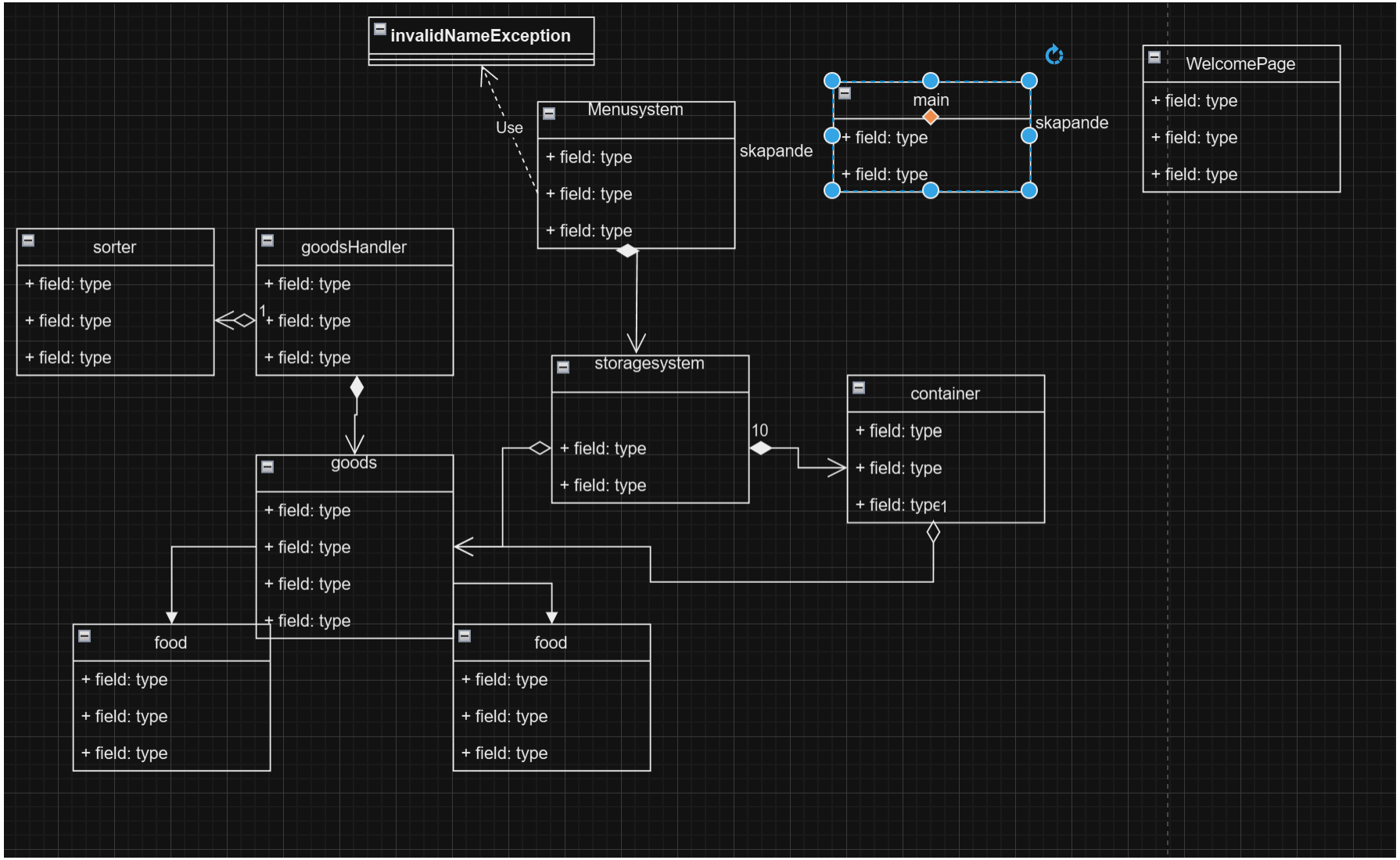
C++ Projekt

Av Liam och Måns.


