

9. Предефиниране на операции

План:

Предефиниране на операции

Предефиниране на операции като обикновени функции

Предефиниране на операции като член-функции на клас

Възможности на езика C++ за предефиниране на операции

Свързан списък с предефинирани операции



Предефиниране на операции.

Даване на ново значение на операция се нарича предефиниране (operator overloading).

До сега използвани предефинирани операции:

<<, >> (потоци)

+ (събиране, конкатенация, адресна аритметика),

*, ++ (операции за итератори).



Предефиниране на операции като обикновени функции.

Операциите в C++ се предефинират за нови аргументи - обекти от даден (потребителски) клас.

При предефиниране на бинарна операция, типът на поне един от аргументите трябва да е от този клас.

Аргументите на предефинирана операция са параметри на функцията за предефиниране.

Пример: За обекти от класа Time предефинираме операциите: -, +, ==, !=, <<, префиксна и постфиксна операция ++.

```
// overload.cpp
```

```
#include <iostream>
```

```
using namespace std;
```

```
#include "ccc_time.h"
```

```
/* Предефиниране на бинарна операция – с аргументи от клас Time и стойност от тип long */
```

```
long operator-(Time a, Time b)
```

```
/* ЦЕЛ: пресмята броя на секундите между два момента от време
```

```
ПОЛУЧАВА: a, b – два момента от време
```

```
ВРЪЩА: броя на секундите между a и b
```

```
*/
```

```
{ return a.seconds_from(b); }
```

```
/* Предефиниране на бинарна операция + с аргументи от клас Time и тип long и стойност от клас Time */
```

```
Time operator+(Time a, long sec)
```

```
/* ЦЕЛ: пресмята момент от време, отдалечен на зададен брой секунди
```

```

    ПОЛУЧАВА: a – момент от време
    ВРЪЩА: момент от време, отдалечен на зададения брой секунди
*/
{   Time r = a;
    r.add_seconds(sec);
    return r;  }

/* Предефиниране на бинарна операция == с аргументи от клас Time и стойност от тип bool */
bool operator==(Time a, Time b)
/* ЦЕЛ: сравнява два момента от време
   ПОЛУЧАВА: a, b – два момента от време
   ВРЪЩА: true ако те са равни и false ако не са равни
*/
{   return a.seconds_from(b) == 0;  }

/* Предефиниране на бинарна операция != с аргументи от клас Time и стойност от тип bool */
bool operator!=(Time a, Time b)
/* ЦЕЛ: сравнява два момента от време
   ПОЛУЧАВА: a, b – два момента от време
   ВРЪЩА: true – ако те не са равни
*/
{   return a.seconds_from(b) != 0;  }

/* Предефиниране на бинарна операция << с аргументи от клас ostream и клас Time и стойност от тип ostream */
ostream &operator<<(ostream &out, const Time &a)
/* ЦЕЛ: отпечатва обект от тип Time
   ПОЛУЧАВА: out – изходен поток, a – момент от време
   ВРЪЩА: входния параметър out
*/
{   out << a.get_hours() << ":";
    if (a.get_minutes() < 10) out << "0";
    out << a.get_minutes() << ":";
    if (a.get_seconds() < 10) out << "0";
    out << a.get_seconds();
    return out;  }

/* Предефиниране на унарна операция – префиксна форма на ++ с аргумент от клас Time и стойност от клас Time */
Time operator++(Time &a)
/* ЦЕЛ: добавя към момент от време 1 секунда
   ПОЛУЧАВА: a – момент от време
   ВРЪЩА: новият момент от време и променя a

```

```

*/
{ a = a + 1; return a; }

/* Предефиниране на унарна операция – постфиксна форма на ++ с аргумент от клас Time и стойност от клас Time */
Time operator++(Time &a, int dummy)
/* ЦЕЛ: добавя към момент от време 1 минута
   ПОЛУЧАВА: a – момент от време
   ВРЪЩА: непромененият момент от време и променя a
*/
{ Time b = a;
  a = a + 60; return b;
}

int main()
{ Time now;
  cout << "Now it is " << now << "\n";
  Time later = now + 1000;
  cout << "A thousand seconds later it is " << later << "\n";
  Time now2;
  if (now == now2)
    cout << "It still is " << now2 << "\n";
  if (now != now2)
    cout << "It is already " << now2 << "\n";
  cout << "Another " << later - now2 << " seconds until "
    << later << "\n";
  now = Time();
  cout << "Now it is " << now << "\n";
  cout << "One second later " << (++now) << "\n";
  cout << "The same time " << (now++);
  cout << " and 60 seconds later " << now << "\n";
  return 0;
}

```

```

Now it is 11:42:51
A thousand seconds later it is 11:59:31
It still is 11:42:51
Another 1000 seconds until 11:59:31
Now it is 11:42:51
One second later 11:42:52
The same time 11:42:52 and 60 seconds later 11:43:52

```

Компиляторът замества предефинираната операция с извикване на функцията, която я предефинира.

