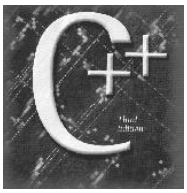

User-Defined Classes

Chapter 10

10.1 Class Definition

- int, float, char are built into C+
- Declare and use
  - int x = 5;
- Create user defined data types+
  - Extensive use throughout remainder of course
- Counter class will introduce the concept
  - counter.h counter class definitions
  - Define the class in the .h file

User Defined Types

- Define what data we want in the class
  - data elements of the class are private
- Create the functionality to operate on the data
  - class member functions access data
- We have created a new data type not known to the compiler

Counter.h

```
// FILE: Counter.h
// COUNTER CLASS DEFINITION

#ifndef COUNTER_H
#define COUNTER_H
class counter
{
   public:
      counter ();
      counter (int);
```


Counter.h

```
      // SET COUNTER VALUE
      void setCount (int);

      // INCREMENT COUNTER
      void increment ();

      // DECREMENT COUNTER
      void decrement ();

      // RETURN CURRENT COUNTER VALUE
      int getCount () const;
```


Counter.h

```
      // RETURN MAXIMUM COUNTER VALUE
      int getMaxValue () const;

   private:
      // Data Members (Attributes)...
      int count;
      int maxValue;

};

#endif
```

Compiler Directives

- Prevent multiple definitions
  - #ifndef COUNTER_H
  - #define COUNTER_H
  - #endif
- 1st the #ifndef tests to see if COUNTER_H has been defined
  - If not the 2nd line #define defines the content
  - If defined skips to the #endif

Using the counter Class

- Driver program CounterTest.cpp will help us see how the Counter Class is used
- Must #include "Counter.h" because our class definitions are contained in it.
  - Change this view based on our compiler
  - #include "Counter.cpp"

CounterTest.cpp

```
// File: CounterTest.cpp
// Test the counter class

#include <iostream>
#include "counter.h"
#include "counter.cpp"

using namespace std;

int main()
{
```


CounterTest.cpp

```
   counter c1;
   counter c2(10);

   // Test setValue, increment, decrement, and
   // accessValue functions.
   c1.setCount(50);
   c1.decrement();
   c1.decrement();
   c1.increment();
   cout << "Final value of c1 is " <<
                  c1.getCount() << endl;
```


CounterTest.cpp

```
   c2.increment();
   c2.increment();
   c2.decrement();
   cout << "Final value of c2 is " <<
                  c2.getCount() << endl;

   return 0;
}
```


10.2 Class Implementation

- Counter.cpp implementation details
- Hidden from users (details)
- Scope resolution operator
  - **::** prefix for each member function
  - Informs the compiler that the function is a member of the class
- Class member functions can access all class data members
  - Avoid using member functions inside member functions

Constructors

- Two special member functions
  - Same name as class name
- Constructor executes each time an object of type counter is declared
  - counter mike;
- Initializes object
- Types
  - Default
  - Class

Constructors

- Default Constructor
  - Used when no arguments passed as part of the declaration
- Class Constructor
  - Used when arguments are passed as part of the declaration
- Constructors do NOT specify a return type

Member Functions

- Member functions that modify data are
  - setValue
  - increment
  - decrement
- Member functions that retrieve data are
  - getvalue
  - getmaxvalue
- Const; at the end of the function means no data changes by the function

Counter.cpp

```
// File: counter.cpp
// Counter class implementation
#include "counter.h"
#include <iostream>
#include <climits>
using namespace std;

// Default constructor
counter::counter()
{
   count = 0;
   maxValue = INTMAX;
}
```


Counter.cpp

```
// Constructor with argument
counter::counter(int mVal)
{
   count = 0;
   maxValue = mVal;
}

// Increment counter
void counter::increment()
{
   if (count < maxValue)
      count++;
```


Counter.cpp

```
   else
      cerr << "Counter overflow. Increment
                          ignored." << endl;
}
// Decrement counter
void counter::decrement()
{
   if (count > 0)
      count--;
   else
      cerr << "Counter underflow. Decrement
                   ignored." << endl;
}
```

Counter.cpp

```
// Set counter value
void counter::setCount(int val)
{
   if (val >= 0  &&  val <= maxValue)
      count = val;
   else
      cerr << "New value is out of range. Value
                  not changed." << endl;
}
// Set maximum counter value
void counter::setMaxValue(int val)
{
```


Counter.cpp

```
   if (val >= 0  &&  val <= INTMAX)
      maxValue = val;
   else
      cerr << "New maxValue is out of range --
 not changed." << endl;
}

// Return current counter value
int counter::getCount() const
{
   return count;
}
```


