proj3

Generated by Doxygen 1.12.0

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 files with brief descriptions:

Chapter 3

Class Documentation

3.1 MergeSorter< T > Class Template Reference

The class inside this template is responsible for sorting vectors using the merge sort algorithm.

`#include <MergeSorter.hpp>`

Public Member Functions

- void operator() (std::vector< T > &toSort)
  Will sort any vector inserted into the function using the merge sort algorithm.

3.1.1 Detailed Description

template<typename T>
class MergeSorter< T >

The class inside this template is responsible for sorting vectors using the merge sort algorithm.

The sorting functionality can be accessed with `operator()()`

3.1.2 Member Function Documentation

3.1.2.1 operator()()

```
template<typename T >
void MergeSorter< T >::operator() (
            std::vector< T > & toSort)  [inline]
```

Will sort any vector inserted into the function using the merge sort algorithm.

This operation happens recursively and IS NOT in place

Parameters

toSort	The vector that is to be sorted

Chapter 4

File Documentation

4.1 src/main.cpp File Reference

```
#include <print>
```

Functions

- int main (int argc, char ∗argv[ ])

4.1.1 Function Documentation

4.1.1.1 main()

```
int main (
            int argc,
            char * argv[ ])
```

4.2 src/MergeSorter.hpp File Reference

```
#include <ranges>
#include <vector>
```

Classes

- class MergeSorter< T >

  The class inside this template is responsible for sorting vectors using the merge sort algorithm.

4.3 MergeSorter.hpp

Go to the documentation of this file.

```
00001 #include <ranges>
00002 #include <vector>
00003
00007 template<typename T>
00008 class MergeSorter {
00009 private:
00010     std::vector<T> mergeSort(const std::vector<T>& toMerge) {
00011         if(toMerge.size() <= 1) {
00012             return toMerge;
00013         }
00014
00015         auto left = toMerge | std::views::take(toMerge.size() / 2) | std::ranges::to<std::vector>();
00016         auto right = toMerge | std::views::drop(toMerge.size() / 2) | std::ranges::to<std::vector>();
00017
00018         auto sortedLeft = mergeSort(left);
00019         auto sortedRight = mergeSort(right);
00020
00021         return merge(sortedLeft, sortedRight);
00022     }
00023
00024     std::vector<T> merge(const std::vector<T>& left, const std::vector<T>& right) {
00025         std::vector<T> merged;
00026         auto leftIt { left.begin() }, rightIt { right.begin() };
00027
00028         for (; leftIt != left.end() && rightIt != right.end();) {
00029             if(*leftIt <= *rightIt) {
00030                 merged.push_back(*leftIt);
00031                 leftIt++;
00032             } else {
00033                 merged.push_back(*rightIt);
00034                 rightIt++;
00035             }
00036         }
00037
00038         merged.insert(merged.end(), leftIt, left.end());
00039         merged.insert(merged.end(), rightIt, right.end());
00040
00041         return merged;
00042     }
00043
00044 public:
00048     void operator()(std::vector<T>& toSort) {
00049         if(toSort.size() <= 1) {
00050             return;
00051         }
00052
00053         toSort = mergeSort(toSort);
00054     }
00055 };
```

Index