Тема 1. Введение в C++ (продолжение)

Определение операций (перегрузка операций) может реализовываться либо дружественными функциями, либо в некоторых случаях как методы класса. В последнем случае первым аргументом операции является класс, от имени которого вызывается данный метод.

Операция может быть вызвана не только записью ее знака вместе с аргументами, но так же как и функция класса (например `a+b` или `a.operator+(b)`).

Аргументы и возвращаемые значения целесообразно задавать как (константные) ссылки с тем, чтобы лишний раз не задействовать конструктор копирования. При этом надо понимать, что возвращать ссылку можно только на объект, существующий вне контекста данной функции, поскольку локальные объекты (переменные) функции перестают существовать после завершения функции, и ссылка на них уже не будет иметь смысла.

Более подробную информацию о перегрузке операций и прочих рассмотренных вопросах можно получить из справочной литературы или поиском в сети.

Как упражнение по данной лекции предлагается доработать класс Number, внеся в него другие операции и методы, чтобы получить полную переносимость кода с типом double. Кроме этого можно разработать классы Vector или Matrix, с которыми можно работать “по формулам” аналитической геометрии или алгебры.

Нововведения и отличия от C

Язык C++ развивается, есть несколько принципиальных версий. Одна из наиболее “революционных” — C++ 11, но нам надо до нее дорасти. Поэтому пока в рамках “классических” версий.

Перечислим основные отличия и нововведения в языке C++ по сравнению с C.

- объявление объектов в любом месте кода
- ссылочный тип данных — создание “синонима” имени объекта обращение по имени, но работа как по указателю
- понятие класса и все с этим связанное
  - private и public доступ к членам класса и методам
  - конструкторы и деструкторы
  - перегрузка операций
  - указатель this
  - const методы
  - статические члены класса
  - наследование и виртуальные функции [пока не рассматриваем]
- struct есть `class {public: ... };`
- полиморфизм функций — функции могут имет одинаковые имена, но должны иметь разные сигнатуры
- значения параметров по умолчанию `int fun(double a = 3.14, int b = 0);`
- пространства имен namespace
- управление памятью `new delete` (“вызовы” конструктора и деструктора)

- строгая типизация указателей, различные способы преобразования указателей static_cast, dynamic_cast, const_cast, reinterpret_cast
- исключения, try-catch блоки
- механизм шаблонов template
- потоковый ввод-вывод, прегрузка операторов ввода-вывода для классов

Некоторые понятия рассматривались на предыдущей лекции. Некоторые мы здесь кратко поясним, некоторые оставим то тех пор, пока они не будут реально нужны.

Ссылочный тип данных.

```
int x;
int &rx = x;      rx - синоним для x

void fval(int a)        void fptr(int *p)     void fref(int &r)
{                       {                     {
   a = 1;                  *p = 2;               r = 3;
}                       }                     }


int s = 0;
fval(s);          // s есть 0
fptr(&s);         // s есть 2
fref(s);          // s есть 3

void fcref(const int &r)
{
   int b = r;  // можно
   r = 3;      // нельзя
}
```

Классы и все такое ...

private и public доступ ...

friend — предоставление доступа к private для отдельных функций или классов.

Еще есть protected — при работе с порожденными классами (наследование) — отложим до обсуждения понятия наследования

Конструкторы и деструкторы.

Конструкторов может быть много, деструктор один.

ClassName(const ClassName &a); — конструктор копирования (вызывается при передаче класса как параметра или при возврате через return)

ClassName() = default; — компилятор сделает по умолчанию как сочтет нужным

ClassName(...) : x(...), y(...) {...} — вызов конструкторов для объектов x, y данного класса

Конструктор можно вызвать явно для “временного” использования класса (как параметра, как аргумента операции и т.п.)

Перегрузка операций.

operator... — специальное имя функции (внешней или члена класса). Есть правила, что можно перегружать, что нельзя. Сохраняется приоритет и ассоциативность операции. Для функции члена класса сам класс — это первый аргумент операции. Для внешней функции ее аргументы и есть аргументы операции (один аргумент обязательно данный класс (или ссылка)). "Присваивающие операции" перегружаются как члены класса, другие как внешние функции.

Примеры перегрузки.

Перегрузка ++ и - -.

operator тип() — преобразование в указанный тип

Много "тонких" моментов при перегрузке некоторых операций (не останавливаемся на этом).

Статические члены класса.

static

static int w; — устанавливает зону видимости в рамках файла

{ static int v; int d; ... } — сохранение и первичная инициализация в блоке

для классов — общий элемент для всех экземпляров класса

```
class A                              A  a, b, c;
{                                    a.x = 1;                a.fun();
 public:                             b.x = 2;                A::fun();
   int x;                            c.x = 3;
   static int y;                     A::y = 6;
   static void fun() { y = 5; }
};                                   b.y = 7;
                                     a.y = 7;
```

```
int A::y;  - по сути обычная переменная, но только "привязанная" к классу,
должна быть определена вне объявления класса.
```

Указатель this.

