Лабораторная работа №3
по курсу «Высокоуровневое программирование» (2 семестр)
«Перегрузка операторов»

Оглавление

Цель: приобретение практических навыков и знаний по работе с перегрузкой операторов.

Задачи:

1. Изучить понятия оператора и что он из себя представляет;
2. Выяснить виды и способы перегрузки операторов;
3. Научиться применять перегрузку операторов на практике;
4. Изучить методы и случаи применения перегрузок;
5. Научиться соединять пользовательские объекты с потоками ввода / вывода;
6. Познакомиться с понятием функтора.

Содержание отчёта:

1. Титульный лист;
2. Цель, задачи работы;
3. Формулировка общего задания;
4. Uml-диаграмма реализованных классов;
5. Листинги пользовательских функций, классов и основной программы;
6. Результаты работы;
7. Выводы по работе в целом.

В начало

Основные теоретические сведения

Перегрузка операторов

Давайте вспомним, что в языке C++ присутствует возможность перегружать функции. Перегрузка будет работать только в том случаи, если функции имеют одинаковые имена, но разные типы аргументов. **Перегрузка** позволяет использовать одно и тоже имя функции, но разные аргументы и соответственно реализацию.

Помимо функций язык программирования C++ позволяет перегружать операторы. Про операторы Вы можете прочитать в интернете, либо в соответствующем пособии. Сейчас нам важно знать лишь то, что оператор в языке C++ реализован функцией, которая принимает один, ли или два аргумента. Один для унарных операторов, два – для бинарных операторов (тернарный оператор не перегружается). Также нельзя перегрузить оператор `sizeof`, оператор разрешения области видимости «`::`», оператор выбора члена «`.`» и указатель, как оператор выбора члена «`->`». Для бинарных операторов аргумент слева это первый параметр, аргумент справа – второй. Перегрузка операторов представляет из себя обычную функцию.

Также, существуют следующие правила перегрузки:

- Вы можете перегрузить только существующие операторы. Вы не можете создавать новые или переименовывать существующие. Например, вы не можете создать оператор `**` для выполнения операции возведения в степень.
- По крайней мере один из операндов перегруженного оператора должен быть пользовательского типа данных. Это означает, что вы не можете перегрузить `operator+()` для выполнения операции сложения значения типа `int` со значением типа `double`. Однако вы можете перегрузить `operator+()` для выполнения операции сложения значения типа `int` с объектом класса `Mystring`.
- Изначальное количество операндов, поддерживаемых оператором, изменить невозможно. Т.е. с бинарным оператором используются только два операнда, с унарным — только один, с тернарным — только три.
- Все операторы сохраняют свой приоритет и ассоциативность по умолчанию (независимо от того, для чего они используются), и это не может быть изменено.

Для чего использовать перегрузку операторов? Вы можете перегрузить оператор `+` для соединения объектов вашего класса `String` или для выполнения операции сложения двух объектов вашего класса `Fraction`. Вы можете перегрузить оператор `<<` для вывода вашего класса на экран (или записи в файл). Вы можете перегрузить оператор равенства (==) для сравнения двух объектов класса и т.д. Подобные применения делают перегрузку операторов одной из самых полезных особенностей в языке C++, так как это упрощает процесс работы с классами и открывает новые возможности. Например, операторы побитовых сдвигов были перегружены в классах потоков, тот самый `cin` и `cout`.

Способы перегрузки операторов:

- через дружественные функции;
- через обычные функции;
- через методы класса.

В начало

Перегрузка операторов через дружественные функции

Для этого напишем класс денежной единицы, которая будет хранить целую часть и копейки:

```
class Ruble {

public:
	Ruble(unsigned int rub, unsigned int kop) {
		m_rubles = rub;
		m_kopecks = kop;
	}

	void Print() {
		cout << m_rubles << ',' << m_kopecks;
	}

private:
	unsigned int m_rubles{};
	unsigned int m_kopecks{};
};
int main() {

	Ruble sum_1(25, 17);
	Ruble sum_2(17, 12);

	Ruble sum_3 = sum_1 + sum_2;

	return 0;
}
```

При попытке скомпилировать данный код, мы получим ошибку. Это всё из – за того, что компилятор не знает, как складывать наш тип данных. Давайте исправим это используя дружественные функции. **Дружественная функция** – это функция, которая может получать доступ к закрытым члена класса. Чтобы объявить дружественную функцию, необходимо прописать её прототип с ключевым словом: `friend` в теле класса, а потом реализовать её вне класса, словно это обычная функция (не как метод класса).

```
class Ruble {

public:
	Ruble(unsigned int rub, unsigned int kop) {
		m_rubles = rub;
		m_kopecks = kop;
	}

	void Print() {
		cout << m_rubles << ',' << m_kopecks;
	}

	// объявляем, что функция operator+ является дружественной
	// для класса Ruble
	friend Ruble operator+(Ruble s_1, Ruble s_2);

private:
	unsigned int m_rubles{};
	unsigned int m_kopecks{};
};
```

```
// реализация функции
Ruble operator+(Ruble s_1, Ruble s_2) {

	int rub = s_1.m_rubles + s_2.m_rubles;
	int kop = s_1.m_kopecks + s_2.m_kopecks;

	return Ruble(rub, kop);
}

int main() {

	Ruble sum_1(25, 17);
	Ruble sum_2(17, 12);

	Ruble sum_3 = sum_1 + sum_2;

	sum_3.Print();

	return 0;
}
```

Теперь наша программа скомпилируется и отобразит верный результат. Как видно из кода, чтобы выполнить перегрузку оператора необходимо вместо имени функции написать ключевое слово: `operator`, а затем нужный знак, который будем перегружать. В аргументах необходимо прописать те переменные, которые будут складываться, первой переменной будет значение слева от равно, второе справа. Возвращать функция будет объект `Ruble`, который будет создан на основе суммы значений первых двух. Как видно, в функции мы обращаемся непосредственно к приватным полям объекта это возможно благодаря пометки этой функции, как дружественной для класса. Таким образом можно перегрузить и другие операторы для данного класса: «`-`, `*`, `/`».

