Lecture 0xA

ООП 2: Наслідування

1. **Перезавантаження операторів**
2. **Наслідування:**
   - Приклад наслідування
   - Типи наслідування C++
   - Поліморфізм
3. **Віртуальні методи. Абстрактні класи**
4. **Множинне наслідування**
5. **Віртуальне наслідування**

Перезавантаження операторів

Оператори, що не перевантажуються

:: (визначення області, scope resolution),

. (оператор доступу, member access),

.* (оператор доступу по вказівнику, member access through pointer to member),

?: (тернарний, ternary conditional)

Перезавантаження операторів

```
using namespace std;

class Vector2D {//клас вектор
private: // приватні члени
    double x;
    double y;
public: // публічні методи
    Vector2D() {} // конструктори
    Vector2D(double x, double y):x(x),y(y) {}
// перевантаження бінарного мінуса
    Vector2D operator-(const Vector2D& b) {
        return Vector2D(x-b.x, y-b.y);
    }
// перевантаження унарного мінуса
    Vector2D operator-(void) {
        return Vector2D(-x, -y);
    }
    void show() { // метод виведення
        cout<<x<<","<<y<<endl;
    }
};
```

```
int main()
{
   // створили 2
вектори
    Vector2D a(1,1);
    Vector2D b(1,1);
    Vector2D c,d;
   // додали
    c = a + b;
   // показали
    c.show();//2,2
}
```

Перзавантаження cin>>, cout<<

```
#include <iostream>
#include <fstream> // використовуємо ostream,  istream
using namespace std;
class Vector2D{
    double x;
    double y;
public:
    Vector2D() {} // конструктор за замовченням
    Vector2D(double x, double y):x(x),y(y) {} // конструктор по значенням
                        /* дружній оператор cout<< */
friend ostream &operator<<( ostream &output, const Vector2D &D ) {
        output << "(" << D.x << " , " << D.y << ")";
        return output;
    }
/* дружній оператор cin>> */
friend istream &operator>>( istream  &input, Vector2D &D ) {
        input >> D.x >> D.y;
        return input;
    }
}
```

Перезавантаження порівнянь

```
class Vector2D{
    double x;
    double y;
    static double eps(void); // статичний метод для точності рівності
public:
    Vector2D() {}
    Vector2D(double x, double y):x(x),y(y) {}
****
Vector2D operator=(const Vector2D& b) { // оператор присвоєння
        x = b.x;
        y = b.y;
        return Vector2D(x,y);
    }  // оператор порівняння !=
friend bool operator!=(const Vector2D& left, const Vector2D& right) {
        return (abs(left.x-right.x)>=eps()) || (abs(left.y-right.y)>=eps());
    }  // оператор порівняння ==
bool operator==(const Vector2D& right) {
        return (abs(x-right.x)< eps()) && (abs(y-right.y)<eps());
    }
```

Використання перевантаження

```
double Vector2D::eps(void){
    return 0.0001; // точність
}
// константа (0,0)
const Vector2D ZERO(0,0);

int main()
{
    Vector2D a(1,1);
    Vector2D b(1,1);
    Vector2D c,d;
 // порівняння
    if(a!=b){
        cout<<"a!=b"<<endl;
    }else{
        cout<<"a==b"<<endl;
    }
```

```
c = a + b; // додавання
    c.show();
    c = c — a; // віднімання
    cout<<c<<endl; // ОК
    cin>>c; // ввели вектор
   // порівняли його з нулем
    if(-c==ZERO){
        cout<<"zero";
    }
    else{
        cout<<c;
    }
}
/*Результат: a==b
2,2
(1 , 1)
3 4
(3 , 4)
*/
```

Наслідування (inheritance)

Наслідування: передає **всі члени та методи**, що **не є private** класу-нащадку, за ***виключенням***:

- ***конструкторів, деструкторів та конструкторів копіювання*** базового класу;
- ***перевантажених операторів*** базового класу;
- ***дружніх функцій*** базового класу.

