Computer Programming Class Members 9th Lecture

엄현상 (Eom, Hyeonsang)
School of Computer Science and Engineering
Seoul National University

Outline

- Class Scope
- Constructors and Destructors
- Copy Constructors
- **const** Members
- Member Initializer
- **friend** Functions and Classes
- Static Members
- Information Hiding and Abstract Data Types
- Q&A

Preprocessor Wrappers

- Prevents code from being included more than once

```
#ifndef TIME_H
#define TIME_H
… // code
#endif
```

- Prevents multiple-definition errors

Stream Manipulator **setfill**

- Specifies the fill character
  - When an output field wider than the number of digits in the output value
  - Appears to the left of the digits in the number
- Applies for all subsequent values

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Time Class

```
#ifndef TIME_H
#define TIME_H
class Time {
 public:
   Time();
   void setTime(int,int,int);
   void printUniversal();
   void printStandard();

 private:
   int hour;
   int minute;
   int second;
};
#endif

=================================

#include <iostream>
using std::cout;
#include <iomanip>
using std::setfill;
using std::setw;
#include "Time.h"
```

```
Time::Time()
{
   hour = minute = second = 0;
}

void Time::setTime( int h, int m, int s )
{
   …
   second = ( s >= 0 && s < 60 ) ? s : 0;
}

void Time::printUniversal()
{
   cout << setfill( '0' );
   cout << setw( 2 ) << hour;
   …
}

void Time::printStandard()
{
   cout << (( hour == 0 || hour == 12 ) ? 12 :
     hour % 12 ) << “:”;
     …
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Time Class Cont'd

```
#include <iostream>
using std::cout;
using std::endl;

#include "Time.h"

int main()
{
   Time t;

   t.printUniversal();

   t.printStandard();

   t.setTime( 13, 27, 6 );

   t.printUniversal();

   t.printStandard();

   t.setTime( 99, 99, 99 );

   t.printUniversal();

   t.printStandard();

   cout << endl;

   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

sizeof Operator for Classes

- Applying operator sizeof to a class name or to an object of that class
  - will report only the size of the class's data members
- The compiler creates one copy (only) of the member functions for all objects of the class
  - All objects of the class share this copy
- Each object needs its own copy of the class's data

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Scope

- Class scope contains
  - Data members (variables declared in the class definition)
  - Member functions (functions declared in the class definition)
- Nonmember functions are defined at file scope
- Within a class's scope
  - Class members are accessible by all member functions
- Outside a class's scope
  - public class members are referenced through a handle
    - An object name, a reference to an object, or a pointer to an object

Class Scope Cont'd

- Variables declared in a member function
  - Have block scope
  - Known only to that function
- Hiding a class-scope variable
  - In a member function, define a variable with the same name as a variable with class scope
  - To access the hidden class-scope variable, use the scope resolution operator (::)

Class Scope Cont'd

- Dot member selection operator (.)
  - Accesses the object's members
  - Used with an object's name or with a reference to an object
- Arrow member selection operator (->)
  - Accesses the object's members
  - Used with a pointer to an object

Constructors with Default Arguments

- Can initialize data members to a consistent state
- Constructor that defaults all its arguments
  - A default constructor
  - Maximum of one default constructor per class
- Any change to the default argument values of a function requires the client code to be recompiled

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Destructors

- A special member function
  - ~Time()
- Called implicitly when an object is destroyed
  - When program execution leaves the scope in which that object was instantiated
  - Performs "termination housekeeping"
  - Then the system reclaims the object's memory

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Destructors Cont’d

- Receives no parameters and returns no value
  - May not specify a return type—not even void
- A class may have only one destructor
- If the programmer does not explicitly provide a destructor, the compiler creates an “empty” destructor

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

When Constructors and Destructors are Called?

