My Project

AUTHOR

Version 3

Fri Dec 23 2022

Table of Contents

Hierarchical Index

Class Hierarchy

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

Class Index

Class List

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

File Index

File List

Here is a list of all files with brief descriptions:

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**.

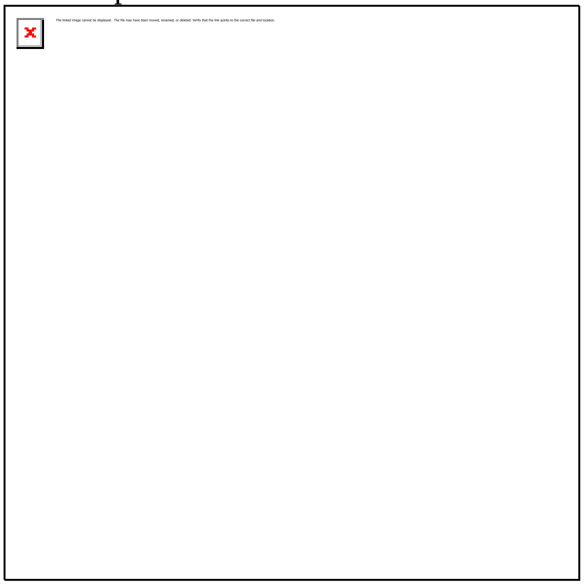
The documentation for this class was generated from the following file:

- 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**.

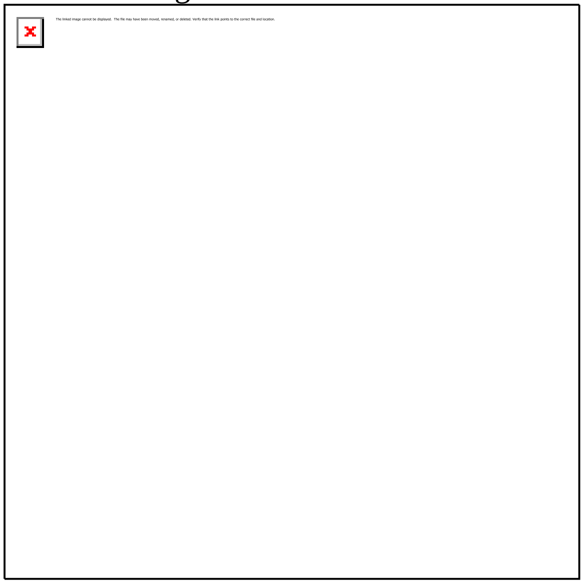
The documentation for this class was generated from the following file:

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

outOfRange Class Reference

`#include <exceptions.h>`

Inheritance diagram for outOfRange:


The linked image cannot be displayed. The file may have been moved, renamed, or deleted. Verify that the link points to the correct file and location.

