Вариант 1 – четни номера

Задача 1.

Да се дефинира клас IntSet, който представя множество от цели числа с максимален размер, динамично определен в конструктора. Няма изисквания към вътрешното представяне, доколкото позволява реализирането на задачата. Освен член-функции, реализиращи каноничното представяне, да се предефинират следните оператори:
+ (намира обединение на множества)
+ (добавя елемент към множество)
- (намира разлика на множества)
- (премахва елемент от множество, ако той съществува)
***** (намира сечение на множества)
^ (намира симетрична разлика)
== (две множества са равни, ако имат еднакъв набор от елементи)
Да се дефинират и методите:
bool exists(int), за проверка дали елемент принадлежи на множество
void toString(char* buffer), който да записва в буфера низ, състоящ се от елементите на множеството, разделени със запетаи
Бонус: да се дефинират и операторите +=, -=, *=, които променят текущото множество по очаквания начин**.**

Задача 2.

Реализирайте Цикличен буфер (Circular buffer). Нека неговите елементи са структури, реализирани с шаблона :

```
template <typename T>
struct elem_cyclic{
        T inf;
        elem_cyclic<T> *link;
};
```

а цикличният линеен списък е предствен с шаблона :

```
template <typename T>
class cyclicList{
        public :
                cyclicList();//създава празен списък
                ~cyclicList();//използвайте помощната функция deleteList()
                cyclicList(const cyclicList&);//използвайте помощната функция copyList()
                cyclicList& operator=(const cyclicList&);
                bool empty()const;
//член-функции за работа с итератора
                void iterStart(elem_cyclic<T>*=NULL);//установяване на итератора в началото
                elem_cyclic<T>* iter();//преместване на итератора в следваща позиция

                void insert(const T&);//включва елемент след последния елемент (в началото) на списък
                void toEnd(const T&);//включва елемент в края на списък
/*първи параметър -  указател към елемент, който трябва да изтрием, като преди това го запомним във втория параметър*/
                void deleteElem(elem_cyclic<T>*, T&);
                void print()const;
                int length()const;
        private:
                elem_cyclic<T> *start, *current;
                void copyList(const cyclicList&);//използвайте функцията toEnd()
                void deleteList();//използвайте итератора
};
```

Бонус : използвайте така реализирания цикличен списък в задачата на Флавий.

Вариант 2 – нечетни номера

Задача 1.

Да се дефинира клас Polynome, който представя полином с реални коефициенти. Степента на полинома да се определя динамично в конструктора. Да се поддържа и конструктор, който приема като параметър масив от реални числа, които служат за коефициенти на създадения полином - степента е размерът на масива. Да се реализират функциите от каноничното представяне на полинома. Да се реализират операторите:
+, -, *, /, % (сума, разлика, произведение, частно и остатък при деление на два полинома)
***** (умножение на полином с реално число. ВНИМАНИЕ: операторът трябва да работи и от двете страни, т.е. p*3 и 3*p да дават един и същи резултат)
^ (производна на полином)
() (по дадено реално число пресмята стойността на полинома в това число)
Да се дефинират и методите:
bool **isNull**(), за проверка дали полиномът е нулев
Бонус: да се дефинират и операторите +=, -=, *=, ^=, които променят полинома по очаквания начин

Задача 2.

Реализирайте Двойно свързан списък (DEQ). Нека неговите елементи са структури, реализирани със шаблона

```
template <typename T>
struct elem_link2{
   T inf;
   elem_link2<T> *pred, *succ;
};
```

а двойно свързаният линеен списък е представен с шаблона :

```
template <typename T>
class DLList{
   public:
      DLList();//създава празен списък
      ~DLList();//използвайте помощната функция deleteList()
      DLList(const DLList&);//използвайте помощната функция copyList()
      DLList& operator=(const DLList&);
      bool empty()const;
//член-функции за работа с итератор
      void iterStart(elem_link2<T>* =NULL);//установяване на currentS  в началото на списъка
      void iterEnd(elem_link2<T>* =NULL);//установяване на current в края на списъка
      elem_link2<T>* iterSucc();//преместване на currentS  в следваща позиция
      elem_link2<T>* iterPred();//преместване на current  в предходна позиция
//включване на елемент
      void toEnd(const T&);//пред първия
      void toStart(const T&);//след последния
      void deleteElem(elem_link2<T>*, T&);//изтрива елемент, сочен от указателя, запомня го в x
      void print() const;
      void printReverse()const;
      int length()const;
   private:
      elem_link2<T> *start,
      *end,
      *currentS, //итератор към началото
      *currentE;//итератор към края
      void copyList(const DLList&);//използвайте функцията toEnd()
      void deleteList();//използвайте указател от тип elem_link2<T>
};
```

Бонус : Линеен списък, съдържащ 2n цели числа a1, a2, …, a2n е представен чрез две връзки (n е дадено естествено число). Да се дефинира функция, която намира :

$S=a_1.a_{2n}+a_2.a_{2n-1}+\ldots+a_n.a_{n+1}$