Lecture 5

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Classes – C++ Primer Chapter 7

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

Constructors

Constructor: Special member function that is used to initialize newly created objects.

The job of a constructor is to ensure that the data members of an object have safe, sensible initial values.

default constructor: The constructor that is used when no explicit values are given for an initializer of a class type object.

For example, the default constructor for string initializes the new string as the empty string.

Classes I

Class: C++ mechanism for defining own abstract data types.

- Classes may have data, function or type members.
- A class defines a new type and a new scope.

member function: Class member that is a function.

- Ordinary member functions are bound to an object of the class type through the implicit this pointer.
- Static member functions are not bound to an object and have no this pointer.

Classes II

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

access label: defines if the following members are accessible to users of the class or only to friends and members

- Each label sets the access protection for the members declared up to the next label.
- Labels may appear multiple times within the class.
- public, private, or protected

Data abstraction and Encapsulation I

abstract data type: data structure (like a class) using encapsulation to hide its implementation.

data abstraction: Programming technique that focuses on the interface to a type.

- Allows programmers to ignore the details of how a type is represented and to think instead about the operations that the type can perform.

Data abstraction and Encapsulation II

Encapsulation: Separation of implementation from interface

encapsulation hides implementation details of a type

- In C++, encapsulation is enforced by preventing general user access to the private parts of a class.
- Access labels enforce abstraction and encapsulation

Definitions

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Interface: The operations supported by a type.

Implementation: The (usually private) members of a class that define the data and any operations that are not intended for use by code that uses the type.

Interface and Implementation Rules

- Well-designed classes separate their interface and implementation, defining the interface in the public part of the class and the implementation in the private parts.
- Data members ordinarily are part of the implementation.
- Function members are part of the interface when they are operations that users of the type are expected to use and part of the implementation when they perform operations needed by the class but not defined for general use.

Data abstraction and Encapsulation III

- Advantages of abstraction and encapsulation
  - Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object
  - The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code

Class Definitions

- **incomplete type**: A type that has been declared (forward declaration) but not yet defined.
  - It is not possible use an incomplete type to define a variable or class member.
  - It is legal to define references or pointers to incomplete types.
- Hints:
  - Use typedefs to streamline classes
  - Use only few (overloaded) member functions
  - Use inlined member functions if possible
- Why a class definitions ends with a semicolon:
  - class Sales_item { /* … */ } accum, trans;

Const and Mutable data members

const member function: member function that may not change an object's ordinary (i.e., neither static nor mutable) data members.

- The this pointer in a const member is a pointer to const.
- A member function may be overloaded based on whether the function is const.

mutable data member: Data member that is never const, even when it is a member of a const object.

- A mutable member can be changed inside a const function.

Class Scope

- **class scope**: Each class defines a scope.
  - Class scopes are more complicated than other scopes—member functions defined within the class body may use names that appear after the definition.
  - Two different classes have two different class scopes (even if they have the same member list)
- Member definitions
  - double MyClass::accumulate() const { }
- Parameter lists and function bodies are in class scope
- Function return types are not always in class scope
  - MyClass::index MyClass::accumulate(index number) const { }

Name Lookup

- **Name lookup**: The process by which the use of a name is matched to its corresponding declaration
- Class definitions are processed in two phases
  - First, the member declarations are compiled
  - Only after all the class members have been seen are the definitions themselves compiled

Constructors

- Constructors are special member functions that are executed whenever we create new objects of a class type
- Constructors have the same name as the class and may not specify a return type
- Constructors may be overloaded
- Constructors may not be declared as const

Constructor Initializer List I

- **constructor initializer list**: Specifies initial values of the data members of a class.
  - The members are initialized to the values specified in the initializer list before the body of the constructor executes.
  - Class members that are not initialized in the initializer list are implicitly initialized by using their default constructor.
  - Image(size_t cols = 0, size_t rows = 0) : cols_(cols), rows_(rows) {}
- Members that must be initialized in the constructor list
  - Members of class type without default constructor
  - const or reference type members

Constructor Initializer List II

- Members are initialized in order of their definitions
- Initializers may be any expression
- Initializers for data members of class type may call any of its constructors
- Hint: prefer to use default arguments in constructors because this reduces code duplication

Default Constructor

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

- **default constructor**: The constructor that is used when no initializer is specified.
- **synthesized default constructor**: The default constructor created (synthesized) by the compiler for classes that do not define any constructors.
  - This constructor initializes members of class type by running that class's default constructor
  - members of built-in type are uninitialized.
- Common mistake when trying to use the default constructor:
  - MyClass myobj(); // defines a function, not an object!

Implicit class-type conversions

- **conversion constructor**: A nonexplicit constructor that can be called with a single argument.
  - A conversion constructor is used implicitly to convert from the argument's type to the class type
  - MyClass(string &s) {}
- **explicit constructor**: Constructor that can be called with a single argument but that may not be used to perform an implicit conversion.
  - A constructor is made explicit by prepending the keyword explicit to its declaration.
  - explicit MyClass(string &s) {}

Friends

- Friend: Mechanism by which a class grants access to its nonpublic members.
  - Both classes and functions may be named as friends.
  - friends have the same access rights as members.
  - A friend declaration introduces the named class or nonmember function into the surrounding scope
  - A friend function may be defined inside the class, then the scope of the function is exported to the scope enclosing the class definition

Static class members

- **static member**: Data or function member that is not a part of any object but is shared by all objects of a given class.
- Advantages of static members compared to globals
  - The name of a static member is in the scope of the class, thereby avoiding name collisions with members of other classes or global objects
  - Encapsulation can be forced since a static member can be private, a global object cannot
  - It is easy to see by reading the program that a static member is associated with a particular class. This visibility clarifies the programmer’s intention
- Static member functions have no this pointer

Sales_item.h: class definition revisited

```cpp
#pragma once

#include <iostream>
#include <string>

class Sales_item {
friend bool operator==(const Sales_item&, const Sales_item&);
public:
    Sales_item(const std::string &book):
            isbn(book), units_sold(0), revenue(0.0) { }
    Sales_item(std::istream &is) { is >> *this; }
    friend std::istream& operator>>(std::istream&, Sales_item&);
    friend std::ostream& operator<<(std::ostream&, const Sales_item&);

    Sales_item& operator+=(const Sales_item&);

    double avg_price() const;
    bool same_isbn(const Sales_item &rhs) const
        { return isbn == rhs.isbn; }
    Sales_item(): units_sold(0), revenue(0.0) { }
private:
    std::string isbn;
    unsigned units_sold;
    double revenue;

};
```