Лабораторна робота № 8. Віртуальні функції і поліморфізм

Мета роботи: вивчення віртуальних функцій і тирів поліморфізму.
Теоретичний матеріал лекції, [1, розділ 7], [2, розділ 17], [3, розділ 13].

1. Короткі теоретичні відомості

1.1. Види поліморфізму

Поліморфізм – це використання під одним іменем різних функцій, призначених для опрацювання даних різних типів. У C++ є такі типи поліморфізму:

• *статичний* – перевантаження і перевизначення функцій та операцій, шаблони функцій та класів;

• *динамічний* – перевизначення віртуальних функцій.

Статичний поліморфізм забезпечується під час компіляції програми, а динамічний – під час її виконання.

Поліморфізм *перевантаження функцій* (overload) полягає у можливості вибору компілятором різних реалізацій однієї і тієї ж самої функції, залежно від кількості або типів її аргументів. Наприклад:

```
#include <iostream>
using namespace std;
class A {
public:
  void f() { cout<<"A::f()"<<endl;}
  void f(int) { cout<<"A::f(int)"<<endl;}
  void f(double) { cout<<"A::f(double)"<<endl;}
};
int main() {
  A a;
  a.f();      // A::f()
  a.f(1);     // A::f(int)
  a.f(3.14);  // A::f(double)
  return 0;
}
```

```
A::f()
A::f(int)
A::f(double)
```

У програмі викликаються перевантажені методи `A::f()`.

Перевантаження операцій основується на операторних функціях `operator`. Приклад програми де перевантажується бінарна операція + і операція <<.

```
#include <iostream>
using namespace std;

class A {
protected:
  int x;
public:
  A(int x=0) {this->x=x;}
  A operator+(A& a) {return A(x+a.x);}
  friend ostream& operator<<(ostream& os, A& a) {os<<a.x<<endl; return os;}
};
class B: public A {
protected:
  int y;
public:
  B(int x=0, int y=0) : A(x) {this->y=y;}
```

```
  B operator+(B& b) {return B(x+b.x, y+b.y);}
  friend ostream& operator<<(ostream& os, B& b) {os<<b.x<<' '<<b.y<<endl; return
os;}
};

int main() {
  A a1(1), a2(2), a3;
  a3=a1+a2;
  cout<<a3;    // 3

  B b1(3,4), b2(5,6), b3;
  b3=b1+b2;
  cout<<b3;    // 8 10
  return 0;
}
```

```
3
8 10
```

Функція демонструє поліморфне застосування перевантаження операцій. Залежно від типів операндів одні і ті самі операції + та << будуть викликані з класу A або з класу B.

У контексті успадкування класів поліморфізм полягає у перевизначенні (`override`) нащадком методів базового класу. Тоді для роботи з об'єктами похідних класів використовується інтерфейс їх базового класу.

```
#include <iostream>
using namespace std;

class A {
public:
  void f() const {cout<<"A::f()"<<endl;}
};
class B: public A {
public:
  void f() const {cout<<"B::f()"<<endl;}
};
class C: public A {
public:
  void f() const {cout<<"C::f()"<<endl;}
};

int main() {
  C *p1 = new C;
  p1->f();       // C::f()
  A *p2 = p1;
  p2->f();       // A::f()

  B *p3 = new B;
  p3->f();       //  B::f()
  A *p4 = p3;
  p4->f();       // A::f()
  return 0;
}
```

```
C::f()
A::f()
B::f()
A::f()
```

У базовому класі визначено метод `f()`, який перевизначається у похідних класах `B` та `C`. У функції `main()` вказівник `p2` на клас `A` проініціалізовано вказівником `p1` на клас `C`. Така ініціалізація допускається без явного перетворення типу, оскільки клас `C` є похідним від класу `A`. Проініціалізовані вказівники використовуються для виклику метода `f()`. Незважаючи на те, що вказівники `p1` та `p2` набувають однакових значень, за допомогою `p1` буде викликано метод `C::f()`, а за допомогою `p2` – метод `A::f()`. Аналогічні дії відбуваються з вказівниками `p3` та `p4` на класи `B` та `A`. Те, який метод буде викликано за допомогою того чи іншого вказівника,

визначається під час компіляції (раннє зв'язування методів і об'єктів). Це означає, що за допомогою об'єкта класу, посилання або вказівника на клас можна викликати тільки методи цього самого класу. Якщо об'єкт, посилання або вказівник на базовий клас проініціалізовані об'єктом (або його адресою – для вказівника) похідного класу, то за їх допомогою не можна викликати методи похідного класу.

