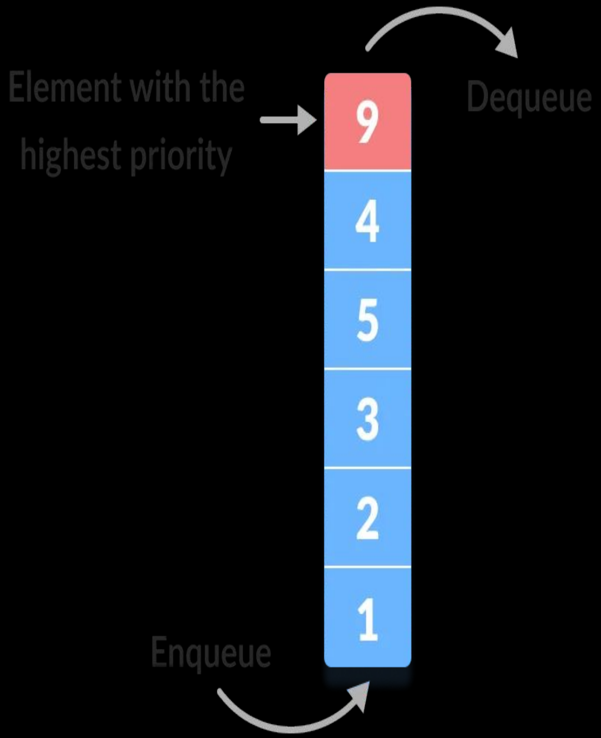
PRIORITY QUEUE

Harold Alejandro Villanueva Borda

¿Qué es?



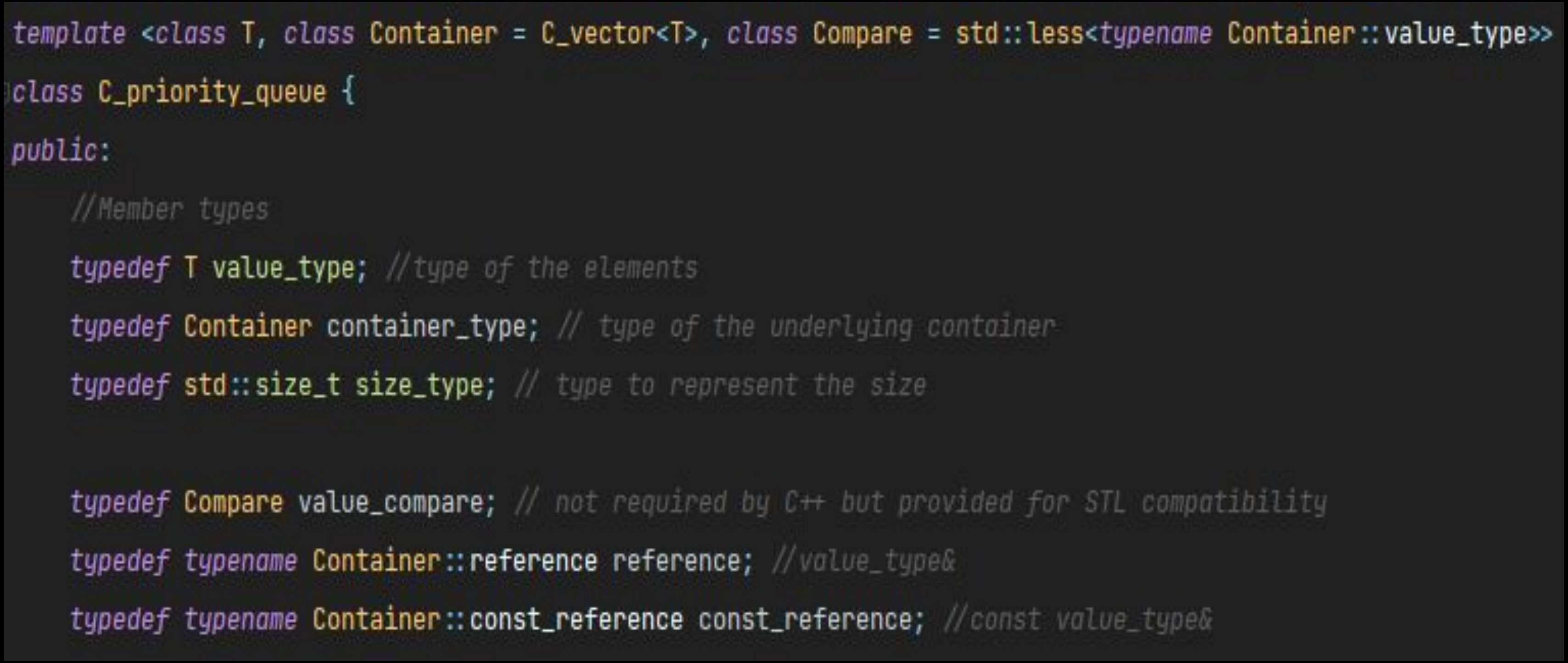
Algunas Aplicaciones

- Dijkstra Algorithm
- Prim Algorithm
- Compresión de datos:

