Лабораторна робота №7. Множинне успадкування класів

Мета роботи: вивчення особливостей множинного успадкування класів.
Теоретичний матеріал лекції, [1, розділ 8], [2, розділ 16, 17], [3, розділи 8, 13].

1. Короткі теоретичні відомості

1.1. Оголошення множинного успадкування

Один клас C++ може успадковувати елементи декількох інших класів. Це забезпечує поєднання властивостей декількох класів в рдному класі.

Оголошення множинного успадкування:

```
class Derived: режим успадкування_1 Base1,
               /* … */
               режим успадкування_1 BaseN
{ … };
```

Семантично множинне успадкування ґрунтується на відношенні “є”: похідний клас у деякому сенсі є екземпляром усіх базових класів.

1.2. Порядок виклику конструкторів і деструкторів

При множинному успадкуванні конструктор похідного класу викликає конструктори базових класів для ініціалізації успадкованих елементів даних. Порядок виклику конструкторів:

- викликаються конструктори базових класів за послідовністю їх множинного успадкування; віртуальні базові класи ініціалізуються перед невіртуальними базовими класами;
- якщо базовий клас є контейнерним, то спочатку будуть викликані конструктори для інкапсульованих у нього об’єктів за послідовністю їх розміщення у протоколі класу, а потім – відповідний конструктор базового класу;
- для тих базових класів, які не вказані у списку ініціалізації конструктора похідного класу, викликаються конструтори за замовчуванням (без параметрів);
- викликаються конструтори об’єктів, інкапсульованих у похідний клас;
- на завершення викликається конструктор похідного класу.

1.3. Доступ до перекритих елементів класі

Оголошуючи власні елементи даних та методи, похідний клас може перекривати однойменні елементи, успадковані від базових класів. Тоді доступ до перекритих методів відбувається через імена базових класів то операції дозволу доступу ‘::’.

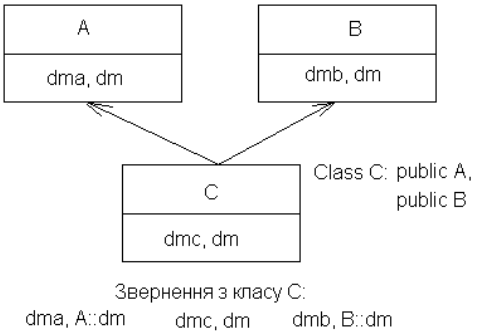

Рис.1. Діаграма множинного успадкування

1.4. Неоднозначність параметричного перевантаження методів

У разі множинного успадкування методів, що мають однакове ім'я, але відрізняються типами параметрів, може виникати неоднозначність їх виклику.

```
class A {
public:
   void output(int i) {cout<<"A::output()="<<i<<endl;}
};
class B {
public:
   void output(double d) {cout<<"B::output()="<<d<<endl;}
};
class C: public A, public B { . . . };
int main() {
  C c;
  c.output(5);
  c.output(3.14);
  return 0;
}
```

При множинному успадкуванні виникає неоднозначність виклику методів `output(int)` та `output(double)` з класу `C` у зв'язку з неявним перетворенням типів їх аргументів. Значення цілого типу перетворюється до дійсного, а дійсне значення – до цілого, тому компілятор не може визначити, який з двох методів необхідно викликати.

Для уникнення неоднозначностей необхідно у класі C перевизначити успадковані методи:

```
class C: public A, public B {
public:
    void output(int i)    {A::output(i);}
    void output(double d) {B::output(d);}
};
```

В результаті виклики методів відбуваються правильно.

1.5. Множинне успадкування класів із загальною базою

На практиці можливі випадки, коли декілька класів успадковують один і той самий клас. Такий варіант називають успадкуванням із загальною базою. При цьому у похідний клас потрапить декілька екземплярів елементів загального базового класу. Тоді для звернення до множинно успадкованих елементів базового класу можна використати назву одного із похідних класів та операцію дозволу доступу.

