ECE2800J

Programming and Introductory Data Structures

Abstract Data Types

Learning Objectives:

Understand what is an abstract data type (ADT)

Understand the usefulness of an ADT

Know how to define an ADT in C++

Last Week

Testing

Exception

Outline

- Introduction to Abstract Data Types
- Class in C++: A Trivial Example
- More Details on Class
- Another Class Example: a Mutable Set of Integers (IntSet)
- Improve the Efficiency of IntSet

Types

- The role of a type:
  - The set of values that can be represented by items of the type
  - The set of operations that can be performed on items of the type.

- Example
  - C++ string values:

    operations:

Struct Types

- Struct types have the following feature:
  - **Every detail** of the type is known to all users of that type.
  - This is sometimes called the **concrete implementation**.

- Example: the `struct Grades` talked before.

```
struct Grades {
   char name[9];
   int  midterm;
   int  final;
};
```

Struct Types

```
struct Grades {
   char name[9];
   int  midterm;
   int  final;
};
```

- Every function knows the details of exactly how `Grades` are represented.
- A change to the `Grades` definition (for example, change C-string for `name` to a C++-String) requires that we **make changes throughout the program** and recompile everything using this struct.

Abstract Data Types

Introduction

- Contrast the property of struct types with that of the functions
  - A function written by others shows **what** the function does, but not **how** it does it

- For function, if we find a faster way to implement, we can just replace the old implementation with the new one
  - No other components of the program calling the function need to change

Abstract Data Types

Introduction

- To solve the problem for struct type, we'll define **abstract data types**, or **ADTs**.
- An ADT provides an **abstract description** of **values** and **operations**.
- The definition of an ADT must combine **both** some notion of **what values** that type represents, and **what operations** on values it supports.
  - However, we can leave off the details of **how**.
- Example: mobile phone
  - Type: a portable telephone that can make and receive calls
  - Operations: turn on/off, make/receive call, text message

We don't know details!

Abstract Data Types

Introduction

- Abstract data types provide the following two advantages:

1. Information hiding: we don't need to know the details of how the **object** is **represented**, nor do we need to know how the **operations on those objects** are **implemented**.
2. Encapsulation: the objects and their operations are defined in the same place; the ADT combines both data and operation in one entity.

Abstract Data Types

Example

- `list_t`:
  - Information Hiding: In the `list_t` data type, you never knew the precise implementation of the `list_t` structure (except by looking in `recursive.cpp`).
  - Encapsulation: The definitions of the operations on lists (`list_print`, `list_make`, etc.) were found in the same header file as the type definition of `list_t`.

Abstract Data Types

Benefits

- Abstract data types have several benefits like we had with functional abstraction:
  - ADTs are **local**: the implementation of other components of the program does not depend on the **implementation** of ADT.
    - To realize other components, you only need to focus **locally**.
  - ADTs are **substitutable**: you can change the implementation and no users of that type can tell.

Which Statements Are Correct?

Select all the correct answers.

- **A.** If the implementation of a function changes, all places that uses this function need to be modified.
- **B.** A type is a set of values plus a set of operations on these values.
- **C.** If the implementation of an ADT changes, all places that uses this ADT need to be modified.
- **D.** If I remove an operation of an ADT, there is no need to check other places that use this ADT.

Abstract Data Types

Introduction

- Someone still needs to know/access the details of how the type is implemented.
  - I.e., how the values are represented and how the operations are implemented
  - This is referred to as the "**concrete representation**" or just the "**representation**"

- Question: Who can access the representation?
- Answer: **only** the operations defined for that type should have access to the representation.
  - Everyone else may access/modify this state only **through** operations.

Abstract Data Types

On to Classes

- C++ "class" provides a mechanism to give **true** encapsulation.

- The basic idea behind a class is to provide **a single entity** that both defines:
  - The **value** of an object.
  - The **operations** available on that object. These operations are sometimes also called **member functions** or **methods**.

Outline

- Introduction to Abstract Data Types
- Class in C++: A Trivial Example
- More Details on Class
- Another Class Example: a Mutable Set of Integers (IntSet)
- Improve the Efficiency of IntSet

Abstract Data Types

Classes – A trivial example

```
class anInt {
    // OVERVIEW: a trivial class to get/set a
    //           single integer value
    int    v;
public:
    int    get_value();
           // EFFECTS: returns the current
           //          value
    void   set_value(int newValue);
           // MODIFIES: this
           // EFFECTS: sets the current value
           // equal to newValue
};
```

Abstract Data Types

Classes – A trivial example

```
class anInt {
    // OVERVIEW: a trivial class to get/set a
    //           single integer value
    int     v;
  public:
    int     get_value();
            // EFFECTS: returns the current
            //          value
    void    set_value(int newValue);
            // RME: Omitted for space
};
```

- There are a few things to notice about this definition:
  - There is a single OVERVIEW specification that describes the class as a whole.

