**Ministry of education and science of the Kyrgyz Republic**

**Kyrgyz State Technical University named after I.Razzakov**

**Faculty of Information Technologies**

**Department of Software of Computer Systems**

**Major: 710400 «Software Engineering»**

Report

Discipline: «**Object-Oriented Design**»

Software requirements

Done by**:** Idirisov Aziret SE(eng)- 1- 21

Checked by: Musabaev E.B.

Bishkek 2024

Task №1

1. Введение

Целью данного программного обеспечения (ПО) является разработка таймера обратного отсчета, который позволяет пользователю задать определенное количество времени в минутах и секундах или в секундах, а затем отсчитывает время до нуля с выводом оставшегося времени и звукового сигнала по завершении отсчета.

2. Требования к функциональности

Возможность ввода времени:

Пользователю предоставляется возможность ввода времени для обратного отсчета.

Время может быть введено как в минутах и секундах, так и в секундах.

Ввод времени в минутах и секундах должен быть в диапазоне от 0 до 59 для каждой единицы времени.

Отсчет времени:

После ввода времени таймер начинает отсчет до нуля.

Оставшееся время выводится пользователю на экран.

Звуковой сигнал по завершении отсчета:

По истечении установленного времени проигрывается звуковой сигнал, уведомляющий пользователя о завершении отсчета.

3. Требования к интерфейсу

Ввод времени:

Пользователю предоставляется интерфейс для ввода времени.

Ввод осуществляется через командную строку или интерфейс программы.

Визуализация отсчета:

Оставшееся время отображается на экране во время работы таймера.

4. Требования к безопасности

Проверка ввода:

Ввод времени должен быть проверен на корректность, чтобы предотвратить ошибочный ввод пользователем.

Обработка исключений:

Программа должна обрабатывать возможные исключения, связанные с некорректным вводом данных, чтобы предотвратить аварийное завершение работы.

5. Требования к производительности

Точность отсчета:

Точность отсчета времени должна быть высокой, чтобы обеспечить точное измерение времени.

Эффективное использование ресурсов:

Программа должна эффективно использовать ресурсы компьютера, чтобы минимизировать нагрузку на систему.

class Timer {

private:

int seconds;

public:

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

Timer(const std::string& time) : seconds(std::stoi(time)) {}

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

void run() {

std::cout << "Timer started." << std::endl;

while (seconds > 0) {

std::cout << "Time remaining: " << seconds << " seconds." << std::endl;

--seconds;

std::clock\_t start = std::clock();

while ((std::clock() - start) / static\_cast<double>(CLOCKS\_PER\_SEC) < 1) {}

}

std::cout << "Time's up! Ringing!" << std::endl;

}

};

int main() {

int minutes, seconds;

std::cout << "Enter minutes: ";

std::cin >> minutes;

while (minutes < 0 || minutes > 59) {

std::cout << "Minutes must be between 0 and 59. Please enter again: ";

std::cin >> minutes;

}

std::cout << "Enter seconds: ";

std::cin >> seconds;

while (seconds < 0 || seconds > 59) {

std::cout << "Seconds must be between 0 and 59. Please enter again: ";

std::cin >> seconds;

}

Timer timer(minutes, seconds);

timer.run();

return 0;

}

Task №2

class Nomenclature {

private:

std::string productName;

short int price;

float retailMarkup;

int quantity;

public:

// Конструктор

Nomenclature(const std::string& product, short int price, float roznas, int quantity)

: productName(product), price(price), retailMarkup(roznas), quantity(quantity) {}

// Деструктор

~Nomenclature() {

std::cout << "Товар удален" << std::endl;

}

// Функция подсчета возможного чистого дохода при продаже товара

float calculateNetIncome() const {

float retailPrice = price \* (1 + retailMarkup / 100);

return (retailPrice - price) \* quantity;

}

// Функция вывода всех данных о товаре на экран

void displayProductDetails() const {

std::cout << "Название товара: " << productName << std::endl;

std::cout << "Оптовая цена: " << price << std::endl;

std::cout << "Розничная наценка: " << retailMarkup << "%" << std::endl;

std::cout << "Количество товаров на складе: " << quantity << std::endl;

}

};

Функциональные требования:

Добавление товара:

Пользователь должен иметь возможность добавить новый товар на склад магазина.

Для каждого товара необходимо указать название, оптовую цену, розничную наценку и количество товаров на складе.

Удаление товара:

Пользователь должен иметь возможность удалить товар из списка номенклатуры.

При удалении товара необходимо освободить ресурсы, занимаемые данным товаром.

Изменение информации о товаре:

Пользователь должен иметь возможность изменить информацию о существующем товаре, такую как его название, цена, наценка и количество на складе.

Подсчет возможного чистого дохода:

ПО должно предоставлять функциональность для автоматического расчета возможного чистого дохода при продаже каждого товара.

Расчет должен основываться на оптовой цене, розничной наценке и количестве товаров на складе.

