

## Лекция 13. Некоторые возможности C++

### Ссылочный тип данных

Язык C++ вводит много новых возможностей для написания кода. Некоторые оказываются удобными и при написании программы в “стиле C”. Одна из таких возможностей — ссылочный тип данных. Ссылка — это по сути указатель, с которым в некотором смысле можно работать как с обычной переменной (объектом).

Ссылка — синоним объекта, который инициализируется один раз в момент ее создания.

```
int x;
double y;
int *p = &x;
double *q;           q+1000
q = &y;
int &r = x;
double &s = y;
```

```
x  *p  r    --- это одно и то же
y  *q  s
```

```
x = 4;    *p = 4;    r = 4;    --- везде идет работа со значением переменной x
```

Зачем нужно?

— переобозначение объектов (для удобства) [но это не главное]

```
int a[4];
int &year = a[0];
int &month = a[1];
int &day = a[2];
int &hour = a[3];

if (hour > 24) { day++; hour -= 24; }
if (a[3] > 24) { a[2]++; a[3] -= 24; }
```

— использование в качестве параметров и возвращаемых значений.

Параметры передаются “по значению”

```
void f1(int x)           void f2(int *p)           void f3(int &r)
{
  x = 100;
}
{
  *p = 100;
}
{
  r = 100;
}
```

```
int x=0, y=0, z=0;
```

```
f1(x);          // x не изменился так как в процедуре x - другой объект
f2(&y);         // y = 100    p = &y работаем по адресу y через указатель p
f3(z);         // z = 100    r есть ссылка (синоним) z,
                //          работаем непосредственно с z через ссылку r
f1(a + b);     можно
f2(&a + &b);    нельзя
f3(a + b);     нельзя
```

```
int f1(int x)           параметр x есть копия фактического параметра
{
    int xx = x + 100;
    return xx;         возвращается копия xx
}
```

```
int * f2(int x)        параметр x есть копия фактического параметра
{
    int xx = x + 100;
    return &xx;        бессмысленно - адрес на уничтоженный объект
}
```

```
int & f3(int x)        параметр x есть копия фактического параметра
{
    int xx = x + 100;
    return xx;         бессмысленно - ссылка на уничтоженный объект
}
```

Осмысленно возвращать ссылку или указатель на объект, существующий вне данной функции.

```
int & f(int &x)
{
    долгий и сложный пересчет значения x ...
    x = x + 1000;
    return x;
}
```

```
int & MaxValue(int n, int *a)
{
    .....
    return a[imax];    ссылка на максимальный элемент массива.
}
```

```
int a[10];
....
printf("max = %d\n", MaxValue(10,a));
```

```
MaxValue(10,a) += 1000;
printf("max = %d\n", MaxValue(10,a)); // теперь оно на 1000 больше
```

## Операторы для структур

В задачах по геометрии рассматривали структуры для геометрических объектов

```
struct Point { double x, y; };
struct Vector { double x, y; };
struct Segment1 { double x1, x2; };
struct Segment2 { Point p1, p2; };
struct Line { double a,b,d; }; // ax + by + d = 0, a^2 + b^2 = 1;
и т.д.
```

C++ позволяет добавить к структурам функции и определить для них разные операции

```
struct Vector {
    double x, y;
    double Length() { return sqrt(x*x + y*y); }
    double OX_Angle { return atan2(y,x); }
};
```

```
Vector a, b;
double u, h;
a.x = 1;
a.y = 2;
....
u = a.OX_Angle();
h = a.Length() + b.Length();
```

Конструкторы. Функции для инициализации объекта

```
struct Vector {
    double x, y;
    Vector() { x=y=0; }
    Vector(double xx, double yy) { x=xx; y=yy; }

    double Length() { return sqrt(x*x + y*y); }
    double OX_Angle { return atan2(y,x); }
};

Vector a, b(12,30), c;
c = Vector(3,-8);
```

Операторы (перегрузка операций)

```

struct Vector {
double x, y;
Vector() { x=y=0; }
Vector(double xx, double yy) { x=xx; y=yy; }

double Length() { return sqrt(x*x + y*y); }
double OX_Angle { return atan2(y,x); }

//Vector operator+ (const Vector &v) const {Vector c(x + v.x, y + v.y); return c;}
double operator* (const Vector &v) const { return x*v.x + y*v.y; }
Vector operator* (double r) const { return Vector(x*r, y*r); }
};

Vector a, b, c, s;      s = a.operator+(b);      s = a + b;
...
a = b + c*2.5;
z = b*a;              z = b.operator*(a);

a = 3*b;  !!!!

Vector operator* (double r, const Vector &v) { return Vector(v.x*r, v.y*r); }
Vector operator+ (const vector &u, const Vector &v)
{
    Vector c(u.x + v.x, u.y + v.y);
    return c;
}

```

И так далее ....