ПЕРЕГРУЗКА ОПЕРАЦИЙ

- *Перегрузка операций* — это пример полиморфизма C++.
- C++ позволяет определять несколько функций с одинаковыми именами и разной сигнатурой (списками аргументов).
- Это называлось *перегрузкой функций* или *функциональным полиморфизмом*. Цель такой перегрузки — позволить использовать одно и то же имя функции для некоторой базовой операции, несмотря на то, что она применяется к данным разных типов. Перегрузка операций расширяет концепцию перегрузки на операции, позволяя трактовать их множеством способов.

- Перегруженные операции часто могут заставить код выглядеть более естественно. Например, общей вычислительной задачей является сложение двух массивов. Обычно это выглядит подобно следующему циклу **for:**

```
for (int i = 0; i < 20; i++)
massC[i]=massA[i]+massB[i];//поэлементное сложение
```

- Но в C++ можно определить класс, который представляет массивы и перегружает операцию + таким образом, что станет возможным приведенный ниже код:

```
massC = massA + massB;// сложить два объекта-массива
```

- Для перегрузки операции используется специальная форма функции, называемая ***функцией операции.*** Функция операции имеет следующую форму, в которой ***op*** — это символ перегружаемой операции:

operator*op* (*список-аргументов)*

- Например, **operator+ ( )** перегружает операцию **+**, а **operator* ( )** — операцию *. Операция ***op*** должна быть допустимой операцией C++, а не произвольным символом.

- Предположим, что имеется класс **Salesperson,** в котором определена функция-член **operator+ ( )** для перегрузки операции + так, что она сможет добавлять зарплату одного лица к зарплате другого лица. Тогда если **sam, sid** и **sara —** объекты класса **Salesperson,** можно написать следующее выражение:

sam = sid + sara;

- Компилятор, распознав операцию как относящуюся к классу **Salesperson,** заменит ее вызовом соответствующей функции операции:

sam = sid.operator+(sara) ;

- Если вы находились в системе под конкретным именем пользователя в течение 2 часов 35 минут с утра и 2 часов 40 минут после обеда, то сколько всего времени вы проработали в системе? Ниже приведен пример, в котором концепция сложения имеет смысл, несмотря на то, что единицы, которые вы складываете (смесь часов и минут) не соответствует какому-либо встроенному типу.

```
// mytime.h — класс Time до перегрузки операции
class Time
{
private:
int hours;
 int minutes;
public:
Time () ;
Time (int h, int m);
void AddMin(int m) ;
void AddHr (int h) ;
void Reset (int h , int m) ;
Time Sum (const Time & t) ;
void Show () ;
} ;
```

```
// mytime.cpp -- реализация методов Time
#include <iostream>
#include "mytime.h"
Time::Time()
{
hours = minutes = 0;
}
Time::Time(int h, int m ) {
hours = h; minutes = m;
}
void Time::AddMin(int m) {
minutes += m;
hours += minutes / 60;
minutes %= 60;
}
void Time::AddHr(int h) {
hours += h;
}
```

```
void Time::Reset (int h, int m) {
hours = h;
minutes = m;
}
Time Time:: Sum (const Time & t) {
Time sum;
sum.minutes = minutes + t.minutes;
sum.hours = hours + t.hours + sum.minutes / 60;
sum.minutes %= 60;
return sum;
}
void Time::Show ()  {
std::cout « hours « " hours, " « minutes « " minutes";
}
```

```
// usetime.cpp -- использование первой черновой версии класса Time
// компилировать usetime.cpp и mytime.cpp вместе
#include <iostream>
#include "mytime.h"
int main()
{
using std::cout;
using std::endl;
Time planning;
Time coding(2, 40);
Time fixing(5, 55);
Time total;
cout « "planning time = ";
planning.Show ();
cout « endl;
cout « "coding time = ";
 coding.Show ();
cout « endl;
 cout « "fixing time = ";
fixing.Show ();
cout « endl;
total = coding.Sum(fixing) ;
 cout « "coding. Sum (fixing) = ";
total.Show();
cout « endl;
return 0;
}
```

