Проектиране и Тестиране на Софтуер
ТУ, кат. КС, летен семестър 2012

Лекция 4b

Тема:

ООП - Полиморфизъм

ООП – Конвертиране на данни

Част 1

ООП
Полиморфизъм

Съдържание:

- Ранно свързване и късно свързване
  - Нормални член функции и виртуални член функции
  - Методи, достъпни с указатели
- Предефинирани операции
- Приятелски функции

Предефинирани Операции
Overloaded Operators

Въведение

Каква е ползата от предефиниране на операциите?

- По-добра четливост

Примери:

- Клас *Counter*
- Клас *Distance*

Предефиниране на едноместни операции

Предефиниране на двуместни операции

Приятелски Функции
Friend Functions

Приятелски функции

- Идея: Да се осигури достъп на функции не-членове на клас до данни на обекти, които са с квалификатор *private* или *protected*.
- Припомняне: Горната идея противоречи на принципа Капсулиране/Скриване на данни
- Има ситуации, когато скриването на данни води до редица неудобства.

Част 2

ООП
Конвертиране на Данни

Съдържание:

- Присвояване и конвертиране на данни:
  - Присвояване с еднакви по тип данни
  - Конвертиране м/у основни типове данни
  - Конвертиране м/у обекти и основни типове
  - Конвертиране между обекти от различни класове
- Конструктори:
  - Подразбирани
  - Дефинирани от потребителя
  - Конструктор за копиране и конструктор за присвояване

Присвояване и конвертиране на данни

- Присвояване с еднакви по тип данни (основни типове или потребителски типове данни)
- Присвояване и конвертиране между различни основни типове данни;
- Присвояване и конвертиране между обекти (потребителски деф.) и основни типове;

From Basic type to User Defined type	From User Defined Type to Basic type
From float to Distance From meters to feet/inches	From Distance to float From feet/inches to meters

Операция присвояване

оператор за присвояване

Присвояване с еднакви по тип данни

- = присвоява една променлива на друга променлива или израз на променлива

  intvar1 = intvar2; *intvar3 = 5 + 6 * 8;*

- = присвоява стойност(и) от един обект (потреб. Деф. тип) на друг обект, като и двата обекта са от един и същ клас.

  dist1 = dist2; *dist3 = dist4 + dist5;*

Присвояване с еднакви по тип данни

- Когато един обект се присвоява на друг обект, неговите член данни се копират ***simply copied*** в новия обект. Компилаторът е инструктиран да прави това.
- Присвояване с еднакви по тип данни (основни типове или класове) се поддържа от компилатора стига данните отляво и отдясно на знака = да са от един и същ тип.

Присвояване с еднакви по тип данни

- Основни (вградени) типове данни:

  int var1, var2=35; *float v3, v4=3.4;*

  var1 = var2; *v3 = v4;*

- Потребителски дефинирани (абстрактни) типове данни:

  Distance dist1, dist2(10, 8.4);

  dist1 = dist2;

Конвертиране м/у основни типове данни

Conversions between Basic Types

Конвертиране между различни основни типове

Този тип конвертиране се среща в оператори като

intvar = floatvar;

Предполага се, че компилаторът ще вгради ф-ия за преобразуване от плаваща запетая във фиксирана запетая, за да присвои цяло на intvar. Други подобни: *int* to *double*, *char* to *float, float* to *double, char* to *int.*

За всяко преобразуване компилаторът вгражда съответна преобразуваща функция, когато от двете страни на знака = се срещат данни от различен базов тип.

Конвертиране между основни типове

- Конвертиране като на предния слайд, е неявно (***implicit)***
- Конвертиране от един тип в друг тип може да се обяви и явно (***explicitly).*** Това се постига по два начина:
  - Операция <тип> (C, C++)
  - Функционален запис (C++)
- Явното и неявното конвертиране се реализира от едни и същи вградени функции за преобразуване.

Конвертиране между основни типове

Пример:

int vari; *float varf = 2.78*;

Целим да присвоим *varf* на *vari*.

Преобразуване се предизвиква по сл. 4 начина:

1/ неявно *vari = varf;*

2a/ явно с операция (тип)

vari = (int)varf;

2b/ явно с функционален запис (C++ само)

vari = int(varf);

2c/ явно с операция static cast

vari = static_cast<int>(varf);

2c пример пояснение

Преди Standard C++, cast операциите се записваха в следния формат:

vari = (int)varf;

vari = int(varf);

Този вид формат прави трудни за забелязване и редактиране с текстов редактор

vari = static_cast<int>(varf);

Новият формат е ясно забележим.

