BINUS UNIVERSITY INTERNATIONAL

Data Structures and Algorithms

Basic Data Types

Week 2

Maria Seraphina Astriani

seraphina@binus.ac.id

BINUS UNIVERSITY INTERNATIONAL

https://1drv.ms/u/s!AkfRgh1gcQzbmyPhuGw5loOJtA23?e=Q3scy9

dsa

BINUS UNIVERSITY INTERNATIONAL

Session Learning Outcomes

Upon completion of this session, students are expected to be able to

- 1. Describe the use of various data structures
- 2. Apply appropriate operations for maintaining common data structures.
- 3. Apply appropriate data structures and simple algorithms for solving computing problems.

Topics

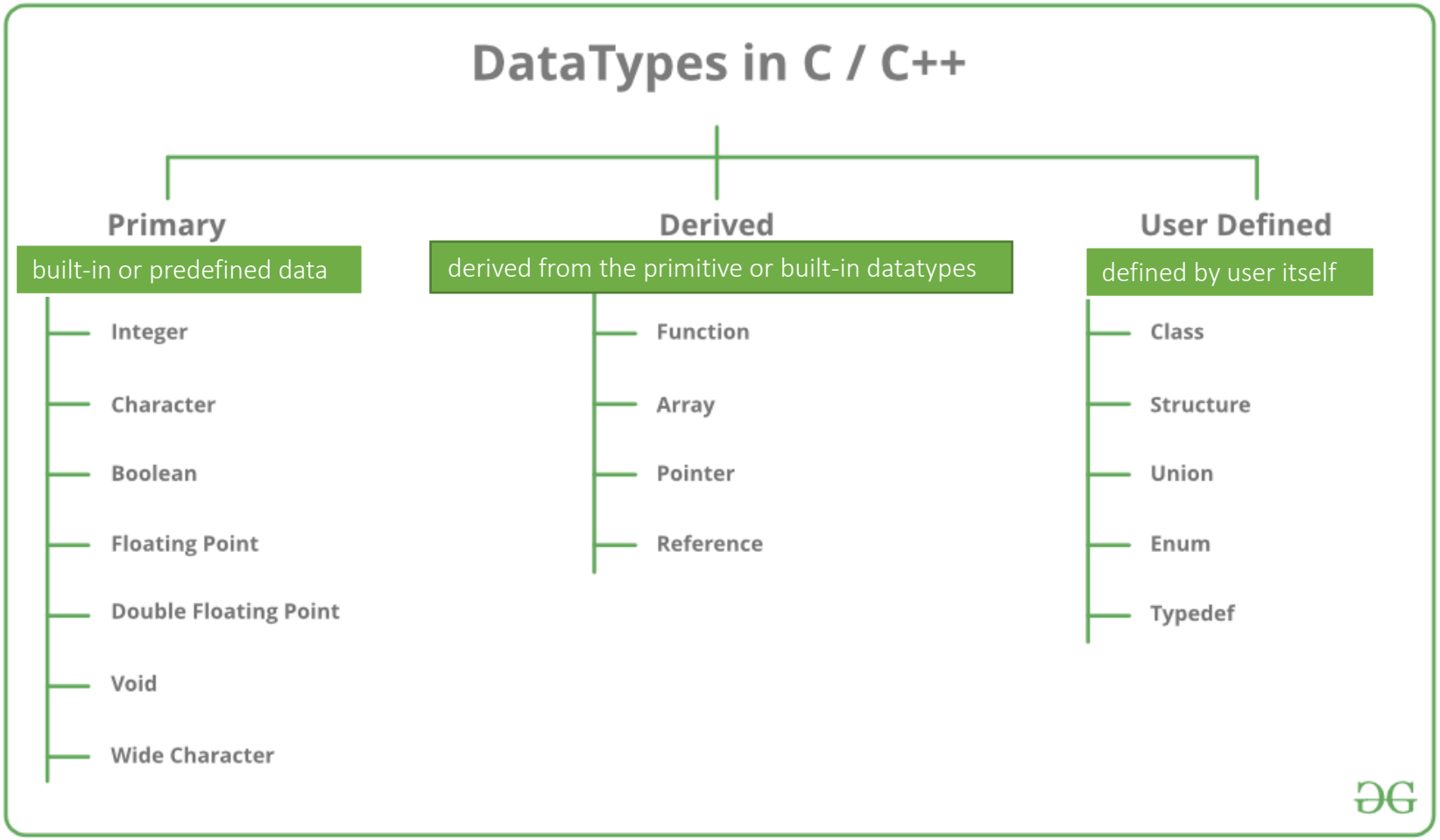
- Basic Data Types
- Object Oriented Design (OOD) and C++

- Session 2: group presentation https://docs.google.com/spreadsheets/d/16CMSU7_3qU8NjYr-1wBx3KMNojSjOtwphqARWofclvM/edit?usp=sharing

