CSearches

Generated by Doxygen 1.9.2

Chapter 1

EECS 114 Searches and Benchmarking

This Project, as a part of EECS 114 at UCI tries to implement many common searches in C.
It Implements the following searches as well as more to come

BinarySearch
LinearSearch

The project also defines a way to log and benchmark searches. It creates generalized searches and provides structures for testing and timing.

Search
TimeSearch
SearchData
BenchmarkSearches
BenchmarkSearch

It also Provides Common Utilities for Other Users and Use Cases

Utils.h

Author

Orion Serup (orionserup@gmail.com)

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Data Structure Documentation

4.1 Search Struct Reference

Structure to Hold The Function and Name of a Search.

`#include <Utils.h>`

Data Fields

- const SearchFunction function

  The Actual Function that does the searching, matches the profile above.
- const char *const name

  The Name of the Search.

4.1.1 Detailed Description

Structure to Hold The Function and Name of a Search.

Definition at line 42 of file Utils.h.

4.1.2 Field Documentation

4.1.2.1 function

`const SearchFunction Search::function`

The Actual Function that does the searching, matches the profile above.

Definition at line 44 of file Utils.h.

4.1.2.2 name

```
const char* const Search::name
```

The Name of the Search.

Definition at line 45 of file Utils.h.

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

- Utils.h

4.2 SearchData Struct Reference

```
#include <Benchmark.h>
```

Data Fields

- size_t n
- double time_ms
- size_t index

  Where the Value was Found.

4.2.1 Detailed Description

Definition at line 33 of file Benchmark.h.

4.2.2 Field Documentation

4.2.2.1 index

```
size_t SearchData::index
```

Where the Value was Found.

Definition at line 37 of file Benchmark.h.

4.2.2.2 n

```
size_t SearchData::n
```

Definition at line 35 of file Benchmark.h.

4.2.2.3 time_ms

```
double SearchData::time_ms
```

Definition at line 36 of file Benchmark.h.

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

- Benchmark.h

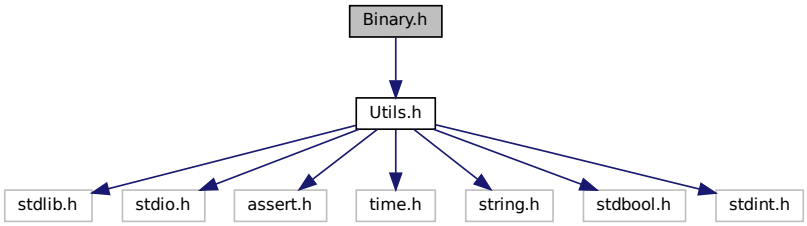
Chapter 5

File Documentation

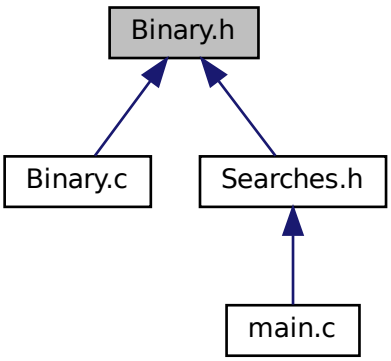
5.1 Binary.h File Reference

```
#include "Utils.h"
```

Include dependency graph for Binary.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define BINARY_SEARCH (Search){ BinarySearch, "BinarySearch" }

  Global Binary Search Literal.

Functions

- size_t BinarySearch (const Data *const array, const size_t size, const Data value)

  Finds the index of the given value in the array using the binary search algorithm.

5.1.1 Detailed Description

Author

Orion Serup (orionserup@gmail.com)

Version

0.1

Date

2022-01-17

Copyright

Copyright (c) 2022

Definition in file Binary.h.

5.1.2 Macro Definition Documentation

5.1.2.1 BINARY_SEARCH

```
#define BINARY_SEARCH (Search){ BinarySearch, "BinarySearch" }
```

Global Binary Search Literal.

Definition at line 26 of file Binary.h.

5.1.3 Function Documentation

5.1.3.1 BinarySearch()

```
size_t BinarySearch (
            const Data *const array,
            const size_t size,
            const Data value )
```

Finds the index of the given value in the array using the binary search algorithm.

Note

Array must be presorted

Parameters

array	Array to Search for the value in
size	The Size of the Array
value	The Value to Look For

Returns

size_t: The Index where the data was found: size if not found

Definition at line 15 of file Binary.c.