- Called implicitly by the compiler
- In general, destructor calls are made in the reverse order of the corresponding constructor calls
- Storage classes of objects can alter the order in which destructors are called

Objects Defined in Global Scope

- Constructors are called before any other function (including main) in that file begins execution
- The corresponding destructors are called when main terminates
  - Function exit
    - Forces a program to terminate immediately
    - Often used to terminate a program when an error is detected
  - Function abort
    - Forces the program to terminate immediately without allowing the destructors of any objects to be called
    - Usually used to indicate an abnormal termination of the program

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Automatic Objects

- Constructors and destructors are called each time execution enters and leaves the scope of the object
- Automatic object destructors are not called if the program terminates with an exit or abort function

Static Local Objects

- Constructor is called only once
  - When execution first reaches where the object is defined
- Destructor is called when main terminates or the program calls function exit
  - Destructor is not called if the program terminates with a call to function abort
- Global and static objects are destroyed in the reverse order of their creation

Class CreatAndDestroy

```
#include <string>
using std::string;

#ifndef CREATE_H
#define CREATE_H

class CreateAndDestroy
{
 public:
   CreateAndDestroy( int, string );
   ~CreateAndDestroy();
 private:
   int objectID;
   string message;
};

#endif
```

```
#include <iostream>
using std::cout;
using std::endl;

#include "CreateAndDestroy.h"

CreateAndDestroy::CreateAndDestroy( int
     ID, string messageString )
{
   objectID = ID;
   message = messageString;

   cout << "Object " << objectID;
   cout << "   constructor runs   ";
   cout << message << endl;
}

CreateAndDestroy::~CreateAndDestroy()
{
   cout << "Object " << objectID;
   cout << "   destructor runs   ";
   cout << message << endl;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class CreatAndDestroy Cont’d

```
#include <iostream>
using std::cout;
using std::endl;
#include "CreateAndDestroy.h“

void create( void );
CreateAndDestroy first( 1,
    "(global before main)" );

int main()
{
   cout << "EXECUTION BEGINS"
    << endl;
   CreateAndDestroy second( 2,
    "(local automatic in main)" );
   static CreateAndDestroy
    third( 3, "(local static in
    main)" );
   create();
   cout << "EXECUTION RESUMES"
    << endl;
```

```
   CreateAndDestroy fourth( 4,
    "(local automatic in main)" );
   cout << "EXECUTION ENDS"
    << endl;
   return 0;
}

void create( void )
{
   cout << "CREATE BEGINS"
    << endl;
   CreateAndDestroy fifth( 5,
    "(local automatic in
    create)" );
   static CreateAndDestroy
    sixth( 6, "(local static in
    create)" );
   CreateAndDestroy seventh( 7,
    "(local automatic in
    create)" );
   cout << "CREATE ENDS" << endl;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class CreatAndDestroy Cont'd

1. Object 1 constructor runs
2. EXECUTION BEGINS
3. Object 2 constructor runs
4. Object 3 constructor runs
5. CREATE BEGINS
6. Object 5 constructor runs
7. Object 6 constructor runs
8. Object 7 constructor runs
9. CREATE ENDS
10. Object 7 destructor runs

1. Object 5 destructor runs
2. EXECUTION RESUMES
3. Object 4 constructor runs
4. EXECUTION ENDS
5. Object 4 destructor runs
6. Object 2 destructor runs
7. Object 6 destructor runs
8. Object 3 destructor runs
9. Object 1 destructor runs

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Returning a Reference to an Object

- Alias for the name of an object
  - May be used on the left side of an assignment statement
  - A const reference cannot be used as a modifiable lvalue
- A public member function of a class returns a reference to a private data member of that class
  - Client code could alter private data
  - Same problem would occur if a pointer to private data were returned

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Default Memberwise Assignment

- Assignment operator (=)
- Can be used to assign an object to another object of the same type
  - Each data member of the right object is assigned to the same data member in the left object
  - Shallow copy
- When data members contain pointers to dynamically allocated memory
  - May cause serious problems