Указатель на класс, от которого вызывается данный метод.

```
struct A
{                                                    A  a(1), b(2), c(3);
   int x;
   A(int xx) { x = xx;}                              ++a;
   // A(int xx) { this->x = xx;}                     int r = ++b;
   // A(int x)  { this->x = x;}                      int r = (b.operator++()).int();
   operator int() { return x;}                       int r = int(++b);
   const A & operator++() { ++x; return *this; }     c.ObjectPtr()  - это  &c
   const A * ObjectPtr() { return this; }            b.ObjectPtr()  - это  &b
};
```

const методы и вообще о константных объектах.

```
double a;
int i, j;
char str[20];
```

```
#define  PI  3.1415926535
const double pi = 3.1415926535;
const double ccc = f(.......);
         a = pi;  // можно
         pi = 2;  // нельзя!
         a = PI;

const int *p;     // или    int const *p;
         p = &i;  // можно
         j = *p;  // можно
         *p = j;  // нельзя!

int * const p = &i;
         p = &j;  // нельзя!
         j = *p;  // можно
        *p = j;   //  можно

const char * s = "the text";
         printf("%s\n", s);  // можно
         s[4] = '_';            // нельзя!
         s = str;               // можно

const char * const s = "the text";   // только читать!
```

```
struct A
{
   int x;
   A(int xx) { x = xx; }
   int Value() const { return x; }
   int & lValue() { return x; }
   const int & lValue() const { return x; }
};
```

```
void fun(const A & r)
{
   int z = r.x;
   r.x = 3;
   z = r.Value();
   z = r.lValue();
   r.lValue() = 25;
}
```

Полиморфизм функций и параметры по умолчанию

```
   int fun();
   int fun(int x);
   int fun(double x);
   int fun(int x, double y);
   int fun(double x, double y);

   double fun(int x);   !!!!!!!!!

   int fun(double x = 1, double y = 2);

       fun(x, y);
```

```
fun(3.0);

fun(1.0, 2);  ???
```

```
fun(x);       fun(x, 2.0);
fun();        fun(1.0, 2.0);

void print(FILE * f = stdout);
print();
print(my_file_ptr);
```

Записывается при объявлении (прототипе) функции. При отдельном определении функции писать уже не надо.

Пространства имен namespace

```
namespace ABC                    ABC::r = 1;         using namespace ABC;
{                                                    r = 1;
    int r;
    ... прочий код ...
}

double r;
r = 1;
ABC::r = 1;

using namespace std;
```

Управление памятью new delete

("вызовы" конструктора и деструктора)

```
void * malloc(size_t num_of_bytes);
void   free(void * ptr);

new <конструктор>                                    Number *p = new Number(1);
new <конструктор без параметров> [количество]        Number *q = new Number [128];
delete указатель                                     delete p;
delete [] указатель                                  delete [] q;

при отказе - исключение std::bad_alloc
```

Строгая типизация указателей, различные способы преобразования указателей.

C-style — (тип)объект или в C++ можно тип(объект).

static_cast, dynamic_cast, const_cast, reinterpret_cast — преобразования типов в разных ситуациях — обсудим чуть позже отдельно.

Исключения, try-catch блоки.

Сигнал, "распространяющийся" вверх по последовательности вызовов функций и, возможно, несущий с собой некоторое значение (экземпляр класса).

Исключение вызывается (throw) явно в коде (своем или библиотечном) или операционной системой в процессе выполнения (run time exception).

Этот сигнал можно перехватить try-catch блоком.

```
if (возникла ошибка) throw  25;
if (другая ошибка) throw  26;
```

```
try {
  .....
} catch (int x) {
   if (x==25) ....
   if (x==26) ....
} catch (double a) {
}
```

```
int main()
{
  try
  {
     ......
  } catch(...) {
  {
   }
```

```
class MyException {....};
if (....) throw new MyException(....);
```

```
try {
      .....
} catch (MyException * p) {
    cout << p->Code() << std::endl;
    cout <<  p->Message();
    delete p;
    exit(-1);
}
```

```
int F(.....);
errcode = F(.....);
switch(errcode) {
   case ....
}
```

```
main ->  f1  ->  f2  ->  f3  -> g -> h
          catch                    throw
```

Механизм шаблонов template.

```
template<typename T>

template<class T>
T abs(T x)
{
   return (x>=0) ? x : -x;
}
```

```
int x, ax;       ax = abs(x);
short y, ay;     ay = abs(y);
double z, az;    az = abs(z);
char * s, *as;  // as = abs(s);
```

```
double r = abs(-2.4);
int s = abs<int>(3.5);
```

```
az = abs<double>(-3);
```

Точно также можно параметризовать описания и определения классов — об этом совсем скоро ...