Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ (ПНИПУ)

Электротехнический факультет

Дисциплина: основы алгоритмизации и программирования, 2 семестр

ОТЧЁТ

"Лабораторная работа ООП № 3. Перегрузка операций."

Выполнил

Студент группы РИС-22-2Б

Хохряков А.С

Проверил

Доцент кафедры ИТАС

Полякова О.А

Пермь 2023

# Содержание отчета

1. Постановка задачи.
2. Анализ задачи.
3. UML-Диаграмма
4. Ответы на вопросы
5. Текст программы

Постановка задачи

1. Определить пользовательский класс.

2. Определить в классе следующие конструкторы: без параметров, с параметрами,

копирования.

3. Определить в классе деструктор.

4. Определить в классе компоненты-функции для просмотра и установки полей

данных (селекторы и модификаторы).

5. Перегрузить операцию присваивания.

6. Перегрузить операции ввода и вывода объектов с помощью потоков.

7. Перегрузить операции, указанные в варианте.

8. Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

Вариант 3:

Создать класс Time для работы с временными интервалами. Интервал должен быть представлен в виде двух полей: минуты типа int и секунды типа int. при выводе минуты отделяются от секунд двоеточием. Реализовать:

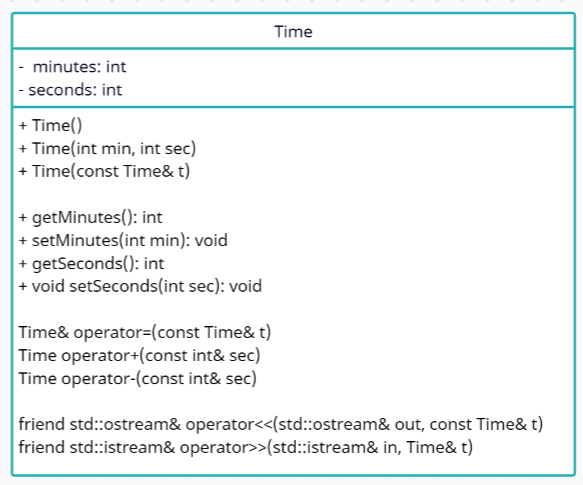
добавление секунд (учесть, что в минуте не может быть более 60 секунд)

вычитание секунд.

Анализ задачи

1. Определить класс "Time" с открытыми и закрытыми полями и методами.
2. Конструкторы класса, которые задают значения полям при создании объекта.
3. Деструктор класса, который освобождает ресурсы, выделенные для объекта.
4. Селекторы и модификаторы, которые возвращают значения полей и изменяют их соответственно.
5. Перегрузка оператора присваивания, который присваивает значения одного объекта класса "Time" другому.
6. Перегрузка операторов ввода и вывода для класса "Time", которые позволяют выводить время на экран и считывать время с клавиатуры в заданном формате.
7. Перегрузка операторов сложения и вычитания, которые позволяют добавлять или вычитать секунды к времени.
8. Функции "addSeconds" и "subtractSeconds", которые добавляют или вычитают секунды к времени, учитывая, что 1 минута = 60 секунд.
9. Функция "main", которая использует объекты класса "Time" и его методы для выполнения некоторых операций со временем.
10. Создание объекта "t1" класса "Time" с заданными значениями минут и секунд, вывод его на экран.
11. Создание объекта "t2" класса "Time" без заданных значений, ввод его с клавиатуры, вывод его на экран.
12. Создание объекта "t3" класса "Time", который является результатом сложения объекта "t1" и 45 секунд, вывод его на экран.
13. Создание объекта "t4" класса "Time", который является результатом вычитания из объекта "t2" 20 секунд, вывод его на экран.

UML – Диаграмма



Ответы на вопросы

1. Дружественные функции и классы используются для предоставления доступа к закрытым членам класса другим классам или функциям, которые не являются его членами. Это может быть полезно, если вы хотите сделать определенные данные доступными для использования в других классах или функциях, но не хотите делать их общедоступными через публичные методы класса.
2. Правила описания дружественных функций:
3. Декларация дружественной функции должна быть помещена внутрь определения класса.
4. Дружественная функция не является методом класса и не может иметь доступа к членам класса напрямую, только через объект класса, переданный в качестве параметра.
5. Для объявления дружественной функции используется ключевое слово friend, которое указывается перед функцией внутри определения класса.

