Обектно ориентирано програмиране

ШАБЛОНИ НА ФУНКЦИИ И КЛАСОВЕ

Повторения

Повторения на функции:

```
int max(int a, int b) {
    return a > b ? a : b;
}
double max(double a, double b) {
    return a > b ? a : b;
}
Rational max(Rational a, Rational b) {
    return a > b ? a : b;
}
```

Повторения

Повторения на класове:

```
class Point {
   double x, y;
   ...
   void translate(double a) {
      x += a; y += a;
   }
};
class IntPoint {
   int x, y;
   ...
   void translate(int a) {
      x += a; y += a;
   }
};
```

```
class RationalPoint {
   Rational x, y;
   ...
   void translate(Rational a) {
      x += a; y += a;
   }
};
```

Шаблони на функции и класове

В предните разглеждания създадохме класа stack, реализиращ стек от цели числа. Възможно е обаче в рамките на една и съща програма да е необходимо конструирането на няколко стека от различен тип данни. Така възниква необходимостта от средства, реализиращи *класове, зависещи от параметри*, задаващи типове данни и при конкретни приложения параметрите да се конкретизират.

Такива средства са **шаблоните**. Те позволяват създаването на класове, използващи неопределени (хипотетични) типове данни за своите аргументи и по такъв начин позволяват да бъдат описвани “обобщени” типове данни. Използват се за изграждане на общоцелеви класове-контейнери (стекове, опашки, списъци и др.)

Шаблони на функции и класове

Например, чрез шаблон може да се дефинира обобщен клас за стек с неопределен тип на елементите, след което от шаблона да се получат специфични класове (клас, реализиращ стек от реални числа; клас, реализиращ стек от символи и т.н.).

При наличие на шаблони на класове възниква необходимостта и от шаблони на функции, използващи шаблони на класове. Например, искаме на реализираме сортиране на елементите на шаблон на стек. Налага се да дефинираме шаблони на функциите за намиране на минимален елемент на стек с произволен тип на елементите и сортиране на елементите на стек с произволен тип на елементите.

Шаблони на функции

Дефиниция на шаблон на функция

<дефиниция_на_шаблон_на_функция> ::=

template < class <параметър> {,class <параметър>}>

<тип_на_функция> <име_на_функция>(<формални_параметри>)

<тяло>

където

- <параметър> и <име_на_функция> са идентификатори;
- <формални_параметри> и <тяло> се определят както при дефиниция на функция. В тях са използвани указаните параметри на шаблона, вместо конкретните типове.

Забележка: Вместо ключовата дума class може да се използва typename

Шаблони на функции

Дефиницията започва със запазената дума template (шаблон), следвана от списък от параметри на шаблона, които трябва да участват като типове на аргументи на дефинираната като шаблон функция.

Използването на дефинираните шаблони на функции става чрез обръщение към “обобщената” функция, която шаблонът дефинира, с параметри от конкретен тип. Компилаторът генерира т. нар. **шаблонна функция**, като замества параметрите на шаблона с типовете на съответните фактически параметри. *При това заместване не се извършват преобразувания на типове.*

Шаблони на функции – примери

```
template <class T>
T max(T a, T b) {
   return a > b ? a : b;
}
int main() {
   int x = 2, y = 3;
   double a = 5, b = 3;
   Rational p(1, 2), q(2, 3);
   cout << max(a, b);
   cout << max(x, y);
   cout << max(p, q); // ВНИМАНИЕ: > трябва да е предефиниран за Rational
}
```

Шаблони на функции - примери

```
template <class T>
void swap(T& a, T& b) {
    T tmp = a; a = b; b = tmp;
}
template <typename T>
void reverse(T* a, int n) {
    for (int i = 0; i < n / 2; i++)
        swap(a[i], a[n - i - 1]);
}
```

Шаблони на функции - примери

Задача. Да се напише програма, която дефинира шаблон на процедура за въвеждане елементите на масив и шаблон на функция, намираща минималния елемент на масив от елементи, които могат да се сравняват.

```
#include <iostream>
using namespace std;
template <class T>
void read(int n, T* a) {
   for (int i = 0; i < n; i++) {
      cout << "a[" << i << "]= ";
      cin >> a[i];
   }
}
```

Шаблони на функции

```
template <class T>
T minarray(int n, T* a) {
   T min = a[0];
   for (int i = 1; i < n; i++)
      if (a[i]<min)
         min = a[i];
   return min;
}
```

Шаблони на функции

```
int main() {
    cout << "n: ";
    int n;
    cin >> n;
    int a[10];
    read(n, a);
    cout << minarray(n, a) << endl;
    double b[10];
    read(n, b);
    cout << minarray(n, b) << endl;
    return 0;
}
```

Шаблони на класове

Дефинирането на шаблон на клас се състои от декларация на шаблона и дефиниране на член-функциите му.

