```cpp
/*
 * Instituto Superior de Formación Técnica 151
 * Tecnicatura Superior en Análisis de Sistemas
 *
 * Programación II
 *
 * Queue Project ( without parametrization )
 * Queue Variant : simple
 *
 * Version:  Refactoring 1 (23/8/2015)
 *           Revision 1.1  (24/8/2015)
 *
 * Grupo:
 *
 * Santiago, Matías Gastón
 * Molina Burgos, Álvaro
 * Mato, Santiago
 * Sosa, Luis
 *
 *
 * File:    QueueGeneric.hpp (TDC)
 *
 */

#ifndef QUEUE_GENERIC_HPP
#define QUEUE_GENERIC_HPP

#include "IQueue.hpp"
#include <assert.h>
#include <malloc.h>
#include <string.h>


class Queue : public IQueue
{
    private:

    typedef char byte;

    enum { FIRST = 1, MAX_ITEMS = 256, TOTAL_ITEMS=257, NO_ITEMS=0, SIZE_DEFAULT = 32 };

    size_t size_item;
    int items;

    byte* block;

    int indexToSize(int index)
    {
        return size_item*index;
    };


    public:

    virtual bool add(void* item)
    {
        bool state = !isFull();

        if ( state )
        {
            memcpy( &block[ indexToSize(++items) ], item, size_item );
        };

        return state;
    }



    virtual bool isFull()
    {
        return ( items == MAX_ITEMS )? true : false;
    }

    virtual bool isEmpty()
    {
        return ( items == NO_ITEMS )? true : false;
    }

    virtual void* read()
    {
        assert( !isEmpty() );
        return &block[ indexToSize(FIRST) ];
    }

    virtual void* pop()
    {
        assert( !isEmpty() );
        memcpy( block, &block[indexToSize(FIRST)], size_item*(--items+1) );
        return block;
    }

    //assignment operator
    Queue& operator=( const Queue& queue )
    {
        size_item = queue.size_item;
        items = queue.items;
        memcpy( block, queue.block, queue.size_item*TOTAL_ITEMS );

        return *this;
    };

    //copy constructor
    Queue( const Queue& queue ) :

        size_item(queue.size_item),
        items(queue.items),
        block(
                (byte*)memcpy( (byte*)malloc(size_item*TOTAL_ITEMS),
                queue.block, queue.size_item*TOTAL_ITEMS )
              )

    {};

    //void constructor (with default parameter)
    Queue( size_t bytes = SIZE_DEFAULT ) :

        size_item(bytes),
        items(0),
        block( (byte*)malloc(size_item*TOTAL_ITEMS) )

    {};

    ~Queue()
    {
        free( block );
    };


};


#endif

```