# Перегрузка операций

- Классификация операций языка C++
- Способы перегрузки операций в языке C++
- Заголовки функций для перегрузки операций
- Дружественные функции
- Особые случаи перегрузки операций

# Перегрузка операций

- Перегрузка операций в языке C++ дает программисту возможность строить выражения, в которых операндами являются объекты.
- Перегрузка операций является *синтаксическим сахаром*, делающим код программы более простым.

# Пример использования перегруженных операций

```
//str3 = str1 + str2 без перегрузки операций
char str1[] = "Строка 1";
char str2[] = "Строка 2";
char str3[81];

strcpy(str2, str1);
strcat(str1, str2);

//str3 = str1 + str2 с перегрузкой операций
QString str1 = "Строка 1";
QString str2 = "Строка 2";
QString str3;

str3 = str1 + str2;
```

# Классификация операций

- По количеству операндов
  - унарные (+, -, ++, --)
  - бинарные (+, *, /, [ ], =, <, &&)
  - тернарная операция ? :
- По виду возвращаемого значения
  - возвращают ссылку, следовательно могут стоять слева от операции присваивания (без потери семантики). Например, операции [ ], унарная *
  - возвращают значение (т.е. копию объекта), следовательно не могут стоять слева от операции присваивания. Например, операция +

# Классификация операций

- По воздействию на операнды
  - операции-конструкторы - создают новый объект - арифметические и логические операции, операции сравнения;
  - модификаторы - изменяют один из операндов - операции присваивания, инкремента
  - селекторы - возвращают существующий объект (или его часть) - операции взятия элемента массива, определения значения по адресу, обращения к полю структуры.

# Поведение операций различных типов

- *Операции-конструкторы* не изменяют своих операндов;  возвращают созданный объект как значение.
- *Модификаторы* изменяют один из своих операндов (как правило, первый) и возвращают ссылку на него или его значение.
- *Селекторы* не изменяют своих операндов; возвращают выбранный объект как ссылку.

# Примеры операций-конструкторов, модификаторов и селекторов

```
  const QString operator+ ( const QString &
          s1, const char * s2 )//1
const QString operator+ ( const QString &
s1, const QString & s2 )//2


// Операция-конструктор – порождает новый
// объект (безымянный), объекты-операнды
// (str1 и str2) не изменяются
QString str1("123"), str2("abc");
str1 + "abc";/*1*/ // создается объект
str1 +  str2;/*2*/ // содержащий "123abc"
```

# Примеры операций: конструкторов, модификаторов и селекторов

```
  QString & operator= ( const char * str )

// Операция-модификатор – изменяет левый
// операнд (str)
QString str;
str = “123”;

   const QChar operator[] ( uint position )
                          const
// Операция-селектор – обращается к части
// строки, точнее к одному символу
QString str(“123”);
QChar ch=str[0];//Получаем '1'
```

# Перегрузка операций

- Поскольку при перегрузке операции сохраняют свои приоритет и ассоциативность (а также привычность восприятия программистом), то категорически не рекомендуется изменять их семантику при перегрузке.
- Исключением из этого правила является устоявшееся использование операций **<<** и **>>** (первоначально операции побитового сдвига) как операций ввода/вывода, закрепленное в стандартной библиотеке языка C++.

# Перегрузка операций

- Не перегружаются следующие операции:
  - `.` (доступ к свойству или методу класса),
  - `::` (операция области видимости),
  - `?  :` (операция условного выполнения),
  - `#` (операции препроцессора).

- Помимо обычных операций могут перегружаться операции работы с динамической памятью **`new`** и **`delete`**

# Способы перегрузки операций

- Функция, описывающая перегружаемую операцию, должна иметь имя `operator <операция>`, и следовать определенным правилам.
- Способы перегрузки операций:
  - перегруженная операция является методом класса;
  - перегруженная операция является свободной (глобальной) функцией.
- Заголовок перегруженной операции определяет, для каких типов данных она будет вызываться. По крайней мере один из операндов должен быть объектом класса.

# Перегрузка операции как метода класса

- Если перегруженная операция является членом класса, то ее первый (левый) операнд будет иметь тип этого класса.
- Левым операндом будет объект, для которого вызвана функция.
- Если операция унарная, то она не имеет входных параметров.
- Если операция бинарная, то она имеет один параметр — второй (правый) операнд.

# Задание

- Объявите класс **`Vector`** (вектор в двумерном пространстве), описываемый вещественными числами **`x`** и **`y`** (**`private`**).
- Создайте конструктор с двумя параметрами для инициализации **`x`** и **`y`**.
- Определите в нем перегруженную операцию **`+`** (сложение двух векторов, **`public`**) как метод класса.
- Код этой операции задайте в описании класса.
- В функции main сложите два вектора **`(3,8)`** и **`(4,-1)`**.