Декларация на шаблон на клас

<декларация_на_шаблон_на_клас> ::=

template < class <параметър>[=<име_на_тип>]

{,class <параметър>[=<име_на_тип>]} >

class <име_на_шаблон_на_клас>

<тяло>;

където

- <параметър>, <име_на_шаблон_на_клас> и <име_на_тип> са идентификатори. В <тяло> са използвани указаните параметри на шаблона, вместо конкретните типове.

Шаблони на класове

Броят на параметрите на шаблон на клас е произволен. Параметрите могат да участват на произволни места в дефиницията на шаблона. Освен това е възможно някои или всички параметри на шаблона да са подразбиращи се. Това се осъществява чрез добавяне на инициализацията =<име_на_тип> след името на параметъра. В този случай, при пропускане на параметър, се използва подразбиращият се тип.

Пример:

```
template <class T, class S = int>
class Example {
public:
   T func1(T x, S y);
   S func2(T x, S y);
private:
   T a;
   S b;
};
```

Шаблони на класове

Дефинирането на член-функции на шаблон се осъществява по два начина – като вградени и не като вградени (описани извън декларацията).

При дефинирането на вградените член-функции няма особености. Ако е необходимо, използват се параметрите на класа.

Пример:

```
template <class T, class S = int>
class Example {
public:
   T func1(T x, S y) {// вградена член-функция
      cout << "func1 \n";
      return x;
   }
   S func2(T x, S y);
private:
   T a;
   S b;
};
```

Шаблони на класове

В другия случай дефиницията се предшества от

template <списък_от_параметри>

а пълното име на член-функцията на шаблона се получава с префикса

<име_на_шаблон_на_клас> **<** <параметър> {,<параметър>} **>**

Пример:

```
template <class T, class S>
S Example<T, S>::func2(T x, S y)
{
    cout << "func2 \n";
    return y;
}
```

Шаблони на класове

Забележка: Префиксът се използва и когато член-функция на шаблона е от тип шаблон на клас, указател или псевдоним на шаблон на клас.

Шаблоните на класове не са истински класове, а описания, които се използват от компилатора за създаване на конкретни (шаблонни) класове. Наричат се още **специализации на шаблона на класа**.

Дефиниция на шаблонен клас

<дефиниция_на_шаблонен_клас> ::=

<име_на_шаблон_на_клас> **<** <тип>**,** <тип>**,...** **>**

<тип> ::= <име_на_тип>

Ако някой <тип> е пропуснат, използва се подразбиращият се, ако декларацията на шаблона е с подразбиращи се параметри, или се съобщава за грешка. При срещане на декларация на шаблонен клас, на базата на зададените типове, компилаторът генерира съответен шаблонен клас.

Шаблони на класове - използване

Шаблоните на класове се използват чрез явно указване на параметрите (параметрите по подразбиране могат да бъдат изпускани)

Директно инстанциране:

```
Point<int> p;
double distance(Point<double> p1, Point<double> p2) { ... }
```

Чрез дефиниране на потребителски тип

```
typedef Point<double> DoublePoint;
```

Използване в шаблон на функция

```
template <typename T>
double distance(Point<T> p1, Point<T> p2) { ... }
```

Шаблони на класове - използване

Шаблоните на класове **не се компилират**

Затова шаблоните на класовете заедно с дефинициите на член-функциите (те са шаблони на функции) се слагат в заглавни файлове

При всяко използване на шаблон с различни параметри се генерира нов шаблонен клас

Компилират се само член-функциите, които се използват от съответния шаблонен клас

- може да не разберем, че има грешка в член-функция на шаблон, докато не я използваме!

Шаблони на класове

Примери:

- Шаблон на клас Stack
- Шаблон на клас, реализиращ динамичен масив