CSC215

Math and Computer Science

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

ADT's (Abstract Data Type) – Objects

- A collection of data values together with a set of well-specified operations on that data.
- You have been using ADTs for some time.
  - ifstream & ofstreams
  - istream & ostreams
  - vectors & strings
- Once you create an instance, you have data and functions that operate on the data.
  - Ie. Different instances of strings contain different data but both have the same functions to manipulate the data.

Why Bother with ADTs?

- The Major Benefit is Information Hiding.
  - The outside world (programmer using your object) knows the interface but not the implementation details.
  - As long as the interface doesn't change, you can modify the implementation and not break anyone's code.

Data Abstraction vs Functional Abstraction

- The two abstractions are very closely related.
- Both ignore how we will implement the tasks.
- Data abstraction – is a design principle that separates the operations that can be performed on a collection of data from the implementation of the operations.
- Function Abstraction – is a design principle that separates the purpose and use of a module from its implementation.

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

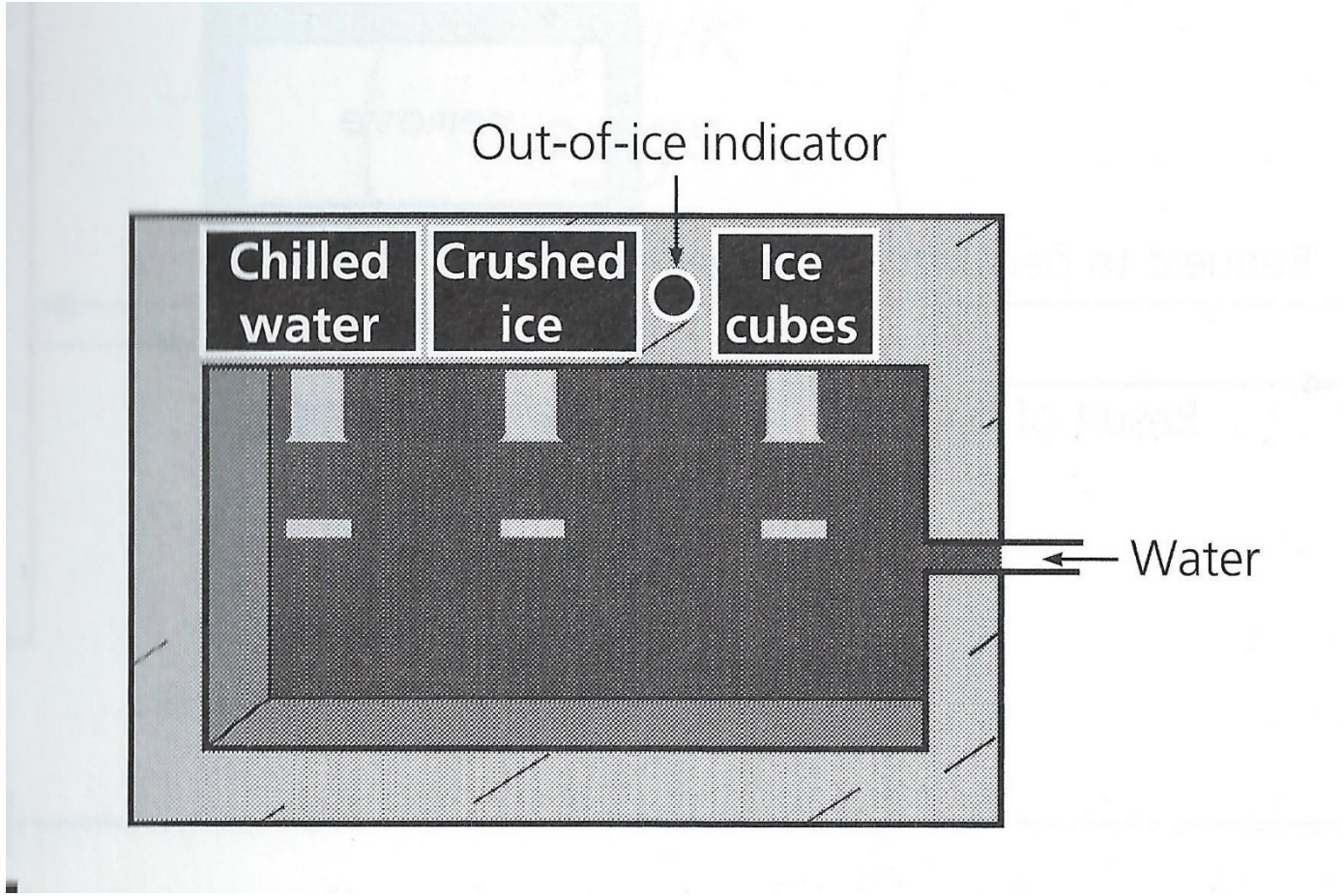
Data Abstraction Example – for a list

- Add data
- Remove data
- Search for data
- Count how many data entries are in the list