Отображение списка товаров:

Пользователь должен иметь возможность просмотреть список всех товаров в номенклатуре магазина.

Для каждого товара должна быть доступна информация о его названии, оптовой цене, розничной наценке и количестве на складе.

Нефункциональные требования:

Интерфейс пользователя:

Интерфейс ПО должен быть интуитивно понятным и простым в использовании для пользователя.

Для взаимодействия с номенклатурой товаров должен быть предоставлен графический или текстовый интерфейс.

Производительность:

ПО должно обеспечивать быструю обработку запросов пользователя, включая добавление, удаление, изменение и просмотр информации о товарах.

Надежность:

ПО должно быть надежным и стабильным, минимизируя возможность ошибок и сбоев.

Должны быть предприняты меры для защиты данных о товарах от потери или повреждения.

Task №3

class Soft {

private:

string name;

string developer;

double size;

time\_t licenseExpirationDate;

public:

Soft(const std::string& name, const std::string& developer, double size, time\_t expirationDate)

: name(name), developer(developer), size(size), licenseExpirationDate(expirationDate) {}

~Soft() {

cout << "Объект класса Soft удален" << endl;

}

int daysUntilExpiration() const {

time\_t now = time(nullptr);

double secondsLeft = difftime(licenseExpirationDate, now);

return static\_cast<int>(secondsLeft / (60 \* 60 \* 24));

}

void display() const {

cout << "Название программы: " << name << endl;

cout << "Разработчик: " << developer << endl;

cout << "Занимаемый объем: " << size << " MB" << endl;

cout << "Дата завершения лицензии: " << ctime(&licenseExpirationDate) << endl;

}

};

Введение

Целью данного программного обеспечения является управление информацией об установленном программном обеспечении, включая его название, разработчика, размер и дату завершения лицензии. ПО также предоставляет возможность подсчитывать количество дней до завершения лицензии и выводить данные об установленном программном обеспечении на экран.

2. Требования к функциональности

2.1. Управление данными об установленном программном обеспечении

ПО должно предоставлять возможность добавления, изменения, удаления и просмотра информации об установленном программном обеспечении.

Для каждого программного продукта должны быть указаны следующие данные:

Название программы.

Разработчик.

Занимаемый объем.

Дата завершения лицензии.

2.2. Подсчет количества дней до завершения лицензии

ПО должно предоставлять возможность подсчитывать количество дней до завершения лицензии для каждого установленного программного обеспечения.

2.3. Вывод данных на экран

ПО должно предоставлять возможность вывода всех данных об установленном программном обеспечении на экран.

3. Требования к интерфейсу пользователя

Интерфейс пользователя должен быть интуитивно понятным и легко использоваться.

Должны быть предоставлены соответствующие элементы управления для выполнения всех операций, описанных в разделе 2.

4. Требования к безопасности

Доступ к данным об установленном программном обеспечении должен быть защищен паролем или другими методами аутентификации, чтобы предотвратить несанкционированный доступ.

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

Конструктор - это специальный метод класса, который вызывается при создании объекта этого класса. Он инициализирует новый объект.

2) Особенности конструктора, отличающие его от других методов класса?

Особенность конструктора заключается в том, что его имя совпадает с именем класса, и он вызывается автоматически при создании объекта класса.

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

Конструктор по умолчанию - это конструктор, который не принимает аргументов. Его следует использовать, когда нет необходимости передавать параметры при создании объекта.

4) Способы инициализации в конструкторе по умолчанию? Какой способ инициализации Способы инициализации в конструкторе по умолчанию могут включать инициализацию членов класса значениями по умолчанию или другими способами инициализации, например, через списки инициализации. Предпочтительнее использовать списки инициализации, так как они более эффективны.

5) Что такое конструктор с аргументами? Когда используют конструктор с аргументами?

Конструктор с аргументами - это конструктор, который принимает параметры при создании объекта. Он используется, когда необходимо передать значения для инициализации объекта.

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

Преимущества конструктора с аргументами перед методом Set() заключаются в том, что конструктор сразу устанавливает значения при создании объекта, что делает код более читаемым и понятным.

7) Что такое деструктор? Для чего его используют?

Деструктор - это специальный метод класса, который вызывается при уничтожении объекта этого класса. Он используется для освобождения ресурсов, занятых объектом.

8) Особенности деструктор

Особенность деструктора заключается в том, что его имя начинается с символа тильды (~), и он вызывается автоматически при уничтожении объекта.

9) В какой последовательности выполняется конструкторы, в какой деструкторы классов?

Конструкторы вызываются в порядке иерархии классов от базового к производным при создании объекта, а деструкторы вызываются в обратном порядке при уничтожении объекта.

10) Можно ли перегружать конструктор?

Да, конструктор можно перегружать, т.е. создавать несколько конструкторов с разным набором параметров.

11) Что такое конструктор копирования?

Конструктор копирования - это специальный конструктор, который создает новый объект, копируя значения членов существующего объекта. Он используется для создания копии объекта.