**МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ**

**КЫРГЫЗСКОЙ РЕСПУБЛИКИ**

**КЫРГЫЗСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. И.Раззакова**

**ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ**

Кафедра: **Программное обеспечение компьютерных систем**

Курс «Объектно-ориентированное программирование»

**ОТЧЕТ**

**Лабораторная работа №3**

Выполнил: студент группы ПИ-3-21

Копжашаров Азамат

Проверил: Мусабаев Э.Б.

**Бишкек 2024**

# **Задание №1**

**Постановка задачи:**

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.

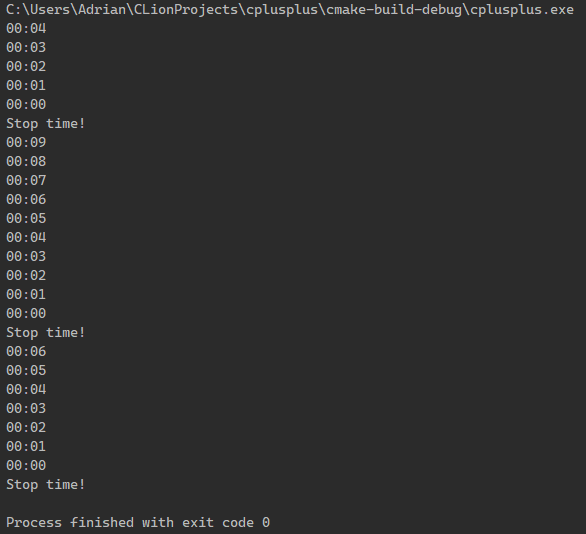
Эта программа использует библиотечную функцию **clock()**, возвращающую число тиков, прошедших с момента запуска программы. Поделив это значение на макрос **CLK\_TCK**, получаем значение в секундах. Прототипы для **clock()** и **CLK\_TCK** содержатся в заголовочном файле time.h.

**Исходные данные:**

**Исходный код программы на C++:**

// 3.1  
#include <iostream>  
#include <ctime>  
#include <iomanip>  
using namespace std;  
  
class Timer{  
private:  
 int seconds;  
  
public:  
 // constructor for int seconds  
 Timer(int sec) : seconds(sec) {}  
  
 // constructor for "min:sec"  
 Timer(const string &timeString) {  
 int minutes, seconds;  
 sscanf(timeString.c\_str(), "%d:%d", &minutes, &seconds); // c.str() not to copy entire string - memory efficient usage  
 this->seconds = minutes \* 60 + seconds;  
 }  
  
 // constructor for int minutes, int seconds  
 Timer(int min, int sec): seconds(min \* 60 + sec) {}  
  
 void run(){  
 clock\_t start = clock();  
  
 while (seconds > 0) {  
 clock\_t current = clock();  
 double elapsedSeconds = static\_cast<double>(current - start) / CLOCKS\_PER\_SEC;  
  
 if (elapsedSeconds >= 1.0) {  
 seconds--;  
 start = current;  
 int minutes = seconds / 60;  
 int remainingSeconds = seconds % 60;  
  
 cout << setfill('0') << setw(2) << minutes << ":"  
 << setfill('0') << setw(2) << remainingSeconds << endl;  
 }  
 }  
  
 cout << "Stop time!" << endl;  
 }  
};  
  
int main() {  
  
 Timer timer1(5);  
 timer1.run();  
  
 Timer timer2("00:10");  
 timer2.run();  
  