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.

```
#ifndef EXCEPTIONS_H_INCLUDED
#define EXCEPTIONS_H_INCLUDED
#include <stdexcept>

using namespace std;

class outOfRange : public exception{
public:
    outOfRange() :errormsg{"Attempted to access OUT OF RANGE element"}{}
    string what(){return errormsg;}
private:
    string errormsg;
};

class noPop : public exception{
public:
    noPop():errormsg{"CAN'T pop, there are no elements in the Vector"}{}
    string what(){return errormsg;}
private:
    string errormsg;
};



#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.

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

using namespace std;
int main(){
    int test[5] = {1, 2, 3, 4, 5};
    YFVector<int> v4(test, sizeof(test)/sizeof(test[0]));

    cout << "v4 after passing an array\n";
    cout << v4 ;
    cout << "\n---------------------------------------------\n";


    YFVector<int> v5(v4);
    cout << "v5 initialized by a copy constructor of v4:\n";
    cout << v5 << endl;

    cout << "\nv5 size = " << v5.size() << "\n";
    cout << "v5 capacity = " << v5.capacity() << "\n";
    cout << "------------------------------------------------\n";

    YFVector<int> v6;
    v6 = v5;
    cout << "v6 after copy assignment of v5:\n";
    cout << v6;
    cout << "\nv6 size = " << v6.size();
    cout << "\nv6 capacity = " << v6.capacity() << "\n";

    cout << "V5 is " << ((v5.empty()) ? "empty": "NOT empty") << endl;
    cout << v5 << endl;
    cout << "------------------------------------------------\n";

    YFVector<int> v7;
    v7 = std::move(v6);
    cout << "v7 after move assignment: \n";
    cout << v7 << endl;
    cout << "V6 is " << ((v6.empty()) ? "empty": "NOT empty") << endl;
    cout << "\nv7 size = " << v7.size();
    cout << "\nv7 capacity = " << v7.capacity() << "\n";
    cout << "------------------------------------------------\n";

    YFVector<char> v1(4);
    YFVector<int> v2;
    YFVector<int> v3(4);

    v1.push_back('f');
    v1.push_back('a');
    v1.push_back('t');
    v1.push_back('m');
    // More capacity *2 to push another element
    v1.push_back('a');

    v2.push_back(10);
    v2.push_back(20);
    v2.push_back(30);
    v2.push_back(40);

    v3.push_back(10);
    v3.push_back(2);
    v3.push_back(30);
    v3.push_back(40);

    cout << "Compare between v2 == v3 : "<< ((v2 == v3) ? "True" : "False") << endl;

    YFVector<int> vec4(3);
    YFVector<int> vec5(3);

    vec4.push_back(6);
    vec4.push_back(4);
    vec4.push_back(1);

    vec5.push_back(6);
```

```cpp
    vec5.push_back(4);
    vec5.push_back(2);
    cout << "--------------------------------------------"<< endl;
    cout << "Is v4 < 45 ?" <<((vec4 < vec5) ? " Yes": " NO" ) << endl;
    cout << "--------------------------------------------" << endl;


    cout << "V1 size: " << v1.size() << endl;
    cout << "V1 capacity: " << v1.capacity() << endl;
    cout << "V2 size: " << v2.size() << endl;
    cout << "V2 capacity: " << v2.capacity() << endl;
    cout << "--------------------------------------------"<< endl;
    cout << "V1 elements: "<< v1 << endl;
    cout << "--------------------------------------------"<< endl;
    cout << "V2 elements: "<< v2 << endl;

    int x;
    char c;
    try {
        x = v2.pop_back();
        c = v1.pop_back();
    }
    catch(noPop& nopop){
        cout << "Exception occurred: " << nopop.what() << endl;
    }


    cout << "--------------------------------------------"<< endl;
    cout << "After deleting last element" << endl;
    cout << "Last elements are deleted:" << endl;

    cout << x << " " << c << endl;
    // V2 elements after pop_back
    cout << "v2 elements: " << v2 << endl;
    cout << "v1 elements: " << v1 << endl;
    cout << "--------------------------------------------"<< endl;
    cout << "v1 size: " << v1.size() << endl;
    cout << "v1 capacity: " << v1.capacity() << endl;
    cout << "v2 size: " << v2.size() << endl;
    cout << "v2 capacity: " << v2.capacity() << endl;
    cout << "--------------------------------------------"<< endl;

    // Clear test
    v1.clear();
    cout << "v1 is " << ((v1.empty()) ? "empty": "NOT empty") << endl;
    cout << "----------------------------------------\n";

    cout << "v3 before erase: \n" << v3 << endl;
    v3.erase(v3.begin());
    cout << "v3 after erase its begin: " << endl;
    cout << v3 << endl;
    cout << "----------------------------------------\n";

    v2.erase(v2.begin(), v2.end());
    cout << "v2 is " << ((v2.empty()) ? "empty" : "NOT empty") << endl;

    cout << "----------------------------------------\n";

    cout << "v3 before: \n" << v3 << endl;
    v3.insert(v3.begin()+2, 5);
    cout << "v3 after insert 5 at index 2: \n" << v3 << endl;
    cout << "----------------------------------------\n";

    try {
        v2[1000] = 5;
    }catch(outOfRange& outofrange) {
        cout << "Exception occurred: " << outofrange.what() << endl;
    }
    try {
        v1.erase(v1.begin(), v1.end() + 2);
    }catch(outOfRange& outofrange) {
        cout << "Exception occurred: " << outofrange.what() << endl;
    }

    try {
        v2.pop_back();
        v2.pop_back();
        v2.pop_back();
        v2.pop_back();
        v2.pop_back();
        // Exception
        v2.pop_back();
    }catch(noPop& nopop){
        cout << "Exception occurred: " << nopop.what() << endl;
    }

    return 0;
}
```

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.

