Наследяване. Конструктор за присвояване и операторна функция за присвояване на производен клас. Преобразуване на типове

гл.ас., д-р. Нора Ангелова

Единично наследяване. Конструктори, деструктор, конструктор за присвояване и операторна функция за присвояване на производен клас

Конструктор за присвояване и операторна функция за присвояване

- В общия случай, производният клас **не** наследява от основния си клас конструктора за присвояване и оператора за присвояване.

Конструктор за присвояване

- Конструкторът за присвояване на производния клас инициализира собствените член-данни на класа
- Конструкторът за присвояване (или друг конструктор) на основния клас инициализира наследените член-данни.

- Конструкторът за присвояване на производен клас се дефинира по аналогичен начин като обикновените конструктори на производни класове.

  Инициализаторът се задава чрез единствения явно указан формален параметър const <име_на_клас>&.

```
<име_на_клас>::<име_на_клас>(const <име_на_клас>&)
<инициализиращ_списък>
{ <тяло>
}
```

КОНСТРУКТОР ЗА ПРИСВОЯВАНЕ

- В производния клас НЕ е дефиниран конструктор за присвояване

Възможни са:

а) в основния клас е дефиниран конструктор за присвояване

В този случай компилаторът генерира конструктор за копиране на производния клас, който преди да се изпълни, активира и изпълнява конструктора за присвояване на основния клас.

В случая се казва, че конструкторът за присвояване на основния клас се наследява от производния клас.

б) в основния клас не е дефиниран конструктор за присвояване

В този случай в основния и в производния му клас се генерират конструктори за копиране.

Конструкторът за копиране на производния клас активира конструктора за копиране на основния клас.

КОНСТРУКТОР ЗА ПРИСВОЯВАНЕ

- В производния клас E е дефиниран конструктор за присвояване

Дефиницията на конструктора за присвояване на производния клас определя как точно ще се инициализира наследената част.

В неговия инициализиращ списък може да има или да няма обръщение към конструктор (за присвояване или обикновен) на основния му клас.

Препоръчва се в инициализиращия списък на производния клас да има обръщение към конструктора за присвояване на основния клас, ако такъв е дефиниран.

Забележка:

Ако не е указано обръщение към конструктор на основния клас, инициализирането на наследените членове се осъществява от подразбиращия се конструктор на основния клас. Ако основният клас няма подразбиращ се конструктор, се съобщава за отсъствието на подходящ конструктор.

Операторна функция за присвояване

- Операторната функция за присвояване на производен клас трябва да указва как да се осъществи присвояването както на собствените, така и на наследените си член-данни.
- За разлика от конструкторите на производния клас тя прави това в тялото си, т.е. не поддържа инициализиращ списък

Операторна функция за присвояване

```
<производен_клас>& <производен_клас>::operator=
(const <производен_клас>& p) {
 if (this != &p) {
  // Дефиниране на присвояването за наследените член-данни
  <основен_клас>::operator=(p);

  // Дефиниране на присвояването за собствените член-данни
  Del(); // разрушаване на онези собствени
  // член-данни на подразбиращия
  // се обект, които са разположени в ДП

  Copy(p); // копиране на собствените член-данни
  // на p в съответните член-данни на
  // подразбиращия се обект
 }
 return *this;
}
```

Операторна функция за присвояване

- В производния клас НЕ е дефинирана операторна функция за присвояване

Компилаторът създава операторна функция за присвояване на производния клас. Тя се обръща и изпълнява операторната функция за присвояване на основния клас (дефинираната или подразбиращата се), чрез която инициализира наследената част, след това инициализира чрез присвояване и собствените член-данни на производния клас.

Затова в този случай се казва, че операторът за присвояване на основния клас се наследява.

Операторна функция за присвояване

- В производния клас E е дефинирана операторна функция за присвояване

Тази член-функция трябва да се погрижи за присвояването на наследените компоненти. В тялото на нейната дефиниция трябва да има обръщение към дефинирания оператор за присвояване на основния клас, ако има такъв. Ако това не е направено явно, стандартът на езика не уточнява как ще стане присвояването на наследените компоненти.

В случая операторът за присвояване на основния клас не се наследява.

Задача

Да се дефинират класовете People, Student и PStudent

В класа Student член-данната fac_numb да е от тип char* и да се добави член-данна addr от тип char*, определяща адреса на студент.