Counter.cpp

```
// Return maximum counter value
int counter::getMaxValue() const
{
   return maxValue;
}
```


10.3 Summary of Rules for Use of Classes and Objects

- Class Instance
  - counter c1;
  - Creates an instance of the counter class (object)
  - Default constructor invoked
  - Allocates space in memory for object
- Similar to other data type declarations
  - int value;
  - Space in memory allocated for a data type int

Private vs Public

- Public member functions allow users to operate on the counter object c1
- May have private member functions
  - If member functions need to call another function not part of the class it should be private
- Use care when defining public access
- Users of a class are *clients*
- Class sometimes referred to as the *server*

Syntax

Form: class className

```
{
    public:
        List all public data and functions
    private:
        List all private data and functions
};
```

Comparing struct and classes

- struct & class define a data type
  - Collection of related elements
  - Prototype declarations
  - Three levels of access control
  - Public
  - Private
  - Protected (not covered)
- Major difference default access opposite
  - class private - struct public

Function Overloading & Polymorphism

- Functions with the same name is called ***function overloading***
- ***Polymorphism*** is what allows functions with the same name to do different things based on its arguments

10.4 Classes as Operands and Arguments

- Assignment operator can be used
- Others must be re-defined (Chap 11)
  - ***Operator Overloading***
- Use C& ob1 as formal reference argument
- Use const C& ob1 to specify object can't be modified by a function
  - Efficiency issues

CompareCounter

```
c1.compareCounter (c2) ;  or  c2.compareCounter (c1);

int counter::compareCounter (const counter& aCounter) const
{
    int     result;
    if (value <= aCounter.count)
            result = -1;
    else if  (value == aCounter.count)
            result = 0;
    else
            result = 1;

    return result;
}
```


10.5 A Fraction Class

- Design of Fraction Class
  - Data represents Numerator
  - Data represents Denominator
- FractionTest.cpp
- Fraction.cpp

FractionTest.cpp

```
// File: fractiontest.cpp
// Tests the fraction class.

#include <iostream>
#include "fraction.h"
#include "fraction.cpp"
using namespace std;

int main()
{
   fraction f1, f2;
   fraction f3;
```

FractionTest.cpp

```
    // Read two fractions
    cout << "Enter 1st fraction: " << endl;
    f1.readFrac();
    cout << "Enter 2nd fraction: " << endl;
    f2.readFrac();

    if (f1 == f2) cout << "same" << endl;

    // Display results of fraction arithmetic
    f3 = f1.multiply(f2);
    f1.displayFrac();  cout << " * ";
    f2.displayFrac();  cout << " = ";
```


FractionTest.cpp

```
    f3.displayFrac();  cout << endl;

    f3 = f1.divide(f2);
    f1.displayFrac();  cout << " / ";
    f2.displayFrac();  cout << " = ";
    f3.displayFrac();  cout << endl;

    //f3 = f1.add(f2);
    f3 = f1 + f2;
    f1.displayFrac();  cout << " + ";
    f2.displayFrac();  cout << " = ";
    f3.displayFrac();  cout << endl;
```


FractionTest.cpp

```
    f3 = f1.subtract(f2);
    f1.displayFrac();  cout << " - ";
    f2.displayFrac();  cout << " = ";
    f3.displayFrac();  cout << endl;

    return 0;
}
```


Fraction.cpp

```
// File: Fraction.cpp
// Fraction class implementation

#include "fraction.h"
#include <iostream>
using namespace std;
```


Fraction.cpp

```
// Member functions
// Constructors
fraction::fraction()
{
   num = 0;
   denom = 0;
}
fraction::fraction(int n)
{
   num = n;
   denom = 1;
}
```


Fraction.cpp

```
fraction::fraction(int n, int d)
{
   num = n;
   denom = d;
}

// Set numerator and denominator
void fraction::setNum(int n)
{
   num = n;
}
```

Fraction.cpp

```
void fraction::setDenom(int d)
{
   denom = d;
}
// Multiply fractions
fraction fraction::multiply(const fraction& f)
{
   fraction temp(num * f.num, denom * f.denom);
   return temp;
}
```


Fraction.cpp

```
// Divide fractions
fraction fraction::divide(const fraction& f)
{
   fraction temp(num * f.denom, denom * f.num);
   return temp;
}
// Add fractions
fraction fraction::add(const fraction& f)
{
   fraction temp(num * f.denom + f.num * denom,
                 denom * f.denom);
   return temp;
}
```


