# C++

## Introduction to class and data abstraction

# Data abstraction

- A data abstraction is a simplified view of an object by specifying what can be done with the object while hiding unnecessary details
- In computer science, the term Abstract Data Type (ADT) is used
- The term interface is also sometimes used
- In OOP, an ADT is implemented as a class
- Example of ADT: a stack
  - It does not matter how the stack is implemented: array, single linked list, double linked list
  - What matters is the operations on the stack: push, pop

# Encapsulation

- **Encapsulation** means preventing access to some piece of information or functionality
- An abstract specification tells us what an object does but not how it does it (information / functionality hiding)
- With encapsulation we can design a program in a way that changing its internal implementation will have no effect on the rest of the program or its users
- Example: a stack can be implemented at first using an array, we can later change this to a single linked list. It should not affect the users of the class Stack.

# Example: stack implemented as an array

```
class Stack {
 private:
  int size;
  int max_size;
  int top;
  int* data;
```

Data members and implementation details
(the stack is implemented by an array)

```
 public:
  Stack (int N);
  ~Stack();

  void push(int el);
  int pop();
  bool is_empty();
  bool is_full();
  int num_elements();
};
```

The public interface:
what the users of the class
and the rest of the program sees

# Example: stack implemented as a linked list

```
class Stack {
 private:
  struct Node {
    int data;
    Node* next;
  };
  int size;
  int max_size;
  Node* top;

 public:
  Stack (int N);
  ~Stack();

  void push(int el);
  int pop();
  bool is_empty();
  bool is_full();
  int num_elements();
};
```

Data members and implementation details
(the stack is implemented by linked list)

The public interface:
what the users of the class
and the rest of the program sees.
It remains unchanged.

# Data member of a class

- Data members of a class can be:
  - Any basic type (int, float, …) or pointer to a basic type
  - Any user defined type or pointer to a user defined type (that has already been **defined** in the program)
- A class name can be used in its own definition but only as a pointer:

```
class Node { public: int data; Node* next; };
```

or

```
struct Node { int data; Node* next; }
```

# Forward declaration

- Forward declaration consists in declaring a class not yet defined such that a pointer (or reference) to that type can be used
- Forward **declaration**:

```
class Node;  // forward declaration
class Stack {
 public:
  Node* top; // ok
  Node  a_node; // compile error
};
```

- Note: as illustrated in the example above, it works for pointers (and references) only

# Initialization of data members

- Data members need to be initialized in constructors or member functions
- For example, the following code is illegal in C++ (but works in Java):

```
class A {
 public:
    int a = 1;  // not ok
};
```

# Initialization of data members

- The following code is correct:

```
class A {
 public:
  int a;
  A() {
    a = 1;
  }
};
```

# Initialization of data members

- Constants in C++ on the other hand can be initialized at definition:

```
class A {
 public:
  static const int a = 1; // legal
};
```

- Notes:
  - It works for integral types only; you can not initialize static const double or static const float this way
  - It can fail with some old C++ compiler

# Member functions

- Member functions (or methods) are used to manipulate objects
- Example: if a class MyString defines a method length() to query the string's length
  - it can be used as:
    MyString str = "some string";
    int len = str.length();
  - Or:
    MyString* str = getPtrToString();
    int len = str->length();

# Member functions

- Methods can be defined inside a class (we call them inline):

```
class A {
  public:
    int a_function(int x) { return x + 1;}
};
```

# Member functions

- Or methods can be declared within a class and defined somewhere else (either in the header file or in an implementation file)
- Example:

```
// within the file A.h
class A { public: int a_function(int x);};
// within the file A.cpp
int A::a_function(int x) { return x + 1;}
```

- Note that when the method is defined outside the class, its name is prefixed by the class name (A:: in the example above)

# Access control

- Access control apply to both member data and member functions
- It allows to control who is able to access a given data or function
- **Public**: means accessible to everybody
- **Protected**: means accessible to member functions and friends of the class and to member functions and friends of the derived classes
- **Private**: means accessible only to member functions and friends of the class
- Trying to access a non-accessible member results in a compile time error

# Example

```
class MyString {
 private:
  char* repr;
  void allocate();
 public:
  int length();
};
```

# Example

- repr and allocate() are accessible by length()
- repr and allocate() would not be accessible by an external function, e.g. main()
- length() can be accessed by any other functions (including methods of other classes(

# Object implementation

- Each object has its own copy of the class data members
  - The exception is the static data members that are shared among instances (we will look at static member later)
- Member functions are shared among the object instances