My Project

Создано системой Doxygen 1.9.6

Глава 1

Иерархический список классов

1.1 Иерархия классов

Иерархия классов.

https://github.com/Faton6/MP_Lab5

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список документированных файлов.

Глава 4

Классы

4.1 Класс ADT

Открытые члены

- virtual std::ostream & print (std::ostream &out) const

Друзья

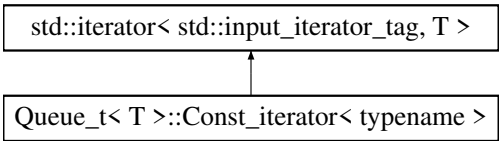
- std::ostream & operator<< (std::ostream &out, const ADT &right)

Объявления и описания членов класса находятся в файле:

- ADT.h

4.2 Шаблон класса Queue_t< T >::Const_iterator< typename >

Граф наследования:Queue_t< T >::Const_iterator< typename >:



Открытые члены

- Const_iterator (T *ptr)
- Const_iterator (const Const_iterator &right)
- const T & operator* () const
- const T & operator[] (size_t pos) const
- bool operator!= (const Const_iterator &right) const
- bool operator== (const Const_iterator &right) const
- Const_iterator operator+ (const Const_iterator< T > &right) const
- Const_iterator operator+ (int right) const
- Const_iterator operator- (const Const_iterator< T > &right) const
- Const_iterator operator- (int right) const
- Const_iterator & operator++ ()
- Const_iterator operator++ (int)
- Const_iterator & operator-- ()
- Const_iterator operator-- (int)
- Const_iterator (T *ptr)
- Const_iterator (const Const_iterator &right)
- const T & operator* () const
- const T & operator[] (size_t pos) const
- bool operator!= (const Const_iterator &right) const
- bool operator== (const Const_iterator &right) const
- Const_iterator operator+ (const Const_iterator< T > &right) const
- Const_iterator operator+ (int right) const
- Const_iterator operator- (const Const_iterator< T > &right) const
- Const_iterator operator- (int right) const
- Const_iterator & operator++ ()
- Const_iterator operator++ (int)
- Const_iterator & operator-- ()
- Const_iterator operator-- (int)

Открытые атрибуты

- T * ptr

Объявления и описания членов классов находятся в файлах:

- queue_t.cpp
- queue_t.hpp

4.3 Шаблон класса Queue_t< T >::Iterator_My< typename >

Открытые типы

- using iterator_category = std::forward_iterator_tag
- using value_type = T
- using difference_type = T
- using pointer = T *
- using reference = T &
- using iterator_category = std::forward_iterator_tag
- using value_type = T
- using difference_type = T
- using pointer = T *
- using reference = T &

Открытые члены

- Iterator_My (T *ptr)
- Iterator_My (const Iterator_My &right)
- T & operator* () const
- T & operator[] (size_t pos) const
- bool operator!= (const Iterator_My< T > &right)
- bool operator== (const Iterator_My< T > &right)
- bool operator< (const Iterator_My< T > &right)
- Iterator_My operator+ (const Iterator_My< T > &right) const
- Iterator_My operator+ (int right) const
- Iterator_My operator- (const Iterator_My< T > &right) const
- Iterator_My operator- (int right) const
- Iterator_My & operator++ ()
- Iterator_My operator++ (int)
- Iterator_My & operator-- ()
- Iterator_My operator-- (int)
- Iterator_My (T *ptr)
- Iterator_My (const Iterator_My &right)
- T & operator* () const
- T & operator[] (size_t pos) const
- bool operator!= (const Iterator_My< T > &right)
- bool operator== (const Iterator_My< T > &right)
- bool operator< (const Iterator_My< T > &right)
- Iterator_My operator+ (const Iterator_My< T > &right) const
- Iterator_My operator+ (int right) const
- Iterator_My operator- (const Iterator_My< T > &right) const
- Iterator_My operator- (int right) const
- Iterator_My & operator++ ()
- Iterator_My operator++ (int)
- Iterator_My & operator-- ()
- Iterator_My operator-- (int)

Открытые атрибуты

- T * ptr

Объявления и описания членов классов находятся в файлах:

- queue_t.cpp
- queue_t.hpp

4.4 Шаблон класса Queue_t< T >

Классы

- class Const_iterator
- class Iterator_My

Открытые члены

- Queue_t (size_t size)
- Queue_t (const Queue_t &right)
- Iterator_My< T > begin ()
- Iterator_My< T > end ()
- Const_iterator< T > begin () const
- Const_iterator< T > end () const
- size_t size () const
- T pop_back ()
- void push_front (T value)
- void erase (size_t pos)
- void emplace (size_t, T value)
- void insert (size_t pos, T value)
- void clear ()
- bool empty () const
- const Queue_t & operator= (const Queue_t &right)
- T & operator[] (size_t pos)
- const T & operator[] (Iterator_My< T > pos) const
- Queue_t (size_t size)
- Queue_t (const Queue_t &right)
- Iterator_My< T > begin ()
- Iterator_My< T > end ()
- Const_iterator< T > begin () const
- Const_iterator< T > end () const
- size_t size () const
- T pop_back ()
- void push_front (T value)
- void erase (size_t pos)
- void emplace (size_t, T value)
- void insert (size_t pos, T value)
- void clear ()
- bool empty () const
- const Queue_t & operator= (const Queue_t &right)
- T & operator[] (size_t pos)
- const T & operator[] (Iterator_My< T > pos) const

Объявления и описания членов классов находятся в файлах:

- queue_t.cpp
- queue_t.hpp

Глава 5

Файлы

5.1 ADT.h

```
00001 #ifndef ADT_h
00002 #define ADT_h
00003
00004 #include <iostream>
00005
00006 class ADT
00007 {
00008     //
00009     public:
00010         virtual ~ADT() = default;
00011         virtual std::ostream &print(std::ostream &out) const { return out; }
00012         friend std::ostream &operator«(std::ostream& out, const ADT& right)
00013         {
00014             return right.print(out);
00015         }
00016
00017 };
00018
00019 #endif // ADT_h
00020
```