Розглянуті вище поліморфізму є *статичними*. При *динамічному* поліморфізмі вибір того або іншого методу здійснюється під час виконання програми (пізнє зв'язування реалізується за допомогою віртуальних методів). *Віртуальний метод* – це сукупність перевизначених функцій-членів, оголошених зі словом `virtual`.

Приклад поліморфізму віртуальних методів. Нехай клас `B` успадковує клас `A` і перевизначає метод `f()`. Якщо віртуальний метод викликається з об'єкта класу, то діє раннє зв'язування. Якщо для виклику віртуального методу використовується вказівник (або посилання), проініціалізований адресою (або іменем – для посилання) об'єкта похідного класу, то діє пізнє зв'язування і в результаті викликається метод з похідного класу.

```
#include <iostream>
using namespace std;

class A {
public:
  virtual void f() const {cout<<"A::f()"<<endl;}
};

class B: public A {
public:
  virtual void f() const {cout<<"B::f()"<<endl;}
};

int main() {
  A a;
  a.f();  // раннє зв'язування, A::f()
  B b;
  b.f();  // раннє зв'язування, B::f()
  a=b;
  a.f();  // раннє зв'язування, A::f()

  A *p1 = new B;
  p1->f();       // пізнє зв'язування, B::f()
  A& p2 = *new B;
  p2.f();        // пізнє зв'язування, B::f()

   return 0;
}
```

1.2. Особливості віртуальних методів

Сукупність класів, у яких оголошується, визначається та перевизначається метод, називається *поліморфічним кластером*. Поліморфічний кластер реалізується тільки для *public-успадкувань* класів. У межах поліморфного кластера віртуальні методи мають однакову адресу. В одній ієрархії успадкування класфі може бути визначено декілька поліморфічних кластерів.

Особливості віртуальних методів:

- віртуальними можуть бути тільки функції-члени класу, друзі не можуть бути віртуальними;
- віртуальний метод може бути оголошений дружнім (`friend`) до іншого класу;
- віртуальні методи не можуть бути статичними (`static`);
- слово `virtual` достатньо записати тільки у першому оголошенні віртуальних методів;
- віртуальний метод повинен бути визначений у класі або оголошений чистим;

- для забезпечення пізнього зв'язування віртуальні методи повинні викликатися за допомогою вказівників або посилань. Вказівник на клас-предок ініціалізується адресою об'єкта-нащадка, а посилання на клас-предок ініціалізується об'єктом класу-нащадка.

- у межаж поліморфічного кластера віртуальний метод не повинен змінювати свій тип, кількість або типи параметрів. Якщо віртуальний метод не перевизначається у похідному класі, то діє віртуальний метод попереднього в ієрархії успадкування класу.

1.3. Виключення поліморфізму віртуальних методів

Для виключення дії механізму пізнього зв'язування при виклику віртуального методу необхідно вказати назву класу, до якого він належить:

```
вказівник->назва класу::назва_методу(аргументи);
посилання.назва_класу::назва_мнтоду(аргументи);
```

Наприклад:

```
#include <iostream>
using namespace std;

class A {
public:
  virtual void f() {cout<<"A::f()"<<endl;}
};
class B: public A {
public:
  virtual void f() {cout<<"B::f()"<<endl;}
};

int main() {
  A *p =new B;
  p->A::f();  // A::f()
  p->f();     // B::f()
return 0;
}
```

В результаті викликається метод `A::f()` базового класу, а не метод `B::f()` похідного класу.

1.4. Коваріантні віртуальні методи

Віртуальний метод не повинен змінювати назву, кількість та типи параметрів і тип результату у межах свого поліморфного кластера.

