Last updated: 2019-03-23
vadimov@i.ua

Практическое занятие 7 (семестр2).

Задача 7.1. Для класса **coord** (См. Unit7 "Overloading An Operator In Which The Order Of The Operands Is Important") перегрузите операторы * и /. Продемонстрируйте их работу.

Задача 7.2. В приведенном ниже примере некорректно перегружен оператор *%*. Почему?

```
coord coord::operator %(const coord obj) {
    double i;
    cout << "Enter a number: ";
    cin >> i;
    cout << "root of " << i << " is ";
    cout << sqrt(i);
}
```

Задача 7.3. Поэкспериментируйте, меняя тип возвращаемого значения оператор-функций на что-нибудь отличное от **coord.** Обратите внимание на генерируемые компилятором сообщения об ошибках. Объясните причины ошибок.

Задача 7.4. Относительно класса **coord** (См. Unit7 "Overloading The Relational And Logical Operators") перегрузите операторы отношения < и >.

Задача 7.5. Перегрузите оператор минус - относительно класса **coord** . Создайте его префиксную и постфиксную формы.

Задача 7.6. Перегрузите оператор + относительно класса **coord** так, чтобы он был как бинарным, так и унарным оператором (См. Unit7 "Overloading A Unary Operator"). При использовании в качестве унарного оператор + должен делать положительным значение любой отрицательной координаты.

Задача 7.7. Перегрузите операторы - и / для класса **coord** посредством дружественных функций.

Задача 7.8. Перепишите класс **coord** так, чтобы можно было использовать объекты типа **coord** для умножения каждой из координат на целое. Должны быть корректными обе следующие инструкции: obj * int и int * obj. Объясните, почему решение требует использования дружественных оператор-функций. Покажите, как с помощью дружественной оператор-функции перегрузить оператор - относительно класса **coord** . Определите как префиксную, так и постфиксную формы.

Задача 7.9. Из материала Unit7 вы должны были уяснить, что с помощью дружественной оператор-функции можно определить разницу между префиксной и постфиксной формами операторов инкремента и декремента точно так же, как это делалось с помощью функций-членов (просто добавляете целый параметр при задании постфиксной версии). Например, здесь приводятся префиксная и постфиксная версии

оператора инкремента относительно класса **coord** :

```
coord operator ++(coord &obj); // prefix
coord operator ++(coord &obj, int notused); // postfix
```

Если оператор ++ находится перед операндом, то вызывается функция `coord operator ++(coord &obj)`. Если оператор ++ находится после операнда, вызывается функция `coord operator ++(coord &obj, int notused)`. В этом случае переменной **notused** будет передано значение 0. Придумайте свой собственный пример (и класс тоже), демонстрирующий все вышесказанное.

Задача 7.10. Пусть дано следующее объявление класса, добавьте все необходимое для создания типа динамический массив. То есть выделите память для массива и сохраните указатель на эту память по адресу ptr. Размер массива в байтах сохраните в переменной **size.** Создайте функцию put(), возвращающую ссылку на заданный элемент массива и функцию get(), возвращающую значение заданного элемента. Обеспечьте контроль границ массива. Кроме этого перегрузите оператор присваивания так, чтобы выделенная каждому массиву такого типа память не была случайно повреждена при присваивании одного массива другому.

```
class dynarray {
    int *ptr;
    int size;
public:
    dynarray(int s); // pass size of array in s
    int &put(int i); // return reference to element i
    int get(int i);  // return value of element i
    // create operator =() function
};
```

Задача 7.11. Сравните реализацию **strtype**, использующую конструктор копий (См. Unit6 "Copy Constructor To Allow Objects To Be Passed To Functions", и реализацию **strtype**, использующую ссылку в качестве параметра и ссылку в качестве возвращаемого значения функции (См. Unit7 "A Closer Look At The Assignment Operator"). Объясните чем одно из решений лучше другого.

*-(Для продвинутых) создайте пример, в котором продемонстрируйте ситуацию, в которой конструктор копий окажется более предпочтительным решением.

