Шаблони на функции и на класове. Приятелски функции и класове.

гл.ас., д-р. Нора Ангелова

Шаблони на функции

- Шаблоните на функциите - позволяват създаването на функции, използващи неопределени (хипотетични) типове данни за свои параметри или за резултата от обръщението към функцията.
- Чрез тях се описват „обобщени“ функции.

Шаблони на функции

- Функция, която въвежда елементите на масив от цели числа

```
void readIntArr(int n, int* a) {
    for (int i = 0; i < n; i++) {
        cout << "a[" << i << "]= ";
        cin >> a[i];
    }
}
```

- Функция, която въвежда елементите на масив от реални числа

```
void readDoubleArr(int n, double* a) {
    for (int i = 0; i < n; i++) {
        cout << "a[" << i << "]= ";
        cin >> a[i];
    }
}
```

ШАБЛОНИ НА ФУНКЦИИ

- Дефиниция на шаблон на функция

<шаблон_на_функция> ::=

template <typename <параметър>
{, typename <параметър> } >

<тип> <име_на_шаблон_на_функция> (<формални_параметри>)
<тяло>

Шаблони на функции

- Използване на шаблон на функция

Използването на дефинираните шаблони на функции се осъществява 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

- Пример

Да се дефинира шаблон на функция за въвеждане на елементите на едномерен масив (за елементите е дефиниран операторът >>)

```
template <typename T>
void read(int n, T* a) {
    for (int i = 0; i < n; i++) {
        cout << "a[" << i << "]= ";
        cin >> a[i];
    }
}

…
int a[10]; double b[10];
read(n, a);
read(n, b);
```

Шаблони на класове

- Шаблоните на класове - позволяват създаването на класове, използващи неопределени (хипотетични) типове данни за свои член-данни, за параметри на член-функции, за резултати от обръщения към член-функции.
- Чрез тях се описват „обобщени“ класове – класове, зависещи от параметри.
- Използват се за изграждане на общоцелеви класове-контейнери (стекове, опашки, списъци и др.).

Шаблони на класове

Пример

- Да се дефинира клас, реализиращ стек от реални числа
- Да се дефинира клас, реализиращ стек от символи

Чрез шаблон може да се дефинира „обобщен“ клас стек с неопределен тип на елементите, след което от шаблона да се получат специфични класове

Шаблони на класове

- Декларация на шаблон на клас

<декларация_на_шаблон_на_клас> ::=

template < <списък_от_параметри> >
class <име_на_шаблон_на_клас>
<тяло>

<списък_от_параметри> ::= typename <параметър>
{, typename <параметър> }

Ако параметър е с подразбираща се стойност, всички параметри след него също трябва да са с подразбиращи се стойности.

Шаблони на класове

- Пример

Шаблон на клас с два параметъра, вторият от които е подразбиращ се.

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

ШАБЛОНИ НА КЛАСОВЕ

- Дефинирането на член-функциите на шаблон на клас се осъществява като:
  - вградени (inline) член-функции;
  - не като вградени член-функции.

Шаблони на класове

- Пример (вградени член-функции)

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

Шаблони на класове

- Пример (невградена член-функции)

Дефиницията се предшества от
template < <списък_от_параметри> >

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

Шаблони на класове

- Дефиниция на шаблонен клас

<шаблонен_клас> ::=
<име_на_шаблон_на_клас> < <тип> { , <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 дефиниция на шаблонен клас, като използва зададените типове, компилаторът генерира съответен клас.

Шаблони на класове

- Пример

```
typedef example<int, double> CL1
```

дефинира класа CL1, който е специализация на шаблона на класа example при T – int и S – double

Шаблони на класове

Забележка.

Ако и двата параметъра на шаблона на класа example са подразбиращи се, ще е възможна и специализацията

typedef example <> CL3;

Скобите <> трябва да присъстват.

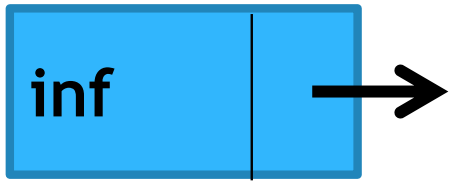
Шаблон на стек

Задача

Да се дефинират шаблон на класа stack

ШАБЛОН НА СТЕК

```
template <typename T>
struct elem {
  T inf;
  elem* link; // elem<T>* link;
};
```



ШАБЛОН НА СТЕК

```
template <typename T>
class stack {
  public:
    stack();
    ~stack();
    stack(const stack&);
    stack& operator=(const stack&);
    void push(const T&);
    void pop(T&);
    bool empty() const;
    void print();
  private:
    elem<T> *start; // указател
    void delstack();
    void copystack(const stack&);
};
```