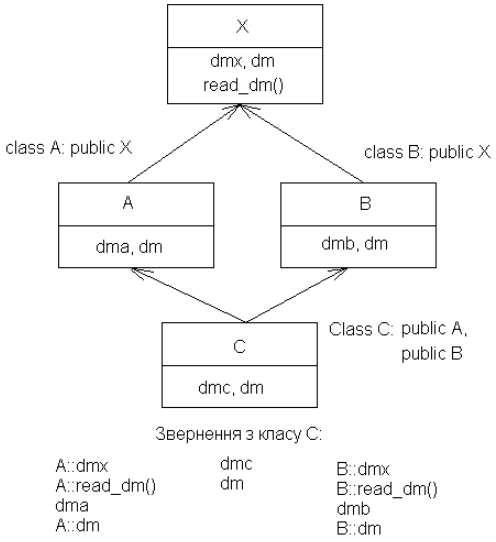

Рис.2. Діаграма множинного успадкування класів із загальною базою

Ситуація ускладнюється, коли перекриваються однойменні елементи даних або однойменні методи класів (таким полем є `dm`). Тому для забезпечення доступу з похідного класу `C` до поля `dm` класу `X` необхідна наявність методу `X::read_dm()`, який не перекривається у

класах A та B.

Клас C не може виконати пряме успадкування класу X. Посиланню на клас X не можна присвоїти об'єкт класу C, а вказівнику на клас X – адресу об'єкта класу C. Однак, для public-успадкування дозволяється ланцюжок присвоєнь від похідного класу C множинного успадкування через проміжний клас A або B до загального базового класу X:

```
X x3=a=c;     //  або  X x3=b=c;
```

Елементи-дані і методи класу X попадають у клас C у двох екземплярах.

1.6. Віртуальне успадкування класів

Щоб існував тільки один екземпляр елементів базового класу при множинному успадкуванні із загальною базою, використовують механізм віртуального успадкування. Для цього при прямому успадкуванні базового класу X похідними класами A та B разом з режимом успадкування вказується режим `virtual`. У цьому випадку для доступу до елементів віртуального базового класу не потрібно використовувати операцію доступу через проміжний клас, оскільки вони існують тільки в одному екземплярі.

При віртуальному успадкуванні об'єкт класу X можна проініціалізувати об'єктом класу C.

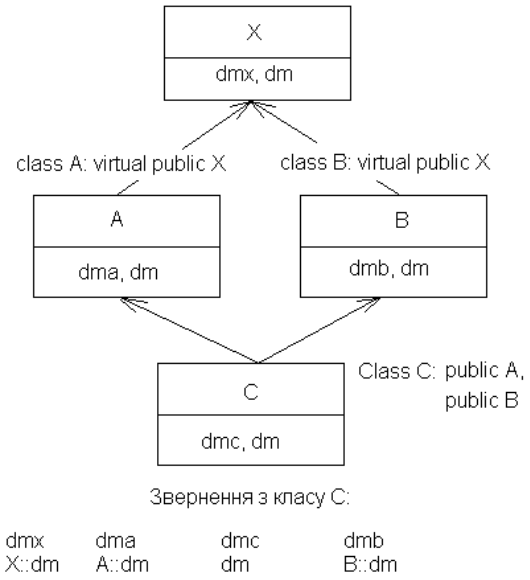

Рис. 3. Діаграма множинного успадкування з віртуальним базовим класом

1.7. Віртуальні методи множинного успадкування

1.7.1. Безпосереднє множинне успадкування

Множинне успадкування підтримує механізм пізнього зв'язування за допомогою віртуальних методів, вказівників або посилань на клас. Нехай у базових класах A та B визначено віртуальні методи `output()`. Клас C множинно успадковує класи A та B і визначає власний віртуальний метод `C::output()`. Для підтримування пізнього зв'язування використовується тільки `public`-успадкування базових класів.

У функції `main()` посилання на базові класи A та B ініціалізуються назвою об'єкта класу C і використовуються для виклику віртуального методу `C::output()`. Завдяки пізньому зв'язуванню ці посилання вказують на похідний клас C, і за їх допомогою викликається віртуальний метод `C::output()`.

```
#include <iostram>
#include <typeinfo.h>
using namespace std;
```

```
class A {
protected:
   int dma;
public:
   A(int x):dma(x) {cout<<"A::A(int)"<<endl;}
   virtual void output() {cout<<dma<<endl;}
};
class B {
protected:
   int dmb;
public:
   B(int x):dmb(x) {cout<<"B::B(int)"<<endl;}
   virtual void output() {cout<<dmb<<endl;}
};
class C: public A, public B {
private:
   int dmc;
public:
   C(int x1, int x2, int x3): A(x1),B(x2),dmc(x3)
    {cout<<"C::C(int,int,int)"<<endl;}
   virtual void output() {cout<<dma<<' '<<dmb<<' '<<dmc<<endl;}
};
int main() {
   C c(1,2,3);
   A& a=c;
   B& b=c;
   cout<<typeid(a).name()<<endl;  // class C
   cout<<typeid(b).name()<<endl;  // class C
   a.output();       // 1 2 3
   b.output();       // 1 2 3
   c.A::output();    // 1
   c.B::output();    // 2
   c.output();       // 1 2 3
}
```