invalidNameException
Use
Menusystem
+ field: type
+ field: type
+ field: type
skapande
main
+ field: type
+ field: type
skapande
WelcomePage
+ field: type
+ field: type
+ field: type
sorter
+ field: type
+ field: type
+ field: type
goodsHandler
+ field: type
1
+ field: type
+ field: type
storagesystem
+ field: type
+ field: type
10
container
+ field: type
+ field: type
+ field: type
1
goods
+ field: type
+ field: type
+ field: type
+ field: type
food
+ field: type
+ field: type
+ field: type
food
+ field: type
+ field: type
+ field: type

L

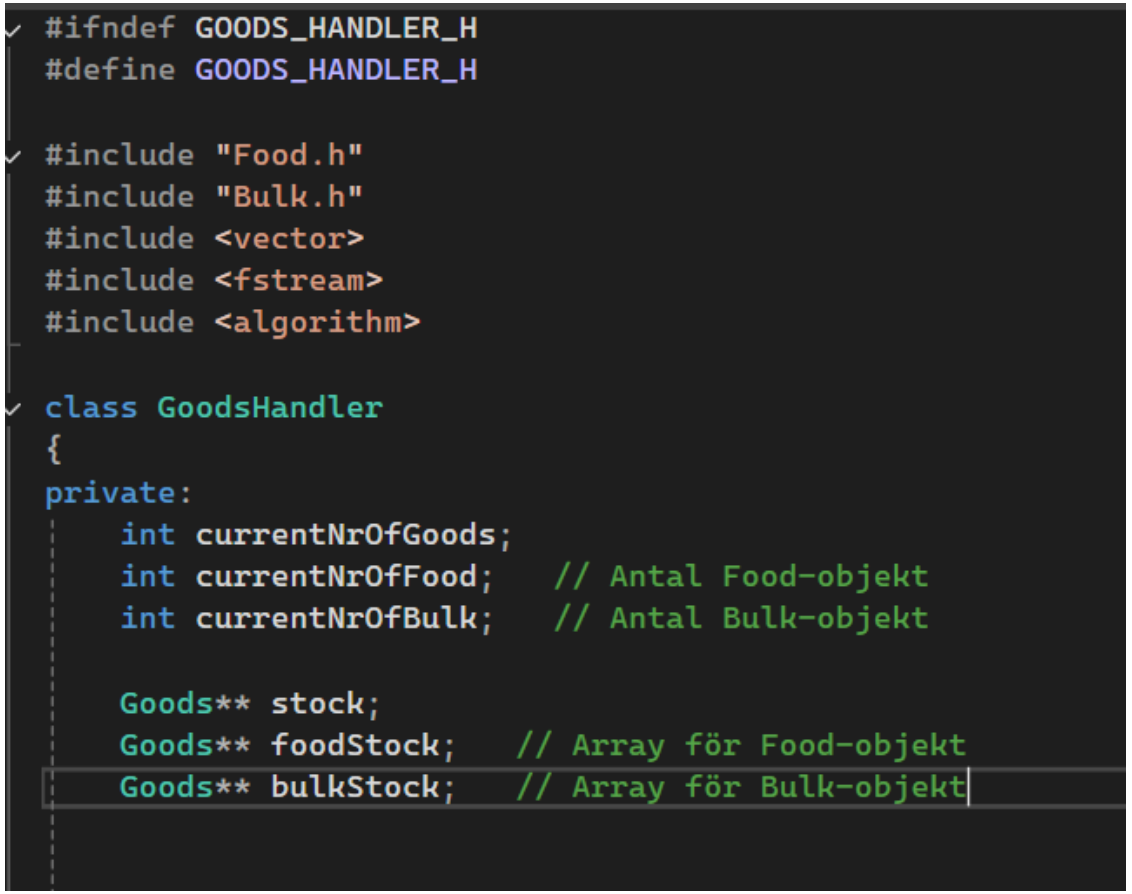
https://www.youtube.com/watch?v=zCuXT2RG1ug