5.2 Binary.h

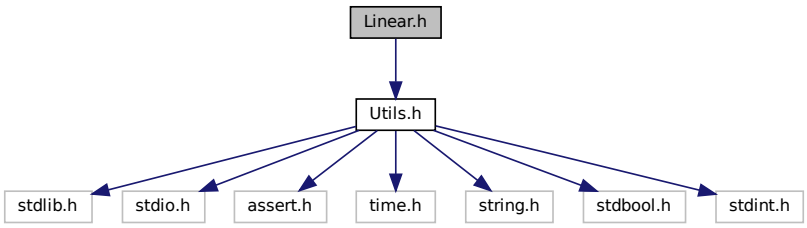
Go to the documentation of this file.

```
00001
00011 #include "Utils.h"
00012
00013 #ifndef BINARY_SEARCH
00014
00023 size_t BinarySearch(const Data* const array, const size_t size, const Data value);
00024
00026 #define BINARY_SEARCH (Search){ BinarySearch, "BinarySearch" }
00027
00028 #endif
```

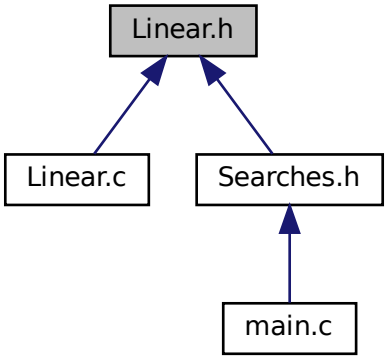
5.3 Linear.h File Reference

`#include "Utils.h"`

Include dependency graph for Linear.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define LINEAR_SEARCH (Search){ LinearSearch, "LinearSearch" }
  Global Linear Search Literal.

Functions

- size_t LinearSearch (const Data *const array, const size_t size, const Data value)
  Uses the Linear Search Algorithm To Find the Value in the array.

5.3.1 Detailed Description

Author

Orion Serup (orionserup@gmail.com)

Version

0.1

Date

2022-01-18

Copyright

Copyright (c) 2022

Definition in file Linear.h.

5.3.2 Macro Definition Documentation

5.3.2.1 LINEAR_SEARCH

```
#define LINEAR_SEARCH (Search){ LinearSearch, "LinearSearch" }
```

Global Linear Search Literal.

Definition at line 27 of file Linear.h.

5.3.3 Function Documentation

5.3.3.1 LinearSearch()

```
size_t LinearSearch (
            const Data *const array,
            const size_t size,
            const Data value )
```

Uses the Linear Search Algorithm To Find the Value in the array.

Parameters

array	Array to search
size	Size of the array
value	Value to look for

Returns

size_t: The Index of the Value: size if not found

Definition at line 15 of file Linear.c.

5.4 Linear.h

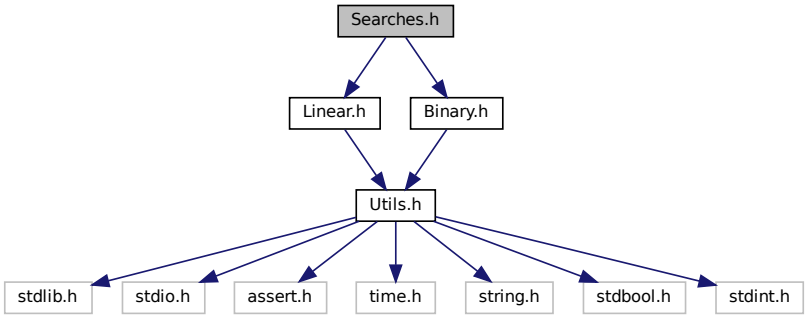
Go to the documentation of this file.

```
00001
00012 #include "Utils.h"
00013
00014 #ifndef LINEAR_SEARCH
00015
00024 size_t LinearSearch(const Data* const array, const size_t size, const Data value);
00025
00027 #define LINEAR_SEARCH (Search){ LinearSearch, "LinearSearch" }
00028
00029 #endif
00030
00031
00032
```

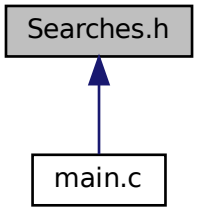
5.5 Searches.h File Reference

```
#include "Linear.h"
#include "Binary.h"
```

Include dependency graph for Searches.h:



This graph shows which files directly or indirectly include this file:



5.5.1 Detailed Description

Author

Orion Serup (orionserup@gmail.com)

Version

0.1

Date

2022-01-19

Copyright

Copyright (c) 2022

Definition in file Searches.h.

5.6 Searches.h

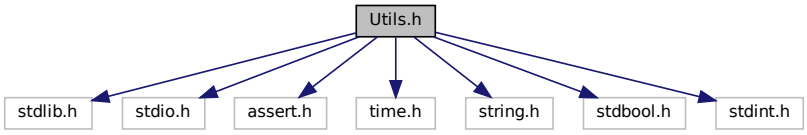
Go to the documentation of this file.

```
00001
00012 #ifndef SEARCHES
00013 #define SEARCHES
00014
00041 #include "Linear.h"
00042 #include "Binary.h"
00043
00044 #endif
```

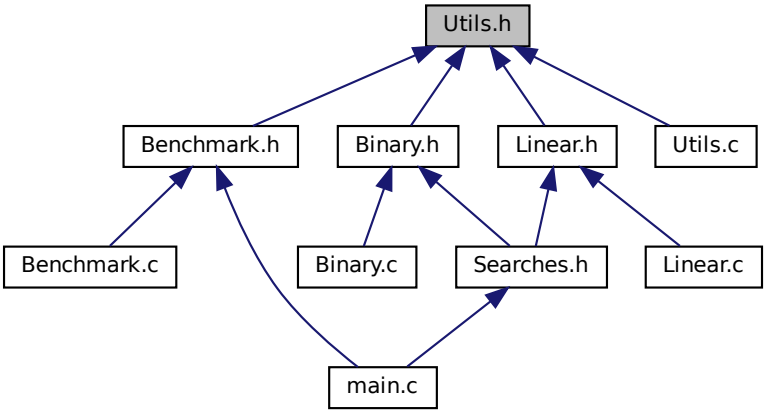
5.7 Utils.h File Reference

```
#include <stdlib.h>
#include <stdio.h>
#include <assert.h>
#include <time.h>
#include <string.h>
#include <stdbool.h>
#include <stdint.h>
```

Include dependency graph for Utils.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct Search

  Structure to Hold The Function and Name of a Search.

