Министерство образования и науки Кыргызской Республики

Кыргызский государственный технический университет

им. И.Раззакова

Факультет информационных технологий

Кафедра «Программное обеспечение компьютерных систем»

Направление: 710400 «Программная инженерия»

Дисциплина: « Объектно-ориентрованное проектирование / Объектно-ориентрованное программирование»

**ОТЧЕТ**

По лабораторной работе №3

Тема: «Конструктор и деструктор»

Выполнила: студентка группы

ПИ(б)-3-21 Табылдиева Шоола

Проверил: Мусабаев Э. Б.

Бишкек – 2024

Задания

1. Разработайте программу с классом **Timer**, в которой таймер действует как таймер обратного отсчета. При создании объекта типа **timer** ему присваивается начальное значение времени. В результате вызова функции **run()** таймер начинает отсчет в сторону уменьшающихся значений, пока не достигнет значения 0, после чего зазвонит звонок. **Создайте 3 варианта конструктора, т.е. конструктор перегрузите 3 раза** для того, чтобы можно было указывать время в секундах с помощью целого числа или строки, или в минутах и секундах, если указываются два целых числа. Эта программа использует библиотечную функцию **clock(),** возвращающую число тиков, прошедших с момента запуска программы. Поделив это значение на макрос **CLK\_TCK**, получаем значение в секундах. Прототипы для **clock**() и **CLK\_TCK** содержатся в заголовочном файле time.h.

#include <iostream>

#include <windows.h>

#include <string>

#include <chrono>

#include <thread>

class Timer {

private:

int seconds;

public:

Timer(int initialseconds) : seconds(initialseconds) {}

Timer(const std::string& timestring) {

parsetimestring(timestring);

}

Timer(int minutes, int seconds) : seconds(minutes \* 60 + seconds) {}

void run() {

while (seconds > 0) {

displaytime();

delay(1000); // delay for 1000 milliseconds (1 second)

seconds--;

}

std::cout << "timer expired. beep beep beep!\n";

playsound();

}

private:

void parsetimestring(const std::string& timestring) {

size\_t pos1 = timestring.find(':');

size\_t pos2 = timestring.rfind(':');

int hours = std::stoi(timestring.substr(0, pos1));

int minutes = std::stoi(timestring.substr(pos1 + 1, pos2 - pos1 - 1));

int secs = std::stoi(timestring.substr(pos2 + 1));

seconds = hours \* 3600 + minutes \* 60 + secs;

}

void displaytime() const {

int hours = seconds / 3600;

int minutes = (seconds % 3600) / 60;

int secs = seconds % 60;

std::cout << "time remaining: " << hours << "h " << minutes << "m " << secs << "s\n";

}

void delay(int milliseconds) {

auto start = std::chrono::steady\_clock::now();

while (std::chrono::duration\_cast<std::chrono::milliseconds>(std::chrono::steady\_clock::now() - start).count() < milliseconds);

}

void playsound() const {

Beep(1000, 500);

}

};

int main() {

Timer timer1(5);

Timer timer2("00:00:03");

Timer timer3(0, 3);

timer1.run();

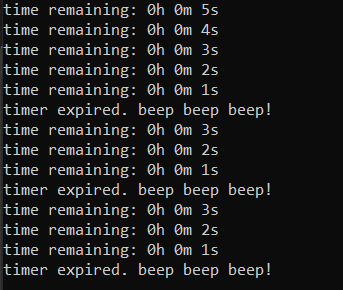
timer2.run();

timer3.run();

return 0;

}

Результат:



1. Создайте класс **Nomenclature**, описывающий товары на складе магазина. Закрытыми элементами класса будут: название товара, оптовая цена, розничная наценка и количество товаров на складе. Включите в класс открытые функции подсчета возможного чистого дохода при продаже этого товара и вывода всех данных о товаре на экран. Для инициализации и удаления объектов класса используйте конструкторы и деструкторы.