1.7.2. Множинне успадкування зі спільним базовим класом

Нехай задано схему множинного успадкування класом `C` класів `A` та `B`, які в свою чергу успадковують спільний базовий клас `X`. Кожен клас визначає віртуальний метод `output()` для виведення на екран його елементів даних. Посилання на класи `A` та `B`, проініціалізовані іменем об'єкту класу `C`, викликають віртуальний метод `C::output()` з похідного класу `C`.

Посилання на клас `X` не може бути проініціалізоване об'єктом класу `C` у зв'язку з неоднозначністю перетворення типу, оскільки успадкування класу `X` відбулося двома шляхами – через клас `A` і через клас `B`.

Однак є можливим ланцюжок присвоєнь: посилання на базовий клас `X` можна проініціалізувати посиланням на клас `A` або `B`, які, в свою чергу, ініціалізуються об'єктом класу `C`. Так проініціалізоване посилання на клас `X` вказуватиме на клас `C` і за його допомогою буде викликаний віртуальний метод `C::output()` класу множинного успадкування `C`.

```
#include <iostream>
#include <typeinfo.h>
using namespace std;
class X {
protected:
   int dmx;
public:
   X(int x=0):dmx(x){cout<<"X::X()"<<endl;
   virtual void output() {cout<<dmx<<endl;}
```

```
};
class A: public X {
protected:
   int dma;
public:
   A(int x1, int x2):X(x1), dma(x2){cout<<”A::A()”<<endl;}
   virtual void output {cout<<dmx<<’ ’<<dma<<endl;
};
class B: public X {
protected:
   int dmb;
public:
   B(int x1, int x2):X(x1), dmb(x2){cout<<”B::B()”<<endl;}
   virtual void output {cout<<dmx<<’ ’<<dmb<<endl;
};
class C:public A, public B {
private:
   int dmc;
public:
   C(int x1, int x2, int x3, int x4, int x5):
     A(x1,x2),B(x3,x4),dmc(x5) {cout<<”C::C()”<<endl;}
     virtual void output()
      {cout<<A::dmx<<’ ’dma<<’ ’<<B::dmx<<’ ’<<dmb<<’ ’<< dmc<<endl;{
};
int main() {
   C c(1,2,3,4,5);
   A& a=c;
   B& b=c;
   a.output();     // 1 2 3 4 5
   b.output();     // 1 2 3 4 5
   c.A::output();  // 1 2
   c.B::output();  // 3 4
   c.output();     // 1 2 3 4 5
   X& x2=a;        // посилання на клас C
   X2.output();    // 1 2 3 4 5
   X& x3=b;        // посилання на клас C
   x3.output();    // 1 2 3 4 5
```

1.7.3. Множинне успадкування з віртуальним базовим класом

За множинного успадкування з віртуальним базовим класом посилання на клас X може бути проініціалізоване об’єктом класу C. Неоднозначності перетворення типів не виникає, оскільки успадкування класу X відбулося тільки один раз.

Так проініціалізоване посилання на клас X можна використати для виклику віртуального методу з класу C. Замість посилання можна використати вказівник на клас X, проініціалізований адресою об’єкта класу C.

```
#include <iostream>
using namespace std;
class X {
protected:
   int dmx;
```

```
public:
   X(int x=0):dmx(x){cout<<"X::X()"<<endl;}
   virtual void output() {cout<<dmx<<endl;}
};
class A: virtual public X {
protected:
   int dma;
public:
   A(int x1, int x2):X(x1), dma(x2){cout<<"A::A()"<<endl;}
   virtual void output() {cout<<dmx<<' '<<dma<<endl;}
};
class B: virtual public X {
protected:
   int dmb;
public:
   B(int x1, int x2):X(x1), dmb(x2){cout<<"B::B()"<<endl;}
   virtual void output() {cout<<dmx<<' '<<dmb<<endl;}
};
class C:public A, public B {
private:
   int dmc;
public:
   C(int x1, int x2, int x3, int x4, int x5):
     A(x1,x2),B(x3,x4),dmc(x5) {cout<<"C::C()"<<endl;}
     virtual void output()
      {cout<<A::dmx<<' '<<dma<<' '<<B::dmx<<' '<<dmb<<' '<< dmc<<endl;}
};
int main() {
   C c(1,2,3,4,5);
   A& a=c;
   B& b=c;
   a.output();      // 0 2 0 4 5
   b.output();      // 0 2 0 4 5
   c.output();      // 0 2 0 4 5
   X& x1=c;         // посилання на клас C
   x1.output();     // 0 2 0 4 5
   x1.X::output(); // 0
return 0;
}
X::X()
A::A()
B::B()
C::C()
```

