

## Lab 1

Generated by Doxygen 1.9.4



<b>1 Lab 1</b>	<b>1</b>
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Athlete Class Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 Athlete() [1/2]	8
4.1.2.2 Athlete() [2/2]	8
4.1.2.3 ~Athlete()	8
4.1.3 Member Function Documentation	8
4.1.3.1 operator!=(())	9
4.1.3.2 operator<()	9
4.1.3.3 operator<=()	9
4.1.3.4 operator==(())	9
4.1.3.5 operator>()	9
4.1.3.6 operator>=()	9
4.1.4 Member Data Documentation	10
4.1.4.1 age	10
4.1.4.2 height	10
4.1.4.3 name	10
4.1.4.4 sport	10
4.1.4.5 weight	10
<b>5 File Documentation</b>	<b>11</b>
5.1 athlete.h File Reference	11
5.2 athlete.h	12
5.3 main.cpp File Reference	12
5.3.1 Function Documentation	13
5.3.1.1 main()	13
5.4 sorting_algs.h File Reference	13
5.4.1 Function Documentation	14
5.4.1.1 downHeap()	14
5.4.1.2 heapSort()	15
5.4.1.3 insertSort()	15
5.4.1.4 quickSortR()	15
5.4.1.5 standartSort()	16
5.5 sorting_algs.h	16
5.6 utils.h File Reference	17

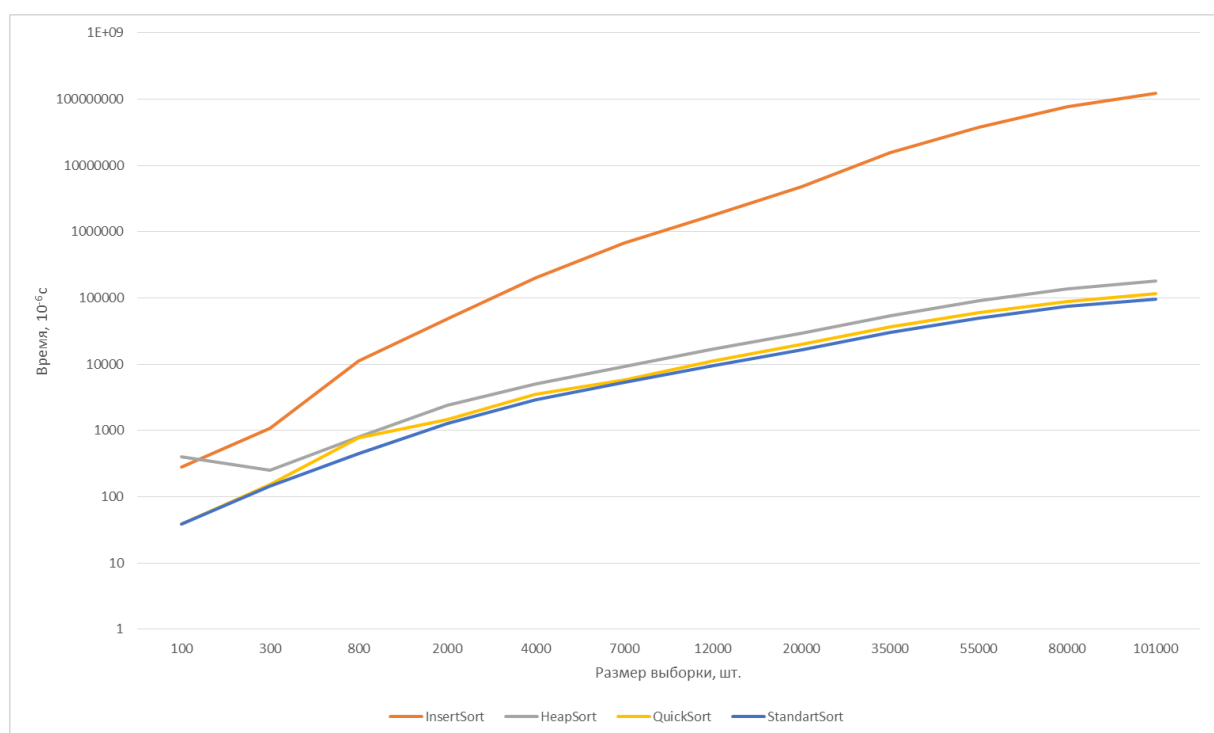
5.6.1 Function Documentation . . . . .	18
5.6.1.1 fillAthleteArray() . . . . .	18
5.6.1.2 generateAthletesCSV() . . . . .	18
5.6.1.3 randomString() . . . . .	18
5.7 utils.h . . . . .	19

