

My Project

AUTHOR
Version 3
Fri Dec 23 2022

Table of Contents

Table of contents

Hierarchical Index

Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

- exception
 - noPop7
 - outOfRange8
- YFVector< T >::iterator< T >5
- YFVector< T >9

Class Index

Class List

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

YFVector< T >::iterator< T >	5
noPop	7
outOfRange	8
YFVector< T >	9

File Index

File List

Here is a list of all files with brief descriptions:

C:/Users/pc/CLionProjects/YFvector/exceptions.h	13
C:/Users/pc/CLionProjects/YFvector/main.cpp	15
C:/Users/pc/CLionProjects/YFvector/YFVector.cpp	19
C:/Users/pc/CLionProjects/YFvector/YFVector.h	25

Class Documentation

YFVector< T >::iterator< T > Class Template Reference

Public Member Functions

- **iterator** (T *_data)
 - **iterator** & **operator++** ()
 - **iterator** & **operator+=** (size_t val)
 - **iterator** **operator+** (size_t val)
 - **iterator** & **operator--** ()
 - **bool operator<** (**iterator** const &other) const
 - **bool operator>** (**iterator** const &other) const
 - **bool operator==** (const **iterator** &anotherIter) const
 - **bool operator!=** (const **iterator** &anotherIter) const
 - T & **operator*** ()
-

Detailed Description

template<class T>

template<class T>

class YFVector< T >::iterator< T >

Definition at line 137 of file **YFVector.cpp**.

Constructor & Destructor Documentation

template<class T > template<class T > YFVector< T >::iterator< T >::iterator (T *_data)[**inline**], [**explicit**]

Definition at line 143 of file **YFVector.cpp**.

Member Function Documentation

template<class T > template<class T > bool YFVector< T >::iterator< T >::operator!= (const **iterator< T >** & *anotherIter*) const [**inline**]

Definition at line 187 of file **YFVector.cpp**.

template<class T > template<class T > T & YFVector< T >::iterator< T >::operator* () [**inline**]

Definition at line 192 of file **YFVector.cpp**.


```
template<class T > template<class T > iterator YFVector< T >::iterator< T >::operator+
(size_t  val)[inline]
```

Definition at line 159 of file YFVector.cpp.

```
template<class T > template<class T > iterator & YFVector< T >::iterator< T
>::operator++ ()[inline]
```

Definition at line 147 of file YFVector.cpp.

```
template<class T > template<class T > iterator & YFVector< T >::iterator< T
>::operator+= (size_t  val)[inline]
```

Definition at line 153 of file YFVector.cpp.

```
template<class T > template<class T > iterator & YFVector< T >::iterator< T
>::operator-- ()[inline]
```

Definition at line 166 of file YFVector.cpp.

```
template<class T > template<class T > bool YFVector< T >::iterator< T >::operator<
(iterator< T > const &  other) const[inline]
```

Definition at line 172 of file YFVector.cpp.

```
template<class T > template<class T > bool YFVector< T >::iterator< T >::operator==
(const iterator< T > &  anotherIter) const[inline]
```

Definition at line 182 of file YFVector.cpp.

```
template<class T > template<class T > bool YFVector< T >::iterator< T >::operator>
(iterator< T > const &  other) const[inline]
```

Definition at line 177 of file YFVector.cpp.

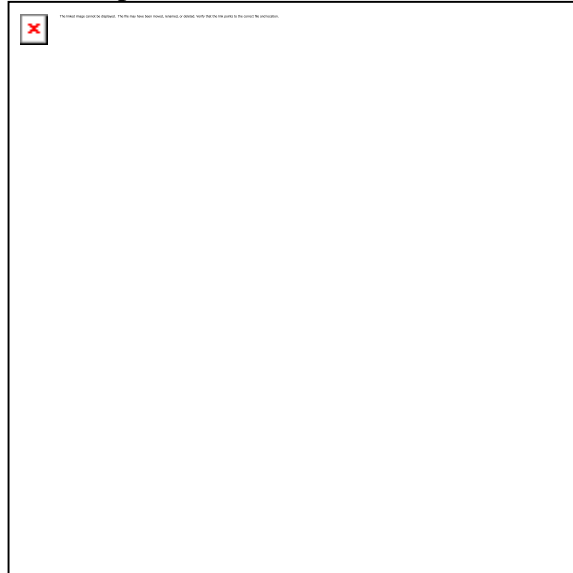
The documentation for this class was generated from the following file:

- C:/Users/pc/CLionProjects/YFvector/YFVector.cpp

noPop Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for noPop:



Public Member Functions

- `noPop ()`
- `string what ()`

Detailed Description

Definition at line **15** of file **exceptions.h**.