1/7 -M

- Ett visst antal, minst 5, egendefinierade **relevanta** klasser, vilka på förhand ska vara **godkända av lärare**

2/7 -M

- **Flera objekt** av någon av de egendefinierade klasserna ska finnas. Dessa ska hanteras i en behållare (array, vector, ...).



```
#ifndef GOODS_HANDLER_H
#define GOODS_HANDLER_H

#include "Food.h"
#include "Bulk.h"
#include <vector>
#include <fstream>
#include <algorithm>

class GoodsHandler
{
private:
    int currentNrOfGoods;
    int currentNrOfFood;   // Antal Food-objekt
    int currentNrOfBulk;   // Antal Bulk-objekt

    Goods** stock;
    Goods** foodStock;   // Array för Food-objekt
    Goods** bulkStock;   // Array för Bulk-objekt
```

```h
#ifndef GOODS_H
#define GOODS_H

#include <iostream>
#include <string>

class Goods
{
private:
    float weight;
    std::string name;
    std::string type;

public:
    Goods(float weight = 0.0f, const std::string& name = "Unknown");
    virtual ~Goods();

    //GET
    std::string getName() const;
    float getWeight() const;
    //SET
    void setWeight(float weight);
    void setName(std::string name);

    virtual std::string toString() const = 0;
};
#endif // !GOODS_H
```

```h
#ifndef BULK_H
#define BULK_H
#include "Goods.h"

class Bulk : public Goods
{
private:
    float volume;

public:
    Bulk(float volume = 0.0f, float wei
    ~Bulk();
    void setVolume(float volume);
    float getVolume() const;
    std::string toString() const overri
};
#endif // !BULK_H
```

```h
#ifndef FOOD_H
#define FOOD_H
#include "Goods.h"

class Food : public Goods
{
private:
    int quantity;

public:
    Food(int quantity = 0, float weight = 0.0f, const std::string& name = "Unknown")
    ~Food(); // Destruktor
    void setQuantity(int quantity);  // Setter för quantity
    int getQuantity() const;  // Getter för quantity
    std::string toString() const override;
};
#endif // !FOOD_H
```

3/7 -M

- Egendefinierat **logiskt och rimligt arv** med **abstrakt klass/er**

4/7 -L

- **Överskuggning och dynamisk bindning** på ett lämpligt och relevant sätt

```
Container(double maxWeight);
Container(double maxWeight, double maxVolume);
```

```
#ifndef GOODS_H
#define GOODS_H

#include <iostream>
#include <string>

class Goods
{
private:
    float weight;
    std::string name;
    std::string type;

public:
    Goods(float weight = 0.0f, const std::string& name = "Unknown");
    virtual ~Goods();

    //GET
    std::string getName() const;
    float getWeight() const;
    //SET
    void setWeight(float weight);
    void setName(std::string name);

    virtual std::string toString() const = 0;
};
#endif // !GOODS_H
```

```
void GoodsHandler::showFood() const
{
    std::cout << "\nShowing food items:\n";
    for (int i = 0; i < this->currentNrOfFood; i++) {
        if (this->foodStock[i] != nullptr) {
            std::cout << this->foodStock[i]->toString() << std::endl;
        }
    }
}

void GoodsHandler::showBulk() const
{
    std::cout << "\nShowing bulk items:\n";
    for (int i = 0; i < this->currentNrOfBulk; i++) {
        if (this->bulkStock[i] != nullptr) {
            std::cout << this->bulkStock[i]->toString() << std::endl;
        }
    }
}
```