Для перегрузок операторов работают свои перегрузки с разными аргументами. Например, давайте добавим ещё одну перегрузку для оператора суммы, которая будет прибавлять число к нашему классу:

```
class Ruble {

public:
	Ruble(unsigned int rub, unsigned int kop) {
		m_rubles = rub;
		m_kopecks = kop;
	}

	void Print() {
		cout << m_rubles << ',' << m_kopecks;
	}

	// объявляем, что функция operator+ является дружественной
	// для класса Ruble
	friend Ruble operator+(Ruble s_1, Ruble s_2);
	friend Ruble operator+(Ruble s_1, int num);

private:
	unsigned int m_rubles{};
	unsigned int m_kopecks{};
};
```

```
Ruble operator+(Ruble s_1, int num) {

	int rub = s_1.m_rubles + num;
	int kop = s_1.m_kopecks;

	return Ruble(rub, kop);
}
int main() {

	Ruble sum_1(25, 17);
	Ruble sum_2(17, 12);

	Ruble sum_3 = sum_1 + 12;

	sum_3.Print();

	return 0;
}
```

Теперь стало возможным прибавление числа к нашему классу.

Здесь стоит обратить внимание на то, что порядок аргументов в операторе важен!

```
Ruble sum_3 = sum_1 + 12;
Ruble sum_3 = 12 + sum_1;
```

Эти две строки не эквивалентны. В первой строке будет вызван наш перегруженный оператор, который принимает сперва класс `Ruble`, а потом число типа `int`. А во – второй строчке нужен оператор, который будет принимать сперва число, а потом наш класс, у нас такого оператора нет, поэтому компилятор выдаст нам ошибку.

В начало

Перегрузка операторов через обычные функции

Использование дружественной функции для перегрузки оператора удобно тем, что мы имеем прямой доступ ко всем членам класса, с которым работаем. В примере, приведенном выше, наша дружественная функция перегрузки оператора + имеет прямой доступ к закрытым членам класса `Ruble`. Однако, если нам не нужен доступ к членам определенного класса, мы можем перегрузить оператор и через обычную функцию. Обратите внимание, в классе должен присутствовать геттеры `getRub()` и `getKop()`, с помощью которых мы можем получить доступ к `m_rub` и `m_kop` извне класса. Перепишем перегрузку оператора `+` через обычную функцию:

```
class Ruble {

public:
	Ruble(unsigned int rub, unsigned int kop) {
		m_rubles = rub;
		m_kopecks = kop;
	}
```

```
        void Print() {
                cout << m_rubles << ',' << m_kopecks;
        }

        unsigned int getRub() {
                return m_rubles;
        }

        unsigned int getKop() {
                return m_kopecks;
        }

private:
        unsigned int m_rubles{};
        unsigned int m_kopecks{};
};

Ruble operator+(Ruble s_1, int num) {

        int rub = s_1.getRub() + num;
        int kop = s_1.getKop();

        return Ruble(rub, kop);
}

Ruble operator+(Ruble s_1, Ruble s_2) {

        int rub = s_1.getRub() + s_2.getRub();
        int kop = s_1.getKop() + s_2.getKop();

        return Ruble(rub, kop);
}
```

Поскольку принцип перегрузки операторов через обычные и дружественные функции почти идентичен (они просто имеют разные уровни/условия доступа к закрытым членам класса), то единственное отличие заключается в том, что в случае с дружественной функцией, её нужно обязательно объявить в классе + определить вне тела класса (или в классе), в то время как обычную функцию достаточно просто определить вне тела класса, без указания дополнительного прототипа функции.

В начало

Перегрузка операторов ввода / вывода

Для классов с множеством переменных-членов, выводить в консоль каждую переменную по отдельности может быть несколько утомительно. Конечно, было бы проще написать отдельную функцию, или метод для вывода, которую можно было бы повторно использовать. Например, функция `print()` из наших примеров:

```
        void Print() {
                cout << m_rubles << ',' << m_kopecks;
        }
```

Теперь уже намного лучше, но здесь также есть свои нюансы. Поскольку метод `print()` имеет тип `void`, то его нельзя вызывать в середине стейтмента вывода. Вместо этого стейтмент вывода придется разбить на несколько частей (строк). А вот если бы мы могли просто написать:

```
cout << sum_1 << sum_2 << sum_3 << endl;
```

И получить тот же результат, но без необходимости разбивать стейтмент вывода на несколько строк и помнить название функции вывода. К счастью, это можно сделать, перегрузив оператор вывода <<.

Перегрузка оператора вывода << аналогична перегрузке оператора + (оба являются бинарными операторами), за исключением того, что их типы различны.

