复习材料：函数与操作符 (第四章)

1. 函数与操作符如何改变状态 (Functions and Operators Transform State)

- **函数的参数列表** 映射为输出（可能为空），不同的参数列表可以重载同一个函数名或操作符。
  - 函数参数在调用点初始化。
- **作用域的建立**：每个函数或操作符都会建立作用域。
  - 局部自动变量：生命周期仅限于函数调用期间。
  - 局部静态变量：生命周期从调用到调用之间保持。
- **传值与传引用的语义**：
  - **传值 (Pass by Value)**：传递参数时会创建参数的副本，对副本的修改不会影响原始参数。
  - **传引用 (Pass by Reference)**：传递参数时传递的是参数的引用，修改引用会改变原始参数。
  - **传指针变量的方式**：既可以通过值传递指针，也可以通过引用传递指针。

2. 操作符的优先级和结合性 (Operator Precedence & Associativity)

- **C++ 操作符的列表是固定的**：不能新增操作符。
- 每个操作符都有其特定的结合性和优先级：
  - 左结合性： `cout << j << endl;` （左移操作符 `<<` ）。
  - 右结合性： `i = j = k = 0;` （赋值操作符 `=` ）。
- **操作符的相对优先级**：可以通过括号来改变默认的优先级和结合性。
- **内置类型的操作符**：对于如 `int` 等内置类型，操作符的优先级已经定义好。
  - 隐式类型转换增加了操作符的适用范围（例如 `<` 操作符）。

示例代码

```
#include "Pressure.h"
#include "Weekday.h"
#include <set>
#include <algorithm>
#include <iterator>
#include <iostream>

using namespace std;

int main() {
    set<Weekday> Weekdays;
    Weekdays.insert(Weekday::Mon);
    Weekdays.insert(Weekday::Tue);
    Weekdays.insert(Weekday::Wed);

    set<Weekday> Weekdays2;
    Weekdays2.insert(Weekday::Mon);
    Weekdays2.insert(Weekday::Tue);
    Weekdays2.insert(Weekday::Thu);

    set<Weekday> result;
    result.insert(Weekday::Sun);

    // 使用 set_union 算法，计算两个集合的并集
    set_union(Weekdays.begin(), Weekdays.end(),
              Weekdays2.begin(), Weekdays2.end(),
              inserter(result, result.begin()));

    // 输出并集结果
    for (Weekday day : result) {
        cout << day << endl;
    }

    return 0;
}
```

传值、传引用、传指针的详细讲解

1. 传值 (Pass by Value)

- 当参数按值传递时，函数接收到的是参数的副本，函数内的操作不会影响原始值。
- **优点**：确保原始值不会被函数改变。
- **缺点**：对于大对象的参数，拷贝副本的开销较大。

示例：

```
void increment(int x) {
    x++;
}

int main() {
    int a = 5;
    increment(a);  // a 传递的是副本，increment 修改的是副本
    cout << a;     // 输出：5，a 未被修改
}
```

2. 传引用 (Pass by Reference)

- 传引用时，函数操作的是参数的实际值，因此函数内部的修改会影响原始值。
- **优点**：避免拷贝，直接操作原始数据。
- **缺点**：如果不小心，可能会意外修改原始值。

示例：

```
void increment(int& x) {
    x++;
}

int main() {
    int a = 5;
    increment(a);  // a 被传递为引用，increment 修改的是原始变量
    cout << a;     // 输出：6，a 被修改
}
```

3. 传指针变量的方式

3.1 传指针 (Pass by Pointer)

- 传指针是传递变量的地址，可以通过指针在函数中访问和修改实际的值。
- **优点**：与传引用类似，但更灵活，可以使用 `nullptr` 或进行更复杂的指针操作。
- **缺点**：需要检查指针是否为空，容易引发内存管理问题。

示例：

```
void increment(int* x) {
    if (x != nullptr) {
        (*x)++;
    }
}

int main() {
    int a = 5;
    increment(&a);  // 传递 a 的地址
    cout << a;      // 输出：6，a 被修改
}
```

3.2 传指针的引用 (Pass by Reference to Pointer)

- 传递指针的引用允许修改指针本身的指向，即可以在函数内改变指针指向的对象。
- **优点**：可以更改指针的指向，特别适合动态内存分配等操作。
- **缺点**：需要管理指针的生命周期。

示例：

```
void allocate(int*& p) {
    p = new int(10);  // 修改了 p 的指向，指向了新分配的内存
}

int main() {
    int* ptr = nullptr;
    allocate(ptr);  // 函数内修改了 ptr
    cout << *ptr;   // 输出：10，ptr 指向了新分配的内存
    delete ptr;     // 释放内存
}
```

考题示例

考题示例 1:
"What is the difference between pass by value and pass by reference?"

答案:

- 传值会创建参数的副本，函数对参数的修改不会影响原始变量。
- 传引用则直接操作原始变量，函数内的修改会影响原始变量。

考题示例 2:
"What is operator precedence and how does it affect expression evaluation?"

答案:
操作符优先级决定了表达式中操作符的执行顺序。优先级较高的操作符会优先执行，除非通过括号来改变优先级。结合性决定了操作符的结合方向，如 `<<` 为左结合， `=` 为右结合。

考题示例 3:
"How does pass by pointer differ from pass by reference?"

答案:

- 传指针时传递的是变量的地址，可以通过指针访问和修改变量。
- 传引用直接操作原始变量，不能改变变量的内存地址。
  以下是 C++ 中操作符的优先级和结合性的完整表格。优先级从高到低排列，结合性指示了多个相同优先级的操作符如何结合。

优先级	运算符	描述	结合性
1	`::`	作用域解析运算符	左结合
2	`++ --`	后置递增、递减	左结合
	`()`	函数调用	左结合
	`[]`	下标访问	左结合
	`. ->`	成员访问（对象/指针）	左结合

优先级	运算符	描述	结合性
	typeid	类型信息	左结合
	const_cast dynamic_cast reinterpret_cast static_cast	类型转换运算符	右结合
3	++ --	前置递增、递减	右结合
	+ -	一元正号、负号	右结合
	! ~	逻辑非、按位非	右结合
	* &	解引用，取地址	右结合
	sizeof alignof	取大小、取对齐要求	右结合
	new new[] delete delete[]	动态内存分配与释放	右结合
4	.* ->*	指针到成员运算符	左结合
5	* / %	乘法、除法、取模	左结合
6	+ -	加法、减法	左结合
7	<< >>	移位运算符	左结合
8	< <= > >=	比较运算符	左结合
9	== !=	相等运算符	左结合
10	&	按位与	左结合
11	^	按位异或	左结合
12	`	`	按位或
13	&&	逻辑与	左结合
14	`		`
15	?:	条件运算符（三目运算符）	右结合
16	= += -= *= /= %= <<= >>= &= ^= `	=`	赋值及复合赋值运算符
17	throw	抛出异常	右结合

优先级	运算符	描述	结合性
18	,	逗号运算符	左结合

解释

1. **优先级**：优先级高的运算符会先被计算。例如，`*` 和 `/` 比 `+` 和 `-` 有更高的优先级。
2. **结合性**：如果两个运算符具有相同的优先级，结合性决定它们的计算顺序。左结合表示从左到右依次计算，右结合表示从右到左依次计算。

例如：

- 在表达式 `a + b * c` 中，乘法 `*` 的优先级比加法 `+` 高，因此先计算 `b * c`，再计算 `a + (b * c)`。
- 在表达式 `a = b = c` 中，赋值运算符 `=` 是右结合的，因此先计算 `b = c`，再将结果赋值给 `a`。