5/7 –L

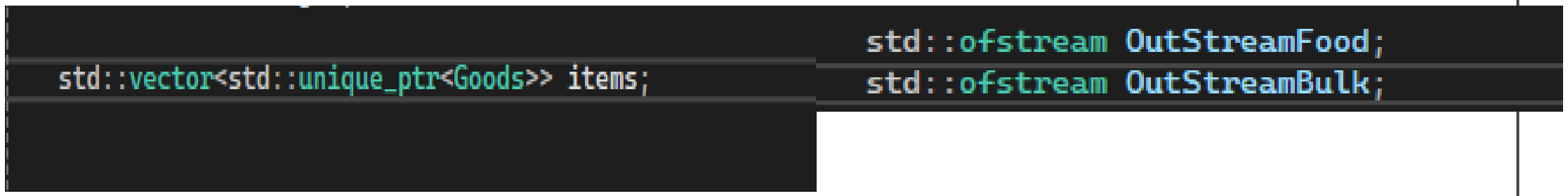
- Rimlig användning **av dynamisk minneshantering** (genom användande av new och delete)

```
InStream.open(filename);
if (InStream.is_open()) {

    if (filename == "StoredFood.txt") {
        while (InStream >> quantity >> weight >> name) {
            if (this->capacity == currentNrOfGoods)
            {
                this->expandStock();
            }
            this->foodStock[this->currentNrOfFood] = new Food(quantity, weight, name);
            this->stock[this->currentNrOfGoods] = this->foodStock[this->currentNrOfFood];
            this->currentNrOfFood++;
            this->currentNrOfGoods++;
        }
    }
    else if (filename == "StoredBulk.txt") {
        while (InStream >> volume >> weight >> name) {
            if (this->capacity == currentNrOfGoods)
            {
                this->expandStock();
            }
            this->bulkStock[this->currentNrOfBulk] = new Bulk(volume, weight, name);
            this->stock[this->currentNrOfGoods] = this->bulkStock[this->currentNrOfBulk];
            this->currentNrOfBulk++;
            this->currentNrOfGoods++;
        }
    }
}
InStream.close();
```

```
//----->Destructor<-----

GoodsHandler::~GoodsHandler() {

    delete[] foodStock;
    delete[] bulkStock;
    delete[] stock;


    std::cout << "\nstock ptr deleted" << std::endl;
}
```

```
    delete[] stock;
    this->stock = newStock;
    delete[] foodStock;
    this->foodStock = newFoodStock;
    delete[] bulkStock;
    this->bulkStock = newBulkStock;
    std::cout << "Expand function called, current capacity = " << this->capacity << std::endl;
```

```
#include "GoodsHandler.h"

GoodsHandler::GoodsHandler(int capacity) : stock(new Goods* [capacity] {nullptr}), currentNrOfGoods(0),
currentNrOfFood(0), currentNrOfBulk(0), foodStock(new Goods* [capacity] {nullptr}), bulkStock(new Goods* [capacity] {nullptr}), capacity(capacity)
{
}
```



6/7 -L

- Någon **datastruktur från standardbiblioteket** (vector, stack, kö, …)