#include <iostream>

#include <string>

class Nomenclature {

private:

std::string itemName;

double wholesalePrice;

double retailMarkup;

int stockQuantity;

public:

Nomenclature(const std::string& name, double wholesale, double markup, int quantity)

: itemName(name), wholesalePrice(wholesale), retailMarkup(markup), stockQuantity(quantity) {}

~Nomenclature() {

std::cout << "Object for item '" << itemName << "' has been destroyed.\n";

}

double calculateNetIncome(int soldQuantity) const {

double netIncome = (retailMarkup - wholesalePrice) \* soldQuantity;

return netIncome;

}

void displayItemDetails() const {

std::cout << "Item Name: " << itemName << "\n";

std::cout << "Wholesale Price: " << wholesalePrice << "\n";

std::cout << "Retail Markup: " << retailMarkup << "\n";

std::cout << "Stock Quantity: " << stockQuantity << "\n";

}

};

int main() {

Nomenclature item("Product", 30, 50, 80);

std::cout << "Item Details:\n";

item.displayItemDetails();

int soldQuantity = 20;

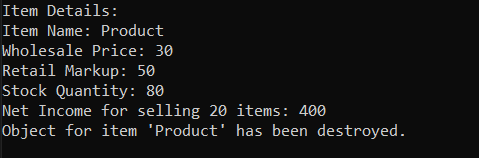
double netIncome = item.calculateNetIncome(soldQuantity);

std::cout << "Net Income for selling " << soldQuantity << " items: " << netIncome << "\n";

return 0;

}

Результат:



1. Создайте класс **Soft**, который содержит информацию об установленном программном обеспечении. Закрытыми элементами класса будут: название программы, разработчик, занимаемый объем, дата завершения лицензии. Включите в класс открытые функции подсчета количества дней до завершения лицензии и вывода всех данных об установленном программном обеспечении на экран. Для инициализации и удаления объектов класса используйте конструкторы и деструкторы.

#include <iostream>

#include <string>

#include <ctime>

class soft {

private:

std::string programname;

std::string developer;

double occupiedspace;

time\_t licenseexpirationdate;

public:

soft(const std::string& name, const std::string& dev, double space, const std::string& expirationdate)

: programname(name), developer(dev), occupiedspace(space) {

tm expirationdatestruct = parsedate(expirationdate);

licenseexpirationdate = mktime(&expirationdatestruct);

}

~soft() {

std::cout << "software for program '" << programname << "' has been uninstalled.\n";

}

void daysuntilexpiration() const {

time\_t currenttime;

time(&currenttime);

tm currentlocaltime;

localtime\_s(&currentlocaltime, &currenttime);

currentlocaltime.tm\_hour = 0;

currentlocaltime.tm\_min = 0;

currentlocaltime.tm\_sec = 0;

double secondsremaining = difftime(licenseexpirationdate, mktime(&currentlocaltime));

int daysremaining = static\_cast<int>(secondsremaining / (60 \* 60 \* 24));

if (daysremaining < 0) {

std::cout << "license already expired\n";

}

else if (daysremaining == 0) {

std::cout << "license expires today\n";

}

else {

std::cout << "days until license expiration: " << daysremaining << " days\n";

}

}

void displaysoftwaredetails() const {

std::cout << "software details:\n";

std::cout << "program name: " << programname << "\n";

std::cout << "developer: " << developer << "\n";

std::cout << "occupied space: " << occupiedspace << " gb\n";

tm expirationdatestruct;

localtime\_s(&expirationdatestruct, &licenseexpirationdate);

std::cout << "license expiration date: " << expirationdatestruct.tm\_mday << "/"

<< expirationdatestruct.tm\_mon + 1 << "/" << expirationdatestruct.tm\_year + 1900 << "\n";

}

private:

tm parsedate(const std::string& datestring) const {

tm result = {};

sscanf\_s(datestring.c\_str(), "%d/%d/%d", &result.tm\_mday, &result.tm\_mon, &result.tm\_year);

result.tm\_mon -= 1;

result.tm\_year -= 1900;

return result;

}

};

int main() {

soft software("mysoftware", "it company", 2.5, "15/02/2023");

software.displaysoftwaredetails();

software.daysuntilexpiration();

std::cout << "\n";

soft software2("mysoftware2", "it company", 2.5, "15/02/2024");

software2.displaysoftwaredetails();

software2.daysuntilexpiration();

std::cout << "\n";

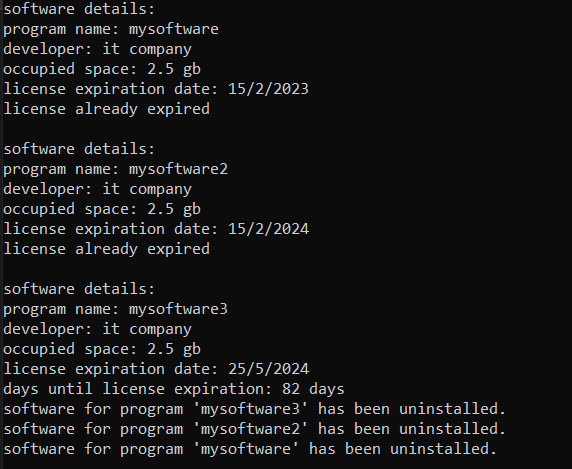
soft software3("mysoftware3", "it company", 2.5, "25/05/2024");

software3.displaysoftwaredetails();

software3.daysuntilexpiration();

return 0;

}



Ответы на вопросы

1. Что такое конструктор? Как он вызывается на выполнение?

