Основные принципы ООП :

- *инкапсуляция,*
- *полиморфизм,*
- *наследование.*

Перегрузка функций

Под перегрузкой функции понимается, определение нескольких функций (две или больше) с одинаковым именем, но различными параметрами. Наборы параметров перегруженных функций могут отличаться порядком следования, количеством, типом. Таким образом перегрузка функций нужна для того, чтобы избежать дублирования имён функций, выполняющих сходные действия, но с различной программной логикой.

Приведем пример:
Функция areaRectangle() расчитывает площадь прямоугольника

1. Предположим, что длина и ширина прямоугольника задана в сантиметрах

```
float areaRectangle(float, float) //функция, вычисляющая площадь прямоугольника с двумя параметрами a(см) и b(см)
{
 return a * b; // умножаем длинны сторон прямоугольника и возвращаем полученное произведение
}
```

2. Предположим, что длина и ширина прямоугольника задана в метрах и сантиметрах

```
float areaRectangle(float a_m, float a_sm, float b_m, float b_sm) // функция, вычисляющая площадь прямоугольника с 4-мя параметрами a(м) a(см); b(м) b(см)
{
  return (a_m * 100 + a_sm) * (b_m * 100 + b_sm);
}
```

Компилятор самостоятельно выберет нужную функцию, анализируя только лишь сигнатуры перегруженных функций. Минуя перегрузку функций, можно было бы просто объявить функцию с другим именем, но если таких функций надо больше, чем две, например 10. И для каждой нужно придумать осмысленное имя, а сложнее всего их запомнить. Вот именно поэтому проще и лучше перегружать функции, если конечно в этом есть необходимость.

Исходный код программы

```
#include <iostream>
using namespace std;
// прототипы перегруженных функций
float areaRectangle(float a, float b);
float areaRectangle(float a_m, float a_sm, float b_m, float b_sm);

int main()
{
	cout << "S1 = " << areaRectangle(32,43) << endl; // вызов перегруженной функции 1
	cout << "S2 = " << areaRectangle(4, 43, 2, 12) << endl; // вызов перегруженной функции 2
	return 0;
}

// перегруженная функция 1
float areaRectangle(float a, float b) //функция, вычисляющая площадь прямоугольника с двумя параметрами a(см) и b(см)
{
	return a * b; // умножаем длинны сторон прямоугольника и возвращаем полученное произведение
}

// перегруженная функция 2
float areaRectangle(float a_m, float a_sm, float b_m, float b_sm) // функция, вычисляющая площадь прямоугольника с 4-мя параметрами a(м) a(см); b(м) b(см)
{
	return (a_m * 100 + a_sm) * (b_m * 100 + b_sm);
}
```

Перегрузка методов класса

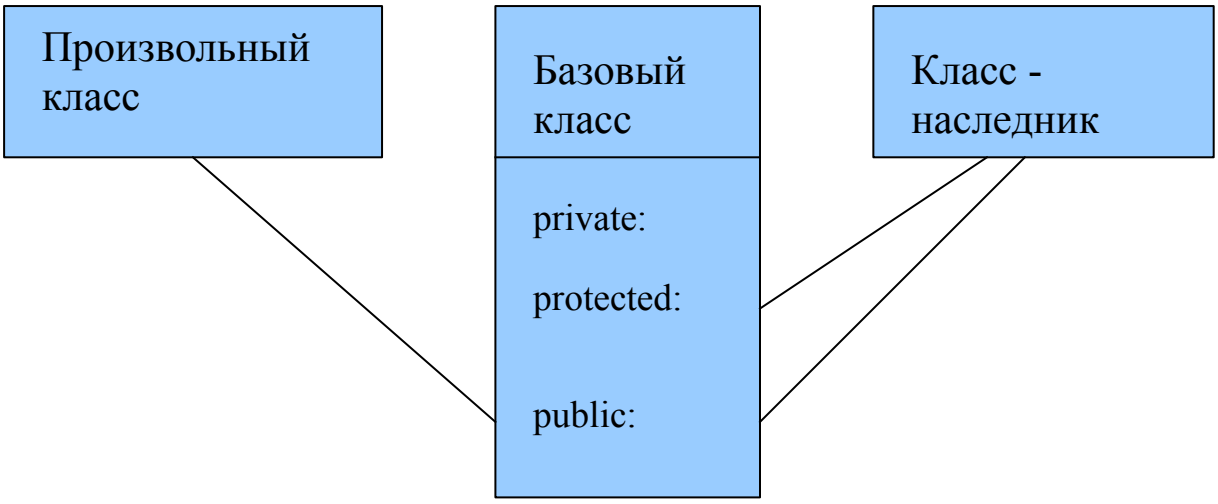
Пример – класс «Графический редактор»

```
// Объявление класса
class Draw
{
    char* message; // private по умолчанию
public:
    Draw ();
```

```
    ~Draw ();
    void paint (); // Рисует текст
    void paint (int, int,int,int); // Рисует прямоугольник
    void paint (int, int, int);  // Рисует окружность
} ;


// Реализация методов


Draw::Draw ()
{
message = new char [10];
strcpy (message, "text");
}
Draw::~Draw ()
{
delete[ ] message;
// message = NULL; - разрушается весь объект!
}

void Draw::paint ()
{
std::cout << message << std::endl;
}
void Draw::paint(int cx1, int cy1, int cx2, int cy2)
{
    std::cout<<"Рисуем прямоугольник" <<std::endl;
}
void Draw::paint (int cx1, int cy1, int r)
{
    std::cout << "Рисуем окружность" << std::endl;
}

// Функция main ()
int main ()
{
 Draw draw;
 draw.paint (); // Рисует текст
 draw.paint (1,1,10,15); // Прямоугольник
 draw.paint (5,5,10); // Окружность
return 0;
}
```

Полиморфизм (polymorphism) (от греческого polymorphos) - это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий.

Наследование



Создание класса-наследника в C++

```
//базовыйкласс
class A
{
...
};
```

```
        //класс - наследник
class B : public A
{
...
};
```

Пример базового класса

```
class Person
{
protected:
      char name[30];
public:
      Person ();
      void setName (char*);
      void show();
};



class Student: public Person
{
      char dept[30];
public:
      Student (char*, char*);
      void setDept (char* );
      void show ();
};




Person::Person()
{
      strcpy (name, "Noname");
}
void Person::setName (char* n)
{
      strcpy (name, n);
}
void Person::show ()
{
      std::cout << "My name is " << name << std::endl;
```

```
}


Student::Student (char* n, char* d)
{
	strcpy (name, n);
	strcpy (dept, d);
}
void Student::setDept (char* d)
{
	strcpy (dept, d);
}
void Student::show ()
{
	std::cout << name << " " << dept << std::endl;
}



int main ()
{
  Person a;
  a.setName ("Tom");
  Student b("Ann", "MTS");
  a.show ();
  b.show ();
  Person* pperson = &b;
  Student* pstudent = &b;
  pperson ->show ();
  pstudent ->show ();
  b.setName ("Kate");
  b.setDept ("GF");
  pstudent ->show ();
  return  0;
}
```

На экране:

```
My name is Tom
Ann MTS
My name is Ann
Ann MTS
Kate GF
```