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Date

```
#ifndef DATE_H
#define DATE_H

class Date
{
 public:
   Date( int = 1, int = 1, int
    = 2000 );
   void print();

 private:
   int month;
   int day;
   int year;
};
#endif
```

```
#include <iostream>
using std::cout;
using std::endl;

#include "Date.h"

Date::Date( int m, int d, int
    y )
{
   month = m;
   day = d;
   year = y;
}

void Date::print()
{
   cout << month << '/'
    << day << '/' << year;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Date Cont'd

```
#include <iostream>
using std::cout;
using std::endl;

#include "Date.h"

int main()
{
   Date date1( 7, 4, 2004 );
   Date date2;

   cout << "date1 = ";
   date1.print();
   cout << "\ndate2 = ";
   date2.print();

   date2 = date1;

   date2.print();
   cout << endl;

   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Copy Constructors

- Enables pass-by-value for objects
  - Used to copy original object's values into new object to be passed to a function or returned from a function
- Compiler provides a default copy constructor
  - Copies each member of the original object into the corresponding member of the new object (i.e., memberwise assignment)
  - Shallow copy

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Copy Constructors Cont'd

- When data members contain pointers to dynamically allocated memory
  - May cause serious problems
    - Need to have a deep copy
    - May need a destructor and operator=

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Point

```
class Point
{
public:
   …
   Point();
   Point(const Point& p);
   …
 private:
   int x;
   int y;
};

Point::Point(int px, int py)
{
   x = px;
   y = py;
}

Point::Point(const Point& p)
{
   x = p.x;
   y = p.y;
}
```

```
Point p(1,2); //constructor
Point q(3,4); //constructor
Point r(p);   //copy constructor
Point t = q;  //copy constructor
p = t;        //assignment
…
foo(p);       //copy constructor
…
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Const Objects

- Keyword const
- The object is not modifiable
  - compilation errors
  - Attempts to modify the object are caught at compile time rather than causing execution-time errors
- A const object cannot be modified by assignment, so it must be initialized

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Const Member Functions

- Only for const objects
- Not allowed to modify the object
- Specified as const both in its prototype and in its definition
- Not allowed for constructors and destructors
- Can be overloaded with a non-const version
  - The compiler chooses which overloaded member function to use based on the object on which the function is invoked

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Time

```
class Time
{
 public:
   Time( int = 0, int = 0, int = 0 );

   void setTime( int, int, int );
   void setHour( int );
   void setMinute( int );
   void setSecond( int );

   int getHour() const;
   int getMinute() const;
   int getSecond() const;

   void printUniversal() const;
   void printStandard(); // const

 private:
   int hour;
   int minute;
   int second;
};
```

```
Time::Time( int hour, int minute, int
     second )
{
   setTime( hour, minute, second );
}

void Time::setTime( int hour, int
     minute, int second )
{
   setHour( hour );
   setMinute( minute );
   setSecond( second );
}

void Time::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0;
}

void Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m :
     0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Time Cont’d

```
void Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ?
    s : 0;
}

int Time::getHour() const
{
   return hour;
}

int Time::getMinute() const
{
   return minute;
}

int Time::getSecond() const
{
   return second;
}
```

```
void Time::printUniversal() const
{
   cout << setfill( '0' )
    << setw( 2 ) << hour << ":"
    << setw( 2 ) << minute << ":"
    << setw( 2 ) << second;
}

void Time::printStandard() //
    const
{
   cout << ( ( hour == 0 || hour
    == 12 ) ? 12 : hour % 12 )
    << ":" << setfill( '0' )
    << setw( 2 ) << minute << ":"
    << setw( 2 ) << second
    << ( hour < 12 ? " AM" : "
    PM" );
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Time Cont'd

```
int main()
{
   Time wakeUp(6,45,0);
   const Time noon(12,0,0);

   wakeUp.setHour( 18 );
   noon.setHour( 12 );
   wakeUp.getHour();
   noon.getMinute();
   noon.printUniversal();
   noon.printStandard();