Зміна типу результату можлива лише для коваріантних віртуальних методів. Методи є *коваріантними*, якщо їх типи є узгодженими з типами класів в іншій ієрархії успадкувань.

```
#include <iostream>
using namespace std;

class B1 {
public:
  virtual void f1() {cout<<"B1::f1()"<<endl;}
};
class D1:public B1 {
public:
  virtual void f1() {cout<<"D1::f1()"<<endl;}
};

class B2 {
public:
  virtual B1* f2(){cout<<"B2::f2()"<<endl; return new B1;}
};

class D2:public B2 {
```

```
public:
  virtual D1* f2(){cout<<"D2::f2()"<<endl; return new D1;}
};

int main() {
  B2* p=new B2;
  p->f2()->f1();  // B2::f2()
                 // B1::f1()
  B2* q=new D2;
  q->f2()->f1();  // D2::f2()
                  // D1::f1()
  delete p;
  delete q;
  return 0;
}
```

У програмі оголошено дві ієрархії успадкувань класів: `B1<-D1` та `B2<-D2`. Віртуальний метод базового класу `B2` повертає вказівник на базовий клас `B1`, а віртуальний метод похідного класу `D2` – на похідний клас `D1`. Ці вказівники використано для викликів віртуальних методів.

Віптуальні методи є також коваріантними, якщо їхні типи є вказівниками або посиланнями на класи у межах дії поліморфічного кластера, наприклад:

```
#include <iostream>
using namespace std;

class A {
protected:
  int x;
public:
  A(int x=0) {this->x=x;}
  virtual A& inc(){++x;return *this;}
  virtual void output() {cout<<x<<endl;}
};
class B:public A {
  int y;
public:
  B(int x=0, int y=0):A(x){this->y=y;}
  virtual B& inc() { ++x;++y; return *this;}
  virtual void output() {cout<<x<<' '<<y<<endl;}
};

int main() {
  A& a = *new B(2,7);
  a.inc();
  a.output();   // 3 8
  return 0;
}
```

Віртуальний метод `inc()` відрізняється типом результату в ієрархії успадкування класів. У класі `A` він має тип посилання на `A`, а у класі `B` – тип посилання на `B`. Ці типи є узгодженими в ієрархії успадкування класів (у межах поліморфічного кластера). Незважаючи на це, механізм дії пізнього зв'язування не порушується.

1.5. Віртуальні операторні методи

Операторні методи класів можуть бути віртуальними. Для ілюстрації цього модифікується попередня програма, в якій замінюється віртуальний метод `inc()` операторним методом префіксного інкременту.

```
#include <iostream>
using namespace std;
class A {
protected:
  int x;
```

```
public:
  A(int x=0) {this->x=x;}
  virtual A& operator++(){++x;return *this;}
  virtual void output() {cout<<x<<endl;}
};
class B:public A {
  int y;
public:
  B(int x=0, int y=0):A(x){this->y=y;}
  virtual B& operator++(){ ++x;++y; return *this;}
  virtual void output() {cout<<x<<' '<<y<<endl;}
};

int main() {
  A& a = *new B(2,7);
  ++a;
  a.output();   // 3 8
  return 0;
}
```

У функції `main()` посилання на базовий клас `A` проініціалізовано об'єктом похідного класу `B`. Операція префіксного інкременту, застосована до такого посилання, призведе до виклику віртуального операторного методу `B::operator++()` з похідного класу.

У цьому прикладі віртуальні методи мають коваріантні типи результатів. Оскільки посилання на базовий клас може бути проініціалізоване об'єктом похідного класу, то результат роботи не зміниться, якщо у класі `B` операторний метод матиме тип результату, тотожний типу результтату операторного методу з класу `A`.

1.6. Вказівники на віртуальні методи

Поліморфізм може бути реалізований також за допомогою вказівників на віртуальні методи. У наступній програмі віртуальний метод `Get()` визначає поліморфічний кластер у межах успадкування класів `A<-B`. Вказівник на базовий клас `A` проініціалізовано адресою об'єкта похідного класу `B` і використано для виклику віртуального методу `Get()` за допомогою вказівника `pf` на цей метод. Звернення до віртуального методу `Get()` за допомогою вказівника `pf` еквівалентне зверненню `p->Get()`.

```
#include <iostream>
using namespace std;

class A {
protected:
  int x;
public:
  A(int x=0) {this->x=x;}
  virtual void Get() {cout<<x<<endl;}
};
class B:public A {
  int y;
public:
  B(int x=0, int y=0):A(x){this->y=y;}
  virtual void Get() {cout<<x<<' '<<y<<endl;}
};

void (A::*pf) ()=&A::Get; // PIPєP°P·C–PIPSPёPє PSP° PIC–CЂC,CґP°P»CЊPSPёPЅ
PjPµC,PsPґ

int main() {
  A *p = new B(1,2);
  //p->Get();
  (p->*pf) ();  //  1 2
  delete p;
```

```
  return 0;
}
```

