Полиморфизм

Общие идеи

- Использование общего кода для разных типов
- Унифицированный вид кода (одни и те же операторы для разных случаев)
- Основные языковые инструменты:
  - приведение типов
  - указатели на функции
  - перегрузка функций
  - шаблоны

Как это было в чистом C

- Приведение типов
  - 10 это int, short или long?
  - (char*)->(void*)->(char*)
  - http://cppstudio.com/post/310/
- Указатели на функции.
  - Функции тоже могут быть аргументами других функций.
  - Можно писать применяющий функцию ко всему списку/дереву/массиву

Перегрузка функций(overloading)

- Поддержка нескольких функций с одинаковым именем
  - Как отличать? Разные аргументы. При компиляции к именам функций добавляются аргументы и длина имени функции foo(int a,float b) -> _Z3fooif
  - Компилятор сам выбирает. что подставить, по аргументам

- Не путайте с переопределением функций(overriding) - это будет в наследовании

Пример. Перегрузка функций

```
int sum(int a, int b)           {return a+b;}
float sum(float a,float b)      {return a+b;}
float sum(float a,int b)        {return sum(a,(float)b);}
float sum(int a, int b); //нельзя, поскольку sum с двумя int-аргументами уже был
```

Перегрузка методов(overloading)

- Поддержка нескольких методов с одинаковым именем
  - Методы, в зависимости от аргументов, могут быть устроены по-разному
  - Мы уже видели это для конструкторов

- Не путайте с переопределением методов(overriding) - это будет в наследовании

Пример. Перегрузка методов

```
Class Matrix{
        double** a;
public:
...
        Matrix Multiply(double b);   // умножение на константу
        Matrix Multiply(Vector b);   // умножение на вектор(в матем. смысле)
        Matrix Multiply(Matrix b);   // матричное умножение
};
```

Перегрузка функций в классе

- Мы можем для своего класса перегрузить функции
- Можно перегрузить функцию, имеющую доступ к private-полям класса - дружественную
  - Пример - функция swap меняет местами объекты и почти всегда может быть перегружена:
    ```
    void swap(MyClass& A, MyClass& B);
    ```

Перегрузка операторов

- Операторы транслируются компилятором в функции/методы и поэтому тоже могут перегружены
- … почти все. Не могут быть - <.*>,<.>,<?:> и <::>
- Мы уже видели это для оператора присвоения(=)
- При перегрузке сохраняется арность оператора (не впихнуть в + три аргумента)
- Общий вид типов для перегрузки есть в стандарте
  - Поскольку из объекта виден только интерфейс класса, обычно довольно строго смотрят на константность аргументов
- Переопределение операторов для стандартных типов - тоже невозможно
  - можно создать свой тип-синоним: typedef int MyType, и с ним делать, что нужно.

Перегрузка операторов

- Перегрузка оператора в классе может быть:
  - Внешней - функция
  - Внутренней - метод
- Внешняя перегрузка обычно используется для бинарных операторов
- Внутренняя - для унарных

Перегрузка сравнений

- Обычно определяется минимальный набор, а остальные операторы через определенные, например == и > - основа

```
bool operator==(MyType a, MyType b){/*определяем*/}
bool operator>(MyType a, MyType b){/*определяем*/}
bool operator>=(MyType a, MyType b){return (a==b)||(a>b);}
bool operator!=(MyType a, MyType b){return !(a==b);}
bool operator<(MyType a, MyType b){return (!(a==b) || !(a>b));}
...
```

Инкремент/декремент

- ```
  MyType& operator++();          // prefix - ++a
  MyType& operator++(int a);     // postfix   - a++
  ```
- Аргумент в постфиксной форме может быть использован(но не стоит):
  `i.operator++( 25 );`
- Аргумент нужен только для различий имен функций при трансляции
  ```

Перегрузка арифметики

- Арифметика используется с минимизацией кода
  - +=, -=, *=,etc - определяется, как методы класса
  - +,-*, etc - определяется, как функции класса(зачастую дружественные), через вызов +=...

```
friend MyType operator+(MyType const a, MyType const b){
        return a+=b; // не забываем про конструктор копий
}
```

  - https://habrahabr.ru/post/132014/

Перегрузка специальных операторов

- Операторы "`a->`", "`[]`", "`()`", "`=`" и "`(type)`" можно переопределить только как методы класса.
- class vector{

  …

  ```
  int operator [](size_t i) const {return array[i];}    // получаем значение
  int& operator [](size_t i) {return array[i];}         // получаем ссылку
  ```

  };

  …

  ```
  a[i]=0;                  //какой метод
  std::cout<<a[i];         //где будет
  b=a[i];                  //вызван?
  ```
- Для перегрузки <<,>> (вывод) - необходимо подключить iostream

Ввод/вывод

- Для перегрузки <<,>> (вывод) - необходимо пордключить iostream

```
#include <iostream>
...
std::istream & operator >>(std::istream & is, MyClass& n)
{
        ...
}
std::ostream & operator <<(std::ostream & os, MyClass const & n)
{
        ...
}
```