ШАБЛОН НА СТЕК

```
template <typename T>
stack<T>::stack() {
    start = NULL;
}

template <typename T>
stack<T>::~stack() {
    delstack();
}
```

ШАБЛОН НА СТЕК

```
template <typename T>
stack<T>::stack(const stack<T>& r) {
    copystack(r);
}

template <typename T>
stack<T>& stack<T>::operator=(const stack<T>& r)
{
  if (this != &r) {
    delstack();
    copystack(r);
  }
  return *this;
}
```

ШАБЛОН НА СТЕК

```
template <typename T>
void stack<T>::delstack() {
    elem<T> *p;
    while (start) {
        p = start;
        start = start->link;
        delete p;
    }
}
```

ШАБЛОН НА СТЕК

```
template <typename T>
void stack<T>::copystack(const stack<T>& r) {
    if (r.start) {
        elem<T> *p = r.start, *q = NULL, *s = NULL;
        start = new elem<T>;
        assert(start != NULL);
        start->inf = p->inf;
        start->link = q;
        q = start;
        p = p->link;
        while (p) {
            s = new elem<T>;
            assert(s != NULL);
            s->inf = p->inf;
            q->link = s;
            q = s;
            p = p->link;
       }
       q->link = NULL;
   } else start = NULL;
}
```

ШАБЛОН НА СТЕК

```
template <typename T>
void stack<T>::copystack(const stack<T>& r) {
    if (!r.start) {
      start = NULL;
      return;
    }
    elem<T> *p = r.start, *q = NULL, *s = NULL;
    start = new elem<T>;
    assert(start != NULL);
    start->inf = p->inf;
    start->link = q;
    q = start;
    p = p->link;
    while (p) {
        s = new elem<T>;
        assert(s != NULL);
        s->inf = p->inf;
        q->link = s;
        q = s;
        p = p->link;
    }
    q->link = NULL;
}
```

Шаблон на стек

```
template <typename T>
void stack<T>::push(const T& s) {
      elem<T> *p = start;
      start = new elem<T>;
      assert(start != NULL);
      start->inf = s;
      start->link = p;
}
```

Шаблон на стек

```
template <typename T>
void stack<T>::pop(T& s) {
    if (start) {
        s = start->inf;
        elem<T> *p = start;
        start = start->link;
        delete p;
    } else {
        cout << "The stack is empty.\n";
        exit(1);
    }
}
```

ШАБЛОН НА СТЕК

```
template <typename T>
bool stack<T>::empty() const {
  return start == NULL;
}


template <typename T>
void stack<T>::print() {
  T x;
  while (!empty()) {
    pop(x);
    cout << x << " "; //! Ако за елементите на T е дефиниран <<
  }
  cout << endl;
}
```

Шаблони на клас

Шаблонът на клас дефинира съвкупност от класове.

Понякога се налага за конкретен тип данни член-функция на шаблона на класа да се реализира по по-различен алгоритъм.

В C++ е възможно предефинирането на член-функция на шаблон на клас за конкретен тип.