Рассмотрим выражение `std::cout << point`. Если оператором является `<<`, то чем тогда являются операнды? Левым операндом является объект `std::cout`, а правым – объект нашего класса `Ruble`. `std::cout` фактически является объектом типа `std::ostream`, поэтому перегрузка оператора `<<` выглядит следующим образом:

```
friend ostream& operator<<(ostream& out, const Ruble& sum);
ostream& operator<<(ostream& out, const Ruble& sum) {
	sum.Print();

	return out;
}
```

Всё довольно просто. Обратите внимание, насколько проще стал стейтмент вывода по сравнению с другими стейментами из вышеприведенных примеров. Наиболее заметным отличием является то, что `std::cout` стал параметром out в нашей функции перегрузки (который затем станет ссылкой на `std::cout` при вызове этого оператора).

Самое интересное здесь – тип возврата. С перегрузкой арифметических операторов мы вычисляли и возвращали результат по значению. Однако, если вы попытаетесь возвратить `std::ostream` по значению, то получите ошибку компилятора. Это случится из-за того, что `std::ostream` запрещает свое копирование. В этом случае мы возвращаем левый параметр в качестве ссылки. Это не только предотвращает создание копии `std::ostream`, но также позволяет нам «связать» стейтменты вывода вместе, например, `std::cout << sum_1 << std::endl`.

Вы могли бы подумать, что, поскольку оператор `<<` не возвращает значение обратно в `caller`, то мы должны были бы указать тип возврата `void`. Но подумайте, что произойдет, если наш оператор `<<` будет возвращать `void`. Когда компилятор обрабатывает `std::cout << sum_1 << std::endl`, то, учитывая правила приоритета/ассоциативности, он будет обрабатывать это выражение как

`(std::cout << sum_1) << std::endl`. Тогда `std::cout << sum_1` приведет к вызову функции перегрузки оператора `<<`, которая возвратит `void`, и вторая часть выражения будет обрабатываться как `void << std::endl` – в этом нет смысла!

Возвращая параметр out в качестве возвращаемого значения выражения `(std::cout << sum_1)` мы возвращаем `std::cout`, и вторая часть нашего выражения обрабатывается как `std::cout << std::endl`.

Каждый раз, когда мы хотим, чтобы наши перегруженные бинарные операторы были связаны таким образом, то левый операнд обязательно должен быть возвращен (по ссылке). Возврат левого параметра по ссылке в этом случае работает, так как он передается в функцию самим вызовом этой функции, и должен оставаться даже после выполнения и возврата этой функции. Таким образом, мы можем не беспокоиться о том, что ссылаемся на что-то, что выйдет из области видимости и уничтожится после выполнения функции.

Здесь также стоит обратить внимание на второй параметр перегруженного оператора, он имеет константный тип, это гарантирует то, что объект не будет изменён функцией. В таком случаи мы не сможем у этого объекта вызвать методы, которые не помечены константой, которая будет говорить о том, что метод на сто процентов не меняет состояние объекта. Это делается так:

```
void Print() const {
	cout << m_rubles << ',' << m_kopecks;
}
```

Также можно перегрузить и оператор ввода. Всё почти так же, как и с оператором вывода, но главное, что нужно помнить – `std::cin` является объектом типа `std::istream`. Вот наш класс `Ruble` с перегруженным оператором ввода `>>`:

```
	friend istream& operator>>(istream& in, Ruble& sum);
	istream& operator>>(istream& in, Ruble& sum) {
		in >> sum.m_rubles;
		in >> sum.m_kopecks;

		return in;
	}
cin >> sum_1;
```

В начало

Перегрузка операторов через методы класса

Конвертация перегрузки через дружественную функцию в перегрузку через метод класса довольно-таки проста:

Перегружаемый оператор определяется как метод класса, вместо дружественной функции (Ruble::operator+ вместо friend operator+). Левый

параметр из функции перегрузки выбрасывается, вместо него — неявный объект, на который указывает указатель `*this`.

Внутри тела функции перегрузки все ссылки на левый параметр могут быть удалены (например, `sum_1.m_rub` становится `m_rub`, который неявно ссылается на текущий объект с помощью указателя `*this`). Теперь та же перегрузка оператора `+`, но уже через метод класса:

```
class Ruble {

public:
	Ruble(unsigned int rub, unsigned int kop) {
		m_rubles = rub;
		m_kopecks = kop;
	}

	void Print() const {
		cout << m_rubles << ',' << m_kopecks;
	}

	unsigned int getRub() {
		return m_rubles;
	}

	unsigned int getKop() {
		return m_kopecks;
	}

	Ruble operator+(int num) {

		int rub = m_rubles + num;
		int kop = m_kopecks;

		return Ruble(rub, kop);
	}

	Ruble operator+(Ruble s_1) {

		int rub = m_rubles + s_1.m_rubles;
		int kop = m_kopecks + s_1.m_kopecks;

		return Ruble(rub, kop);
	}

	friend ostream& operator<<(ostream& out, const Ruble& sum);

	friend istream& operator>>(istream& in, Ruble& sum);

private:
	unsigned int m_rubles{};
	unsigned int m_kopecks{};
};
```

Обратите внимание, использование оператора `+` не изменяется, но реализация отличается. Наша дружественная функция с двумя параметрами становится методом класса с одним параметром, причем левый параметр в перегрузке через дружественную функцию `(&ruble)`, в перегрузке через метод класса становится неявным объектом, на который указывает указатель `*this`.

Итак, если мы можем перегрузить оператор через дружественную функцию или через метод класса, то что тогда выбрать? Прежде чем мы дадим ответ на этот вопрос, вам нужно узнать еще несколько деталей.

Не всё может быть перегружено через дружественные функции. Операторы присваивания (=), индекса ([]), вызова функции (()) и выбора члена (->) перегружаются через методы класса – это требование языка C++.