5.2 queue_t.hpp

```
00001 #ifndef Queue_t_h
00002 #define Queue_t_h
00003
00004 #include "ADT.h"
00005
00006
00007 template <typename T>
00008 class Queue_t
00009 {
00010     private:
00011
00012         size_t _size;    // размер очереди
00013         size_t _quantity;// количество элементов
00014         T* arr;
00015
00016         void resize(size_t new_size);
00017
00018     public:
00019
00020         template <typename>
00021         class Iterator_My;
00022         template <typename>
00023         class Const_iterator;
00024
00025         Queue_t() = delete;
00026         Queue_t(size_t size);
00027         Queue_t(const Queue_t &right);
00028         ~Queue_t();
00029
00030         Iterator_My<T> begin();
00031         Iterator_My<T> end();
```

```
00032         Const_iterator<T> begin() const;
00033         Const_iterator<T> end() const;
00034         size_t size() const;
00035         T pop_back();
00036         void push_front(T value);
00037         void erase(size_t pos);
00038         void emplace(size_t, T value);
00039         void insert(size_t pos, T value);
00040         void clear();
00041         bool empty() const;
00042 
00043         const Queue_t &operator= (const Queue_t &right);
00044         T &operator[](size_t pos);
00045         const T &operator[](Iterator_My<T> pos) const { return *pos.ptr; }
00046 
00047         template <typename>
00048         class Iterator_My //: public std::iterator<std::input_iterator_tag, T>
00049         {
00050             public:
00051                 using iterator_category = std::forward_iterator_tag;
00052                 using value_type = T;
00053                 using difference_type = T;
00054                 using pointer = T*;
00055                 using reference = T&;
00056                 T *ptr;
00057 
00058                 Iterator_My() : ptr(nullptr) {}
00059                 Iterator_My(T *ptr) : ptr(ptr) {}
00060 
00061                 Iterator_My(const Iterator_My &right) : ptr(right.ptr) {}
00062 
00063                 T &operator*() const { return *ptr; }
00064                 T &operator[](size_t pos) const { return ptr[pos]; }
00065                 //T *operator->() const { return ptr->data; }
00066                 bool operator!=(const Iterator_My<T> &right) {return ptr != right.ptr; }
00067                 bool operator==(const Iterator_My<T> &right) {return ptr == right.ptr; }
00068                 bool operator<(const Iterator_My<T> &right)  {return ptr < right.ptr; }
00069                 Iterator_My operator+(const Iterator_My<T> &right) const { return this->ptr + right.ptr; }
00070                 Iterator_My operator+(int right) const { return Iterator_My(this->ptr + right); }
00071                 Iterator_My operator-(const Iterator_My<T> &right) const { return this->ptr - right.ptr; }
00072                 Iterator_My operator-(int right) const { return Iterator_My(this->ptr - right); }
00073 
00074                 Iterator_My &operator++() {++ptr; return *this; }
00075                 Iterator_My operator++(int) {T *var_ptr = this->ptr; ++ptr; return *var_ptr; }
00076                 Iterator_My &operator--() {--ptr; return *this; }
00077                 Iterator_My operator--(int) {T *var_ptr = this->ptr; --ptr; return *var_ptr; }
00078         };
00079 
00080         template <typename>
00081         class Const_iterator : public std::iterator<std::input_iterator_tag, T>
00082         {
00083             public:
00084                 T *ptr;
00085 
00086                 Const_iterator() : ptr(nullptr) {}
00087                 Const_iterator(T *ptr) : ptr(ptr) {}
00088 
00089                 Const_iterator(const Const_iterator &right) : ptr(right.ptr) {}
00090 
00091                 const T &operator*() const { return *ptr; }
00092                 const T &operator[](size_t pos) const { return ptr[pos]; }
00093                 //T *operator->() const { return ptr->data; }
00094                 bool operator!=(const Const_iterator &right) const {return ptr != right.ptr;}
00095                 bool operator==(const Const_iterator &right) const {return ptr == right.ptr;}
00096                 Const_iterator operator+(const Const_iterator<T> &right) const { return this->ptr + right.ptr; }
00097                 Const_iterator operator+(int right) const { return Const_iterator(this->ptr + right); }
00098                 Const_iterator operator-(const Const_iterator<T> &right) const { return this->ptr - right.ptr; }
00099                 Const_iterator operator-(int right) const { return Const_iterator(this->ptr - right); }
00100                 Const_iterator &operator++() {++ptr; return *this; }
00101                 Const_iterator operator++(int) {T *var_ptr = this->ptr; ++ptr; return *var_ptr; }
00102                 Const_iterator &operator--() {--ptr; return *this; }
00103                 Const_iterator operator--(int) {T *var_ptr = this->ptr; --ptr; return *var_ptr; }