Двете член-данни да се реализират в динамичната памет.

В класа PStudent да се добави член-данна workplace от тип char*, определяща месторабoтата на студент от платена форма на обучение и да се реализира в динамичната памет.

За всеки от класовете да се дефинира каноничното представяне.

Всеки клас на програмата съдържа две помощни член-функции – за копиране и за изтриване (разрушаване) на собствените им компоненти.

ЗАДАЧА

```
class People {
 public:
   // канонично представяне
   People(char* = "", char* = "");
   ~People();
   People(const People&);
   People& operator=(const People& p);
   // член-функция за извеждане
   void PrintPeople() const;
 private:
   char* name; // име
   char* ucn; // ЕГН
   // помощни член-функции за копиране и изтриване
   void copyPeople(char*, char*);
   void delPeople();
};
```

ЗАДАЧА

```
void People::copyPeople(char* nameStr, char* ucnStr) {
  name = new char[strlen(nameStr)+1];
  assert(name != NULL);
  strcpy(name, nameStr);
  ucn = new char[strlen(ucnStr)+1];
  assert(ucn != NULL);
  strcpy(ucn, ucnStr);
}

void People::delPeople() {
  delete [] name;
  delete [] ucn;
}
```

ЗАДАЧА

```
People::People(char* nameStr, char* ucnStr) {
  copyPeople(nameStr, ucnStr);
}

People::~People() {
  delPeople();
}

People::People(const People& p) {
  copyPeople(p.name, p.ucn);
}

People& People::operator=(const People& p) {
  if (this!=&p) {
    delPeople();
    copyPeople(p.name, p.ucn);
  }
  return *this;
}
```

ЗАДАЧА

```
void People::PrintPeople() const {
  cout << "Name: " << name << endl;
  cout << "UCN: " << ucn << endl;
}
```

ЗАДАЧА

```
class Student : public People {
public:
  // канонично представяне
  Student(char* = "", char* = "", double = 0, char* = "",
char* = "");
  ~Student();
  Student(const Student&);
  Student& operator=(const Student&);
  // член-функция за извеждане
  void PrintStudent() const;
private:
  double gpa; // среден успех
  char* facNumb; // факултетен номер
  char* addr; // адрес
  // помощни член-функции
  // за копиране и изтриване
  void copyStudent(double, char*, char*);
  void delStudent();
};
```

ЗАДАЧА

```
void Student::copyStudent(double gpaData, char* fNumb,
char* addrStr) {
  gpa = gpaData;
  facNumb = new char[strlen(fNumb)+1];
  assert(facNumb != NULL);
  strcpy(facNumb, fNumb);
  addr = new char[strlen(addrStr)+1];
  assert(addr != NULL);
  strcpy(addr, addrStr);
}

void Student::delStudent() {
  delete [] facNumb;
  delete [] addr;
}
```

ЗАДАЧА

```
Student::Student(char* nameStr, char* ucnStr, double gpaData, char*
fNumb, char* addrStr) : People(nameStr, ucnStr) {
  copyStudent(gpaData, fNumb, addrStr);
}

Student::~Student() {
  delStudent();
}

Student::Student(const Student& st) : People(st) {
  copyStudent(st.gpa, st.facNumb, st.addr);
}



Student& Student::operator=(const Student& st) {
  if (this != &st) {
    People::operator=(st);
    delStudent();
    copyStudent(st.gpa, st.facNumb, st.addr);
  }
  return *this;
}
```

ЗАДАЧА

```
void Student::PrintStudent() const {
  PrintPeople();
  cout << "GPA of student: " << gpa << endl;
  cout << "Fac. nomer: " << facNumb << endl;
  cout << "Address: " << addr << endl;
}
```

ЗАДАЧА

```
class PStudent : public Student {
  public:
    // канонично представяне
    PStudent(char* = "", char* = "", double = 0,
char* = "", char* = "", double = 0,
char* = "");
    ~PStudent();
    PStudent(const PStudent&);
    PStudent& operator=(const PStudent&);
    // член-функция за извеждане
    void PrintPStudent() const;
  private:
    double fee; // такса
    char* workplace; // месторабота
    // помощни член-функции за копиране и изтриване
    void copyPStudent(double, char*);
    void delPStudent();
};
```