Constructor & Destructor Documentation

noPop::noPop ()`[inline]`

Definition at line **17** of file **exceptions.h**.

Member Function Documentation

string noPop::what ()`[inline]`

Definition at line **18** of file **exceptions.h**.

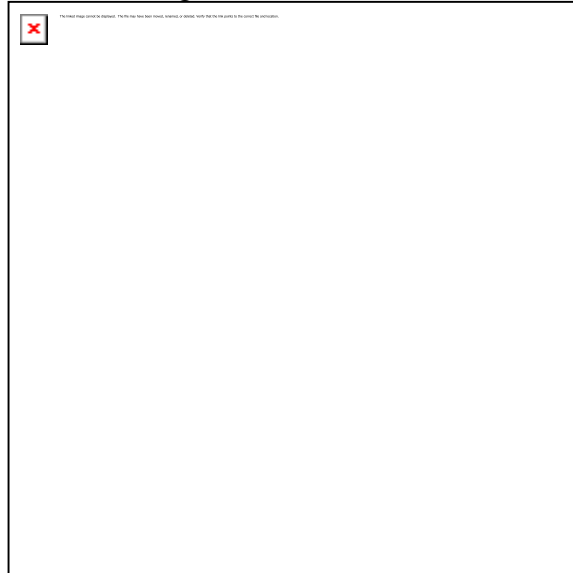
The documentation for this class was generated from the following file:

- `C:/Users/pc/CLionProjects/YFvector/exceptions.h`

outOfRange Class Reference

```
#include <exceptions.h>
```

Inheritance diagram for outOfRange:



Public Member Functions

- `outOfRange ()`
- `string what ()`

Detailed Description

Definition at line 7 of file `exceptions.h`.

Constructor & Destructor Documentation

`outOfRange::outOfRange () [inline]`

Definition at line 9 of file `exceptions.h`.

Member Function Documentation

`string outOfRange::what () [inline]`

Definition at line 10 of file `exceptions.h`.

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

- `C:/Users/pc/CLionProjects/YFvector/exceptions.h`

YFVector< T > Class Template Reference

```
#include <YFVector.h>
```

Classes

- class **iterator**

Public Member Functions

- **YFVector** ()
- **YFVector** (int)
- **YFVector** (T *, int)
- **YFVector** (const **YFVector** &)
- **~YFVector** ()
- **YFVector** & **operator=** (const **YFVector** &)
- **YFVector** & **operator=** (**YFVector** &&) noexcept
- T & **operator[]** (int)
- void **push_back** (T)
- T **pop_back** ()
- void **erase** (iterator)
- void **erase** (iterator, iterator)
- void **clear** ()
- void **insert** (iterator, T const &)
- **iterator** **begin** ()
- **iterator** **end** ()
- bool **operator==** (const **YFVector**< T > &) const
- bool **operator<** (const **YFVector**< T > &) const
- int **size** () const
- int **capacity** () const
- int **resize** ()
- bool **empty** () const

Friends

- ostream & **operator<<** (ostream &out, **YFVector**< T > &vt)

Detailed Description

template<class T>

class YFVector< T >

Definition at line **9** of file **YFVector.h**.

Constructor & Destructor Documentation

template<typename T > YFVector< T >::YFVector [**inline**]

Definition at line **10** of file **YFVector.cpp**.

template<typename T > YFVector< T >::YFVector (int *n*) [inline], [explicit]

Definition at line 15 of file YFVector.cpp.

template<typename T > YFVector< T >::YFVector (T * *data*, int *n*) [inline]

Definition at line 20 of file YFVector.cpp.

template<typename T > YFVector< T >::YFVector (const YFVector< T > & *anotherVec*) [inline]

Definition at line 25 of file YFVector.cpp.

template<typename T > YFVector< T >::~~YFVector [inline]

Definition at line 31 of file YFVector.cpp.

