cs225f20-b.sg

Description | Submission view

Grade

Reviewed on Thursday, 10 December 2020, 9:29 AM by Slawomir Grzegorz Wlodkowski
grade: A

Assessment report[-]
[+]**Summary of tests**

Submitted on Thursday, 26 November 2020, 8:27 PM (Download)

allocator.cpp

```
/* Start Header
*******************************************************************/
/*!
\file   allocator.cpp
\author Goh Wei Zhe, weizhe.goh, 440000119
\par    email: weizhe.goh\@digipen.edu
\date   November 26, 2020
\brief  Contains definition of class allocator

Copyright (C) 2020 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/* End Header
*******************************************************************/

#include "allocator.h"
#include <iostream>

unsigned char* cs225::allocator::allocate(size_t count)
{
    size_t size = (count + CHAR_BIT - 1) / CHAR_BIT;
    unsigned char* bitset = new unsigned char[size] {0};

    std::cout << "Allocate " << size << " elements." << std::endl;
    return bitset;
}

void cs225::allocator::deallocate(unsigned char* ptr, size_t count)
{
    size_t size = (count + CHAR_BIT - 1) / CHAR_BIT;
    delete[] ptr;

    std::cout << "Deallocate " << size << " elements." << std::endl;
}
```

allocator.h

```
/* Start Header
*****************************************************************/
/*!
\file   allocator.h
\author Goh Wei Zhe, weizhe.goh, 440000119
\par    email: weizhe.goh\@digipen.edu
\date   November 26, 2020
\brief  Contains declaration of class allocator

Copyright (C) 2020 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/* End Header
*******************************************************************/

#ifndef ALLOCATOR_H
#define ALLOCATOR_H

#include <memory>
#include <climits>

namespace cs225
{
    class allocator
    {
    public:
        static unsigned char* allocate(size_t count);
        static void deallocate(unsigned char* ptr, size_t count);
    };
}

#endif
```

bitset.h

```
/* Start Header
*****************************************************************/
/*!
\file   bitset.h
\author Goh Wei Zhe, weizhe.goh, 440000119
\par    email: weizhe.goh\@digipen.edu
\date   November 26, 2020
\brief  Contains declaration of member functions of class bitset

Copyright (C) 2020 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/* End Header
*******************************************************************/

#ifndef BITSET_H
#define BITSET_H

#include <bitset>

namespace cs225
{
    template<std::size_t N, typename Allocator>
    class bitset
    {
    public:

        unsigned char* _bitset;

        bitset();

        std::size_t count() const;
        std::size_t size() const;

        bitset<N, Allocator>& set(std::size_t pos, bool value = true);
        bitset<N, Allocator>& reset(std::size_t pos);
        bitset<N, Allocator>& flip(std::size_t pos);

        bool test(std::size_t pos) const;
        bool operator[](std::size_t pos) const;

        std::string to_string(char c1 = '0' , char c2 = '1') const;

        ~bitset();
    };
}

#include "bitset.hpp"

#endif
```