Week	Topic(s)	A Class	B Class
2	Basic Data Types	Karsten Eugene Lie, Ellyz Yaory, Darren Pangesa	Edsel,Raffles
4	Stacks and Queues	Jayson M, Bernard W, Nathanael S	Jason Jeremy, Monique Senjaya, Christopher Tendi
5	Trees	Karel Bondan, Leon Jayakusuma, Nathanael Jason L	Ryogassa Avatara, Rendy Divian, Ariel Putra
6	Sets (ADT)	Fadhlan Muhammad Razan, Jevon Danaristo, Rafi Muzakki	Morris Kim, Edward, Brilian Yudha
8	Directed Graphs	Wais Ibrahim, Alifsyah Sutarjadi, Rafian Athallah	
9	Undirected Graphs	Jonathan Wenan, Edward Matthew, Shravan Srinivasan	Raphael Reynaldi, Sri Kalyan Rohan, Ardelia Shaula Araminta
10	Sorting and Searching	Angeline Karen, Vania Natalie, Michael Christopher	Dhifan Fauzan, Ali Abdilahi, Hassan Mohamed
11	Algorithm Analysis Techniques (asymptotic analysis, binary trees, etc.)	Nathaniel Alvin	

BINUS UNIVERSITY INTERNATIONAL

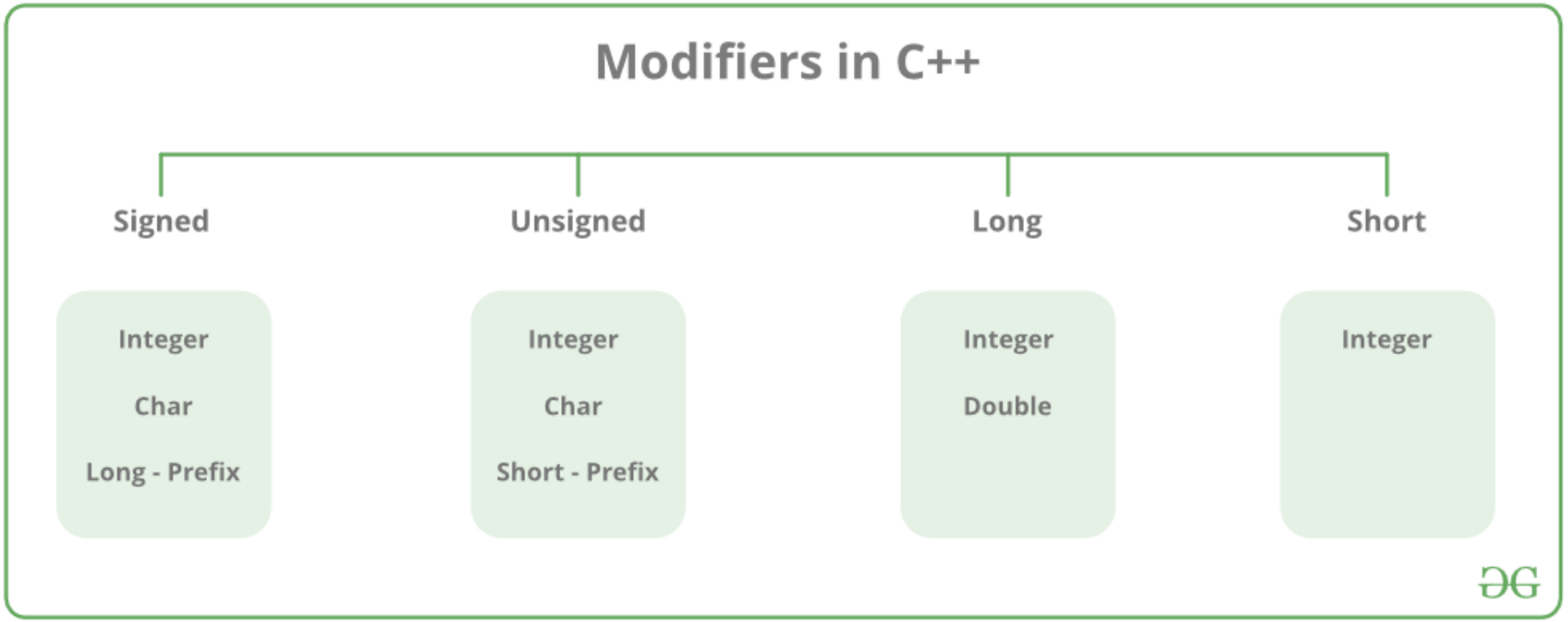
Basic Data Types



Further reading: https://www.geeksforgeeks.org/c-data-types/

Datatype Modifiers

- Datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold
- Example: unsigned int, long integer, etc.



IMPORTANT!!!

- You need to know the range, size (in bytes) to choose which data type is the proper one
  - Integer (4 bytes) : -2147483648 to 2147483647
  - Short integer (2 bytes) : -32768 to 32767
- Values **may vary** from compiler to compiler
- Example:
  If you need to print some values from 0 to 4.000.000.000, you need to choose **unsigned long integer** instead of integer

Further reading: https://www.tutorialspoint.com/cplusplus/cpp_data_types.htm

BINUS UNIVERSITY INTERNATIONAL

Common Data Structures

- Arrays
- Linked Lists
- Stacks (LIFO)
- Queues (FIFO)
- Hash Tables
- Trees
- Heaps
- Graphs