ЗАДАЧА

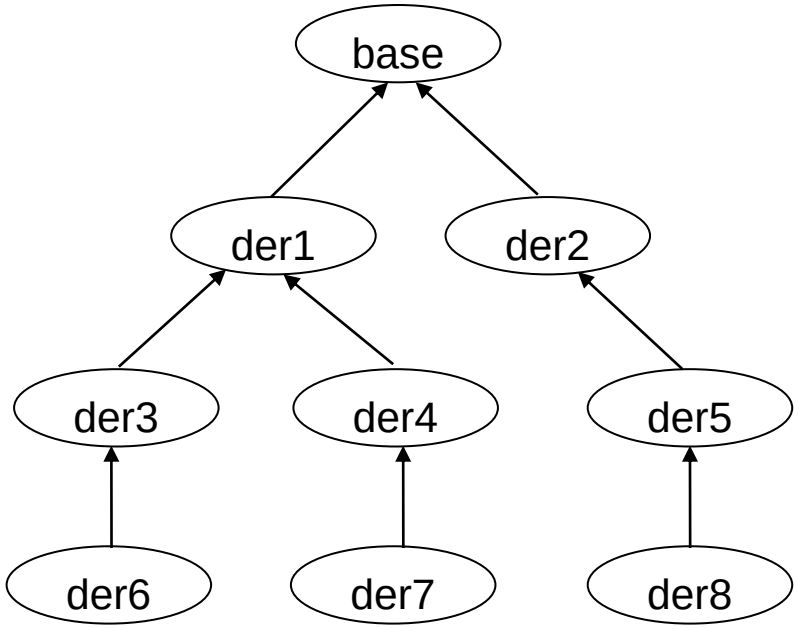
```
void PStudent::copyPStudent(double feeData, char*
workplaceStr) {
  fee = feeData;
  workplace = new char[strlen(workplaceStr)+1];
  assert(workplace!= NULL);
  strcpy(workplace, workplaceStr);
}

void PStudent::delPStudent() {
  delete [] workplace;
}
```

ЗАДАЧА

```
PStudent::PStudent(char* nameStr, char* ucnStr, double gpaData,
char* fNumb, char* addrStr, double feeData, char* workplaceStr)
: Student(nameStr, ucnStr, gpaData, fNumb, addrStr) {
  copyPStudent(feeData, workplaceStr);
}

PStudent::~PStudent() {
  delPStudent();
}

PStudent::PStudent(const PStudent& ps) : Student(ps) {
  copyPStudent(ps.fee, ps.workplace);
}

PStudent& PStudent::operator=(const PStudent& ps) {
  if (this != &ps) {
    Student::operator=(ps);
    delPStudent();
    copyPStudent(ps.fee, ps.workplace);
  }
  return *this;
}
```

ЗАДАЧА

```
void PStudent::PrintPStudent() const {
  PrintStudent();
  cout << "Fee: " << fee << endl
       << " Workplace:" << workplace<< endl;
}
```




```
class base {
   public:    int b1;
   protected: int b2;
   private:   int b3;
} b;

class der1 : protected base {
   public:    int d11;
   protected: int d12;
   private:   int d13;
} d1;

class der2 : public base {
   public:    int d21;
   protected: int d22;
   private:   int d23;
} d2;
```

```
class der3 : public der1 {
   public:    int d31;
   protected: int d32;
   private:   int d33;
} d3;

class der4 : der1 {
   public:    int d41;
   protected: int d42;
   private:   int d43;
} d4;

class der6 : protected der3 {
   public:    int d61;
   protected: int d62;
   private:   int d63;
} d6;

class der7 : public der4 {
   public:    int d71;
   protected: int d72;
   private:   int d73;
} d7;

class der5 : protected der2 {
   public:    int d51;
   protected: int d52;
   private:   int d53;
} d5;

class der8 : public der5 {
   public:    int d81;
   protected: int d82;
   private:   int d83;
} d8;
```

ПД & ВД

Решение:

ВД

b -
d1 -
d2 -
d3 -
d4 -
d5 -
d6 -
d7 -
d8 -

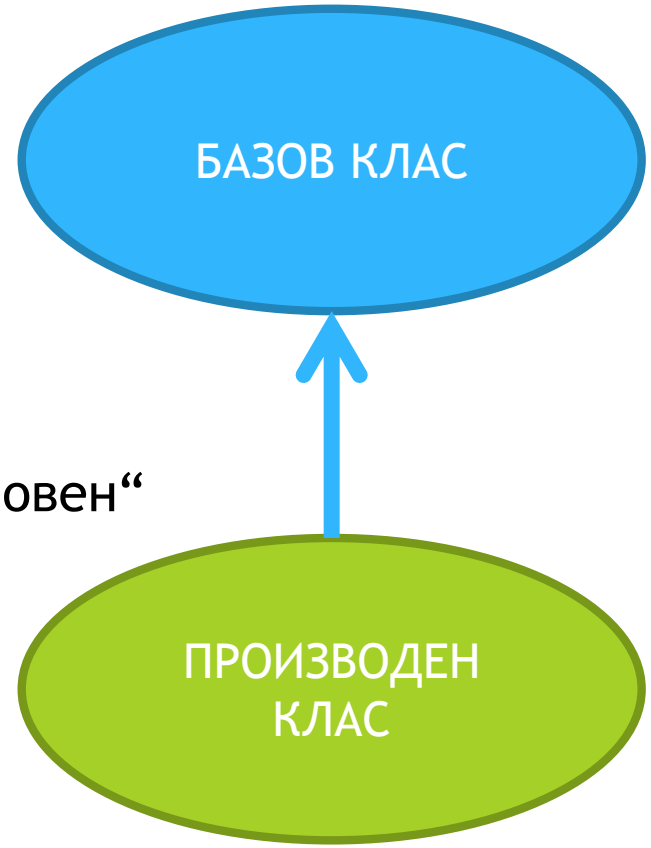
ПД

base -
der1 -
der2 -
der3 -
der4 -
der5 -
der6 -
der7 -
der8 -

Преобразуване на типове

ПРЕОБРАЗУВАНЕ НА ТИПОВЕ

- Ако наследяването на базовия клас е с атрибут public, възможно е заменяне на обекти.
- Могат да се заменят обекти, псевдоними на обекти, указатели към обекти.



- Замяната в посока „производен с основен“ е безопасна.
- Замяната в посока „основен с производен“ може да предизвика проблеми.

„ПРОИЗВОДЕН ОСНОВЕН“

- Обект,
  псевдоним на обекти,
  указател към обект
  на производен клас се преобазуват съответно в
  обект, псевдоним, указател към обект на основен клас с *неявни стандартни преобразувания*.

- Свежда се до използването на наследените копмоненти на класа.

„Основен производен“

- Основният клас не съдържа собствените компоненти на производния клас.
- *Реализира се чрез явно указване*

```
base x;
der y = (der) x;
```

- Собствените компоненти остават неинициализирани
- Опитът за използването им може да доведе до сериозни последици!

„ПРОИЗВОДЕН ОСНОВЕН“

- Основният клас не съдържа собствените компоненти на производния клас.
- *Реализира се чрез явно указване*

```
der y;
der * ptr = &y;
base * basePtr = ptr;
```

- Указателите сочат към обекта y
- Указателите са различни
- С указателя ptr са допустими извиквания на методи на класа der
- С указателя basePtr НЕ са допустими извиквания на методи на класа der
  Може да се реализира с явно преобразуване:
  `((der *) basePtr) -> derFunc();`

УКАЗАТЕЛ КЪМ МЕТОД

- Указател към метод на основният клас присвоява указател към метод на производен клас

```
void (base::*bPtr)() = base::bFunc;
```

- За да се използва този указател, е необходимо той да се свърже с конкретен обект.

```
base x;
(x.*bPtr)();
```

Възможно е:

```
void (der::*dPtr)() = pPtr;
der y;
(y.*dPtr)();
```

Ще се извърши неявно преобразуване на типовете.
Обектът y съдържа наследената част от основния клас.

Указател към метод

- Обратното присвояване – изисква явно преобразуване и да ли ще се използва правилно, зависи единствено от програмиста.

```
void (der::*dPtr)() = der::dFunc;
void (base::*bPtr)() = dPtr;
```

```
void (base::*bPtr)();
bPtr = (void (base::*)()) der::dFunc;
```

КРАЙ