# Lab5TDDExample

0.1.0

Generated by Doxygen 1.9.6

# Chapter 1

# Class Index

## 1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 2

# File Index

## 2.1 File List

Here is a list of all documented files with brief descriptions:

# Chapter 3

# Class Documentation

## 3.1 Buffer< T, N > Class Template Reference

allows to store data with ability to retrieve last N stored items.

`#include <Buffer.h>`

### Public Member Functions

- void add (T value)

  *add item to the buffer.*
- std::array< T, N > output ()

  *show contents of the buffer In order they were added (last - first).*
- void clean ()

  *empty the buffer.*
- void **add** (T value)
- std::array< T, N > **output** ()
- void **clean** ()

### 3.1.1 Detailed Description

**template<typename T, int N>**
**class Buffer< T, N >**

allows to store data with ability to retrieve last N stored items.

**Template Parameters**

| | |
|---|---|
| *T* | – type of items that are stored in buffer. |
| *N* | – size of a buffer. |

### 3.1.2 Member Function Documentation

#### 3.1.2.1 add()

```
template<typename T , int N>
void Buffer< T, N >::add (
            T value )  [inline]
```

add item to the buffer.

**Parameters**

| | |
|---|---|
| *value* | – item to be added. |

#### 3.1.2.2 clean()

```
template<typename T , int N>
void Buffer< T, N >::clean  [inline]
```

empty the buffer.

#### 3.1.2.3 output()

```
template<typename T , int N>
std::array< T, N > Buffer< T, N >::output  [inline]
```

show contents of the buffer In order they were added (last - first).

**Returns**

std::array<T,N> – resulting array.

The documentation for this class was generated from the following files:

- Buffer.h
- Buffer_refactoring.h

## 3.2 Buffer_bad< N > Class Template Reference

### Public Member Functions

- void **add** (int value)
- int * **output** ()
- void **clean** ()

## Public Attributes

- int **array** [N]
- unsigned long **next**

The documentation for this class was generated from the following file:

- Buffer_refactoring.h

# Chapter 4

# File Documentation

## 4.1 Buffer.h

```
00001 #pragma once
00002 #include <array>
00003
00010 template<typename T, int N>
00011 class Buffer
00012 {
00013     std::array<T,N> array;
00014     unsigned long next;
00015
00016 public:
00017     Buffer();
00023     void add(T value);
00029     std::array<T,N> output();
00034     void clean();
00035 };
00036
00037 template <typename T, int N>
00038 inline Buffer<T, N>::Buffer()
00039 {
00040     clean();
00041 }
00042
00043 template <typename T, int N>
00044 inline void Buffer<T, N>::add(T value)
00045 {
00046     array[next] = value;
00047     next++;
00048     if (next >= array.size()) next = 0;
00049 }
00050
00051 template <typename T, int N>
00052 inline std::array<T, N> Buffer<T, N>::output()
00053 {
00054     std::array<T, N> output;
00055
00056     for(unsigned int i = 0, j = next; i < N; i++)
00057     {
00058         output[i] = array[j];
00059         j++;
00060         if(j >= array.size()) j = 0;
00061     }
00062
00063     return output;
00064 }
00065
00066 template <typename T, int N>
00067 inline void Buffer<T, N>::clean()
00068 {
00069     for(unsigned int i = 0; i < N; i++)
00070     {
00071         array[i] = T();
00072     }
00073     next = 0;
00074 }
```

## 4.2 Buffer_refactoring.h

```
00001 #pragma once
00002 #include <array>
00003
00004 template<int N>
00005 class Buffer_bad
00006 {
00007 public:
00008     int array[N];
00009     unsigned long next;
00010     Buffer_bad();
00011     void add(int value);
00012     int* output();
00013     void clean();
00014 };
00015
00016 template <int N>
00017 inline Buffer_bad<N>::Buffer_bad()
00018 {
00019     for(unsigned int i = 0; i < N; i++)
00020     {
00021         array[i] = T();
00022     }
00023     next = 0;
00024 }
00025
00026 template <int N>
00027 inline void Buffer_bad<N>::add(int value)
00028 {
00029     array[next] = value;
00030     next++;
00031     if (next >= array.size()) next = 0;
00032 }
00033
00034 template <int N>
00035 inline int* Buffer_bad<N>::output(){
00036     T output[N];
00037
00038     for(unsigned int i = 0, j = next; i < N; i++)
00039     {
00040         output[i] = array[j];
00041         j++;
00042         if(j >= array.size()) j = 0;
00043     }
00044
00045     return &output[0];
00046 }
00047
00048 template <int N>
00049 inline void Buffer_bad<N>::clean()
00050 {
00051     for(unsigned int i = 0; i < N; i++)
00052     {
00053         array[i] = T();
00054     }
00055     next = 0;
00056 }
00057
00058
00059 //The code above has multiple problems:
00060 //- Primitive Obsession (usage of c style arrays)
00061 //- Indecent Exposure (there is no need to access array and next from outside)
00062 //- Dublicated Code (in constructor and clean for example)
00063 //- Alternative Classes (potentially, if we want to store other types inside of container)
00064
00065 //Refactored class will look like this
00066
00067 #pragma once
00068 #include <array>
00069
00070 template<typename T, int N>
00071 class Buffer
00072 {
00073 private:
00074     std::array<T,N> array;
00075     unsigned long next;
00076
00077 public:
00078     Buffer();
00079     void add(T value);
00080     std::array<T,N> output();
00081     void clean();
00082
00083 private:
00084     unsigned long _add_n_fold(unsigned long x);
00085 };
```

```
00086
00087 template <typename T, int N>
00088 inline Buffer<T, N>::Buffer()
00089 {
00090     clean();
00091 }
00092
00093 template <typename T, int N>
00094 inline void Buffer<T, N>::add(T value)
00095 {
00096     array[next] = value;
00097     _add_n_fold(next);
00098 }
00099
00100 template <typename T, int N>
00101 inline std::array<T, N> Buffer<T, N>::output()
00102 {
00103     std::array<T, N> output;
00104
00105     for(unsigned int i = 0, j = next; i < N; i++)
00106     {
00107         output[i] = array[j];
00108         _add_n_fold(j);
00109     }
00110
00111     return output;
00112 }
00113
00114 template <typename T, int N>
00115 inline void Buffer<T, N>::clean()
00116 {
00117     for(unsigned int i = 0; i < N; i++)
00118     {
00119         array[i] = T();
00120     }
00121     next = 0;
00122 }
00123
00124 template <typename T, int N>
00125 inline unsigned long Buffer<T, N>::_add_n_fold(unsigned long x)
00126 {
00127     auto out = x++;
00128     if(x >= array.size()) x = 0;
00129     return x;
00130 }
```