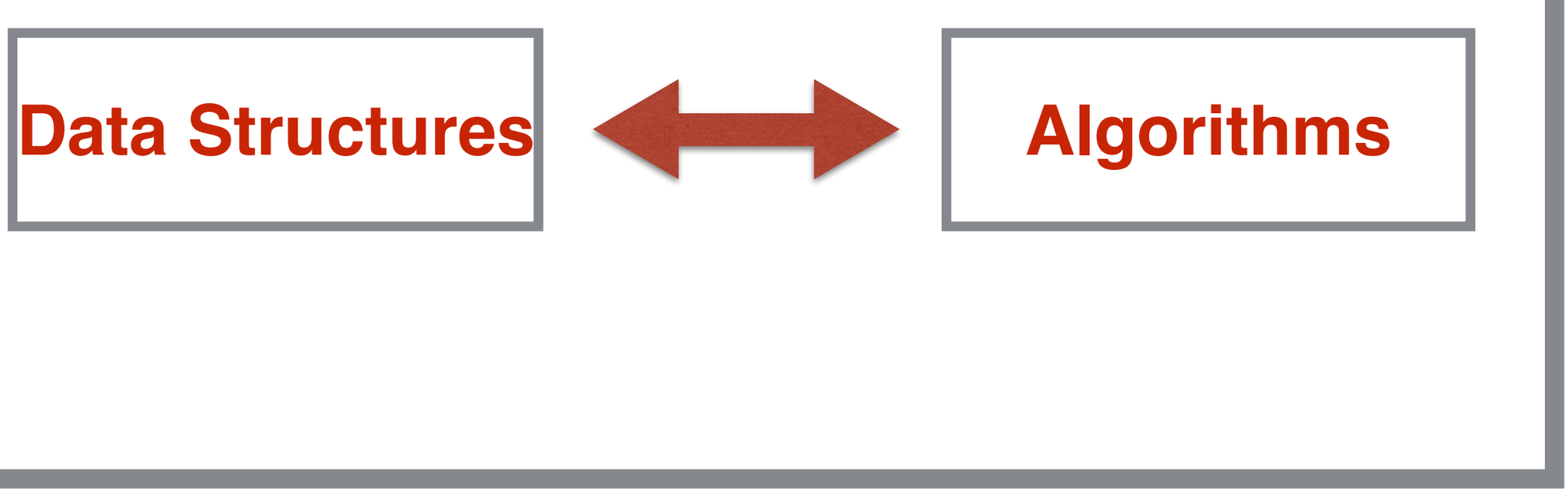
Algorithms and (Abstract) Data Structures.

Programming

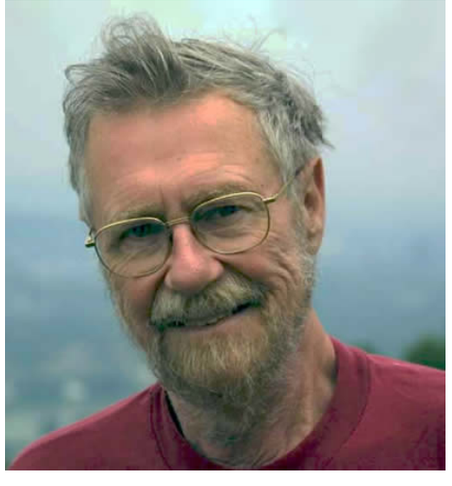


Computer Science or Computing Science?

Computer science is no more about computers than astronomy is about telescopes.

Object-oriented programming is an exceptionally bad idea which could only have originated in California.

Edsger W. Dijkstra (1930-2002)



Beware of bugs in the above code; I have only proved it correct, not tried it.

Donald Knuth (1938-)

Author of : “The Art of Computer Programming”

Barbara Liskov (Turing Award) (1939-)
Abstract Data Structures.

Abstract Data Structures

Abstraction:
Separate the purpose of a program's module from its implementation:

- First phase: the purpose of the code must be specified in details
- Second phase: implementation.

