Лекція 5

Тема лекції: Проста спадкоємність. Оголошення похідних класів. Використання конструкторів та функцій-членів похідних класів. похідних класів. Правила доступу до окремих членів класу

Проста спадкоємність

Похідний клас успадковує з базового класу дані-члени і функції-члени, але не конструктори і деструктори.

Похідний клас починає своє існування з копіювання членів базових класів, включаючи всі члени, успадковані з більш далеких родинних класів.

Розглянемо базовий клас:

```
class TBase
{
 private:
  int count;
 public:
  TBase()
   {
    count = 0;
   }
 void SetCount(int n)
  {
   count = n;
  }
 int GetCount(void)
  {
   return count;
  }
};
```

Тут *count* - закрите дане-член. Конструктор за замовчуванням *TBase()* ініціалізує *count* нульовим значенням. Функція-член *SetCount()* привласнює *count* нове значення. Функція-член *GetCount()* повертає поточне значення *count*.

Всі ці функції є вбудованими. Створимо клас, що має усі властивості TBase, але крім того, спроможний збільшити значення об’єкту на заданий розмір.

```
class TDerived:public TBase
{
  public:
   TDerived():TBase(){ }
   void ChangeCount(int n)
   {
    SetCount(GetCount()+n);
   }
};
```

Похідний клас названий *TDerived*. Відразу за ім'ям класу йде двокрапка й одне з ключових слів - *public, protected* або *private*. Після цих елементів слідує ім'я базового класу, від якого похідний клас одержує спадщину. В даному випадку усі відкриті члени *TBase* стають відкритими членами й у *TDerived*. Всі закриті члени залишаються закритими в своєму, спочатку оголошеному класі. В *TDerived* не можна одержати доступ до цих членів *TBase*.

Можна також оголосити базовий клас закритим:

```
class TDerived:private TBase{};
```

У цьому випадку усі відкриті члени *TBase* стають закритими членами в *TDerived*. Третій тип специфікатора *protected* розглядається пізніше.

Завичай у похідному класі є конструктор, якщо він є й у базовому класі. Крім того, конструктор похідного класу повинен викликати конструктор базового класу. У *TDerived* у рядку

```
TDerived():TBase(){}
```

оголошується конструктор похідного класу *TDerived()* і викликається конструктор базового класу за допомогою спеціального позначення *:TBase()*. Не можна викликати конструктори базових класів в операторах. У конструкторі можуть виконуватися оператори. Наприклад, конструктор *TDerived()* можна оголосити вбудованим таким чином:

```
TDerived():TBase()
{
  cout << “Ініціалізація\n”;
```

```
}
```

Конструктори похідних класів не обов'язково повинні бути вбудовані. Наприклад:

```
class TDerived:public TBase
{
   public:
     TDerived();
     void ChangeCount(int n)
     {
       SetCount(GetCount()+n);
     }
};
```

У цьому випадку необхідно реалізувати конструктор *TDerived()* окремо:

```
TDerived::TDerived():TBase()
{/*Оператори конструктора*/}
```

Похідний клас успадковує c*ount* із базового класу. В класі *TDerived* не можна одержати до класу *TBase* безпосередній доступ, оскільки *count* закритий у *TBase*. Тільки члени класу можуть мати доступ до закритих членів цього ж класу. У похідному класі можна оголосити свої власні закриті члени:

```
class TDerived:public TBase
{
 private:
  int SecondCount;
  void ChangeCount(int n)
   {... .}
};
```

TDerived успадковує властивості *TBase*, включаючи закритий член базового класу *count*. У новому класі додався новий закритий член *SecondCount*. Для його ініціалізації реалізуємо конструктор:

```
TDerived::TDerived():TBase
{
  SecondCount = 0;
}
```

Виклик конструктора *TBase()* ініціалізує об'єкт класу *TBase*. Оператор привласнення

```
SecondCount=0;
```

у конструкторі похідного класу ініціалізує об'єкт класу *TDerived*. Конструктор базового класу повинен викликатися першим. У новому класі *TDerived* оголошена відкрита функція-член *ChangeCount()* для збільшення значення члена *count*, успадкованого з класу *TBase*. У функції *ChangeCount() не* можна одержати безпосередній досуп до члена *count,* тому що він закритий у класі *TBase*:

```
void ChangeCount(int n)
{
   count+=n;
}
```

Цей оператор викликає повідомлення про помилку при компіляції.

Правила використання відкритих, закритих і захищених членів класу

Захищений член класу - щось середнє між закритим і відкритим членом. Подібно до закритих членів, захищені члени доступні тільки функціям-членам класу. Поза класом захищені члени невидимі. Подібно до відкритих членів, захищені успадковуються похідними класами і доступні функціям-членам похідних класів.

- Закриті члени класу доступні тільки в класі, де вони оголошені.
- Захищені члени доступні членам їхнього власного класу та усім членам похідних класів.
- Відкриті члени доступні членам їхнього власного класу, членам похідних класів і всіх інших користувачів цього класу.
- За замовчуванням члени класу, що успадковується, закриті. Специфікатори *private*, *protected* і *public* впливають на статус успадкованих членів.
- Члени відкритого базового класу в похідному класі зберігають свої специфікатори доступу.
- Відкриті члени захищеного базового класу стають захищеними в похідному класі. Захищені і закриті члени зберігають свої початкові

специфікації доступу.

• Всі члени закритого базового класу стають закритими в похідному класі незалежно від початкової специфікації доступу цих членів.

