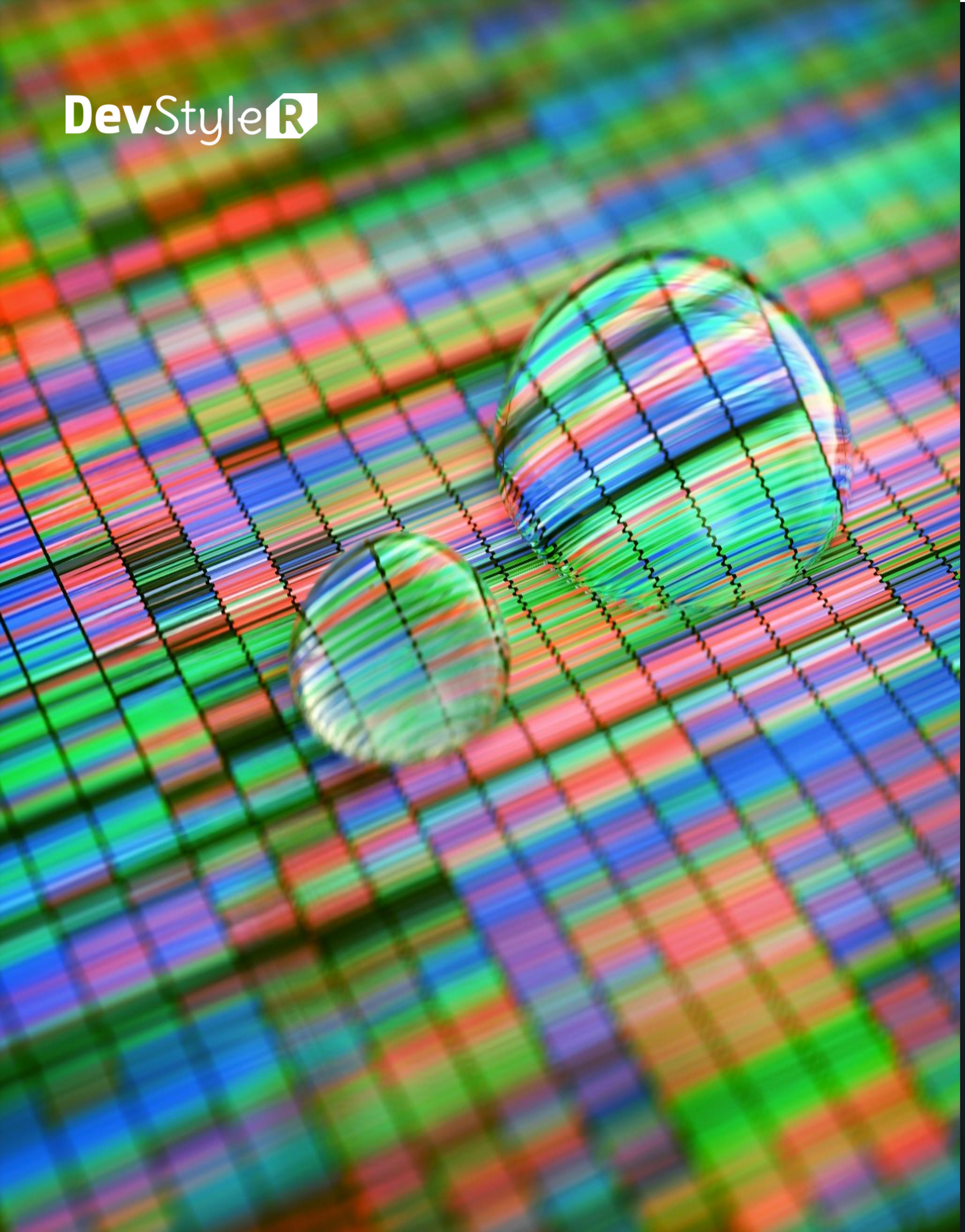
DevStyleR ACADEMY

Programming with C++ Core

CLASSES: A DEEPER LOOK, PART 2

2024



Summary

- Const (Constant) Objects and const Member Functions
- Composition: Objects as Members of Classes
- Friend Functions and friend Classes
- Dynamic Memory Management with Operators new and delete
- Static Class Members