1.7. Динамічні віртуальні методи

Динамічні віртуальні методи – це підклас віртуальних методів, який відрізняється способом виклику на етапі виконання. Оголошуються за допомогою індексу динамічного методу: `virtual void f(void)=[100];`

Індекс задається у квадратних дужках константою цілого типу. Він повинен бути унікальним серед індексів інших динамічних методів, але однаковим в ієрархії успадкування конкретного методу.

Якщо клас оголошує або успадковує віртуальні методи, то для його об'єктів автоматично під час компіляції створюється таблиця віртуальних методів (ТВМ). У ТВМ записуються адреси усіх віртуальних методів. Для виклику віртуальних методів спочатку з об'єкта читається адреса ТВМ, а потім з неї вибирається адреса потрібного віртуального методу та здійснюється її виклик.

Для роботи з динамічними віртуальними методами створюється додаткова таблиця динамічних віртуальних методів (ТДВМ). Адреса ТДВМ записується на початку ТВМ. У ТДВМ записуються тільки адреси тих динамічних віртуальних методів, які визначаються або перевизначаються на даному рівні успадкування. У результаті ТДВМ займає менший обсяг оперативної памяті, ніж ТВМ. Однак для пошуку адреси динамічного віртуального методу витрачається більше часу, ніж у ТВМ.

Функція `main()` демонструє виклик динамічного віртуального методу за допомогою посилання та вказівника на клас `A`, проініціалізованих відповідно іменем та адресою об'єкта класу `B`. В обох випадках викликається віртуальний метод з похідного класу `B`.

Примітка. Підтримка динамічних віртуальних методів у версії g++ 4.8.1 не реалізована.

```
#include <iostream>
using namespace std;

class A {
public:
  virtual void f(void)=[100];
};
void A::f(void) { cout<<"A::f"<<endl;}

class B:public A {
public:
  virtual void f(void)=[100];
};
void B::f(void) { cout<<"B::f"<<endl;}

int main() {
 B b;
 A*p=&b;
 p->f();    // B::f()
 A &a=b;
 a.f();     // B::f()
  return 0;
}
```

1.8. Віртуальний деструктор

При роботі з вказівниками та посиланнями на тип базового класу, які ініціалізуються об'єктами похідних класів, у базовому класі бажано використовувати віртуальний деструктор. Віртуальний деструктор започатковує поліморфічний кластер. Якщо базовий клас містить віртуальний деструктор, то деструктор похідного від нього класу теж буде віртуальним.

```
#include <iostream>
#include <cstdlib>
```

```
using namespace std;

class A {
protected:
  int *p;
public:
  A(int x=0) {cout<<"A(int)"<<endl; p=new int(x); }

  virtual
 ~A() {cout<<"~A()"<<endl; delete p;}
  virtual void print(){cout<<*p<<endl;}
};

class B:public A {
  int *q;
public:
  B(int x=0,int y=0):A(x) {cout<<"B(int,int)"<<endl; q=new int(y);}
 ~B() {cout<<"~B()"<<endl; delete q;}
  virtual void print() {cout<<*p<<' '<<*q<<endl;}
};

int main() {
  //cout<<hex;
 // cout<<coreleft()<<endl;
  A *p=new B(1,2);
  p->print();
 // cout<<coreleft()<<endl;
  delete p;
  //cout <<coreleft()<<endl;
  return 0;
}

// A(int)
// B(int,int)
// 1 2
// ~B() СЦРєС‰Ps virtual ~A()
// ~A()
```

Якщо деструктор `~A()` не віртуальний, то при звільненні динамічної памяті, закріпленої за вказівником `p`, викликається тільки деструктор `~A()`, що може привести до втрати ресурсів похідного класу `B`.

Якщо деструктор `~A()` віртуальний, то викликаються деструктор `~B()`, а потім деструктор `~A()`, і об'єкт класу `B` повністю знищується. Зміну розміру динамічної памяті можна відстежити за допомогою функції `coreleft()` (в g++ відсутня).