  - Huffman
- Inteligencia Artificial:

  - A* Search Algorithm
- Heap Sort

Templates y Member types

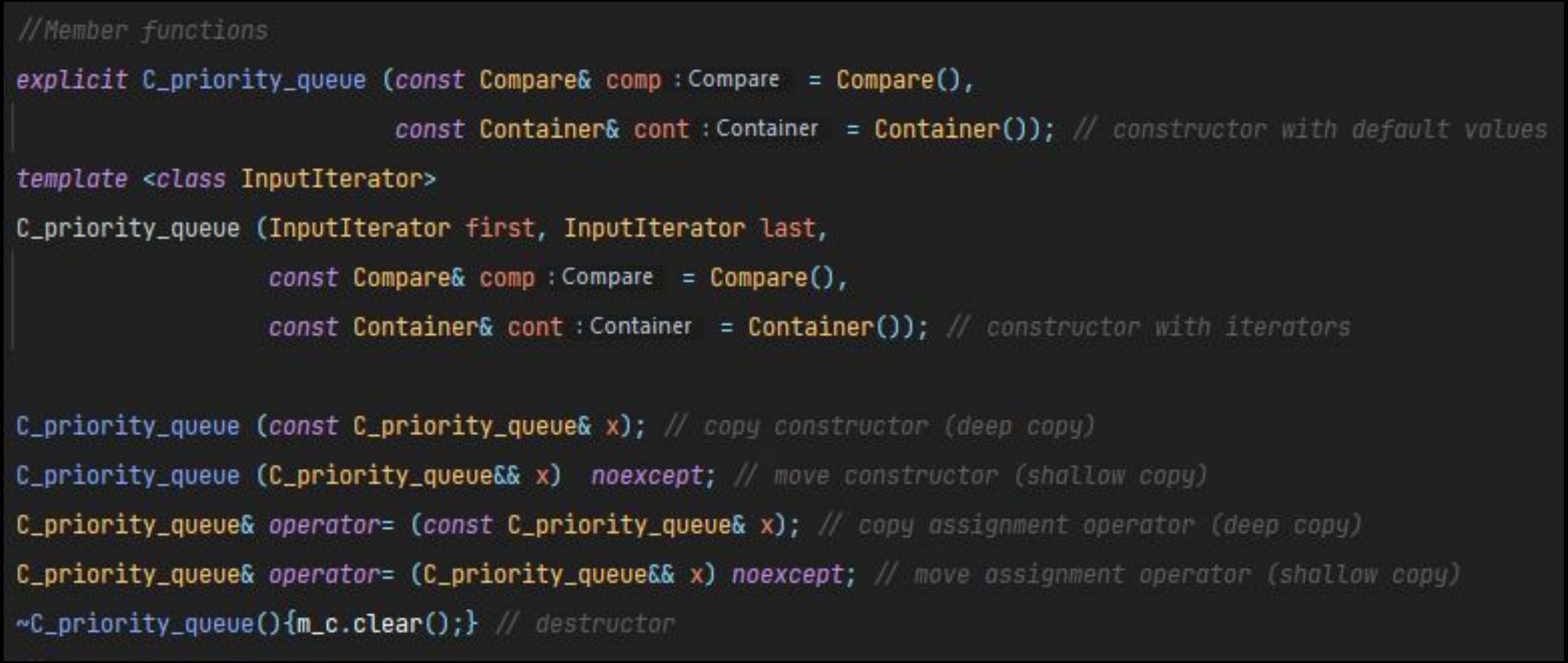


```cpp
template <class T, class Container = C_vector<T>, class Compare = std::less<typename Container::value_type>>
class C_priority_queue {
public:
    //Member types
    typedef T value_type; //type of the elements
    typedef Container container_type; // type of the underlying container
    typedef std::size_t size_type; // type to represent the size

    typedef Compare value_compare; // not required by C++ but provided for STL compatibility
    typedef typename Container::reference reference; //value_type&
    typedef typename Container::const_reference const_reference; //const value_type&
```