Member Function Documentation

template<typename T > YFVector< T >::iterator YFVector< T >::begin [inline]

Definition at line 199 of file YFVector.cpp.

template<typename T > int YFVector< T >::capacity [inline]

Definition at line 294 of file YFVector.cpp.

template<typename T > void YFVector< T >::clear [inline]

Definition at line 124 of file YFVector.cpp.

template<typename T > bool YFVector< T >::empty [inline]

Definition at line 300 of file YFVector.cpp.

template<typename T > YFVector< T >::iterator YFVector< T >::end [inline]

Definition at line 206 of file YFVector.cpp.

template<typename T > void YFVector< T >::erase (iterator *iter*) [inline]

Definition at line 231 of file YFVector.cpp.

template<typename T > void YFVector< T >::erase (iterator *iter1*, iterator *iter2*) [inline]

Definition at line 212 of file YFVector.cpp.

```
template<typename T > void YFVector< T >::insert (iterator  iter, T const & val)[inline]
```

Definition at line 239 of file YFVector.cpp.

```
template<typename T > bool YFVector< T >::operator< (const YFVector< T > &  v) const[inline]
```

Definition at line 274 of file YFVector.cpp.

```
template<typename T > YFVector< T > & YFVector< T >::operator= (const YFVector< T > &  anotherVec)[inline]
```

Definition at line 49 of file YFVector.cpp.

```
template<typename T > YFVector< T > & YFVector< T >::operator= (YFVector< T > && anotherVec)[inline], [noexcept]
```

Definition at line 60 of file YFVector.cpp.

```
template<typename T > bool YFVector< T >::operator== (const YFVector< T > &  v) const[inline]
```

Definition at line 259 of file YFVector.cpp.

```
template<typename T > T & YFVector< T >::operator[] (int  index)[inline]
```

Definition at line 76 of file YFVector.cpp.

```
template<typename T > T YFVector< T >::pop_back[inline]
```

Definition at line 114 of file YFVector.cpp.

```
template<typename T > void YFVector< T >::push_back (T  data)[inline]
```

Definition at line 103 of file YFVector.cpp.

```
template<typename T > int YFVector< T >::resize [inline]
```

Definition at line 86 of file YFVector.cpp.

```
template<typename T > int YFVector< T >::size [inline]
```

Definition at line 288 of file YFVector.cpp.

Friends And Related Function Documentation

`template<class T > ostream & operator<< (ostream & out, YFVector< T > & vt) [friend]`

Definition at line 55 of file **YFVector.h**.

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

- C:/Users/pc/CLionProjects/YFvector/**YFVector.h**
- C:/Users/pc/CLionProjects/YFvector/**YFVector.cpp**

File Documentation

C:/Users/pc/CLionProjects/YFvector/exceptions.h File Reference

```
#include <stdexcept>
```

Classes

- class **outOfRange**
- class **noPop**

exceptions.h

Go to the documentation of this file.

```
00001 #ifndef EXCEPTIONS_H_INCLUDED
00002 #define EXCEPTIONS_H_INCLUDED
00003 #include <stdexcept>
00004
00005 using namespace std;
00006
00007 class outOfRange : public exception{
00008 public:
00009     outOfRange() :errmsg{"Attempted to access OUT OF RANGE element"}{}
00010     string what(){return errmsg;}
00011 private:
00012     string errmsg;
00013 };
00014
00015 class noPop : public exception{
00016 public:
00017     noPop():errmsg{"CAN'T pop, there are no elements in the Vector"}{}
00018     string what(){return errmsg;}
00019 private:
00020     string errmsg;
00021 };
00022
00023
00024
00025 #endif // EXCEPTIONS_H_INCLUDED
```

C:/Users/pc/CLionProjects/YFvector/main.cpp File Reference

```
#include <bits/stdc++.h>
#include "YFVector.cpp"
#include "exceptions.h"
```

Functions

- `int main ()`
-

Function Documentation

`int main ()`

Definition at line **6** of file **main.cpp**.