Fraction.cpp

```
// Subtract Fractions
fraction fraction::subtract(const fraction& f) {
   fraction temp(num * f.denom - f.num * denom,
                 denom * f.denom);
   return temp;
}
```


Fraction.cpp

```
// Read a fraction
void fraction::readFrac()
{
   char slash;       // storage for /
   do
   {
      cout << "Enter numerator / denominator: ";
      cin >> num >> slash >> denom;
   }
   while (slash != '/');
}
```


Fraction.cpp

```
// Display a fraction
void fraction::displayFrac() const
{
   cout << num << '/' << denom;
}

// Accessors
int fraction::getNum() const
{
   return num;
}
```


Fraction.cpp

```
fraction fraction::operator + (const fraction&
                  f2)   // IN: right-operand
{
   fraction temp(num * f2.denom + f2.num * denom,
                 denom * f2.denom);
   return temp;
}

bool fraction::operator == (const fraction& f)
{
   return(num == f.num && denom == f.denom);
}
```

Fraction.cpp

```
int fraction::getDenom() const
{
   return denom;
}
```


10.6 Circle Class

- Data members to represent x & y coordinates
- Radius
- Color
- Area
- Perimeter
- See specification on page 509

Circle.h

```
// File circle.h
// Circle class definition

#ifndef CIRCLE_H
#define CIRCLE_H

class circle
{
   public:
      // enumeration type
      enum color {black, blue, green, cyan, red,
                  magenta, brown, lightgray,
                                    nocolor};
```


Circle.h

```
      // Member Functions
      // constructor
      circle();

      // Set center coordinates
      void setCoord(int, int);

      // Set radius
      void setRadius(float);

      // Set color
      void setColor(color);
```


Circle.h

```
      // Compute the area
      void computeArea();

      // Compute the perimeter
      void computePerimeter();

      // Display attributes
      void displayCircle() const;
```


Circle.h

```
      // accessor functions
      int getX() const;
      int getY() const;
      float getRadius() const;
      color getColor() const;
      float getArea() const;
      float getPerimeter() const;
```

Circle.h

```
   private:
      // Data members (attributes)
      int x;
      int y;
      float radius;
      color cColor;
      float area;
      float perimeter;
};

#endif // CIRCLE_H
```


CircleTest.cpp

```
// File circletest.cpp
// Tests the Circle class

#include "circle.h"
#include "circle.cpp"
#include <iostream>

using namespace std;

int main()
{
```


CircleTest.cpp

```
   circle myCircle;

   // Set circle attributes.
   myCircle.setCoord(150, 100);
   myCircle.setRadius(10);
   myCircle.setColor(circle::magenta);

   // Compute area and perimeter
   myCircle.computeArea();
   myCircle.computePerimeter();
```


CircleTest.cpp

```
   // Display the circle attributes.
   cout << "The circle attributes follow:" <<
                                    endl;
   myCircle.displayCircle();

   return 0;
}
```


CircleTest.cpp

Program Output

The circle attributes follow:
x-coordinate is 150
y-coordinate is 100
radius is 10
color is 5
area is 314.159
perimeter is 62.8318

Circle.cpp

```
// File circle.cpp
// Circle class implementation

#include "circle.h"
#include <iostream>
using namespace std;

const float pi = 3.14159;
```

Circle.cpp

```
// Member Functions...
// constructor
circle::circle()
{
   x = 0;
   y = 0;
   radius = 0;
   cColor = nocolor;
   area = 0.0;
   perimeter = 0.0;
}
```


Circle.cpp

```
// Set center position
void circle::setCoord(int x, int y)
{
   x = x;
   y = y;
}
// Set radius
void circle::setRadius(float r)
{
   radius = r;
   computeArea();
   computePerimeter();
}
```


Circle.cpp

```
// Set color
void circle::setColor(color c)
{
   cColor = c;
}


// Compute the area
void circle::computeArea()
{
   area = pi * radius * radius;
}
```


Circle.cpp

```
// Compute the perimeter
void circle::computePerimeter()
{
   perimeter = 2 * pi * radius;
}

// Display attributes
void circle::displayCircle() const
{
   cout << "x-coordinate is " << x << endl;
   cout << "y-coordinate is " << y << endl;
   cout << "color is " << int(cColor) << endl;
```


Circle.cpp

```
   cout << "area is " << area << endl;
   cout << "perimeter is " << perimeter << endl;
}

// accessor functions
circle::color circle::getColor() const
{
   return cColor;
}

//    Insert definitions for rest of accessor
  functions here.
//       ...
```