Не всё может быть перегружено через методы класса. Через метод класса перегрузить оператор `<<` мы не сможем. Почему?

Потому что при перегрузке через метод класса в качестве левого операнда используется текущий объект. В этом случае левым операндом является объект типа `std::ostream`. `std::ostream` является частью Стандартной библиотеки C++. Мы не можем использовать `std::ostream` в качестве левого неявного параметра, на который бы указывал скрытый указатель `*this`, так как указатель `*this` может указывать только на текущий объект текущего класса, члены которого мы можем изменить, поэтому перегрузка оператора `<<` должна осуществляться через дружественную функцию.

Аналогично, хотя мы можем перегрузить `operator+(Ruble, int)` через метод класса (как мы делали выше), мы не можем перегрузить `operator+(int, Ruble)` через метод класса, поскольку `int` теперь является левым операндом, на который указатель `*this` указывать не может.

Перегрузка операторов через методы класса не используется, если левый операнд не является классом (например, `int`), или это класс, который мы не можем изменить (например, `std::ostream`).

Какой способ перегрузки и когда следует использовать?

В большинстве случаев язык C++ позволяет выбирать самостоятельно способ перегрузки операторов. Но при работе с бинарными операторами, которые не изменяют левый операнд (например, `operator+()`), обычно используется перегрузка через обычную или дружественную функции, поскольку такая перегрузка работает для всех типов данных параметров (даже если левый операнд не является объектом класса или является объектом класса, который изменить нельзя). Перегрузка через обычную/дружественную функцию имеет дополнительное преимущество «симметрии», так как все операнды становятся явными параметрами (а не как у перегрузки через метод класса, когда левый операнд становится неявным объектом, на который указывает указатель `*this`).

При работе с бинарными операторами, которые изменяют левый операнд (например, `operator+=()`), обычно используется перегрузка через методы класса. В этих случаях левым операндом всегда является объект класса, на который указывает скрытый указатель `*this`.

Унарные операторы обычно тоже перегружаются через методы класса, так как в таком случае параметры не используются вообще.

Поэтому:

- Для операторов присваивания (`=`), индекса (`[]`), вызова функции (`()`) или выбора члена (`->`) используйте перегрузку через методы класса;
- Для унарных операторов используйте перегрузку через методы класса;
- Для перегрузки бинарных операторов, которые изменяют левый операнд (например, `operator+=()`) используйте перегрузку через методы класса, если это возможно;
- Для перегрузки бинарных операторов, которые не изменяют левый операнд (например, `operator+()`) используйте перегрузку через обычные/дружественные функции.

В начало

Перегрузка унарных операторов +, – и логического НЕ

Рассмотрим унарные операторы плюс (`+`), минус (`-`) и логическое НЕ (`!`), которые работают с одним операндом. Так как они применяются только к одному объекту, то их перегрузку следует выполнять через методы класса.

Например, перегрузим унарный оператор минус (`-`) для класса `Ruble`:

```
Ruble operator-() const {
              return Ruble(-m_rubles, -m_kopecks);
}
```

Всё довольно-таки просто. Перегрузка отрицательного унарного оператора минус (`-`) осуществляется через метод класса, так как явные параметры в функции перегрузки отсутствуют (только неявный объект, на который указывает скрытый указатель `*this`). Оператор `-` возвращает объект `Ruble` с отрицательным значением `m_rubles` и `m_kopecks`. Поскольку этот оператор не изменяет объект класса `Ruble`, то мы можем (и должны) сделать функцию перегрузки константной (чтобы иметь возможность использовать этот оператор и с константными объектами класса `Ruble`).

Обратите внимание, путаницы между отрицательным унарным оператором минус (`-`) и бинарным оператором минус (`-`) нет, так как они имеют разное количество параметров.

Вот еще один пример: оператор `!` является логическим оператором НЕ, который возвращает `true`, если результатом выражения является `false` и возвращает `false`, если результатом выражения является `true`. В языке C++ значение 0 обозначает `false`, а любое другое ненулевое значение обозначает `true`, поэтому, если логический оператор `!` применять к целочисленным

значениям, то он будет возвращать `true`, если значением переменной является `0`, в противном случае – `false`.

Следовательно, при работе с классами, оператор `!` будет возвращать `true`, если значением объекта класса является `false`, `0` или любое другое значение, заданное как дефолтное (по умолчанию) при инициализации, в противном случае – оператор `!` будет возвращать `false`.

В следующем примере мы добавим перегрузку оператора отрицания для класса ответа на вопрос:

```
class Answer {
public:
	Answer(bool check) {
		m_state = check;
	}

	bool operator!() {
		return !m_state;
	}

private:
	bool m_state{};
};
```

В начало

Перегрузка операторов сравнения

Принципы перегрузки операторов сравнения те же, что и в перегрузке других операторов, которые мы рассматривали на предыдущих уроках. Поскольку все операторы сравнения являются бинарными и не изменяют свои левые операнды, то выполнять перегрузку следует через дружественные функции. Например, перегрузим оператор равенства `==` и оператор неравенства `!=` для класса очков, а также оператор больше `>` и меньше `<`:

```
class Score {
public:
	Score(int score) {
		m_score = score;
	}

	friend bool operator>(const Score &s_1, const Score &s_2);
	friend bool operator<(const Score &s_1, const Score &s_2);
	friend bool operator==(const Score &s_1, const Score &s_2);
	friend bool operator!=(const Score &s_1, const Score &s_2);

private:
	int m_score{};
};

bool operator>(const Score &s_1, const Score &s_2) {
	return s_1.m_score > s_2.m_score;
}

bool operator<(const Score &s_1, const Score &s_2) {
	return s_1.m_score < s_2.m_score;
```

```
}

bool operator==(const Score &s_1, const Score &s_2) {
	return s_1.m_score == s_2.m_score;
}

bool operator!=(const Score &s_1, const Score &s_2) {
	return !(s_1 == s_2);
}
```

Всё просто. Поскольку результат выполнения оператора `!=` является прямо противоположным результату выполнения оператора `==`, то мы определили оператор `!=`, используя уже перегруженный оператор `==` (уменьшив, таким образом, количество кода, сложность и возможность возникновения ошибок).