   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Member Initializer

- Required for initializing,
  - Const data members
  - Data members that are references
- Can be used for any data member
- Member initializer list
  - Between a constructor's parameter list and the constructor's body
  - Separated from the parameter list with a colon (:)
  - The data member name followed by parentheses containing the member's initial value

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Member Initializer

- Member initializer list
  - Multiple member initializers are separated by commas
  - Executes before the body of the constructor executes
- For a const data member of a class, a member initializer must be used to provide the constructor with the initial value of the data member for an object of the class
  - The same is true for references

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Increment

```
class Increment
{
 public:
   Increment(int c=0,int i=1);

   void addIncrement()
   {
      count += increment;
   }

   void print() const;

 private:
   int   count;
   const int increment;
};
```

```
Increment::Increment( int c,
    int i )
   : count( c ),
     // initializer for
     // non-const member
     increment( i )
     // required initializer
     // for const member
{
}

void Increment::print() const
{
   cout << "count = "
    << count << ", increment =
    " << increment << endl;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Composition

- Has-a relationship
- A class can have objects of other classes as members
- Initializing member objects
  - Member initializers pass arguments from the object's constructor to member-object constructors
  - Member objects are constructed in the order in which they are declared in the class definition
    - Not in the order they are listed in the constructor's member initializer list
    - Before the enclosing class object (host object) is constructed

Class Date

```
class Date
{
 public:
   Date( int = 1, int = 1, int =
    1900 );
   void print() const;
   ~Date();

 private:
   int month;
   int day;
   int year;

   int checkDay( int ) const;
};
```

```
Date::Date( int mn, int dy, int
    yr )
{
   if ( mn > 0 && mn <= 12 )
      month = mn;
   else
   {
      month = 1;
      cout << "Invalid month (";
      cout << mn << ") set to
    1.\n";
   }
   year = yr;
   day = checkDay( dy );

   cout << "Date object
    constructor for date ";
   print();
   cout << endl;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Date Cont'd

```
void Date::print() const
{
   cout << month << '/' << day
    << '/' << year;
}

Date::~Date()
{
   cout << "Date object
    destructor for date ";
   print();
   cout << endl;
}
```

```
int Date::checkDay( int testDay )
    const
{
   static const int
    daysPerMonth[ 13 ] =
      { 0, 31, 28, 31, 30, 31, 30,
    31, 31, 30, 31, 30, 31 };

   if ( testDay > 0 && testDay <=
    daysPerMonth[ month ] )
      return testDay;

   if ( month == 2 && testDay ==
    29 && ( year % 400 == 0 ||
      ( year % 4 == 0 && year %
    100 != 0 ) ) )
      return testDay;

   cout << "Invalid day ("
    << testDay << ") set to 1.\n";
   return 1;
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Employee

```
class Employee
{
public:
   Employee( const char * const, const
     char * const,
      const Date &, const Date & );
   void print() const;
   ~Employee();

private:
   char firstName[ 25 ];
   char lastName[ 25 ];
   const Date birthDate;
   const Date hireDate;
};
```

```
Employee::Employee( const char * const
     first, const char * const last,
   const Date &dateOfBirth, const Date
     &dateOfHire )
   : birthDate( dateOfBirth ),
     hireDate( dateOfHire )
{
   int length = strlen( first );
   length = ( length < 25 ? length :
     24 );
   strncpy( firstName, first, length );
   firstName[ length ] = '\0';

   length = strlen( last );
   length = ( length < 25 ? length :
     24 );
   strncpy( lastName, last, length );
   lastName[ length ] = '\0';