Старите формати се поддържат, но не се препоръчват.

Повече за casts в C++

- Следните операции casts в Standard C++:
  - Static casts – Лафоре 58-60
  - Dynamic casts – Лафоре 553
  - Reinterpret casts – Лафоре 591
  - Const casts – Лафоре 860
- Подробности в лекцията сл. Седмица.

Динамични casts

- Възможно е да се извлича инфо за класа на обекта (an object's class) и дори да се променя класа на обекта по време на изпълнение
- Два начина: *dynamic_cast* и *typeid*

Проверка типа на клас

- *Dynamic_cast* позволява проверка типа на клас с предпоставката, че класовете, чийто обекти се проверяват, са породени от общ базов клас.

Смяна тип на указател

- dynamic-cast операция позволява смяна типа на обекта в рамките (upward and downward) на йерархията от класове, обвързани с релацията наследяване.
- R.Lafore, 555

The *typeid* operator

- Sometimes you want more info about an object than simple check that it's of a certain class. You can obtain info about the type of an unknown object, such as its class name, using the *typeid* operator

Reinterpret casts

Ако се налага промяна типа на указател по неясни причини, ползвайте *reinterpret* cast.

```
int intvar;
float flovar;
int* ptrint;
float* ptrflo;

// applying re interpret cast
ptrint = reinterpret_cast<int*>(flovar);
ptrflo = repinterpret_cast<float*>(intvar);
```

Const casts

- No info

Конвертиране между обекти и осн. типове

Conversions between Objects and Basic Types

Конвертиране м/у обекти и осн. типове

- Този тип конвертиране не може да се постигне с вградени от компилатора преобразуващи функции.
- Съставянето на тази категория преобразуващи функции е отговорност на програмиста.

От основен тип д. >към> Потребителски
float >към> Distance
meters >към> feet + inches

Конвертиране м/у обекти и осн. типове

От основен тип д.	>към>	Потребителски
float	>към>	Distance
meters	>към>	feet + inches

Distance d1(2.35f), d2 = 4.15f, d3, d4;
float m1 = 10.05f;

d3 = 8.16f; *d4 = m1;*

Решение: 1-аргументен конструктор или предефинирана = операция

From Basic type to User defined Type
Решение: 1-аргументен конструктор

```
Distance (float meters)  // one-arg constructor
                         {
                          float fltfeet = meters * MTF;
                          feet = int(fltfeet);
                          inches = 12 *(fltfeet - feet);
                         }
```

Такъв конструктор се активира при създаване на обект с един аргумент (стойност в метри) в случаи като следните декларации

Distance dist1(3.45f), dist2=2.35f;

From Basic type to User defined Type Решение: предефинирана = операция

Още един случай за същото конвертиране

dist1 = 1.0f;

dist1.operator=(1.0f);

Тук конвертирането *float* в *Distance* става при присвояване.

Компилаторът търси и намира предефинирана = операция, която конвертира *float* в *Distance*.

```
void operator=(float meters)  // предефнирана = операция
                              {
                               float fltfeet = meters * MTF;
                               feet = int(fltfeet);
                               inches = 12 *(fltfeet - feet);
                              }
```

From Basic type to User defined Type

Още един случай за същото конвертиране

dist1 = 1.0f;

Тук конвертирането float to Distance става без явно създаване на нов обект от клас *Distance.*

Компилаторът първо търси предефинирана = функция. Ако не намери, той отново търси и намира 1-арг конструктор да конвертира float to Distance. Прилага го на оператора за присвояване като създава анонимен (anonymous unnamed temporary) *Distance* обект, чийто feet/inches член данни съответстват на 1.0f метър. Тези стойности за feet/inches се копират в съответните feet/inches на обекта *dist1*.

Така се размива разликата м/у дефиниция и присвояване. Ако няма предефинирана операция =, компилаторът търси конструктор, за да осигури необходимото конвертиране.

От Потребителски	>към>	основен тип
Distance	>към>	float
feet + inches	>към>	meters

Конвертиране м/у обекти и осн. типове

От Потребителски >към> основен тип

Distance >към> float

feet + inches >към> meters

```
Distance dist1(6, 8.3);          float m1, m2, m3;

m1 = dist1;                      // implicit conversion
m2 = (float)dist1;               // explicit conversion
m3 = float(dist1);               // explicit conversion
```

Решение: конвертираща функция

From User defined Type to Basic type
Решение: конвертираща функция

```
operator float()        // conversion function
    {
     float fracfeet = inches/12;
          fracfeet += float(feet);
          return fracfeet / MTF;
    }
```

Методът предефинира унарна (тип) операция и затова няма формален параметър (Правилото).