Please answer the following terms
(15 mins)
https://forms.gle/ZW5QBFRkqrbg5bYD9

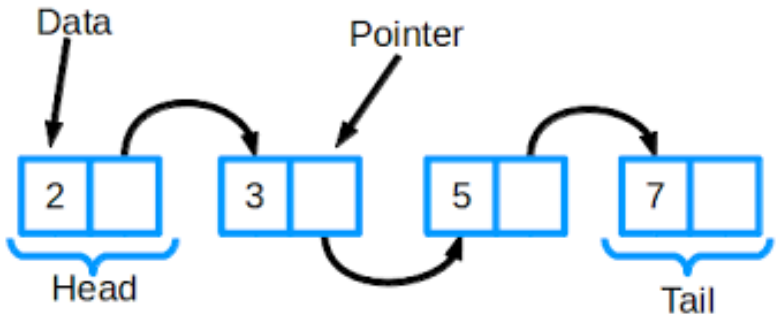
BINUS UNIVERSITY INTERNATIONAL

Limitation of Arrays

- You have already seen how data is organized and processed sequentially using an array.
  - operations on arrays include sorting, inserting, deleting, and searching.
  - if data is not sorted, searching for an item in the list can be very time consuming, especially with large lists. Once the data is sorted, a binary search can improve the search algorithm.
- However, **insertion and deletion are inefficient**, especially with large lists because these operations require data movement (shifting the elements)
- The **array size must be fixed during execution**, new items can be added only if there is room.

What is A Linked List

- A linked list is a collection of components, called nodes.
- Every node (except the last node) contains the address of the next node.
- Every node in a linked list has two components:
  1. **Data**: The data part holds the application data—the data to be processed; and
  2. **One or more links**: Links are used to chain the data together. They contain pointers that identify the next element or elements in the list.
- The address of the first node in the list is stored in a separate location, called the head or first.



BINUS UNIVERSITY INTERNATIONAL

Linked-list vs Array

- The major advantage of the linked list over the array is that data are easily inserted and deleted.
  - It is not necessary to shift elements of a linked list to make room for a new element or to delete an element.
  - On the other hand, because the elements are no longer physically sequenced, we are limited to sequential searches: we cannot use a binary search.

Aspect	Arrays	LinkedList
Memory structure	Contiguous memory	Different memory location joined through pointers
Access	Random access using array subscript	Sequential Access by traversal
Size	Fixed size, may waste memory	Dynamic size, no memory wasting
Insertion and Deletion	Inefficient, shifting elements	Efficient, pointer re-addressing

Implementing the Nodes

- Because each node of a linked list has two components, we need to declare each node as a class or struct.
- **Self-referential structure**
  - The nodes in a linked list are called self-referential structures.
  - In a self-referential structure, each instance of the structure contains one or more **pointers to other instances of the same structural type**.

```
struct Student {
        int id;
        string name;
        double gpa;
        Student* link;
};
```

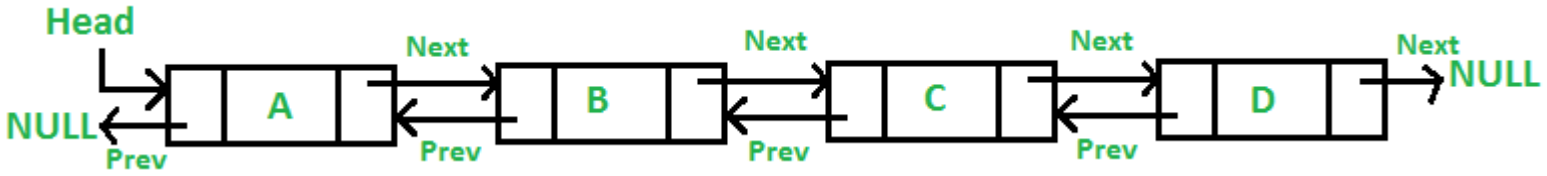
a group of data elements grouped together under one name

```
template <typename T>
struct NODE {
        T data;
        NODE<T>* link;
};
```

Linked List as an ADT

- Linked list as ADT encapsulate the physical data structure from being accessed directly by the client program.
- Templates is used to define a generic of linked lists
- Linked list: A list of items, called nodes, in which the order of the nodes is determined by the address, called the link, stored in each node.
- The basic operations on linked lists are as follows:
  1. Initialize the list.
  2. Determine whether the list is empty.
  3. Print the list.
  4. Find the length of the list.
  5. Destroy the list.
  6. Retrieve the info contained in the first node.
  7. Retrieve the info contained in the last node.
  8. Search the list for a given item.
  9. Insert an item in the list.
  10. Delete an item from the list.
  11. Make a copy of the linked list.

STL Sequence Container: list



- List containers are implemented as **doubly linked lists**.
- Every element in a list points to its immediate predecessor and to its immediate successor (except the first and last elements).
  - To access, for example, the fifth element in the list, we must first traverse the first four elements.
- The name of the class containing the definition of the class list is list.
  - The definition of the class list, and the definitions of the functions to implement the various operations on a list, are contained in the header file list.

```
#include <list>
```

STL List Constructors

`list<elemType> listCont;`	Creates the empty `list` container `listCont`. (The default constructor is invoked.)
`list<elemType> listCont(otherList);`	Creates the `list` container `listCont` and initializes it to the elements of `otherList`. `listCont` and `otherList` are of the same type.
`list<elemType> listCont(size);`	Creates the `list` container `listCont` of size `size`. `listCont` is initialized using the default constructor.
`list<elemType> listCont(n, elem);`	Creates the `list` container `listCont` of size `n`. `listCont` is initialized using `n` copies of the element `elem`.
`list<elemType> listCont(beg, end);`	Creates the `list` container `listCont`. `listCont` is initialized to the elements in the range `[beg, end)`, that is, all the elements in the range `beg...end-1`. Both `beg` and `end` are iterators.