Нащадок Vector2D (private:= protected)

```
class NamedVector2D: public Vector2D { //NamedVector2D — нащадок вектору
private: // до членів x, y додали ще два члени
    string name_x;
    string name_y;
public: // конструктори потрібні нові
    NamedVector2D(){}
    NamedVector2D(double x1, double y1): x(x1),y(y1),name_x(“x"), name_y ("y"){  } // можна
:Vector2D (x1,y1),name_x(“x"), name_y ("y")
    NamedVector2D(double x1, double y1, string name, string
name2)x(x1),y(y1),name_x(“x"), name_y ("y"){  }
   // оператори потрібно переписати
   NamedVector2D operator+(const NamedVector2D& b) {
        return NamedVector2D(x+b.x, y+b.y, name_x, name_y );
    }
   // show() зміниться — тому перевантажимо його
    void show() {
        cout<<name_x<<","<<name_y<<endl; // виводимо назви
        cout<<this->x<<","<<this->y<<endl; // Vector2D::show()
    }
};
```

Використання класу нащадка

```
int main(){
    NamedVector2D z(2,2);
    NamedVector2D t(3,4,"x","y");
    NamedVector2D w("X","Y");
    z.show();
    t.show();
    w = z + t;
    w.show();
}
/*
x,y
2,2
x,y
3,4
X,Y
5,6
*/
```

Типи доступу при наслідуванні

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

class BaseA{
private:
    int private_a;
    void private_method() {
        cout<<"Use a";
    }
protected:
    int protected_b;
    void protected_method() {
        cout<<"Use b";
    }
public:
    int public_c;
    BaseA():private_a(1),protected_b(2), public_c(3) {}//ініціалізація
    void public_method() {
        cout<<"Use c";
    }
};
```

Загальнодоступне наслідування

```
class InheritPublic: public BaseA{
public:
    void use() {
        //cout<<  private_a;
        cout<<  protected_b;
        cout<<  public_c;
        //private_method();
        protected_method();
        public_method();
    }
};

Int main(){
    BaseA p1;
    p1.public_method();
    cout<<p1.public_c;
    InheritPublic p2;
    p2.public_method();
    cout<<p2.public_c;
    p2.use();
}
```

```
class InheritPublic2: public InheritPublic{
public:
    void use() {
        //cout<<  private_a;
        cout<<  protected_b;
        cout<<  public_c;
        //private_method(); нема доступу
        protected_method();
        public_method();
    }
};

***
 InheritPublic2 p5;
    p5.use();
    cout<<p5.public_c;
***
```

Приватнє наслідування

```
class InheritPrivate: private BaseA{
public:
    void use() {
        //cout<<  private_a; нема доступу
        cout<<  protected_b;
        cout<<  public_c;
        //private_method(); нема доступу
        protected_method();
        public_method();
    }
};
class InheritPrivate2: private InheritPrivate{
public:
    void use() {
        //cout<<  private_a; нема доступу
        //cout<<  protected_b; ні до чого ;-((
        //cout<<  public_c;
        //private_method();
        //protected_method();
        //public_method();
    }
};
```

```
InheritPrivate p4;
    p2.public_method();
    cout<<p2.public_c;
    p2.use(); // є доступ

    InheritPrivate2 p7;
    p7.use();
    //cout<<p7.public_c; ///
нема
```

Захищене наслідування

```
class InheritProtected: protected BaseA{
public:
    void use() {
        //cout<<  private_a; // нема доступу
        cout<<  protected_b;
        cout<<  public_c;
        //private_method(); // нема доступу
        protected_method();
        public_method();
    }
};
class InheritProtected2: protected InheritProtected{
public:
    void use() {
        //cout<<  private_a; //нема доступу
        cout<<  protected_b; //
        cout<<  public_c;
        //private_method(); // нема доступу
        protected_method();
        public_method();
    }
};
```

```
    InheritProtected p3;
    //p3.public_method(); нема
доступу
    //cout<<p3.public_c;
    p3.use();