Запитання.

1. Як оголосити множинне успадкування.
2. Порядок виклику конструкторів і деструкторів при множинному успадкуванні.
3. Доступ до перекритих елементів класу.
4. Неоднозначність параметричного пере визначення методів.
5. Множинне успадкування класів із загальною базою.

6. Віртуальне успадкування класів.
7. Безпосереднє множинне успадкування.
8. Множинне успадкування зі спільним базовим класом.
9. Множинне успадкування з віртуальним базовим класом.

Завдання.

Сформувати варіант успадкування класів, вилучивши із зображеного на рис. 4 орієнтованого графу вершини, номер `k` якої визначається як порядковий номер `n` студента в алфавітному списку групи за модулем 12 (`k=n mod 12`) та вершину з номером 12-`k`. Нумерація вершин здійснюється послідовно, спочатку зліва направо, а потім зверху вниз. Разом з вершиною вилучаються усі її вхідні та вихідні ребра. Біля стрілок вказано режими успадкування класів відповідно до варіанта завдання.

У кожному класі визначити дані та методи роботи з ними. Визначити дані або методи, які перекриваються у похідних класах. Кожен клас повинен містити конструктор ініціалізації, копіювання, деструктор, функцію для встановлення та читання значень даних, перевантажені операції для введення і виведення даних. Функція `main()` повинна ілюструвати доступ до даних та методів у кожному класі ієрархії успадкування.

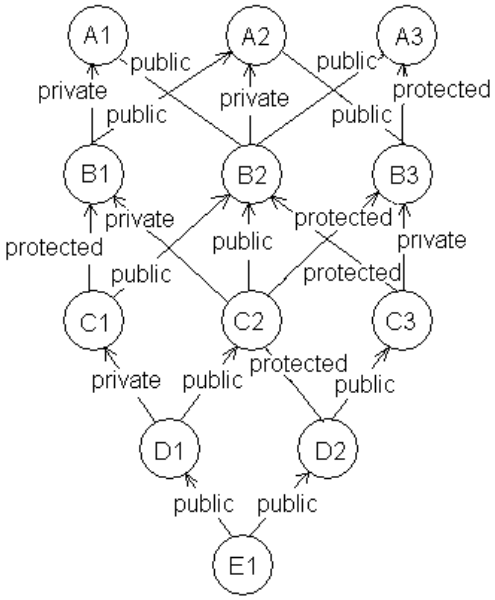

Рис. 7.4. Варіанти множинного успадкування

2. Самостійна робота

1. Приклад програми.

Створити два класи: Рамка (координати головної діагоналі) та Меню (масив рядків символів з назвами команд меню та індекс активної команди меню).