Macros

- #define Assert(condition, msg) assert(condition || !fprintf(stderr, "\t%s on Line: %d of File %s\n", msg, __LINE__, __FILE__))

  Implementation of any other OS, no Colors but otherwise the same.
- #define Max(array, size) array[FindMax(array, size)]

  Finds the Minimum Value of an Array.
- #define Min(array, size) array[FindMin(array, size)]

  Finds the Minimum Value of an Array.

Typedefs

- typedef uint64_t Data
- typedef size_t(*const SearchFunction) (const Data *const array, const size_t size, const Data value)

Functions

- size_t FindMax (const Data *const array, const size_t size)

  Finds the Index of the Maximum Value in the.
- size_t FindMin (const Data *const array, const size_t size)

  Finds the Minimum Element of an Array and returns its index.

5.7.1 Macro Definition Documentation

5.7.1.1 Assert

```
#define Assert(
              condition,
              msg ) assert(condition || !fprintf(stderr, "\t%s on Line:  %d of File %s\n", msg,
__LINE__, __FILE__))
```

Implementation of any other OS, no Colors but otherwise the same.

Definition at line 36 of file Utils.h.

5.7.1.2 Max

```
#define Max(
              array,
              size ) array[FindMax(array, size)]
```

Finds the Minimum Value of an Array.

Definition at line 59 of file Utils.h.

5.7.1.3 Min

```
#define Min(
              array,
              size ) array[FindMin(array, size)]
```

Finds the Minimum Value of an Array.

Definition at line 71 of file Utils.h.

5.7.2 Typedef Documentation

5.7.2.1 Data

```
typedef uint64_t Data
```

Definition at line 25 of file Utils.h.

5.7.2.2 SearchFunction

```
typedef size_t(*const SearchFunction) (const Data *const array, const size_t size, const Data
value)
```

Definition at line 39 of file Utils.h.

5.7.3 Function Documentation

5.7.3.1 FindMax()

```
size_t FindMax (
             const Data *const array,
             const size_t size )
```

Finds the Index of the Maximum Value in the.

Parameters

array	Array to Look into
size	Size of the Array

Returns

size_t: The Index of the Maximum Value in the Array

Definition at line 14 of file Utils.c.

5.7.3.2 FindMin()

```
size_t FindMin (
            const Data *const array,
            const size_t size )
```

Finds the Minimum Element of an Array and returns its index.

Parameters

array	The Array to Look Through
size	The Size of the Array

Returns

size_t: The Index of the Minimum Value

Definition at line 27 of file Utils.c.

5.8 Utils.h

Go to the documentation of this file.

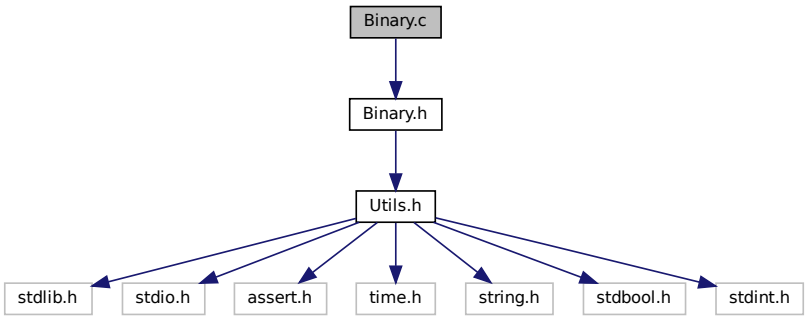
```
00001
00012 #ifndef SEARCH_UTILS
00013 #define SEARCH_UTILS
00014
00015 #include <stdlib.h>
00016 #include <stdio.h>
00017 #include <assert.h>
00018 #include <time.h>
00019 #include <string.h>
00020 #include <stdbool.h>
00021 #include <stdint.h>
00022
00023 #ifndef SEARCH_DATA
00024 #pragma message "Please Define SEARCH_DATA to the Appropriate data type"
00025 typedef uint64_t Data;
00026 #else
00027 typedef SEARCH_DATA Data;
00028 #endif
00029
00030 // Fancy Assert For Much Easier Debug, Prints Custom Message, Line Number and File Name in Red
00031 #ifdef __unix__
```

```
00033 #define Assert(condition, msg) assert(condition || !fprintf(stderr, "\x1B[31m%s on Line: %d of File
       %s\n\x1B[0m", msg, __LINE__, __FILE__))
00034 #else
00036 #define Assert(condition, msg) assert(condition || !fprintf(stderr, "\t%s on Line: %d of File %s\n",
       msg, __LINE__, __FILE__))
00037 #endif
00038
00039 typedef size_t (*const SearchFunction)(const Data* const array, const size_t size, const Data value);
00040
00042 typedef struct {
00043
00044     const SearchFunction function;
00045     const char* const name;
00046
00047 } Search;
00048
00056 size_t FindMax(const Data* const array, const size_t size);
00057
00059 #define Max(array, size) array[FindMax(array, size)]
00060
00068 size_t FindMin(const Data* const array, const size_t size);
00069
00071 #define Min(array, size) array[FindMin(array, size)]
00072
00073 #endif
```

5.9 Binary.c File Reference

Contains the Implementation of the Binary Search Algorithm in C.

