DATA STRUCTURE

TITTLE - ABSTRACT DATA TYPE

(ADT)

SUBMITTED TO - Ms. NIDHI BHATIA MA'AM
SUBMITTED By - SHILPI MITTAL

WHAT IS A DATA STRUCTURE?

A data structure is a way of organizing the data so that it can be used efficiently. Here, we have used the word efficiently, which in terms of both the space and time. For example, a stack is an ADT (Abstract data type) which uses either arrays or linked list data structure for the implementation

GROW PLANT AS MUCH AS YOU CAN

ABSTRACT DATA TYPE

In computer science, an abstract data type (ADT) is a mathematical model for data types. An abstract data type is defined by its behavior (semantics) from the point of view of a user, of the data, specifically in terms of possible values, possible operations on data of this type, and the behavior of these operations. This mathematical model contrasts with data structures, which are concrete representations of data, and are the point of view of an implementer, not a user.

ACCESS LABELS ENFORCE ABSTRACTION

In C++, we use access labels to define the abstract interface to the class. A class may contain zero or more access labels –

- Members defined with a public label are accessible to all parts of the program. The data-abstraction view of a type is defined by its public members.

- Members defined with a private label are not accessible to code that uses the class. The private sections hide the implementation from code that uses the type.

BENEFITS OF DATA ABSTRACTION

Data abstraction provides two important advantages –

- Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.
- The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code.

NOTE – By defining data members only in the private section of the class, the class author is free to make changes in the data. If data is public, then any function that directly access the data members of the old representation might be broken.

DATA ABSTRACTION EXAMPLE

Any C++ program where you implement a class with public and private members is an example of data abstraction. Consider the following example –

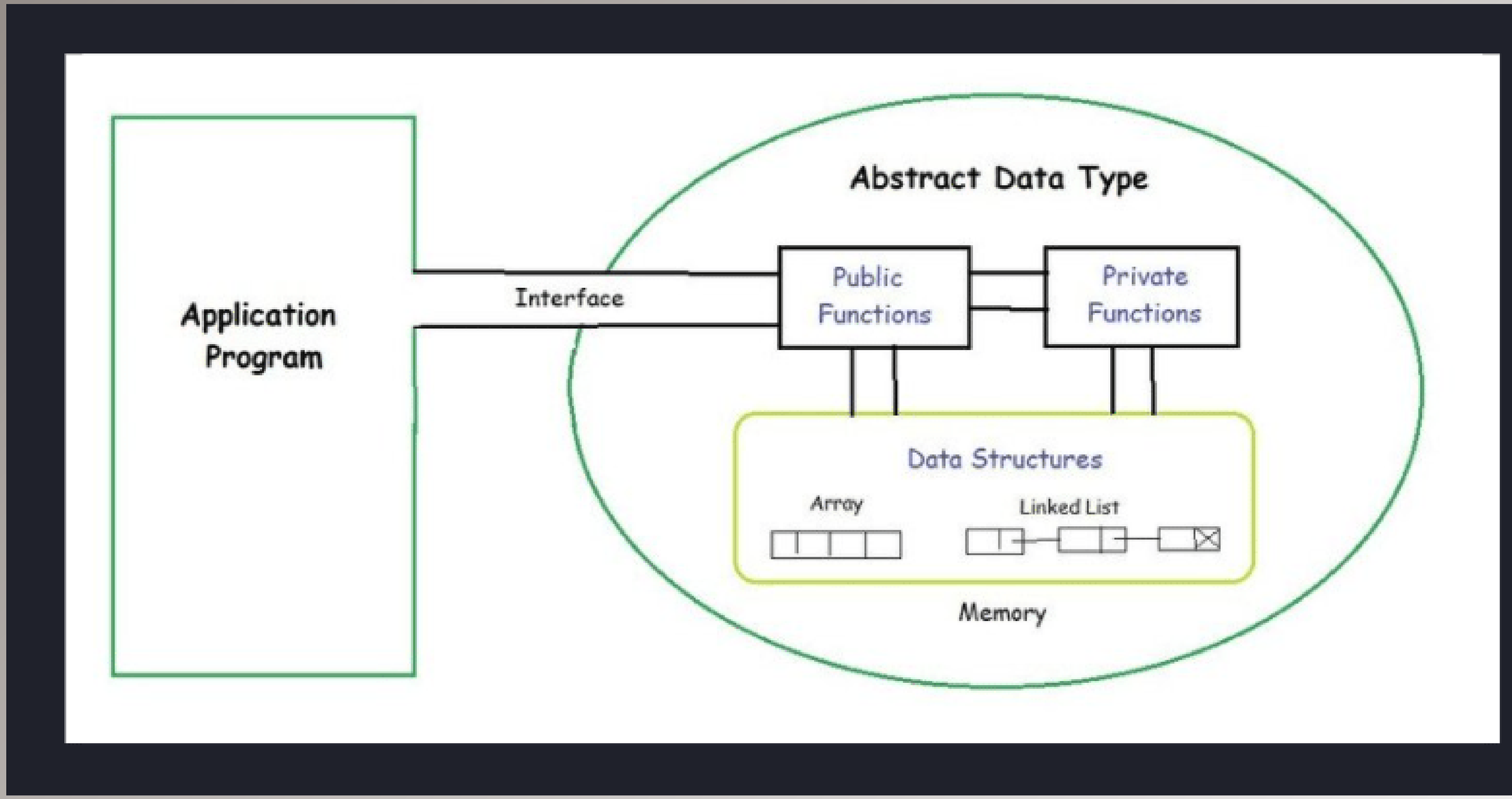
```
#include <iostream>
using namespace std;

class Adder {
   public:
   // constructor
   Adder(int i = 0) {
      total = i;
   }
```

```
// interface to outside world
  void addNum(int number) {
      total += number;
  }

// interface to outside world
  int getTotal() {
      return total;
  };

private:
// hidden data from outside world
  int total;
};
```