Пример: Операцията + в израза `now + 1000` се замества с `operator+(now, 1000)`.



Предефиниране на операции като член-функции на клас.

Първият аргумент на предефинираната операция е текущият обект (неявният параметър).

Унарна операция се предефинира с член-функция без параметри, бинарна операция се предефинира с член-функция с един параметър.

Пример: Разширяваме класа `Time` в производен клас `Time_new`, като добавяме предефинирани операции. При декларации на функции имената на формалните параметри могат да не се пишат (пишат се само типовете на параметрите).

```
// overload1.cpp
#include <iostream>
#include <iomanip>
using namespace std;

#include "ccc_time.h"

class Time_new : public Time {
public:
    int operator-(Time_new) const; // изминнали секунди между два момента от време
    Time_new operator+(int) const; // добавяне на секунди към даден момент
    bool operator==(Time_new) const;
    bool operator!=(Time_new) const;
    Time_new operator++(); // prefix - добавяне на една секунда
    Time_new operator++(int); // postfix - добавяне на една секунда
    friend ostream& operator<<(ostream&, Time_new); // отпечатване на момент (час:минути:секунди)
};

/**
 * Compute the number of seconds between two points in time.
 * @param b another point in time
 * @return the number of seconds that a is away from b
 */
int Time_new::operator-(Time_new b) const
{ return this->seconds_from(b); }

/**
 * Compute a point in time that is some number of seconds away.
 * @param sec the seconds to add
 * @return a point in time that is sec seconds away from a
 */
Time_new Time_new::operator+(int sec) const
{ Time_new r = *this;
  r.add_seconds(sec); }
```

```

    return r;
}

/**
    Compare two points in time
    @param b another point in time
    @return true if they are the same
*/
bool Time_new::operator==(Time_new b) const
{ return this->seconds_from(b) == 0;
}

/**
    Compare two points in time.
    @param b another point in time
    @return true if they are the different
*/
bool Time_new::operator!=(Time_new b) const
{ return !(*this == b);
}

/**
    Prefix increment by 1 second.
    @return the new value
*/
Time_new Time_new::operator++()    // prefix
{ *this = *this + 1;
  return *this;
}

/**
    Postfix increment by 1 second.
    @return the old value
*/
Time_new Time_new::operator++(int dummy)    // postfix
{ Time_new t = *this;
  *this = *this + 1;
  return t;
}

/**
    Print a Time object
    @param out an output stream
    @param a a point in time
    @return out
*/
ostream& operator<<(ostream& out, Time_new a)
{ out << a.get_hours() << ":"

```

```

        << setw(2) << setfill('0')
        << a.get_minutes() << ":"
        << setw(2) << a.get_seconds() << setfill(' ');
    return out;
}


int main()
{
    Time_new now;
    cout << "now: " << now << endl;
    Time_new later = now + 1000;
    cout << "later: " << later << endl;
    Time_new now2;
    if (now == now2)
        cout << "now == now2: " << now2 << endl;
    if (now != now2)
        cout << "now != now2 " << now2 << endl;
    cout << "now++: " << now++
        << " ++now2: " << ++now2 << endl;
    cout << "now: " << now << " now2: " << now2 << endl;
    cout << "later - now2: " << later - now2 << endl;
    return 0;
}


```

Компиляторът замества предефинираната операция с извикване на член-функцията, която я предефинира.

Пример: Операцията + в израза `now + 1000` се замества с `now.operator+(1000)`.

Вариант на програмата без наследяване - предефиниране на операциите в класа Time ([overload2.cpp](#)).

 Възможности на езика C++ за предефиниране на операции
 Customizes the C++ operators for operands of user-defined types ([operator overloading](#)).

 Свързан списък с предефинирани операции
 Сега можем да заменим функциите в класовете за реализация на свързан списък ([list2.cpp](#)), които заместваха операциите от STL-реализацията ([list1.cpp](#)) с предефинирани операции:

```

bool operator!=(Iterator a, Iterator b); /* външна за класа функция */
bool Iterator::operator!=(Iterator b); /* член-функция */

```

```

string Iterator::get() const
{
    assert(position != NULL);
    return position->data;
}

```

```

void Iterator::next()

```

```

string Iterator::operator*() const
{
    assert(position != NULL);
    return position->data;
}

```

```

void Iterator::operator++(int dummy)

```

```
{ assert(position != NULL);  
  position = position->next;  
}  
  
void Iterator::previous()  
{ if (position == NULL) position = last;  
  else position = position->previous;  
  assert(position != NULL);  
}  
  
bool Iterator::not_equal(Iterator b) const  
{  
    return position != b.position;  
}
```

```
{ assert(position != NULL);  
  position = position->next;  
}  
  
void Iterator::operator--(int dummy)  
{ if (position == NULL) position = last;  
  else position = position->previous;  
  assert(position != NULL);  
}  
  
bool Iterator::operator!=(Iterator b) const  
{  
    return position != b.position;  
}
```
