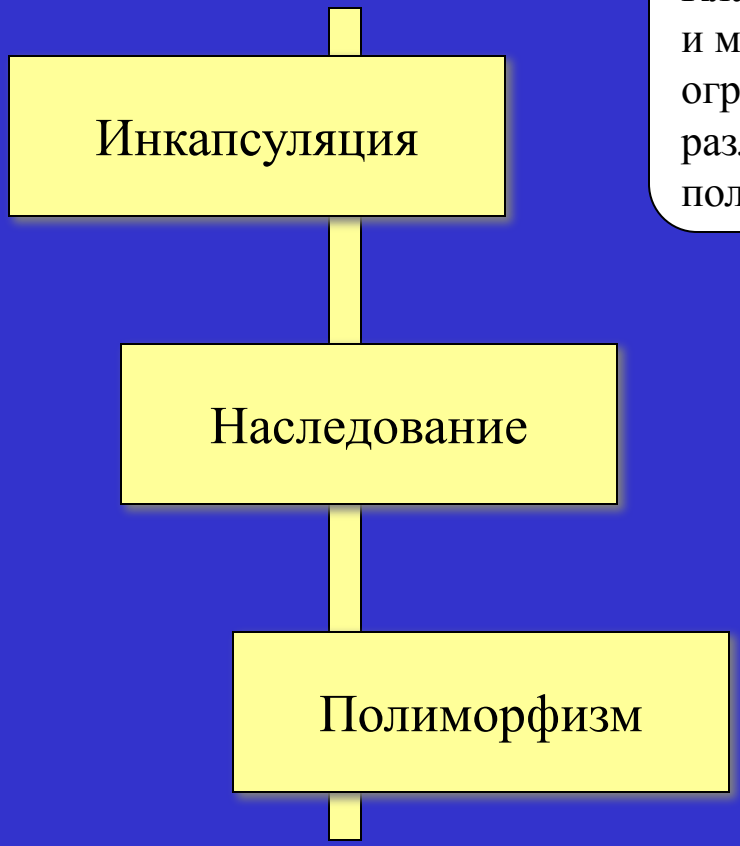
Тема 4. Объектно-ориентированное программирование. Классы

Объектно-ориентированное программирование



Классы объединяют данные и методы их обработки, ограничивая доступ для различных категорий пользователей

Создание новых классов на основе существующих с дополнением или изменением их функциональности

Выполнение разных действий одноименными методами различных классов

Цель - повышение скорости разработки и качества программ за счет лучшей структуризации и повторного использования кода

Объектно-ориентированное программирование и C++

Бьерн Страуструп

Язык C++ был разработан Бьерном Страуструпом в 1985 как расширение языка C.

Одним из основных принципов, лежащих в основе C++, является его практически полная совместимость с C, которая существенно облегчила переход программистов, работавших на языке C, к объектно-ориентированному программированию и гарантировала применимость разработанных на C программ.

Языки объектно-ориентированного программирования

C#, C++, Java, Delphi, Eiffel, Simula, D, Io, Objective-C, Object Pascal, VB.NET, Visual DataFlex, Perl, Php, PowerBuilder, Python, Scala, ActionScript, JavaScript, JScript.NET, Ruby, Smalltalk, Ada, Xbase++, X++, Vala

Отличительные особенности C и C++

C	C++
• Небольшое число элементов языка • Высокая скорость выполнения программ • Поддержка модульного программирования • Хорошая мобильность наряду с работой на "нижнем уровне"	• Простота и надежность использования • Возможность повторного использования программного кода • Хорошая скорость • Ясность и читабельность