В начало

Перегрузка операторов инкремента и декремента версии префикс

Перегрузка операторов инкремента (`++`) и декремента (`--`) довольно-таки проста, но с одним маленьким нюансом. Есть две версии операторов инкремента и декремента: версия префикс (например, `++x`, `--y`) и версия постфикс (например, `x++`, `y--`). Поскольку операторы инкремента и декремента являются унарными и изменяют свои операнды, то перегрузку следует выполнять через методы класса. Перегрузка операторов инкремента и декремента версии префикс аналогична перегрузке любых других унарных операторов:

```
class Score {
public:
	Score(int score) {
		m_score = score;
	}

	Score& operator++() {
		++m_score;

		// возвращаем разыменованный указатель на текущий объект класса
		return *this;
	}

	Score& operator--() {
		--m_score;

		// возвращаем разыменованный указатель на текущий объект класса
		return *this;
	}

private:
	int m_score{};
};
```

В начало

Перегрузка операторов инкремента и декремента версии постфикс

Обычно перегрузка функций осуществляется, если они имеют одно и то же имя, но разное количество и типы параметров. Рассмотрим случай с операторами инкремента/декремента версий префикс/постфикс. Оба имеют одно и то же имя (например, `operator++`), унарные и принимают один параметр одного и того же типа данных. Как же тогда их различить при перегрузке?

Дело в том, что язык C++ использует фиктивную переменную (или «фиктивный параметр») для операторов версии постфикс. Этот фиктивный целочисленный параметр используется только с одной целью: отличить версию постфикс операторов инкремента/декремента от версии префикс. Выполним перегрузку операторов инкремента/декремента версии префикс и постфикс в одном классе:

```
class Score {
public:
	Score(int score) {
		m_score = score;
	}

	Score& operator++() {
		++m_score;

		// возвращаем разыменованный указатель на текущий объект класса
		return *this;
	}

	Score operator++(int) {
		Score tmp(m_score);

		++(*this);

		// возвращаем объект со старым значением
		return tmp;
	}

	Score& operator--() {
		--m_score;

		// возвращаем разыменованный указатель на текущий объект класса
		return *this;
	}

	Score operator--(int) {
		Score tmp(m_score);

		--(*this);

		// возвращаем объект со старым значением
		return tmp;
	}

private:
	int m_score{};
};
```

Здесь есть несколько интересных моментов:

- Во-первых, мы отделили версию постфикс от версии префикс использованием целочисленного фиктивного параметра в версии постфикс.
- Во-вторых, поскольку фиктивный параметр не используется в реализации самой перегрузки, то мы даже не предоставляем ему имя. Таким образом, компилятор будет рассматривать эту переменную, как простую заглушку (заполнитель места), и даже не будет предупреждать нас о том, что мы объявили переменную, но никогда её не использовали.
- В-третьих, операторы версий префикс и постфикс выполняют одно и то же задание: оба увеличивают/уменьшают значение переменной объекта. Разница между ними только в значении, которое они возвращают.

Рассмотрим последний пункт детально. Операторы версии префикс возвращают объект после того, как он был увеличен или уменьшен. В версии постфикс нам нужно возвращать объект до того, как он будет увеличен или уменьшен. И тут конфуз! Если мы увеличиваем или уменьшаем объект, то мы не можем возвратить его до выполнения инкремента/декремента, так как операция увеличения/уменьшения уже произошла. С другой стороны, если мы возвращаем объект до выполнения инкремента/декремента, то сама операция увеличения/уменьшения объекта не выполнится.

Решением является использование временного объекта с текущим значением переменной-члена. Тогда можно будет увеличить/уменьшить исходный объект, а временный объект возвратить обратно в `caller`. Таким образом, `caller` получит копию объекта до того, как фактический объект будет увеличен или уменьшен, и сама операция инкремента/декремента выполнится успешно.

Обратите внимание, это означает, что возврат значения по ссылке невозможен, так как мы не можем возвратить ссылку на локальную переменную (объект), которая будет уничтожена после завершения выполнения тела функции. Также это означает, что операторы версии постфикс обычно менее эффективны, чем операторы версии префикс, из-за дополнительных расходов ресурсов на создание временного объекта и выполнения возврата по значению вместо возврата по ссылке.

Наконец, мы реализовали перегрузку операторов версии постфикс через уже перегруженные операторы версии префикс. Таким образом, мы сократили дублированный код и упростили внесение изменений в наш класс в будущем (т.е. упростили поддержку кода).