    InheritProtected2 p6;
    p6.use();
    //cout<<p6.public_c; нема
доступу
```

```
#include <iostream>
using namespace std;
 class Shape {
   protected:
     int width, height;
   public:
     Shape( int a = 0, int b = 0){
        width = a;  height = b;
     }
     int area() {
        cout << "Parent class area :" <<endl;
        return 0;
     }
};
```

```
class Rectangle: public Shape {
   public:
     Rectangle( int a = 0, int b = 0):
                                  Shape(a, b) { }

     int area () {
        cout << "Rectangle class area :" <<endl;
        return (width * height);
     }
};

 int main() {
    Shape shape ;
    Rectangle rec(10,7);

     shape.area();
     rec.area();
 }
```

Раннє зв’язування

```
class Triangle: public Shape {
   public:
      Triangle( int a= 0, int b = 0):
                                  Shape(a, b) { }

      int area () {
         cout << "Triangle class area :"
<<endl;
         return (width * height / 2);
      }
};

int main2(){
    Shape * s1 = new Shape;
    Rectangle *r1 = new Rectangle(2,3);
    s1->area(); r1->area(); // 0, 6
   Shape * s2 = new Rectangle(2,3);
    //Rectangle *r2 = new Shape;  заборонено!!!!
    s2->area(); // 0 !!!! Буде викликаний Shape
}
```

```
int main() {
   Shape *shape;
   Rectangle r1 = rec(10,7);
   Triangle  tri(10,5);

   // беремо адресу Rectangle
   shape = &rec;
   // отримуємо його площу
   shape->area();// Parent class area

   // беремо адресу Triangle
   shape = &tri;
     // отримуємо його площу
   shape->area();// Parent class area
}
```

Пізнє зв’язування

```
class Shape {
   protected:
      int width, height;
    public:
      Shape(int a = 0, int b = 0) {
         width = a;
         height = b;
      }

      // чисто віртуальна функція (pure virtual function)
      virtual int area() = 0;
};

int main(){
   Shape * s2 = new Rectangle(2,3);
   //Shape s2;  // Нема, тому що є чисто віртуальна функція – отже абстрактний клас
   cout<<s2->area(); // 6  метод Rectangle.area();
   delete s2;
}
```

Приклад віртуального методу

```
#include <iostream>
#include <cmath>
using namespace std;
class Figure{
      public:
         virtual double area() = 0;
         virtual ~Figure() {} =0; // віртуальний
деструктор
};
class Circle : public Figure{
      double r;
 public:
    Circle() {}
    Circle(double r_):r(r_){}
    double area();
};
class Rectangle: public Figure{
    double w;    double h;
public:    Rectangle() {}
Rectangle(double a, double b):w(a),h(b){}
    double area();};
```

```
Figure::~Figure() {} // віртуальний
деструктор

double Circle::area(){
    return M_PI*r*r;
}
double Circle::perimeter(){
    return 2*M_PI*r;}

double Rectangle::area(){
    return w*h;
}
double Rectangle::perimeter(){
    return 2*(w+h);
}
```

Використання віртуального методу

```
Int main(){
 ****
Circle c = Circle(4);
cout<<c.area()<<endl;

Rectangle r = Rectangle(5,5);
cout<<r.area()<<endl;

Figure* f2;
 f2 = &c; // OK. upcast
 cout<<"F"<<f2->area()<<endl;
 f2 = &r; // OK. upcast
 cout<<"F"<<f2->area()<<endl;
Figure* fgs =  new Circle(2);
  //fgs = &c; Not Ok: downcast
cout<<"F"<<fgs->area()<<endl;
delete fgs;
```

```
Figure* fg2[20];
for(int i=0; i<2; ++i) {
  if(i%2 ==0){
        fg2[i] = new Circle(1);
   }
  else{
        fg2[i] = new Rectangle(2,3);
   }
}

for(int i=0; i<20; ++i) {
      cout<<fg2[i]->area()<<endl;
}

for(int i=0; i<20; ++i) {
        delete fg2[i];
}
```

```
#include <iostream.h>
#include <string.h>
class monitor
{
public:
   monitor(char *, char *, int, int);
   void show_monitor(void);
protected:
   char type[32];
   char colors[15];
   int x;
   int y;
};
```

