

C++ Classes

Combining Everything You Know

10 Exercises from First Class to Advanced Patterns

Difficulty Key	
★ ■ ■	Part 1 — Absolute Beginner (Exercises 1–5) Classes, methods, encapsulation, vectors
★ ★ ★	Part 2 — Combining Everything (Exercises 6–10) Templates, operator overloading, file I/O, dynamic memory

This booklet assumes you have worked through the core C++ exercises. Each exercise here requires you to design and implement a class from scratch.

Table of Contents

Part 1 — Absolute Beginner

- 01. Your First Class — Dog (Classes) ★■■■
- 02. Rectangle — Area and Perimeter (Classes) ★■■■
- 03. Counter Class (Classes) ★■■■
- 04. BankAccount — Deposit and Withdraw (Classes) ★■■■
- 05. Student — Grade Average (Classes) ★■■■

Part 2 — Combining Everything

- 06. Stack Using a Vector (Classes) ★★★
- 07. Linked List of Integers (Classes) ★★★
- 08. Matrix Class with Operator Overloading (Classes) ★★★
- 09. Student Database with File I/O (Classes) ★★★
- 10. Temperature Logger Class (Classes) ★★★

Exercise 1 of 10

★

Description

Declare (but do not define) `void set_values(std::string n, int a)` inside the class, then define it **outside** using the scope resolution operator `Dog::`.

Hint

Expected Output

Your Solution

★

Description

Add two methods:

- *double perimeter()* that returns $2 \times (\text{width} + \text{height})$.

Hint

Expected Output

Perimeter: 16

★

Add three public methods:

- `void increment()` — adds 1 to count.

- *void reset()* — sets count back to 0.

— *int getCount()* — returns the current value.

In *main*, increment the counter 5 times, print it, reset it, then print it again.

Define a class *BankAccount* with a private *double balance* initialised to 0.0.
Add these public methods:

- *void deposit(double amount)* — adds amount to balance (ignore negatives).
- *bool withdraw(double amount)* — subtracts amount if funds are sufficient; returns *true* on success, *false* if not enough funds.
- *double getBalance()* — returns the current balance.

In *main*, deposit 100, withdraw 30, attempt to withdraw 200, and print the balance after each operation.

Check amount <= balance before subtracting in withdraw. Return false without changing the balance when the check fails.

```
Balance: 100
Balance: 70
Withdrawal failed.
Balance: 70
```



Define a class *Student* with a public `std::string name` and a private `std::vector<int> grades`.

— *Student(std::string name)* — constructor.

— *void addGrade(int g)* — appends a grade.

— *double average()* — returns the mean of all grades (return 0 if no grades).

— *void report()* — prints the student name and their average.

In *main*, create a student, add grades 85, 92, 76, 90, and call *report()*.

Sum the vector with a range-based for loop, then divide by (double)grades.size(). Include <vector> and <string>.

```
Student: Alice
Average: 85.75
```

Exercise 6 of 10

★ ★ ★

Description

- *void push(T val)* — adds to the top.

- `pop()` — removes and returns the top element; print an error and return a default value if empty.

- `T peek()` — returns the top without removing it.

— *bool isEmpty()* — returns true if no elements.

— *int size()* — returns the number of elements.

In *main*, push integers 1–5, print the size, pop twice printing each popped value, then peek and print the new top.

Hint

Declare the class as `template <typename T> class Stack { ... };`. Use `data.back()` for peek and `data.pop_back()` for pop.

Expected Output

```
Size: 5
Popped: 5
Popped: 4
Top: 3
```

Your Solution

★ ★ ★

Description

- *struct Node* with an *int* value and a *Node** next pointer.
- *class LinkedList* with a private *Node* head* (starts as *nullptr*), a destructor that frees all nodes, and these methods:

- In *main*, push 3 to the front, then 1 to the front, then 5 to the back, and print.

For pushFront: create a new node, set its next to head, then update head. For pushBack: traverse to the last node first. The destructor should walk the list deleting each node.

```
1 -> 3 -> 5 -> null
```

★ ★ ★

Description

Include:

- A constructor that takes four values (row-major: a, b, c, d).
 - *operator+* — element-wise addition.
 - *operator** — matrix multiplication.
 - *double det()* — returns the determinant ($ad - bc$).
 - *operator<<* (friend) — prints the matrix row by row.
- In *main*, define $A = (1, 2, 3, 4)$ and $B = (5, 6, 7, 8)$, print $A+B$, $A*B$, and $\det(A)$.

Overload `operator+` and `operator*` as member functions returning a new `Matrix2x2`. For `operator<<` declare it as friend `std::ostream& operator<<(std::ostream&, const Matrix2x2&)` inside the class.

```
A+B:
6 8
10 12
A*B:
19 22
43 50
det(A): -2
```



Student Database with File I/O

Description

Create a *class* *Student* with *std::string name* and *double gpa*. Build a *class* *Database* that holds a *std::vector<Student>* and provides:

- *void add(Student s)* — adds a student.
- *void saveTo(std::string filename)* — writes each student (name and GPA) to a file, one per line.
- *void loadFrom(std::string filename)* — reads students back from the file.
- *Student* topStudent()* — returns a pointer to the student with the highest GPA (or *nullptr* if empty).

In *main*, add three students, save to *students.txt*, clear and reload, then print the top student.

Hint

Use `std::ofstream` for saving and `std::ifstream` for loading. For `topStudent`, iterate the vector keeping a pointer to the current best.

Expected Output

```
Loaded 3 students.  
Top student: Alice (GPA: 3.9)
```

Your Solution

