

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

1

ООП

Конструкторы и деструкторы

2

```
class Date
{
    int day, month, year;
    static int default_day, default_month, default_year;
public:
    int *day_in_month;
    Date(int d = 0, int m = 0, int y = 0);
    void add_year(int n);
    void add_month(int n);
    void add_day(int n);
    static void set_default(int d, int m, int y);
}
```

Конструкторы и деструкторы

3

```
int main()
{
    Date::set_default(4, 5, 1945);
    Date today = Date(23, 6, 1983);
    today.day_in_month = new int[12];
    today.day_in_month[0] = 31;
    today.day_in_month[1] = 30;
    Date start_date = Date(21, 1, 1993);
    today.day_in_month = new int[12];
    today.day_in_month[0] = 31;
    today.day_in_month[1] = 30;
    return 0;
}
```

Конструкторы и деструкторы

4

```
class Date
{
    int day, month, year;
    static Date default_date;
public:
    int *day_in_month;
    Date(int d = 0, int m = 0, int y = 0);
    void add_year(int n);
    void add_month(int n);
    void add_day(int n);
    static void set_default(int d, int m, int y);
}
```

Конструкторы и деструкторы

5

```
Date::Date(int dd, int mm, int yy)
{
    day = dd ? dd : default_date.day;
    month = mm ? mm : default_date.month;
    year = yy ? yy : default_date.year;
    day_in_month = new int[12];
    day_in_month[0] = 31;
    day_in_month[1] = 30;
    //...//
}
```

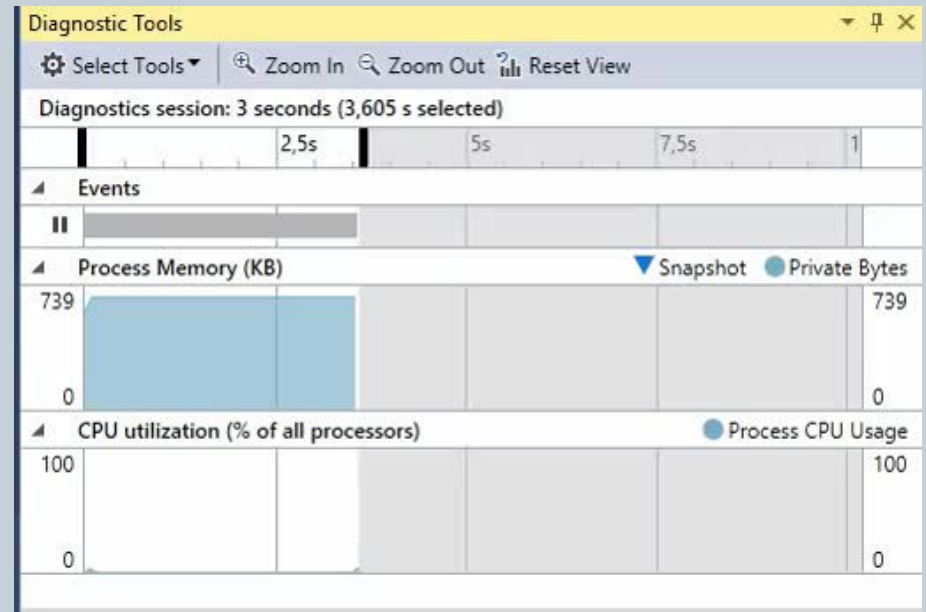
Конструкторы и деструкторы

6

```
int main()
{
    Date::set_default(4, 5, 1945);
    for (int i = 1; i < 1000; i++)
    {
        Sleep(10);
        Date today;
    }
    return 0;
}
```

Скомпилируется программа?

Все ли хорошо работает?