bitset.hpp

```cpp
/* Start Header
*******************************************************************/
/*!
\file   bitset.hpp
\author Goh Wei Zhe, weizhe.goh, 440000119
\par    email: weizhe.goh\@digipen.edu
\date   November 26, 2020
\brief  Contains definition of member functions of class bitset

Copyright (C) 2020 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/* End Header
*******************************************************************/

#ifndef BITSET_HPP
#define BITSET_HPP

#include "bitset.h"

template<std::size_t N, typename Allocator>
cs225::bitset<N, Allocator>::bitset()
{
    _bitset = cs225::allocator::allocate(N);
}

template<std::size_t N, typename Allocator>
cs225::bitset<N, Allocator>& cs225::bitset<N, Allocator>
::set(size_t pos, bool value)
{
    if (value)
        _bitset[pos / CHAR_BIT] |=
        static_cast<unsigned char>(1U << pos % CHAR_BIT);
    else
        _bitset[pos / CHAR_BIT] &=
        static_cast<unsigned char>(~(1U << pos % CHAR_BIT));

    return *this;
}

template<std::size_t N, typename Allocator>
cs225::bitset<N, Allocator>& cs225::bitset<N, Allocator>::reset(size_t pos)
{
    _bitset[pos / CHAR_BIT] &=
        static_cast<unsigned char>(~(1U << pos % CHAR_BIT));

    return *this;
}

template<std::size_t N, typename Allocator>
cs225::bitset<N, Allocator>& cs225::bitset<N, Allocator>::flip(size_t pos)
{
    _bitset[pos / CHAR_BIT] ^=
        static_cast<unsigned char>(1U << (pos % CHAR_BIT));

    return *this;
}

template<std::size_t N, typename Allocator>
size_t cs225::bitset<N, Allocator>::count() const
{
    size_t count = 0;

    for (size_t i = N; i > 0;)
        (bitset<N, Allocator>::test(--i) ? ++count : count);

    return count;
}

template<std::size_t N, typename Allocator>
std::string cs225::bitset<N, Allocator>::to_string(char c1, char c2) const
{
    std::string s;

    char zero = static_cast<unsigned char>(c1);
    char one = static_cast<unsigned char>(c2);

    for (size_t i = N; i > 0;)
        s.push_back(bitset<N, Allocator>::test(--i) ? one : zero);

    return s;
}

template<std::size_t N, typename Allocator>
size_t cs225::bitset<N, Allocator>::size() const
{
    return N;
}

template<std::size_t N, typename Allocator>
bool cs225::bitset<N, Allocator>::test(size_t pos) const
{
    if (N <= pos)
        throw std::out_of_range("Out of range");

    return (_bitset[pos / CHAR_BIT] & (1U << pos % CHAR_BIT)) != 0;
}

template<std::size_t N, typename Allocator>
bool cs225::bitset<N, Allocator>::operator[](size_t pos) const
{
    return (_bitset[pos / CHAR_BIT] & (1U << pos % CHAR_BIT)) != 0;
}

template<std::size_t N, typename Allocator>
cs225::bitset<N, Allocator>::~bitset()
```

```
{
    cs225::allocator::deallocate(_bitset, N);
}

```

bitset_tep.h

```
/* Start Header
*****************************************************************/
/*!
\file   bitset_tep.h
\author Goh Wei Zhe, weizhe.goh, 440000119
\par    email: weizhe.goh\@digipen.edu
\date   November 26, 2020
\brief  Contains declaration of class bitset_tep

Copyright (C) 2020 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/* End Header
*******************************************************************/

#ifndef BITSET_TEP_H
#define BITSET_TEP_H

#include <string>
#include <iostream>
#include <memory>

namespace cs225
{
    class bitset_tep
    {
        struct IConcept
        {
            virtual size_t size() const = 0;
            virtual size_t count() const = 0;

            virtual void set(size_t, bool = true) = 0;
            virtual void flip(size_t) = 0;
            virtual void reset(size_t) = 0;

            virtual bool test(size_t) const = 0;
            virtual bool operator[](std::size_t pos) const = 0;

            virtual std::string to_string(char = '0', char = '1') const = 0;
            virtual ~IConcept() = default;
        };

        template<typename T>
        class Model : public IConcept
        {
            T _instance;

        public:

            template<typename... Args>
            Model(Args&&... args);

            virtual size_t size() const override;
            virtual size_t count() const override;

            virtual void set(size_t pos, bool value = true) override;
            virtual void flip(size_t pos) override;
            virtual void reset(size_t pos) override;

            virtual bool test(size_t pos) const override;
            virtual bool operator[](std::size_t pos) const override;

            virtual std::string to_string(char = '0', char = '1')const override;
        };

        std::unique_ptr<IConcept> _concept;
        bitset_tep(std::unique_ptr<IConcept> concept);

    public:
        std::size_t count() const;
        std::size_t size() const;

        void set(size_t pos, bool value = true);
        void flip(size_t pos) const;
        void reset(size_t pos) const;

        bool test(size_t pos) const;
        bool operator[](std::size_t pos) const;

        std::string to_string(char c1 = '0', char c2 = '1') const;

        template<typename T, typename... Args>
        static bitset_tep create(Args&&... args);
    };
}

#include "bitset_tep.hpp"

#endif // !_BITSET_TEP_H