Особенности дружественных функций:

1. Дружественные функции могут быть объявлены как внутри класса, так и вне его.
2. Дружественные функции не наследуются от базового класса и не являются членами класса.
3. Дружественные функции не имеют доступа к закрытым членам класса без передачи объекта класса в качестве параметра.
4. Унарные операции могут быть перегружены путем объявления функции-оператора внутри класса и указания имени оператора в качестве имени функции.
5. Унарная функция-операция, определяемая внутри класса, должна иметь только один операнд, который является объектом класса.
6. Унарная функция-операция, определяемая вне класса, должна иметь только один операнд, который должен быть объектом класса и передаваться в функцию в качестве параметра.
7. Бинарная функция-операция, определяемая внутри класса, должна иметь два операнда: левый операнд, который является объектом класса, и правый операнд, который может быть любым типом данных.
8. Бинарная функция-операция, определяемая вне класса, должна иметь два операнда, которые должны передаваться в функцию в качестве параметров.
9. Префиксные и постфиксные унарные операции отличаются тем, что в префиксной форме оператор выполняется перед получением значения операнда, а в постфиксной - после получения значения операнда.
10. Операция присваивания может быть перегружена путем объявления функции-оператора внутри класса и указания имени оператора в качестве имени функции.
11. Операция присваивания должна возвращать ссылку на объект класса (A&), чтобы позволить выполнить цепочку операций присваивания. Это также позволяет использовать конструкцию вида a = b = c.
12. Операции ввода-вывода могут быть перегружены путем объявления функций-операторов внутри класса.
13. В данном случае, компилятор будет воспринимать операцию как префиксную форму инкремента объекта класса Student. То есть, вызов оператора ++ будет выполнять увеличение значения объекта s на 1 и возвращать измененный объект s.
14. В данном случае, компилятор будет воспринимать операцию как вызов дружественной функции-оператора ++, которая также будет выполнять увеличение значения объекта класса Student на 1 и возвращать измененный объект. Однако, для того чтобы вызвать данную функцию, необходимо явно указать объект класса, передав его в качестве аргумента функции, например: ++s;
15. В данном случае, компилятор будет воспринимать операцию как вызов функции-оператора < для объектов класса Student. Данная функция будет сравнивать значения объектов a и b и возвращать true, если значение объекта a меньше значения объекта b, и false в противном случае. Результат операции будет передан в стандартный поток вывода cout.
16. В данном случае, компилятор будет воспринимать операцию как вызов дружественной функции-оператора > для объектов класса Student. Данная функция также будет сравнивать значения объектов a и b и возвращать true, если значение объекта a больше значения объекта b, и false в противном случае. Результат операции будет передан в стандартный поток вывода cout. Обратите внимание, что в данном случае функция-оператор объявлена как дружественная, а не метод класса, поэтому объекты класса не передаются в качестве аргументов.

Текст программы

#include <iostream>

class Time {

private:

int minutes;

int seconds;

public:

Time() : minutes(0), seconds(0) {}

Time(int min, int sec) : minutes(min), seconds(sec) {}

Time(const Time& t) : minutes(t.minutes), seconds(t.seconds) {}

~Time() {}

int getMinutes() const {

return minutes;

}

void setMinutes(int min) {

minutes = min;

}

int getSeconds() const {

return seconds;

}

void setSeconds(int sec) {

seconds = sec;

}

Time& operator=(const Time& t) {

minutes = t.minutes;

seconds = t.seconds;

return \*this;

}

friend std::ostream& operator<<(std::ostream& out, const Time& t) {

out << t.minutes << ":" << t.seconds;

return out;

}

friend std::istream& operator>>(std::istream& in, Time& t) {

in >> t.minutes >> t.seconds;

return in;

}

Time operator+(const int& sec) const {

Time result = \*this;

result.addSeconds(sec);

return result;

}

Time operator-(const int& sec) const {

Time result = \*this;

result.subtractSeconds(sec);

return result;

}

void addSeconds(int sec) {

int totalSeconds = minutes \* 60 + seconds + sec;

minutes = totalSeconds / 60;

seconds = totalSeconds % 60;

}

void subtractSeconds(int sec) {

int totalSeconds = minutes \* 60 + seconds - sec;

minutes = totalSeconds / 60;

seconds = totalSeconds % 60;

}

};

int main() {

Time t1(3, 30);

std::cout << t1 << std::endl;

Time t2;

std::cout << "Enter time: ";

std::cin >> t2;

std::cout << "You entered time: " << t2 << std::endl;

Time t3 = t1 + 45;

std::cout << "t1 + 45 seconds = " << t3 << std::endl;

Time t4 = t2 - 20;

std::cout << "t2 - 20 seconds = " << t4 << std::endl;

return 0;

}