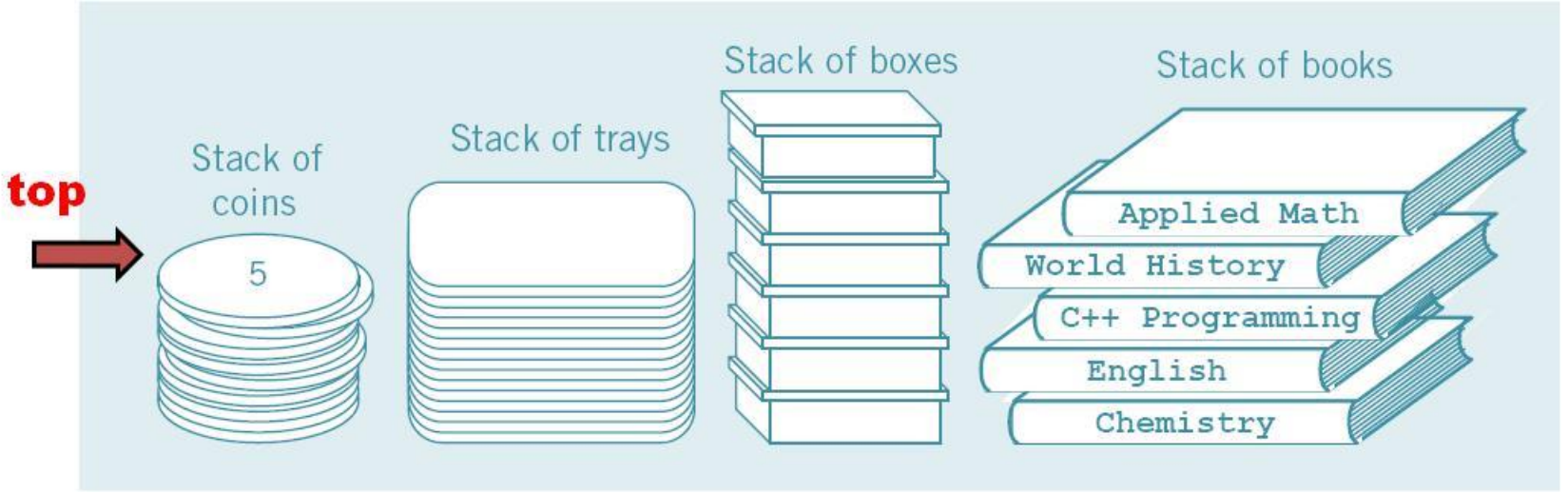
BINUS UNIVERSITY INTERNATIONAL

STL List Operations

`listCont.assign(n, elem)`	Assigns `n` copies of `elem`.
`listCont.assign(beg, end)`	Assigns all the elements in the range `beg...end-1`. Both `beg` and `end` are iterators.
`listCont.push_front(elem)`	Inserts `elem` at the beginning of `listCont`.
`listCont.pop_front()`	Removes the first element from `listCont`.
`listCont.front()`	Returns the first element. (Does not check whether the container is empty.)
`listCont.back()`	Returns the last element. (Does not check whether the container is empty.)
`listCont.remove(elem)`	Removes all the elements that are equal to `elem`.
`listCont.remove_if(oper)`	Removes all the elements for which `oper` is `true`.
`listCont.unique()`	If the consecutive elements in `listCont` have the same value, removes the duplicates.
`listCont.unique(oper)`	If the consecutive elements in `listCont` have the same value, removes the duplicates, for which `oper` is `true`.
`listCont1.splice(pos, listCont2)`	All the elements of `listCont2` are moved to `listCont1` before the position specified by the iterator `pos`. After this operation, `listCont2` is empty.

Stack in daily life

- We use many different types of stacks in our daily lives.
  - We often talk of a stack of coins or a stack of dishes.
  - Any situation in which you can only add or remove an object at the top is a stack.
  - If you want to remove any object other than the one at the top, you must first remove all objects above it