```
monitor::monitor(char *type, char *colors, int x, int y)
{
   strcpy(monitor::type, type);
   strcpy(monitor::colors, colors);
   monitor::x = x;
   monitor::y = y;
}
void monitor::show_monitor(void)
{
   cout << "Тип екранн: " << type << endl;
   cout << “Кольори: " << colors << endl;
   cout << "Розд.Здатн: " << x << " на " << y << endl;
}
```

```
class mother_board{
public:
   mother_board(char *, int, int);
   void show_mother_board(void);
protected:
   char processor[20];
   int speed;
   int RAM;
};
```

```
void mother_board::show_mother_board()
{
   cout << "Процессор: " << processor <<
endl;
   cout << "Частота: " << speed << " МГц"
<< endl;
   cout << "ОЗУ: " << RAM << " Мбайт" <<
endl;
}
```

```
mother_board::mother_board(char *proc, int speed, int RAM)
{
   strcpy(mother_board::processor, proc);
   mother_board::speed = speed;
   mother_board::RAM = RAM;
}
```

```
class computer :
 public monitor,  public mother_board {

public:
   computer(char *,int , float, char *,char *,
                        int,int,char *, int, int);
   void show_computer (void);
private:
   char name [64];
   int hard_disk;
   float floppy;
};
computer::computer(char *name, int hard_disk,
   float floppy, char *screen, char *colors, int x, int y,
   char *processor, int speed, int RAM) :
               monitor(screen, colors, x, y),
               mother_board(processor, speed, RAM){
   strcpy(computer::name, name);
   computer::hard_disk = hard_disk;
   computer::floppy = floppy;
}
```

```
 void computer::show_computer(void)
 {
    cout << "Тип: " << name << endl;
    cout<<"Жосткий диск: "<<hard_disk<<"
 Гбайт"<<endl;
    cout <<"Гибкий диск: " <<floppy<<"
 Мбайт"<<endl;
    show_mother_board();
    show_monitor();
 }

int main()  {
    computer my_pc("IBM", 120, 1.44,
"SVGA", “TC", 1600, 1200,"Celeron 3.2
ГГц", 533, 512);
      my_pc.show_computer();
    }
```

Проблема множинного наслідування: спільні методи

Problem 1:

```
class A { virtual void f(); };
class B { virtual void f(); };
class C : public A ,public B { void f(); }
```

c.f()??? - як викликати справжню?

Solution:

```
C* pc = new C;
pc->f();
pc->A::f(); //викличе f() з класу A
pc->B::f(); // викличе f() з класу B
```

Problem 2:

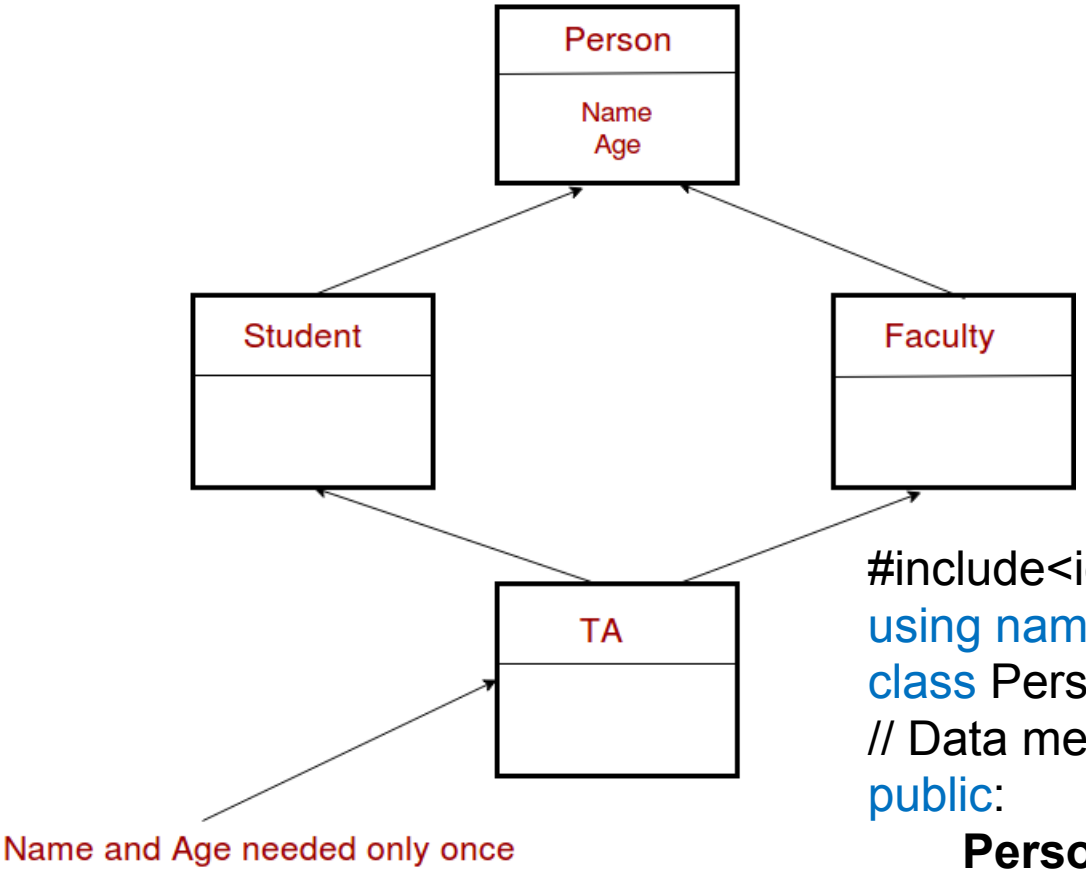
```
class A { virtual void f(); };
class B { virtual void f(); };
class C : public A ,public B {
//void f(); - немає f();
}
```

c.f()??? - яку викликати?

Solution:

```
C* pc = new C;
A* pa = pc;
pc->f();
```

Даймонд проблем




```
#include<iostream>
using namespace std;
class Person {
// Data members of person
public:
      Person(int x) {
cout << "Person::Person(int ) called" << endl; }
};
```

Якщо проблему не розв’язувати ….

```
class Faculty : public Person {
// data members of Faculty
public:
      Faculty(int x):Person(x) {
      cout<<"Faculty::Faculty(int ) called"<<
endl;
      }
};

class Student : public Person {
// data members of Student
public:
      Student(int x):Person(x) {
            cout<<"Student::Student(int )
called"<< endl;
      }
};
```

```
class TA : public Faculty, public Student {
public:
      TA(int x):Student(x), Faculty(x) {
            cout<<"TA::TA(int ) called"<< endl;
      }
};

int main() {
      TA ta1(30);
}
```

```
Person::Person(int ) called
Faculty::Faculty(int ) called
Person::Person(int ) called
Student::Student(int ) called
TA::TA(int ) called
```

Розв’язання: віртуальне наслідування

```
#include<iostream>
using namespace std;
class Person {
public:
	Person(int x) { cout << "Person::Person(int ) called" << endl; }
	Person() { cout << "Person::Person() called" << endl; }
};

class Faculty : virtual public Person {
public:
	Faculty(int x):Person(x) {
	cout<<"Faculty::Faculty(int ) called"<< endl;
	}
};

class Student : virtual public Person {
public:
	Student(int x):Person(x) {
		cout<<"Student::Student(int ) called"<< endl;
	}
};
```

Розв’язання: віртуальне наслідування

```
class TA : public Faculty, public Student {
public:
        TA(int x):Student(x), Faculty(x), Person(x) {
                cout<<"TA::TA(int ) called"<< endl;
        }
};

int main() {
        TA ta1(30);
}
```

```
Person::Person(int ) called
Faculty::Faculty(int ) called
Student::Student(int ) called
TA::TA(int ) called
```