В начало

Перегрузка оператора индексации []

Оператор индексации является одним из операторов, перегрузка которого должна выполняться через метод класса. Функция перегрузки оператора `[]` всегда будет принимать один параметр: значение индекса (элемент массива, к которому требуется доступ). В нашем случае с `IntArray` нам нужно, чтобы пользователь просто указал в квадратных скобках индекс для возврата значения элемента по этому индексу:

```
class IntArray {
public:
	int& operator[] (const int index) {
		return m_array[index];
	}

private:
	int m_array[10]{};
};
```

Теперь всякий раз, когда мы будем использовать оператор индексации (`[]`) с объектом класса `IntArray`, компилятор будет возвращать соответствующий элемент массива `m_array`! Это позволит нам непосредственно как получать, так и присваивать значения элементам `m_array`.

Почему оператор индексации `[]` использует возврат по ссылке?

Рассмотрим подробнее, как обрабатывается стейтмент `array[4] = 5`. Поскольку приоритет оператора индексации выше приоритета оператора присваивания, то сначала выполняется часть `array[4]`. `array[4]` приводит к вызову функции перегрузки оператора `[]`, которая возвратит `array.m_array[4]`. Поскольку оператор `[]` использует возврат по ссылке, то он возвращает фактический элемент `array.m_array[4]`. Наше частично обработанное выражение становится `array.m_array[4] = 5`, что является прямой операцией присваивания значения элементу массива.

Любое значение, которое находится слева от оператора присваивания, должно быть `l-value` (переменной с адресом в памяти). Поскольку результат выполнения оператора `[]` может использоваться в левой части операции присваивания (например, `array[4] = 5`), то возвращаемое значение оператора `[]` должно быть `l-value`. Ссылки же всегда являются `l-values`, так как их можно использовать только с переменными, которые имеют адреса памяти. Поэтому, используя возврат по ссылке, компилятор останется доволен, что возвращается `l-value`, и никаких проблем не будет.

Рассмотрим, что произойдет, если оператор `[]` будет использовать возврат по значению вместо возврата по ссылке. `array[4]` приведет к вызову функции перегрузки оператора `[]`, который будет возвращать значение элемента `array.m_array[4]` (не индекс, а значение по указанному индексу). Например, если значением `m_array[4]` является `7`, то выполнение оператора `[]`

приведет к возврату значения `7`. `array[4] = 5` будет обрабатываться как `7 = 5`, что является бессмысленным.

В начало

Перегрузка оператора () и функторы

Перегрузка оператора `()` используется в реализации **функторов** (или «функциональных объектов») – классы, которые работают как функции. *Преимущество функтора над обычной функцией заключается в том, что функторы могут хранить данные в переменных-членах (поскольку они сами являются классами)*. Вот пример использования простого функтора:

```
class Accumulator {
public:
	Accumulator() {}

	int operator() (int i) {
		return (m_counter += i);
	}

private:
	int m_counter = 0;
};

int main()
{
	Accumulator accum;
	std::cout << accum(30) << std::endl; // выведется 30
	std::cout << accum(40) << std::endl; // выведется 70

	return 0;
}
```

Обратите внимание, использование класса `Accumulator` выглядит так же, как и вызов обычной функции, но наш объект класса `Accumulator` может хранить значение, которое увеличивается.

Вы можете спросить: «Зачем использовать класс, если всё можно реализовать и через обычную функцию со статической локальной переменной?». Можно сделать и через `static`, но, поскольку функции представлены только одним глобальным экземпляром (т.е. нельзя создать несколько объектов функции), использовать эту функцию мы можем только для выполнения чего-то одного за раз. С помощью функторов мы можем создать любое количество отдельных функциональных объектов, которые нам нужны, и использовать их одновременно.

Более подробно о функторах мы узнаем в следующих Л/Р.

В начало

Общее задание

В следующем цикле Л/Р по высокоуровневому программированию вам будет предложено реализовать полноценную программу по управлению, каталогизации и обработки информации. Каждая программа будет представлять из себя логически законченную платформу по типу: «Автоматизированная система управления» и содержать в себе следующий функционал:

- Обработка базы данных
  - Создание / удаление / редактирования записей
  - Сортировка и фильтрация записей
- Управление пользователями
  - Создание
  - Удаление
  - Авторизация

Каждая программа будет снабжена интерактивным меню пользователя. В этой Л/Р вам будет предложено реализовать такое меню, а также создать файл с функциями – затычками для тестирования меню. В каждом варианте задания будет описана функциональность программы, которую вы должны реализовать в меню. Продуктивной и интересной работы!

Вариант №1

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система института». Которая будет включать следующий функционал:

- Ведение базы студентов и преподавателей (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы предметов
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект

вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: студент, преподаватель и предмет. Для классов: студент и преподаватель создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (преподаватель, студент, предмет). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (студенты, преподаватели, предметы) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (предмет, студент, преподаватель).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять студента / преподавателя / предмет. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: студент, предмет и преподаватель – пока не должны быть жёстко связаны между собой.

В начало

Вариант №2

Основная задача:

Вам будет предложено написать программу – «Автоматизированная система школы». Которая будет включать следующий функционал:

- Ведение базы школьников и учителей (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы предметов
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: школьник, учитель и предмет. Для классов: школьник и учитель создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (учитель, школьник, предмет). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (школьники, учителя, предметы) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (предмет, учитель, школьник).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять школьника / учителя / предмет. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: школьник, учитель и преподаватель – пока не должны быть жёстко связаны между собой.

В начало

Вариант №3

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система парикмахерской». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы услуг
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, клиент, услуга. Для классов: сотрудник и клиент создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть

поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, клиент, услуга). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудник, клиент, услуга) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (клиент, сотрудник, услуга).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / клиента/услугу. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: сотрудник, клиент и услуга – пока не должны быть жёстко связаны между собой.