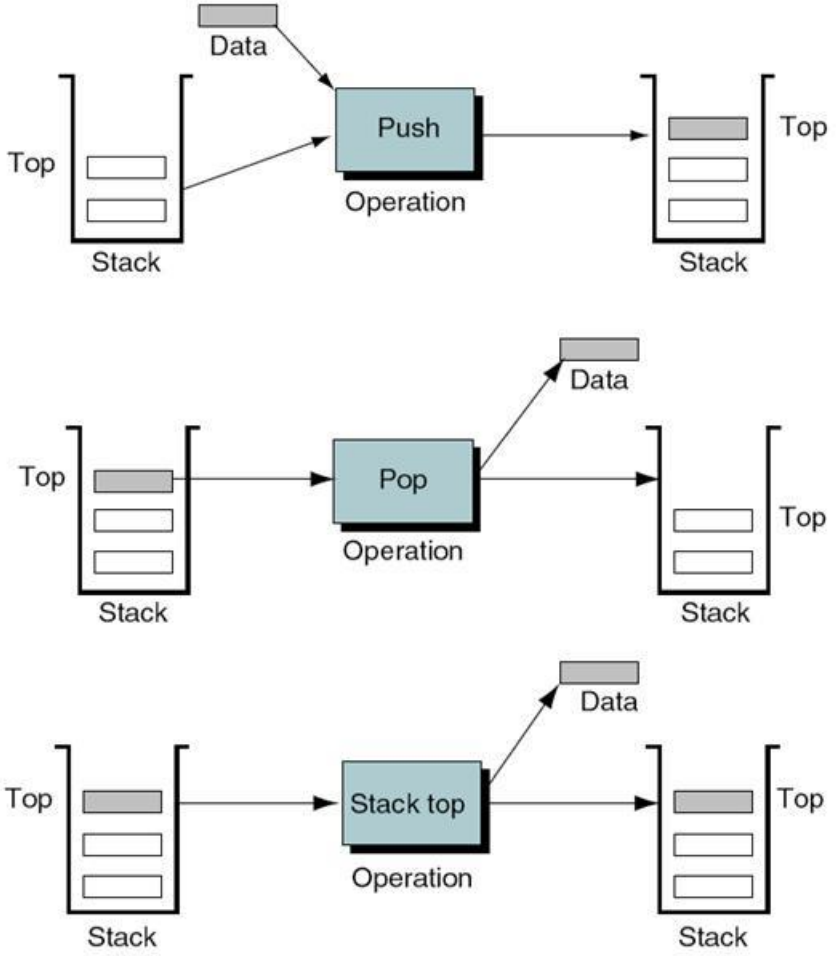


What is Stack?

- Since the elements are added and removed from one end (top), it follows that the item that is added last will be removed first.
- For this reason, a stack is also called a **Last In First Out (LIFO)** data structure.
- Definition
  - **Stack: A data structure in which the elements are added and removed from one end only (top); a Last In First Out (LIFO) data structure**.

Operation on Stack

- **Push**: a new element is **added** to the stack.
- **Pop**: top element is **removed** from the stack.
- **top**: top element is **retrieved** of the stack.

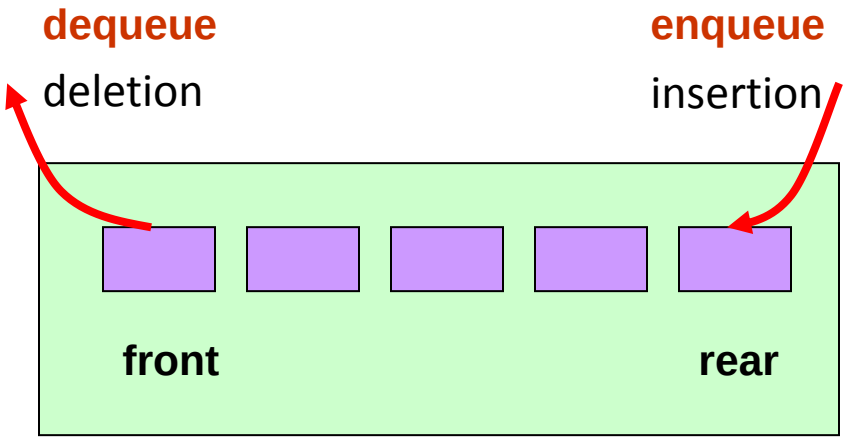


Queue in Everyday Life

- A queue of customers in a bank or in a grocery store
- A queue of cars waiting to pass through a tollbooth
- A queue a computer send a print request to a printed

Queue is FIFO

- A queue is a **First In First Out** (FIFO) data structure.
  - A new element is inserted at the rear of the queue
  - A deletion of an element always occurs at the front of the queue.



Abstract Data Type (ADT)

- An abstract data type is a data declaration packaged together with the operations that are meaningful for the data type.
  - In other words, we encapsulate the data and the operations on the data, and then we hide them from the user.

Queue Major Operations

1. **addQueue or insert (enqueue)**
   - adds a new element to the rear of the queue.

the queue must exist and must not be full.

2. **deleteQueue o remove (dequeue)**
   - removes the front element from the queue.
3. **front**
   - returns the first element of the queue
4. **back (or rear)**
   - returns the last element of the queue

the queue must exist and must not be empty.

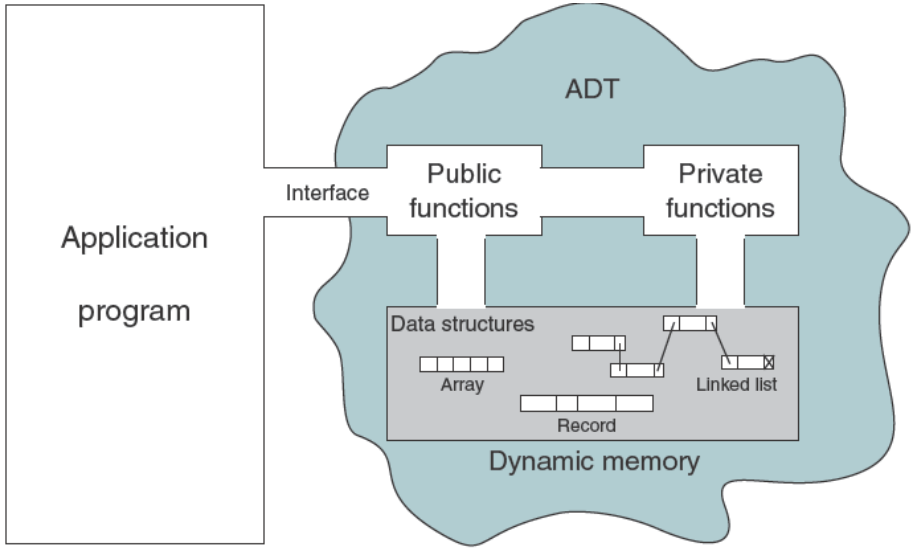
BINUS UNIVERSITY INTERNATIONAL

ADT as a Way of Thinking

- We cannot overemphasize the importance of hiding the implementation.
- The **user should not have to know the data structure to use the ADT**.
  - Referring to our queue example, the **application program should have no knowledge of the data structure**.
  - All references to and manipulation of the data in the queue must be **handled through defined interfaces to the structure**.

