Тема 3. Производност и наследяване, приятелски класове и функции

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

- Производни класове. Контрол на достъпа;
- Приятелски класове;
- Приятелски функции;
- Глобални приятелски класове и функции:
  - ограничения при използване
- Глобални приятелски оператори;
- Предефиниране на глобални оператори;
- Приятелски класове и функции (правила).

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Дефиниции:

Базов клас е клас, който има наследници. Всеки клас може да се използва като базов.

Производен клас или ***клас наследник*** е клас, който използва (част от) членовете на базов клас.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Дефиниции:

Производност е дефиниране на нов клас чрез разширение на един или няколко вече съществуващи такива. Новият клас се нарича ***производен клас***, а съществуващият преди това клас, от който той е наследен се нарича ***базов клас***.

Обикновено има само един базов клас.

C++ позволява съществуването на

повече от един базов клас. В този случаи се

получава ***множествено наследяване***.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Производните класове са особено полезно свойство на ООП, защото:

- Моделират отношението „е като“ (is-a);
- Позволяват дефиниция на нови класове чрез разширяване на съществуващи такива;
- Програмистът може да използва общите член променливи и функции на класовете в дадена програма;
- Различни класове могат да споделят данни (член променливи), операции (член функции) и нива на достъп.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Синтаксис за деклариране на наследник:

Най-често използван:

```
class <име на класа наследник> : public < име на базов клас> {
// декларации
};
```

Общ:

```
class <име на класа наследник> : [virtual] public | private |protected
  < име на базов клас> {
// декларации
};
```

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Промяна на нивото на достъпа до членовете на клас Base при наследяването му от наследник - Derivative: частните членове на базовия клас Base са недостъпни в трите вида:

	class Base { private:...	class Base { protected:...	class Base {public:
class Derivative:private Base	X	private:	private:
class Derivative:protected Base	X	protected:	protected:
class Derivative:public Base	X	protected:	public:

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Наследника не наследява следните функции на базовия клас:

- Конструктори и деструктори;
- Предефинирани оператори;
- Приятелски функции.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Пример:

Програма, която дефинира класовете CPerson и CParent. Класът CParent е производен на класа CPerson, защото описваните чрез него обекти (родителите) също са хора.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Пример човек личност и родител:

```
#include <iostream>
#include <string>
#define MAX_CHILDS 10
using namespace std;
enum Gender { male, female, unknown };
class CPerson {
 protected:
   string m_strName;
   Gender m_eGender;
 public:
   CPerson() {
     m_strName="unknown";     m_eGender=unknown;
   }
   void SetName( const string& strName) {
     m_strName = strName;
   }
   void SetGender( Gender from) {
     m_eGender = from;
   }
```

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Пример (продължение):

```
string GetName() const    { return m_strName; }
ostream& Output( ostream& toStream ) const {
   toStream << "Person Name = " << m_strName;
   switch( m_eGender ){
    case  male:
          toStream << " Gender = " << "male" << endl;
      break;
    case  female:
          toStream << " Gender = " << "female" << endl;
      break;
    default:
       toStream<<  " Gender = " << “unknown" << endl;
   }
   return toStream;
 }
}; // край на класа CPerson
```

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

```
class CParent : public CPerson {
 unsigned int m_uiNumChildren;
 CPerson* m_pChild[MAX_CHILDS];
 public:
  CParent() :m_uiNumChildren(0) { }
  ~CParent() {
    if(m_pChild) {
      for(int i=0; i<m_uiNumChildren; i++)
        delete m_pChild[i];
    }
  }
  void CreateChild(Gender gend, const string& strName) {
    CPerson *pChild = new CPerson;
    pChild->SetName(strName);
    pChild->SetGender(gend);
    m_pChild[m_uiNumChildren] = pChild;
    m_uiNumChildren++;
  }
  CPerson* GetChild( unsigned int iNum ) const {
    if( iNum >= m_uiNumChildren ) {
      return NULL;
   }
    return m_pChild[iNum];
  }
  ostream& Output( ostream& toStream )const {
    toStream << "Parent num chids : " << m_uiNumChildren << endl;
    return CPerson::Output(toStream);
  }
// други член променливи и функции
};
```