7/7 -M

- Läsning från och/eller skrivning till **textfiler**

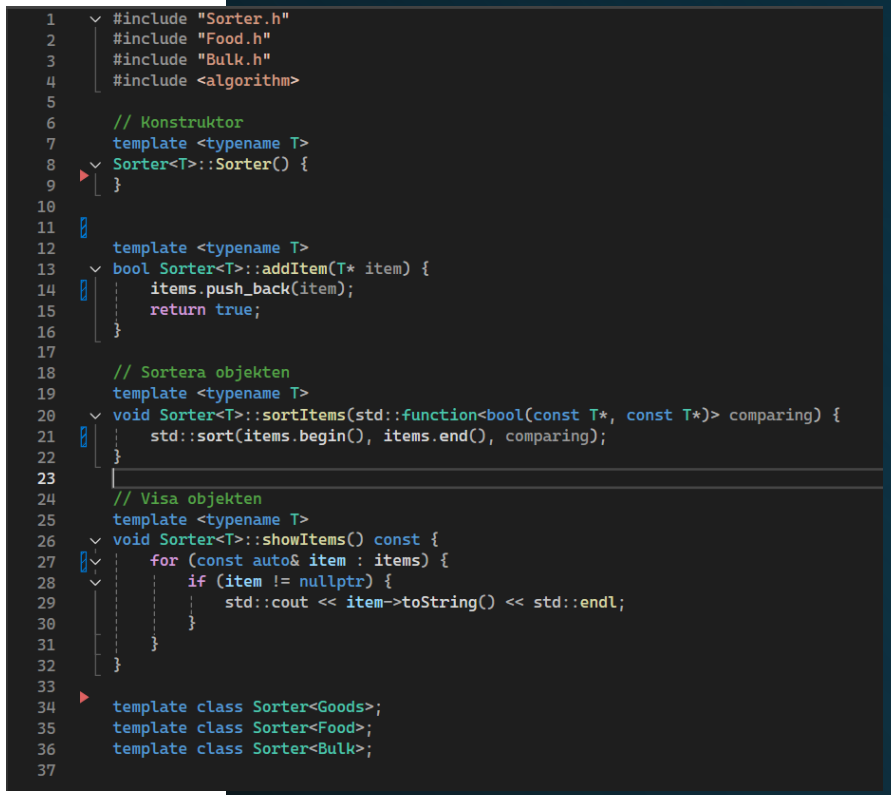
```cpp
//-----> Output into text file <-----

void GoodsHandler::addToFile()
{
    std::ofstream OutStreamFood;
    std::ofstream OutStreamBulk;
    OutStreamBulk.open("StoredBulk.txt");
    OutStreamFood.open("StoredFood.txt");
    if (OutStreamFood.is_open() || OutStreamBulk.is_open())
    {
        for (int i = 0; i < this->currentNrOfGoods; i++)
        {
            Food* fPtr = dynamic_cast<Food*>(this->stock[i]);
            if (fPtr != nullptr)
            {
                OutStreamFood << std::to_string(fPtr->getQuantity()) + "\n";
                OutStreamFood << std::to_string(fPtr->getWeight()) + "\n";
                OutStreamFood << fPtr->getName() + "\n";
            }
            else {
                Bulk* bPtr = dynamic_cast<Bulk*>(this->stock[i]);
                if (bPtr != nullptr)
                {
                    OutStreamBulk << std::to_string(bPtr->getVolume()) + "\n";
                    OutStreamBulk << std::to_string(bPtr->getWeight()) + "\n";
                    OutStreamBulk << bPtr->getName() + "\n";
                }
            }
        }
    }
    OutStreamFood.close();
    OutStreamBulk.close();
}
```

```cpp
void GoodsHandler::readFromFile(const std::string& filename)
{
    int quantity;
    float weight;
    std::string name;
    float volume;
    std::ifstream InStream;

    InStream.open(filename);
    if (InStream.is_open()) {

        if (filename == "StoredFood.txt") {
            while (InStream >> quantity >> weight >> name) {
                if (this->capacity == currentNrOfGoods)
                {
                    this->expandStock();
                }
                this->foodStock[this->currentNrOfFood] = new Food(quantity, weight, name);
                this->stock[this->currentNrOfGoods] = this->foodStock[this->currentNrOfFood];
                this->currentNrOfFood++;
                this->currentNrOfGoods++;
            }
        }
        else if (filename == "StoredBulk.txt") {
            while (InStream >> volume >> weight >> name) {
                if (this->capacity == currentNrOfGoods)
                {
                    this->expandStock();
                }
                this->bulkStock[this->currentNrOfBulk] = new Bulk(volume, weight, name);
                this->stock[this->currentNrOfGoods] = this->bulkStock[this->currentNrOfBulk];
                this->currentNrOfBulk++;
                this->currentNrOfGoods++;
            }
        }
    }
    InStream.close();
}
```