Эквивалентные понятия C и C++

C	C++
Тип данных	Класс
Функция	Метод
Переменная	Объект

Инкапсуляция

Класс

Закрытая часть

```
if(parm->parm.getqinfoparm.gq_mask&IIAPI_GQ_ROW_COUNT)
   putrowcount = parm->parm.getqinfoparm.gq_rowCount;

if(repeat and !query_hdl)
{
   query_hdl = (unsigned int)parm->parm.getqinfoparm.gq_repeatQueryHandle;

   if(RepeatQuery* r = FIND(repeatquery, query_id, query_id))
      r->query_hdl = query_hdl;
   else
   {
      repeatquery.New();
      repeatquery.tmp.query_id  = query_id;
      repeatquery.tmp.query_hdl = query_hdl;
      repeatquery.Modify();
   }

   StmtClose();
   Query(NULL);
   return(true);
}
```

```
if(type == INPPAR)
{
   if(itype == IIAPI_DTE_TYPE)
   {
      if(data->dtype() == _TTime)
         DateUnixToIng(inp->date[inp->cnt], (unsigned long)*(DTTime*)data);

      if(data->dtype() == _TDate)
         DTDateToIng(inp->date[inp->cnt], *(DTDate*)data);
   }

   if(itype == IIAPI_MNY_TYPE)
   {
      double d = *(DTMoney*)data;
      long i = (long)d;

      if(d-(double)i > 0.5) i++;

      *(DTMoney*)data = (double)i;
   }

   if(itype == IIAPI_CHA_TYPE and repeat) itype = IIAPI_VCH_TYPE;

   DefPar(INPPAR, inp, inp->cnt++, itype, *data);
```

Интерфейс

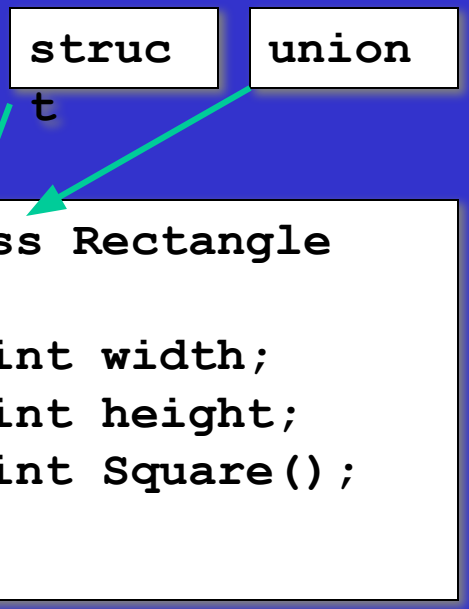
Программа

Понятие класса в C++

Классы служат для объединения данных и методов работы с ними в новые типы данных.

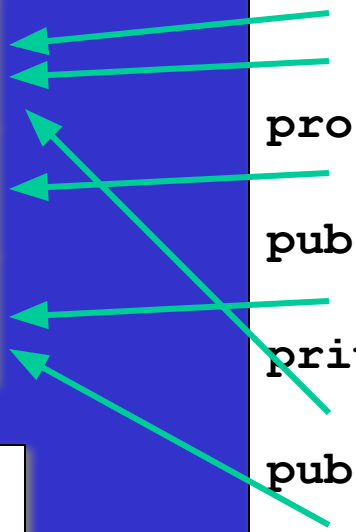
Классы могут предоставлять различные права доступа к данным и методам.

Возможно создание иерархии классов посредством наследования.



Ограничение доступа к элементам класса

Класс	
Данные + методы	**Private**
Данные + методы	**Protected**
Данные + методы	**Public**



```
class Example
{
    int a;
    int b;
protected:
    int c;
public:
    int d;
private:
    int e;
public:
    int f;
};
```

Private *- могут использоваться только методами данного класса или дружественными функциями.*

Protected *- могут использоваться методами данного и наследуемых классов.*

Public *- могут использоваться любыми функциями программы.*

Определение методов класса

В рамках класса (inline)

```
class Rectangle
{
private:
   int   w;
   int   h;
public:
   int   Square()
         {
            return(w*h);
         }
};
```

Вне класса

```
class Rectangle
{
private:
   int   w;
   int   h;
public:
   int   Square();
};

int Rectangle::Square()
{
   return(w*h);
}
```