Конструкторы и деструкторы

7

```
class Date
{
    int day, month, year;
    static int default_day, default_month, default_year;

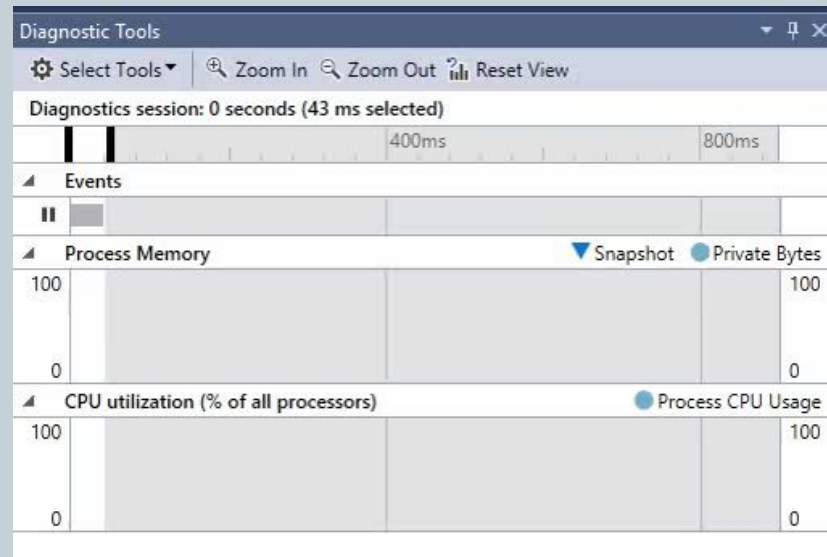
public:
    int *day_in_month;
    Date(int d = 0, int m = 0, int y = 0); // день, месяц, год
    ~Date();
    void add_year(int n); // прибавить n лет к d
    void add_month(int n); // прибавить n месяцев к d
    void add_day(int n); // прибавить n дней к d
    static void set_default(int d, int m, int y);
};
```

Конструкторы и деструкторы

8

```
Date::~~Date()
{
    delete[] day_in_month;
}

int main()
{
    Date::set_default(4, 5, 1945);
    for (int i = 1; i < 1000; i++)
    {
        Sleep(10);
        Date today;
    }
    return 0;
}
```



Копирование объектов класса

9

```
int main()
{
    Date::set_default(4, 5, 1945);
    Date today = Date(23, 6, 1983);
    Date xmas(25, 12, 1990); // сокращенная форма
    Date birthday = today;
    return birthday;
}
```

birthday	{day=23 month=6 year=1983 ...}
day	23
month	6
year	1983

- По умолчанию, объекты класса можно копировать.

```
int main()
{
    Date::set_default(4, 5, 1945);
    Date today = Date(23, 6, 1983);
    Date xmas(25, 12, 1990); // сокращенная форма
    Date birthday = today; // копирующая инициализация
    return 0;
}
```

- Копия объекта класса содержит копию каждого члена класса.

Копирование объектов класса

10

```
class Date
{
    int day, month, year;
    static int default_day, default_month, default_year;
public:
    int *av_day;
    Date(int d=0, int m=0, int y=0); // день, месяц, год
    void add_year(int n); // прибавить n лет
    void add_month(int n); // прибавить n месяцев
    void add_day(int n); // прибавить n дней
    static void set_default (int d, int m, int y);
};
```

Копирование объектов класса

11

```
int main()
{
    Date::set_default(4, 5, 1945);
    Date today = Date(23, 6, 1983);
    *(today.av_day) = 10;
    Date birthday=today;
    *(birthday.av_day) = 15;
    cout << *(today.av_day) << " " << *(birthday.av_day) <<
endl;
    return 0;
}
```

Скомпилируется программа?

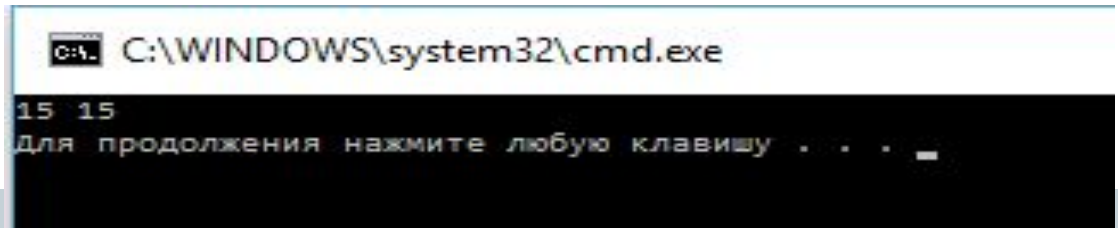
Что мы увидим на экране
после выполнения программы?

Копирование объектов класса

12

- Будьте внимательны, необдуманное прямое копирование приводит к ошибкам!!!

```
int main()
{
    Date::set_default(4, 5, 1945);
    Date today = Date(23, 6, 1983);
    today.av_day = new int;
    *(today.av_day) = 10;
    Date birthday=today;
    *(birthday.av_day) = 15;
    cout << *(today.av_day) << " " << *(birthday.av_day) <<
endl;
    return 0;
}
```



```
C:\WINDOWS\system32\cmd.exe
15 15
для продолжения нажмите любую клавишу . . .
```