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



//*******************************CONSTRUCTORS AND BIG
4*********************************************

// Default constructor
template<typename T>
inline YFVector<T>::YFVector(): arr(new T[1]), _capacity(1),_size(0) {
}

// Parametrized constructor (capacity)
template<typename T>
inline YFVector<T>::YFVector(int n): arr(new T[n]), _capacity(n), _size(0) {
}

// Initialize parametrized constructor (array,size)
template<typename T>
inline YFVector<T>::YFVector(T* data, int n): arr(data), _capacity(n), _size(n){
}

// Copy Constructor
template<typename T>
inline YFVector<T>::YFVector(const YFVector& anotherVec){
        copyInternalData(anotherVec);
}

// Destructor
template<typename T>
inline YFVector<T>::~YFVector() {
    delete[] arr;
}

// Private Copy data function
template<typename T>
inline void YFVector<T>::copyInternalData(const YFVector& anotherVec){
    arr = new T[anotherVec._capacity];
    _size = anotherVec._size;
    _capacity = anotherVec._capacity;

    for(int i = 0; i < anotherVec._size; ++i){
        arr[i] = anotherVec.arr[i];
    }
}

// Copy Assignment
template<typename T>
inline YFVector<T>& YFVector<T>::operator= (const YFVector& anotherVec){
    if(this != &anotherVec){
        delete [] arr;
        copyInternalData(anotherVec);
    }
    return *this;

    }

// Move Assignment
template<typename T>
inline YFVector<T>& YFVector<T>::operator=(YFVector&& anotherVec) noexcept{
    if(this != &anotherVec){
        delete[] arr;
        arr = anotherVec.arr;
        _size = anotherVec._size;
        _capacity = anotherVec._capacity;
        anotherVec.arr = nullptr;
        anotherVec._size = 0;
        anotherVec._capacity = 0;
    }
    return *this;
}
```

```cpp
//*****************************ACCESS OPERATIONS***********************************

// Index operator
template<typename T>
inline T& YFVector<T>::operator[](int index){
        if(index >= _size || index < 0)
            throw outOfRange();
        else
            return arr[index];
}

//***************************************MODIFYING OPERATIONS************************************

template<typename T>
inline int YFVector<T>::resize() {
    T* temp = new T[2 * _capacity];

    // copying old array elements to new array
    for (int i = 0; i < _capacity; i++) {
        temp[i] = arr[i];
    }

    // deleting previous array
    delete[] arr;
    _capacity *= 2;
    arr = temp;

    return _capacity;
}
// Push Back Operator
template<typename T>
inline void YFVector<T>::push_back(T data) {
    if (_size == _capacity) {
        resize();
    }

    arr[_size] = data;
    _size++;
}

// Pop Back Operator
template<typename T>
inline T YFVector<T>::pop_back() {
        if(_size == 0){
            throw noPop();
        }
        else
            return arr[--_size];

}
// Clear Vector Elements
template<typename T>
inline void YFVector<T>::clear(){
    int i = 0;
    while (i < _size){
        arr[i].~T();
        i++;
    }
    _size = 0;
}

//***********************************ITERATORS**************************************************


template<class T>
class YFVector<T>::iterator{
private:
    T* _curr;

public:
    // Parametrized constructor(&_data)
    inline explicit iterator(T* _data)
        :_curr(_data){}

    // Prefix ++
    inline iterator& operator++(){
        _curr++;
        return *this;
    }

    // Overloaded operator +=
    inline iterator& operator+=(size_t val){
        _curr += val;
        return *this;
    }

    // Overloaded operator +
    inline iterator operator+(size_t val){
         iterator tmp = iterator(*this);
         tmp += val;
         return tmp;
     }

    // Prefix --
    inline iterator& operator--(){
        _curr--;
        return *this;
    }

    // Comparison operator <
    inline bool operator<(iterator const &other) const{
            return _curr < other._curr;
    }

    // Comparison operators >
    inline bool operator>(iterator const &other) const{
        return _curr > other._curr;
    }

    // Comparison operator ==
    inline bool operator==(const iterator& anotherIter) const{
        return *_curr == *anotherIter._curr;
    }

    // Comparison operator !=
    inline bool operator!=(const iterator& anotherIter) const{
        return *_curr != *anotherIter._curr;
    }

    // Dereferencing operator *
    inline T& operator*(){
        return *_curr;
    }
};

