Тема 5. Средства за организация на програмите

1. ПРЕДЕФИНИРАНЕ НА ФУНКЦИЯ
2. ПРЕДЕФИНИРАНЕ НА ОПЕРАТОРИ
3. ШАБЛОНИ (TEMPLATES)

доц. д-р инж. Владимир Николов

20.10.2017 г.

Предефиниране на функция

Дефиниция:

Най-често функциите се именоват с различни наименования, но при функции, които изпълняват еднакви операции върху различни типове данни е по-удобно да се даде еднакво наименование.

Използването на еднакво име на функция върху различни типове данни се нарича ***предефиниране на функция***.

Предефиниране на функция-пример

Пример:

Конструкторите на класа CComplex използват предефинирани функции, описващи различните видове конструктори на класа:

- Те имплементират една и съща операция - конструиране на обекта;
- С различни типове формални параметри (при различни видове данни).

Предефиниране на функция-използване

- Конструктори;
- Когато се създава функция с общоизвестно наименование (математически, вход/изход...);
- При предефиниране на оператори, чието значение е всеизвестно, а семантиката му се пренася и върху други аргументи (обекти).

Предефиниране на функция-използване

- Математически операции (sqrt...);
- Операции за извеждане на данни (print, write...);
- Файловите операции (open, close...);
- Математическите оператори, чието значение е всеизвестно, а семантиката му се пренася и върху други аргументи(+,-,*...);

Предефиниране на функция-използване

Примери в потребителски класове:

Алгебричните оператори (+, -, *) се предефинират за оперираne **на обекти** с числа или помежду им. За целта:

- Предекларира се в класа;
- Дефинира се функционалност, съответстваща на тези операции;
- Функционалността оперира с член променливите на класа и/или други данни (числа).

Предефиниране на функция-действие

Действие:

При извикването на функция с име f, която е предефинирана за различни типове аргументи трябва да подбере коя от функциите с име f трябва да бъде активирана. Последователност:

- Сравнява типовете на фактическите аргументи (при извикването) с типовете на формалните такива на всички функции, именовани с f ;
- Активира функцията, чиито аргументи съответстват най-добре на текущото обръщение;
- Извежда грешка по време на компилирането, ако такава липсва или има противоречие.

Предефиниране на функция-действие

Пример:

Декларация:

```
void print(double dArg);
void print(long lArg);
```

Код на обръщението:

```
print(2L); //извиква print(long)-> константа от тип
                  // long
print(2.0); //извиква print( double )-> константа от
                   //тип double
print(2);     // противоречие поради двусмислие:
                  // print(double) или print(long)
```

Предефиниране на функция-действие

Търсене на функцията за изпълнение:

- *Точно съвпадение, без никакви или с тревиални превръщания :*
  - *име на масив към указател;*
  - *име на функция към указател към функция);*
- *Съвпадение чрез разширяване на цялочислени типове, например:*

bool => int;
char => int;
short => int;
unsigned варианти на горните (char,short);
float => double;

Предефиниране на функция-действие

- *Съвпадение след стандартни превръщания :*

int=>double;

double=>int;

double=>long;

long=>double;

int=>unsigned int;

Указател към тип T към указател void: (T => void*);*

Указател към производен клас към указател към базов клас: (Производен => Базов*);*

- *Съвпадение след дефинирани от програмиста превръщания на типове;*
- *Съвпадение след използване на многоточието от декларацията на функцията.*

Предефиниране на функция-примери

Примери:

```
void print(char cArg);
void print(int iArg);
void print(long lArg);
void print(double dArg);
void print(const char * pArg);
void print(const void * pArg);
// Добавяне на копиращ конструктор към класа CPerson
class CPerson {
public:
//..........
   CPerson(const CPerson*  pCPerson) {
     m_Name= pCPerson->m_Name;
     m_Gender=pCPerson->m_Gender;
    }
//..........
};
```