# Chapter 1

## Lab 1

Link to repository: <https://github.com/mrzrow/pm-labs>

### Results





## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">Athlete</a>	
<a href="#">Athlete</a> Class . . . . .	<a href="#">7</a>





## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">athlete.h</a>	11
<a href="#">main.cpp</a>	12
<a href="#">sorting_algs.h</a>	13
<a href="#">utils.h</a>	17



## Chapter 4

# Class Documentation

### 4.1 Athlete Class Reference

[Athlete](#) Class.

```
#include <athlete.h>
```

#### Public Member Functions

- [Athlete](#) ()  
*Default constructor.*
- [Athlete](#) (const std::string &n, int a, int h, int w, const std::string &s)
- [~Athlete](#) ()  
*Destructor.*
- bool [operator==](#) (const [Athlete](#) &other)  
*overloading equal*
- bool [operator<](#) (const [Athlete](#) &other)  
*overloading less*
- bool [operator!=](#) (const [Athlete](#) &other)  
*overloading not equal*
- bool [operator<=](#) (const [Athlete](#) &other)  
*overloading less or equal*
- bool [operator>](#) (const [Athlete](#) &other)  
*overloading greater*
- bool [operator>=](#) (const [Athlete](#) &other)  
*overloading greater or equal*

#### Public Attributes

- std::string [name](#)
- int [age](#)
- int [height](#)
- int [weight](#)
- std::string [sport](#)

### 4.1.1 Detailed Description

[Athlete](#) Class.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 Athlete() [1/2]

```
Athlete::Athlete ( ) [inline]
```

Default constructor.

#### 4.1.2.2 Athlete() [2/2]

```
Athlete::Athlete (
    const std::string & n,
    int a,
    int h,
    int w,
    const std::string & s ) [inline]
```

Constructor

Parameters

<i>n</i>	name
<i>a</i>	age
<i>h</i>	height
<i>w</i>	weight
<i>s</i>	sport

#### 4.1.2.3 ~Athlete()

```
Athlete::~~Athlete ( ) [inline]
```

Destructor.

### 4.1.3 Member Function Documentation

#### 4.1.3.1 operator!=(())

```
bool Athlete::operator!=(  
    const Athlete & other ) [inline]
```

overloading not equal

#### 4.1.3.2 operator<()

```
bool Athlete::operator<(  
    const Athlete & other ) [inline]
```

overloading less

#### 4.1.3.3 operator<=()

```
bool Athlete::operator<= (  
    const Athlete & other ) [inline]
```

overloading less or equal

#### 4.1.3.4 operator==(())

```
bool Athlete::operator==(  
    const Athlete & other ) [inline]
```

overloading equal

#### 4.1.3.5 operator>()

```
bool Athlete::operator> (  
    const Athlete & other ) [inline]
```

overloading greater

#### 4.1.3.6 operator>=()

```
bool Athlete::operator>= (  
    const Athlete & other ) [inline]
```

overloading greater or equal

## 4.1.4 Member Data Documentation

### 4.1.4.1 age

```
int Athlete::age
```

### 4.1.4.2 height

```
int Athlete::height
```

### 4.1.4.3 name

```
std::string Athlete::name
```

### 4.1.4.4 sport

```
std::string Athlete::sport
```

### 4.1.4.5 weight

```
int Athlete::weight
```

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

- [athlete.h](#)

