

Лекция № 12

Перегрузка операторов и функций.

Методы класса

Определяя класс, мы определяем новый тип.
Есть новый тип, какие доступны методы ?

Методы – это функции реализующие разные действия.

Действия могут записываться символом.

Создаем класс вектор

```
class vector
{
    public:
        int data[10];
        vector(int);    // конструктор
        void print(void); // печать
        void sum(vector); // сумма
};
```

Конструктор в классе вектор

```
vector :: vector(int param)
{
    int k;
    for (k=0; k<10; k++)
    {
        data[k]=rand() %param;
    }
}
```

Метод печать в классе вектор

```
void vector :: print(void)
{
    int k;
    for (k=0; k<10; k++)
    {
        printf("%3d", data[k]);
    }
    printf("\n");
}
```

Метод сложение в классе вектор

```
void vector::sum(vector VC)
{
    int k;
    for (k=0; k<10; k++)
    {
        data[k] += VC.data[k];
    }
}
```

Использование класса вектор

```
int main(void)
{
    vector A(10) , B(3) ;
    A.print() ;
    B.print() ;

    A.sum(B) ; // сложили с A вектор B

    A.print() ;
}
```

Улучшим метод sum

Ранее мы написали

```
A . sum (B) ;
```

Программа лучше читается, если писать так:

```
A = A . sum (B) ;
```


Метод сложение в классе вектор

Вариант 2

```
vector vector::sum(vector VC)
{
    vector T(10); int k;
    for (k=0; k<10; k++)
    {
        data[k]+=VC.data[k];
        T.data[k]=data[k];
    }
    return T;
}
```

Зачем и когда перегружать ?

Ранее мы написали

```
A . sum (B) ;
```

Еще лучше, если записать

```
A = A + B ;
```

т. е. вместо функции `sum` пишем значок `+` 😊

Перегружать операторы следует, если это упрощает понимание программы.

Ключевое слово **operator**

Вместо имени **sum** пишем **operator +**

```
class vector
```

```
{
```

```
...
```

```
void operator + (vector);
```

```
};
```

```
vector vector::operator + (vector VC)
```

```
{
```

```
...
```

Окончательный вариант программы (1)

```
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>

class vector
{
public:
    int data[10];
    vector(int);
    void print(void);
    vector operator +(vector);
};

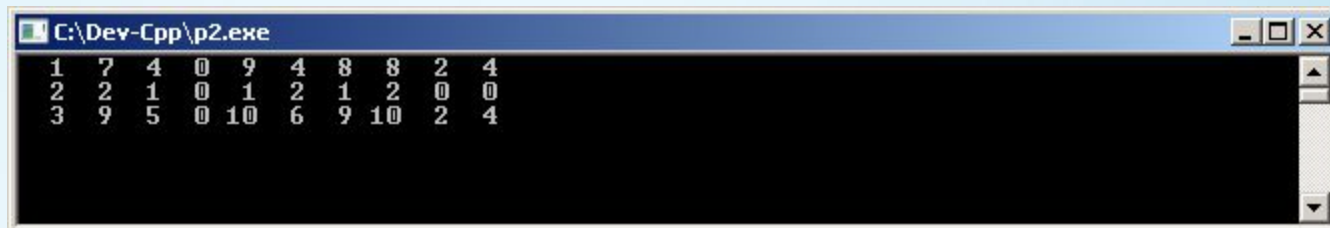
//----- конструктор
vector::vector(int param)
{
    int k;
    for (k=0; k<10; k++)
    {
        data[k]=rand()%param;
    }
}

//----- печать вектора
void vector::print(void)
{
    int k;
    for (k=0; k<10; k++)
    {
        printf("%3d",data[k]);
    }
    printf("\n");
}
```

Окончательный вариант программы (2)

```
//----- сложение векторов
vector vector::operator +(vector vec)
{
    vector T(10);
    int k;
    for (k=0; k<10; k++)
    {
        data[k]+=vec.data[k];
        T.data[k]=data[k];
    }
    return T;
}

int main(void)
{
    vector A(10), B(3);
    A.print();
    B.print();
    A=A+B;
    A.print();
    getch();
    return 0;
}
```



```
C:\Dev-Cpp\p2.exe
1 7 4 0 9 4 8 8 2 4
2 2 1 0 1 2 1 2 0 0
3 9 5 0 10 6 9 10 2 4
```

Какие операции не перегружаются ?

- . выбор элемента
- . * указатель на элемент
- :: разрешение области видимости
- ? : операция сравнения

Перегрузка операторов

Перегрузка действует только для класса, в котором он определяется.

Если программа использует оператор с неклассовыми переменными (например, **int** или **float**), используется стандартное определение оператора.

Вспомним функции языка C

// Арифметика над длинными числами

```
int A[100];  
int B[100];  
int C[1000000];
```

```
int* sum(int*, int*); // сумма  
int* mul(int*, int*); // умножение  
int* scal(int*, int); // умножение  
                        // на число
```


Трудности с функциями

$A = 329392749837549265785367325$

$B = 83748374837483748234961414$

$$A * (A + 5 * B)$$

`mul (A, sum (A, scal (5, B)))`

Если выражение станет сложнее ?

Дополнительные действия, переменные и т. п. 😞

Перегрузка функций

Складывать вектора умеем ☺

$$A = A + B;$$

Как сделать, чтобы можно было прибавлять массив ?

```
int MS = {1,1,1,1,1,2,2,2,2,2};
```

...

```
A = A + MS;
```

Перегрузка функций

Заведем еще одну функцию с именем **sum** для сложения объекта **vector** с массивом.

```
class vector
{
    public:
        int data[10];
        vector(int);    // конструктор
        void print(void); // печать
        vector sum(vector); // + вектор
        vector sum(int*);    // + массив
};
```

Перегрузка функций

```
vector vector::sum(int* VC)
{
    vector T(10);
    int k;
    for (k=0; k<10; k++)
    {
        data[k]+=VC[k];
        T.data[k]=data[k];
    }
    return T;
};
```

Вызов перегруженной функции

```
int main(void)
{
    int MS[10]={1,1,1,1,1,2,2,2,2,2};
    vector B(5);
    B.print(); B=B.sum(MS); B.print();

    vector C(2); C.print();
    B=B.sum(C); B.print();

    getch();
    return 0;
}
```

Перегрузка функций

Перегрузка функций предоставляет несколько "взглядов" на одну и ту же функцию.

Несколько функций с одним и тем же именем и типом возвращаемого значения, которые отличаются только количеством и типом параметров.

При компиляции C++ определит, какую функцию следует вызвать.

Перегрузка функций повышает удобочитаемость программы.

Класс точка на плоскости

```
#include <stdio.h>
#include <math.h>
#include <conio.h>

class point
{
    public:
        float x;
        float y;
        point(float, float); // конструктор
        float dist(void);    // -----перегрузка
        float dist(point);   // -----
};

point::point(float x0, float y0) // конструктор
{
    x=x0; y=y0;
}

float point::dist(void)
{
    return hypot(x,y);
}

float point::dist(point p)
{
    return hypot(x-p.x,y-p.y);
}

int main(void)
{
    point p_first(0,1); // создали один объект
    printf("Distance to (0,0) %f\n",p_first.dist());
    point p_second(0,3); // создали еще один объект
    printf("Distance between 1 and 2 points %f\n",p_first.dist(p_second));
    getch();
    return 0;
}
```

Три кита ООП

Перегрузка функций и операторов

это реализация

Наследования

Полиморфизма

Инкапсуляции

?