1 Aufgabe 6

The Programm `clock.cpp` is written in `C++`. The clock algorithm is implemented the following way:

- A `page` struct, that contains page number and an R-Bit

```
struct page {
    char pnumber;
    bool R;
}
```

- A Ringbuffer, implemented by a simple `page`-Array

Output for Reference A:

```
Number of Frames (default 3): 3
Reference Set (default 70120304230321201701):
7 -> [(7), 10 , 10 ]
0 -> [ 7 ,(0), 10 ]
1 -> [ 7 , 0 ,(1)]
2 -> [(2), 0 , 1 ]
0 -> [ 2 , 0 , 1 ]
3 -> [ 2 , 0 ,(3)]
0 -> [ 2 , 0 , 3 ]
4 -> [(4), 0 , 3 ]
2 -> [ 4 , 0 ,(2)]
3 -> [(3), 0 , 2 ]
0 -> [ 3 , 0 , 2 ]
3 -> [ 3 , 0 , 2 ]
2 -> [ 3 , 0 , 2 ]
1 -> [ 3 ,(1), 2 ]
2 -> [ 3 , 1 , 2 ]
0 -> [(0), 1 , 2 ]
1 -> [ 0 , 1 , 2 ]
7 -> [ 0 , 1 ,(7)]
0 -> [ 0 , 1 , 7 ]
1 -> [ 0 , 1 , 7 ]
```

Output for Reference B:

```
Number of Frames (default 3): 3
Reference Set (default 70120304230321201701): 232152453252
2 -> [(2), 10 , 10 ]
3 -> [ 2 ,(3), 10 ]
2 -> [ 2 , 3 , 10 ]
1 -> [ 2 , 3 ,(1)]
5 -> [ 2 ,(5), 1 ]
2 -> [ 2 , 5 , 1 ]
4 -> [ 2 , 5 ,(4)]
5 -> [ 2 , 5 , 4 ]
3 -> [ 2 , 5 ,(3)]
2 -> [ 2 , 5 , 3 ]
5 -> [ 2 , 5 , 3 ]
2 -> [ 2 , 5 , 3 ]
```

For the LRU algorithm, we expand our data structure to include a string where the R-Bits are saved, and a counter that can be accessed when a page conflict is taking place (this is

usually handled by the OS, but implementing it here in the Data structure made things much more simple to program).
Output for Reference A (Epoch every 3rd step):

```
Number of Frames (default 3):
Reference Set (default 70120304230321201701):
7 -> [ 10 , 10 ,(7)]
0 -> [ 10 ,(0), 7 ]
1 -> [(1), 0 , 7 ]
2 -> [ 1 , 0 ,(2)]
0 -> [ 1 , 0 , 2 ]
3 -> [ 1 ,(3), 2 ]
0 -> [(0), 3 , 2 ]
4 -> [ 0 , 3 ,(4)]
2 -> [ 0 , 3 ,(2)]
3 -> [ 0 , 3 , 2 ]
0 -> [ 0 , 3 , 2 ]
3 -> [ 0 , 3 , 2 ]
2 -> [ 0 , 3 , 2 ]
1 -> [ 0 , 3 ,(1)]
2 -> [ 0 , 3 ,(2)]
0 -> [ 0 , 3 , 2 ]
1 -> [ 0 ,(1), 2 ]
7 -> [(7), 1 , 2 ]
0 -> [ 7 , 1 ,(0)]
1 -> [ 7 , 1 , 0 ]
```

Output for Reference B (Epoch every 3rd step):

```
Number of Frames (default 3):
Reference Set (default 70120304230321201701): 232152453252
2 -> [ 10 , 10 ,(2)]
3 -> [ 10 ,(3), 2 ]
2 -> [ 10 , 3 , 2 ]
1 -> [(1), 3 , 2 ]
5 -> [ 1 ,(5), 2 ]
2 -> [ 1 , 5 , 2 ]
4 -> [ 1 ,(4), 2 ]
5 -> [(5), 4 , 2 ]
3 -> [ 5 , 4 ,(3)]
2 -> [ 5 , 4 ,(2)]
5 -> [ 5 , 4 , 2 ]
2 -> [ 5 , 4 , 2 ]
```

Aufgabe 7

In this approach, a ringbuffer that stores Δ pagenumbers was used. That way, only the last Δ pagenumbers are stored. If a new page is accessed, the oldest entry in the buffer gets deleted. Pseudocode looks like this:

```
int main(){

    int WS;

    // Enter how many memory accesses should be used for the working set
    read_input(pWS);
```

```
    // Initialize buffer
    int ringbuffer[WS];

    // Read line by line into the buffer. Once its full, overwrite the oldest
    // entry.
    int counter = 0;
    while (not done) {
        ringbuffer[counter] = line;
        counter++;
        if ( counter == WS ) { counter = 0; }
    }

    // Print to console
    for (int i = 0; i < sizeof(ringbuffer); i++) {
        if (ringbuffer[i] has been printed) continue;
        else print(ringbuffer[i]);

    return 0;
}
```

An implementation of this can be found in the file `getWorkingSet.cpp` This program reads the Working Set size from stdin, and prints out the pages the working set contains. With $\Delta = 5$, the working set of each reference string has 3 unique pages. Increasing Δ to 6 adds another entry to the set:

A

```
Enter size of working set:
5
Working set:
0
1
7
```

```
Enter size of working set:
6
Working set:
0
1
2
7
```

B

```
Enter size of working set:
5
Working set:
5
2
3
```

```
Enter size of working set:
6
Working set:
4
5
3
2
```