В начало

Вариант №4

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система авто – салона». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы автомобилей и услуг
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на

экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, клиент и услуга. Для классов: сотрудник и клиент создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (клиент, сотрудник, услуга). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудники, клиенты, услуги) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (клиент, сотрудник, услуга).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять клиента / сотрудника / услугу. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: клиент, сотрудник, услуга – пока не должны быть жёстко связаны между собой.

В начало

Вариант №5

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система банка». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы счетов
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, клиент, счёт. Для классов: сотрудник и клиент создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, клиент, счёт). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудники, клиенты, счета) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, клиент, счёт).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / клиента / счета. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: сотрудник, клиент, счёт – пока не должны быть жёстко связаны между собой.

В начало

Вариант №6

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система мастерской». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы услуг
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, клиент, услуга. Для классов: сотрудник, клиент создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть

поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, клиент, услуга). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудник, клиент, услуга) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, клиент, услуга).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / клиента / услугу. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: клиент, сотрудник, услуга – пока не должны быть жёстко связаны между собой.

В начало

Вариант №7

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система кинотеатра». Которая будет включать следующий функционал:

- Ведение базы сотрудников и клиентов
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы сеансов, фильмов, залов (разные базы)
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на

экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, клиент, сеанс, фильм и зал. Для классов: сотрудник и клиент создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, клиент, фильм, сеанс, зал). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудник, клиент, сеанс, фильм, зал) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, клиент, сеанс, фильм, зал).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / клиента / сеанс / зал / фильм. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: сотрудник, клиент, зал, сеанс, фильм – пока не должны быть жёстко связаны между собой.

В начало

Вариант №8

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система диалога (чат бот)». Которая будет включать следующий функционал:

- Ведение базы пользователей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы диалогов, тем, интересов и напоминаний
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: пользователь, диалог, тема, интерес, напоминание. Для класса: пользователь создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (пользователь, диалог, тема, интерес, напоминание). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (пользователи, напоминания, темы и т д) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (пользователь, напоминания, темы и т д).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять пользователя / диалог / темы / напоминание / интерес. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Все сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №9

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система поликлиники». Которая будет включать следующий функционал:

- Ведение базы врачей и пациентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы приёмов и записей в амбулаторных картах
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: врач, пациент, приём, карта, запись. Для классов: врач и пациент создайте общий класс

родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (врач, пациент, приём, карта и запись). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (врач, пациент, карта, запись и т д) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (врач, пацент, приём и т д).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять врача/пациента/приём/карты/запись в карту. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Все сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №10

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система почты». Которая будет включать следующий функционал:

- Ведение базы сотрудников клиентов (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы отправлений
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, клиент, письмо. Для классов: сотрудник, клиент создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, клиент, письмо). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудник, клиент, письмо) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, клиент, письмо).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / клиента / письмо. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: сотрудник, клиент и письмо – пока не должны быть жёстко связаны между собой.

В начало

Вариант №11

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система гостиницы». Которая будет включать следующий функционал:

- Ведение базы сотрудников и посетителей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы номеров и услуг (предусмотреть напоминание о выезде)
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, посетитель, номер и услуга. Для классов: сотрудник и посетитель создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, посетитель, услуга и номер). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудники, посетители, номера и услуги) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, посетитель, номер, услуга).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / посетителя / номер / услугу. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №12

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система завода». Которая будет включать следующий функционал:

- Ведение базы сотрудников и поставщиков (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы производства и продаж
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, поставщик, продажа, производство. Для классов: сотрудник и поставщик создайте общий класс родитель – человек и унаследуйтесь от него. У каждой

сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, поставщик, производство и продажа). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудник, поставщик, производство, продажа) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, поставщик, производство, продажа).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / поставщика / производство / продажу. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №13

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система управления музыкальной коллекцией». Которая будет включать следующий функционал:

- Ведение базы исполнителей и пользователей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы треков, жанров (реализовать подборку)
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: пользователь, исполнитель, трек, жанр. Для классов: пользователь и исполнитель создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (пользователь, исполнитель, трек, жанр). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (пользователь, исполнитель, трек, жанр) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (пользователь, исполнитель, трек, жанр).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять пользователя / исполнителя / трек / жанр. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №14

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система электронной почты». Которая будет включать следующий функционал:

- Ведение базы администраторов и пользователей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы отправлений (реализовать возможность отправки и получения писем)
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: администратор, пользователь, письмо. Для классов: администратор и пользователь создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (администратор, пользователь, письмо). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (администраторы, пользователи, письма) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с

отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (администратор, пользователь, письмо).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять администратора / пользователя / письмо. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №15

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система управления фотографиями». Которая будет включать следующий функционал:

- Ведение базы фотографий и альбомов
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы отмеченных людей на фото
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: пользователь, фотография, альбом. Для класса – пользователь, создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (пользователи, фотографии, альбомы). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (пользователи, фотографии, альбомы) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (пользователь, фотография, альбом).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять пользователя / альбом / фото. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: студент, предмет и преподаватель – пока не должны быть жёстко связаны между собой.