https://cplusplus.com/reference/queue/priority_queue/

Member functions

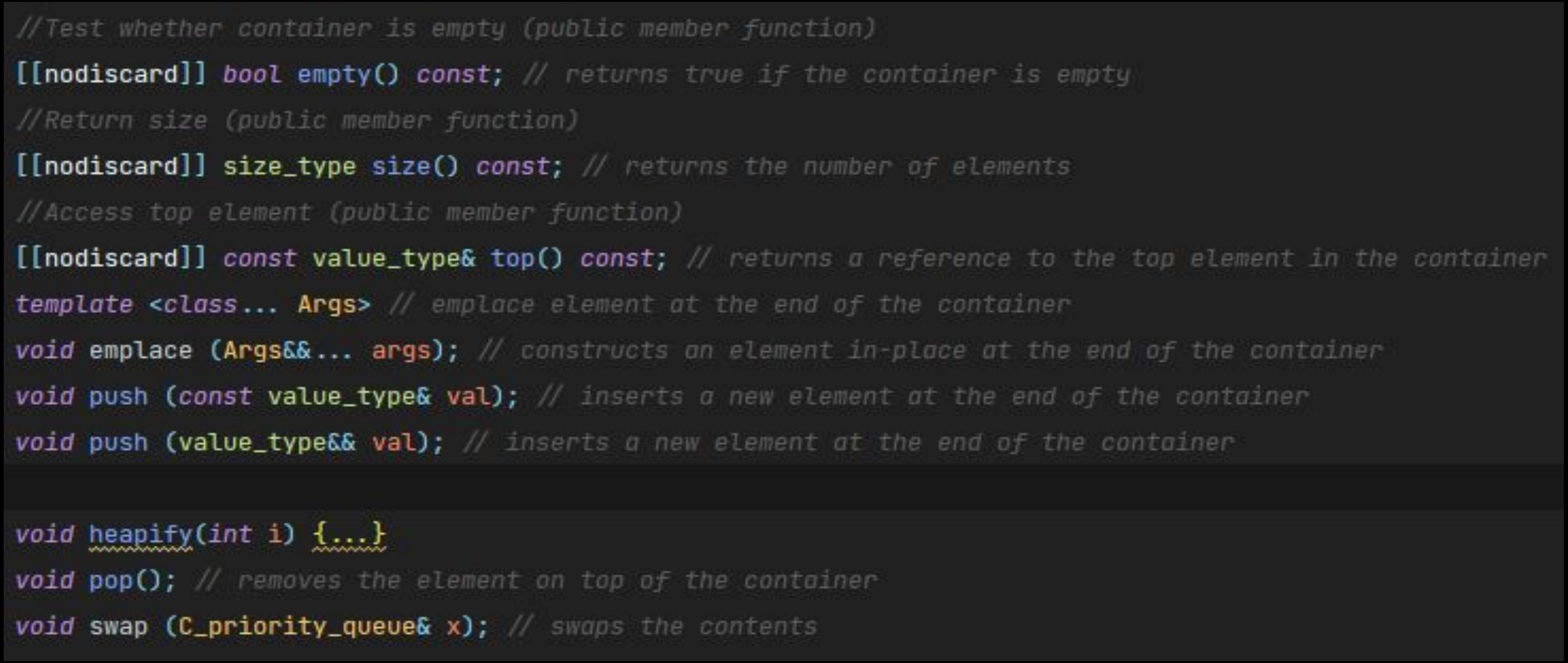


```
//Member functions
explicit C_priority_queue (const Compare& comp :Compare  = Compare(),
                           const Container& cont :Container  = Container()); // constructor with default values
template <class InputIterator>
C_priority_queue (InputIterator first, InputIterator last,
                  const Compare& comp :Compare  = Compare(),
                  const Container& cont :Container  = Container()); // constructor with iterators

C_priority_queue (const C_priority_queue& x); // copy constructor (deep copy)
C_priority_queue (C_priority_queue&& x)  noexcept; // move constructor (shallow copy)
C_priority_queue& operator= (const C_priority_queue& x); // copy assignment operator (deep copy)
C_priority_queue& operator= (C_priority_queue&& x) noexcept; // move assignment operator (shallow copy)
~C_priority_queue(){m_c.clear();} // destructor
```