Model for an Abstract Data Type

- The colored area with an irregular outline represents the ADT.



- Inside the ADT are two different aspects of the model:
  - **Data structures and functions (public and private). Both are entirely contained in the model and are not within the application program scope.**
  - **However, the data structures are available to all of the ADT's functions as needed, and a function may call on other functions to accomplish its task.**
  - In other words, the data structures and the functions are within scope of each other.

BINUS UNIVERSITY INTERNATIONAL

ADT Operations

- **Data are entered, accessed, modified, and deleted through the external interface** drawn as a passageway partially in and partially out of the ADT.
  - Only the public functions are accessible through this interface.
  - For each ADT operation there is an algorithm that performs its specific task.
  - Only the operation name and its parameters are available to the application, and they provide the only interface to the ADT.

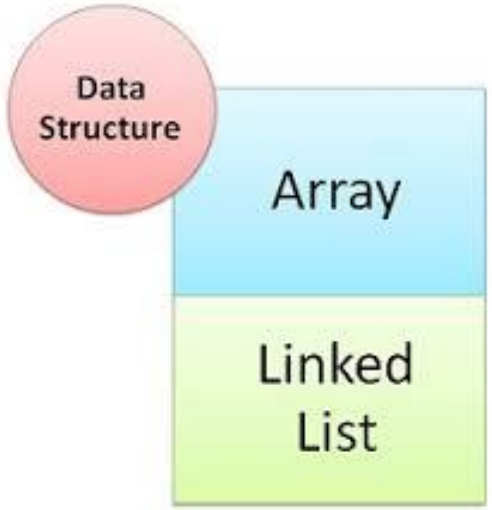
BINUS UNIVERSITY INTERNATIONAL

ADT and Data Structure

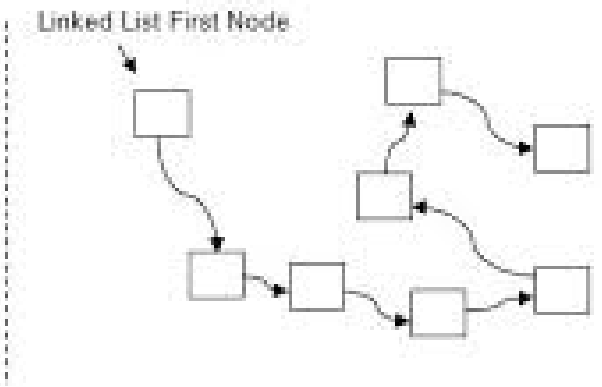
- When a list is **controlled entirely by the program, it is often implemented using simple structures** similar to those used in your (introductory) programming class.
- Because the abstract data type must hide the implementation from the user, however, all data about the structure must be maintained inside the ADT.
  - Just **encapsulating the structure in an ADT** is not sufficient.
  - It is also necessary for **multiple versions of the structure to be able to coexist**. Consequently, we must hide the implementation from the user while being **able to store different data** (TEMPLATE, generic programming).

ADT Implementations

- There are **two basic data structures we can use to implement an ADT** list:
  1. arrays and
  2. linked lists.



physical data structures = storage structure



BINUS UNIVERSITY INTERNATIONAL

Further reading (15 mins)

- ADT VS Concrete Data Type (CDT)

Three kinds of Containers in Math and C++

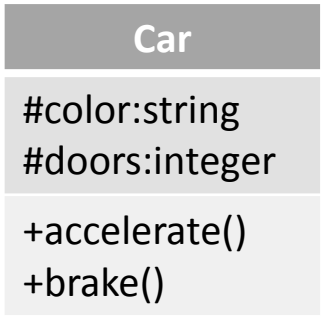
- A **container** is a holder object that stores a collection of other objects (its elements)
- In mathematics there are:
  - **List**: allow duplicate
  - **Set**: distinct elements (does not allow duplicate)
  - **Map**: key-value elements just like a two-dimensional table
- In C++ (Standard Template Library)
  - List is a Sequence container
  - Map is an Associative container
  - Set is unordered Associative container

BINUS
UNIVERSITY
INTERNATIONAL

OOP principles

BINUS UNIVERSITY INTERNATIONAL

OOP



- The core of the pure object-oriented programming is to create an object, in code, that has certain **properties** and **methods**.
  - For example a car is an object which has certain properties such as color, number of doors, etc.
  - It also has certain methods such as accelerate, brake, and so on.
- Most useful aspects of object-oriented programming is **code reusability**

