作业十一

概念题

1. 什么是泛型？具有类属特性的程序实体有哪些？什么是泛型程序设计？

```
泛型编程的核心是模板，允许定义泛型类和函数。
函数模板：定义一个函数，它可以操作多种数据类型的参数。
类模板：定义一个类，它的成员可以是任意数据类型。
模板特化：为特定的数据类型提供模板的特定实现。

泛型程序设计强调使用通用的、可重用的组件来构建软件系统。
在泛型程序设计中，代码被设计成与数据类型无关，从而能够处理任何类型的数据，只要这些数据满足某些预定义的约束或行为。
```

2. C++中实现类属函数有哪两种方式？哪一种方式更好？为什么？

```
(1)内联定义
在类定义的内部直接给出函数的实现。

(2)分离定义
在类定义的外部，即类的定义中给出函数的实现。

选择哪种方式取决于具体的应用场景
对于小型的、频繁调用的函数，内联定义好。而对于较大的、复杂的函数，分离定义可能更好
```

3. 简述C++泛型编程的意义？

```
(1)代码复用
泛型编程可以编写出可以处理多种数据类型的算法，从而提高代码的复用性。

(2)扩展性
泛型编程使得添加新的数据类型变得容易，而不需要修改现有的算法代码，只需要保证新类型满足算法所需的接口即可。

(3)减少重复代码
开发者不需要为每种数据类型编写相同的逻辑，而是可以通过泛型编程来避免这种重复。

(4)算法与数据结构的分离
泛型编程支持将算法逻辑从数据结构中分离出来，使得两者可以独立发展，但又能够无缝配合。
```

4. 为什么尽量要把模板的声明和实现都放在同一个头文件中？

```
如果模板的实现分散在多个文件中，可能会导致链接时的二义性问题。所有编译单元都需要访问到相同的模板实现，以确保一致性。

其次，模板特化通常需要访问模板的原始定义，因此将模板的定义和实现放在同一个头文件中，可以方便地进行特化。

```

5. 假设存在以下函数模板定义

```
template<class T> bool func(const T& a1, const T& a2){
    .......
}
```

请问下列函数调用是否合法？如果合法，T的类型是什么？如果不合法，为什么？

```
func("hi", "world");
```

```
不合法，"hi", "world"分别是char [3] 和char [6]类型的数据，而func模板中两个参数都为T，应该相同，
这会导致模板实例化失败
 deduced conflicting types for parameter ‘const T’ (‘char [3]’ and ‘char [6]’)
```

编程题

1. 请用类模板实现栈。栈中最多可以存放 size 个元素，其中 size 作为模板中的非类型参数，并提供以下方法：

```
bool pop(); // 栈顶元素出栈，成功返回true，失败返回false
bool push(Type x); // 将x入栈，成功返回true，失败返回false
bool is_empty(); // 栈空则返回true，否则返回false
Type & top(); // 返回栈顶元素
```

```
#include <iostream>
#include <vector>

template <typename Type, size_t size>
class FixedSizeStack {
private:
    std::vector<Type> data;
    size_t topIndex;

public:
    FixedSizeStack() : topIndex(0) {}

    bool pop() {
        if (is_empty()) {
            return false;
        }
        data.pop_back();
        --topIndex;
        return true;
    }

    bool push(const Type& x) {
        if (topIndex >= size) {
            return false;
        }
        data.push_back(x);
        ++topIndex;
        return true;
```

```
    }

    bool is_empty() const {
        return topIndex == 0;
    }

    Type& top() {
        if (is_empty()) {
            throw std::out_of_range("Stack is empty");
        }
        return data.back();
    }
};
```

2. 优先级队列是一种常用的数据结构，与先进先出队列FIFO不同，优先级队列中的元素按照权重 `weight` 排序，排在队首的元素为权重最大的元素。C++ 标准库（在头文件中）提供了优先级队列 `std::priority_queue` 的实现，但队列中元素权重的大小默认用元素之间的大小比较。
本题中，我们将优先级队列中元素的值与其权重分离，并将一个队列元素用以下模板类 `Item` 抽象

```
    template<typename value_t, typename weight_t, typename Compare =
    std::less<weight_t>>
    class Item {
         value_t v;
         weight_t w;
         public:
         Item(value_t v, weight_t w):v(v), w(w) {}
         value_t getValue() const { return v; }
         weight_t getWeight() const { return w; }
    };
```

模板中将元素的值类型 `value_t` 和权重类型 `weight_t` 都声明成模板参数，以便于我们实例化成所期望的值和权重组合。其中模板类型参数 Compare 是一个函数对象的类型，该类型的实例（不妨设为 `func_obj`）可以用于比较两个权重，即可以调用函数对象的操作符 `"( )"`，并传入两个权重值（不妨设为 `w1`,`w2`）得到 `w1` 是否小于/大于/等于 `w2` 的结果，例如 `func_obj(w1, w2)`。

操作符 `"( )"` 的原型通常声明为 `bool operator() (weight_t w1, weight_t w2)`。`Item` 的权重比较需要借助 `Compare` 类对象实现且默认为 `weight_t` 上的“小于”语义（`std::less<weight_t>`）。例如，我们假设队列元素的值为 int 类型，权重也为 int 类型，那么两个元素之间的大小由整数型权重的大小定义。

```
Item<int, int> item1(0, 1);
Item<int, int> item2(2, 0);
bool foo = item1 < item2; // false
```