main.cpp

```
Go to the documentation of this file.00001 #include <bits/stdc++.h>
00002 #include "YFVector.cpp"
00003 #include "exceptions.h"
00004
00005 using namespace std;
00006 int main(){
00007     int test[5] = {1, 2, 3, 4, 5};
00008     YFVector<int> v4(test, sizeof(test)/sizeof(test[0]));
00009
00010     cout << "v4 after passing an array\n";
00011     cout << v4 ;
00012     cout << "\n-----\n";
00013
00014     YFVector<int> v5(v4);
00015     cout << "v5 initialized by a copy constructor of v4:\n";
00016     cout << v5 << endl;
00017
00018     cout << "\nv5 size = " << v5.size() << "\n";
00019     cout << "v5 capacity = " << v5.capacity() << "\n";
00020     cout << "-----\n";
00021
00022     YFVector<int> v6;
00023     v6 = v5;
00024     cout << "v6 after copy assignment of v5:\n";
00025     cout << v6;
00026     cout << "\nv6 size = " << v6.size();
00027     cout << "\nv6 capacity = " << v6.capacity() << "\n";
00028
00029     cout << "v5 is " << ((v5.empty()) ? "empty": "NOT empty") << endl;
00030     cout << v5 << endl;
00031     cout << "-----\n";
00032
00033     YFVector<int> v7;
00034     v7 = std::move(v6);
00035     cout << "v7 after move assignment: \n";
00036     cout << v7 << endl;
00037     cout << "v6 is " << ((v6.empty()) ? "empty": "NOT empty") << endl;
00038     cout << "\nv7 size = " << v7.size();
00039     cout << "\nv7 capacity = " << v7.capacity() << "\n";
00040     cout << "-----\n";
00041
00042     YFVector<char> v1(4);
00043     YFVector<int> v2;
00044     YFVector<int> v3(4);
00045
00046     v1.push_back('f');
00047     v1.push_back('a');
00048     v1.push_back('t');
00049     v1.push_back('m');
00050     // More capacity *2 to push another element
00051     v1.push_back('a');
00052
00053     v2.push_back(10);
00054     v2.push_back(20);
00055     v2.push_back(30);
00056     v2.push_back(40);
00057
00058     v3.push_back(10);
00059     v3.push_back(2);
00060     v3.push_back(30);
00061     v3.push_back(40);
00062
00063     cout << "Compare between v2 == v3 : "<< ((v2 == v3) ? "True" : "False") << endl;
00064
00065     YFVector<int> vec4(3);
00066     YFVector<int> vec5(3);
00067
00068     vec4.push_back(6);
00069     vec4.push_back(4);
00070     vec4.push_back(1);
00071
00072     vec5.push_back(6);
```

```

00075     vec5.push_back(4);
00076     vec5.push_back(2);
00077     cout << "-----" << endl;
00078     cout << "Is v4 < 45 ?" << ((vec4 < vec5) ? " Yes": " NO" ) << endl;
00079     cout << "-----" << endl;
00080
00081
00082     cout << "V1 size: " << v1.size() << endl;
00083     cout << "V1 capacity: " << v1.capacity() << endl;
00084     cout << "V2 size: " << v2.size() << endl;
00085     cout << "V2 capacity: " << v2.capacity() << endl;
00086     cout << "-----" << endl;
00087     cout << "V1 elements: " << v1 << endl;
00088     cout << "-----" << endl;
00089     cout << "V2 elements: " << v2 << endl;
00090
00091     int x;
00092     char c;
00093     try {
00094         x = v2.pop_back();
00095         c = v1.pop_back();
00096     }
00097     catch(noPop& nopop) {
00098         cout << "Exception occurred: " << nopop.what() << endl;
00099     }
00100
00101
00102     cout << "-----" << endl;
00103     cout << "After deleting last element" << endl;
00104     cout << "Last elements are deleted:" << endl;
00105
00106     cout << x << " " << c << endl;
00107     // V2 elements after pop_back
00108     cout << "v2 elements: " << v2 << endl;
00109     cout << "v1 elements: " << v1 << endl;
00110     cout << "-----" << endl;
00111     cout << "v1 size: " << v1.size() << endl;
00112     cout << "v1 capacity: " << v1.capacity() << endl;
00113     cout << "v2 size: " << v2.size() << endl;
00114     cout << "v2 capacity: " << v2.capacity() << endl;
00115     cout << "-----" << endl;
00116
00117     // Clear test
00118     v1.clear();
00119     cout << "v1 is " << ((v1.empty()) ? "empty": "NOT empty") << endl;
00120     cout << "-----\n";
00121
00122     cout << "v3 before erase: \n" << v3 << endl;
00123     v3.erase(v3.begin());
00124     cout << "v3 after erase its begin: " << endl;
00125     cout << v3 << endl;
00126     cout << "-----\n";
00127
00128     v2.erase(v2.begin(), v2.end());
00129     cout << "v2 is " << ((v2.empty()) ? "empty" : "NOT empty") << endl;
00130
00131     cout << "-----\n";
00132
00133     cout << "v3 before: \n" << v3 << endl;
00134     v3.insert(v3.begin()+2, 5);
00135     cout << "v3 after insert 5 at index 2: \n" << v3 << endl;
00136     cout << "-----\n";
00137
00138     try {
00139         v2[1000] = 5;
00140     } catch(outOfRange& outofrange) {
00141         cout << "Exception occurred: " << outofrange.what() << endl;
00142     }
00143     try {
00144         v1.erase(v1.begin(), v1.end() + 2);
00145     } catch(outOfRange& outofrange) {
00146         cout << "Exception occurred: " << outofrange.what() << endl;
00147     }
00148
00149     try {
00150         v2.pop_back();
00151         v2.pop_back();
00152     }