Конструктор – это метод класса, выполняющийся автоматически в момент создания объекта. Конструктор предназначен для инициализации объекта. Конструктор вызывается автоматически при создании объекта класса.

1. Особенности конструктора, отличающие его от других методов класса?
2. Имя конструктора совпадает с именем класса.
3. У конструкторов не существует возвращаемых значений.
4. Отсутствие типа возвращаемого значения. Нельзя получить указатель на конструктор.
5. Если пользователь не указал ни одного конструктора, компилятор создает его автоматически.
6. Конструктор не наследуется.
7. Конструктор можно перегружать по разным аргументам.
8. Существует конструктор по умолчанию (без аргументов) и конструктор с аргументами.
9. Что такое конструктор по умолчанию? В каком случае следует его использовать?

Конструктор по умолчанию – это конструктор, который может быть вызван без передачи аргументов. Для первоначальной инициализации нулевым значением можно использовать конструктор по умолчанию.

1. Способы инициализации в конструкторе по умолчанию? Какой способ инициализации предпочтительнее?
2. Инициализация полей объекта с помощью списка инициализации:

***Counter(): count(0)***

***{ };***

Список инициализации находится после двоеточия. Инициализирующее значение находится в скобках после имени поля.

1. Инициализация полей объекта с помощью присвоения им значений в теле конструктора. Её в нашем примере можно выполнить так:

***Counter ()***

***{count =0;}***

Второй способ не содержит ошибок, но он не рекомендуется.

Причины:

1. Инициализация полей должна происходить до начала исполнения тела конструктора. Чтобы в теле конструктора можно было производить более сложные действия, чем обычная инициализация.
2. Список инициализации – это единственный способ задать первоначальные значения константам и ссылкам.
3. Что такое конструктор с аргументами? Когда используют конструктор с аргументами?

Конструктор с аргументами – это конструктор, имеющий хотя бы один аргумент, может инициализировать поля значениями, переданными ему в качестве аргументов. К тому же этот конструктор значительно упрощает код программы.

1. Какие преимущества у конструктора с аргументами перед методом Set()?

В функции main () пишем одну команду при использовании метода “конструктор с аргументами” для инициализации трех полей объекта. Код программы значительно сокращается при использовании конструктора с аргументами.

1. Что такое деструктор? Для чего его используют?

Деструктор – это метод, автоматически вызываемый при уничтожении объекта.

Для чего используют деструктор:

Освобождение ресурсов: Деструктор часто используется для освобождения ресурсов, которые были выделены объекту во время его жизни. Например, это может быть освобождение динамически выделенной памяти, закрытие файловых дескрипторов и т.д.

Завершение процессов: В деструкторе можно реализовать логику, которая должна выполниться перед уничтожением объекта. Например, сохранение данных в файл перед выходом.

Контроль жизненного цикла объекта: Деструктор обеспечивает контроль за тем, что происходит при уничтожении объекта, и позволяет разработчику выполнять необходимые действия в этот момент.

1. Особенности деструктора.
2. Имеет имя, совпадающее с именем конструктора, а, следовательно, и класса.
3. Предваряется символом тильда.
4. Не возвращает значения.
5. Не имеет аргументов
6. В какой последовательности выполняется конструкторы, в какой деструкторы классов?

Конструкторы базовых классов вызываются в порядке их объявления. Деструкторы вызываются в обратном порядке.

1. Можно ли перегружать конструктор?

Рассмотрим два основных способа перегрузкиконструктора:

Способ 1. Использование в одной программе обоих типов конструктора.

Способ 2. Для перегрузки используются конструкторы с одним именем, но отличающиеся количеством аргументов.

1. Что такое конструктор копирования?

Конструктор копирования – это конструктор, с помощью которого возможно сделать третий способ инициализации полей объекта. При этом способе для инициализации полей объекта используются поля уже существующего объекта. Для этого не нужно самим создавать специальный конструктор, поскольку такой конструктор предоставляется компилятором для каждого создаваемого класса и называется конструктором копирования по умолчанию.