Abstract Data Types

Classes – A trivial example

```
class anInt {
    // OVERVIEW: Omitted for space
    int     v;
  public:
    int     get_value();
        // EFFECTS: returns the current value
    void    set_value(int newValue);
        // RME: Omitted for space
};
```

- There are a few things to notice about this definition:
  - The definition includes both data elements (`int v`) and member functions/methods (`get_value` and `set_value`).

Abstract Data Types

Classes – A trivial example

```
class anInt {
    // OVERVIEW: Omitted for space
    int     v;
  public:
    int     get_value();
            // EFFECTS: returns the current
            //          value
    void    set_value(int newValue);
            // MODIFIES: this
            // EFFECTS: sets the current value
            // equal to arg
};
```

- There are a few things to notice about this definition:
  - Each function that is declared must have a corresponding specification.

Abstract Data Types

Classes – A trivial example

```
class anInt {
    // OVERVIEW: Omitted for space
    int    v;
  public:
    int    get_value();
           // EFFECTS: returns the current value
    void   set_value(int newValue);
           // MODIFIES: this
           // EFFECTS: sets the current value
           // equal to arg
};
```

- There are a few things to notice about this definition:
  - `set_value` says it MODIFIES `this`. This is the generic name for "**this object**".

Outline

- Introduction to Abstract Data Types
- Class in C++: A Trivial Example
- More Details on Class
- Another Class Example: a Mutable Set of Integers (IntSet)
- Improve the Efficiency of IntSet

Abstract Data Types

Classes – More Details

- By default, every member of a class is **`private`**.
  - Members = data members + function members
- A private member is visible **only** to **other members** of this class.
  - `int v` was a private member in the class `anInt`.
  - "Private" hides the implementation of the type from the user.

Abstract Data Types

Classes – More Details

- However, if everything were private, the class wouldn't be particularly useful!
- So, the **`public`** keyword is used to signify that some members are **visible** to anyone who sees the class definition, not only visible to other members of this class.
  - Everything **after** the **`public`** keyword is **visible** to others.

Abstract Data Types

Classes – A trivial example

```
class anInt {
    // OVERVIEW: a trivial class to get/set a
    //           single integer value
    int     v;
  public:
    int     get_value();
            // EFFECTS: returns the current
            //          value
    void    set_value(int newValue);
            // MODIFIES: this
            // EFFECTS: sets the current value
            // equal to arg
};
```

Abstract Data Types

Classes – A trivial example

This definition, as it is, is incomplete. We have not yet defined the bodies of the member functions.

```
class anInt {
    // OVERVIEW: a trivial class to get/set a
    //           single integer value
    int    v;
  public:
    int    get_value();
           // EFFECTS: returns the current
           //          value
    void   set_value(int newValue);
           // MODIFIES: this
           // EFFECTS: sets the current value
           // equal to arg
};
```

Abstract Data Types

Classes – Defining member functions

```
class anInt {
    // OVERVIEW: a trivial class to get/set a
    //           single integer value
```

Note: You can actually define the functions within the class definition, but this "exposes" information, which is best left hidden!

```
int anInt::get_value() {
  return v;
}
void anInt::set_value(int newValue) {
  v = newValue;
}
```

The definitions of member functions are usually put in the .cpp file;
You should include .h in the .cpp!

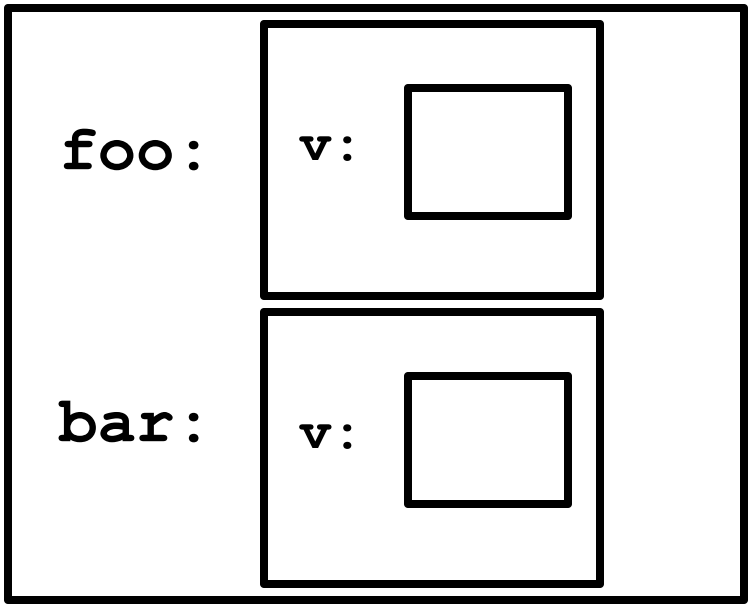
Abstract Data Types

Classes – Declaring class objects

- We can declare **objects** of type `anInt` as you would expect:

```
anInt     foo;
anInt     bar;
```

- This produces an environment with two objects:



These values are still undefined (i.e. there is no initial value). We'll see several ways to set an initial value for data members later.