Предефиниране на функция-примери

```
void f( char cArg, int iArg, short sArg, float fArg )
{
  print(cArg); //точно съвпадение: print(char)
  print(iArg); //точно съвпадение: print(int)
  print(sArg); //разширение на цял тип: print(int)
  print(fArg); // разширение float double: print(double)
  print('a'); //точно съвпадение print(char)
  print(78); //точно съвпадение print(int)
  print(0); //точно съвпадение print(int)
  print("b");// точно съвпадение print(const char *)
  unsigned char pToChar[4] = {'a','b','c','d'}; // разширение
        unsigned char* към void*,
  print(pToChar);  // print(const void *);
 CPerson oPerson2(new CParent());
//указател към производен клас =>  указател към базовия клас
}
```

2. Предефиниране на оператори

Дефиниция:

Предекларирane на оператор за даден клас означава да се състави функция, която да се изпълнява, когато един или повече обекти от този клас се подложат на съответния дефиниран оператор.

Операторните **декларации** са разширение на съществуващите в езика оператори върху новосъздадените обекти от класове.

2. Предефиниране на оператори

Примери:

Операциите на езика C++ **“=”, “+”, “-”, “*”, “/”,“+-”, “==”** и други имат общоприето значение в синтаксиса на езика.

При създаването на нови класове:

- Семантичното значение се пренеася върху обектите на класа;
- Осъществява се със съответната **дефинирана** от създателя на класа функция.
- **Предефиниране на оператора** е създадената от програмиста функция, която се изпълнява за (върху) обектите от новосъздадения клас.

Предефиниране на оператори

Пример:

Бинарната операция сума (математическа операция).

- Предекларииране върху класа string;
- Имплементиране като съединяване (конкатенация) на обектите от класа (низове):

```
string str1("aaa") ;
string str2("bbb") ;
string str3 = str1+str2; // aaabbb
```

Предефиниране на оператори

За всеки от създадените от програмиста класове могат:

- Да се предекларират общоизвестните оператори;
- Да се декларират напълно нови като символна последователност и начин на действие оператори.

Предефиниране на оператори

Примери:

Операторите (+, -, *...) за сумиране, изваждане, умножение **на обекти** с числа или помежду им. За целта се дефинира функционалност, съответстваща на тези операции, обработваща член променливите на класа.

Сравнителни оператори (==, !=, <=...) за сравнение с логически резултат.

Предефиниране на оператори

Ограничения:

Не могат да се предефинират като оператори символите:

“.” (точка) достъп до член чрез име на обекта;

“.*” (точка звезда) указател към член на обекта;

“::” (двойно двуеточие) за определяне обхват;

“?:” (въпросителна друеточие) алгебричен условен оператор.

Предефиниране на оператори

Общи синтактични правила при декларирането на оператори

- Не може да се използват други символи за предекларииране на съществуващите оператори. Използват се само тези, които се използват в C++;
- Предекларираният оператор запазва семантиката на действието си от този, който предекларира:
  - Асоциативността;
  - Приоритета.
  - се възприемат от този, който е предекларииран.
- Операторът не е комутативен, докато не се дефинира като такъв;

Предефиниране на оператори

Общи синтактични правила при декларирането на оператори

- Всички функции за предекларирaне на оператори се наследяват с изключение на оператор за присвояване “=”;
- Всички оператори се дефинират самостоятелно (като отделен низ от символи). Например дефинирането на оператори “-” и “=” не се пренася автоматично върху оператора “-=”;
- Операторите не могат да се реализират като статични член функции, освен операторите **new** и **delete** които са статични за езика C++.

Предефиниране на оператори-видове

Унарни оператори

Могат да се предекларират операторите + и – (в семантиката на знакови оператори на числа, а не в семантиката на сумиране/изваждане).

Синтаксис за оператор +:

ТипНаРезултата Тип::operator+();

където **ТипНаРезултата** е връщаната стойност от предекларираната функция на оператора **+;**

Тип е типът на операнда. **Борави с членовете на класа.**

Или:

friend ТипНаРезултата operator+()(Тип);

Борави с членовете на Тип.

Предефиниране на оператори

Декларацията на типа може (и трябва) да има и атрибути, като референция **&** (и константна спецификация - **const).**

При наличие на дефиниция-тяло на функцията, която се изпълнява при срещането на оператора може да се използва операцията върху обект от клас **Тип.**

Пример:

+ОбектОтКласа**Тип**;

Предефиниране на оператори

- ***Бинарни оператори***

Дефиниция

Бинарните оператори са операторите, които свързват два аргумента.