`#include "Binary.h"`
Include dependency graph for Binary.c:



Functions

- size_t BinarySearch (const Data *const array, const size_t size, const Data value)

  Finds the index of the given value in the array using the binary search algorithm.

5.9.1 Detailed Description

Contains the Implementation of the Binary Search Algorithm in C.

Author

Orion Serup (orionserup@gmail.com)

Version

0.1

Date

2022-01-18

Copyright

Definition in file Binary.c.

5.9.2 Function Documentation

5.9.2.1 BinarySearch()

```
size_t BinarySearch (
            const Data *const array,
            const size_t size,
            const Data value )
```

Finds the index of the given value in the array using the binary search algorithm.

Note

Array must be presorted

Parameters

array	Array to Search for the value in
size	The Size of the Array
value	The Value to Look For

Returns

size_t: The Index where the data was found: size if not found

Definition at line 15 of file Binary.c.

5.10 Binary.c

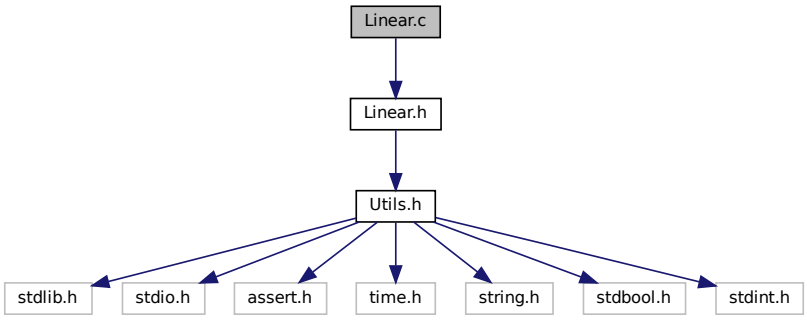
Go to the documentation of this file.

```
00001
00012 #include "Binary.h"
00013
00014 // Finds The value value in the array and returns its index if found
00015 size_t BinarySearch(const Data* const array, const size_t size, const Data value) {
00016
00017     Assert(array, "Invalid Array in Binary Search");
00018
00019     if(size == 1) // if the array is only one big if that element is the value then return its index
00020         return array[0] == value? 0: SIZE_MAX; // otherwise return Not Found
00021
00022     size_t middle = size / 2; // find the middle index
00023     Data middleval = array[middle]; // find the middle value
00024
00025     if(middleval == value)
00026         return middle; // if the middle value matches the value return the middle index
00027     else if(middleval < value)  // If we are lower than the value we look to the right
00028         return BinarySearch(array + middle, size - middle, value) + middle; // the value is on the
      right, search the right
00029     else
00030         return BinarySearch(array, middle, value); // the value is on the left, search the left
00031
00032 }
```

5.11 Linear.c File Reference

Contains the Implementation of the Linear Search in C.

`#include "Linear.h"`
Include dependency graph for Linear.c:



Functions

- size_t LinearSearch (const Data *const array, const size_t size, const Data value)

  Uses the Linear Search Algorithm To Find the Value in the array.

5.11.1 Detailed Description

Contains the Implementation of the Linear Search in C.

Author

Orion Serup (orionserup@gmail.com)

Version

0.1

Date

2022-01-18

Copyright

Copyright (c) 2022

Definition in file Linear.c.

5.11.2 Function Documentation

5.11.2.1 LinearSearch()

```
size_t LinearSearch (
            const Data *const array,
            const size_t size,
            const Data value )
```

Uses the Linear Search Algorithm To Find the Value in the array.

Parameters

array	Array to search
size	Size of the array
value	Value to look for

Returns

size_t: The Index of the Value: size if not found

Definition at line 15 of file Linear.c.