```

bitset_tep.hpp

```
/* Start Header
*****************************************************************/
/*!
\file   bitset_tep.hpp
\author Goh Wei Zhe, weizhe.goh, 440000119
\par    email: weizhe.goh\@digipen.edu
\date   November 26, 2020
\brief  Contains definition of templated alias of class bitset_tep

Copyright (C) 2020 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/* End Header
*******************************************************************/
#include <iostream>

#ifndef BITSET_TEP_HPP
#define BITSET_TEP_HPP

#include "bitset_tep.h"
#include <iostream>

template<typename T> template <typename... Args>
cs225::bitset_tep::Model<T>::Model(Args&&... args)
: _instance{std::forward<Args>(args)...}{}

template<typename T>
size_t cs225::bitset_tep::Model<T>::size() const
{
   return _instance.size();
}

template<typename T>
size_t cs225::bitset_tep::Model<T>::count() const
{
    return _instance.count();
}

template<typename T>
void cs225::bitset_tep::Model<T>::set(size_t pos, bool value)
{
    _instance.set(pos, value);
}

template<typename T>
void cs225::bitset_tep::Model<T>::flip(size_t pos)
{
    _instance.flip(pos);
}

template<typename T>
void cs225::bitset_tep::Model<T>::reset(size_t pos)
{
    _instance.reset(pos);
}

template<typename T>
bool cs225::bitset_tep::Model<T>::test(size_t pos) const
{
    return _instance.test(pos);
}

template<typename T>
std::string cs225::bitset_tep::Model<T>::to_string(char c1, char c2) const
{
    return _instance.to_string(c1, c2);
}

template<typename T>
bool cs225::bitset_tep::Model<T>::operator[](size_t pos) const
{
    return _instance.operator[](pos);
}

template<typename T, typename... Args>
cs225::bitset_tep cs225::bitset_tep::create(Args&&... args)
{
    return bitset_tep(std::make_unique<Model<T, Args...>>
                    (std::forward<Args>(args)...));
}

#endif // !BITSET_TEP_HPP

```

bitset_tep.cpp

```
/* Start Header
*****************************************************************/
/*!
\file   bitset_tep.cpp
\author Goh Wei Zhe, weizhe.goh, 440000119
\par    email: weizhe.goh\@digipen.edu
\date   November 26, 2020
\brief  Contains definition of non-templated alias of class bitset_tep

Copyright (C) 2020 DigiPen Institute of Technology.
Reproduction or disclosure of this file or its contents without the
prior written consent of DigiPen Institute of Technology is prohibited.
*/
/* End Header
*******************************************************************/
#include <iostream>

#include "bitset_tep.h"

cs225::bitset_tep::bitset_tep(std::unique_ptr<cs225::bitset_tep::IConcept>concept)
    : _concept{std::move(concept) } {}

size_t cs225::bitset_tep::count() const
{
    return _concept->count();
}

size_t cs225::bitset_tep::size() const
{
    return _concept->size();
}

void cs225::bitset_tep::set(size_t pos, bool value)
{
    _concept->set(pos, value);
}

void cs225::bitset_tep::flip(size_t pos) const
{
    _concept->flip(pos);
}

void cs225::bitset_tep::reset(size_t pos) const
{
    _concept->reset(pos);
}

bool cs225::bitset_tep::test(size_t pos) const
{
    return _concept->test(pos);
}

bool cs225::bitset_tep::operator[](size_t pos) const
{
    return _concept->operator[](pos);
}

std::string cs225::bitset_tep::to_string(char c1, char c2) const
{
    return _concept->to_string(c1, c2);
}
```

VPL

◄ Assignment 6 files

Jump to...

Attendance cs225f20-b.sg Tuesday 24/11/2020 1:30pm-3:10pm ►