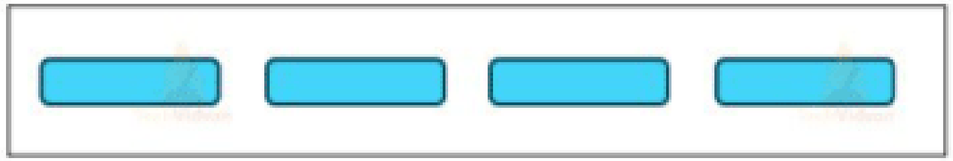
```
int main() {
    Adder a;

    a.addNum(10);
    a.addNum(20);
    a.addNum(30);

    cout << "Total " << a.getTotal() <<endl;
    return 0;
}
```

OUTPUT - **Total 60**

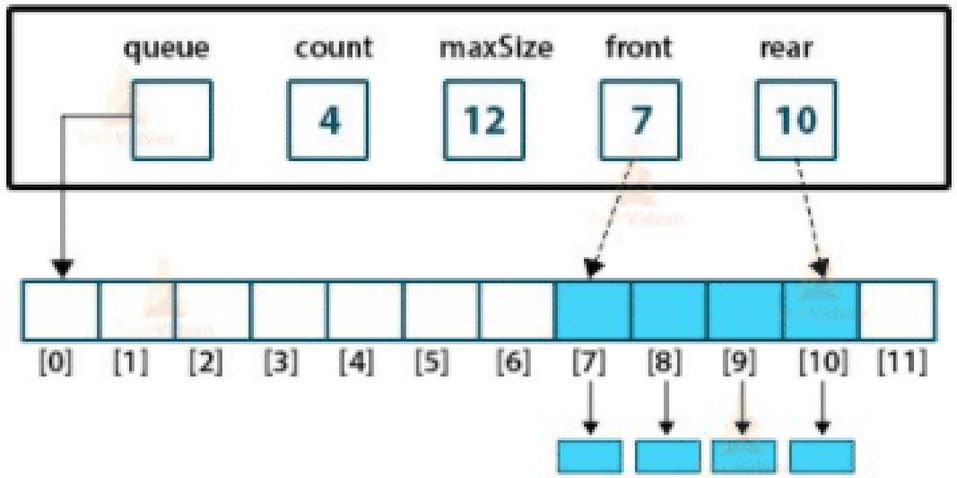
Above class adds numbers together, and returns the sum. The public members - addNum and getTotal are the interfaces to the outside world and a user needs to know them to use the class. The private member total is something that the user doesn't need to know about, but is needed for the class to operate properly.


Abstract Data Type
Application Program
Interface
Public Functions
Private Functions
Data Structures
Array
Linked List
Memory

Structure of Queue Abstract Data Type in Java



a) Conceptual



b) Physical Structures

Examples

Abstraction (ADT)	Implementation (DS)
List	Dynamic Array Linked List
Queue	Linked List based Queue Array based Queue Stack based Queue
Map	Tree Map Hash Map / Hash Table
Vehicle	Golf Cart Bicycle Smart Car

ADVANTAGES OF ABSTRACT DATA TYPING

Encapsulation

Abstraction provides a promise that any implementation of the ADT has certain properties and abilities; knowing these is all that is required to make use of an ADT object.

Localization of change

Code that uses an ADT object will not need to be edited if the implementation of the ADT is changed. Since any changes to the implementation must still comply with the interface, and since code

using an ADT object may only refer to properties and

abilities specified in the interface, changes may be made to the implementation without requiring any changes in code where the ADT is used.

Flexibility

Different implementations of the ADT, having all the same properties and abilities, are equivalent and may be used somewhat interchangeably in code that uses the ADT. This gives a great deal of flexibility when using ADT objects in different situations. For example, different implementations of the ADT may be more efficient

in different situations; it is possible to use each in the situation where they are preferable, thus increasing overall efficiency.

DISADVANTAGES OF ADT

This implementation is unsatisfactory, as it exposes the data representation details of the nat datatype. For example, this implementation of add will not work with a more space efficient binary representation of natural numbers.

BE VACCINATED FOR THE SAFETY OF YOURSELF AND YOUR OWN LOVE ONES


Thank you