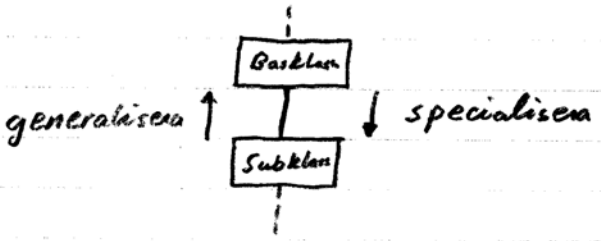
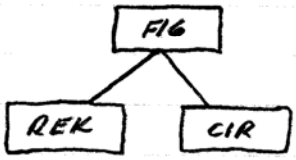
5/ Arv och dynamisk bindning

Arv – relation av typen är en is a



(Ex) En rektangel är en figur och en cirkel är en figur




```
// Fig.h

class FIG
{
  private:
    char namn[5];
  public:
    FIG(char *namn);
    void skriv();
};
```

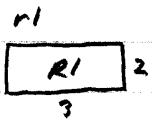
```
//Fig.cpp

FIG::FIG( char *namn)
{
    strcpy (this->namn, namn);
}

void FIG::skriv()
{
    cout << "Namn: " << namn << endl;
}


//Rek.h

class REK: public FIG //OBS! Arv
{
    private:
        float sid1, sid2;
    public:
        REK(char *namn, float sid1, float sid2);
        void skriv();
        float area();
};
```

```
//Rek.cpp

REK::REK(char *namn, float sid1, float sid2)
        : FIG(namn)  //Obs! Anrop av FIG-konstruktor
{
   this->sid1 = sid1;
   this->sid2 = sid2;
}

void REK::skriv()
{
   FIG::skriv();    //Alt om ut-operator definierad i FIG
                    // cout << FIG(*this)
   cout << "Sidor " << sid1 << " och " << sid2 << endl;
}

float REK::area()
{
   return sid1 * sid2;
}


//Cir.h

class CIR:public FIG
{
   private:
      float radie;
```

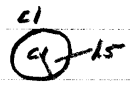
```
    public:
        CIR(char *namn, float radie);
        void skriv();
        float area();
};

//Cir.cpp

CIR::CIR(char *namn, float radie): FIG(namn)
{
    this->radie = radie;
}

void CIR::skriv()
{
    FIG::skriv();
    cout << "Radie: " << radie << endl;
}

float CIR::area()
{
    return M_PI * radie * radie;
}


//Figmain.cpp

void main()
```

```
{
    REK r1("R1", 2, 3);

    CIR c1("C1", 1.5);

    r1.skriv();
    cout << "Rekarea : " << r1.area() << endl;

    c1.skriv();
    cout << "Cirarea : " << c1.area() << endl;
}
```

r1

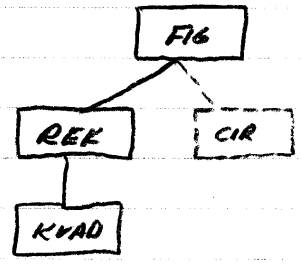


c1



Återanvändning - basklass finns gör en egen subklass

(Ex) En kvadrat är en rektangel med lika långa sidor




```
//Kvad.h

class KVAD : public REK
{
    public:
```

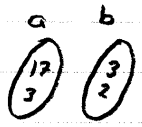
```
    KVAD(char *namn, float sid);
    void skriv();
};

// Kvad.cpp

KVAD::KVAD(char *namn, float sid)
        : REK(namn, sid, sid)   //OBS! REK-konstruktorn
{
}

void KVAD::skriv()
{
    FIG::skriv();
    cout << "Sida: " << sid1 << endl;
}

// Kvadmain.cpp

void main()
{
    KVAD k1("K1", 5.0);

    k1.skriv();
    cout << "Kvadarea: " << k1.area() << endl;
```

OBS! Måste markeras protected i REK-klassen för att KVAD-klassen ska komma åt sid1 direkt. Alternativt med funktion get_sid!

Letar efter area-funktion i KVAD-klassen. Finns ej! Går till basklassen REK och kör area-funktionen där!

(Ex) Ett allmänt bråk är ett bråk ex 5 2/3 = 17/3

```
//Abrak.h

class ABRAK : public RTAL
{
    public:
        ABRAK(int hel, int t, int n);
};

//Abrak.cpp

ABRAK::ABRAK(int hel, int t, int n)
        : RTAL(hel*n+t, n)
{
}

//Abrakmai.cpp

void main()
{
    ABRAK a(5,2,3), b(1,1,2);

    cout << (a+b) << endl;
```



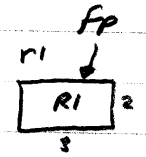
RTAL-klassen + körs

RTAL-klassens << körs

Dynamiek bindning

(Ex) Nytt huvudprogram

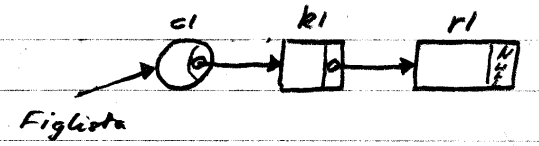
```
// Figmain.cpp

void main()
{
    REK r1("R1", 2, 3);
    FIG *fp = &r1;

    r1.skriv(); // skriva alla data
    fp->skriv(); // skriva endast namn
}
```