.sf-sub-indicator
.cart-menu .cart-icon-wr
er-outer.transparent header#to
.sf-menu > li.current_page_
.sf-menu > li.current-menu-
> ul > li > a:hover > .sf-su
ul #search-btn a:hover span,
.sf-menu > li.current-menu-i
hover .icon-salient-cart,.ascend
!important;color:#ffffff!impo
ent header#top nav>ul>li.but
-widget-area-toggle a i
-outer.transparent

1/6 –L **

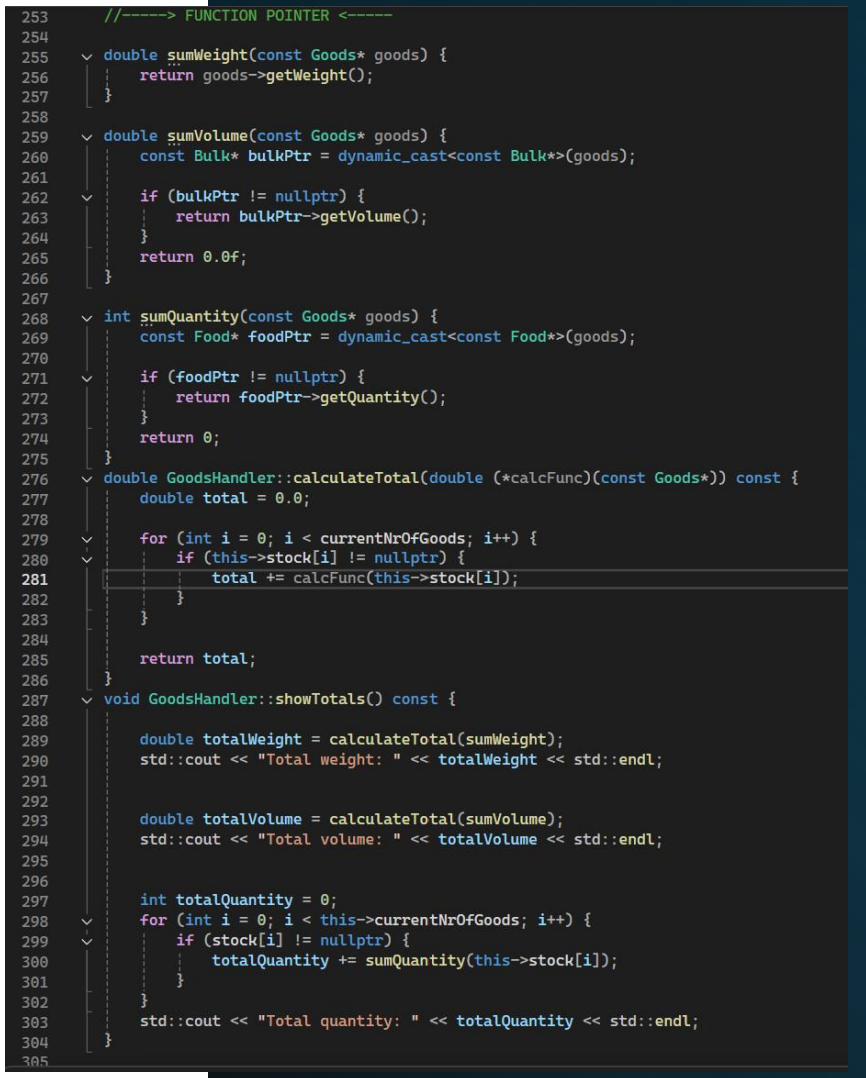
- Egendefinerad rimlig generisk klass som används på ett relevant sätt: 2 poäng



```cpp
#include "Sorter.h"
#include "Food.h"
#include "Bulk.h"
#include <algorithm>

// Konstruktor
template <typename T>
Sorter<T>::Sorter() {
}


template <typename T>
bool Sorter<T>::addItem(T* item) {
    items.push_back(item);
    return true;
}

// Sortera objekten
template <typename T>
void Sorter<T>::sortItems(std::function<bool(const T*, const T*)> comparing) {
    std::sort(items.begin(), items.end(), comparing);
}

// Visa objekten
template <typename T>
void Sorter<T>::showItems() const {
    for (const auto& item : items) {
        if (item != nullptr) {
            std::cout << item->toString() << std::endl;
        }
    }
}

template class Sorter<Goods>;
template class Sorter<Food>;
template class Sorter<Bulk>;
```

