מבוא לתכנות מונחה עצמים

סטודנטית 1: דליה וויליאם

סטודנט 2: גיא רחמים

Topping.h

//Dalya William & Guy Rahamim

#pragma once

#include<iostream>

#include <cassert>

class Topping

{

private:

//member variables

char\* name;

float price;

char coverage;

public:

//constructors

Topping();

Topping(const char\* name, char coverage, float price = 10 );

Topping(const Topping& other);

//destructor

~Topping();

//Getters

inline char\* getName() const;

inline float getPrice() const;

inline char getCoverage() const;

//Setters

void setName(const char\* name);

void setPrice(float price);

void setCoverage(const char coverage);

//Operator overloaders

void operator=(const Topping& other);

friend std::ostream& operator<<(std::ostream& os, const Topping& other);

friend bool operator>(const Topping& one, const Topping& other);

friend bool operator<(const Topping& one, const Topping& other);

friend bool operator==(const Topping& one, const Topping& other);

};

Topping.cpp

//Dalya William & Guy Rahamim

#include "Topping.h"

Topping::Topping()

{

this->name = NULL;

setCoverage('f');

setPrice(0);

}

Topping::Topping(const char\* name, char coverage, float price)

{

this->name = NULL;

setName(name);

setPrice(price);

setCoverage(coverage);

}

Topping::Topping(const Topping& other)

{

this->name = NULL;

\*this = other;

}

Topping::~Topping()

{

delete[] (this->name);

}

char\* Topping::getName() const

{

return name;

}

float Topping::getPrice() const

{

return price;

}

char Topping::getCoverage() const

{

return coverage;

}

void Topping::setName(const char\* newName)

{

//if (name)

//delete[] this->name;

int newLength = strlen(newName);

this->name = new char[newLength + 1];

assert(name);

strcpy\_s(this->name, newLength + 1, newName);

}

void Topping::setPrice(float price)

{

this->price = price;

}

void Topping::setCoverage(const char coverage)

{

this->coverage = coverage;

}

void Topping::operator=(const Topping& other)

{

if (name)

delete[] this->name;

setName(other.getName());

setPrice(other.getPrice());

setCoverage(other.getCoverage());

}

std::ostream& operator<<(std::ostream& os, const Topping& other)

{

os << "Topping name: " << other.name << ", price: " << other.price << ", coverage: " << other.coverage;

return os;

}

bool operator>(const Topping& one, const Topping& other)

{

return (one.price > other.price);

}

bool operator<(const Topping& one, const Topping& other)

{

return (one.price < other.price);

}

bool operator==(const Topping& one, const Topping& other)

{

return (one.coverage == other.coverage);

}

Pizza.h

//Dalya William & Guy Rahamim

#pragma once

#include"Topping.h"

#include <iostream>

class Pizza

{

private:

//member variables

char\* type;

float basePrice;

int num\_top;

Topping\* toppings;

public:

//constructors

Pizza(char\* type, float basePrice, int num\_top);

Pizza(const Pizza& other);

//destructor

~Pizza();

//getters

Topping\* getToppings();

//setters

void setType(char\* type);

void setBasePrice(float basePrice);

void setNum\_top(int num\_top);

void initToppings(int num\_top);

void copyToppings(int num\_top, Topping\* toppings);

//class specific functions

float calcPrice() const;

//operator overrloaders

Pizza& operator=(const Pizza& other);

friend std::ostream& operator<<(std::ostream& os, const Pizza& other);

friend void operator+(Pizza& one, const Topping& other);

friend bool operator==(const Pizza& one, const Pizza& other);

};

Pizza.cpp

//Dalya William & Guy Rahamim

#include "Pizza.h"

Pizza::Pizza(char\* type, float basePrice, int num\_top)

{

this->type = NULL;

this->toppings = NULL;

setType(type);

setBasePrice(basePrice);

setNum\_top(num\_top);

initToppings(num\_top);

}

Pizza::Pizza(const Pizza& other)

{

this->type = NULL;

this->toppings = NULL;

\*this = other;

}

Pizza::~Pizza()

{

delete[] type;

delete[] toppings;

}

Topping\* Pizza::getToppings()

{

return toppings;

}

void Pizza::setType(char\* type)

{

delete[] this->type;

int newLength = strlen(type);

this->type = new char[newLength + 1];

assert(type);

strcpy\_s(this->type, newLength + 1, type);

}

void Pizza::setBasePrice(float basePrice)