Ниже показан вывод программы из листингов

planning time = 0 hours, 0 minutes
coding time = 2 hours, 40 minutes
fixing time = 5 hours, 55 minutes
coding.Sum(fixing) = 8 hours, 35 minutes

```
// nmytime.h — класс Time после перегрузки операции
class Time {
private:
int hours;
int minutes;
public:
Time();
Time (int h, int m) ;
void AddMin (int m) ;
void AddHr(int h);
void Reset (int h , int m) ;
Time operator+(const Time & t);
void Show();
};
```

```
// nmytime.cpp — реализация методов Time
#include <iostream>
#include "nmytime. h"
Time: :Time () {
hours = minutes = 0;
}
Time::Time (int h, int m ) {
hours = h; minutes = m;
}
void Time::AddMin (int m) {
minutes += m;
hours += minutes / 60;
minutes %= 60;
}
void Time::AddHr(int h) {
hours += h;
}
```

```
void Time::Reset (int h, int m) {
hours = h; minutes = m;
}
Time Time::operator+(const Time & t)  {
Time sum;
sum.minutes = minutes + t.minutes;
sum.hours = hours + t.hours + sum.minutes / 60;
sum.minutes %= 60;
return sum;
}
void Time::Show() {
std::cout « hours « " hours, " « minutes « " minutes";
}
```

- Метод **operator+** () можно вызвать с использованием того же синтаксиса, что и в случае с **Sum ():**

total=coding.operator+(fixing);//нотация с функцией

- Но назначение методу имени **operator+** () позволяет также применить нотацию операции:

total = coding + fixing;// нотация с операцией

- Оба варианта вызывают метод **operator+** (). Обратите внимание, что в нотации с операцией объект слева от операции (в рассматриваемом случае **coding)** является вызывающим объектом, а объект справа (в данном случае **fixing)** передается в качестве аргумента.

```
// nusetime.cpp – использование второй черновой версии
класса Time
// компилировать nusetime.cpp и nmytime.cpp вместе
#include <iostream>
#include “nmytime.h"
int main () {
using std::cout; using std::endl;
Time planning; Time coding(2, 40); Time fixing (5,
55); Time total;
cout « "planning time = "; planning.Show () ;
cout « endl; cout « "coding time = "; coding.Show ();
cout « endl; cout « "fixing time = "; fixing.Show () ;
cout « endl; total = coding + fixing;
// Нотация с операцией
```

```
cout << "coding + fixing = ";
total.Show(); cout « endl;
Time morefixing (3, 28);
cout « "more fixing time = ";
morefixing.Show (); cout « endl;
total = morefixing.operator+(total) ;
cout<<"morefixing.operator+(total)= ";
total.Show(); }cout « endl; return 0;}
```

int a, b, c;

Time A, B, C;

c = a + b; // используется сложение значений int

C = A + B; // используется сложение, определенное для объектов Time

ОГРАНИЧЕНИЯ ПЕРЕГРУЗКИ

- Перегруженные операции должны иметь как минимум один операнд типа, определяемого пользователем. Это предотвращает перегрузку операций, работающих со стандартными типами. То есть переопределить операцию "минус" (-) так, чтобы она вычисляла сумму двух вещественных чисел вместо разности, не получится. Это ограничение сохраняет здравый смысл, заложенный в программу, хотя и несколько препятствует полету творчества.
- Вы не можете использовать операцию в такой манере, которая нарушает правила синтаксиса исходной операции.

- Вы не можете определять новые символы операций.
- Нельзя перегружать следующие операции:

Операция	Описание
`sizeof`	Операция `sizeof`
`.`	Операция членства
`.*`	Операция указателя на член
`::`	Операция разрешения контекста
`:?`	Условная операция
`typeid`	Операция RTTI (runtime type identification — определение типа во время выполнения)
`const_cast`	Операция приведения типа
`dynamic_cast`	Операция приведения типа
`reinterpret_cast`	Операция приведения типа
`static_cast`	Операция приведения типа

- Для перегрузки перечисленных ниже операций можно использовать ***только*** функции- члены:

Операция	Описание
=	Операция присваивания
()	Операция вызова функции
[]	Операция индексации
->	Операция доступа к членам класса через указатель

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