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Производният клас CParent наследява всички членове на базовия си клас CPerson:

- Той съдържа всички член променливи, съществуващи в базовия ;
- Поддържа същите член функции, както базовия;
- Декларации на обекти от базов и производен клас с подразбиращ се конструктор.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Пример за създаване на обекти:

CPerson oPerson;

CParent oParent;

Тъй като обектът oPerson е от клас CPerson:

- *Има член променливите m_strName и m_eGender и член функции- GetName(), Output(...);*

Класът CParent е производен на CPerson:

- *Обектът oParent също има член променливите m_strName и m_eGender и член функциите му – GetName(), Output(...).*

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Производният клас може да *разширява* базовия(те) клас(ове) по два начина:

- Чрез дефиниране на **нови** член променливи и член функции;
- Чрез **предефиниране** на съществуващи член функции:

  Дефиниция на ***предефинирана функция в производен клас***:

  - Ако една функция е дефинирана в производен клас и има същата ***сигнатура*** (име, брой и тип на формалните аргументи) както в базовия клас, функцията в производният клас ***предефинира*** тази в базовия.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Класът CParent добавя

- член променливи: uiNumChildren и m_pChild
- член функции: GetChild и CreateChild

Функцията Output в класа CParent предефинира функцията Output на класа Person.

Например,

oPerson.Output(cout); ще извика Person::Output

oParent.Output(cout); извиква CParent::Output.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Начин на обработка на предефинираните функции:

Търсене на предефинираните функции, определяйки областта на видимост в следния ред:

- ***При член функция се проверява дали сигнатурата на функцията е дефинирана в същия клас. Ако е така, използва се тази версия;***
- ***Проверява се за декларация в базовия клас. Прави се рекурсивно, ако базовия клас има свой базов клас, т.е. всички базови класове;***
- ***Ако не е декларирана в йерархията на класа- проверява се за глобална декларация.***

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Правила за използване:

Обект от призводния клас може да се използва навсякъде в програмата, където може да се използва базовия:

Това означава, че CParent може да бъде подаден като актуален параметър (по стойност или чрез референция) на функция, в която формалният параметър е CPerson.

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Възможно е адресът на производен клас да бъде присвоен на указател или референция към базовия клас:

Пример - указател:

```
CPerson* ptrPerson= new CParent;
//не е възможно да се извика функцията GetChild от CParent
ptrPerson ->GetChild(1); // ! невалидно обръщение
// защото ptrPerson е указател към обект от клас CPerson, а
//CPerson не е задължително CParent.
```

Важи и за референция към Cperson реферираща Cparent:

Пример - референция:

```
CPerson& rPerson = *new CParent;
rPerson.GetChild(0);  // ! невалидно обръщение
```

ПРОИЗВОДНОСТ И НАСЛЕДЯВАНЕ

Пример:

Използване на базовия и производен клас :

```
CParent oParent;                          // създаване на наследник
oParent.SetName("Boris II");        // промяна на име
oParent.SetGender(male);            // промяна на пол
oParent.CreateChild(male,"Simeon II"); // създаване на дете
oParent.CreateChild(female,"Maria Antoanetta");
oParent.Output( cout );    // извежда данните на наследника Boris II
cout << "Childs: " << endl;
CPerson* FirstChild = oParent.GetChild(0);
FirstChild->Output( cout ); // извежда данните на първо дете Simeon
CPerson* SecondChild = oParent.GetChild(1);
SecondChild->Output( cout ); // извежда данните на второ дете
```

ПРОИЗВОДНИ КЛАСОВЕ *КОНТРОЛ НА ДОСТЪПА*

Видове членове на класовете (нива на достъп):

- **private;**
- **public;**
- **protected.**

Частните членове (private) са достъпни само от член функциите и ***приятелите на класа,*** в който са декларирани. Това означава, **че член функциите на производен клас нямат достъп до частните членове на базовия клас**, въпреки че производният клас ги наследява!