Functional Abstraction – for a list

- Add data
  - Given an insertion point, would place the new data between the old data at the given location.
- Remove data
  - Given the value of an item, would go through the data removing all instances from the collection.
- Search
  - Given a value of an item, would go through all the data looking for an occurrence of the requested item. Would return true if found, false if not found.

Example – An Ice Maker



SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

The Interface

- You have a button for crushed ice
- You have a button for ice cubes
- You have a button for chilled water
- You have a display light for out of ice

- How does it create crushed ice? Do we care?
- How does it create ice cubes? Do we Care?
- How does it chill the water? Do we Care?

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Changing the Implementation

- Crushed Ice Button
  - Maybe an ice cube is forced between two rollers in order to create the crushed ice.
  - That implementation could be then replaced with two medal plates that squash an ice cube until it shatters.

- Do you care which you have? You push a button and get crushed ice still.

Example – a circle

- Stay away from implementation
- What can you do with a circle

- Create a circle
- Destroy a circle
- Set the radius

- Get the radius
- Get the diameter
- Get the area
- Get the circumference

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Example – Circle

- Create function
  - In: nothing
  - Out: circle with radius of 1 unit
- Destroy function
  - In: Nothing
  - Out: Nothing
- Set Radius function
  - In: new radius for the circle
  - Out: True / False for success

In: radius

Out: Circle with the specified radius

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Example – Circle

- Get Radius
  - In: Nothing
  - Out: the radius of the circle
- Get diameter
  - In: Nothing
  - Out: the diameter of the circle
- Get Area
  - In: Nothing
  - Out: the area of the circle
- Get Circumference
  - In: Nothing
  - Out: the Circumference of the circle

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Example – Circle

- So far there has been no talk of how we will implement this ADT.
- We haven't even discussed what units the radius, diameter, and area will be in.
  - Inches
  - Millimeters
  - Meters
  - Miles
  - Does it matter.

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Classes

- C++ uses classes to implement ADTs
- Creating a class is no different than creating a structure
- Class Definition is usually placed in a (.h) header file.
- Class Implementation is usually placed in a (.cpp) source file
- Syntax:

```
class objectName
{

};
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Classes – 2 Sections

- There are actually 3 sections to your class, but we will only discuss 2 sections of a class
- Public:
  - This is where your function calls go that will implement your ADT.
  - The outside world (programmer using your class) can call any of these functions.
- Private:
  - This is where your data goes. The outside world can not access the data directly and must use your interface.
  - This is also where functions go that are for implementation purposes only.

Classes – Constructors and Destructors

- Constructors – used to create an instance of your class at instantiation time.
  - Constructors are functions with the same name as the class.
- Destructor – used to clean up your class object when it goes out of scope.
  - Destructor is a function with the same name as the class but preceded with a tilde (~).
- Notes: You can have as many constructors as you want, but you can only have one destructor.

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Classes – Creating your interface

```
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();

    private:
};
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Classes – Adding the Setters

```
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();

        // newradius in, t/f out
        bool setRadius( double newradius );

    private:
};
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Classes – Adding the Get Radius

```
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();
        bool setRadius( double newradius );
        // nothing in, the radius out
        double getRadius( );
    private:
};
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Classes – Adding the Get Diameter

```
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();
        bool setRadius( double newradius );
        double getRadius( );
        // nothing in, the diameter out
        double getDiameter( );

    private:
};
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Classes – Adding the Get Area

```
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();
        bool setRadius( double newradius );
        double getRadius( );
        double getDiameter( );
        // nothing in, the area out
        double getArea( );
    private:
};
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Classes – Adding the Get Circumference

```
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();
        bool setRadius( double newradius );
        double getRadius( );
        double getDiameter( );
        double getArea( );
        // nothing in, the circumference out
        double getCircumference( );
    private:
};
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Classes – Adding the Data Members