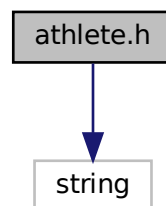
## Chapter 5

# File Documentation

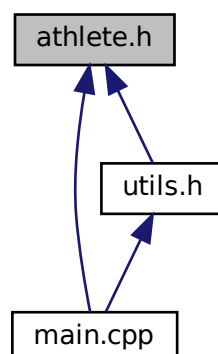
### 5.1 athlete.h File Reference

```
#include <string>
```

Include dependency graph for athlete.h:



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



## Classes

- class [Athlete](#)  
*Athlete Class.*

## 5.2 athlete.h

[Go to the documentation of this file.](#)

```

1 #ifndef ATHLETE_H_
2 #define ATHLETE_H_
3
4 #include <string>
5
6
7
8 class Athlete {
9 public:
10     std::string name;
11     int age;
12     int height;
13     int weight;
14     std::string sport;
15
16 public:
17     Athlete() {}
18     Athlete(const std::string& n, int a, int h, int w, const std::string& s):
19         name(n), age(a), height(h), weight(w), sport(s) {}
20     ~Athlete() {}
21
22     bool operator==(const Athlete& other) {
23         return sport == other.sport &&
24             name == other.name &&
25             age == other.age;
26     }
27
28     bool operator<(const Athlete& other) {
29         if ( sport != other.sport )
30             return sport < other.sport;
31         if ( name != other.name )
32             return name < other.name;
33         return age < other.age;
34     }
35
36     bool operator!=(const Athlete& other) { return !(*this == other); }
37     bool operator<=(const Athlete& other) { return *this == other || *this < other; }
38     bool operator> (const Athlete& other) { return !(*this <= other); }
39     bool operator>=(const Athlete& other) { return *this > other || *this == other; }
40 };
41
42 #endif

```

## 5.3 main.cpp File Reference

```

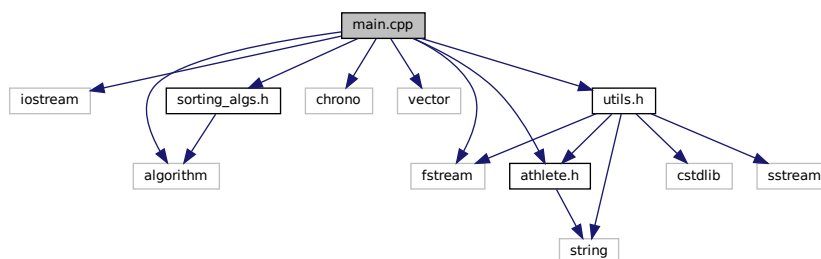
#include <iostream>
#include <algorithm>
#include <chrono>
#include <vector>
#include <fstream>
#include "athlete.h"
#include "sorting_algs.h"

```



```
#include "utils.h"
```

Include dependency graph for main.cpp:



## Functions

- int `main` ()

### 5.3.1 Function Documentation

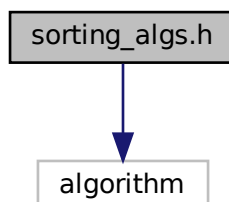
#### 5.3.1.1 main()

```
int main ( )
```

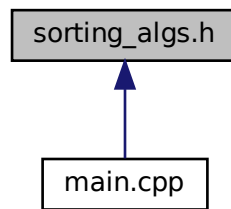
## 5.4 sorting\_algs.h File Reference

```
#include <algorithm>
```

Include dependency graph for sorting\_algs.h:



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



## Functions

- `template<class T >`  
void `insertSort` (T \*a, long size)
- `template<class T >`  
void `downHeap` (T \*a, long k, long n)
- `template<class T >`  
void `heapSort` (T \*a, long size)
- `template<class T >`  
void `quickSortR` (T \*a, long size)
- `template<class T >`  
void `standartSort` (T \*a, long size)

## 5.4.1 Function Documentation

### 5.4.1.1 downHeap()

```

template<class T >
void downHeap (
    T * a,
    long k,
    long n )
  
```

Building pyramid

#### Parameters

<i>k</i>	start index
<i>n</i>	end index

### 5.4.1.2 heapSort()

```
template<class T >
void heapSort (
    T * a,
    long size )
```

Heap sort

#### Parameters

<i>a</i>	array to sort
<i>size</i>	size of the array

### 5.4.1.3 insertSort()

```
template<class T >
void insertSort (
    T * a,
    long size )
```

Insert sort

#### Parameters

<i>a</i>	array to sort
<i>size</i>	size of the array

### 5.4.1.4 quickSortR()

```
template<class T >
void quickSortR (
    T * a,
    long size )
```

Quick sort

#### Parameters

<i>a</i>	array to sort
<i>size</i>	size of the array

### 5.4.1.5 standartSort()

```
template<class T >
void standartSort (
    T * a,
    long size )
```

Wrapper for std::sort

#### Parameters

<i>a</i>	array to sort
<i>size</i>	size of the array

## 5.5 sorting\_algs.h

[Go to the documentation of this file.](#)

```
1 #ifndef SORTING_ALGS_H_
2 #define SORTING_ALGS_H_
3
4 #include <algorithm>
5
11 template<class T> void insertSort(T* a, long size) {
12     T x;
13     long i, j;
14
15     for (i = 0; i < size; i++) {
16         x = a[i];
17
18         for ( j=i-1; j>=0 && a[j] > x; j--)
19             a[j+1] = a[j];
20
21         a[j+1] = x;
22     }
23 }
24
30 template<class T> void downHeap(T* a, long k, long n) {
31     T new_elem;
32     long child;
33     new_elem = a[k];
34
35     while(k <= n/2) {
36         child = 2*k;
37         if( child < n && a[child] < a[child+1] ) child++;
38         if( new_elem >= a[child] ) break;
39         a[k] = a[child];
40         k = child;
41     }
42     a[k] = new_elem;
43 }
49 template<class T> void heapSort(T* a, long size) {
50     long i;
51     T temp;
52
53     for(i=size/2-1; i >= 0; i--)
54         downHeap(a, i, size-1);
55
56     for(i=size-1; i > 0; i--) {
57         std::swap(a[i], a[0]);
58         downHeap(a, 0, i-1);
59     }
60 }
61
67 template<class T> void quickSortR(T* a, long size)
68 {
69     long i = 0, j = size-1;
70     T p = a[ size>>1 ];
71
72     do {
73         while ( a[i] < p ) i++;
74         while ( a[j] > p ) j--;
75
```

```

76         if (i <= j) {
77             std::swap(a[i], a[j]);
78             i++; j--;
79         }
80     } while ( i<=j );
81
82     if ( j > 0 ) quickSortR(a, j);
83     if ( size > i ) quickSortR(a+i, size-i);
84 }
85
91 template<class T> void standartSort(T* a, long size) {
92     std::sort(a, a + size);
93 }
94
95 #endif

