*’
    i. the expression between the parentheses can appear zero or more times

dd. ‘[…]’
    i. match any character in …

ee. ‘*’
    i. any sequence of characters

ff. ‘?’
    i. Any character

gg. ‘.’
    i. Matches any single character

hh. ‘^’
    i. Beginning of file

ii. ‘$’
    i. End of file

jj. ‘\<’
    i. Beginning of word

kk. ‘\>’
    i. End of word

ll. ‘+’
    i. Previous expression one or more times

mm. ‘*’
    i. Previous expression zero or more times

nn. ‘?’
    i. Previous expression zero or one times

oo. ‘{m,n}’
    i. Previous expression at least m and at most n times

pp. ‘|’
    i. Logical OR between parts of the regular expression