```
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();
        bool setRadius( double newradius );
        double getRadius( );
        double getDiameter( );
        double getArea( );
        double getCircumference( );
    private:
        double theRadius;
};
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Class Files

- Place your class definition inside a header file usually named after the class.
- Circle Class for example:
- Place definition inside circle.h
- Make sure and surround the definition with:

```
#ifndef __CIRCLE_H
#define __CIRCLE_H
// your class definition here
#endif
```

- Implementation of the class goes into a source file usually named after the class circle.cpp

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Circle.h

```
#ifndef __CIRCLE_H
#define __CIRCLE_H
class circle
{
    public:
        circle();
        circle( double radius );
        ~circle();
        bool setRadius( double
                        newradius );
        double getRadius( );
        double getDiameter( );
        double getArea( );
        double getCircumference( );

    private:
        double theRadius;
};
#endif
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Circle.cpp

- Make sure and include the class definition (circle.h)
- All functions for the class definition are implemented in the cpp file.
- Just like writing a function learned in csc150 with one exception.
  - It needs to be a member function. This gives it access to the private data of the class.
  - To make it a member function the name of the function must be preceeded by classname::. Ie. circle::

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

Circle.cpp – Constructors and Destructors

- Special syntax on constructors and destructors.
  - There is not return type or void on the function.
- Constructors initialize the data so that the object is valid after instantiation.
- Destructor is in charge of clean up.
  - Closing any files that are open.
  - Freeing any dynamically allocated memory.

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Constructor Function Implementation

```
circle::circle()
{
    theRadius = 1.0;
}

circle::circle( double radius )
{
    theRadius = 1.0;
    if( radius > 0 )
        theRadius = radius;
}
```

Note: the circle:: is what makes the function a member function.

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Destructor Implementation

- We have not memory allocated or files open.
- Nothing to clean up.
- If we didn't write the destructor one is written anyway that does absolutely nothing.
- In CSC 250 every class you write will have a destructor that you write.

```
circle::~circle()
{

}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

setRadius Function Implementation

- Don’t trust the data coming in. Make sure radius is always positive and greater than 0.

```
bool circle::setRadius( double newradius )
{
    if( newradius > 0 )
    {
        theRadius = newradius;
        return true;
    }
    return false;
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

getRadius Function Implementation

```
double circle::getRadius( )
{
    return theRadius;
}
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

getDiameter Function Implementation

```
double circle::getDiameter( )
{
    return 2 * theRadius;
}
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

getArea Function Implementation

```
double circle::getArea( )
{
    return theRadius * theRadius * 3.14159265359;
}
```

SOUTH DAKOTA
SCHOOL OF MINES
& TECHNOLOGY

getCircumference Function Implementation

```
double circle::getCircumference( )
{
    return 2 * theRadius * 3.14159265359;
}
```

The Entire C++ File

```
#include "circle.h"
circle::circle()
{   theRadius = 1.0;
}
circle::circle( double radius )
{   theRadius = 1.0;
    if( radius > 0 )
        theRadius = radius;
}
circle::~circle()
{
}
bool circle::setRadius( double newradius )
{   if( newradius > 0 )
    {
        theRadius = newradius;
        return true;
    }
    return false;
}
```

```
double circle::getRadius( )
{
    return theRadius;
}
double circle::getDiameter( )
{
    return 2 * theRadius;
}
double circle::getArea( )
{
    return theRadius * theRadius * 3.14159265359;
}
double circle::getCircumference( )
{
    return 2 * theRadius * 3.14159265359;
}
```

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY

Demonstration

- Show the constructors are called.
- Show how each object has its own data.
- Try and set the radius to a negative value (should fail).
- Change the radius to a positive value.
- Show the destructors being called.
- Bring in v2 of the ADT with a new constructor.

For Your Own Exercise

- Design an ADT for an led bulb
- Design an ADT for Radio
- Design an ADT for a DVD-R Recorder (Tivo)
- Design an ADT for a printer

- DO NOT WORRY ABOUT CODING IT TO WORK!!!!

SOUTH DAKOTA SCHOOL OF MINES & TECHNOLOGY