CONST (CONSTANT) OBJECTS AND CONST MEMBER FUNCTIONS

- Principle of least privilege
  - One of the most fundamental principles of good software engineering
  - Applies to objects, too
- const objects
  - Keyword *const*
  - Specifies that an object is not modifiable
  - Attempts to modify the object will result in compilation errors

CONST (CONSTANT) OBJECTS AND CONST MEMBER FUNCTIONS

- Const member functions
  - Only const member function can be called for const objects
  - Member functions declared const are not allowed to modify the object
  - A function is specified as const both in its prototype and in its definition
  - const declarations are not allowed for constructors and destructors

LIVE DEMO

ConstNonConst

DevStyle ACADEMY

CONST (CONSTANT) OBJECTS AND CONST MEMBER FUNCTIONS

- Member initializer
  - Required for initializing
    - const data members
    - Data members that are references
  - Can be used for any data member
- Member initializer list
  - Appears between a constructor’s parameter list and the left brace that begins the constructor’s body
  - Separated from the parameter list with a colon (:)
  - Each member initializer consists of the data member name followed by parentheses containing the member’s initial value
  - Multiple member initializers are separated by commas
  - Executes before the body of the constructor executes

DevStyleR ACADEMY

Programming with C++ Core

LIVE DEMO

Increment

2024

DevStyleR ACADEMY

COMPOSITION: OBJECTS AS MEMBERS OF CLASSES

- Composition
  - Sometimes referred to as a *has-a* relationship
  - A class can have objects of other classes as members
  - Example
    - AlarmClock object with a Time object as a member

COMPOSITION: OBJECTS AS MEMBERS OF CLASSES

- 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
  - If a member initializer is not provided
    - The member object's default constructor will be called implicitly

DevStyleR ACADEMY

Programming with C++ Core

LIVE DEMO

Composition

2024

DevStyleR ACADEMY

FRIEND FUNCTIONS AND FRIEND CLASSES

- friend function of a class
  - Defined outside that class's scope
    - Not a member function of that class
  - Yet has the right to access the non-public (and public) members of that class
  - Standalone functions or entire classes may be declared to be friends of a class
  - Can enhance performance
  - Often appropriate when a member function cannot be used for certain operations

DevStyleR ACADEMY

FRIEND FUNCTIONS AND FRIEND CLASSES

- To declare a function as a friend of a class:
  - Provide the function prototype in the class definition preceded by keyword friend
- To declare a class as a friend of a class:
  - Place a declaration of the form

    ```
    friend class ClassTwo;
    ```

    in the definition of class ClassOne
    - All member functions of class ClassTwo are friends of class ClassOne

DevStyleR ACADEMY

FRIEND FUNCTIONS AND FRIEND CLASSES

- Friendship is granted, not taken
  - For class B to be a friend of class A, class A must explicitly declare that class B is its friend
- Friendship relation is neither symmetric nor transitive
  - If class A is a friend of class B, and class B is a friend of class C, you cannot infer that class B is a friend of class A, that class C is a friend of class B, or that class A is a friend of class C
- 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

DevStyleR ACADEMY

Programming with C++ Core

LIVE DEMO

Friend

2024

DevStyleR ACADEMY

USING THE THIS POINTER

- Cascaded member-function calls
  - Multiple functions are invoked in the same statement
  - Enabled by member functions returning the dereferenced this pointer
  - Example
    - t.setMinute(30).setSecond(22);
      - Calls t.setMinute(30);
      - Then calls t.setSecond(22);
-

DevStyleR ACADEMY

Programming with C++ Core

LIVE DEMO

CascadingThisPointer

2024

DYNAMIC MEMORY MANAGEMENT WHIT OPERATORS NEW AND DELETE

- Dynamic memory management
  - Enables programmers to allocate and deallocate memory for any built-in or user-defined type
  - Performed by operators new and delete
  - For example, dynamically allocating memory for an array instead of using a fixed-size array

DevStyleR ACADEMY

DYNAMIC MEMORY MANAGEMENT WHIT OPERATORS NEW AND DELETE

- Operator 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 to the right of new
  - Can be used to dynamically allocate any fundamental type (such as int or double) or any class type
- Free store
  - Sometimes called the heap
  - Region of memory assigned to each program for storing objects created at execution time

