UNIVERSITÄT DUISBURG ESSEN

NES Networked Embedded Systems

Programming in C/C++

Exercises 9

Task 01:
Wireless signals are commonplace in the environment today. Provided in signals.cpp is a class modelling such signals. Complete the class and the program with the following.

- Overload the input operator >> of the `Signal` class to be able to read and parse a comma-separated string to create a Signal (see sample signals.txt file).
- Overload the output operator << of Signal class to output the signal readings in the CSV format (as shown in the sample signals.txt file).
- Implement the function `std::vector readSignals(char* filepath)` to read and parse the provided CSV file. Each line in the file represents a signal and the vector list of signals should be returned. Use the overloaded input operator!
- Implement the function `std::vector<Signal> filterSignals(SignalType type, std::vector<Signal>& )` which returns a new vector containing only signals which have the signal type specified.
- Implement the function void `writeSignals(char* filename, std::vector& signals)` to write the signals from a provided vector to a file with filename specified in the parameter. Use the overloaded output operator!

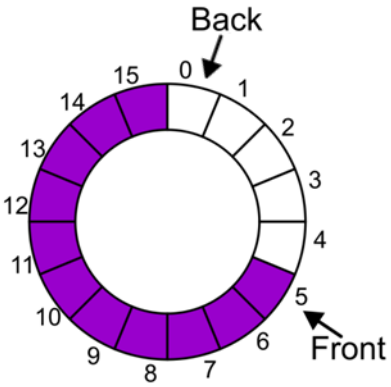
Use the provided signals.cpp file and add your code to it. The read and write functions should output a log line to the console stating what the function is doing using the template "`<operation> signals to/from <filename>`" e.g.

- `"Reading signals from signals.txt"`
- `"Writing signals to filtered_signals.txt"`

Make sure to properly comment the all the functions according to the guidelines!

Submission: Include signals.txt in your zip. The executable should be called **`signals_io`**
Points: (code 30 pts, comments 10 pts)

Task 02:
(Double ended) queues with fixed size can also be implemented as ring buffer. All elements are stored in an array using two indices "back" and "front". As shown in the figure, "front" points to the first element in the queue, "back" to the element past the last element (similar to STL iterators). Pushing to the back or popping from the front increases the back or front indices, respectively. If an index is to be increased at position 15 it restarts at position 0. Popping from the back and pushing to the front works with decreasing the indices and an analogous adjustment.



Implement a template class "`RingBuffer`" for double ended queues in "ringbuffer.hpp". The template can be parameterized with the type of objects and the maximum number of elements that can be stored in the queue, e.g. "`RingBuffer<int, 3> rb;`".
The queue shall support the operations `push_front, pop_front, push_back, pop_back, empty, full, print`.

- The pop operations should return the value from the corresponding position.
- If a push operation is called on a full queue, the object shall be dropped silently.
- If any of the 2 pop operations is called on an empty queue, a `std::range_error()` exception shall be thrown with message "`<method_name> on empty buffer`" e.g. "`pop_back on empty buffer`".
- The print method shall print out "`Buffer is empty`" if the queue is empty, otherwise it shall print "`Content of buffer:`" followed by the complete queue from front to end (separated by "`\t`"). For example:
  - `Content of buffer: 1  2  31  44`

Use an internal counter to keep track of the number of elements in the queue!
You can test your code using the rbtest.cpp file provided.

Submission: Executable should be named **`ringbuffer`**
Points: (code 50 pts, comments 10 pts)

Task 03 - Bonus:
Implement an alternative version of the `RingBuffer` that shall behave in exactly the same way as the original `RingBuffer` from a user's point of view but instead of using an internal counter it relies only on the values of "front" and "back" indices for all operations.

If you do it right, only minimal changes are required.
Create a new file called ringbuffer2.hpp and put your implementation in it. Also test with the provided rbtest.cpp file (change the `#include` statement to use ringbuffer2.hpp).

Submission: The executable should be called **ringbuffer2**
Points: (code 10 pts)

(*Side note*: In good ol' DOS days, the keyboard buffer was organized in this way, having a pointer to the last and next character without an explicit counter.)