https://cplusplus.com/reference/queue/priority_queue/

Member functions

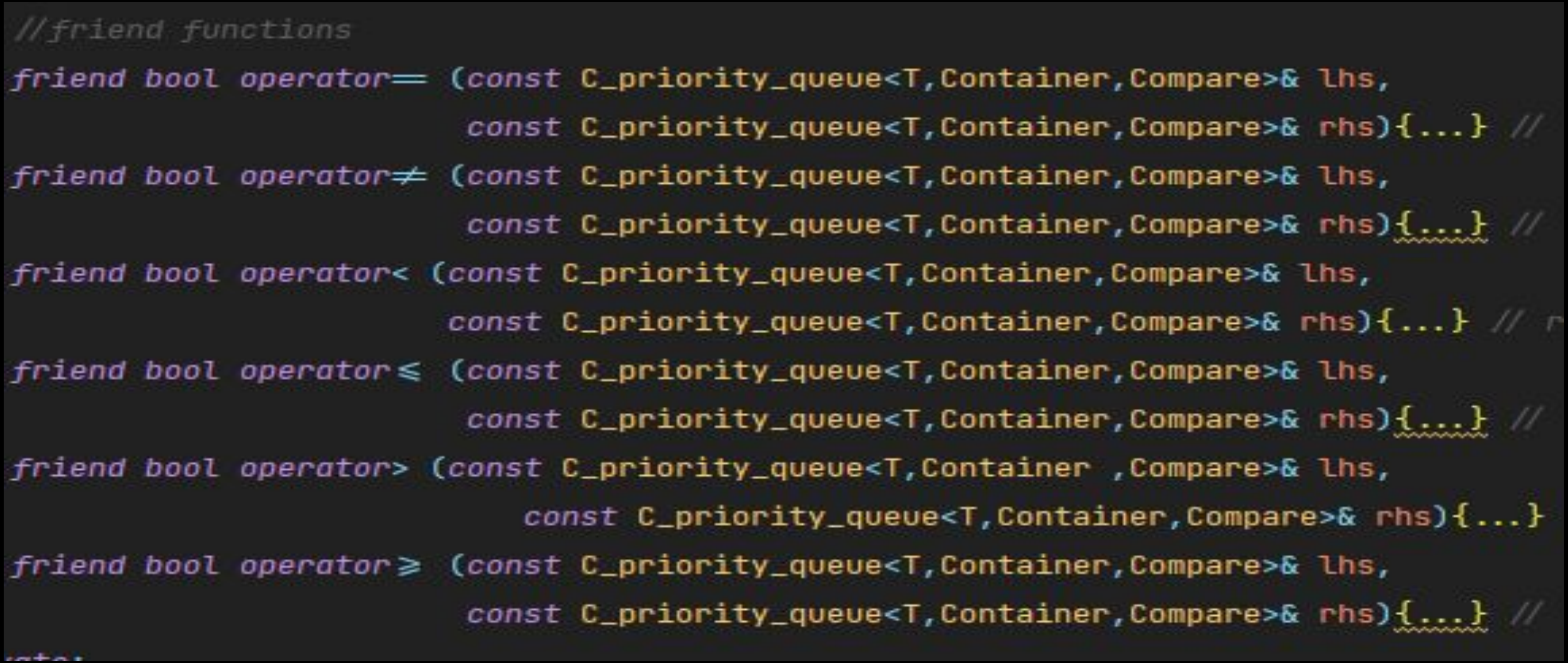


```
//Test whether container is empty (public member function)
[[nodiscard]] bool empty() const; // returns true if the container is empty
//Return size (public member function)
[[nodiscard]] size_type size() const; // returns the number of elements
//Access top element (public member function)
[[nodiscard]] const value_type& top() const; // returns a reference to the top element in the container
template <class... Args> // emplace element at the end of the container
void emplace (Args&&... args); // constructs an element in-place at the end of the container
void push (const value_type& val); // inserts a new element at the end of the container
void push (value_type&& val); // inserts a new element at the end of the container

void heapify(int i) {...}
void pop(); // removes the element on top of the container
void swap (C_priority_queue& x); // swaps the contents
```