Circle.cpp

```
int circle::getX() const
{  return x;
}

int circle::getY() const
{  return y;
}

float circle::getRadius() const
{  return radius;
}
```

Circle.cpp

```
float circle::getArea() const
{  return area;
}

float circle::getPerimeter() const
{  return perimeter;
}
```


10.7 A Simple String Class

- Since we have a string class we might take it for granted
- What do we need in a string class ?
  - create a string
  - length
  - capacity
  - at
  - read & write
- Look at SimpleString.h

SimpleString.h

```
// File simpleString.h
// Simple string class definition

#ifndef SIMPLESTRING_H
#define SIMPLESTRING_H

class simpleString
{
   public:
      // Member Functions
      // constructor
      simpleString();
```


SimpleString.h

```
      // Read a simple string
      void readString();

      // Display a simple string
      void writeString() const;

      // Retrieve the character at a specified
      // position
      // Returns the character \0 if position is
      // out of bounds
      char at(int) const;
```


SimpleString.h

```
      // Return the string length
      int getLength() const;

      // Return the string capacity
      int getCapacity() const;

      void getContents(char[]) const;
```


SimpleString.h

```
   private:
      // Data members (attributes)
      enum {capacity = 255};
      char contents[capacity];
      int length;
};

#endif //SIMPLESTRING_H
```

SimpleStringTest.cpp

```
// File: simpleStringTest.cpp
// Tests the simple string class

#include "simpleString.h"
#include "simpleString.cpp"
#include <iostream>

using namespace std;

int main()
{
```


SimpleStringTest.cpp

```
   simpleString aString;  // input - data string

   // Read in a string.
   cout << "Enter a string and press RETURN: ";
   aString.readString();

   // Display the string just read.
   cout << "The string read was: ";
   aString.writeString();
   cout << endl;
```


SimpleStringTest.cpp

```
   // Display each character on a separate line.
   cout << "The characters in the string
                        follow:" << endl;
   for (int pos = 0; pos < aString.getLength();
                                          pos++)
       cout << aString.at(pos) << endl;


   return 0;
}
```


SimpleStringTest.cpp

Program Output

Enter a string and press RETURN: **Philly cheesesteak**
The string read was: **Philly cheesesteak**
The characters in the string follow:
P
h
i
.etc etc
a
k

SimpleString.cpp

```
// File simplestring.cpp
// Simple string class implementation

#include "simplestring.h"
#include <iostream>
using namespace std;

// Member Functions...
// constructor
simpleString::simpleString()
{
   length = 0;    // Denotes an empty string
}
```


SimpleString.cpp

```
// Read a simple string
void simpleString::readString()
{
   // Local data...
   char next;
   int pos = 0;

   cin.get(next);
   while ((next != '\n') && (pos < capacity))
   {
```

SimpleString.cpp

```
      // Insert next in array contents
      contents[pos] = next;
      pos++;
      cin.get(next);
   }

   length = pos;
}
```


SimpleString.cpp

```
// Write a simple string
void simpleString::writeString() const
{
   for (int pos = 0; pos < length; pos++)
      cout << contents[pos];
}
// Retrieve the character at a specified position
// Returns the character \0 if position is out
// of bounds
char simpleString::at(int pos) const
{
   // Local data
   const char nullcharacter = '\0';
```


SimpleString.cpp

```
   if ((pos < 0) || (pos >= length))
   {
       cerr << "Character at position " <<
                  pos << " not defined." << endl;
       return nullcharacter;
   }
   else
      return contents[pos];
}
```


SimpleString.cpp

```
// Return the string length
int simpleString::getLength() const
{
   return length;
}

// Return the string capacity
int simpleString::getCapacity() const
{
   return capacity;
}
```


SimpleString.cpp

```
void simpleString::getContents(char str[]) const
{
   for (int i = 0; i < length; i++)
      str[i] = contents[i];
}
```


10.8 A Savings Account Class

- Problem Statement *(see page 522)*
- Analysis
- Design
- Implementation
- Test & Verification

Savings.h

```
// File Savings.h
// Savings account class definition

#include "money.h"          // access money class
#include <string>           // access string class
using namespace std;

#ifndef SAVINGS_H
#define SAVINGS_H
```


Savings.h

```
class savings
{
   public:
      // Member Functions...
      // constructor
      savings();

      // Open a new account
      void openAccount();

      // Change account name
      void changeName(int, string);
```


Savings.h

```
      // Add quarterly interest
      void addInterest();

      // Process a deposit
      void deposit(int, money);

      // Process a withdrawal
      void withdraw(int, money);

      // Close an account
      void closeAccount(int);
```