   cout << "Employee object constructor:
     ";
   cout << firstName << ' ' << lastName
     << endl;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Employee Cont'd

```
void Employee::print() const
{
   cout << lastName << ", "
    << firstName << "  Hired: ";
   hireDate.print();
   cout << "  Birthday: ";
   birthDate.print();
   cout << endl;
}

Employee::~Employee()
{
   cout << "Employee object
    destructor: " ;
   cout << lastName << ", "
    << firstName << endl;
}
```

```
int main()
{
   Date birth( 7, 24, 1949 );
   Date hire( 3, 12, 1988 );
   Employee manager( "Bob",
    "Blue", birth, hire );

   cout << endl;
   manager.print();

   cout << "\nTest Date
    constructor with invalid
    values:\n";
   Date lastDayOff( 14, 35,
    1994 );

   cout << endl;
   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Friend Functions and Classes of a Class

- Defined outside that class's scope
- Has the right to access the non-public and public members of that class
- Standalone functions or entire classes
- Can enhance performance
- The function prototype in the class definition preceded by keyword **`friend`**

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Friend Functions and Classes of a Class Cont'd

- Member access notions of private, protected, and public are not relevant to friend declarations
  - Friend declarations can be placed anywhere in a class definition
- Place a declaration of the form "friend class Class2;" in the definition of class Class1
  - All member functions of class Class2 are friends of class Class1

Class Count

```
class Count
{
   friend void setX( Count &,
    int );

 public
   Count()
      : x( 0 )
   {
   }

   void print() const
   {
      cout << x << endl;
   }

 private:
   int x;
};
```

```
void setX( Count &c, int val )
{
   c.x = val;
}

int main()
{
   Count counter;

   cout << "counter.x: ";
   counter.print();

   setX( counter, 8 );
   cout << "counter.x after
    call to setX friend
    function: ";
   counter.print();

   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Friend Functions and Classes of a Class Cont'd

- For class B to be a friend of class A, class A must explicitly declare (in its definition) that class B is its friend
- Friendship relation
  - Neither symmetric nor transitive
- It is possible to specify overloaded functions as friends of a class
  - Each overloaded function intended to be a friend must be explicitly declared as a friend of the class

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

this Pointer

- Access to an object itself through a pointer called this (keyword)
- this pointer is not part of the object itself
- Passed (by the compiler) as an implicit argument to each of the object's non-static member functions
- Implicit access when accessing members directly

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Test

- Type of the this pointer
  - □ Depends on the type of the object and whether the executing member function is const

```
class Test
{
 public:
   Test( int = 0 );
   void print() const;

 private:
   int x;
};

Test::Test( int value )
   : x( value )
{
}
```

```
void Test::print() const
{
    cout << "x= " << x;
    cout << “\nthis->x=" << this-
    >x;
    cout << "\n(*this).x="
    << ( *this ).x << endl;
}

int main()
{
   Test testObject( 12 );

   testObject.print();

   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Cascaded Member-Function Calls

- Enabled by member functions returning the dereferenced this pointer
- **`t.setMinute( 30 ).setSecond( 22 );`**
  - □ Calls t.setMinute(30);
  - □ Then calls t.setSecond(22);

Class Time

```
class Time
{
 public:
   Time( int = 0, int = 0, int = 0 );

   Time &setTime( int, int, int );
   Time &setHour( int );
   Time &setMinute( int );
   Time &setSecond( int );

   int getHour() const;
   int getMinute() const;
   int getSecond() const;

   void printUniversal() const;
   void printStandard() const;
 private:
   int hour;
   int minute;
   int second;
};
```

```
 Time::Time( int hr, int min, int sec )
{
   setTime( hr, min, sec );
}

Time &Time::setTime(int h, int m, int s)
{
   setHour( h );
   setMinute( m );
   setSecond( s );
   return *this;
}

Time &Time::setHour( int h )
{
   hour = ( h >= 0 && h < 24 ) ? h : 0;
   return *this;
}