// Begin Iterator
template<typename T>
inline typename YFVector<T>::iterator YFVector<T>::begin(){
    return iterator(&arr[0]);
}


// End iterator
template<typename T>
inline typename YFVector<T>::iterator YFVector<T>::end(){
    return begin() + _size;
}

// Erase an interval
template<typename T>
inline void YFVector<T>::erase(iterator iter1, iterator iter2){
    if(iter1 < begin() || iter1 > end() || iter2 < iter1 || iter2 > end()){
        throw outOfRange();
    }

    YFVector tmp;
    for(auto iter = begin(); iter != iter1; ++iter){
        tmp.push_back(*iter);
    }

    for(auto iter = iter2; iter != end(); ++iter){
        tmp.push_back(*iter);
    }
    swap(*this, tmp);
}


// Erase item at Iterator
template<typename T>
inline void YFVector<T>::erase(iterator iter){
    erase(iter, iter+1);
}



// Insert item at Iterator
template<typename T>
inline void YFVector<T>::insert(iterator iter, T const &val) {
    YFVector tmp;
    for (auto it1 = begin(); it1 != iter; ++it1) {
        tmp.push_back(*it1);
    }

    tmp.push_back(val);

    for (auto it1 = iter; it1 != end();  ++it1) {
        tmp.push_back(*it1);
    }

    swap(*this, tmp);
}


//******************************COMPARISON
OPERATIONS*******************************************

// operator==
template<typename T>
inline bool YFVector<T>::operator== (const YFVector<T>& v) const {
    if( _size == v.size()){
        int i = 0;
        while (arr[i] == v.arr[i] && i < _size){
            i++;
        }
        if (i == _size)
            return true;
        return false;
    }
    return false;
}

// Operator<
template<typename T>
inline bool YFVector<T>::operator<(const YFVector<T>& v) const{
    for (int i = 0; i < v.size(); ++i) {
        if (arr[i] == v.arr[i]){
            continue;
        }if(arr[i] < v.arr[i])
            return true;
    }
    return false;

}
//**********************************CAPACITY
OPERATIONS***********************************************

// Size function
template<typename T>
inline int YFVector<T>::size() const{
    return _size;
}

// Capacity function
template<typename T>
inline int YFVector<T>::capacity() const{
    return _capacity;
}

```

```
// Empty function
template<typename T>
inline bool YFVector<T>::empty() const{
    if(_size == 0)
        return true;
    return false;
}
```

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.

```cpp
#ifndef YFVECTOR_H_INCLUDED
#define YFVECTOR_H_INCLUDED
#include <bits/stdc++.h>

using namespace std;


template <class T>
class YFVector {
    T* arr;
    size_t _capacity{};
    size_t _size{};
    void copyInternalData(const YFVector&);                   // to copy internal data (can't be used by the client)

public:
    // Constructors and big 4
    YFVector();                                               // default constructor
    explicit YFVector(int);                                   // Parametrized constructor (capacity)
    YFVector(T*, int);                                        // Initialize parametrized constructor (array,size)
    YFVector(const YFVector&);                                // Copy constructor
    ~YFVector();                                              // destructor
    YFVector &operator= (const YFVector&);                    // Copy assignment
    YFVector &operator= (YFVector&&) noexcept ;               // Move assignment

    //Access Operations
    T& operator[](int);                                       // [] with outOfBound Check

    // Iterator class
    class iterator;                                           // declaring iterator class

    // Modifying operations
    void push_back(T);                                        // push_back
    T pop_back();                                             // pop_back
    void erase(iterator);
    void erase(iterator, iterator);
    void clear();                                             // clear
    void insert(iterator, T const&);

    // Iterators
    iterator begin();                                         // iterator begin
    iterator end();                                           // iterator end

    // Comparison operations
    bool operator== (const YFVector<T>&) const;               // == operator
    bool operator<(const YFVector<T>&) const;                 // < operator

    // Capacity Operations

    int size() const;                                         // get_size
    int capacity() const;                                     // get_capacity
    int resize();                                             // resize
    bool empty() const;                                       // empty

    // Friends
    friend ostream& operator << (ostream& out, YFVector<T>& vt){
        for (int i = 0; i < vt.size(); i++) {
            out << vt[i] << " ";
        }
        return out;
    }



};



```

```
#endif // YFVECTOR_H_INCLUDED
```

Index

INDEX