```

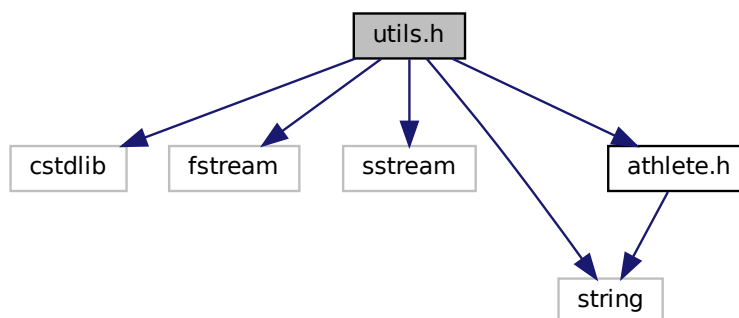
## 5.6 utils.h File Reference

```

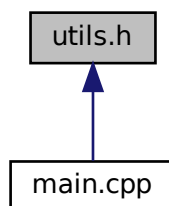
#include <cstdlib>
#include <fstream>
#include <sstream>
#include <string>
#include "athlete.h"

```

Include dependency graph for utils.h:



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



## Functions

- std::string [randomString](#) (int length)
- void [generateAthletesCSV](#) (const std::string &filename, int total=101000)
- void [fillAthleteArray](#) ([Athlete](#) \*arr, int n, const std::string &filename="athletes.csv")

### 5.6.1 Function Documentation

#### 5.6.1.1 fillAthleteArray()

```
void fillAthleteArray (  
    Athlete * arr,  
    int n,  
    const std::string & filename = "athletes.csv" )
```

Fill array with [Athlete](#) objects

##### Parameters

<i>arr</i>	array to fill
<i>n</i>	number of athletes
<i>filename</i>	name of the file to read

#### 5.6.1.2 generateAthletesCSV()

```
void generateAthletesCSV (  
    const std::string & filename,  
    int total = 101000 )
```

Generates file of athletes data

##### Parameters

<i>filename</i>	name of the file to save
<i>total</i>	number of lines of data

#### 5.6.1.3 randomString()

```
std::string randomString (  
    int length )
```

Generates random string

## Parameters

<i>length</i>	length of the string
---------------	----------------------

## Returns

generated string

## 5.7 utils.h

[Go to the documentation of this file.](#)

```

1 #ifndef UTILS_H_
2 #define UTILS_H_
3
4 #include <cstdlib>
5 #include <fstream>
6 #include <sstream>
7 #include <string>
8
9 #include "athlete.h"
10
16 std::string randomString(int length) {
17     const std::string chars = "abcdefghijklmnopqrstuvwxyz";
18
19     std::string result;
20     for (int i = 0; i < length; ++i)
21         result += chars[rand() % chars.size()];
22     return result;
23 }
24
25
31 void generateAthletesCSV(const std::string& filename, int total = 101000) {
32     const std::string sports[] = {"Football", "Basketball", "Tennis", "Swimming", "Running"};
33     int numSport = 5;
34
35     std::ofstream file(filename);
36     if (!file.is_open()) return;
37
38     for (int i = 0; i < total; ++i) {
39         std::string name = randomString(5 + rand() % 5);
40         int age = 18 + rand() % 23;
41         int height = 160 + rand() % 41;
42         int weight = 50 + rand() % 51;
43         std::string sport = sports[rand() % numSport];
44
45         file << name << "," << age << "," << height << "," << weight << "," << sport << "\n";
46     }
47
48     file.close();
49 }
50
51
58 void fillAthleteArray(Athlete* arr, int n, const std::string& filename = "athletes.csv") {
59     std::ifstream file(filename);
60     if (!file.is_open()) return;
61
62     std::string line;
63     int count = 0;
64
65     while (count < n && std::getline(file, line)) {
66         std::stringstream ss(line);
67         std::string name, ageStr, heightStr, weightStr, sport;
68
69         std::getline(ss, name, ',');
70         std::getline(ss, ageStr, ',');
71         std::getline(ss, heightStr, ',');
72         std::getline(ss, weightStr, ',');
73         std::getline(ss, sport);
74
75         int age = std::stoi(ageStr);
76         int height = std::stoi(heightStr);
77         int weight = std::stoi(weightStr);
78
79         arr[count++] = Athlete(name, age, height, weight, sport);
80     }
81
82     file.close();
83 }
84
85 #endif

```