Time &Time::setMinute( int m )
{
   minute = ( m >= 0 && m < 60 ) ? m : 0;
   return *this;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Time Cont'd

```
Time &Time::setSecond( int s )
{
   second = ( s >= 0 && s < 60 ) ?
    s : 0;
   return *this;
}

int Time::getHour() const
{
   return hour;
}

int Time::getMinute() const
{
   return minute;
}

int Time::getSecond() const
{
   return second;
}
```

```
void Time::printUniversal() const
{
   cout << setfill( '0' )
    << setw( 2 ) << hour << ":"
      << setw( 2 ) << minute
    << ":" << setw( 2 ) << second;
}

void Time::printStandard() const
{
   cout << ( ( hour == 0 || hour
    == 12 ) ? 12 : hour % 12 )
      << ":" << setfill( '0' )
    << setw( 2 ) << minute
      << ":" << setw( 2 )
    << second << ( hour < 12 ? "
    AM" : " PM" );
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Time Cont'd

```
int main()
{
   Time t;


     t.setHour( 18 ).setMinute( 30 ).se
     tSecond( 22 );

   cout << "Universal time: ";
   t.printUniversal();

   cout << "\nStandard time: ";
   t.printStandard();

   cout << "\n\nNew standard time: ";

   t.setTime( 20, 20,
     20 ).printStandard();
   cout << endl;

   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Dynamic Memory Management

- To allocate and deallocate memory for any built-in or user-defined type
  - Operators **new** and **delete**
- **new**
  - Allocates (i.e., reserves) storage of the proper size for an object at execution time
  - Calls a constructor to initialize the object
  - Returns a pointer of the type specified
  - Works for any fundamental type or any class type
- Heap

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Dynamic Memory Management Cont'd

- **delete**
  - Destroys a dynamically allocated object
  - Calls the destructor for the object
  - Deallocates (i.e., releases) memory from the free store
- Initializing an object allocated by new
  - Initializer for a newly created fundamental-type variable

    ```
    double *ptr = new double( 3.14159 );
    ```

  - Specify a comma-separated list of arguments to the constructor of an object

    ```
    Time *timePtr = new Time( 12, 45, 0 );
    ```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Dynamic Memory Management Cont'd

- Allocating arrays dynamically

  ```
  int *gradesArray = new int[ 10 ];
  ```

- Delete a dynamically allocated array:

  ```
  delete [] gradesArray;
  ```

  - This deallocates the array to which gradesArray points
  - If the pointer points to an array of objects
    - First calls the destructor for every object in the array
    - Then deallocates the memory
  - If the statement did not include the square brackets ([]) and gradesArray pointed to an array of objects
    - Only the first object in the array would have a destructor call
- After deleting dynamically allocated memory, set the pointer that referred to that memory to 0

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

static Data Member

- Only one copy of a variable shared by all objects of a class
  - Class-wide information
- Declaration begins with keyword **static**
- May seem like global variables but have class scope
- Can be declared public, private, or protected
- static data members of class types (i.e., static member objects) that have default constructors
  - Need not be initialized because their default constructors will be called

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

static Data Member Cont'd

- Fundamental-type static data members
  - Initialized by default to 0
  - A static data member can be initialized once (and only once)
- A const static data member of int or enum type
  - Can be initialized in its declaration in the class definition
- All other static data members
  - Must be defined at file scope (i.e., outside the body of the class definition)
  - Can be initialized only in those definitions

static Data Member Cont'd

- Exists even when no objects of the class exist
  - To access a public static class member when no objects of the class exist
    - Prefix the class name and the binary scope resolution operator (::)

```
Martian::martianCount
```

static Member Function

- Is a service of the class, not of a specific object of the class
- static applied to an item at file scope
    - That item becomes known only in that file
    - The static members of the class need to be available from any client code that accesses the file
        - We cannot declare them static in the .cpp file—we declare them static only in the .h file

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

static Member Function Cont'd

- Declare a member function static
  - If it does not access non-static data members or non-static member functions of the class
- Does not have a **this** pointer
- Static data members and static member functions exist independently of any objects of a class
  - When a static member function is called, there might not be any objects of its class in memory
- Sometimes it is recommended that all calls to static member functions be made using the class name
  - not an object handle
- A **const** static member function is a compilation error