Genom att markera skriv-funktionen virtual i klassen FIG kommer skriv-funktionen bindas dynamiskt under körning och då kommer fp->skriv() att skriva ut all info om rektangeln som den pekar på.

```
// Fig.h

class FIG
{
  :
  virtual void skriv();
  :
};
```

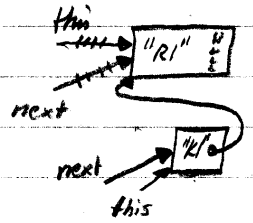
(Ex) Skapa en lista (stack) av figurer och beräkna totala arean.




```
//Fig.h

class FIG
{
    private:
        char namn[5];
        FIG *next;
    public:                          pekar reflexivt
        FIG(char *namn, FIG *&next);
        virtual float area() = 0; //Äkta virtuell
        FIG *get_next();
};
```

funktion som gör att klassen FIG blir abstrakt dvs. inga objekt av klass kan skapas.

```
//Fig.cpp

FIG::FIG(char *namn, FIG *&next)
{
    strcpy(this->namn, namn);
    this->next = next;
    next = this;
}
```



float FIG::area() ska ej implementeras
om åtta virtuell.
OBS! Måste dock implementeras
i alla subklasser annars
FIG *FIG::get_next() blir de också abstrakta

```
{
    return next;
}
```

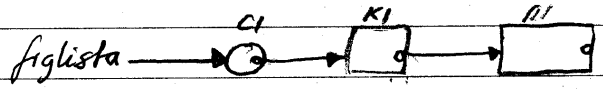
//Rek.cpp

```
REK::REK(char *namn, FIG *&next, float sid1, float sid2)
        : FIG(namn, next)
{
    this->sid1 = sid1;
    this->sid2 = sid2
}
```

//Cir.cpp

```
CIR::CIR(char *namn, FIG *&next, float radie)
        : FIG(namn, next)
{
    this->radie = radie;
}
```

```
//Kvad.cpp

KVAD::KVAD(char *namn, FIG *&next, featsid)
          : REK(namn, next, *id, sid
{
}

// spec Figsystem -- Forml.h

__gc class Forml : public Form
{
   private:
      FIG *figlista;
      REK *rp;
      KVAD *kp;
      CIR *cp;
   public:
    Forml()
    {
       figlista = NULL;
       rp = new REK("R1", figlista, 2.0, 3.0);
       kp = new KVAD("K1", figlista, 5.0);
       cp = new CIR("C1", figlista, 4.0);
    }
```



```
void AreaButtonClick(Object *sender, EventArgs *e)
{
    FIG *fp = fp->figlista;
    float sum = 0.0;
    char abuff[30];

    while (fp != NULL)
    {
        sum += fp->area();
        fp = fp->get_next();
    }
    sprintf(abuff, "Total area = %.2f", sum);
    Messagebox(abuff, "Total area");
}
};
```

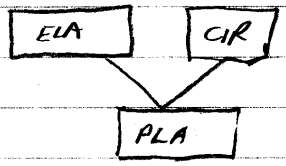
Skydd	Anv	Arv	Medlem
	Direktåtkomst		
class A			
{			
private:	Nej	Nej	Ja
protected:	Nej	Ja	Ja
public:	Ja	Ja	Ja
};			

Olika typer av arv:

- class B: privat A — Allt ärvt från A blir privat i B.
- class B: protected A — Allt ärvt från A
- class B: public A — Samma skydd i B som i A.

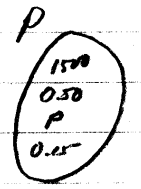
Multipelt arv

(Ex) En spisplatta är en klappad och en cirkel




```
//Ela.h

class ELA
{
    private:
        int me; //Märkeffekt
        float pos; // Knappposition 0.0-1.0
    public:
        ELA(int me, float pos);
        float effekt();
};

//Ela.cpp

ELA::ELA(int me, float pos)
{
    this->me = me;
    this->pos = pos;
}

float ELA::effekt()
{
    return me*pos;
}
```

```
// Pla.h

class PLA : public ELA, public CIR
{
    public:
        PLA( int me, float pa, char *nam, float radie);
        float area_effekt();
};

// Pla.cpp

PLA::PLA (int me, float pa, char *nam, float radie)
        : ELA(me, pa), CIR (nam, radie)
{
}

float PLA:: area_effekt()
{
    return  ELA:: effekt() / CIR:: area();
}

void main()
{
   PLA p (1500, 0.50, "P", 0.15);

   cout << p.area_effekt();
}
```

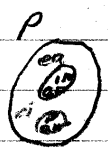


Problem! Namnkollisioner – använd alltid CIR::area() ...

Ärver flera ggr – virtual aw!

(Ex) Istället för multipelt arv kan man ofta använda aggregat (består av),

```
class PLA
{
   private:
      ELA ea;  // Inre objekt
      CIR ci;  // -"-
   public:
      PLA(int me, float pos, char *name, float radie);
      float area_effekt();
};

PLA::PLA(int me, float pos, char *name, float radie)
      : ea(me,pos), ci(name, radie)
{
}

float PLA::area_effekt()
{
   return ea.effekt()/ci.area();
}
```



Anrop!

```
PLA p(1500, 0.5, "P", 0.15);
cout<< p.area_effekt();
```

Hemuppgift: En Lärare är en person med lön. Skapa en klass LARAREsom ärver klassen PERSON och implementera konstruktorn i LARARE samt läs och skriv funktionen