Доступ к элементам класса

```
class Rectangle
{
public:
    int    w;
    int    h;
    int    Square();
};
```

Через объект

```
Rectangle rect;

rect.w = 200;
rect.h = 100;

int s = rect.Square();
```

Через указатель на объект

```
Rectangle rect;
Rectangle *r = &rect;

r->w = 200;
r->h = 100;

int s = r->Square();
```

Пример ограничения доступа к элементам класса

```
class Rectangle
{
private:
    int   w;
    int   h;
public:
    int   Square();
};

void main()
{
    Rectangle r;
    r.w = 100;
    r.h = 50;
}
```

```
[C++ Error] test.cpp(50): E2247 'Rectangle::w' is not
accessible
[C++ Error] test.cpp(51): E2247 'Rectangle::h' is not
accessible
```

Специальный метод класса - конструктор

```
class Rectangle
{
private:
    int   w;
    int   h;
public:
    Rectangle(int Width, int Height);

    int   Square();
};

Rectangle::Rectangle(int Width, int Height)
{
    w = Width;
    h = Height;
}
```

Конструктор

Свойства конструктора

1. Имя конструктора совпадает с именем класса.

2. Конструктор не возвращает никакого значения.

3. Для класса без конструктора генерируется конструктор по умолчанию.

4. Конструкторы могут быть перегружены.

5. Конструкторы не наследуются.

6. Конструкторы не могут вызываться явно из программы.

Перегрузка конструктора

```
class Rectangle
{
private:
    int   w;
    int   h;
public:
    Rectangle(int Width, int Height);
    Rectangle(int Side);

    int   Square();
};
```

Перегруженный конструктор

```
Rectangle::Rectangle(int Side)
{
    w = Side;
    h = Side;
}
```

```
Rectangle::Rectangle(int
Side)
{
    Rectangle(Side, Side);
}
```

Конструктор копирования

```
class Rectangle
{
private:
    int   w;
    int   h;
public:
    Rectangle(int Width, int Height);
    Rectangle(int Side);
    Rectangle(Rectangle &R);

    int   Square();
};

Rectangle::Rectangle(Rectangle &R)
{
    w = R.w;
    h = R.h;
}
```

Конструктор копирования

Создание объектов класса

Статическое

```
void func()
{
   Rectangle r1(200, 100), r2(150), r3(r1);
   ...
}
```

Динамическое

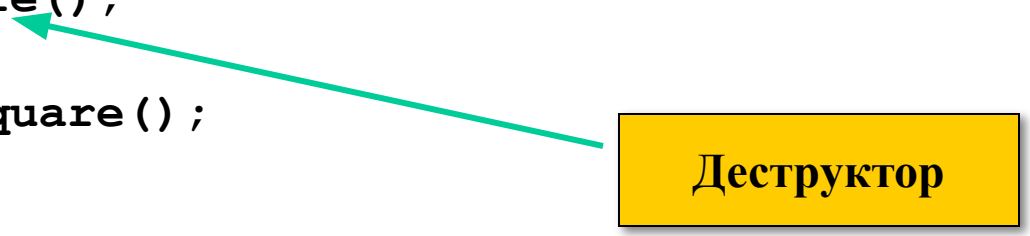
```
void func()
{
   Rectangle *r1 = new Rectangle(200, 100);
   Rectangle *r2 = new Rectangle(150);
   Rectangle *r3 = new Rectangle(*r1);
}
```

Специальный метод класса - деструктор

```
class Rectangle
{
private:
    int    w;
    int    h;
public:
    Rectangle(int Width, int Height);
   ~Rectangle();

    int    Square();
};

Rectangle::~Rectangle()
{
    ShowMessage("Прямоугольник удален.");
}
```



Свойства деструктора

1. Имя деструктора совпадает с именем класса и предваряется тильдой.
2. Деструктор не возвращает никакого значения.
3. Для класса без деструктора генерируется деструктор по умолчанию.
4. Деструкторы не наследуются.
5. Деструкторам не могут передаваться аргументы.
6. Деструкторы могут описываться как виртуальные (virtual)
7. Деструкторы могут вызываться явно.

