

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
- 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;
```


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



```
};
```

Data member of a class

- Data members of a class can be:
 - Any basic type (int, float, ...) or pointer to 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**:

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

Data member of a class

- Data members need to be initialized in constructors or member functions
- The following does not work:

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

Member functions

- Member functions are used to manipulate objects
- They can be defined inside the class:

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

- Or outside the class:

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

Access control

- Apply to both member data and member functions
- **Public**: accessible to everybody
- **Protected**: accessible to member functions and friends of the class and to member functions and friends of the derived classes
- **Private**: accessible only to member functions and friends of the class
- Trying to access a non-accessible member results in a compile time error

Object implementation

- Each object has its own copy of the class data member (except for static data member that will be discussed later)
- Member functions are shared among the object instances