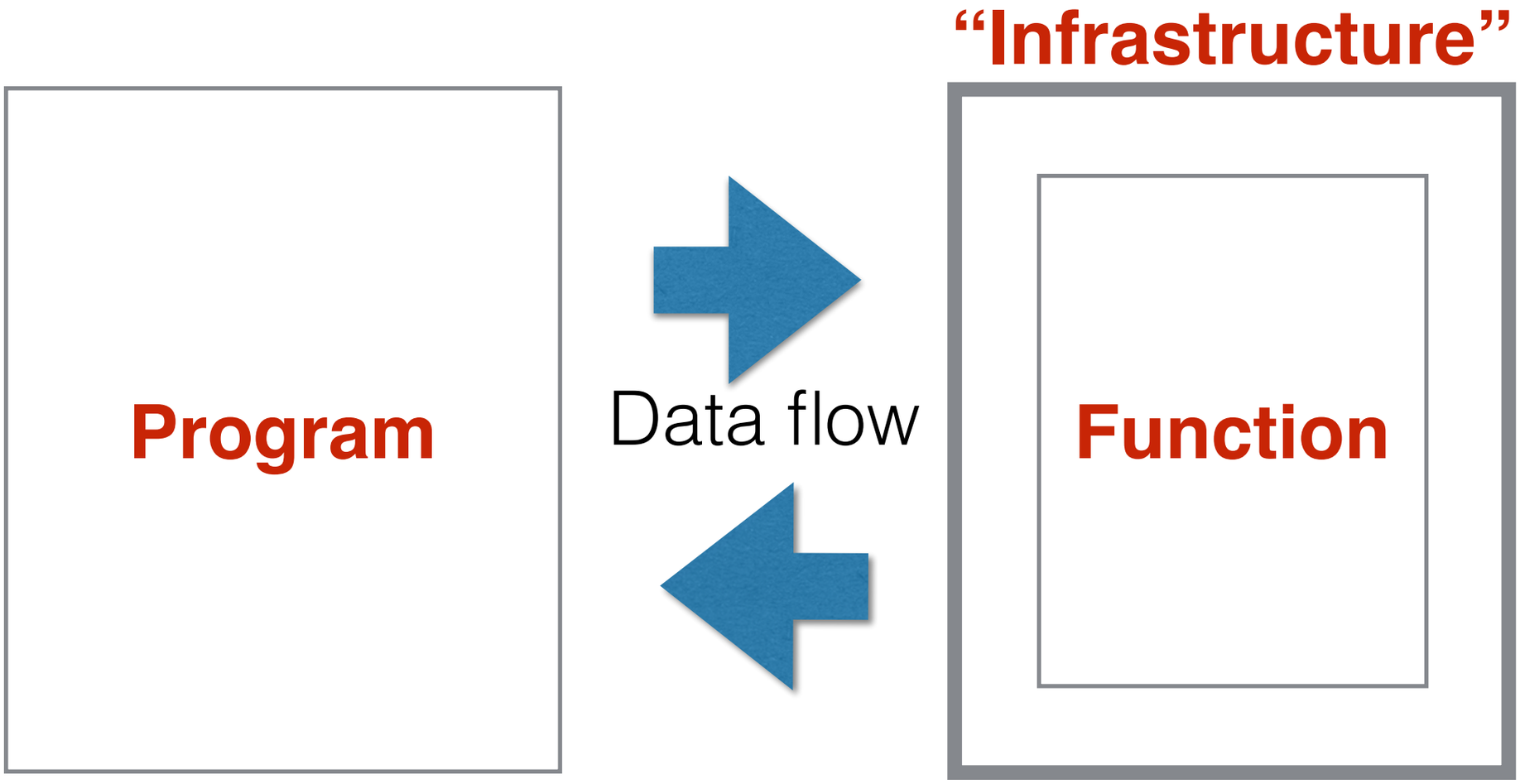
The first phase is conceptual and it does not have anything to do with the details of the programming language or the specific computing machine.

The final code/module must respect the principles of abstraction: operate on certain data in a certain way, hiding the implementation details.

ADVANTAGES:
- Conceptual
- Team work

Abstract Data Structures:
A class of objects whose logical behaviour is defined by a set of values and a set of operations, in analogy (for example) to algebraic structures in mathematics.

Abstract Data Structures



It should be possible to change the function's implementation without changing the rest of the "infrastructure" and the data flow.

A concrete example: the "bag"

A "bag" is a container of a finite number of objects having the same data type and no special order. Duplicates are allowed. A proposed **abstract** interface is:

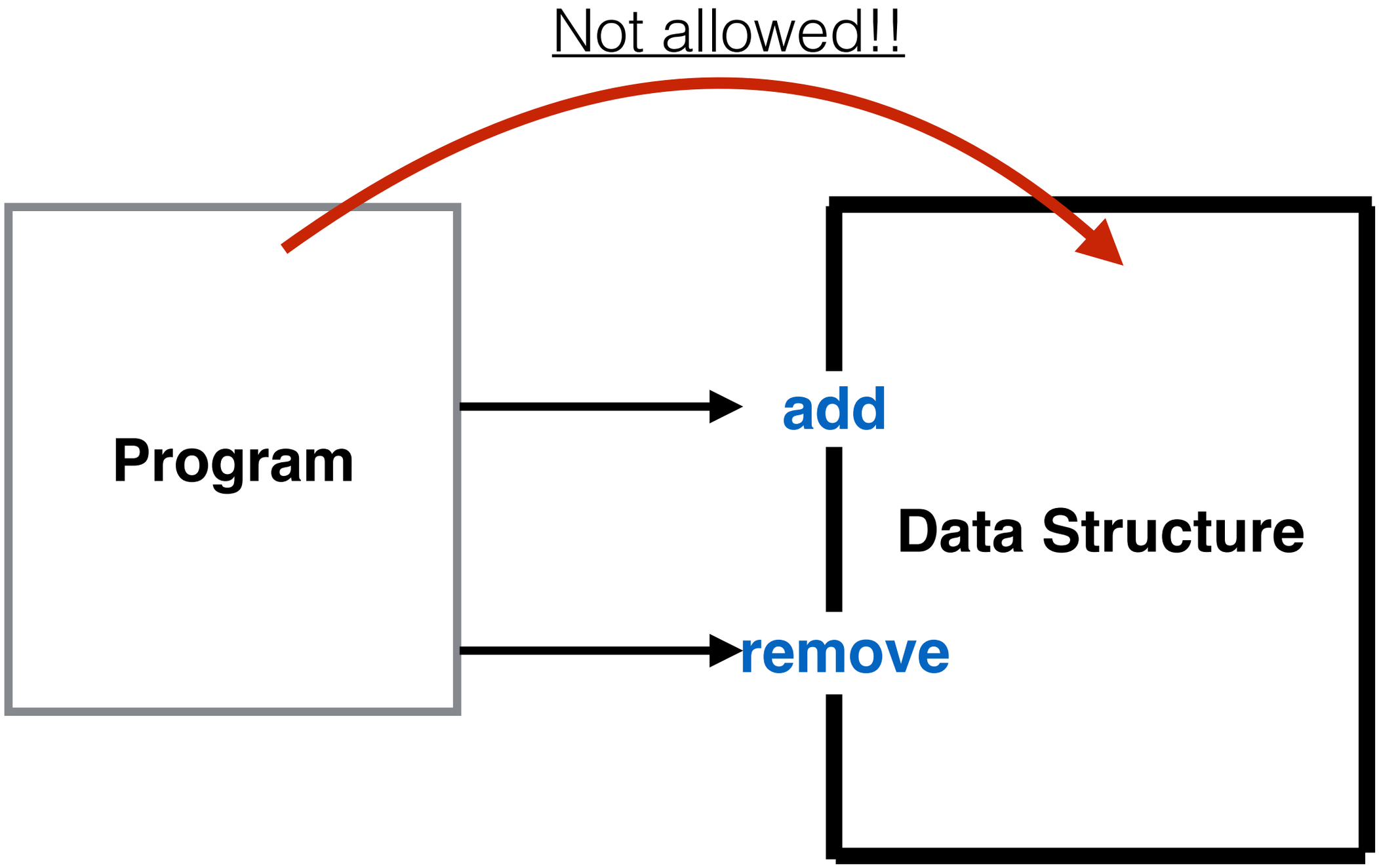
```
#ifndef _BAG_INTERFACE
#define _BAG_INTERFACE

#include <vector>

using namespace std;

template<class ItemType>
class BagInterface
{
 public:
  virtual int getCurrentSize() const = 0;
  virtual bool isEmpty() const = 0;
  virtual bool add(const ItemType& newEntry) = 0;
  virtual bool remove(const ItemType& anEntry) = 0;
  virtual void clear() = 0;
  virtual int getFrequencyOf(const ItemType& anEntry) const = 0;
  virtual bool contains(const ItemType& anEntry) const = 0;
  virtual vector<ItemType> toVector() const = 0;
};

#endif
```



The program must communicate with the data structure only along specified channels.

Implementation

Now that we have an abstract definition of our data structure and the operations on it, we have to choose an implementation. First choice: **ARRAY**.

```
//ArrayBag.h file.
#ifndef _ARRAYBAG
#define _ARRAYBAG

#include "BagInterface.h"

template<class ItemType>
class ArrayBag : public BagInterface<ItemType>
{
 private:
  static const int DEFAULT_CAPACITY = 6;
  ItemType items[DEFAULT_CAPACITY];
  int itemCount;
  int maxItems;
  int getIndexOf(const ItemType& target) const;

 public:
  ArrayBag();
  int getCurrentSize() const;
  bool isEmpty() const;
  bool add(const ItemType& newEntry);
  bool remove(const ItemType& anEntry);
  void clear();
  int getFrequencyOf(const ItemType& anEntry) const;
  vector<ItemType> toVector() const;
};

#include "ArrayBag.cpp"
#endif
```

Implementation (some methods)

```
//Constructor with INITIALIZERS
template<class ItemType>
ArrayBag<ItemType>::ArrayBag() : itemCount(0),maxItems(DEFAULT_CAPACITY)
{
}

template<class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& newEntry)
{
  bool hasRoomToAdd = (itemCount < maxItems);
  if (hasRoomToAdd)
    {
      items[itemCount] = newEntry;
      itemCount++;
    }
  return hasRoomToAdd;
}
```

Implementation (some methods)

```
template<class ItemType>
vector<ItemType> ArrayBag<ItemType>::toVector() const
{
  vector<ItemType> bagContents;
  for (int i=0;i<itemCount;i++) bagContents.push_back(items[i]);
  return bagContents;
}
```

```
template<class ItemType>
int ArrayBag<ItemType>::getCurrentSize() const
{
  return itemCount;
}
```

EXERCISE:

Complete the ArrayBag DataType by yourself and/or the help of the textbook. Finish it at home if needed.

- Complete the abstract class.
- Implement all the remaining methods.
- Test all the methods with a driver program.