Третій випадок найбільш істотно впливає на такі члени:

```
class TBase
{
 protected:
   int x;
 public:
   int y;
...... .
};
class TDerived:private TBase
{
 public:
   void f(void);
};
class TDesc:public TDerived
{
 public:
   void g(void);
};
```

Функція-член *f()* класу *TDerived* має доступ до членів *x* і *y*, що успадковані від класу *TBase*. Але оскільки *TBase* оголошений закритим для класу *TDeruved*, то статус членів *x* і *y* змінюється на *private*. Функція-член *g()* класу *TDesc*, похідного класу від *TDerived*, не має доступу до членів *x* і *y*, незважаючи на те, що спочатку ці члени мали статус *protected* і *public*.

Специфікатор базового класу потенційно впливає на всі члени, що успадковуються. При необхідності зробити закритими в похідному класі тільки деякі з успадкованих членів, можна вибірково кваліфікувати один або декілька з них. Приклад:

```
class A
{
 public:
  int x;
```

```
  A()
   {
     x=0;
   }
   void Display(void)
    {
      cout << "Конструктор B \n";
    }
};
```

Запис:

```
B object;
object.Display();
```

помилковий, оскільки клас *A* оголошений закритим для класу *B*. Щоб зробити функцію-член класу *A Display()* відкритою для класу *B*, залишивши дане-член класу *A x* закритим, запишемо:

```
class B:private A
{
 public:
  A::Display();
  B():A()
   {
      cout << "Конструктор B\n";
   }
};
```

Тепер можна створити об'єкт типу *B* і викликати функцію *Display()* з класу *A*.

Використання конструкторів і деструкторів

У похідному класі деструктор потрібен тільки при наявності членів, які необхідно видаляти, коли об'єкт похідного класу виходить з області бачення. Розглянемо базовий клас:

```
class TBase
{
 private:
```

```
 char *basep;
public:
 TBase(count char *s)
 {
  basep = strdup(s);
 }
 ~TBase()
 {
  delete basep;
 }
 const char *GetStr(void)
 {
  return basep;
 }
};
```

Клас *TBase* має один захищений член - покажчик *basep* на рядок символів. Конструктор класу викликає бібліотечну функцію *strdup()* для створення копії рядкового аргументу у вигляді динамічної змінної з привласненням виділеної адреси покажчику basep. Деструктор *~TBase()* звільняє цей блок пам'яті, коли об'єкт типу *TBase* виходить з області бачення. Оголосимо похідний клас:

```
class TDerived:public TBase
{
 private:
  char *uppercasep;
 public:
  TDerived(const char *s);
  ~TDerived()
  {
   delete uppercasep;
  }
  const char *GetUStr(void)
  {
   return uppercasep;
  }
};
```

До членів, успадкованих із *TBase*, *TDerived* додає закритий член - символьний покажчик з ім'ям *uppercasep*, який посилається на копію рядка, що зберігається в *TBase*, із символами, перетвореними на великі. Деструктор похідного класу видаляє цей новий рядок. Конструктор *TDerived()* можна реалізувати таким чином:

```
TDerived::TDerived(const char *s):TBase(s)
{
 uppercasep = strupr(strdup(s));
}
```

Конструктор *TDerived()* викликає конструктор *TBase()*, що копіює рядок, на який посилається *s*, до купи і покажчик, що ініціалізує *basep* адресою рядка. Після завершення цієї операції конструктор похідного класу створює ще одну копію рядка, перетворюючи його символи на прописні букви за допомогою бібліотечної функції *strupr()* і привласнює *uppercasep* адресу цієї копії рядка. При створенні об’єкту типу *TDerived* створюється 2 копії рядка: одна незмінна та одна, символи якої перетворені на прописні.

Створимо такий об'єкт:

TDerived country (“Ukraine”);

Представлення цього об’єкту в пам'яті таке:

початковий рядок, ініціалізований конструктором базового класу та копія рядка з перетворених рядкових літер на прописні, ініціалізована конструктором похідного класу:

Об'єкт coutnty класу TBase:

*char *basep ---> ukraine*

*char *uppercasep ---> UKRAINE*

Об'єкту *country* належить два блоки пам'яті. Коли об'єкт виходить з області бачення, його деструктор звільняє цю пам'ять і повертає її купі для наступного використання. Деструктор класу *TBase* звільняє пам'ять, на яку посилається член *basep*. Деструктор класу *TDerive* звільняє пам'ять, на яку посилається *uppercasep*. На відміну від конструкторів, деструктор похідного класу ніколи не викликає безпосередньо деструктор базового класу. C++ автоматично спочатку викликає деструктор похідного класу, потім - деструктор

базового класу. Порядок виклику деструкторів - обернений порядку виклику конструкторів.

У похідному класі можна змінити поведінку успадкованого елементу. Наприклад, можна оголосити змінну або функцію з тим самим ім'ям, що й у базовому класі, але яка має інший тип або виконує інші дії. Це називається перевизначенням. Якщо в похідному класі є елемент з таким самим ім'ям, що й у базовому, використовується елемент похідного класу. Якщо з'являється необхідність викликати елемент базового класу, потрібно застосувати оператор розширення області бачення.

При створенні нової програми бажано пошукати серед класів такі, що можуть бути потрібні надалі. Використавши їх у якості базових, можна вивести нові класи, тобто використовувати повторно вже існуючий код.