https://cplusplus.com/reference/queue/priority_queue/

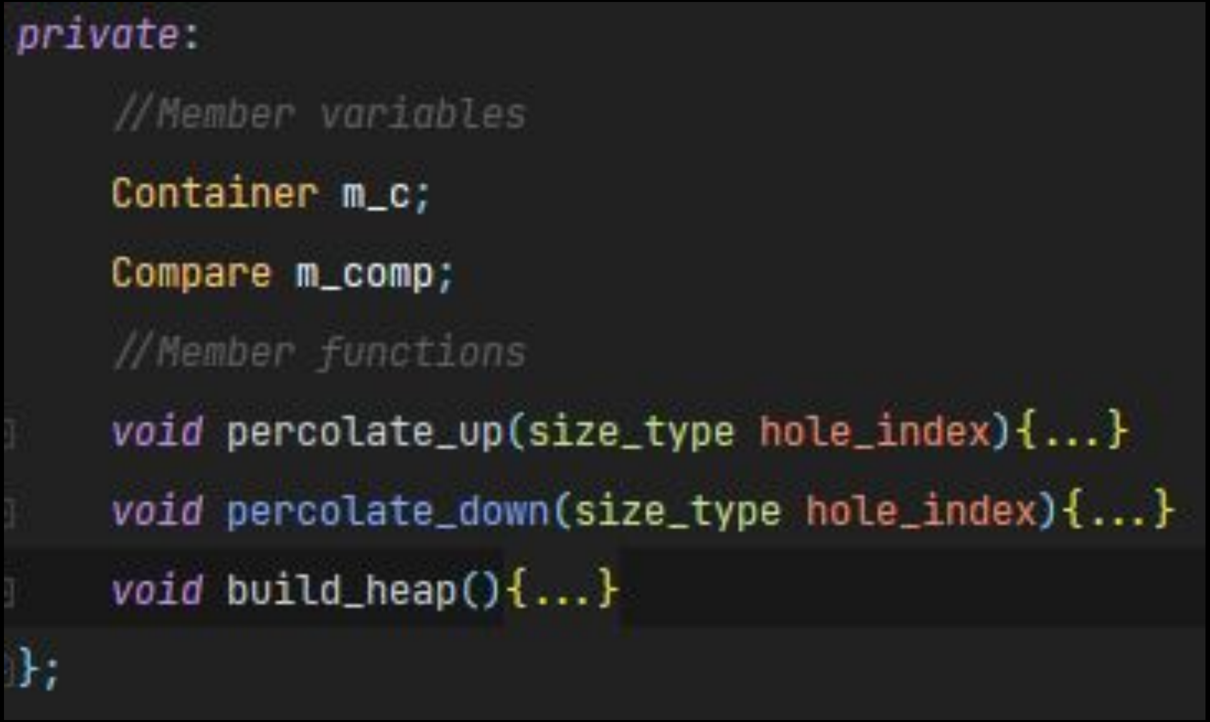
Member functions



```
//friend functions
friend bool operator== (const C_priority_queue<T,Container,Compare>& lhs,
                        const C_priority_queue<T,Container,Compare>& rhs){...} //
friend bool operator!= (const C_priority_queue<T,Container,Compare>& lhs,
                        const C_priority_queue<T,Container,Compare>& rhs){...} //
friend bool operator< (const C_priority_queue<T,Container,Compare>& lhs,
                       const C_priority_queue<T,Container,Compare>& rhs){...} // r
friend bool operator<= (const C_priority_queue<T,Container,Compare>& lhs,
                        const C_priority_queue<T,Container,Compare>& rhs){...} //
friend bool operator> (const C_priority_queue<T,Container ,Compare>& lhs,
                           const C_priority_queue<T,Container,Compare>& rhs){...}
friend bool operator>= (const C_priority_queue<T,Container,Compare>& lhs,
                        const C_priority_queue<T,Container,Compare>& rhs){...} //
```

https://cplusplus.com/reference/queue/priority_queue/

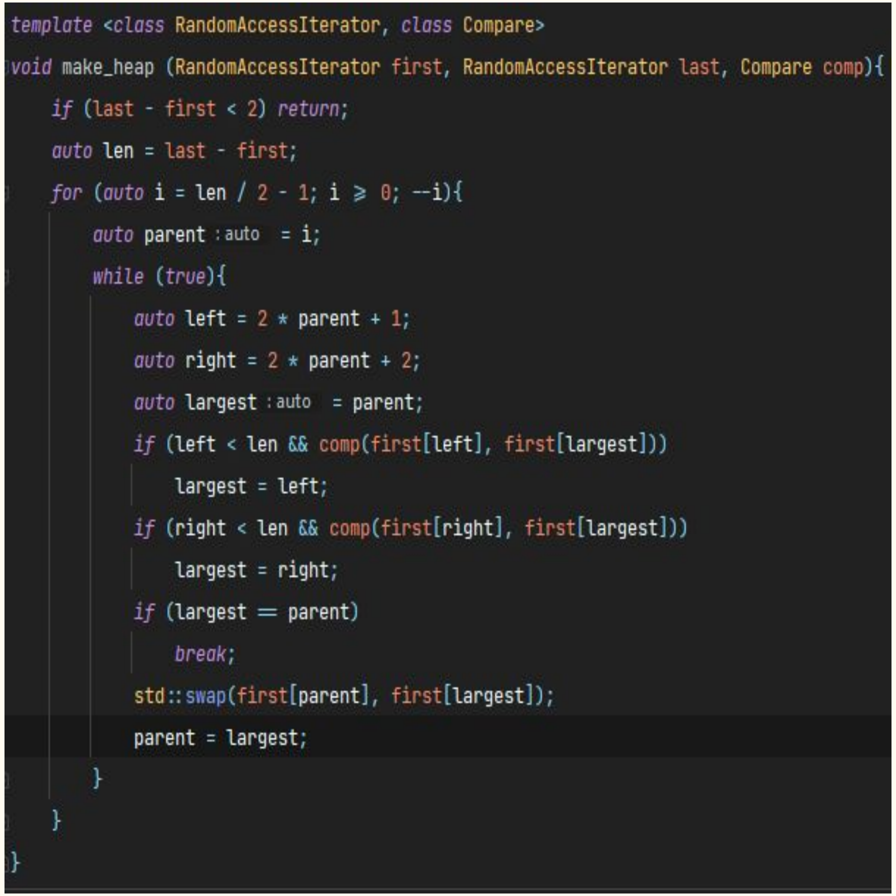
Private



```
private:
    //Member variables
    Container m_c;
    Compare m_comp;
    //Member functions
    void percolate_up(size_type hole_index){...}
    void percolate_down(size_type hole_index){...}
    void build_heap(){...}
};
```