Використовуючи множинне успадкування цих класів, створити новий клас – Вікно з рамкою та меню. Додатково задати назву заголовків вікна та колір фону (рядки символів). Визначити необхідні дані, методи для роботи з даними, конструктори та деструктори, операторні функції введення-виведення даних.

```
// 7_1.cpp
// g++ 7_1.cpp –o 7_1  - компіляції і компонування програми
#include <iostream>
#include <cstring>
using namespace std;

class Frame {
protected:
   int x1,y1,x2,y2;  // координати рамки
public:
  Frame(int x1,int y1,int x2,int y2)
  {
    this->x1=x1; this->y1=y1; this->x2=x2; this->y2=y2;
  }
  // конструктор копіювання
  Frame(Frame& f)
  {
    x1=f.x1; y1=f.y1; x2=f.x2; y2=f.y2;
  }
// операція введення з потоку
  friend istream& operator>>(istream& is, Frame& f)
  {
    is>>f.x1>>f.y1>>f.x2>>f.y2;
    return is;
  }
// операція виведення у потік
  friend ostream& operator<<(ostream& os, Frame& f)
  {
    os<<f.x1<<' '<<f.y1<<' '<<f.x2<<' '<<f.y2<<endl;
    return os;
  }
};
// клас меню
class Menu {
protected:
  int count;       // кількість команд
  int active;      // номер активної команди
  char ** command; // назва команди
public:
// конструктор
  Menu(int count, int active, char **command) {
    this->count=count;
    this->active=active;

    this->command=new char*[count];
    for(int i=0; i<count; i++)
    {
      this->command[i]=new char[20];
      strcpy(this->command[i], command[i]);
    }
  }
// конструктор копіювання
  Menu(Menu& m) {
    count=m.count;
```

```
    active=m.active;

    command=new char*[m.count];
    for(int i=0; i<count; i++)
    {
      command[i]=new char[20];
      strcpy(command[i],m.command[i]);
    }
  }
//деструктор
  ~Menu()
  {
    for(int i=0; i<count; i++) delete []command[i];
  }
// операція введення з потоку
  friend istream& operator>>(istream& is, Menu& m)
  {
    for(int i=0;i<m.count;i++) is>>m.command[i];
    return is;
  }
// операція виведення у потік
  friend ostream& operator<<(ostream& os, Menu& m)
  {
    for(int i=0;i<m.count;i++) os<<m.command[i]<<endl;
    return os;
 }
};

// клас вікна
class Window: public Frame, public Menu
{
protected:
  char *title;    // заголовок вікна
  int bkcolor;    // колір фону
public:
// конструктор
Window(Frame f,Menu m,char *title,int bkcolor):Frame(f),Menu(m)
{
  this->title=new char[100];
  strcpy(this->title,title);
  this->bkcolor=bkcolor;
}
// деструктор
~Window()
{
  delete []title;
}
// операція введення з потоку
friend istream& operator>>(istream& is, Window& w)
{
  is>>w.x1>>w.y1>>w.x2>>w.y2;
  for(int i=0; i<w.count; i++) is>>w.command[i];
  is>>w.title;
  is>>w.bkcolor;
  return is;
}

// операція виведення у потік
friend ostream& operator<<(ostream& os, Window& w)
{
  Frame f(w.x1,w.y1,w.x2,w.y2),fr=f;
  os<<f;
  Menu m(w.count,w.active,w.command),mr=m;
  os<<m;
```

```
  os<<w.title<<endl;
  os<<w.bkcolor<<endl;
  return os;
}

};

// головна функція
int main()
{
  char
s0[8]="File",s1[8]="Edit",s2[8]="View",s3[8]="Project",s4[8]="Build",s5[8]="Help";
  //char * command[6]={"File","Edit","View","Project","Build","Help"};
  char * command[6]={s0,s1,s2,s3,s4,s5};
  enum colors {RED, GREEN, BLUE, WHITE, BLACK};
  Frame f(1,1,80,25),&fr=f;
  cout<<"Склад об'єкта Frame\n"<<fr;
  Menu m(6,0,command),&mr=m;
  cout<<"Склад об'єкта Menu\n"<<mr;
  char s6[10]="MyWindow";
  Window w(fr,mr,s6,WHITE),&wr=w;
  cout<<"Склад об'єкта Window\n"<<wr;
}
>./7_1
Склад об'єкта Frame
1 1 80 25
Склад об'єкта Menu
File
Edit
View
Project
Build
Help
Склад об'єкта Window
1 1 80 25
File
Edit
View
Project
Build
Help
MyWindow
3

/* 2. rozdil 16, listing 7 — множинне успадкування */
#include <iostream>
using namespace std;

class base1 {
protected:
  int x;
public:
  void showx() { cout << x << "\n"; }
};

class base2 {
protected:
  int y;
public:
  void showy() {cout << y << "\n";}
};

// множинне успадкування базових класів
class derived: public base1, public base2 {
```

```
public:
  void set(int i, int j) { x=i; y=j; }
};

int main()
{
  derived ob;

  ob.set(10, 20); // функція належить класу  derived
  ob.showx(); // функція належить класу base1
  ob.showy(); // функція належить класу base2

  return 0;
}
10
20
```