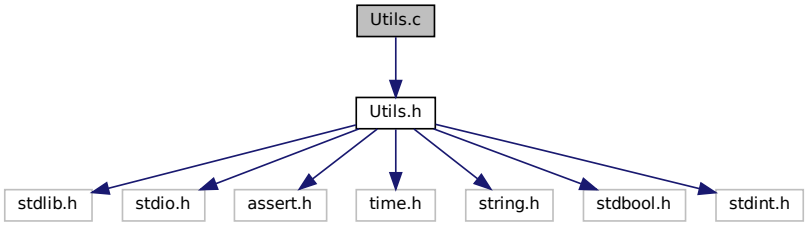
5.12 Linear.c

Go to the documentation of this file.

```
00001
00012 #include "Linear.h"
00013
00014 // Searches the Array Using the Linear Search Algorithm
00015 size_t LinearSearch(const Data* const array, const size_t size, const Data value) {
00016
00017     Assert(array, "Invalid Array in the Linear Search"); // make sure we have a valid array
00018
00019     for(size_t i = 0; i < size; i++) // go through each element in the array
00020         if(array[i] == value) // if the current value matches
00021             return i; // return the index of the value
00022
00023     return SIZE_MAX; // if the value isn't found return the max size
00024
00025 }
```

5.13 Utils.c File Reference

`#include "Utils.h"`
Include dependency graph for Utils.c:



Functions

- size_t FindMax (const Data *const array, const size_t size)
  Finds the Index of the Maximum Value in the.
- size_t FindMin (const Data *const array, const size_t size)
  Finds the Minimum Element of an Array and returns its index.

5.13.1 Function Documentation

5.13.1.1 FindMax()

```
size_t FindMax (
            const Data *const array,
            const size_t size )
```

Finds the Index of the Maximum Value in the.

Parameters

array	Array to Look into
size	Size of the Array

Returns

size_t: The Index of the Maximum Value in the Array

Definition at line 14 of file Utils.c.

5.13.1.2 FindMin()

```
size_t FindMin (
            const Data *const array,
            const size_t size )
```

Finds the Minimum Element of an Array and returns its index.

Parameters

array	The Array to Look Through
size	The Size of the Array

Returns

size_t: The Index of the Minimum Value

Definition at line 27 of file Utils.c.

5.14 Utils.c

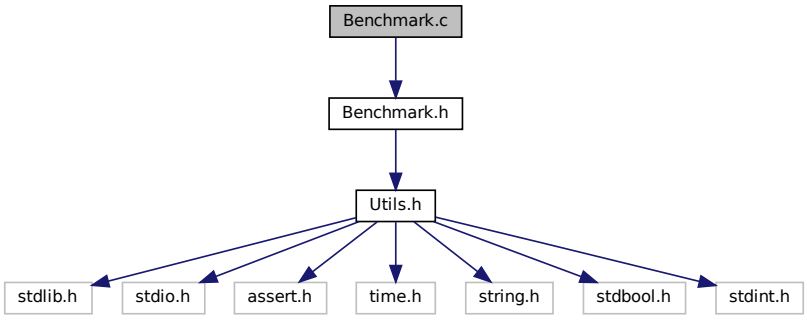
Go to the documentation of this file.

```
00001
00012 #include "Utils.h"
00013
00014 size_t FindMax(const Data* const array, const size_t size) {
00015
00016     Assert(array, "Invalid Array in Find Max");
00017
00018     size_t index = 0; // track the highest value
00019     for(size_t i = 1; i < size; i++)
00020         if(array[i] > array[index])
00021             index = i;
00022
00023     return index;
00024
00025 }
00026
00027 size_t FindMin(const Data* const array, const size_t size) {
00028
00029     Assert(array, "Invalid Array in Find Max");
00030
00031     size_t index = 0; // track the index with the minimum value
```

```
00032     for(size_t i = 1; i < size; i++)
00033         if(array[i] < array[index])
00034             index = i; // if we find a lower value store the index
00035
00036     return index; // return the Index of the Lowest Value
00037
00038 }
00039
```

5.15 Benchmark.c File Reference

`#include "Benchmark.h"`
Include dependency graph for Benchmark.c:



Functions

- SearchData TimeSearch (const SearchFunction search, const Data *const array, const size_t size, const Data value)
- void BenchmarkSearch (const Search search, const Data *const trials, const size_t numtrials, const size_t numtimes)
- void BenchmarkSearches (const Search *const searches, const size_t numsearches, const Data *const trials, const size_t numtrials, const size_t numtimes)
- void fPrintArray (FILE *const file, const Data *const array, const size_t size)

  Prints an Array to a File.

5.15.1 Function Documentation

5.15.1.1 BenchmarkSearch()

```
void BenchmarkSearch (
            const Search search,
            const Data *const trials,
            const size_t numtrials,
            const size_t numtimes )
```

Parameters

search	
trials	
numtrials	
numtimes	

Definition at line 27 of file Benchmark.c.

5.15.1.2 BenchmarkSearches()

```
void BenchmarkSearches (
            const Search *const searches,
            const size_t numsearches,
            const Data *const trials,
            const size_t numtrials,
            const size_t numtimes )
```

Parameters

searches	
numsearches	
trials	
numtrials	
numtimes	

Definition at line 115 of file Benchmark.c.

5.15.1.3 fPrintArray()

```
void fPrintArray (
            FILE *const file,
            const Data *const array,
            const size_t size )
```

Prints an Array to a File.

Note

File can Also be stdout, see PrintArray

Parameters

array	Array to Print
size	Size of the Array

Definition at line 135 of file Benchmark.c.

5.15.1.4 TimeSearch()

```
SearchData TimeSearch (
            const SearchFunction search,
            const Data *const array,
            const size_t size,
            const Data value )
```

Parameters

search	The Function Pointer of the Search
array	
size	
value	

Returns

SearchData

Definition at line 16 of file Benchmark.c.