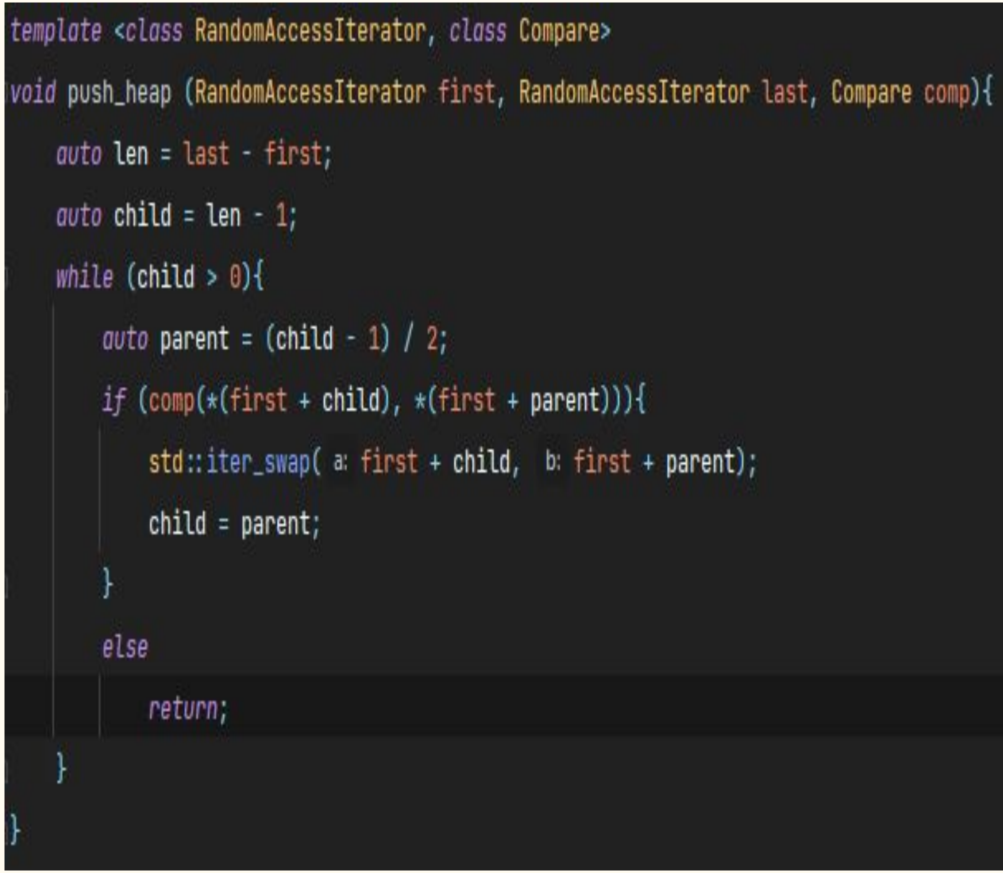
https://cplusplus.com/reference/queue/priority_queue/

make_heap(): O(n)



