Introduction to C++ - Practice Exercises

Worksheet for Students

Computer Science Department

2025-10-10

Instructions

This worksheet contains exercises covering the fundamental concepts of C++ programming. Complete each exercise, test your code, and make sure it compiles without errors. Focus on using modern C++ practices (C++17/20).

Important Guidelines:

- Use meaningful variable names
- Add comments to explain your logic
- Test your code with different inputs
- Follow modern C++ best practices
- Handle potential errors appropriately

Section 1: Basic Types and Variables

Exercise 1.1: Variable Declaration and Initialization

Write a program that declares variables of different types using modern initialization syntax:

Requirements:

a) Declare an integer variable `age` and initialize it to 25 using uniform initialization
b) Declare a double variable `pi` using `constexpr` and set it to 3.14159
c) Declare a boolean variable `is_student` and set it to `true`
d) Declare a string variable `name` and initialize it to your name
e) Use `auto` to declare a variable that stores the result of `10 * 5`
f) Print all variables to the console

Expected Output:

```
Age: 25
Pi: 3.14159
Is Student: 1
```

```
Name: [Your Name]
Result: 50
```

Your Solution:

```
// Write your code here
```

Exercise 1.2: Type Sizes

Write a program that displays the size (in bytes) of the following types:

- `bool`, `char`, `short`, `int`, `long`, `long long`
- `float`, `double`
- Your system's pointer size using `void*`

Also, use `std::numeric_limits` to print the minimum and maximum values for `int` and `double`.

Your Solution:

```
// Write your code here
```

Section 2: Functions

Exercise 2.1: Simple Functions

Write the following functions and test them in `main()`:

a) `square(double x)` - returns the square of x
b) `is_even(int n)` - returns true if n is even, false otherwise
c) `max_of_three(int a, int b, int c)` - returns the largest of three integers
d) `print_separator()` - a void function that prints "===========" to console

Your Solution:

```
// Write your code here
```

Exercise 2.2: Function with Default Parameters

Create a function `greet(std::string name, std::string greeting = "Hello")` that prints a greeting message. If no greeting is provided, it should use “Hello” as default.

Test it with: - `greet("Alice")` - `greet("Bob", "Good morning")`

Your Solution:

```
// Write your code here
```

Section 3: Arrays and Loops

Exercise 3.1: Array Operations

Write a program that:

a) Creates a `std::array<int, 5>` with values {10, 20, 30, 40, 50}
b) Prints all elements using a traditional for loop
c) Prints all elements using a range-based for loop
d) Calculates and prints the sum of all elements
e) Finds and prints the maximum value

Your Solution:

```
// Write your code here
```

Exercise 3.2: Vector Manipulation

Write a program that:

a) Creates an empty `std::vector<int>`
b) Adds the numbers 1 through 10 to the vector using a loop
c) Doubles each element in the vector using a range-based for loop with references
d) Removes all elements greater than 15
e) Prints the final contents

Your Solution:

```
// Write your code here
```

```

```

Section 4: Pointers and References

Exercise 4.1: Understanding References

Complete this program by filling in the missing parts:

```cpp
#include <iostream>

void swap_by_value(int a, int b) {
    // TODO: Implement (this won't work for swapping)
}

void swap_by_reference(/* TODO: Add parameters */) {
    // TODO: Implement correct swap
}

int main() {
    int x = 10, y = 20;

    std::cout << "Before: x = " << x << ", y = " << y << "\n";

    swap_by_value(x, y);
    std::cout << "After swap_by_value: x = " << x << ", y = " << y << "\n";

    swap_by_reference(x, y);
    std::cout << "After swap_by_reference: x = " << x << ", y = " << y <<
"\n";

    return 0;
}
```

Your Solution:

```cpp
// Write your complete code here
```

Exercise 4.2: Pointer Basics

Write a program that:

a) Declares an integer variable `num` with value 42
b) Creates a pointer `ptr` that points to `num`
c) Prints the value of `num`, the address of `num`, and the value pointed by `ptr`
d) Changes the value of `num` through the pointer to 100
e) Verifies that `num` has changed

Your Solution:

```
// Write your code here
```

Section 5: Structures and Classes

Exercise 5.1: Student Structure

Create a `Student` struct with the following members: - `std::string name` - `int id` - `double gpa`

Then write a program that:

a) Creates three Student objects with different data
b) Stores them in a `std::vector<Student>`
c) Prints information for all students
d) Finds and prints the student with the highest GPA

Your Solution:

```
// Write your code here
```

Exercise 5.2: Rectangle Class

Create a `Rectangle` class with:

Private members: - `double width` - `double height`

Public members: - Constructor that takes width and height - `area()` method that returns the area - `perimeter()` method that returns the perimeter - `is_square()` method that returns true if width equals height - `scale(double factor)` method that multiplies both dimensions by factor

Test your class by creating rectangles and calling all methods.

Your Solution:

```
// Write your code here
```

Section 6: Enumerations

Exercise 6.1: Traffic Light System

Create an `enum class TrafficLight` with values: `red`, `yellow`, `green`.

Write a function `get_action(TrafficLight light)` that returns a string: - Red ⟶ "Stop" - Yellow ⟶ "Prepare to stop" - Green ⟶ "Go"

Also implement an `operator++` that cycles through the lights in order.

Your Solution:

```
// Write your code here
```

Exercise 6.2: Days of Week

Create an `enum class Day` representing days of the week.