 Timer timer3(0, 7);  
 timer3.run();  
  
 return 0;  
}

**Тесты:**

****

# **Задание №2**

**Постановка задачи:**

**Опишите класс «студенческая группа».**

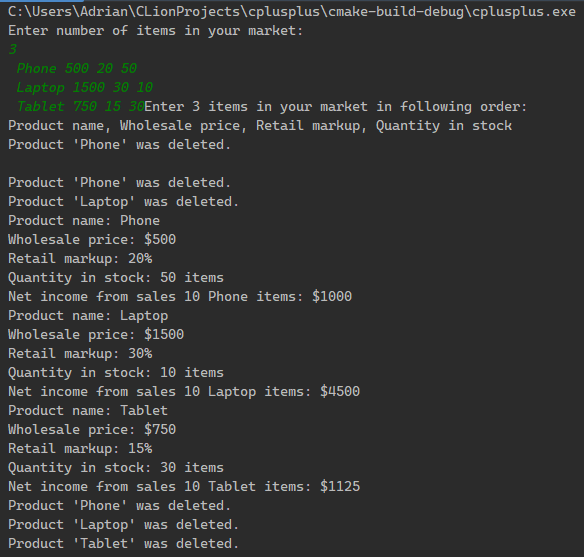
Создайте класс **Nomenclature**, описывающий товары на складе магазина. Закрытыми элементами класса будут: название товара, оптовая цена, розничная наценка и количество товаров на складе. Включите в класс открытые функции подсчета возможного чистого дохода при продаже этого товара и вывода всех данных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

**Исходные данные:**

**Исходный код программы на C++:**

// 3.2  
#include <iostream>  
#include <string>  
#include <vector>  
  
using namespace std;  
  
class Nomenclature {  
private:  
 string productName;  
 double wholesalePrice;  
 double retailMarkup;  
 int quantityInStock;  
  
public:  
 // Конструктор с параметрами  
 Nomenclature(const string& pn, double wp, double rm, int q)  
 : productName(pn), wholesalePrice(wp), retailMarkup(rm), quantityInStock(q) {}  
  
 // Деструктор  
 ~Nomenclature() {  
 cout << "Product '" << productName << "' was deleted. " << endl;  
 }  
  
 // Функция для подсчета чистого дохода  
 double calculateNetIncome(int soldQuantity) const {  
 double revenue = soldQuantity \* (wholesalePrice \* (retailMarkup / 100 + 1.0) );  
 double expenses = soldQuantity \* wholesalePrice;  
 double netIncome = revenue - expenses;  
 cout << "Net income from sales " << soldQuantity << " " << this->productName << " items: $" << netIncome << endl;  
 return netIncome;  
 }  
  
 // Функция для вывода данных о товаре на экран  
 void displayProductDetails() const {  
 cout << "Product name: " << productName << endl;  
 cout << "Wholesale price: $" << wholesalePrice << endl;  
 cout << "Retail markup: " << retailMarkup << "%" << endl;  
 cout << "Quantity in stock: " << quantityInStock << " items" << endl;  
 }  
};  
  
int main() {  
 vector<Nomenclature> market;  
 cout << "Enter number of items in your market: " << endl;  
 int n;  
 cin >> n;  
 cout << "Enter " << n << " items in your market in following order: " << endl;  
 cout << "Product name, Wholesale price, Retail markup, Quantity in stock" << endl;  
 for(int i = 0; i < n; i++){  
 string pn;  
 double wp, rm;  
 int q;  
 cin >> pn >> wp >> rm >> q;  
 market.emplace\_back(pn, wp, rm, q);  
 }  
 for(Nomenclature &item : market){  
 item.displayProductDetails();  
 // item.calculateNetIncome(10);  
 double lol = item.calculateNetIncome(10);  
 }  
 return 0;  
}  
  
