Пояснение:

- Реализирайте задачите спазвайки добрите ООП практики (валидация на данните, подходяща капсулация и тн.)
- Решение, в които не са спазени ООП принципите ще бъдат оценени с 0 точки.
- Предадените от вас решения трябва да могат да се компилират успешно на Visual C++ или GCC.
- **Не е разрешено** да ползвате библиотеки от STL и STL функции.

Изисквания за предаване:

- Всички задачи ще бъдат проверени автоматично за преписване. Файловете с голямо съвпадение ще бъдат проверени ръчно и при установено плагиатство ще бъдат анулирани.
- Предаване на домашното в указания срок от всеки студент като .zip архив със следното име:

  (номер_на_домашно)_SI_(курс)_(група)_(факултетен_номер)

  - (номер_на_домашно) е цяло число, отговарящо на номера на домашното за което се отнася решението (например 1);
  - (курс) е цяло число, отговарящо на курс (например 1);
  - (група) е цяло число, отговарящо на групата Ви (например 1);
  - (факултетен_номер) е цяло число, отговарящо на факултетния Ви номер (например 12345);

Пример за .zip архив за домашно: 2_SI_1_1_12345.zip

Архивът да съдържа само изходен код (.cpp и .h/.hpp файлове) с решение отговарящо на условията на задачите, като файловете изходен код за всяка задача трябва да са разположени в папка с име (номер_на_задача).

Качване на архива на посоченото място в Moodle

Задача 1 (7 точки):

Задача. Разглеждаме абстрактен базов клас *Множество*, което може да съдържа елементи от цели 32-битови числа и задължително притежава операция за проверка дали даден елемент от съответния тип принадлежи на множеството.

Да се реализират следните конкретни наследници на абстрактния базов клас *Множество*:

- *Множество по критерий* – в конструктора се подава *предикат (това е булева функция или обект, който се държи като такава)*, който по подаден ѝ като аргумент елемент решава дали той да принадлежи на множеството или не.
- *Сечение на множества* – в конструктора се подават няколко *множества* и създаденият обект (сечение) трябва да съдържа точно онези елементи, които се съдържат във всяко от посочените множества. Елементите на всички множества са от един и същ тип.
- *Обединение на множества* – в конструктора се подават няколко *множества* и създаденият обект (обединение) трябва да съдържа точно онези елементи, които се съдържат в поне едно от посочените множества. Елементите на всички множества са от един и същ тип.

Да се реализира програма, която прочита от двоичен файл `set.dat` информация за множество и конструира ново множество съгласно указаните в двоичния файл правила.

Двоичният файл има следната структура:

- две цели неотрицателни 16-битови числа N и T, където стойността на N не надхвърля 32
- стойността на T определя формата на двоичния файл по-нататък и как се конструира съответното множество, както следва:
    - 0 – следват N цели 32-битови числа, които определят крайно множество, състоящо се точно от тях
    - 1 – следват N цели 32-битови числа, които определят крайно множество, състоящо се от точно тези цели 32-битови числа, които **не се делят на нито едно** от дадените числа
    - 2 – следват N цели 32-битови числа, които определят крайно множество, състоящо се от точно тези цели 32-битови числа, които **се делят на точно едно** от дадените числа
    - 3 – следват N низа, всеки от тях терминиран с 0, които описват пътища към файлове, задаващи множества, чиито обединение представя текущото множество
    - 4 – следват N низа, всеки от тях терминиран с 0, които описват пътища към файлове, задаващи множества, чиито сечение представя текущото множество

Програмата да работи в два режима:

1. Въвежда от стандартния вход две цели числа **a** и **b** и извежда всички числа в интервала [**a**; **b**], които са в построеното множество.
2. Позволява последователно генериране на всички елементи от множеството, като всеки следващ елемент се генерира при поискване от потребителя.

Да се обработват по подходящ начин различните грешки, свързани с некоректен вход.

Пример:

set.dat	first.dat	second.dat	third.dat
3 4 first.dat second.dat third.dat	7 0 1 3 4 6 2 8 9	2 1 4 6	3 2 2 3 5

При въведени числа **a** = 0 и **b** = 10, се очаква да се изведат числата 2, 3 и 9 в някакъв ред.

Забележка: Съдържанието на двоичните файлове е показано като текст само за удобство на примера. Файловете **трябва да са двоични!**

Задача 2 (3 точки):

Имплементирайте шаблонна опашка с k приоритета. При вземане на елемент от опашката трябва да се връща **най-отдавна добавения елемент от тези с най-висок приоритет.**

Класовете, които използвате, трябва да бъдат с коректно хвърляне/обработване на изключения.

```
int main()
{
 kPriorityQueue<char> q(4); // 4 priorities - 0, 1, 2 и 3.

 q.push('A',0): //adds A with lowest priority: 0
 q.push('B',3);
 q.push('C',2);
 q.push('D',2);
 q.push('E',1);

 //q.push('F', 5); //error! No such priority!

 q.peek(); // B
```

```
  q.pop();

  q.peek(); // C
  q.pop();

  q.peek(); // D
  q.pop();

  q.peek(); // E
  q.pop();

  q.peek(); // A
  q.pop();
}
```