Член данните (feet/inches) на Distance обекта, чийто метод се активира, са достъпни, преобразуват се като метри и се връщат като операнд на оператор *return*.

From User defined Type to Basic type
Решение: конвертираща функция

Предефинираната операция се активира явно:

mtrs = float(dist2); *mtrs = (float)dist2;*

Или неявно:

mtrs = dist2;

Ефектът е един и същ. Когато компилаторът открие необходимост от преобразуване от клас към основен тип, той първо търси предефинирана операция =. Ако не я намери, той ползва съответна преобразуваща функция.

Пример с клас Distance

- Демо програма

  oop4aDistance.cpp

  oop4aDistance.exe

Конвертиране м/у обекти от различни Класове

Конвертиране между обекти от различни класове

- Как?
- Като се ползва подход подобен на преобразуването между обекти и основни типове
  - 1-аргументен конструктор
  - Преобразуваща функция
- Изборът зависи дали ще се добави метод в дефиницията на класа източник (source object, sender, r-value) или в дефиницията на класа приемник (destination object, receiver, l-value)

Конвертиране между обекти от различни класове

class A { . . . };

A obja;

class B { . . . };

B objb;

<destination>	=	*<source>*
obja	=	*objb;*

Два вида
2-D координатна система:

ПРАВОЪГЪЛНА & ПОЛЯРНА

Конвертиране между обекти от различни класове

```
class Rect { . . . };
        Rect  rec1,   rec2(3.0, 4.0);

class Polar { . . };
        Polar pol1(10.0, 0.7858), pol2;

// присвояване и конвертиране на данни
        rec1 = pol1;
        pol2 = rec2;
```

Пример

- От Полярни към Правоъгълни координати
  - class Polar { . . . }; // source class(object)
  - class Rect { . . . }; // destination class(object)

- Polar pol(10.0, 0.785398); Rect rec;

<dest> = <source>

rec = pol;

Конвертиране м/у обекти от разл. класове: Доб в Source клас(преобразуваща ф-ия)

```
Rect rec;
Polar pol1(10.0, 0.785398);  // ъгъл 45° = 3.1415/4.

rec = pol1;   // Polar >  Rect  неявно конвертиране

Polar pol2(10.0, 3.141592/6.); // ъгъл 30°
rec = Rect(pol2);      // Polar  >  Rect  явно конвертиране

Polar pol3(10.0, 3.141592/3.);  // ъгъл 60°
rec = (Rect)pol3;      //  Polar  >  Rect  явно конвертиране
```

Конвертиране м/у обекти от разл. класове: Доб в Source клас(преобразуваща ф-ия)

```
class Rect  { private: double xco, yco;     // x, y правоъг. Коорд.
public:  Rect() {xco = yco = 0.0; }
    Rect(double x, double y) {xco = x;  yco = y;   }
    void ShowData() {. . .}
};
class Polar{private: double radius, angle; // полярни коорд.
public:  Polar() {radius = angle = 0.0; }
    Polar(double r, double a) {radius = r;  angle = a;}
    void ShowData(){ . . .}
    operator Rect(){      double x = radius * cos(angle);
                          double y = radius * sin(angle);
                          return Rect(x,y); }
};
```

Конвертиране м/у обекти от разл. класове: Доб в Source клас(преобразуваща ф-ия)

Демо:

Първичен текст: OOP4cPolarToRect1.cpp

Изпълним: OOP4cPolarToRect1.exe

Конвертиране м/у обекти от разл. класове: Доб в Dest клас(1-арг конструктор)

```
Rect rec;
Polar pol1(10.0, 0.785398);  // ъгъл 45° = 3.1415/4.

rec = pol1;   // Polar  >  Rect конверт при присвояване

Polar pol2(10.0, 3.141592/6.); // ъгъл 30°
Rect rec2(pol2); // Polar > Rect конверт при създ на обект

Polar pol3(10.0, 3.141592/3.);  // ъгъл 60°
Rect rec3=pol3; // Polar > Rect конверт при създ на обект
```