Write functions: - `is_weekend(Day d)` - returns true for Saturday and Sunday - `day_name(Day d)` - returns the string name of the day - `next_day(Day d)` - returns the next day (Sunday follows Saturday)

Your Solution:

```
// Write your code here
```

Section 7: Error Handling

Exercise 7.1: Safe Division

Write a function `safe_divide(double a, double b)` that:

- Returns `a / b` if `b` is not zero
- Throws `std::invalid_argument` exception if `b` is zero

Write a main function that uses try-catch to handle the exception properly.

Your Solution:

```
// Write your code here
```

Exercise 7.2: Array Bounds Checking

Create a class `SafeArray` that:

- Has a private `std::array<int, 10>` member
- Has an `at(int index)` method that returns the element at index
- Throws `std::out_of_range` if index is invalid
- Has a `set(int index, int value)` method with the same error checking

Test your class with both valid and invalid indices.

Your Solution:

```
// Write your code here
```

Section 8: Templates and Generic Programming

Exercise 8.1: Generic Maximum Function

Write a template function `maximum` that works with any type that supports the `>` operator:

```cpp
template<typename T>
T maximum(T a, T b) {
    // TODO: Implement
}
```

Test it with: - Integers: `maximum(5, 10)` - Doubles: `maximum(3.14, 2.71)` - Strings: `maximum(std::string("apple"), std::string("banana"))`

Your Solution:

```cpp
// Write your code here
```

Exercise 8.2: Generic Container Statistics

Write a template function that calculates the average of elements in any container:

```cpp
template<typename Container>
double average(const Container& c) {
    // TODO: Implement
    // Hint: use range-based for loop
}
```

Test it with: - `std::vector<int>` - `std::array<double, 5>` - `std::list<float>`

Your Solution:

```cpp
// Write your code here
```

Section 9: Modern C++ Features

Exercise 9.1: Smart Pointers

Rewrite this code to use `std::unique_ptr` instead of raw pointers:

```cpp
#include <iostream>

class Resource {
public:
    Resource() { std::cout << "Resource acquired\n"; }
    ~Resource() { std::cout << "Resource destroyed\n"; }
    void use() { std::cout << "Resource in use\n"; }
};
```

```
int main() {
    Resource* ptr = new Resource();
    ptr->use();
    delete ptr;  // Easy to forget!
    return 0;
}
```

Your Solution:

```
// Write your code here
```

Exercise 9.2: Lambda Expressions

Write a program that:

a) Creates a `std::vector<int>` with values {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
b) Uses `std::count_if` with a lambda to count even numbers
c) Uses `std::for_each` with a lambda to print each element
d) Uses `std::transform` with a lambda to square each element

Your Solution:

```
// Write your code here
```

Section 10: Comprehensive Challenge

Exercise 10.1: Bank Account System

Create a complete bank account system with the following requirements:

1. Create an `enum class AccountType`: - `checking`, `savings`, `business`

2. Create a `BankAccount` class with:

Private members: - `std::string owner_name` - `int account_number` - `double balance` - `AccountType type`

Public members: - Constructor that initializes all members - `deposit(double amount)` - throws exception if amount <= 0 - `withdraw(double amount)` - throws exception if insufficient funds or amount <= 0 - `get_balance()` const - returns current balance - `get_info()` const - returns formatted string with account info - `transfer(BankAccount& other, double amount)` - transfers money to another account

3. In main(): - Create at least 3 different accounts - Perform various operations (deposits, withdrawals, transfers) - Use try-catch blocks to handle exceptions - Print account information after each operation

Your Solution:

```
// Write your complete solution here
```

Section 11: Bonus Challenges

Exercise 11.1: Temperature Converter

Create a temperature conversion system:

1. Create an `enum class TempScale`: `celsius`, `fahrenheit`, `kelvin`

2. Create a `Temperature` class: - Stores value and scale - Has methods to convert to other scales - Overloads comparison operators (<, >, ==) - Overloads arithmetic operators (+, -)

3. Formulas: - C to F: `(C × 9/5) + 32` - C to K: `C + 273.15` - F to C: `(F - 32) × 5/9` - K to C: `K - 273.15`

Your Solution:

```
// Write your code here
```

Exercise 11.2: Simple Vector Class

Implement a simplified version of `std::vector` called `SimpleVector`:

Requirements: - Uses dynamic memory allocation with smart pointers - Has `push_back()`, `pop_back()`, `size()`, `capacity()` methods - Implements `operator[]` for element access - Has proper copy/move constructors and assignment operators - Automatically resizes when capacity is reached

Your Solution:

```
// Write your code here
```

Submission Guidelines

When submitting your worksheet:

1. **Code Quality:**
   - All code must compile without errors
   - Use modern C++ features (C++17/20)
   - Follow consistent naming conventions
   - Add meaningful comments
2. **Testing:**
   - Test each exercise with multiple test cases
   - Include edge cases (empty inputs, negative numbers, etc.)
   - Handle errors appropriately
3. **Documentation:**
   - Write a brief comment explaining your approach for complex exercises
   - Note any assumptions you made
4. **Format:**
   - Submit as a `.cpp` file for each exercise, or
   - Submit a single file with clearly marked sections

Grading Rubric

Category	Points	Description
Correctness	40	Code produces correct output for all test cases
Modern C++	20	Uses C++17/20 features appropriately
Code Style	15	Clean, readable code with proper formatting
Error Handling	15	Appropriate use of exceptions and validation
Comments	10	Clear explanations of logic and approach
Total	100	

Additional Resources

- **C++ Reference:** cppreference.com

- **Compiler Explorer:** godbolt.org
- **C++ Core Guidelines:** isocpp.github.io/CppCoreGuidelines

Good luck with your exercises!