5.16 Benchmark.c

Go to the documentation of this file.

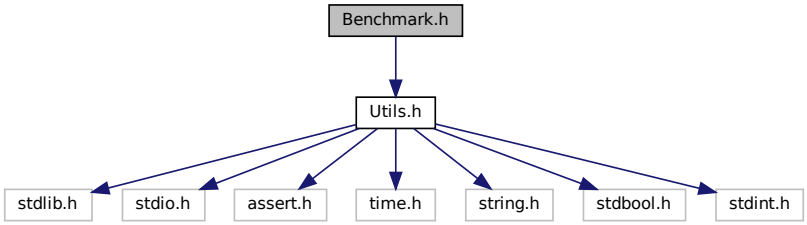
```
00001
00012 #include "Benchmark.h"
00013
00014 static void WriteSearchDataToFile(FILE* const file, const SearchData* const data, const size_t size);
00015
00016 SearchData TimeSearch(const SearchFunction search, const Data* const array, const size_t size, const
       Data value) {
00017
00018     clock_t begin = clock();
00019     size_t index = search(array, size, value);
00020     clock_t end = clock();
00021
00022     return (SearchData){ .n = size, .time_ms = (end - begin)*1000.0f/CLOCKS_PER_SEC, .index = index };
00023
00024 }
00025
00026 // Benchmark One Sort
00027 void BenchmarkSearch(const Search search, const Data* const trials, const size_t numtrials, const
       size_t numtimes) {
00028
00029     #ifndef _MSC_VER
00030     const char path[] = "../data/";
00031     #else
00032     const char* const path = "..\\..\\data";
00033     #endif
00034
00035     char buffer[100];
00036
00037     strcpy(buffer, path);
00038     strcat(buffer, search.name);
00039     strcat(buffer, ".csv");
00040
00041     FILE* const file = fopen(buffer, "w");
00042     const size_t size = Max(trials, numtrials);
00043
```

```
00044     #ifdef VLA
00045     SearchData data[numtrials * numtimes];
00046     Data array[size];
00047     #else
00048     SearchData* data = malloc(sizeof(SearchData) * numtrials * numtimes);
00049     Data* array = malloc(sizeof(Data) * size);
00050     #endif
00051 
00052     #ifdef LOG
00053     const size_t offset = sizeof(path) + strlen(search.name);
00054     strcpy(buffer + offset - 1, ".log");
00055     FILE* log = fopen(buffer, "w");
00056     fprintf(log, "%s Log\n\n", search.name);
00057     #endif
00058 
00059     srand(clock());
00060 
00061     for(size_t i = 0; i < numtimes; i++) {
00062 
00063         Data start = (Data)rand();
00064         while((start + Max(trials, numtrials)) > maxof(Data))
00065             start = (Data)rand(); // make sure we find a good, random starting point
00066 
00067         for(Data i = 0; i < Max(trials, numtrials); i++)
00068             array[i] = start + i;
00069 
00070         for(size_t j = 0; j < numtrials; j++) {
00071 
00072             Data value = start + rand() % trials[j];
00073             data[j + i * numtrials] = TimeSearch(search.function, array, trials[j], value);
00074 
00075             #ifdef LOG
00076             SearchData curr = data[j + i * numtrials];
00077             fPrintArray(log, array, trials[j]);
00078 
00079             bool found = (curr.index != SIZE_MAX);
00080 
00081             if(!found)
00082                 fprintf(log, "Value %lu Not Found\n", (uint64_t)value);
00083             else
00084                 fprintf(log, "Value %lu Found at Index %zu in %lf ms\n", (uint64_t)value, curr.index,
      curr.time_ms);
00085 
00086             fprintf(log, "Valid Result: %s\n\n", (found? "True": "False"));
00087             #endif
00088         }
00089     }
00090 
00091     for(size_t i = 0; i < numtrials; i++) {
00092 
00093         for(size_t j = 1; j < numtimes; j++)
00094             data[i].time_ms += data[ i + j * numtrials].time_ms;
00095 
00096         data[i].time_ms /= numtimes;
00097 
00098     }
00099 
00100     WriteSearchDataToFile(file, data, numtrials);
00101 
00102     #ifndef VLA
00103     free(data);
00104     free(array);
00105     #endif
00106 
00107     fclose(file);
00108     #ifdef LOG
00109     fclose(log);
00110     #endif
00111 
00112 }
00113 
00114 // Benchmark Every Search in an Array
00115 void BenchmarkSearches(const Search* const searches, const size_t numsearches, const Data* const
      trials, const size_t numtrials, const size_t numtimes) {
00116 
00117     Assert(searches, "Invalid Search Array in Benchmark");
00118 
00119     for(size_t i = 0; i < numsearches; i++)
00120         BenchmarkSearch(searches[i], trials, numtrials, numtimes);
00121 
00122 }
00123 
00124 
00125 void WriteSearchDataToFile(FILE* const file, const SearchData* const data, const size_t size) {
00126 
00127     Assert(file, "Bad File Pointer in Writing Search Data");
00128     Assert(data, "Bad SearchData Array in Writing Search Data to File");
```