Savings.h

```
      // Get account balance
      money getBalance() const;

   private:
      // Data members (attributes)
      int id;
      string name;
      money balance;
      float interestRate;
```


Savings.h

```
      // Member functions...
      // Validate user identification
      bool validId(int) const;
};

#endif // SAVINGS_H
```


Savings.cpp

```
// File savings.cpp
// Savings account implementation file

#include "savings.h"
#include "money.h"
#include <string>
#include <iostream>
using namespace std;
```

Savings.cpp

```
// Member Functions...
// constructor
savings::savings()
{
   name = "";
   id = 0;
   balance = 0.0;
   interestRate = 0.0;
}
```


Savings.cpp

```
// Open a new account
void savings::openAccount()
{
   cout << "Account name: ";
   getline(cin, name, '\n');
   cout << "Account ID: ";
   cin >> id;
   cout << "Initial balance: $";
   cin >> balance;
   cout << "Annual interest rate percentage: %";
   cin >> interestRate;
}
```


Savings.cpp

```
// Validate user id
bool savings::validId(int ident) const
{
   if (id == ident)
      return true;
   else
   {
      cerr << "Error - ID's do not match! ";
      return false;
   }
}
```


Savings.cpp

```
// Change account name
void savings::changeName(int ident, string na)
{
   if (validId(ident))
   {
      name = na;
      cout << "Changing account name to " <<
                                      na << endl;
   }
   else
      cerr << "Reject name change request." <<
                                           endl;
}
```


Savings.cpp

```
// Add quarterly interest
void savings::addInterest()
{
   // Local data
   float quarterRateFrac;
   quarterRateFrac = interestRate / 400.0;
   balance += balance * quarterRateFrac;
}
```


Savings.cpp

```
// Process a deposit
void savings::deposit(int ident, money amount)
{
   if (validId(ident))
   {
      balance += amount;
      cout << "Depositing " << amount << endl;
   }
   else
      cerr << "Reject deposit of " << amount <<
                                           endl;
}
```

Savings.cpp

```
// Process a withdrawal
void savings::withdraw(int ident, money amount)
{
   if ((validId (ident)) && (amount <= balance))
   {
      balance -= amount;
      cout << "Withdrawing " << amount << endl;
   }
   else
      cerr << "Reject withdrawal of " <<
                                 amount << endl;
}
```


Savings.cpp

```
// Close an account
void savings::closeAccount(int ident)
{
   if (validId(ident))
   {
      cout << "Final balance for account number "
           << id << " is " << balance << endl;
      cout << "Account has been closed" << endl;
      balance = 0.0;
      id = 0;
      name = "";
   }
```


Savings.cpp

```
   else
      cerr << "Account not closed" << endl;
}


// Get account balance
money savings::getBalance() const
{
   return balance;
}
```


Testing the Class

- Use a driver to test
  - SavingsTest.cpp
- Create ***savings*** objects
- Operates on the declared objects
- Try valid and invalid data

SavingsTest.cpp

```
// File savingsTest.cpp
// Tests the savings class

#include <iostream>
#include "savings.h"
#include "savings.cpp"
#include "money.h"
#include "money.cpp"

using namespace std;
```


SavingsTest.cpp

```
int main()
{
   savings myAccount;

   // Open a savings account.
   myAccount.openAccount();
   cout << endl;

   // Make valid and invalid deposit.
   myAccount.deposit(1234, 500.00);
   myAccount.deposit(1111, 300.00);
```

SavingsTest.cpp

```
// Get and display balance.
cout << endl << "Current balance is "
     << myAccount.getBalance() << endl;

// Make valid and invalid withdrawal.
myAccount.withdraw(1234, 750.00);
myAccount.withdraw(1234, 15000.00);

// Add interest.
myAccount.addInterest();
```


SavingsTest.cpp

```
   // Close the account.
   myAccount.closeAccount(1234);

   return 0;
}
```


SavingsTest.cpp

Program Output

Account name: **William Gates**
Account ID: **1234**
Initial Balance: **$1000.00**
Annual interest rate percentage: **%5**

Depositing **$500.00**
Error - Ids do not match ! Reject deposit of **$300.00**

Current balance is **$1,500.00**
Withdrawing **$750.00**
Reject withdrawal of **$15,000.00**
Final balance for account number **1234** is **$759.38**
Account has been closed

10.9 Common Programming Errors

- Function argument & return value errors
- Not defining a function as a member of a class
- Prefixing a class function call
  - obj.function();
- Referencing a private attribute of a class
- Missing header files
- Missing **;** on class definition