```

```
00154     v2.pop_back();
00155     v2.pop_back();
00156     v2.pop_back();
00157     // Exception
00158     v2.pop_back();
00159 }catch(noPop& nopop){
00160     cout << "Exception occurred: " << nopop.what() << endl;
00161 }
00162
00163     return 0;
00164 }
```

C:/Users/pc/CLionProjects/YFvector/YFVector.cpp File Reference

```
#include "YFVector.h"  
#include "exceptions.h"
```

Classes

- class `YFVector< T >::iterator< T >`

YFVector.cpp

```
Go to the documentation of this file.00001 #include "YFVector.h"
00002 #include "exceptions.h"
00003
00004
00005
00006 //*****CONSTRUCTORS AND BIG
4*****
00007
00008 // Default constructor
00009 template<typename T>
00010 inline YFVector<T>::YFVector(): arr(new T[1]), _capacity(1), _size(0) {
00011 }
00012
00013 // Parametrized constructor (capacity)
00014 template<typename T>
00015 inline YFVector<T>::YFVector(int n): arr(new T[n]), _capacity(n), _size(0) {
00016 }
00017
00018 // Initialize parametrized constructor (array,size)
00019 template<typename T>
00020 inline YFVector<T>::YFVector(T* data, int n): arr(data), _capacity(n), _size(n){
00021 }
00022
00023 // Copy Constructor
00024 template<typename T>
00025 inline YFVector<T>::YFVector(const YFVector& anotherVec){
00026     copyInternalData(anotherVec);
00027 }
00028
00029 // Destructor
00030 template<typename T>
00031 inline YFVector<T>::~~YFVector() {
00032     delete[] arr;
00033 }
00034
00035 // Private Copy data function
00036 template<typename T>
00037 inline void YFVector<T>::copyInternalData(const YFVector& anotherVec){
00038     arr = new T[anotherVec._capacity];
00039     _size = anotherVec._size;
00040     _capacity = anotherVec._capacity;
00041
00042     for(int i = 0; i < anotherVec._size; ++i){
00043         arr[i] = anotherVec.arr[i];
00044     }
00045 }
00046
00047 // Copy Assignment
00048 template<typename T>
00049 inline YFVector<T>& YFVector<T>::operator= (const YFVector& anotherVec){
00050     if(this != &anotherVec){
00051         delete [] arr;
00052         copyInternalData(anotherVec);
00053     }
00054     return *this;
00055 }
00056
00057
00058 // Move Assignment
00059 template<typename T>
00060 inline YFVector<T>& YFVector<T>::operator=(YFVector&& anotherVec) noexcept{
00061     if(this != &anotherVec){
00062         delete[] arr;
00063         arr = anotherVec.arr;
00064         _size = anotherVec._size;
00065         _capacity = anotherVec._capacity;
00066         anotherVec.arr = nullptr;
00067         anotherVec._size = 0;
00068         anotherVec._capacity = 0;
00069     }
00070     return *this;
00071 }
```

```

00072 //*****ACCESS
OPERATIONS*****
00073
00074 // Index operator
00075 template<typename T>
00076 inline T& YFVector<T>::operator[](int index){
00077     if(index >= _size || index < 0)
00078         throw outOfRange();
00079     else
00080         return arr[index];
00081 }
00082
00083 //*****MODIFYING
OPERATIONS*****
00084
00085 template<typename T>
00086 inline int YFVector<T>::resize() {
00087     T* temp = new T[2 * _capacity];
00088
00089     // copying old array elements to new array
00090     for (int i = 0; i < _capacity; i++) {
00091         temp[i] = arr[i];
00092     }
00093
00094     // deleting previous array
00095     delete[] arr;
00096     _capacity *= 2;
00097     arr = temp;
00098
00099     return _capacity;
00100 }
00101 // Push Back Operator
00102 template<typename T>
00103 inline void YFVector<T>::push_back(T data) {
00104     if (_size == _capacity) {
00105         resize();
00106     }
00107
00108     arr[_size] = data;
00109     _size++;
00110 }
00111
00112 // Pop Back Operator
00113 template<typename T>
00114 inline T YFVector<T>::pop_back() {
00115     if(_size == 0){
00116         throw noPop();
00117     }
00118     else
00119         return arr[--_size];
00120 }
00121
00122 // Clear Vector Elements
00123 template<typename T>
00124 inline void YFVector<T>::clear(){
00125     int i = 0;
00126     while (i < _size){
00127         arr[i].~T();
00128         i++;
00129     }
00130     size = 0;
00131 }
00132
00133 //*****ITERATORS*****
00134
00135
00136 template<class T>
00137 class YFVector<T>::iterator{
00138 private:
00139     T* _curr;
00140
00141 public:
00142     // Parametrized constructor(&_data)
00143     inline explicit iterator(T* _data)
00144         :_curr(_data){}
00145