{

this->basePrice = basePrice;

}

void Pizza::setNum\_top(int num\_top)

{

this->num\_top = num\_top;

}

void Pizza::initToppings(int num\_top)

{

this->toppings = NULL;

this->toppings = new Topping[num\_top];

assert(this->toppings);

}

void Pizza::copyToppings(int num\_top, Topping\* toppings)

{

delete[] this->toppings; // delete previous value

this->toppings = new Topping[num\_top];

for (int i = 0; i < num\_top; i++)

{

this->toppings[i] = toppings[i];

}

}

float Pizza::calcPrice() const

{

float finalPrice = basePrice;

for (int i = 0; i < num\_top; i++)

{

finalPrice += toppings[i].getPrice();

}

return finalPrice;

}

Pizza& Pizza::operator=(const Pizza& other)

{

setType(other.type);

basePrice = other.basePrice;

num\_top = other.num\_top;

copyToppings(other.num\_top, other.toppings);

return \*this;

}

std::ostream& operator<<(std::ostream& os, const Pizza& other)

{

os << "type is: " << other.type << "\n";

for (int i = 0; i < other.num\_top; i++)

{

os << "toping "<<i+1<<" " <<other.toppings[i] << std::endl;

}

os << "\nFinal price is: " << other.calcPrice();

return os;

}

void operator+(Pizza& one, const Topping& other)

{

bool toppingAdded = false;

for (int i = 0; (i < one.num\_top) && (toppingAdded==false); i++)

{

if (one.toppings[i].getName() == NULL)

{

one.toppings[i] = other;

toppingAdded = true;

}

}

}

bool operator==(const Pizza& one, const Pizza& other)

{

if (one.num\_top != other.num\_top)

return false;

for (int i = 0; i < one.num\_top; i++)

{

if (!(one.toppings[i] == other.toppings[i]))

return false;

}

return true;

}

MainFile.cpp

//Dalya William & Guy Rahamim

#include <iostream>

#include "Pizza.h"

#include "Topping.h"

#define MAX\_LENGTH 200

Pizza initPizza();

Topping initTopping();

int main()

{

//taking input for the first pizza

std::cout << "Please enter details of pizza 1:" << std::endl;

Pizza pizza1 = initPizza();

//taking input for the second pizz

std::cout << "\nPlease enter details of pizza 2:" << std::endl;

Pizza pizza2 = initPizza();

//printing first pizza

std::cout<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pizza 1\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

std::cout << "Pizza 1:\n" << pizza1 << std::endl << std::endl;

//printing second pizza

std::cout<< "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Pizza 2\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

std::cout << "Pizza 2:\n" << pizza2 << std::endl << std::endl;

//creating new toppings for comparison operators

Topping t1("olives", 'f', 3);

Topping t2("tuna", 'f', 5);

Topping t3("pineapple", 'f', 10);

//comparisons

std::cout << "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*COMPARISONS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*" << std::endl;

std::cout << "pizza1 == pizza2: " << ((pizza1 == pizza2)? "true":"false") << std::endl;

std::cout << "t1 == t2: " << (t1 == t2 ? "true" : "false") << std::endl;

std::cout << "t2 > t3: " << (t2 > t3 ? "true" : "false") << std::endl;

std::cout << "t2 < t3: " << (t2 < t3 ? "true" : "false") << std::endl;

}

Pizza initPizza()

{

//declaring variables for creating a pizza.

Topping\* toppings;

char type[MAX\_LENGTH];

float baseprice = 25.f;

int num\_top;

//asking for type and number of number of toppings

std::cout << "What type of dough would you like? ";

std::cin >> type;

std::cout << "\nHow many toppings would you like? ";

std::cin >> num\_top;

//creating a pizza.

Pizza pizza(type, baseprice, num\_top);

//adding topping to pizza using initTopping function

for (int i = 0; i < num\_top; i++)

{

Topping tempTopping = initTopping();

pizza + tempTopping;

}

return pizza;

}

Topping initTopping()

{

//declaring variables for creating a topping.

char coverage;

char name[MAX\_LENGTH];

float price =5.f;

//asking for topping name and coverge.

std::cout << "what topping would you like to add? ";

std::cin >> name;

std::cout << "\nWhat part of the pizza should the " << name << " cover?\n";

std::cout << "l-left half \nr-right half \nf-the entire pizza" << std::endl;

std::cin >> coverage;

//creating the topping

Topping top(name, coverage, price);

return top;

}