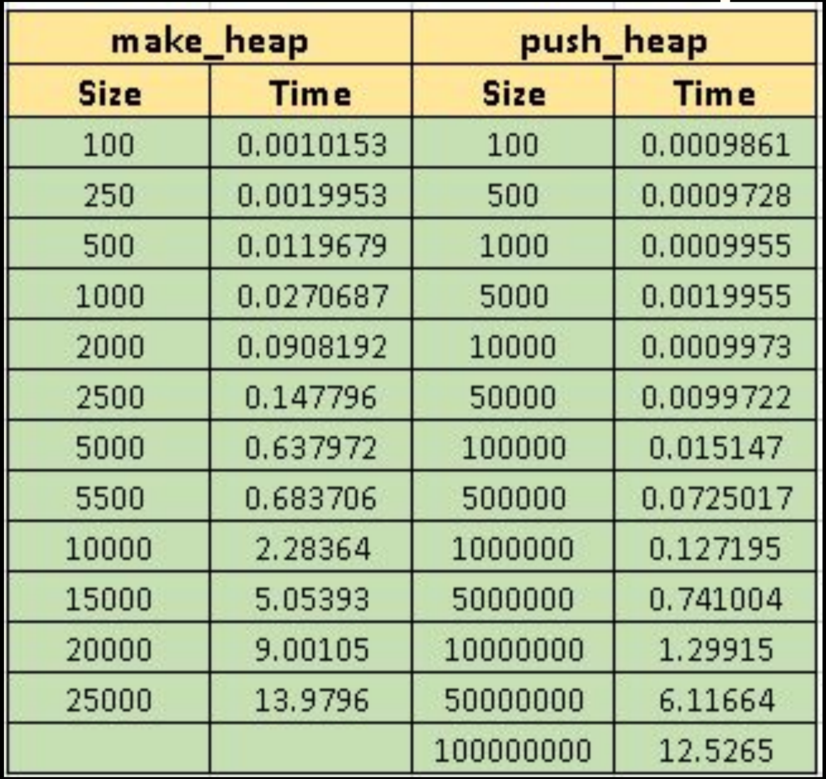
```
template <class RandomAccessIterator, class Compare>
void make_heap (RandomAccessIterator first, RandomAccessIterator last, Compare comp){
    if (last - first < 2) return;
    auto len = last - first;
    for (auto i = len / 2 - 1; i >= 0; --i){
        auto parent :auto  = i;
        while (true){
            auto left = 2 * parent + 1;
            auto right = 2 * parent + 2;
            auto largest :auto  = parent;
            if (left < len && comp(first[left], first[largest]))
                largest = left;
            if (right < len && comp(first[right], first[largest]))
                largest = right;
            if (largest == parent)
                break;
            std::swap(first[parent], first[largest]);
            parent = largest;
        }
    }
}
```

push_heap: O(log n)



```
template <class RandomAccessIterator, class Compare>
void push_heap (RandomAccessIterator first, RandomAccessIterator last, Compare comp){
    auto len = last - first;
    auto child = len - 1;
    while (child > 0){
        auto parent = (child - 1) / 2;
        if (comp(*(first + child), *(first + parent))){
            std::iter_swap( a: first + child,  b: first + parent);
            child = parent;
        }
        else
            return;
    }
}
```

Tabla de tiempos



make_heap		push_heap	
Size	Time	Size	Time
100	0.0010153	100	0.0009861
250	0.0019953	500	0.0009728
500	0.0119679	1000	0.0009955
1000	0.0270687	5000	0.0019955
2000	0.0908192	10000	0.0009973
2500	0.147796	50000	0.0099722
5000	0.637972	100000	0.015147
5500	0.683706	500000	0.0725017
10000	2.28364	1000000	0.127195
15000	5.05393	5000000	0.741004
20000	9.00105	10000000	1.29915
25000	13.9796	50000000	6.11664
		100000000	12.5265

Gráfico priority queue usando make_heap()