```



```

00146 // Prefix ++
00147 inline iterator& operator++(){
00148     _curr++;
00149     return *this;
00150 }
00151
00152 // Overloaded operator +=
00153 inline iterator& operator+=(size_t val){
00154     _curr += val;
00155     return *this;
00156 }
00157
00158 // Overloaded operator +
00159 inline iterator operator+(size_t val){
00160     iterator tmp = iterator(*this);
00161     tmp += val;
00162     return tmp;
00163 }
00164
00165 // Prefix --
00166 inline iterator& operator--(){
00167     _curr--;
00168     return *this;
00169 }
00170
00171 // Comparison operator <
00172 inline bool operator<(iterator const &other) const{
00173     return _curr < other._curr;
00174 }
00175
00176 // Comparison operators >
00177 inline bool operator>(iterator const &other) const{
00178     return _curr > other._curr;
00179 }
00180
00181 // Comparison operator ==
00182 inline bool operator==(const iterator& anotherIter) const{
00183     return *_curr == *anotherIter._curr;
00184 }
00185
00186 // Comparison operator !=
00187 inline bool operator!=(const iterator& anotherIter) const{
00188     return *_curr != *anotherIter._curr;
00189 }
00190
00191 // Dereferencing operator *
00192 inline T& operator*(){
00193     return *_curr;
00194 }
00195 };
00196
00197 // Begin Iterator
00198 template<typename T>
00199 inline typename YFVector<T>::iterator YFVector<T>::begin(){
00200     return iterator(&arr[0]);
00201 }
00202
00203
00204 // End iterator
00205 template<typename T>
00206 inline typename YFVector<T>::iterator YFVector<T>::end(){
00207     return begin() + _size;
00208 }
00209
00210 // Erase an interval
00211 template<typename T>
00212 inline void YFVector<T>::erase(iterator iter1, iterator iter2){
00213     if(iter1 < begin() || iter1 > end() || iter2 < iter1 || iter2 > end()){
00214         throw outOfRange();
00215     }
00216
00217     YFVector tmp;
00218     for(auto iter = begin(); iter != iter1; ++iter){
00219         tmp.push_back(*iter);
00220     }
00221
00222     for(auto iter = iter2; iter != end(); ++iter){