Биват два основни вида в съответствие с връщания резултат:

- *бинарни аритметични оператори на езика (+, -, *, /)*
- *бинарни оператори за сравнение (<,>,<=,>=,==,!=).*

Те се прилагат при изрази между обектите от даден клас, например:

*Обект1ОтКласа**Тип** + Обект2ОтКласа**Тип;***

Изчислението на изрази от тази форма се свежда до извикване на предефинираната функция, която се изпълнява. Функцията, предефинираща оператора връща резултат, например обект:

*ОбектРезултатОтКласа**Тип***

Предефиниране на оператори

Синтаксис за оператор +:

ТипНаРезултата ТипLeft::operator+(ТипRight);

Или

friend ТипНаРезултата operator+(ТипLeft,ТипRight);

където ТипНаРезултата е връщаната стойност от предекларираната функция на оператора +;

ТипLeft и ТипRight са типове на операндите.

Декларациите на типовете могат/трябва да имат и атрибути, като референция & и/или const.

Предефиниране на оператори

- **Оператор за присвояване**

Операторът за присвояване е подобен на конструктора за копиране с тази разлика, че той оперира със съществуващия обект, вместо да създава нов:

- Ако програмистът не предефинира този оператор, компилаторът генерира подразбиращ се такъв и генерира обръщение към него;
- При създаването на някои от класовете в предходните лекции има предекларирaн оператор за присвояване.

Предефиниране на оператори

Пример: За класа CPoint той е дефиниран по сл. начин:

```
const CPoint& operator =(const CPoint& xy) {
  ix = xy.ix;
  iy = xy.iy;
  return *this;
}
```

Предефиниране на оператори

Пример: Ако са създадени обектите xy1, xy2 от класа CPoint е възможно да се използва операцията:

xy1=xy2= CPoint(0, 0);

Действие:

Чрез експлицитният конструктор се създава обекта с нулеви член променливи- 0,0.

xy2 получава стойностите, които директно подменят стойностите на член променливите му в тялото на xy2.

Върнатата стойност е референция към обекта xy2;

По референцията на xy2 се прави обръщение към неговите член променливи, като получените стойности се присвояват на член променливите на обекта xy1 с левия оператор равенство.

Асоциативността повтаря известната от езика C:

xy1=(xy2= CPoint(0, 0));

Предефиниране на оператори

- **Оператори за нарастване и намаляване**

Въпреки че операторите ++ и -- са унарни и се подчиняват на правилата за унарните оператори е необходимо да се съставят двете функции-префисна и постфиксна:

- ***Префиксната версия*** първо променя стойността и след това я връща. Трябва да се връща референция към обекта от класа (*this), тъй като стойностите на обекта (след прилагане на нарастване/намаляване) и на резултата са еднакви;
- ***Постфиксната версия*** връща стойността на операнда преди промяната му и затова не връща референция. Тази версия изисква фиктивен аргумент от тип int, на който се предава стойност 0 по време на обръщението. Този аргумент разграничава дефинициите към различните функции по време на обръщението.

Предефиниране на оператори

```
class CPoint {
private:
int m_x, m_y;
public:// необходимите конструктори get/set функции
   CPoint& operator++() { // префиксен
    m_x++;
    m_y++;
    return *this; // връща референцията след промяната
  }
  CPoint  operator++(int dummy) {//постфиксен
    CPoint point(*this);
    m_x++;
    m_y++;
    return point; // връща се копие на обекта преди промяната
  }
};
CPoint oP1;
  oP1++;   ++oP1;
```

Предефиниране на оператори

- **Индексен оператор([])**

Реализацията му се налага когато се имплементира собствен клас за масиви или контейнери (динамични масиви):

- Функцията operator[] има аргумент цяло число и се имплементира като позицията (указател или референция) по нулево базиран индекс на предадения аргумент;
- Ако се върне референция, операторът може да се използва в двете страни на операция присвояване.