2/6 -L

- Relevant användande av funktionspekare: 1 poäng



```
253     //-----> FUNCTION POINTER <-----
254
255     double sumWeight(const Goods* goods) {
256         return goods->getWeight();
257     }
258
259     double sumVolume(const Goods* goods) {
260         const Bulk* bulkPtr = dynamic_cast<const Bulk*>(goods);
261
262         if (bulkPtr != nullptr) {
263             return bulkPtr->getVolume();
264         }
265         return 0.0f;
266     }
267
268     int sumQuantity(const Goods* goods) {
269         const Food* foodPtr = dynamic_cast<const Food*>(goods);
270
271         if (foodPtr != nullptr) {
272             return foodPtr->getQuantity();
273         }
274         return 0;
275     }
276     double GoodsHandler::calculateTotal(double (*calcFunc)(const Goods*)) const {
277         double total = 0.0;
278
279         for (int i = 0; i < currentNrOfGoods; i++) {
280             if (this->stock[i] != nullptr) {
281                 total += calcFunc(this->stock[i]);
282             }
283         }
284
285         return total;
286     }
287     void GoodsHandler::showTotals() const {
288
289         double totalWeight = calculateTotal(sumWeight);
290         std::cout << "Total weight: " << totalWeight << std::endl;
291
292
293         double totalVolume = calculateTotal(sumVolume);
294         std::cout << "Total volume: " << totalVolume << std::endl;
295
296
297         int totalQuantity = 0;
298         for (int i = 0; i < this->currentNrOfGoods; i++) {
299             if (stock[i] != nullptr) {
300                 totalQuantity += sumQuantity(this->stock[i]);
301             }
302         }
303         std::cout << "Total quantity: " << totalQuantity << std::endl;
304     }
305
```

3/6 -M

- Relevant användande av Lamdautryck : 1 poäng

```
//----->SHOW_SORTED_GOODS<-----

void GoodsHandler::showAll(char sortChoice)
{
    Sorter<Goods> sorter;
    for (int i = 0; i < this->currentNrOfGoods; ++i) {
        sorter.addItem(this->stock[i]);
    }
    if (sortChoice == '1') {
        sorter.sortItems([](const Goods* a, const Goods* b) {
            return a->getWeight() > b->getWeight();
            });
    }
    else if (sortChoice == '2') {
        sorter.sortItems([](const Goods* a, const Goods* b) {
            return a->getName() < b->getName();
            });
    }
    else {
        std::cout << "Invalid choice. Defaulting to sorting by Weight.\n";
        sorter.sortItems([](const Goods* a, const Goods* b) {
            return a->getWeight() > b->getWeight();
            });
    }
    sorter.showItems();
}
```