```

```

00223         tmp.push_back(*iter);
00224     }
00225     swap(*this, tmp);
00226 }
00227
00228
00229 // Erase item at Iterator
00230 template<typename T>
00231 inline void YFVector<T>::erase(iterator iter){
00232     erase(iter, iter+1);
00233 }
00234
00235
00236
00237 // Insert item at Iterator
00238 template<typename T>
00239 inline void YFVector<T>::insert(iterator iter, T const &val) {
00240     YFVector tmp;
00241     for (auto it1 = begin(); it1 != iter; ++it1) {
00242         tmp.push_back(*it1);
00243     }
00244     tmp.push_back(val);
00245     for (auto it1 = iter; it1 != end(); ++it1) {
00246         tmp.push_back(*it1);
00247     }
00248     swap(*this, tmp);
00249 }
00250
00251
00252 }
00253
00254
00255 //*****COMPARISON
OPERATIONS*****
00256
00257 // operator==
00258 template<typename T>
00259 inline bool YFVector<T>::operator== (const YFVector<T>& v) const {
00260     if( _size == v.size()){
00261         int i = 0;
00262         while (arr[i] == v.arr[i] && i < _size){
00263             i++;
00264         }
00265         if (i == _size)
00266             return true;
00267         return false;
00268     }
00269     return false;
00270 }
00271
00272 // Operator<
00273 template<typename T>
00274 inline bool YFVector<T>::operator<(const YFVector<T>& v) const{
00275     for (int i = 0; i < v.size(); ++i) {
00276         if (arr[i] == v.arr[i]){
00277             continue;
00278         }if(arr[i] < v.arr[i])
00279             return true;
00280     }
00281     return false;
00282 }
00283
00284 //*****CAPACITY
OPERATIONS*****
00285
00286 // Size function
00287 template<typename T>
00288 inline int YFVector<T>::size() const{
00289     return _size;
00290 }
00291
00292 // Capacity function
00293 template<typename T>
00294 inline int YFVector<T>::capacity() const{
00295     return _capacity;
00296 }
00297

```

```
00298 // Empty function
00299 template<typename T>
00300 inline bool YFVector<T>::empty() const{
00301     if(_size == 0)
00302         return true;
00303     return false;
00304 }
00305
00306
00307
00308
00309
00310
00311
00312
```

C:/Users/pc/CLionProjects/YFvector/YFVector.h File Reference

```
#include <bits/stdc++.h>
```

Classes

- class **YFVector**< T >

YFVector.h

```
Go to the documentation of this file.00001 #ifndef YFVECTOR_H_INCLUDED
00002 #define YFVECTOR_H_INCLUDED
00003 #include <bits/stdc++.h>
00004
00005 using namespace std;
00006
00007
00008 template <class T>
00009 class YFVector {
00010     T* arr;
00011     size_t _capacity{};
00012     size_t _size{};
00013     void copyInternalData(const YFVector&);           // to copy internal
data (can't be used by the client)
00014
00015 public:
00016     // Constructors and big 4
00017     YFVector();                                     // default
constructor
00018     explicit YFVector(int);                         // Parametrized
constructor (capacity)
00019     YFVector(T*, int);                             // Initialize
parametrized constructor (array,size)
00020     YFVector(const YFVector&);                       // Copy constructor
00021     ~YFVector();                                    // destructor
00022     YFVector &operator= (const YFVector&);           // Copy assignment
00023     YFVector &operator= (YFVector&&) noexcept ;     // Move assignment
00024
00025     //Access Operations
00026     T& operator[] (int);                           // [] with
outOfBound Check
00027
00028     // Iterator class
00029     class iterator;                                // declaring
iterator class
00030
00031     // Modifying operations
00032     void push back(T);                             // push back
00033     T pop_back();                                  // pop_back
00034     void erase(iterator);
00035     void erase(iterator, iterator);
00036     void clear();                                  // clear
00037     void insert(iterator, T const&);
00038
00039     // Iterators
00040     iterator begin();                               // iterator begin
00041     iterator end();                                 // iterator end
00042
00043     // Comparison operations
00044     bool operator== (const YFVector<T>&) const;     // == operator
00045     bool operator<(const YFVector<T>&) const;       // < operator
00046
00047     // Capacity Operations
00048
00049     int size() const;                              // get size
00050     int capacity() const;                          // get_capacity
00051     int resize();                                  // resize
00052     bool empty() const;                            // empty
00053
00054     // Friends
00055     friend ostream& operator << (ostream& out, YFVector<T>& vt){
00056         for (int i = 0; i < vt.size(); i++) {
00057             out << vt[i] << " ";
00058         }
00059         return out;
00060     }
00061
00062
00063
00064 };
00065
00066
00067
```

```
00068 #endif // YFVECTOR_H_INCLUDED
```

Index

INDEX