В начало

Вариант №16

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система вокзала». Которая будет включать следующий функционал:

- Ведение базы сотрудников, машинистов и пассажиров (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы направлений поездов и самих составов (возможность продажи билетов)
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, пассажир, рейс, поезд, машинист. Для классов: сотрудник, машинист и пассажир создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, машинист, пассажир, рейс, поезд). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудники, пассажиры, машинисты, рейсы, поезда) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, пассажир, машинист, рейс, поезд).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / машиниста / рейс / поезд. Редактировать эти записи, удалять, сортировать по

заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №17

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система магазина». Которая будет включать следующий функционал:

- Ведение базы сотрудников и покупателей (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы товаров и продаж (продажа товара)
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, покупатель, товар, продажа. Для классов: сотрудник, покупатель создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, покупатель, товар, продажа). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудники, покупатели, товары, продажи) и редактирует его (либо сортирует записи, либо

добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, покупатель, товар, продажа).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / покупателя / товар / продажу. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №18

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система космодрома». Которая будет включать следующий функционал:

- Ведение базы сотрудников и астронавтов
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы космических судов и запланированных запусков (реализовать возможность запуска – виртуально!)
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода.

Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, астронавт, судно, запуск. Для классов: сотрудник и астронавт создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, астронавт, запуск и судно). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудники, астронавты, судна и запуски) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, астронавт, судно, запуск).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / астронавта / судно / запуск. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №19

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система библиотеки». Которая будет включать следующий функционал:

- Ведение базы сотрудников и читателей
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы книг, журналов, авторов
- Возможность авторизации

- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, читатель, произведение, автор. Для классов: сотрудник, читатель и произведение создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, читатель, произведение, автор). Для этих функций будет создан интерфейс, который принимает массив нужных объектов (сотрудники, читатели, авторы, произведения) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудники, читатели, произведения, авторы).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / читателя / автора / произведение. Редактировать эти записи, удалять,

сортировать по заданному полю и фильтровать по заданному полю. Сущности программы – пока не должны быть жёстко связаны между собой.

В начало

Вариант №20

Общая задача:

Вам будет предложено написать программу – «Автоматизированная система аэропорта». Которая будет включать следующий функционал:

- Ведение базы сотрудников, пилотов и пассажиров (разные базы)
  - Создание / удаление / редактирование записей
  - Сортировка / фильтрация
- Ведение базы рейсов, самолётов и продаж билетов
- Возможность авторизации
- Создание файлов – отчётов и сохранения состояния

Индивидуальные задания:

Задача 1

Добавьте в класс меню, который вы разрабатывали на прошлой Л/Р перегрузку оператора вывода (cout). Таким образом, при выполнении команды std::cout << menu; - где menu – это объект класса Menu, меню выводилось на экран. Данная перегрузка оператора должна использовать встроенный метод вывода меню на экран.

Задача 2

Добавьте в свой класс меню перегрузку оператора cin, таким образом, чтобы при использовании команды: std::cin >> menu, где menu – это объект вашего класса меню, меню выполняло считывание пользовательского ввода. Данная перегрузка оператора должна использовать встроенный метод считывания пользовательского ввода.

Задача 3

Создайте классы – сущности данных вашей программы: сотрудник, пассажир, пилот, рейс, самолёт, билет. Для классов: сотрудник, пассажир и пилот создайте общий класс родитель – человек и унаследуйтесь от него. У каждой сущности должно быть поле уникального идентификатора – id, которое представлять из себя тип: date, либо unsigned long int, а также дата создания.

Задача 4

В отдельном файле функций – app.cpp (app.h) создайте функции сортировки, фильтрации, удаления, добавления и редактирования сущностей данных (сотрудник, пассажир, пилот, рейс, билет, самолёт). Для этих функций будет создан интерфейс, который принимает массив нужных объектов

(сотрудники, пилоты, пассажиры, рейсы, самолёты, билеты) и редактирует его (либо сортирует записи, либо добавляет новые, либо редактирует нужную, либо удаляет заданную). Для функций фильтрации данных интерфейс будет принимать константный массив объектов и возвращать новый массив с отфильтрованными данными (придумайте, как можно возвращать вместе с этим массивом его размер).

Задача 5

Отдельно создайте функции добавления соответствующих записей (сотрудник, пассажир, пилот, самолёт, билет, рейс).

Задача 6

Соедините созданные функции и модели данных в созданном меню и протестируйте прототип вашей программы. Устраните выявленные ошибки. На данном этапе ваша программа должна уметь: добавлять сотрудника / пассажира / рейс / самолёт / пилота / билет. Редактировать эти записи, удалять, сортировать по заданному полю и фильтровать по заданному полю. Сущности: студент, предмет и преподаватель – пока не должны быть жёстко связаны между собой.

В начало

Контрольные вопросы

1. Что такое оператор в языке C++?
2. Что такое перегрузка операторов?
3. Какие операторы можно перегружать?
4. Как происходит перегрузка операторов?
5. Какие существуют способы перегрузки операторов?
6. Когда, какие способы перегрузки нужно применять?
7. Как выполнить перегрузку операторов ввода / вывода?
8. Как происходит перегрузка унарных операторов?
9. Как выполняется перегрузка операторов сравнения?
10. Перегрузка операторов инкремента и декремента.
11. Чем отличается перегрузка префиксной и постфиксной формы?
12. Как перегрузить оператор индексации?
13. Почему оператор индексации возвращает ссылку, а не значение?
14. Перегрузка оператора «()».
15. Что такое функторы?

Список литературы

1. Курс лекций доцента кафедры ФН1-КФ Пчелинцевой Н.И.
2. Программирование на языке высокого уровня C/C++ [Электронный ресурс]: конспект лекций / – Электрон. текстовые данные. – М.: Московский государственный строительный университет, Ай Пи Эр Медиа, ЭБС АСВ, 2016. – 140 с. – Режим доступа: http://www.iprbookshop.ru/48037.

В начало