Предефиниране на оператори

Пример:

```
template <class T>
class Cvector {
  T* m_pData;
  int m_iCapacity;
  int m_iSize;
public:
  Cvector (int s=20) {
    if(m_pData = new T [m_iCapacity = s]) m_iSize = 0;
    else throw INVALID_CAPACITY;
  }
  ~Cvector () { delete[] m_pData; }
  T& operator[] (int intIndex) {
   if(intIndex>=m_iCapacity) throw INVALID_INDEX;
     if( intIndex >= m_iSize ) m_iSize=intIndex+1;
     return m_pData[intIndex];
  }
  int get_size() { return m_iSize; }
  int get_capacity() { return m_iCapacity; }
};
```

Предефиниране на оператори

Пример за тестова програма:

```
void main() {
    try {
// тест на вектор от int
    Cvector<int> oVInt;
      for(int i=0; i< oVInt.get_capacity(); i++ )  oVInt[i] = i;
      for(i=0; i<oVInt.get_size(); i++ )  cout << oVInt[i] << endl;
    }
    catch(int i) {
       switch( i ) {
       case INVALID_INDEX:  cout << "Invalid Index .... " <<  endl;       break;
       case INVALID_CAPACITY: cout << "Invalid Capacity... " <<  endl; break;
    }
   }
}
```

Предефиниране на оператори

Пример 2 за тестова програма:

```
void main() {
    try {
// тест на вектор от string
      string str1("str1");     string str2("str2");     string str3("str3");
      Cvector<string> oVStr;
      oVStr[0] = str1;     oVStr[1] = str2;     oVStr[2] = str3;
      for(i=0; i<3; i++ )   cout << oVStr[i]<<endl;
   }
   catch(int i) {
       switch( i ) {
       case INVALID_INDEX:  cout << "Invalid Index .... " <<  endl;       break;
       case INVALID_CAPACITY: cout << "Invalid Capacity... " <<  endl; break;
     }
   }
}
```

Предефиниране на оператори

Пример 3 за тестова програма:

```
void main() {
   try {
// тест на вектор от CStudent
    CStudent oSt1("Ivan Petrov", "008614", male);
    CStudent oSt2("Ivanka Petrova", "008624", female);
    CStudent oSt3("Petar Ivanov", "008634", male);
Cvector<CStudent> oVStudents(20);
    oVStudents[0]=oSt1;
    oVStudents[1]=oSt2;
    oVStudents[2]=oSt3;
for(i=0; i<3; i++ ) cout << oVStudents[i];
  }
  catch(int i) {
    switch( i ) {
      case INVALID_INDEX:  cout << "Invalid Index .... " <<  endl;       break;
      case INVALID_CAPACITY: cout << "Invalid Capacity... " <<  endl; break;
      }
  }
}
```

Предефиниране на оператори

- **Оператор обръщение към функция()**

Целта на този оператор е да позволи на обекта от класа да се използва като име на функция. Този оператор може да се дефинира с няколко функции (предекларирани):

Пример:

```
class Cprint {
public:
  int operator()(long lArg);
  int operator()(char *szArg, int iLen);
};
// използване:
Cprint oPrint;
oPrint("abc", 3);
oPrint(2L);
```

Предефиниране на оператори

Варианти на предефиниране на оператори:

- Чрез дефиниране на член функция към класовата декларация;
- Чрез глобална приятелска функция.

Примери:

```
Rectangle operator +(const Rectangle& obj) const {
   return Rectangle(getIX1()+obj.calcArea(),0,0,getIX1()+obj.calcArea());
}
friend Rectangle& operator+(Rectangle& oRes, const Rectangle& toObj){
     oRes.iX1=(oRes.iX1+toObj.calcArea());
     return oRes;
}
```

Разлика между двете предефиниции!!!

3. Шаблони (Templates)

Дефиниция:

В C++ *templates* се използват за дефиниране на генерируеми функции и класове.

Генерална (*generic)* функция или клас е такава функция или клас, която приема като актуален параметър *тип*.

Цел и предназначение на шаблонни декларации

- Създава фамилия от функции или класове, които се компилират от един и същ код, но работят с различни класове или типове данни.