Този процес се нарича **специализация на член-функцията за съответния тип.**

ШАБЛОНИ НА КЛАС

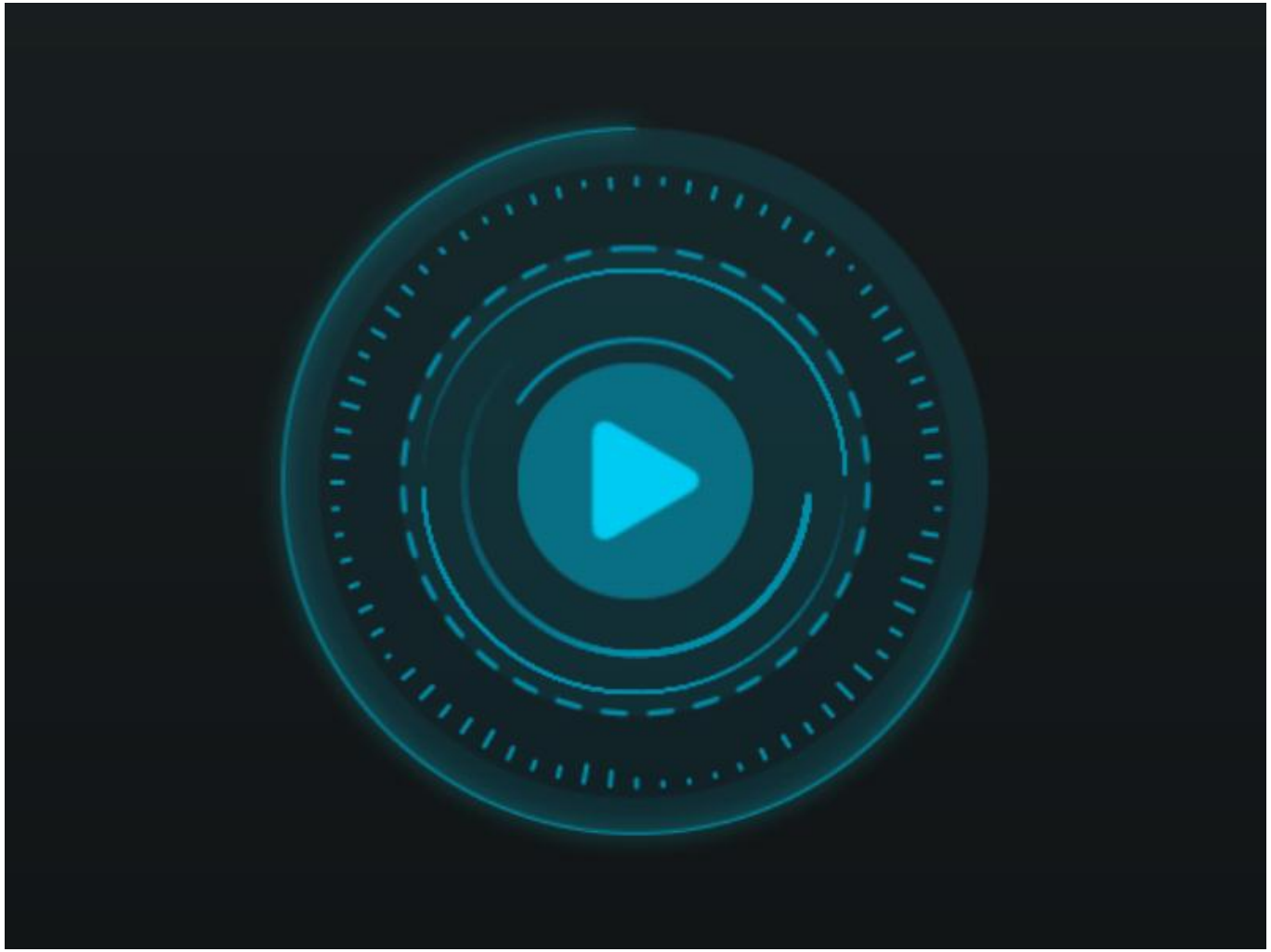
Специализация на член-функцията за съответния тип.

Пример.

Ако стекът, за който използваме print() е от символи, ще бъдат изведени символите.

Ако е необходимо само в този случай (на тип символен) да се извеждат ASCII-кодовете на символите, член-функцията print може да бъде предефинирана като функция от вида

```
void stack<char>::print() {
   char x;
   while (!empty()) {
     pop(x);
     cout << (int)x << " ";
   }
   cout << endl;
}
```



Приятелски функции и класове

- Функциите, които са приятели на даден клас имат пряк достъп до всички негови компоненти, т.е. всички членове на класа са public за функциите приятели.
- Декларират се като такива като се постави запазената дума friend пред прототипа на функцията.

Приятелски функции и класове

- Пример

```
class point {
 public:
   point();
   point(double, double);
   void read();
   void move(double, double); // транслира точка
   double get_x() const; // връща абсцисата
   double get_y() const; // връща ординатата
   void print() const; // извежда точка
 private:
   double x, y;
};
```

Приятелски функции и класове

Разстоянието между две точки се намира от външната функция dist, дефинирана по следния начин:

```
double dist(const point& p1, const point& p2) {
    return sqrt(pow(p2.get_x() - p1.get_x(), 2)
    + pow(p2.get_y() - p1.get_y(), 2));
}
```

ПРИЯТЕЛСКИ ФУНКЦИИ И КЛАСОВЕ

```
class point;
double dist(const point&, const point&);
class point {
   friend double dist(const point&, const point&);
};
…

double dist(const point& p1, const point& p2) {
   return sqrt(pow(p2.x – p1.x, 2) +
   pow(p2.y – p1.y, 2));
}
```

ПРИЯТЕЛСКИ ФУНКЦИИ И КЛАСОВЕ

```
class A {
   …
   …
   friend class B; // всички методи на класа B
                   // имат пряк достъп до всички
                   // компоненти на класа A
}
```

КРАЙ