(Ако се обявят членовете на базовия клас за **публични**, всички класове биха имали достъп до тях директно, не само производните класове)

ПРОИЗВОДНИ КЛАСОВЕ *КОНТРОЛ НА ДОСТЪПА*

Членовете ***protected*** могат да бъдат използвани от всички член функции и ***приятели на класа***, в който са декларирани.

Освен това те са достъпни и от член функциите и приятелите на всички класове, **производни на класа**, в който са декларирани.

ПРИЯТЕЛСКИ КЛАСОВЕ И ФУНКЦИИ (FRIEND)

Дефиниция:

Приятелски клас на даден клас A е този клас B, който има достъп до всички членове на A, включително скритите (private) и защитените (protected).

- Приятелският клас се обявява в **класа A, който предоставя своите членове** на друг клас B (наречен приятелски клас). Ключовата дума ***friend*** в декларацията на класа (или структурата) предоставя пълен достъп до декларирания след нея клас.

ПРИЯТЕЛСКИ КЛАСОВЕ И ФУНКЦИИ (FRIEND)

Предназначение :

Приятелските класове имплементират **релационна връзка** между обекти от различни класове:

Релация между класове има тогава, когато те са свързани с обща функционалност.

- Това означава, че член променливите им имат функционални зависимости, които могат да бъдат в единия или другия клас.

Приятелският клас оперира с член променливите и член функциите на класа като с публично обявени-“между приятелите няма нищо скрито”.

ПРИЯТЕЛСКИ КЛАСОВЕ И ФУНКЦИИ

Синтаксис:

```
class име {
  friend class име_на_клас_приятел_на_текущия;
// .....
};
```

```
/** Клас планета(Planet) пример за производен клас на клас орбита (Orbiter). Планетата
    има приятелски клас луна  Moon:
*/
class Orbiter {
protected:
  double m_mass;
//. . .
};
class Planet : public Orbiter {
  friend class Moon;
public:
// .....
};
```

ПРИЯТЕЛСКИ КЛАСОВЕ И ФУНКЦИИ

```
class Moon: public Orbiter {
private:
  Planet* m_pPlanet;
public:
//.....
  Planet *GetPlanet() const { return m_pPlanet; }
  void SetPlanet( Planet * pPlanet) { m_pPlanet = pPlanet; }
  double MassProduct();
};
/** Функцията предоставя достъп до член променливата на Planet,
        която е privete в собствения си клас
*/
double Moon:: MassProduct() {
// Достъп до член променливата m_mass
  return GetPlanet()->m_mass * m_mass;
}
```

ПРИЯТЕЛСКИ ФУНКЦИИ

Дефиниция:
Приятелска функция в даден клас A е тази функция на класа, която има пред декларацията си ключовата дума friend и декларацията на класа на който се предоставя достъп (приятелски клас).
Приятелските функции ограничават релацията между класовете само до определена функция (не се допуска пълен достъп до всички членове).

Пример: Ограничаване релацията между Planet и Moon само до функцията MassProduct():

```
class Planet : public Orbiter {
  friend double Moon::MassProduct(); //от кой клас е?
public:
// ....
};
```

ГЛОБАЛНИ ПРИЯТЕЛСКИ ФУНКЦИИ

Начини за създаване на глобални приятелски функции:

- Обявява се общодостъпна функция, декларирана в даден клас A, която се предоставя на всички класове, които използват класа A.
- Обявява се глобална функция (няма клас принадлежност, но използва като формален параметър класа), обявена в класа като приятелска. Такава функция се нарича ***глобална приятелска функция***:

ГЛОБАЛНИ ПРИЯТЕЛСКИ ФУНКЦИИ

Пример за първи вариант:

Нека клас CXY представя вектор и е необходимо изчислението на тангенса на този вектор. Функцията може да се добави като член функция към класа:

```
/** Изчисляване на тангенс като член променлива на класа
*/
double CXY::tan(void) const {
  return y!=0.0 ? x/y: 0.0;
}
```

Тогава функцията може да се извика ако е създаден обект от класа CXY:

```
CXY vector(1.0, 1.0);
double result = vector.tan();
```

ГЛОБАЛНИ ПРИЯТЕЛСКИ ФУНКЦИИ

Пример за втори вариант:

Алтернатива на първото решение е обявяването на глобална функция tan, която няма клас принадлежност, но използва като формален параметър класа CXY, обявена в класа като приятелска - ***глобална приятелска функция***:

```
double tan( const CXY& xy ) {
  return xy.y!=0.0 ? xy.x/xy.y: 0.0;
}
class CXY {
  friend double tan( const CXY& ); //глобална функция без клас
//...
};
```

Пример за използване:

```
CXY vector(1.0, 1.0);
double result = tan(vector);
```

ГЛОБАЛНИ ПРИЯТЕЛСКИ КЛАСОВЕ И ФУНКЦИИ (ОГРАНИЧЕНИЯ ПРИ ИЗПОЛЗВАНЕ)

Приложение:

- Приятелските класове трябва да се избягват:
  - те нарушават или отслабват капсулирането на класовете;
- Те са въведени в езика C++ с една единствена цел - да подпомогнат предефинирането на оператори:
  - обектите от класа се появяват в дясната страна на оператор-появата на не-публични член променливи на класа в дясната страна на оператора ще предизвиква грешка.

ГЛОБАЛНИ ПРИЯТЕЛСКИ ФУНКЦИИ

Използване:

- Предефиниране на глобални **оператори:**
  - Вход/изход: >> и <<
  - Присвояване: = -= +=
  - Алгебрични: + - * / ++ -- %...
  - Логически: || && !
  - Отношения: == != >= <= < >
  - Битови: & | ^ ~
- Пример-оператора за извеждане.

Той се използва за извеждане на обекти от даден клас, когато член променливите му **са скрити или защитени.**

ГЛОБАЛНИ ПРИЯТЕЛСКИ ОПЕРАТОРИ

Пример:

```
#include <string>
#include <iostream>
using namespace std;
enum Gender {male,female,unknown};
class CPerson { // базов клас, използван в упражненията
  public: CPerson(const string&){}
};
class CStudent  : public CPerson{
string m_strFacultyNum;
 Gender m_gend;
 ostream& Output( ostream& toStream ) const;
public:
   CStudent(const string& strName, const string& strFacultyNum, Gender gend) :
         CPerson(strName) , m_strFacultyNum(strFacultyNum) , m_gend(gend) {}
   friend ostream& operator<<(ostream&, const CStudent&);
};
```

ПРЕДЕФИНИРАНЕ НА ГЛОБАЛНИ ОПЕРАТОРИ

```
/* Дефиниция на глобалната приятелска функция, предекларираща оператор
        за извеждане на конзолния изход за класа CStudent */

ostream& operator<<(ostream& os, const CStudent& st) {
  return st.Output(os);// извиква частна функция
}

// Пример за използване
{
  CStudent oSt1("Ivan Petrov", "61562101", male);
  cout << oSt1;
}
```

ПРИЯТЕЛСКИ КЛАСОВЕ И ФУНКЦИИ ПРАВИЛА

Има две основни правила относно приятелските класове и функции:

1. Липса на транзитивност:

   Ако даден клас A е приятелски на класа B, тогава приятелят C на клас B не е приятелски на класа A, освен ако това се декларира изрично.

   - “Приятелят на моя приятел не е непременно мой приятел”;

ПРИЯТЕЛСКИ КЛАСОВЕ И ФУНКЦИИ (ПРАВИЛА)

2. Липса на наследяване:

Ако класът A е приятелски на класа B, тогава производният клас C на A не е непременно приятелски клас на B:

- **“Родителят на моя приятел не е непременно мой приятел”.**

Въпроси?