4/6 -M

- Användande av undantagshantering där detta är lämpligt : 1 poäng

```cpp
#ifndef INVALID_NAME_EXCEPTION_H
#define INVALID_NAME_EXCEPTION_H

#include <exception>
#include <string>
#include <cctype>  // För isalpha() och isdigit()

class InvalidNameException : public std::exception {
public:
    const char* what() const  {
        return "Invalid input! The input cannot be empty or
    }

    static bool isValidName(const std::string& name) {
        for (int i = 0; i < name.length(); ++i) {
            if (!std::isalpha(name[i]) && name[i] != ' ') {
                return false;
            }
        }
        return true;
    }

    static bool isValidNumber(const std::string& number) {
        bool decimalPointFound = false;
        for (int i = 0; i < number.length(); ++i) {
            if (number[i] == '.' && !decimalPointFound) {
                decimalPointFound = true;
            }
            else if (!std::isdigit(number[i])) {
                return false;
            }
        }
        return !number.empty();  // Returnera false för tomm
    }
};

#endif // INVALID_NAME_EXCEPTION_H
```

```cpp
void MenuSystem::addBulkToContainer() {
    try {
        std::string name, weightStr, volumeStr;

        std::cout << "Enter Bulk name:" << std::endl;
        std::cout << ">> ";
        std::cin >> name;

        if (name.empty() || !InvalidNameException::isValidName(name)) {
            throw InvalidNameException();
        }

        std::cout << "\nEnter Bulk weight (must be a positive number):" << std::endl;
        std::cout << ">> ";
        std::cin >> weightStr;

        if (!InvalidNameException::isValidNumber(weightStr)) {
            throw  InvalidNameException();;
        }
        float weight = std::stof(weightStr);

        std::cout << "\nEnter Bulk volume (must be a positive number):" << std::endl;
        std::cout << ">> ";
        std::cin >> volumeStr;

        if (!InvalidNameException::isValidNumber(volumeStr)) {
            throw  InvalidNameException();;
        }
        float volume = std::stof(volumeStr);
```

```cpp
    }
    catch (const InvalidNameException& e) {
        std::cout << "Error: " << e.what() << std::endl;
    }
    catch (const std::exception& e) {
        std::cout << "General error: " << e.what() << std::endl;
    }
}
```

```
std::unique_ptr<Container> containers[MAX_CONTAINERS];
```

```
GoodsHandler::~GoodsHandler() {

    delete[] foodStock;
    delete[] bulkStock;
    delete[] stock;


    std::cout << "\nstock ptr deleted" << std::endl;
}
```

5/6 -M

- Användande av smarta pekare där detta är lämpligt: 1 poäng
    - Tar Bort alla container objekt

6/6 L

- Grafiskt användargränssnitt (ex-vis SFML) : 4 poäng

```cpp
#include "WelcomePage.h"
#include <SFML/Window.hpp>
#include <SFML/System.hpp>
#include <cstdlib>

WelcomePage::WelcomePage() {

    windowWidth = 1500.f;
    windowHeight = 1300.f;

    rect.setSize(sf::Vector2f(200.f, 200.f));
    rect.setPosition(windowWidth / 2.f, windowHeight / 2.f);
    rect.setFillColor(sf::Color::Blue);

    speedX = 700.f;
    speedY = 700.f;


    if (!collisionSoundBuffer.loadFromFile("collision.wav")) {

    }
    collisionSound.setBuffer(collisionSoundBuffer);
}

void WelcomePage::moveRectangle(float deltaTime) {
    rect.move(speedX * deltaTime, speedY * deltaTime);



    if (rect.getPosition().x <= 0.f || rect.getPosition().x + rect.getSize().x >= windowWidth) {
        speedX = -speedX;
        changeColor();
        collisionSound.play();
    }

    if (rect.getPosition().y <= 0.f || rect.getPosition().y + rect.getSize().y >= windowHeight) {
        speedY = -speedY;
        changeColor();
        collisionSound.play();
    }


}
```

```cpp
void WelcomePage::changeColor() {
    currentColor = sf::Color(rand() % 256, rand() % 256, rand() % 256);
    rect.setFillColor(currentColor);
}

// START ANIMATION
void WelcomePage::start() {

    sf::RenderWindow window(sf::VideoMode(windowWidth, windowHeight), "Välkomstsida med Animation");

    sf::Clock clock;

    while (window.isOpen()) {
        sf::Event event;
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed)
                window.close();
        }

        float deltaTime = clock.restart().asSeconds();

        moveRectangle(deltaTime);

        window.clear();
        window.draw(rect);
        window.display();
    }
}
```


Frågor?