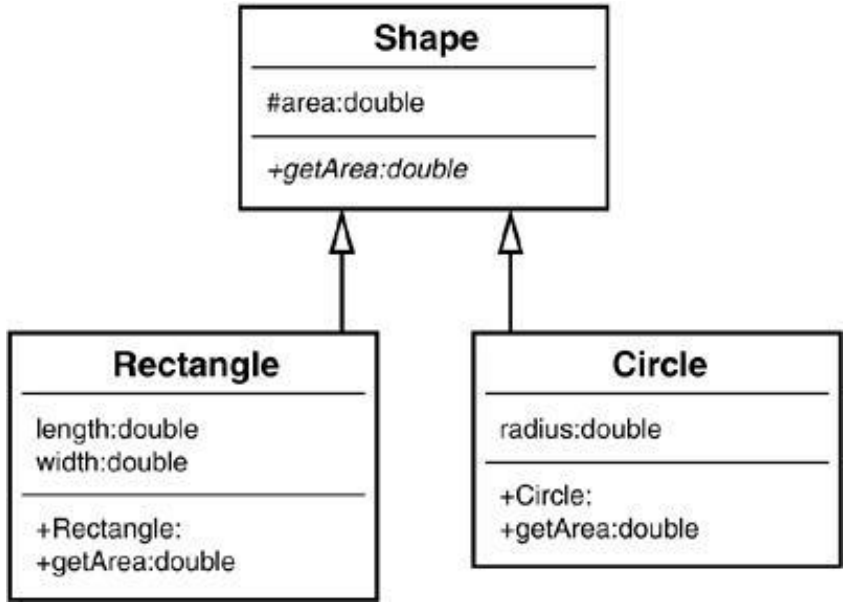
By using classes, you can combine data and operations in a single unit

- Object
  - Basic unit.
  - Data and function that operate on data are bundled as a unit.
- Class
  - “blueprint” for an object.
  - What an object of the class will consist of and what operations can be performed on such an object

https://www.tutorialspoint.com/cplusplus/cpp_object_oriented.htm

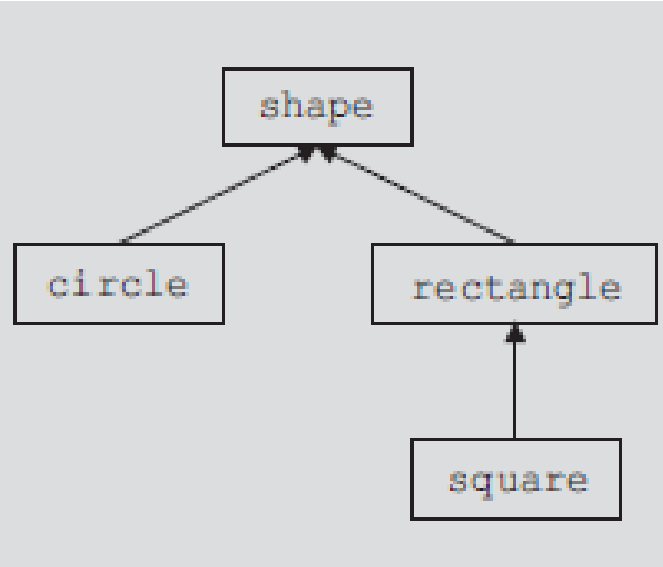
OOP

- Abstraction
  - providing only essential information to the outside world and hiding their background details,
    - i.e., to represent the needed information in program without presenting the details
- Inheritance (***helps to reduce the code size***)
  - The process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.
- Polymorphism
  - Use an operator or function in different ways
  - A single function or an operator functioning in many ways different upon the usage
    - i.e. **Shape** it can be a **rectangle** or **circle**



Inheritance

- Inheritance can be viewed as a treelike or hierarchical, structure wherein a base class in shown with its derived class




```
class className: memberAccessSpecifier baseClassName
{
        member list
};
```

In this diagram, **shape is the base class**. The classes **circle** and **rectangle** are derived from shape, and the class **square** is derived from **rectangle**.

Every circle and every rectangle is a shape.

Every square is a rectangle.

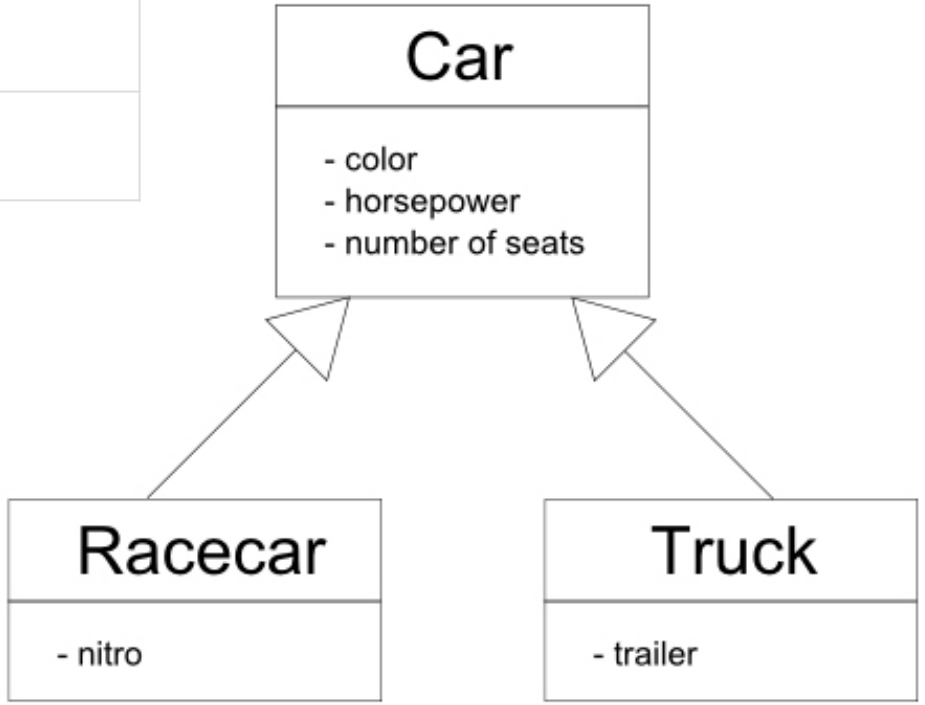
BINUS UNIVERSITY INTERNATIONAL

Type of Inheritance

- Private
  - The private members of a base class are private to the base class;
  - hence, the members of the derived class cannot directly access them.
- Protected
  - The type or member can only be accessed by code in the same class or struct, or in a derived class.
- Public
  - The type or member can be accessed by any other code in the same assembly or another assembly that references it.