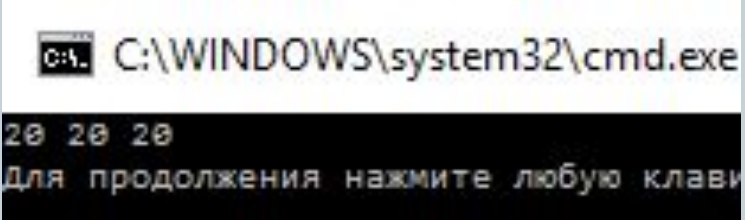
Копирование объектов класса

13

```
int main()
{
    Date::set_default(4, 5, 1945);
    Date today = Date(23, 6, 1983);
    today.av_day = new int;
    *(today.av_day) = 10;
    Date birthday=today; // копирующая инициализация
    *(birthday.av_day) = 1 // копирующее
    Date day_start;      присваивание
    day_start = today;
    *(day_start.av_day) = 20;
    cout << *(today.av_day) << " " << *(birthday.av_day) << " " <<
    *(day_start.av_day) << endl;
    return 0;
}
```

Скомпилируется программа?

Что мы увидим на экране
после выполнения программы?



```
C:\WINDOWS\system32\cmd.exe
20 20 20
Для продолжения нажмите любую клавишу
```

Копирование объектов класса

14

- Можно избежать подобных аномалий, определив, что понимать под копированием

Date:

```
class Date
{
    int day, month, year;
    static Date default_date;
public:
    /...../
    Date(const Date&);
    Date& operator=(const Date&);
};
```

Копирование объектов класса

15

```
Date::Date(const Date& D) // копирующий конструктор
```

```
{  
    av_day = new int;  
    *av_day = *(D.av_day);  
    day = D.day;  
    month = D.month;  
    year = D.year;  
}
```

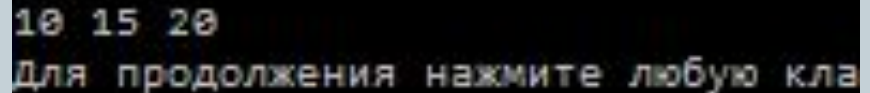
```
Date& Date::operator=(const Date& D) // присваивание
```

```
{  
    if (this != &D) // чтобы уберечься от присваивания самому себе: t=t  
        *av_day = *(D.av_day);  
    day = D.day;  
    month = D.month;  
    year = D.year;  
    return *this;  
};
```

Копирование объектов класса

16

```
int main()
{
    Date::set_default(4, 5, 1945);
    Date today = Date(23, 6, 1983);
    today.av_day = new int;
    *(today.av_day) = 10;
    Date birthday=today; // копирующая инициализация
    *(birthday.av_day) = 15;
    Date day_start;
    day_start = today;
    *(day_start.av_day) = 20;
    cout << *(today.av_day) << " " << *(birthday.av_day) << " " <<
        *(day_start.av_day) << endl;
    return 0;
}
```



```
10 15 20
Для продолжения нажмите любую клавишу
```


Перегрузка операторов

17

- Как и большинство других языков C++ поддерживает набор операторов для встроенных типов.
- Например:
 - постфиксный инкремент $lvalue ++$
 - постфиксный декремент $lvalue --$
 - умножение $expr * expr$
 - сложение (плюс) $expr + expr$
 - И т.д
- C++ позволяет объявить функции, определяющие смысл некоторых операторов.

Перегрузка операторов

18

- Можно объявить функции, определяющие смысл

+	-	*	/	%	^	&
	~	!	=	<	>	+=
--	*=	/=	%=	^=	&=	=
<<	>>	>>=	<<=	==	!=	<=
>=	&&		++	--	->*	,
->	[]	()	<i>new</i>	<i>new</i> []	<i>delete</i>	<i>delete</i> []

- Следующие операторы не могут быть определены пользователем:

- :: (разрешение области видимости);
- . (выбор члена);
- .* (выбор члена через указатель на член).

Перегрузка операторов

19

- Имя операторной функции начинается с ключевого слова ***operator***, за которым следует сам оператор; например ***operator<<***.
- Операторная функция объявляется и может быть вызвана, как любая другая функция.
- Использование операторной функции как оператора является просто сокращенной формой ее явного вызова.

Перегрузка операторов

20

- Имя операторной функции начинается с ключевого слова ***operator***, за которым следует сам оператор; например ***operator<<***.
- Операторная функция объявляется и может быть вызвана, как любая другая функция.
- Использование операторной функции как оператора является просто сокращенной формой ее явного вызова.