/\*  
 3  
 Phone 500 20 50  
 Laptop 1500 30 10  
 Tablet 750 15 30  
 \*/

**Тесты:**



# **Задание №3**

**Постановка задачи:**

**Опишите класс, реализующий стек (Stack).**

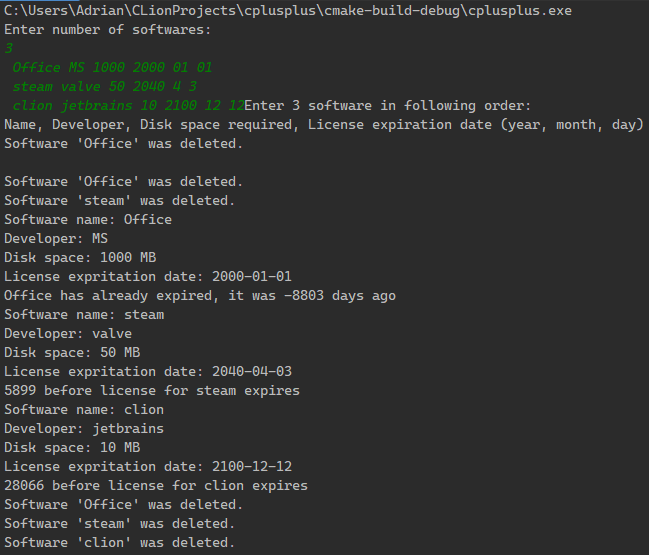
Создайте класс **Soft**, который содержит информацию об установленном программном обеспечении. Закрытыми элементами класса будут: название программы, разработчик, занимаемый объем, дата завершения лицензии. Включите в класс открытые функции подсчета количества дней до завершения лицензии и вывода всех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

**Исходные данные:**

**Исходный код программы на C++:**

// 3.3  
#include <iostream>  
#include <string>  
#include <ctime>  
#include <iomanip>  
#include <vector>  
  
using namespace std;  
  
class Soft {  
private:  
 string programName;  
 string developer;  
 double diskSpace;  
 tm licenseExpirationDate; // Используем структуру tm для хранения даты  
  
public:  
 // Конструктор с параметрами  
 Soft(const string& name, const string& dev, double space, int year, int month, int day)  
 : programName(name), developer(dev), diskSpace(space) {  
 licenseExpirationDate.tm\_year = year - 1900; // tm\_year отсчитывается от 1900 года  
 licenseExpirationDate.tm\_mon = month - 1; // tm\_mon отсчитывается от 0 (январь)  
 licenseExpirationDate.tm\_mday = day;  
 licenseExpirationDate.tm\_hour = 0;  
 licenseExpirationDate.tm\_min = 0;  
 licenseExpirationDate.tm\_sec = 0;  
 }  
  
 // Деструктор  
 ~Soft() {  
 cout << "Software '" << programName << "' was deleted." << endl;  
 }  
  
 // Функция для подсчета количества дней до завершения лицензии  
 int daysUntilLicenseExpiration() const {  
 time\_t now = time(nullptr); // Получаем текущее время в секундах с 1 января 1970 года  
 tm currentTime = \*localtime(&now); // Создаем копию текущего времени  
 tm expirationDate = licenseExpirationDate; // Создаем копию даты окончания лицензии  
  
 // Разница в секундах между текущим временем и датой завершения лицензии  
 time\_t difference = mktime(&expirationDate) - mktime(&currentTime);  
  
 // Преобразуем разницу в дни  
 int days = difference / (60 \* 60 \* 24);  
 if(difference >= 0) cout << days << " before license for " << this->programName << " expires" << endl;  
 else cout << this->programName << " has already expired, it was " << days << " days ago" << endl;  
  
  
 return days;  
 }  
  
 // Функция для вывода данных о программном обеспечении на экран  
 void displaySoftwareDetails() const {  
 cout << "Software name: " << programName << endl;  
 cout << "Developer: " << developer << endl;  
 cout << "Disk space: " << diskSpace << " MB" << endl;  
 cout << "License expritation date: " << put\_time(&licenseExpirationDate, "%Y-%m-%d") << endl;  
 }  
};  
  
int main() {  
  
 vector<Soft> software;  
  
 cout << "Enter number of softwares: " << endl;  
 int n;  
 cin >> n;  
 cout << "Enter " << n << " software in following order: " << endl;  
 cout << "Name, Developer, Disk space required, License expiration date (year, month, day)" << endl;  
 for(int i = 0; i < n; i++){  
 string name, dev;  
 double disk;  
 int year, month, day;  
 cin >> name >> dev >> disk >> year >> month >> day;  
 software.emplace\_back(name, dev, disk, year, month, day);  
 }  
  
 for(auto &iter : software){  
 iter.displaySoftwareDetails();  
 // iter.daysUntilLicenseExpiration();  
 int foo = iter.daysUntilLicenseExpiration();  
 }  
  
 return 0;  
}  
/\*  
 3  
 Office MS 1000 2000 01 01  
 steam valve 50 2040 4 3  
 clion jetbrains 10 2100 12 12  
 \*/