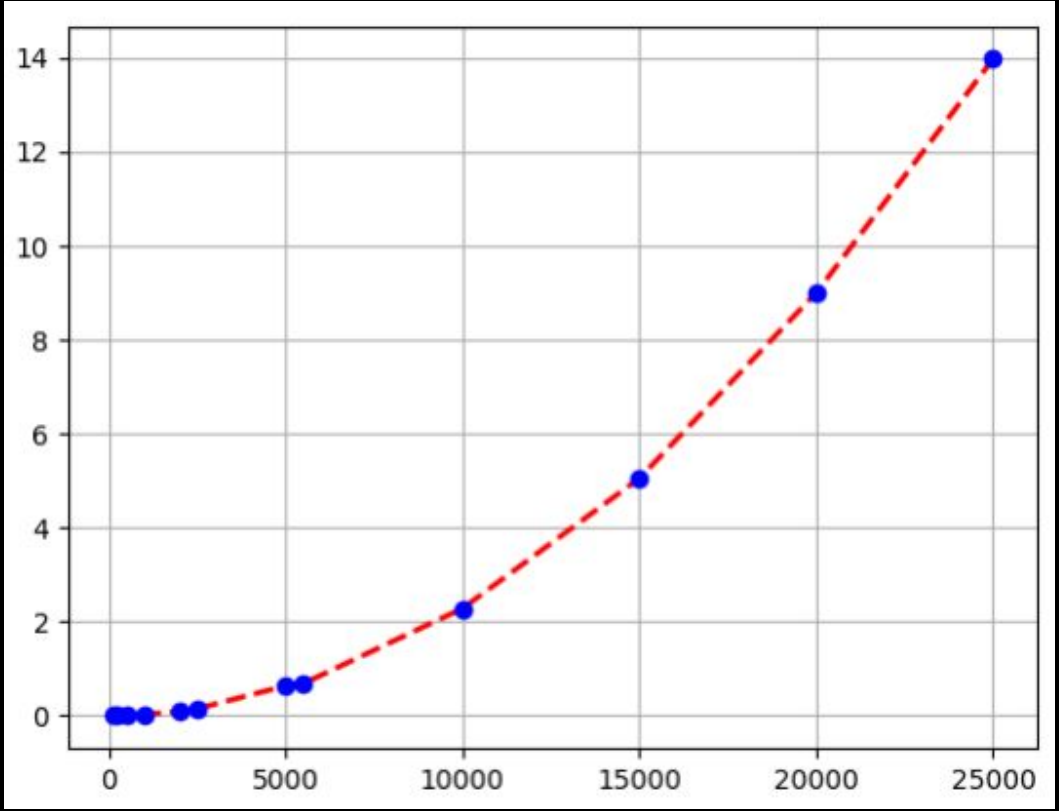
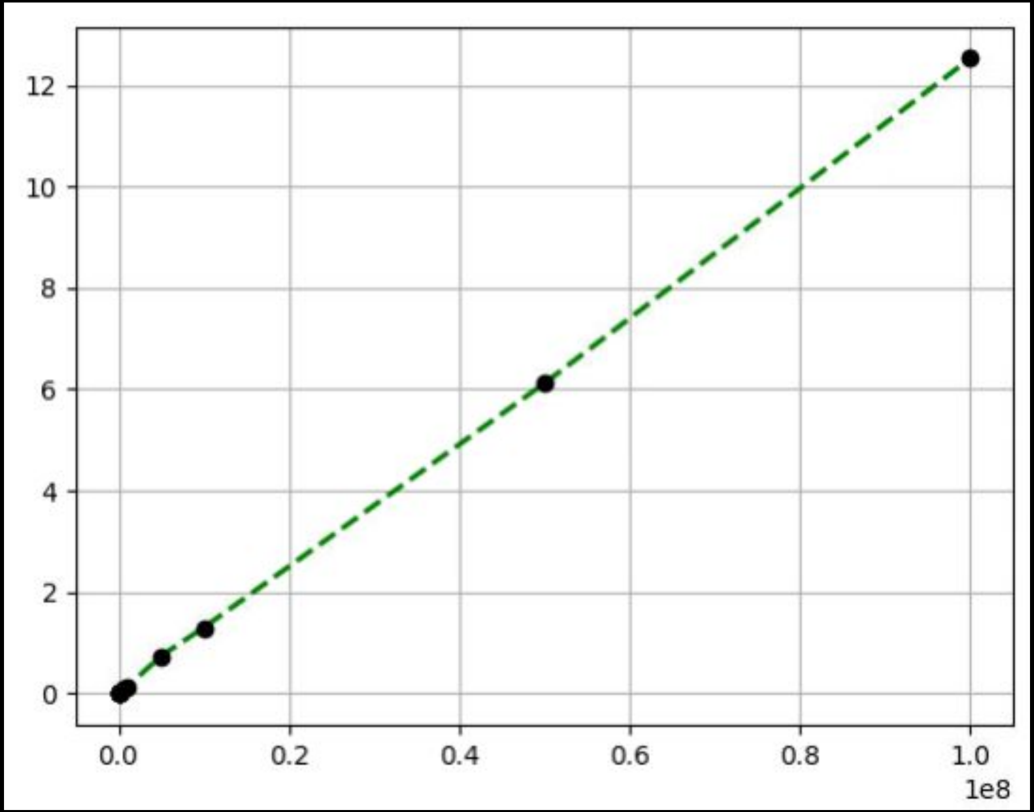


Gráfico priority queue usando push_heap()



GRACIAS