Перегрузка операторов

21

```
class complex {  
    double re, im;  
public:  
    complex();  
    complex(double r, double i);  
    complex operator+ (complex);  
    complex operator* (complex);  
};
```

Перегрузка операторов

22

```
int main()
{
    complex a = complex(1, 3.1);
    complex b = complex(1.2, 2);
    complex c = b;
    a = b + c;
    a = b.operator+(c);
    b = b + c* a;
    c = a*b + complex(1, 2);
    return 0;
}
```

Бинарные и унарные операторы

23

- Бинарный оператор можно определить
 - либо в виде нестатической функции-члена с од-ним аргументов
 - либо в виде функции-не-члена с двумя аргументами.
- Для любого бинарного оператора @ выражение ***aa@bb*** интерпретируется либо как ***aa.operator@(bb)*** либо ***operator@(aa, bb)***

Бинарные и унарные операторы

24

```
class X {
    public:
        void operator+(int);
        X(int);
};

void operator+ (X, X),
void operator+ (X, double);
void f(X a)
{
    a + 1; // a.operator+(i)
    1 + a; // operator+(X(1), a)
    a + 1; // operator+(a, 1.0)
}
```


Бинарные и унарные операторы

25

- Унарный оператор, префиксный или постфиксный, можно определить либо в виде нестатической функции-члена без аргументов, либо в виде функции-не-члена, с одним аргументом.
- Для любого унарного оператора @ выражение **@aa** интерпретируется либо как **aa.operator@()**, либо как **operator@ (aa)**.

Бинарные и унарные операторы

26

- Для любого постфиксного оператора @ выражение ***aa@*** интерпретируется либо как ***aa.operator@ (nt)*** либо как ***operator@ (aa, int)***.
- Если определены обе функции, для определения того, какую (возможно, никакую) из них использовать, применяется механизм разрешения перегрузки

Бинарные и унарные операторы

27

Какие из определений функций верны?

```
class complex{
    // члены (с неявным указателем)
    complex* operator&();
    complex operator&(complex);
    complex operator++(int);
    complex operator& (X,X);
    complex operator/();
};
//функции-не-члены
complex operator-(complex);
complex operator-(complex, complex);
complex operator--(complex&, int);
complex operator-();
complex operator- (complex, complex, complex);
complex operator%(complex);

// префиксный унарный оператор &
// бинарный оператор & (и)
// постфиксный инкремент
// ошибка: три операнда
// ошибка: унарный оператор

// префиксный унарный минус
// бинарный минус
// постфиксный декремент
// ошибка : отсутствует операнд
// ошибка : три операнда
// ошибка: унарный оператор %
```

Какие из прототипов функций верны?

Перегрузка операторов

28

- Рассмотрим бинарный оператор $@$, если x имеет тип X , а y имеет тип Y , правила разрешения выражения $x@y$ применяются следующим образом:
 - если X является классом, выяснить, определяется ли *operator@* в качестве члена класса X , либо базового класса X ;
 - если X определен в пространстве имен N , поискать объявления *operator@* в N ;
 - если Y определен в пространстве имен M , поискать объявления *operator@* в M .

Операторы-члены и не-члены

29

- Рекомендуется сводить к минимуму количество функций, непосредственно манипулирующих представлением объекта.
- Старайтесь определять в теле самого класса только те операторы, которые должны модифицировать значение первого аргумента, например оператор +=.
- Операторы, которые просто выдают новое значение на основе своих аргументов, такие как +, рекомендуется определять вне класса

Операторы-члены и не-члены

30

```
class complex {
    double re, im;
public:
    complex& operator+=(complex a); // требует доступа к представлению
    // . . .
};
complex operator+(complex a, complex b)
{
    complex r = a;
    return r += b; // доступ к представлению при помощи +=
}
inline complex& complex::operator+=(complex a)
{
    re += a.re;
    im += a.im;
    return *this;
}
```

Операторы-члены и не-члены

31

```
void f(complex x, complex y, complex z) // r1 = operator+(operator+(x,y), z)
{
    // r2 = x
    complex r1 = x + y + z; // r2.operator+=(y)
    complex r2 = x; // r2.operator+=(z)
    r2 += y; // operator+(2,b)
    r2 += z; // Ошибка!!!
    complex d = 2 + b;
}
```

Операторы-члены и не-члены

32

```
complex& complex::operator+=(double a){
    re += a;
    return *this;
}
complex operator+(complex a, complex b){
    complex r = a;
    return r+= b; // вызывает complex: :operator+= (complex)
}
complex operator+(complex a, double){
    complex r = a;
    return r += b; // вызывает complex::operator+=(double)
}
complex operator+ (double a, complex b){
    complex r = b;
    return r += a; // вызывает complex::operator+= (double)
}
```