Разрушение объектов класса

Статическое

```
void func()
{
   Rectangle rect(200, 100);
   ...
}
```

Динамическое

```
void func()
{
   Rectangle *rect = new Rectangle(200, 100);
   ...
   delete rect;
}
```

Работа с памятью в конструкторах и деструкторах

```
class Array
{
private:
   int *data;
public:
   Array(int Number);
  ~Array();
};

Array::Array(int Number)
{
   data = new int[Number];
}

Array::~Array()
{
   delete [] data;
}
```

Статические элементы класса

```
class Rectangle
{
private:
   int   w;
   int   h;
   static int count;
public:
   Rectangle(int Width, int Height);
   Rectangle(int Side);
   Rectangle(Rectangle &R);
  ~Rectangle();

   int   Square();
   static int Count() { return(count); }
};
```

Статические данные

Статический метод

Использование статических элементов в методах класса

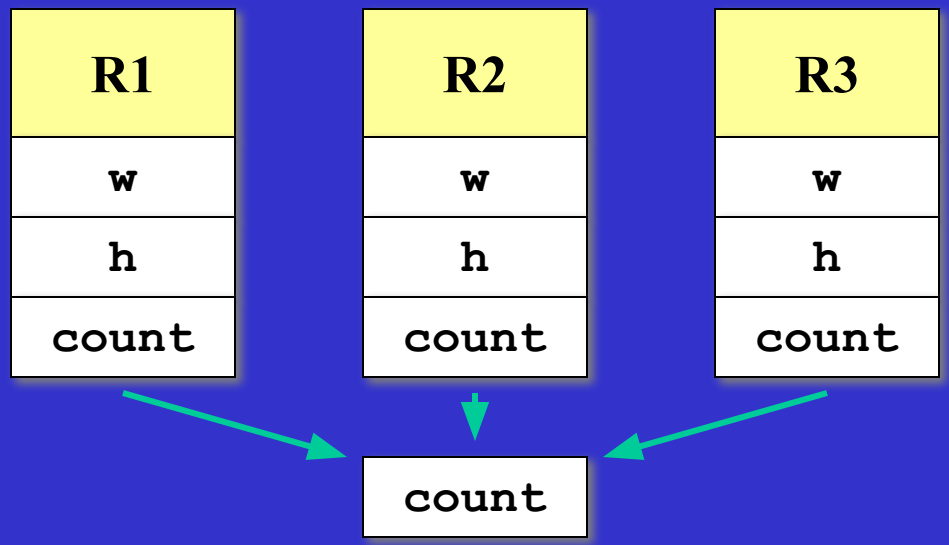
```
int Rectangle::count = 0;

Rectangle::Rectangle(int Width, int Height)
{
   int w = Width;
   int h = Height;
   count++;
}

Rectangle::~Rectangle()
{
   count--;
}
```

Использование статических элементов класса

```
void main()
{
    Rectangle R1(200, 100);
    Rectangle R2(150, 120);
    Rectangle R3(160);
    int c = Rectangle::Count();
    ShowMessage(c);
}
```



Указатель на текущий объект класса

*Компилятор предоставляет каждому объекту указатель на него самого - **this**.*

Этот указатель может быть использован только внутри методов класса для получения адреса текущего объекта.

```
class Rectangle
{
private:
   int   w;
   int   h;
public:
   Rectangle& Zoom(float Rate);
};

Rectangle& Rectangle::Zoom(float Rate)
{
   w = (float)w*Rate;
   h = (float)h*Rate;
   return(*this);
}

Rectangle r(300, 200);
int s = r.Zoom(0.25).Square();
```

Указатель на текущий объект класса

```
void main()
{
   Rectangle R1(200, 100);
   Rectangle R2(150, 120);
   Rectangle R3(160);
   R1.Zoom(2);
   R2.Zoom(5);
   R3.Zoom(0.5);
}
```