上例中 `Compare` 的实际参数为 `std::less<int>`。C++标准库提供了可以实例化成类 `std::less<int>` 的模板定义，并基于 int 类的小于运算符 `<` 为 `std::less<int>` 重载了操作符 `bool operator()(int a, int b);`。故 `std::less<int>` 类对象可用于判断传入的两个 `int` 参数是否满足 `a < b`。

当然，上述 `Item` 类并不完整，我们无法直接使用 `<` 对两个元素进行比较，你需要补全它。除了单一权重外，有时我们需要比较的权重是多个维度的，即权重元组。权重元组之间的大小比较与字典序相似，n 维的元组 $t_1 = (a_1, a_2, \cdots, a_n)$ 小于 $t_2 = (b_1, b_2, \cdots, b_n)$ 当且仅当存在 $k(1 \le k \le n)$ 使得 $a_k < b_k$，且对所有的 $j(1 \le j < k)$ 有 $a_j = b_j$。例如：$(1, 2, 8) < (1, 3, 2)$。

下面给出不完整的权重元组模板类定义:

```
template<typename weight_t, int num, typename Compare = std::less<weight_t>>
class WeightTuple{

};
```

其中类型参数 `weight_t` 为单维权重的类型，非类型参数 `num` 为权重元组的维数，以及单维权重之间的比较函数类型 `Compare` 。

你的任务：补全上面两个类，使得以下代码能够成功运行，并得到对应的输出。

提示：

- 我们假设所有可能的权重类型（无论 int，float，double 还是自定义类型），它们的权重值都可以排一个全序
- 你需要观察测试代码来思考 WeightTuple 的初始化方式
- std::less 为类T的小于比较器（函数对象）的类型，需要为类T重载小于比较操作符才可使用。基本类型如 int ， double 等对应的 std::less ， std::less由C++默认提供，采用C++中运算符 < 的语义实现。
- 可以进一步思考以下问题：既然我们可以给权重类型重载 operator < 等操作符，从而用于Item 类比较的实现，为什么我们还需要在类模板中提供 Compare 参数并用函数对象来完成大小比较?

```
#include <iostream>
#include <queue>
using namespace std;
// your code
int main() {
  using QueueItem1 = Item<int, int>;
  priority_queue<QueueItem1, std::vector<QueueItem1>, std::less<QueueItem1>>
q1;
  int value[] = {10, 9, 8, 7, 6, 5, 4, 3, 2, 1};
  int weight[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
  for (size_t i = 0; i < 10; ++i) {
    QueueItem1 tmp(value[i], weight[i]);
    q1.push(tmp);
    cout << q1.top().getValue() << ", ";
  }
  cout << endl;
  while (!q1.empty()) {
    cout << q1.top().getValue() << ", ";
    q1.pop();
  }
  cout << endl;
  int weight_tuple[][2] = {{1, 10}, {1, 9}, {2, 8}, {2, 7}, {3, 6},
                           {3, 5},  {4, 4}, {4, 3}, {5, 2}, {5, 1}};
  using QueueItem2 = Item<int, WeightTuple<int, 2>>;
  priority_queue<QueueItem2, std::vector<QueueItem2>, std::less<QueueItem2>>
q2;
  for (size_t i = 0; i < 10; i++) {
    WeightTuple<int, 2> w(weight_tuple[i]);
    QueueItem2 tmp(value[i], w);
    q2.push(tmp);
    cout << q2.top().getValue() << ", ";
```

```
  }
  cout << endl;
  while (!q2.empty()) {
    cout << q2.top().getValue() << ", ";
    q2.pop();
  }
  cout << endl;
  return 0;
}
```

输出：

```
10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
10, 10, 8, 8, 6, 6, 4, 4, 2, 2,
2, 1, 4, 3, 6, 5, 8, 7, 10, 9,
```

```
template<typename value_t, typename weight_t, typename Compare =
std::less<weight_t>>
class Item {
    value_t v;
    weight_t w;
    Compare comp;

    public:
    Item(value_t v, weight_t w): v(v), w(w), comp() {}

    value_t getValue() const { return v; }
    weight_t getWeight() const { return w; }


    bool operator<(const Item& other) const {
        return comp(w, other.w);
    }
};

template<typename weight_t, int num, typename Compare = std::less<weight_t>>
class WeightTuple {
    std::array<weight_t, num> weights;
    Compare comp;

    public:

    WeightTuple(weight_t w1, weight_t w2, ...) : weights{{w1, w2, ...}} {}


    weight_t getWeight(int index) const {
        if (index < 0 || index >= num) {
            throw std::out_of_range("Index out of bounds");
        }
        return weights[index];
    }


```

```
    bool operator<(const WeightTuple& other) const {
        for (int i = 0; i < num; ++i) {
            if (comp(weights[i], other.weights[i])) {
                return true;
            } else if (comp(other.weights[i], weights[i])) {
                return false;
            }
        }
        return false;
    }
};
```

提交注意事项

截止时间：2024-5-21 23:59

文件格式：姓名-学号.pdf

提交方式：南大计科在线实验教学平台

请同学们于截止时间前在南大计科在线实验教学平台上提交，每次作业最终只需要提交一个pdf文件即可，以“姓名-学号.pdf“的方式命名。

注意：请按要求命名文件，并且只提交一个PDF文件，编程题代码请附在PDF中。任何错误的命名和文件格式将影响你的作业得分。