**Тесты:**



# **Ответы на вопросы:**

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

Конструктор — это специальный метод класса, который инициализирует поля объекта значениями в момент создания объекта автоматически.

Всю необходимую работу по вызову конструктора выполняет компилятор.

**2. Особенности конструктора, отличающие его от других методов класса?**

1. Имя конструктора совпадает с именем класса.
2. У конструктора нет возвращаемого значения, даже результата типа void.  
   Это объясняется тем, что конструктор автоматически вызывается системой, следовательно, не существует вызывающей функции, которой конструктор мог бы возвратить значение. Значит, указание возвращаемого типа в конструкторе не имеет смысла.
3. Конструктор выполняется при создании объектов класса. Вернее, при создании каждого объекта класса.
4. Конструктор не наследуется.
5. Конструктор может иметь аргументы, а может не иметь.
6. Конструктор можно перегружать.
7. Конструктор, отработав, запускает деструктор.

**3. Что такое конструктор по умолчанию? В каком случае следует его использовать?**

**Конструктор по умолчанию —** это конструктор, который может быть вызван без передачи аргументов.

Для того, чтобы быть уверенным в том, что все поля объекта первоначально инициализированы корректными значениями, используется первоначальная инициализация полей всех объекта одинаковыми значениям автоматически в момент его создания. Чаще всего нулевыми значениями (для первоначальной очистки данных).

Использование этого конструктора удобно при наличии большого количества полей. В этом случае конструктор подстраховывает, чтобы не забыть инициализировать все поля. А потом можно присваивают другие значения, отличные от нуля, или всем или определенным полям объекта.

**4. Способы инициализации в конструкторе по умолчанию? Какой способ инициализации предпочтительнее?**

1. Инициализация полей объекта с помощью **списка инициализации:**

class Count{  
 int count;  
 Count(): count(0){}  
};

2. Инициализация полей объекта с помощью присвоения им значений в теле конструктора:

class Count{  
 int count;  
 Count() {  
 count = 0;  
 }  
};

**5. 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.

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

Уменьшение кода при одинаковых результатах.

**7. Что такое деструктор? Для чего его используют?**

Существует другой специальный метод класса, который автоматически вызывается при уничтожении объекта/ов, созданного/ых конструктором. Этот метод называется **деструктором**. Он работает в паре с конструктором.

Конструктор инициализирует поля объекта, а также отслеживает этот объект, пока он не выполнит свою задачу. В этот момент конструктор вызывает деструктор, который должен освободить памяти от отработанного объекта и весь оставшийся мусор от работы объекта.

Если память под объект отведена в динамической памяти с помощью спецификаторов new и delete, писать деструктор надо обязательно. Если же new не используется, писать деструктор все равно нужно для последующих проверок деструктором работы класса и удаления нединамических объектов.

**8. Особенности деструктор**

1. Деструктор имеет имя, совпадающее с именем конструктора, а, следовательно, и с именем класса.
2. Перед деструктором пишут символ тильда **~**.
3. Аргументов не имеет.
4. Возвращаемого значения не имеет.
5. Пишется следом за конструктором.

**9. В какой последовательности выполняется конструкторы, в какой деструкторы классов?**

Конструкторы базовых классов вызываются в порядке их объявления, а деструкторы вызываются в обратном.

**10. Можно ли перегружать конструктор?**

Да, для перегрузки используются конструкторы с одним именем, но отличающиеся количеством аргументов.

**11. 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. Копирующий конструктор имеет единственный аргумент, являющийся объектом того же класса, что и конструктор.