Шаблони

Пример:

Дефиниция на функция:

```
int Max (int x, int y)
    { return x > y ? x : y; }
```

Шаблонна дефиниция:

```
template <class T>
T Max (T x, T y)
    { return x > y ? x : y; }
```

Шаблони

Действие:

Шаблонната дефиниция създава фамилия от функции Max, по една за всеки различен тип T:

- ✓ C++ компилаторът създава съответстваща функция Max когато се срещне фактическо обръщение.

Шаблони

Пример за използване:

```
int i = Max (1, 2);
double d = Max (1.0, 2.0);
```

Създават се две функции Max - с аргументи от тип int и с аргументи double.

Шаблони

Шаблоните могат да се използват за дефиниране на генерируеми класове:
Пример – известната дефиниция на клас CStack:
Декларацията на клас стек за цели числа:

```
class CStack
{
    // ...
public:
    void push (int);
    int pop ();
};
```

определя CStack като клас за цели числа. Член функциите push и pop се използват за запис и извличане на int от стека.

Шаблони

Пример: Програма, реализирана на C++ за стекова реализация чрез класа CStack, работещ с шаблонен тип данни:

```
// декларация, определяща T като шаблонен клас
template <class T>
struct ptr{T key; ptr *next;}; // вместо тип се използва T
// Навсякъде вместо тип се използва шаблонен
//идентификатор (T)
template <class T>
class CStackT {
ptr<T> *sp; // шаблонен указател
public:
CStackT() {sp = NULL;}
 ~CStackT();
void push(const T &ky); // шаблонна декларация на push
int pop(T &x); // шаблонна декларация на функцията pop
}; // край на класа
```

Шаблони

```
// Функцията е с шаблонен тип вместо char
template <class T>
void CStackT<T>::push(const T &ky)
{  ptr<T> *p = sp;
   sp = new ptr<T>;
   sp->key = ky; sp->next = p;
}
// Операцията е същата, както в C++ кода, с шаблонен тип вместо char
template <class T>
int CStackT<T>::pop(T &x)
{  ptr<T> *p;
  if (sp)
  {  x = sp->key;
     p = sp; sp = sp->next;
     delete p;
     return 1;
  }  else return 0;
}
```

Шаблони

Използване на шаблонен клас:

```
CStack<int> s1; // генериране на конкретен клас int
s1.push (1);
CStack<double> s2; // генериране на клас double
s2.push (1.0);
```

Класовете изискват задължителен шаблонен спецификатор!

Шаблони

```
int main()
{  char ch; // променлива за входните символи
   CStack<char> c_st; // стекова променлива с дефиниран тип char
   cout << "Enter some characters followed by a "
           "character *:\n";
   try
   {  while (cin >> ch) {
              if(ch!='*') c_st.push(ch);
              else break;
      }
   }
   catch( ... )// защита срещу грешки, напр. недостиг на памет
   {  cout << "Stack overflow\n";
   }
   cout << "\nItems popped from char stack:\n";
   while (c_st.pop(ch)) cout << ch << " ";
   cout << endl;
   return 0;
}
```

Шаблони-специализация

Дефиниция:

Шаблонна специализация позволява на шаблоните да създават специфична имплементация, когато шаблонът е от определен тип.

Синтаксис:

template <> class *име_на клас* *<специализиращ_тип>*

Шаблони-специализация

Пример за създаване на шаблонен клас-обща дефиниция:

```
template <class T>
class CPair {
    T value1, value2;
  public:
    CPair (T first, T second)
      {value1=first; value2=second;}
    T module () {return 0;}
};
template <>
class CPair <int> {// специализация на класа за целочислен тип
    int value1, value2;
  public:
    CPair (int first, int second)
      {value1=first; value2=second;}
    int module ();
};
```

Шаблони-специализация

```
// Специализация на член функцията,
// дефинирана извън класа. Пример:
template <>
int CPair<int>::module() {
  return value1%value2;
}
```

Клас функциите, дефинирани извън класовете изискват задължителен шаблонен спецификатор!

Шаблони-специализация

Пример за използване:

```
int main () {
  CPair <int> myints (100,75);
  CPair <float> myfloats (100.0,75.0);
  cout << myints.module() << '\n';
  cout << myfloats.module() << '\n';
  return 0;
}
```

Класовете се генерират по шаблонна специализация (int) и по обща шаблонна дефиниция (float)!

Въпроси?