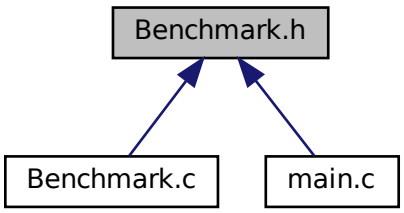
```
00129
00130     for(size_t i = 0; i < size; i++)
00131         fprintf(file, "%zu,%lf\n", data[i].n, data[i].time_ms);
00132
00133 }
00134 // Prints An Array to a File
00135 void fPrintArray(FILE* const file, const Data* const array, const size_t size) {
00136
00137     Assert(file, "Invalid File Pointer in Print Array");
00138     Assert(array, "Invalid Array Pointer in Print Array");
00139
00140     for(size_t i = 0; i < size; i++)
00141         fprintf(file, "%lu ", (uint64_t)array[i]);
00142
00143     fputc('\n', file);
00144 }
```

5.17 Benchmark.h File Reference

`#include "Utils.h"`
Include dependency graph for Benchmark.h:



This graph shows which files directly or indirectly include this file:



Data Structures

- struct SearchData

Macros

- #define issigned(t) (((t)(-1)) < ((t) 0))

  Checks if a Data Type is Signed or Not.
- #define umaxof(t)

  Gets the Unsigned Max of a Type.
- #define smaxof(t)

  Gets the Signed Max of a Type t.
- #define maxof(t) ((unsigned long long) (issigned(t) ? smaxof(t) : umaxof(t)))

  Gets the Maximum Value of a Type t.
- #define minof(t) ((issigned(t) * -maxof(t)) - 1 + !issigned(t)*1)

  Gets the Minimum Value of a Type t.
- #define PrintArray(array, size) (fPrintArray(stdout, array, size))

  A Special Case to Print to the Standard output.

Functions

- SearchData TimeSearch (const SearchFunction search, const Data *const array, const size_t size, const Data value)
- void BenchmarkSearch (const Search search, const Data *const trials, const size_t numtrials, const size_t numtimes)
- void BenchmarkSearches (const Search *const searches, const size_t numsearches, const Data *const trials, const size_t numtrials, const size_t numtimes)
- void fPrintArray (FILE *const file, const Data *const array, const size_t size)

  Prints an Array to a File.

5.17.1 Detailed Description

Author

Orion Serup (orionserup@gmail.com)

Version

0.1

Date

2022-01-17

Copyright

Copyright (c) 2022

Definition in file Benchmark.h.

5.17.2 Macro Definition Documentation

5.17.2.1 issigned

```
#define issigned(
              t ) (((t)(-1)) < ((t) 0))
```

Checks if a Data Type is Signed or Not.

Definition at line 17 of file Benchmark.h.

5.17.2.2 maxof

```
#define maxof(
              t ) ((unsigned long long) (issigned(t) ?  smaxof(t) :  umaxof(t)))
```

Gets the Maximum Value of a Type t.

Definition at line 28 of file Benchmark.h.

5.17.2.3 minof

```
#define minof(
              t ) ((issigned(t) * -maxof(t)) - 1 + !issigned(t)*1)
```

Gets the Minimum Value of a Type t.

Definition at line 31 of file Benchmark.h.

5.17.2.4 PrintArray

```
#define PrintArray(
              array,
              size ) (fPrintArray(stdout, array, size))
```

A Special Case to Print to the Standard output.

Definition at line 82 of file Benchmark.h.

5.17.2.5 smaxof

```
#define smaxof(
              t )
```

Value:

```
(((0x1ULL « ((sizeof(t) * 8ULL) - 1ULL)) - 1ULL) | \
(0x7ULL « ((sizeof(t) * 8ULL) - 4ULL)))
```

Gets the Signed Max of a Type t.

Definition at line 24 of file Benchmark.h.

5.17.2.6 umaxof

```
#define umaxof(
              t )
```

Value:

```
(((0x1ULL « ((sizeof(t) * 8ULL) - 1ULL)) - 1ULL) | \
(0xFULL « ((sizeof(t) * 8ULL) - 4ULL)))
```

Gets the Unsigned Max of a Type.

Definition at line 20 of file Benchmark.h.

5.17.3 Function Documentation

5.17.3.1 BenchmarkSearch()

```
void BenchmarkSearch (
            const Search search,
            const Data *const trials,
            const size_t numtrials,
            const size_t numtimes )
```

Parameters

search	
trials	
numtrials	
numtimes	

Definition at line 27 of file Benchmark.c.

5.17.3.2 BenchmarkSearches()

```
void BenchmarkSearches (
            const Search *const searches,
            const size_t numsearches,
            const Data *const trials,
            const size_t numtrials,
            const size_t numtimes )
```

Parameters

searches	
numsearches	
trials	
numtrials	
numtimes	

Definition at line 115 of file Benchmark.c.

5.17.3.3 fPrintArray()

```
void fPrintArray (
            FILE *const file,
            const Data *const array,
            const size_t size )
```

Prints an Array to a File.