00104         };
00105 
00106 };
00107 
00108 template <typename T>
00109 Queue_t<T>::Queue_t(size_t size) : _size(size)
00110 {
00111     arr = new T[_size];
00112     _quantity = 0;
00113 }
00114 
00115 template <typename T>
00116 Queue_t<T>::Queue_t(const Queue_t<T> &right)
00117 {
00118     if (arr != nullptr)
```

```
00119     {
00120         delete [] arr;
00121         arr = nullptr;
00122     }
00123 
00124     this->_size = right._size;
00125     this->_quantity = right._quantity;
00126     this->arr = new T[this->_size];
00127     for (size_t i = 0; i < _quantity; ++i)
00128     {
00129         arr[i] = right.arr[i];
00130     }
00131 
00132 }
00133 
00134 template <typename T>
00135 Queue_t<T>::~Queue_t()
00136 {
00137     if (arr != nullptr)
00138     {
00139         delete [] arr;
00140         arr = nullptr;
00141     }
00142 
00143 }
00144 
00145 template <typename T>
00146 size_t Queue_t<T>::size() const { return this->_quantity; }
00147 
00148 
00149 template <typename T>
00150 void Queue_t<T>::resize(size_t new_size)
00151 {
00152     T *new_arr;
00153 
00154     new_arr = new T[new_size];
00155     for (size_t i = 0; i < this->_quantity; ++i )
00156     {
00157         new_arr[i] = arr[i];
00158     }
00159 
00160     delete [] arr;
00161     arr = new_arr;
00162     this->_size = new_size;
00163 
00164 }
00165 
00166 
00167 template <typename T>
00168 T Queue_t<T>::pop_back()
00169 {
00170     if (arr != nullptr && _quantity > 0)
00171     {
00172         T val = arr[_quantity-1];
00173         --_quantity;
00174         return val;
00175     }
00176     else return nullptr;
00177 }
00178 
00179 template <typename T>
00180 void Queue_t<T>::push_front(T value)
00181 {
00182     this->insert(0, value);
00183 }
00184 
00185 template <typename T>
00186 void Queue_t<T>::emplace(size_t pos, T value)
00187 {
00188     arr[pos] = value;
00189 }
00190 
00191 template <typename T>
00192 void Queue_t<T>::insert(size_t pos, T value)
00193 {
00194     if (empty())
00195     {
00196         arr[0] = value;
00197         ++_quantity;
00198     }
00199     else if (_quantity < _size)
00200     {
00201         ++_quantity;
00202         T var = arr[pos];
00203         arr[pos] = value;
00204         T qwa;
00205         for (size_t i = pos+1; i < _quantity; ++i)
```

```
00206         {
00207             qwa = arr[i];
00208             arr[i] = var;
00209             var = qwa;
00210         }
00211     }
00212     else if (_quantity == _size)
00213     {
00214
00215         for (size_t i = 0; i < pos; ++i)
00216             arr[i+1] = arr[i];
00217         arr[0] = arr[_quantity-1];
00218         T var = arr[pos];
00219         arr[pos] = value;
00220         T qwa;
00221         for (size_t i = pos+1; i < _quantity; ++i)
00222         {
00223             qwa = arr[i];
00224             arr[i] = var;
00225             var = qwa;
00226         }
00227     }
00228 }
00229
00230
00231 template <typename T>
00232 void Queue_t<T>::erase(size_t pos)
00233 {
00234     --_quantity;
00235     for (size_t i = pos; i < _quantity; ++i)
00236         arr[i] = arr[i+1];
00237 }
00238
00239 template <typename T>
00240 void Queue_t<T>::clear()
00241 {
00242     delete [] arr;
00243     arr = nullptr;
00244     _size = 0;
00245     _quantity = 0;
00246 }
00247
00248 template <typename T>
00249 bool Queue_t<T>::empty() const { return _quantity == 0; }
00250
00251 template <typename T>
00252 const Queue_t<T> &Queue_t<T>::operator= (const Queue_t<T>& right)
00253 {
00254     if (_size != right._size)
00255     {
00256         this->_quantity = right._quantity;
00257         if (arr != nullptr)
00258         {
00259             delete [] arr;
00260             arr = nullptr;
00261         }
00262         this->_size = right._size;
00263         this->arr = new T[right._size];
00264     }
00265     for ( size_t i = 0; i < right._quantity; ++i )
00266         arr[i] = right.arr[i];
00267     return *this;
00268 }
00269
00270 template <typename T>
00271 T &Queue_t<T>::operator[](size_t pos) { return arr[pos]; }
00272
00273
00274 template <typename T>
00275 Queue_t<T>::Iterator_My<T> Queue_t<T>::begin()
00276 {
00277     return Iterator_My<T>(arr);
00278 }
00279 template <typename T>
00280 Queue_t<T>::Iterator_My<T> Queue_t<T>::end()
00281 {
00282     return Iterator_My<T>(arr + this->_size);
00283 }
00284
00285 template <typename T>
00286 Queue_t<T>::Const_iterator<T> Queue_t<T>::begin() const
00287 {
00288     return Const_iterator<T>(arr[0]);
00289 }
00290 template <typename T>
00291 Queue_t<T>::Const_iterator<T> Queue_t<T>::end() const
00292 {
```

```
00293     return Const_iterator<T>(arr[this->_size]);
00294 }
00295
00296
00297
00298 #endif // Queue_t_h
00299
00300 int main()
00301 {
00302     Queue_t<int> qwa_queue{6};
00303     std::cout « "TEST\n";
00304
00305 }
```

Предметный указатель