Type of Inheritance

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no



https://www.tutorialspoint.com/cplusplus/cpp_inheritance.htm

BINUS UNIVERSITY INTERNATIONAL

Inheritance – C++

- Create a class (base class)
  - setWidth
  - setHeight
  - *print width & height
- Access it through int main()

insertion operator <<
the extraction operator >>

- Create derived class
  - Create the protected type inheritance for width & height in base class to be accessed in derived class
  - Create getArea function to count width & height
- Access derived class through int main()

Polymorphism – C++

- Polymorphism in C++ is a salient feature of object oriented programming that allows the object to perform different tasks

Polymorphism: Operator and Function Overloading

- C++ are used to combine data and operations on that data in a single entity.
- The ability to combine data and operations is called **encapsulation**

- C++ allows the programmer to extend the definitions of most of the operators so that operators such as relational operators, arithmetic operators, insertion operators for data output, and extraction operators for data input can be applied to classes
- In C++ terminology, this is called **operator overloading**.

BINUS UNIVERSITY INTERNATIONAL

Operator Overloading

- In C++, operator is a reserved word
  - i.e.: + , -
- Operator overloading provides the same concise expressions for user-defined data types as it does for built-in data types
- To overload an operator for a class, you do the following:
  - Include the statement to declare the function to overload the operator (that is, the operator function) in the definition of the class.
  - Write the definition of the operator function.

- **Operator function**: The function that overloads an operator.

BINUS UNIVERSITY INTERNATIONAL

Operator overloading provides the programmer with the same concise notation for user-defined data types as the operator has with built-in types. The types of arguments used with an operator determine the action to take.

Overloading an Operator: Some Restrictions

When overloading an operator, keep the following in mind:

- You cannot change the precedence of an operator.
- The associativity cannot be changed. (For example, the associativity of the arithmetic operator + is from left to right and it cannot be changed.)
- You cannot use default arguments with an overloaded operator.
- You cannot change the number of arguments that an operator takes.
- You cannot create new operators. Only existing operators can be overloaded. The operators that cannot be overloaded are

```
.    .*    ::    ?:    sizeof
```

- The meaning of how an operator works with built-in types, such as `int`, remains the same.
- Operators can be overloaded either for objects of the user-defined type, or for a combination of objects of the user-defined type and objects of the built-in type.

BINUS UNIVERSITY INTERNATIONAL

Function Overloading

- C++ allows the programmer to overload a function name

- Overloading a function refers to the creation of several functions with the same name.
- However, if several functions have the same name, every function must have a different set of parameters. The types of parameters determine which function to execute.

Discussion (10 mins)

Inheritance
VS
Polymorphism

BINUS UNIVERSITY INTERNATIONAL

Templates

- Templates are very powerful features of C++.
- By using templates, you can write a single code segment for a set of related functions, called a **function template**, and for related classes, called a **class template**.

- Syntax:

```
template <class Type>
declaration;
```

- **Class Templates** Like function templates, class templates are useful when a class defines something that is independent of data type. Can be useful for classes like LinkedList, BinaryTree, Stack, Queue, Array, etc.

BINUS UNIVERSITY INTERNATIONAL

Group Presentation

Description (what is it?)
Why we need that?
Examples in real-life
How to implement it (code), show the examples in C++
Analyze it (based on Big-O/ processing time/ etc.)
**All team members should be the presenter*

Week	Topic(s)	A Class	B Class
2	Basic Data Types	Karsten Eugene Lie, Ellyz Yaory, Darren Pangesa	Edsel,Raffles
4	Stacks and Queues	Jayson M, Bernard W, Nathanael S	Jason Jeremy, Monique Senjaya, Christopher Tendi
5	Trees	Karel Bondan, Leon Jayakusuma, Nathanael Jason L	Ryogassa Avatara, Rendy Divian, Ariel Putra
6	Sets (ADT)	Fadhlan Muhammad Razan, Jevon Danaristo, Rafi Muzakki	Morris Kim, Edward, Brilian Yudha
8	Directed Graphs	Wais Ibrahim, Alifsyah Sutarjadi, Rafian Athallah	
9	Undirected Graphs	Jonathan Wenan, Edward Matthew, Shravan Srinivasan	Raphael Reynaldi, Sri Kalyan Rohan, Ardelia Shaula Araminta
10	Sorting and Searching	Angeline Karen, Vania Natalie, Michael Christopher	Dhifan Fauzan, Ali Abdilahi, Hassan Mohamed
11	Algorithm Analysis Techniques (asymptotic analysis, binary trees, etc.)	Nathaniel Alvin	

Group Discussion

- Final Project Requirements

- Week 3 - Final Project: Idea – consultation
- Week 7 - Final Project: Proposal - presentation
- Week 13 - Final Project Presentation

BINUS
UNIVERSITY
INTERNATIONAL

Thank you