# Перегрузка операции как метода класса

```
class Vector
{
private:
   float x, y;

public:
   // Конструктор по умолчанию
   Vector()
   { x = 0; y = 0; }

   // Конструктор с параметрами
   Vector(float _x, float _y)
   { x = _x; y = _y; }
...
```

# Перегрузка операции как метода класса

```
...
  Vector operator+(Vector other)
  {
    Vector result;
    result.x = x + other.x;
    result.y = y + other.y;
    return result;
  }
};

void main(void)
{
  Vector vt1(3, 8), Vector vt(4, -1);
  Vector goal = vt1 + vt2;
}
```

# Перегрузка операции как свободной функции

- Если перегружаемая операция унарная, то она имеет один входной параметр — ее операнд.
- Если перегружаемая операция бинарная, то она имеет два входных параметра — левый и правый операнды соответственно.
- Перегрузка операции как свободной функции является единственной возможностью описать операцию, чей первый операнд не является объектом (имеет стандартный тип языка С).

# Задание

- Для определенного выше класса `Vector` перегрузите операцию умножения вектора на (вещественное) число так, чтобы число могло стоять как первым, так и вторым операндом.
- Перегрузку организовать свободными функциями.
- В функции main умножьте вектор `(3,8)` на `4`.

# Перегрузка операции как свободной функции

```
Vector operator*(Vector vt, int val)
{
  Vector result;
  result.x = vt.x * val;
  result.y= vt.y * val;
  return result;
}

Vector operator*(int val, Vector vt)
{ return result vt * val; }

void main(void)
{ // Vector goal = Vector(3, 8) * 4;
  Vector goal = 4 * Vector(3, 8);
}
```

# Дружественные функции и классы

- Функция, объявленная как дружественная к другому классу, имеет полный доступ к его защищенной (`protected` и `private`) части.
- Дружественные функции часто необходимы для реализации перегруженных операций как свободных функций.
- Если класс `A` объявлен дружественным классу `B`, то все функции `A` являются дружественными классу `B`.
- Отношение дружбы между классами нетранзитивно (т.е. если `A` друг `B` то это еще не означает, что `B` — друг `A`) .

# Дружественные функции и классы

- Функция (класс) объявляется дружественной классу **A** в описании класса **A** (т.е. только сам класс определяет, кто ему друг).
- Дружественная функция задается:

```
friend <прототип функции>
```

- Дружественный класс задается:

```
friend class <имя класса>;
```

- При описании функции как дружественной она считается объявленной, т.е. ее прототип отдельно описывать не нужно; при этом дружественная функция не является членом класса.

# Задание

Объявите перегруженные выше операции умножения вектора на число дружественными классу `Vector` для получения доступа к его переменным.

# Дружественные функции

```
class Vector
{
private:
   float x,y;
   ...
friend Vector operator*(Vector vt,int val);
friend Vector operator*(int val,Vector vt);
};
```

# Перегрузка модификаторов и селекторов

- Если операция изменяет свой операнд и он передается как параметр, то операнд должен передаваться по ссылке.
- Если результатом операции должно быть значение,  которое может стоять слева от оператора присваивания, то возвращаемое значение должно быть ссылкой.
- Если требуется вернуть ссылку на текущий объект (или его значение), то можно воспользоваться выражением `*this`

# Задание

- Перегрузите для класса **`Vector`** следующие операции:
- присваивания (один вектор присваивается другому)
- взятия элемента массива (должен возвращать ссылку на компонент вектора). В этом случае индексом должен являться символ:
  - **`'x'`** — вернуть значение координаты **`x`**;
  - **`'y'`** — вернуть значение координаты **`y`**;
  - иное — вернуть **`x`**.
- Перегрузку операций осуществлять методами класса.

# Перегрузка модификаторов и селекторов

```
class Vector
{
   ...
public:

   Vector & operator=(Vector other);

   /*Координатам можно присваивать значения,
     пользуясь []*/
   float & operator[](char coord);
};
```

# Перегрузка модификаторов и селекторов

```
Vector & Vector::operator=(Vector other)
{
   x = other.x; y = other.y;
   return *this;
}

float & Vector::operator[](char coord)
{
   switch(coord)
   {
      case 'x':   return x;   break;
      case 'y':   return y;   break;
      default:    return x;
   }
}
```

# Задание

- Сложите два вектора `(3,8)` и `(4,-1)` и распечатайте `x`-составляющую суммы. Переменную для хранения суммы не создавать.
- Создайте массив из трех векторов. Присвойте в одном выражении 3-м векторам одно значение `(1,2)`.

# Использование модификаторов и селекторов

```
void main(void)
{
  Vector vt1(3, 8), vt2(4, -1);
  printf("%f", (vt1 + vt2)['x']);

  Vector mass[3];
  mass[0] = mass[1] = mass[2] = Vector(1,2);
}
```

# Особые случаи перегрузки операций

- Операции `++` и `--` имеют две формы записи: префиксную и постфиксную
  - префиксная форма описывается как обычно, при перегрузке постфиксной необходимо добавить к функции дополнительный параметр типа **`int`**, значение его не используется.
- Операция присваивания, а также **`[]`**, **`->`**, **`new`** и **`delete`** должны описываться как методы класса, эти операции не могут быть свободными функциями.
- Операции **`new`** и **`delete`** описываются статическими методами класса.