1.9. Чисті віртуальні методи та абстрактні класи

Чисто віртуальна функція – це функція-член (метод) класу, для якої оголошено лише інтерфейс, а реалізація знаходиться в одному з похідних класів. Чистий віртуальний метод оголошується за допомогою специфікатор `= 0`.

Чистий віртуальний метод не може бути викликаний явно або неявно з конструктора.

Абстрактний клас містить один або декілька чистих віртуальних методів. Чистий абстрактний клас складається тільки з чистих віртуальних методів. Основне призначення абстрактного класу – це оголошення інтерфейсу похідних від нього класів.

Абстрактний клас може використовуватися тільки як базовий для інших класів. Неможливо оголосити об'єкт абстрактного класу, але можна оголосити вказівник на нього. Можна оголосити посилання на абстрактний клас, якщо воно ініціалізується об'єктом похідного класу.

Якщо похідний від абстрактного клас не визначає всіх чистих віртуальних методів, то він

теж є абстрактним.

```
#include <iostream>
using namespace std;

class A {     // абстрактний клас
protected:
  int x;
public:
  A(int x1):x(x1){}
  virtual void output() const=0; // чистий ВМ
};

class B:public A {  // абстрактний клас
public:
  B(int x):A(x){}
  virtual void output() const=0;
};

class C:public B {
public:
  C(int x):B(x){}
  virtual void output() const {cout<<x<<endl;}
};

int main() {
  C c(3);
  c.output();    // 3
  A *p=&c;
  B *q=&c;
  p->output();   // 3
  q->output();   // 3
}
```

Параметри та результат функції не можуть мати тип абстрактного класу. Однак вони можуть бути вказівниками або посиланнями на абстрактний клас:

```
#include <iostream>
using namespace std;

class B{
public:
  virtual void f()=0;
};
class D:public B {
public:
  virtual void f() {cout<<"D::f()"<<endl;}
};
B& ref_B(B* p, B& r) {
  //...
  return r;
}
int main() {
  B *p = new D;
  B &r = *new D;
  ref_B(p,r).f();
  delete p;
  delete &r;
  return 0;
}
// D::f()
```

1.10. Приклад програми застосування абстрактних класів

Оголосити абстрактний клас з віртуальною функцією Площа. Оголосити похідні класи – Трикутник, Прямокутник та Круг, у яких визначити функції обчислення площі:

- трикутника – $S=[p(p-a)(p-b)(p-c)]^{0.5}$, де $p=(a+b+c)/2$, a,b,c – сторони трикутника;
- прямокутника – $S=xy$, де x,y – сторони прямокутника;
- круга – $S=\pi r^2$, де r – радіус.

Реалізувати механізм пізнього зв'язування за допомогою масиву вказівників на абстрактний клас, елементам якого присвоєно адреси об'єктів похідних неабстрактних класів. Використати вказівники для виклику віртуального методу.

```
#include <iostream>
#define _USE_MATH_DEFINES
#include <cmath>
#include <iomanip>
using namespace std;

class Figure {
public:
  virtual float Area()=0;
};

class Triangle:public Figure {
  float a,b,c;
public:
  Triangle(float a1=0, float b1=0, float c1=0) {
    a=a1; b=b1;c=c1;
  }
  virtual float Area() {
    cout<<"Площа трикутника: ";
    float p=(a+b+c)/2;
    return sqrt(p*(p-a)*(p-b)*(p-c));
  }
};

class Rectangle:public Figure {
  float x,y;
public:
  Rectangle(float x1=0, float y1=0) {x=x1;y=y1;}
  virtual float Area() {
    cout <<"Площа прямокутника: ";
    return x*y;
  }
};

class Circle:public Figure {
  float r;
public:
  Circle(float r1=0) {r=r1;}
  virtual float Area() {
    cout<<"Площа круга: ";
    return M_PI*r*r;
  }
};

float GetArea(Figure* f) { return f->Area();}

int main() {
  Figure* p[3]={
    new Triangle(5,8,7),
    new Rectangle(3,6),
    new Circle(4)
  };
  cout.setf(ios::fixed);
  cout.precision(2);
```

```
  for(int i=0; i<3; ++i)
  cout << GetArea(p[i]) << endl;
  return 0;
}
// Площа трикутника: 17.32
// Площа прямокутника: 18.00
// Площа круга: 50.27
```

Запитання.