DYNAMIC MEMORY MANAGEMENT WHIT OPERATORS NEW AND DELETE

- Operator delete
  - Destroys a dynamically allocated object
  - Calls the destructor for the object
  - Deallocates (i.e., releases) memory from the free store
  - The memory can then be reused by the system to allocate other objects

DYNAMIC MEMORY MANAGEMENT WHIT OPERATORS NEW AND DELETE

- Initializing an object allocated by new
  - Initializer for a newly created fundamental-type variable
    - Example
      - double *ptr = new double(3.14159);
  - Specify a comma-separated list of arguments to the constructor of an object
    - Example
      - Time *timePtr = new Time(12, 45, 0);

DevStyleR ACADEMY

DYNAMIC MEMORY MANAGEMENT WHIT OPERATORS NEW AND DELETE

- new operator can be used to allocate arrays dynamically
  - Dynamically allocate a 10-element integer array:
    int *gradesArray = new int[10];
  - Size of a dynamically allocated array
    - Specified using any integral expression that can be evaluated at execution time

DevStyleR ACADEMY

DYNAMIC MEMORY MANAGEMENT WHIT OPERATORS NEW AND DELETE

- 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

STATIC CLASS MEMBERS

- static data member
  - Only one copy of a variable shared by all objects of a class
    - “Class-wide” information
    - A property of the class shared by all instances, not a property of a specific object of the class
  - Declaration begins with keyword static

DevStyleR ACADEMY

STATIC CLASS MEMBERS

- static data member (Cont.)
  - Example
    - Video game with Martians and other space creatures
      - Each Martian needs to know the martianCount
      - martianCount should be static class-wide data
      - Every Martian can access martianCount as if it were a data member of that Martian
      - Only one copy of martianCount exists
  - May seem like global variables but have class scope
  - Can be declared public, private or protected

STATIC CLASS MEMBERS

- static data member (Cont.)
  - Fundamental-type static data members
    - Initialized by default to 0
    - If you want a different initial value, 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
    - Can be initialized only in those definitions
  - 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

STATIC CLASS MEMBERS

- static data member (Cont.)
  - 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 (::) to the name of the data member
      - Example
        - Martian::martianCount
  - Also accessible through any object of that class
    - Use the object’s name, the dot operator and the name of the member
      - Example
        - myMartian.martianCount

DevStyleR ACADEMY

STATIC CLASS MEMBERS

- 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
    - So we cannot declare them static in the .cpp file—we declare them static only in the .h file.

STATIC CLASS MEMBERS

- Declare a member function static
  - If it does not access non-static data members or non-static member functions of the class
  - A static member function 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

DevStyleR ACADEMY

Programming with C++ Core

LIVE DEMO

Employee

2024

DevStyleR ACADEMY

THANK YOU

ACADEMY@DEVSTYLER.IO

DevStyleR ACADEMY

Programming with C++ Core

EXERCISES

2024

TASK 1

Create a *SavingsAccount* class. Use a static data member *annualInterestRate* to store the annual interest rate for each of the savers.

- Each member of the class contains a private data member *savingsBalance* indicating the amount the saver currently has on deposit.
- Provide member function *calculateMonthlyInterest* that calculates the monthly interest by multiplying the balance by *annualInterestRate* divided by 12; this interest should be added to *savingsBalance*.
- Provide a static member function *modifyInterestRate* that sets the static *annualInterestRate* to a new value. Write a driver program to test class *SavingsAccount*.
- Instantiate two different objects of class *SavingsAccount*, *saver1* and *saver2*, with balances of $2000.00 and $3000.00, respectively. Set the *annualInterestRate* to 3 percent.
- Then calculate the monthly interest and print the new balances for each of the savers.
- Then set the *annualInterestRate* to 4 percent, calculate the next month's interest and print the new balances for each of the savers.

TASK 2

Create class *IntegerSet* for which each object can hold integers in the range 0 through 100. A set is represented internally as an array of ones and zeros. Array element a[i] is 1 if integer i is in the set. Array element a[j] is 0 if integer j is not in the set. The default constructor initializes a set to the so-called "empty set," i.e., a set whose array representation contains all zeros.

- Provide member functions *unionOfSets*, *intersectionOfSets*, *insertElement*, *deleteElement*, *printSet*, *isEqualTo.*
- Provide an additional constructor that receives an array of integers and the size of that array and uses the array to initialize a set object.
- Write a driver program to test your class.