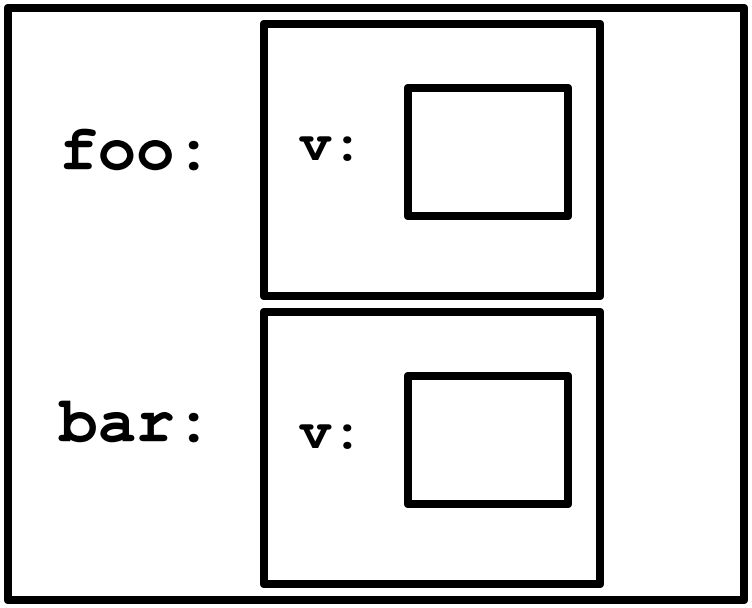
Abstract Data Types

Classes – Establishing data member values

- We can call the set_value member function to establish a value:

```
foo.set_value(1);
```

This calls foo's set_value() method.



Abstract Data Types

Classes – Establishing data member values

- There is one very important difference between normal function calls and **member** function calls:
  - The **other** members of the object are **also visible** to the function members!
  - For example, `v` is visible to the function `set_value()`

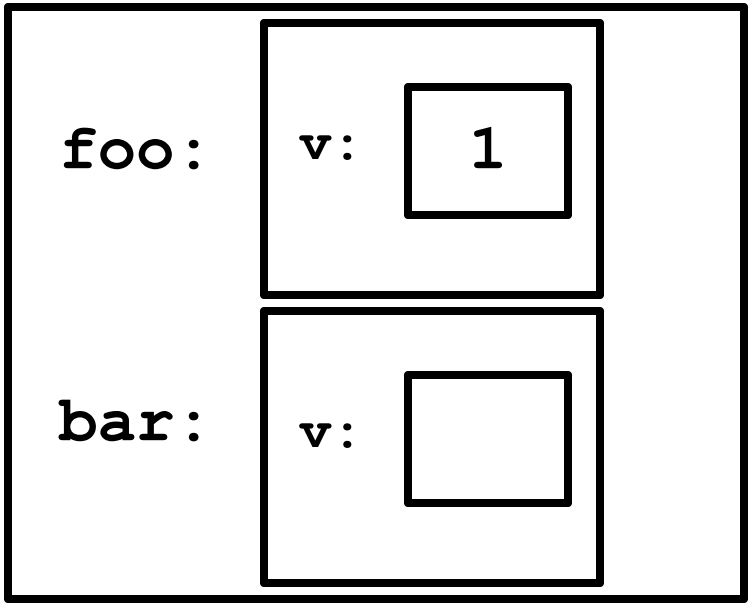
```
void anInt::set_value(int newValue) {
   v = newValue;
}
```

Abstract Data Types

Classes – Establishing data member values

- So, set value changes **foo**'s v:

```
foo.set_value(1);
```



Abstract Data Types

Classes – Accessing data member values

- We can't access v directly:

```
cout << foo.v; // Compile-time error
```

because v is **private**!

- However, we can use the get_value() method to do so for us:

```
cout << foo.get_value(); // OK.
```

because `get_value()` is **public**!

- Finally, class objects can be passed just like anything else.
- Like everything else (except arrays), they are passed by value.

Abstract Data Types

Class Example: Classes

- What is the result of the following?

```
void add_one(anInt i) {
  i.set_value(i.get_value()+1);
}
int main() {
  anInt foo;
  foo.set_value(0);
  add_one(foo);
  cout << foo.get_value() << endl;
  return 0;
}
```

Abstract Data Types

Classes – Passing by reference

- To pass a class object by reference, you use either a pointer argument or a reference argument, i.e.:

```
void add_one(anInt *ip) {
  ip->set_value(ip->get_value() + 1);
}
```

- This version would change the class object passed to it!

Which Statements Are Correct?

- **A.** A C++ class can define a type.
- **B.** The information stored in an object of a class is accessible to any one.
- **C.** A class defines some basic operations that are possible on objects of that class.
- **D.** All member functions of a class are accessible to any one.

Exercise: Interval Class

- A closed interval [a,b] represents a set of numbers.
- Basic operations are:
  - set a,b
  - get a, get b
  - check if an interval overlaps with another interval
  - compute the intersection of two intervals
  - compute the union of two intersected intervals
- Create a header file, a source file, and a main function that uses this interval class.

References

- **Problem Solving with C++ (8th Edition)**
  - Chapter 10.3 Abstract Data Types
  - Chapter 10.2 Classes