1. Що таке поліморфізм і які є його типи.
2. Що таке раннє і пізнє звязування.
3. Який метод називається віртуальним і які його особливості.
4. Як виключити поліморфізм віртуальних методів.
5. Які віртуальні методи є коваріантними.
6. Що таке выртуальны операторні методи.
7. Як реалізувати вказівники на віртуальні методи
8. Динамічні віртуальні методи.
9. Особливості застосування віртуальних деструкторів.
10. Що таке чисті віртуальні методи і для чого вони використовуються.
11. Що таке абстрактний і чистий абстрактний класи.
12. Пояснити приклад програми застосування абстрактних класів

Завдання.

1. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – площа. Створити похідні класи: ПРЯМОКУТНИК, КОЛО, ПРЯМОКУТНИЙ ТРИКУТНИК, ТРАПЕЦІЯ зі своїми функціями площі. Для перевірки викликів віртуальних функцій визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів.

2. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – норма. Створити похідні класи: КОМПЛЕКСНІ ЧИСЛА, ВЕКТОР, МАТРИЦЯ. Визначити функцію норми: для комплексних чисел – модуль комплексного числа, для вектора – корінь квадратний із суми квадратів елементів, для матриці – максимальне значення за модулем. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

3. Створити абстрактний клас (КРИВІ) обчислення залежності y від x. Створити похідні класи: ПРЯМА, ЕЛІПС, ГІПЕРБОЛА зі своїми функціями обчислення $y(x)$ залежно від вхідного параметра x. Ріняння прямої: $y=ax+b$; еліпса: $x^2/a^2+y^2/b^2=1$, де a – велика піввісь, b – мала піввісь; гіперболи: $x^2/a^2-y^2/b^2=1$, де a – дійсна піввісь, b – уявна піввісь. Для перевірки визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

4. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – сума прогресії. Створити похідні класи: АРИФМЕТИЧНА ПРОГРЕСІЯ і ГЕОМЕТРИЧНА ПРОГРЕСІЯ. Кожен клас маєдва поля типу double. Перше – перший елемент прогресії, друге – постійна різниця для арифметичної і постійне відношення для геометричної прогресії. Визначити функцію обчислення суми, де параметром є кількість елементів прогресії. Арифметична прогресія $a_j=a_0+jd,\ j=0,1,2,\ldots$.Сума арифметичної прогресії $s_n=(n+1)(a_0+a_n)/2$. Геометрична прогресія: $a_j=a_0r^j,\ j=0,1,2,\ldots$. Сума геометричної прогресії: $s_n=(a_0-a_nr)/(1-r)$. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

5. Створити базовий клас – СПИСОК. Реалізувати на базі списку СТЕК і ЧЕРГУ з віртуальними функціями включення і вилучення елементів. Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуальних функцій.

6. Створити базовий клас – геометрична ФІГУРА, і похідні класи – КОЛО,

ПРЯМОКУТНИК, ТРАПЕЦІЯ. Визначити віртуальні функції визначення площі, периметра і виведення на екран. Площа круга $S=\pi r^2$ (r – радіус); площа прямокутника $S=ab$ (a, b – сторони); площа трапеції $S=(a+b)h/2$ (a, b – основа трапеції, h – висота). Довжина кола $L=2\pi r$; периметр прямокутника $L=2(a+b)$; периметр трапеції $L=a+b+c+d$ (a, b, c, d – сторони). Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуальних функцій.

7. Створити базовий клас – ПРАЦІВНИК і похідні класи – СЛУЖБОВЕЦЬ З ПОГОДИННОЮ ОПЛАТОЮ, СЛУЖБОВЕЦЬ З ОКЛАДОМ. Визначити віртуальну функцію нарахування зарплати. Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуальних функцій.

8. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – площа поверхні. Створити похідні класи: прямокутний ПАРАЛЕЛЕПІПЕД, ТЕТРАЕДР, КУЛЯ зі своїми функціями площі поверхні. Площа поверхні паралелепіпеда: $S=2(ab+bc+ca)$, де a, b, c – ребра. Площа поверхні кулі: $S=4\pi r^2$, де r – радіус. Площа поверхні тетраедра: $S=a^2s^{0.5}$, де a – довжина ребра. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

9. Створити абстрактний клас – ССАВЦІ. Визначити похідні класи – ТВАРИНИ і ЛЮДИ. У тварин визначити похідні класи КОНЕЙ і КОРІВ. Визначити віртуальні функції опису людини, коня і корови. Для перевірки пізнього зв'язування визначити масив вказівників на абстактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

10. Створити базовий клас – БАТЬКО, у якого є ім'я. Визначити віртуальну функцію виведення імені на екран. Створити похідний клас ДИТИНА, у якої є імя та успадковане поле по батькові. Для перевірки пізнього зв'язування визначити масив вказівників на базовий клас, яким присвоюються адреси об'єктів створених класів. Використати вказівники для виклику віртуального методу.

12. Створити абстрактний БАЗОВИЙ клас з віртуальною функцією – корені рівняння. Створити похідні класи: клас ЛІНІЙНИХ РІВНЯНЬ і клас КВАДРАТНИХ РІВНЯНЬ. Визначити функцію обчислення коренів рівнянь. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюються адреси об'єктів неабстрактних класів. Використати вказівники для виклику віртуальної функції.

13. Створити абстрактний клас для роботи з геометричними ФІГУРАМИ на екрані. У захищеній частині класу знаходяться такі дані: координати центра фігури; кут повороту (у градусах); масштабний фактор. У відкритій частині розміщено функції-методи: зобразити фігуру на екрані; зробити фігуру невидимою; повернути фігуру на заданий кут (задається у градусах); перемістити фігуру на заданий вектор. Застосовуючи успадкування та наведений вище абстрактний клас, створити похідні класи для роботи з фігурою: ТРИКУТНИК, ЧОТИРИКУТНИК, БАГАТОКУТНИК. Для перевірки пізнього зв'язування визначити масив вказівників на абстрактний клас, яким присвоюється адреси об'єктів похідних класів. Використати вказівники для виклику віртуальної функції.

2. Самостійна робота

```
/* listing 1 — виклик віртуальної функції за допомогою вказівника на об'єкти */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з базового класу.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

class derived2 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived2.\n";
  }
};

int main()
{
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник на базовий клас
  p = &b;
  p->vfunc(); // виклик функції vfunc() з класу base()
  // вказівник на об'єкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції vfunc() з класу derived1

  // вказівник на об'єкт класу derived2
  p = &d2;
  p->vfunc(); // виклик функції vfunc() з класу derived2

  return 0;
}
Функція vfunc() з базового класу.
Функція vfunc() з класу derived1.
Функція vfunc() з класу derived2.

/* listing 3  - викликати віртуальну функцію можна за допомогою посилання на об'єкт базового класу  */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};
```

```
class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1\n";
  }
};

class derived2 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived2\n";
  }
};

// Використовується параметр, який є посиланням на об’єкт базового класу
void f(base &r) {
  r.vfunc();
}

int main()
{
  base b;
  derived1 d1;
  derived2 d2;

  f(b);  // функції f() передається об'єкт класу base
  f(d1); // функції f() передається об'єкт класу derived1
  f(d2); // функції f() передається об'єкт класу derived2

  return 0;
}
```

```
Функція vfunc() з класу base.
Функція vfunc() з класу derived1
Функція vfunc() з класу derived2
```

```
/* listing 4 – успадкування атрибуту virtual */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

/* Клас derived2 успадковує віртуальну функцію vfunc() від класу derived1. */
class derived2 : public derived1 {
public:
  // функція vfunc() залишається віртуальною
  void vfunc() {
    cout << "Функція vfunc() з класу derived2.\n";
  }
};

int main()
{
```

```
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник на обєкт класу  base
  p = &b;
  p->vfunc(); // виклик функції vfunc() з класу base

  // вказівник на обєкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції vfunc() з класу derived1

  // вказівник на клас  derived2
  p = &d2;
  p->vfunc(); // виклик функції vfunc() з класу derived2

  return 0;
}
```

Функція vfunc() з класу base.
Функція vfunc() з класу derived1.
Функція vfunc() з класу derived2.