Note

File can Also be stdout, see PrintArray

Parameters

array	Array to Print
size	Size of the Array

Definition at line 135 of file Benchmark.c.

5.17.3.4 TimeSearch()

```
SearchData TimeSearch (
            const SearchFunction search,
            const Data *const array,
            const size_t size,
            const Data value )
```

Parameters

search	The Function Pointer of the Search
array	
size	
value	

Returns

SearchData

Definition at line 16 of file Benchmark.c.

5.18 Benchmark.h

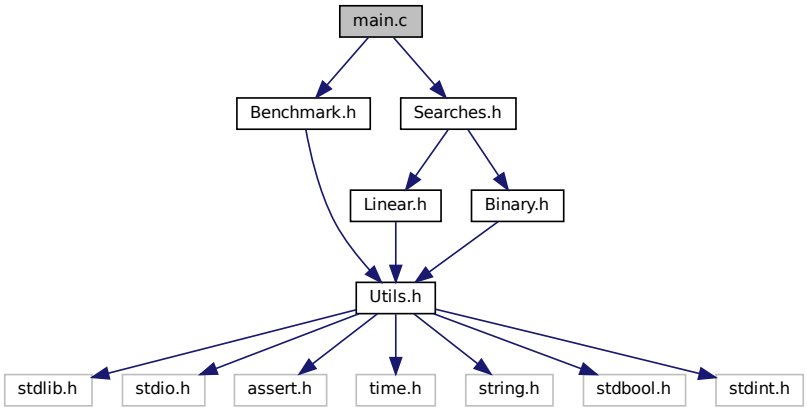
Go to the documentation of this file.

```
00001
00011 #include "Utils.h"
00012
00013 #ifndef SEARCH_BENCHMARK
00014 #define SEARCH_BENCHMARK
00015
00017 #define issigned(t) (((t)(-1)) < ((t) 0))
00018
00020 #define umaxof(t) (((0x1ULL << ((sizeof(t) * 8ULL) - 1ULL)) - 1ULL) | \
00021                    (0xFULL << ((sizeof(t) * 8ULL) - 4ULL)))
00022
00024 #define smaxof(t) (((0x1ULL << ((sizeof(t) * 8ULL) - 1ULL)) - 1ULL) | \
00025                    (0x7ULL << ((sizeof(t) * 8ULL) - 4ULL)))
00026
00028 #define maxof(t) ((unsigned long long) (issigned(t) ? smaxof(t) : umaxof(t)))
00029
00031 #define minof(t) ((issigned(t) * -maxof(t)) - 1 + !issigned(t)*1)
00032
00033 typedef struct {
00034
00035     size_t n;
00036     double time_ms;
00037     size_t index;
00038
00039 } SearchData;
00040
00050 SearchData TimeSearch(const SearchFunction search, const Data* const array, const size_t size, const
       Data value);
00051
00060 void BenchmarkSearch(const Search search, const Data* const trials, const size_t numtrials, const
       size_t numtimes);
00061
00071 void BenchmarkSearches(const Search* const searches, const size_t numsearches, const Data* const
       trials, const size_t numtrials, const size_t numtimes);
00072
00079 void fPrintArray(FILE* const file, const Data* const array, const size_t size);
00080
00082 #define PrintArray(array, size) (fPrintArray(stdout, array, size))
00083
00084
00085 #endif
00086
00087
```

5.19 main.c File Reference

```
#include "Benchmark.h"
#include "Searches.h"
```

Include dependency graph for main.c:



Macros

- #define MIN 1
- #define MAX 5
- #define STEP 1
- #define SIZE (MAX - MIN)/STEP + 1

Functions

- int main ()

5.19.1 Detailed Description

Author

Orion Serup (orionserup@gmail.com)

Version

0.1

Date

2022-01-18

Copyright

Copyright (c) 2022

Definition in file main.c.

5.19.2 Macro Definition Documentation

5.19.2.1 MAX

```
#define MAX 5
```

Definition at line 16 of file main.c.

5.19.2.2 MIN

```
#define MIN 1
```

Definition at line 15 of file main.c.

5.19.2.3 SIZE

```
#define SIZE (MAX - MIN)/STEP + 1
```

Definition at line 18 of file main.c.

5.19.2.4 STEP

```
#define STEP 1
```

Definition at line 17 of file main.c.

5.19.3 Function Documentation

5.19.3.1 main()

```
int main ( )
```

Definition at line 20 of file main.c.

5.20 main.c

Go to the documentation of this file.

```
00001
00012 #include "Benchmark.h"
00013 #include "Searches.h"
00014
00015 #define MIN  1
00016 #define MAX  5
00017 #define STEP 1
00018 #define SIZE (MAX - MIN)/STEP + 1
00019
00020 int main() {
00021
00022     Search searches[] = { LINEAR_SEARCH, BINARY_SEARCH };
00023     size_t numsearches = sizeof(searches)/sizeof(Search);
00024
00025     Data n[SIZE];
00026
00027     for(size_t i = 0; i < SIZE; i++)
00028         n[i] = MIN + STEP * i;
00029
00030     BenchmarkSearches(searches, numsearches, n, SIZE, 5);
00031
00032 }
```

Index