Конвертиране м/у обекти от разл. класове: Доб в Dest клас(1-арг конструктор)

```
class Polar{private: double radius, angle;//   polar coordinates
public:    Polar() {  radius = angle = 0.0;                                    }
     Polar(double r, double a) {          radius = r;  angle = a;          }
     void ShowData(){ . . .}
     double getrad() { return radius; }    double getang() { return angle;  }
};
class Rect{private: double xco, yco;// rectangle coordinates
public:    Rect() {   xco = yco = 0.0;     }
     Rect(double x, double y) {xco = x;  yco = y;   }
     Rect(Polar p) {
                    double rad = p.getrad();      double ang = p.getang();
                    xco = rad * cos(ang);         yco = rad * sin(ang);          }
     void ShowData() { . . .}
};
```

Конвертиране м/у обекти от разл. класове: Доб в Dest клас(1-арг конструктор)

Демо:

Първичен текст: OOP4dPolarToRect2.cpp

Изпълним: OOP4dPolarToRect2.exe

Присвояване и Конвертиране на Данни

Data Types Conversions		
	Routine in Destination	**Routine in Source**
from Basic type to Basic type	**Built-in conversion functions**	
from Basic to Class	**Constructor**	**NA**
from Class to Basic	**NA**	**Conversion function**
from Class to Class	**Constructor**	**Conversion function**

Два вида време Time

цикъл 12-часа и цикъл 24-часа

Примери

12-Hour Time	**24-Hour Time**
12:00 a.m. (полунощ)	00:00
12:01 a.m.	00:01
6:00 a.m.	06:00
11:59 a.m.	11:59
12:00 p.m. (пладне)	12:00
12:01 p.m.	12:01
6:00 p.m.	18.00
11:59 p.m.	23:59

Конвертиране: кога и какво?

- В случай на достъп и до двата класа (източник и приемник), изборът зависи от вас
- Има предрешени случаи, например
  - Закупена библиотека от класове само с обектен код, без достъп до първичния текст на класовете

Заключение

- Избягвайте не еднозначности, като
  - Осигурите две различни възможности с преобразуваща функция и 1-арг конструктор за конвертиране като примера (Polar > Rect)
  - Няма критерий за компилатора кой конверт да ползва и следва грешка.
  - Препоръка: да не се прави един конверт по два различни начина. Последните демо примери бяха в две отделни програми, а не обединени в една обща програма.

Допълнение за Конструктори

Конструктори/Деструктори

Да разгледаме клас X с два член данни

```
class X {        private: int x,y;
                 public:  X(int, int);
                 . . .                    };
```

Възможни са 3 начина за дефиниране на к-тор с два параметъра:

X::X(int a, int b) *{ x=a; y=b; }*
X::X(int a, int b) : x(a) *{ y=b; }*
X::X(int a, int b) : x(a) , y(b) *{ }*

Конструктори/Деструктори

```
class Distance
{ private: const float MTF; int feet; float inches;
  public: Distance()
                { MTF = 3.2808; feet=0; inches=0.0; }
// ИЛИ
        Distance(): MTF(3.2808), feet(0), inches(0.0)
                { }
  . . .
};
```

Конструктори по подразбиране

- Въведение в :
  - Конструктор за копиране;
  - Конструктор за присвояване.

Демо програма

oop4b.cpp

Конструктор за копиране и Конструктор за присвояване

- При декларация на клас без конструктор, системно се поддържа без-арг конструктор

  X::X() { . . . } *CL::CL() { . . . }*

- При дефиниция на обект, чийто член данни се инициализират с (зависят от) член данните на друг обект от същия клас, се активира друг системно поддържан (вграден) конструктор (**copy constructor)** или предефинираната му версия (**assignment constructor**)

  X::X(X &) *CL::CL(CL &)*

Чете се: X of X ref

Конструктор за копиране

```
class CL {        private: int x, y;
                  public:  CL(); . . .          };

// user specified constructor
CL::CL(){         cout<<“\nEnter x,y:”;  cin>>x>>y;     }

void main()
 {  CL  obj1;                  // user specified constructor
    CL  obj2=obj1;             // default copy constructor
    CL  obj3(obj1);
   . . .
  }
```

Конструктор за присвояване

```
class CL {          private: int x, y;
                    public:  CL();  CL(CL &);  . . .       };

// user specified constructor
CL::CL(){           cout<<“\nEnter x,y:”;      cin>>x>>y;       }
// user specified assignment constructor
CL::CL(CL &p) {             x = p.x +1;
                            y = p.y +2;          }

void  main()
 {  CL   obj1;                  // user specified constructor
    CL  obj2=obj1;              // user specified assignment constructor
    CL  obj3(obj1);
   . . .
 }
```

Демо програма

Програма за илюстрация:

подразбирани конструктори,

конструктор за копиране,

конструктор за присвояване

oop4b.cpp oop4b.exe

Благодаря
За
Вниманието