/* listing 5 - віртуальні функції є ієрархічними. Віртуальну функцію не обов'язково заміщати. */

```
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

class derived2 : public base {
public:
// функція не заміщується в класі derived2, а використовується версія з класу base
};

int main()
{
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник тна об'єкт класу base
  p = &b;
  p->vfunc(); // виклик функції  vfunc() з класу base

  // вказівник на об'єкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції vfunc() з класу derived1

  // вказівник на об'єкт класу derived2
  p = &d2;
  p->vfunc(); // виклик функції vfunc() з класу base
```

```
  return 0;
}
```
Функція vfunc() з класу base.
Функція vfunc() з класу derived1.
Функція vfunc() з класу base.

```
/* listing 6 - віртуальні функції є ієрархічними. Якщо віртуальна функція не
заміщається то викликається її попередня версія  */
#include <iostream>
using namespace std;

class base {
public:
  virtual void vfunc() {
    cout << "Функція vfunc() з класу base.\n";
  }
};

class derived1 : public base {
public:
  void vfunc() {
    cout << "Функція vfunc() з класу derived1.\n";
  }
};

class derived2 : public derived1 {
public:
/* Функція vfunc() не заміщається в класі derived2.
   Так як клас derived2 є наслідником класу derived1,
   то викликається функція vfunc() з класу derived2 */
};

int main()
{
  base *p, b;
  derived1 d1;
  derived2 d2;

  // вказівник на об'єкт класу base
  p = &b;
  p->vfunc(); // виклик функції з класу base

  // вказівник на об'єкт класу derived1
  p = &d1;
  p->vfunc(); // виклик функції з класу derived1

  // вказівник на об'єкт derived2
  p = &d2;
  p->vfunc(); // функція vfunc() з класу derived1

  return 0;
}
```
Функція vfunc() з класу base.
Функція vfunc() з класу derived1.
Функція vfunc() з класу derived1.

```
/* listing 7 — чисто віртуальна функція не має визначення у базовому класі */
#include <iostream>
using namespace std;

class number {
protected:
  int val;
public:
```

```
  void setval(int i) { val = i; }

  // функція show() є чисто віртуальною, признак  show() = 0;
  virtual void show() = 0;
};

class hextype : public number {
public:
  void show() {
    cout << hex << val << " ";
  }
};

class dectype : public number {
public:
  void show() {
    cout << val << " ";
  }
};

class octtype : public number {
public:
  void show() {
    cout << oct << val << "\n";
  }
};

int main()
{
  dectype d;
  hextype h;
  octtype o;

  d.setval(20);
  d.show(); // виводить десяткове число 20

  h.setval(20);
  h.show(); // виводить шістнадцяткове число 14

  o.setval(20);
  o.show();  // виводить вісімкове число 24

  return 0;
}
```

20 14 24

/* listing 8 - застосування віртуальних функцій у ієрархії класів. Клас, який містить хоча б одну чисто віртуальну функцію називається абстрактним. Так як віртуальні функції не мають визначення, то створити об'єкт абстрактного класу неможливо. Абстрактні класи можуть бути тільки основою для похідних класів. */

```
#include <iostream>
using namespace std;

class convert {     // абстрактний клас
protected:
  double val1;  // початкові значення
  double val2;  // перетворене значення
public:
  convert(double i) {
    val1 = i;
  }
  double getconv() { return val2; }
  double getinit() { return val1; }
```

```
  virtual void compute() = 0;    // чисто віртуальна функція
};

// перетворення літрів в галони
class l_to_g : public convert {
public:
  l_to_g(double i) : convert(i) { }
  void compute() {
    val2 = val1 / 3.7854;
  }
};

// перетворення шкали Фарангейта в шкалу Цельсія
class f_to_c : public convert {
public:
  f_to_c(double i) : convert(i) { }
  void compute() {
    val2 = (val1-32) / 1.8;
  }
};

int main()
{
  convert *p;  // вказівник на базовий клас

  l_to_g lgob(4);
  f_to_c fcob(70);

  // застосування віртуальної функції для конвертації
  p = &lgob;
  cout << p->getinit() << " літри дорівнює ";
  p->compute();
  cout << p->getconv() << " галонів\n";  // l_to_g

  p = &fcob;
  cout << p->getinit() << " по Фарангейту дорівнює ";
  p->compute();
  cout << p->getconv() << " по Цельсію\n";  // f_to_c

  return 0;
}
```

4 літри дорівнює 1.05669 галонів
70 по Фарангейту дорівнює 21.1111 по Цельсію