Übung 12 infT

1.1
Erstellen Sie eine Klasse **`StringRingbuffer`** für Objekte, die fünf Strings aufnehmen können. Die Klassendefinition sehe wie folgt aus:

```
class StringRingbuffer {
  static const int len = 5;

  string sa[len];                // string array
  int inx;                       // input index
  int outx;                      // output index
  int entries;                   // number of current entries
public:
  StringRingbuffer();
  void reset();                  // empty buffer
  bool isEmpty();                // answer whether buffer is empty
  bool isFull();                 // answer whether buffer is full
  bool in(string& str);          // input string and answer if successful
  string out();                  // output string or ""-string if empty
  void print();                  // print state of ringbuffer
};
```

In der Klassendefinition fehlen **`const`**-Angaben. Fügen Sie solche ein, wo dies sinnvoll ist. Wo möglich, arbeiten Sie mit **`inline`**-Funktionen.
Testen Sie die Klasse mit geeigneten Aufrufen.

1.2
Testen Sie die Klasse aus **1.1** hinsichtlich des Kopierens. Führen die untenstehenden Zeilen zu den erwarteten Ergebnissen, ohne dass Sie einen Copy Constructor zur Verfügung stellen?

```
StringRingbuffer rb1, rb2;
rb1.in("string 1");
rb1.in("string 2");
// ...
rb2 = rb1;                       // copy object
// ...
StringRingbuffer rb3(rb1);       // copy object
// ...
```

1.3
Legen Sie einen **`StringRingbuffer`** auf dem Heap an. Klonen Sie das Objekt (d.h. legen Sie eine Kopie des ersten Objekts an, das ebenfalls auf dem Heap liegt). Testen Sie, ob es sich tatsächlich um ein eigenständiges Objekt handelt. Ist ein Destruktor nötig?

2.1
Legen Sie eine modifizierte **StringRingbuffer**-Klasse an, die statt mit **string**-Objekten mit **char**-Arrays im C-Stil arbeitet. Die Klassendefinition sieht nun (unvollständig!) wie folgt aus:

```
class StringRingbuffer {
  static const int len = 5;

  char *ca[len];                // array of pointers to c-strings
  int inx;                      // input index
  int outx;                     // output index
  int entries;                  // number of current entries
public:
  StringRingbuffer();
  void reset();                  // empty buffer
  bool isEmpty();                // answer whether buffer is empty
  bool isFull();                 // answer whether buffer is full
  bool in(char *cp);             // input string and answer if successful
  // copy string into arr with maxLength chars. Answer true if a string
  // was available, else false
  bool out(char arr[], int maxLength);
  void print();                  // print state of ringbuffer
};
```

Die Funktion **out** unterscheidet sich von der Aufgabe **1**. Hier werden die Adresse eines **char**-Array und die Länge des Array übergeben. In das Array hat **out** den C-String aus dem Ringbuffer zu kopieren, falls ein solcher vorhanden ist.

2.2
Testen Sie die Klasse aus **2.1** hinsichtlich des Kopierens. Führen die untenstehenden Zeilen zu den erwarteten Ergebnissen, ohne dass Sie einen Copy Constructor zur Verfügung stellen?

```
StringRingbuffer rb1;
const int len = 80;
char carr[len + 1];

rb1.in("00000");
rb1.in("1111");
rb1.out(carr, len);
rb1.in("333");
// ...
StringRingbuffer rb2(rb1);        // copy object
// ...
```

Schaffen Sie allenfalls mit einem eigenen Copy Constructor Abhilfe. Testen Sie mit Beispielen.

2.3
Wiederholen Sie die Tests aus **1.3** für Objekte der **StringRingbuffer**-Klasse aus **2.1**.