Подсказка: Как вы узнали из Unit6, создание конструктора копий - это другой путь решения проблем, описанных в Unit7. Конструктор копий может оказаться не столь эффективным решением, как ссылка в качестве параметра + ссылка в качестве возвращаемого значения функции. Это происходит потому, что использование ссылки исключает затраты ресурсов, связанные с копированием объекта в каждом из двух указанных случаев. В C++ часто имеется несколько способов достижения одной и той же цели. Понимание их преимуществ и недостатков есть часть процесса вашего становления как профессионального программиста C++.

Задача 7.12. В Unit5 "Bounded Array" рассматривался простейший способ реализации безопасного массива, в котором для доступа к элементам массива использовались функции **get()** и **put()**. Перегрузка оператора **[]** позволяет создать такой массив гораздо проще. Для создания безопасного массива реализуйте в функции **operator[]()** контроль границ. Кроме этого, функция **operator[]()** должна возвращать ссылку на индексируемый элемент. В представленном ниже фрагменте сделана попытка добавить контроль границ

массива, что позволяет при нарушении границ генерировать соответствующую ошибку. Подсказка: Вспомните, что безопасный массив - это массив, который инкапсулирован в классе, и при этом класс обеспечивает контроль границ массива. Благодаря перегрузке оператора **[]** , работать с безопасным массивом можно так же, как с обычным.

```
// A safe array example:
...
class arraytype {
    ...
};
// Provide range checking for arraytype .
int &arraytype::operator [](int i) {
    if (...) {
        cout << "Index value of " << i << " is out of bounds.\n";
        ...
    }
    ...
}
int main() {
    arraytype obj;
    // this generates a runtime error because SIZE+100 is out of range:
    obj[SIZE+100] = 99; // error
    return 0;
}
```

Помните! Безопасный массив увеличивает расход ресурсов, что далеко не всегда может оказаться приемлемым. Именно из-за непроизводительного расхода ресурсов в C++ (в отличие от Java) отсутствует встроенный контроль границ массивов. Тем не менее, в тех приложениях, в которых желательно обеспечить контроль целостности границ, реализация безопасного массива может оказаться возможным вариантом решения.

Задача 7.12. Переделайте Example 7.10 (См. Unit7 "A Closer Look At The Assignment Operator") так, чтобы относительно класса **strtype** перегрузить оператор **[]** . Этот оператор должен возвращать символ по заданному индексу. Кроме этого, необходима возможность задавать оператор [] в левой части инструкции присваивания. Покажите, что ваша программа работает.

Задача 7.13. Измените ваше решение Задачи 7.12 так, чтобы использовать оператор **[]** для индексирования динамического массива. То есть замените функции **get()** и **put()** оператором **[]** .

Задача 7.14. Перегрузите операторы сдвига ≫ и ≪ относительно класса **coord** так, чтобы стали возможными следующие типы операций:

```
obj << integer
obj >> integer
```

Удостоверьтесь, что ваши операторы действительно сдвигают значения x и y на заданное количество разрядов.

Задача 7.15. У вас есть класс:

```
class three_d {
    int x, y, z;
public:
    three_d(int i, int j, int k) { x=i; y=j; z=k; }
```

```
    three_d(){ x=0; y=0; z=0; }
    void get(int &i, int &j, int &k) { i=x; j=y; k=z; }
};
```

Перегрузите для этого класса операторы +, -, ++ и --. Для операторов инкремента и декремента перегрузите только префиксную или только постфиксную форму.

Задача 7.16. Измените ваше решение Задачи 7.15 так, чтобы в оператор-функциях вместо параметров-значений использовать параметры-ссылки.
Подсказка: Для операторов инкремента и декремента потребуются дружественные функции.
Перегрузите операторы ==, !=, || и + относительно класса three_d Задачи 7.15 так, чтобы иметь возможность выполнять следующие типы операций:

```
obj + int;
int + obj;
```

Задача 7.17. Создайте свой вариант класса strtype, который допускает следующие типы операций:

- Конкатенацию строк с помощью оператора +
- Присваивание строк с помощью оператора =
- Сравнение строк с помощью операторов <, > и ==

Разрешено пользоваться строками фиксированной длины. На первый взгляд это может показаться непростой задачей, но, немного подумав и поэкспериментировав, вы обязательно справитесь. ВВ. в вас верит! :)