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Employee

```
#ifndef EMPLOYEE_H
#define EMPLOYEE_H
class Employee
{
public:
   Employee( const char * const,
    const char * const );
   ~Employee();
   const char *getFirstName()
    const;
   const char *getLastName()
    const;


   static int getCount();
private:
   char *firstName;
   char *lastName;

   static int count;
};
#endif
```

```
#include <iostream>
using std::cout;
using std::endl;

#include <cstring>
using std::strlen;
using std::strcpy;

#include "Employee.h"

int Employee::count = 0;

int Employee::getCount()
{
   return count;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Employee Cont’d

```
Employee::Employee( const char * const
     first, const char * const last )
{
   firstName = new char[ strlen( first )
     + 1 ];
   strcpy( firstName, first );

   lastName = new char[ strlen( last )
     + 1 ];
   strcpy( lastName, last );

   count++;

   cout << "Employee constructor for "
     << firstName << ' ' << lastName
     << " called." << endl;
}

const char *Employee::getFirstName()
     const
{
   return firstName;
}
```

```
Employee::~Employee()
{
   cout << "~Employee() called for "
     << firstName
      << ' ' << lastName << endl;

   delete [] firstName;
   delete [] lastName;

   count--;
}

const char *Employee::getLastName()
     const
{
   return lastName;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Class Employee Cont'd

```
#include <iostream>
using std::cout;
using std::endl;

#include "Employee.h"

int main()
{
   cout << "Number of employees before
     instantiation of any objects is "
      << Employee::getCount() << endl;

   Employee *e1Ptr = new
     Employee( "Susan", "Baker" );
   Employee *e2Ptr = new
     Employee( "Robert", "Jones" );

   cout << "Number of employees after
     objects are instantiated is "
      << e1Ptr->getCount();
```

```
   cout << "\n\nEmployee 1: "
      << e1Ptr->getFirstName() << " "
     << e1Ptr->getLastName()
      << "\nEmployee 2: "
      << e2Ptr->getFirstName() << " "
     << e2Ptr->getLastName() << "\n\n";

   delete e1Ptr;
   e1Ptr = 0;
   delete e2Ptr;
   e2Ptr = 0;

   cout << "Number of employees after
     objects are deleted is "
      << Employee::getCount() << endl;
   return 0;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Data Abstraction and Information Hiding

- Information Hiding
- Data abstraction
  - Client cares about what functionality a class offers, not about how that functionality is implemented
- Primary activities of object-oriented programming in C++
  - Creation of types (i.e., classes)
  - Expression of the interactions among objects of those types

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Abstract data types (ADTs)

- Improve the program development process
- Representing real-world notions Types like **int**, **double**, **char** and others are all ADTs
  - e.g., **int** is an abstract representation of an integer
- Capture two notions:
  - Data representation
  - Operations that can be performed on the data

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Array Abstract Data Type

- Many array operations not built into C++
  - e.g., subscript range checking
- Programmers can develop an array ADT as a class that is preferable to primitive arrays
- C++ Standard Library class template vector

Container Classes

- Collection classes
- Classes designed to hold collections of objects
- Services such as insertion, deletion, searching, sorting, and member testing
- Arrays, Vectors, Stacks, Queues, Trees, Linked lists

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008

Iterators

- Iterator objects
- Commonly associated with container classes
- An object that walks through a collection, returning the next item (or performing some action on the next item)
- A container class can have several iterators operating on it at once
- Each iterator maintains its own position information

```
vector<int> v; // fill up v with data...
vector<int>::iterator it;
for ( it = v.begin(); it != v.end(); it++ ) {
    cout << *it << endl;
}
```

C++ How to Program 6th Ed., P. Deitel and H. M. Deitel, Pearson Education, 2008