8. Objektorientering

Skälet till att C++ är ett av de mest använda programspråken är att det är ***objektorienterat.*** Detta bygger vidare på begreppet *struct* (ursprungligen från språket C som inte är objektorienterat), som i objektorienteringen kallas *klass*. Främsta skillnaden mellan begreppen struktur och klass är att en klass även kan innehålla funktioner (metoder), samt att man kan härleda nya klasser utifrån en annan (bas-) klass.

Klass och objekt

Nedan skapar vi klassen *Person* innehållande *datamedlemmar* för förnamn, efternamn, ålder, längd och vikt:

```
//inkluderingsfilerna utelämnas här av utrymmesskäl
using namespace std;
//-------------------------------------------------------------------------
-------------------------------------------------------
class Person          //Definition av klassen Person börjar. Versal på Person
{
public:                 //Anger att datamedlemmarna nås utanför klassen
char fnamn[20], enamn[20];    //Deklaration av datamedlemmar (string ?)
int age, lengd, vikt;         //Deklaration av datamedlemmar
};
//----Definition av klassen slutar

int main()
{
Person p;                    //Variabeln p är ett objekt av klassen Person
(instans av klassen)
cout << "Ange förnamn: ";
cin >> p.fnamn;            //Klassens datamedlemmar anges med punktoperatorn
cout << "Ange efternamn: ";
cin >> p.enamn;
cout << "Ange ålder, längd och vikt: ";
cin >> p.age>> p.lengd >> p.vikt;
cout << p.fnamn << p.enamn << p.age << p.lengd << p.vikt <<endl;
return 0;
}
//-------------------------------------------
```

Klassnamnet brukar anges med stor bokstav för att lättare kunna särskiljas från variabler. Främsta skillnaden hittills mellan struktur och klass är annars att ordet ***public*** måste anges för att datamedlemmarna (variablerna) skall vara åtkomliga (synliga) utanför klassen. Anges inte *public* blir de automatiskt ***private*** och osynliga utanför klassen. Observera även att variabeldeklarationen av *p* av klasstypen *Person* gör att *p* blir ett ***objekt*** **tillhörande klassen** ***Person***.

Variabeldeklarationen innebär att p är en ***instans*** av klassen.

Medlemsfunktioner

Vi vill emellertid utöka vår klassdefinition med två *medlemsfunktioner* (metoder) för inläsning respektive utskrift. Vi vill dessutom inte att datamedlemmarna skall vara synliga utifrån; en användare har inget intresse av att veta hur dessa ser ut, utan vill ju bara kunna använda klassens funktioner. Klassvariablerna deklareras därför som *private*. För att komma åt klassvariablerna måste vi därför skapa ytterligare fem medlemsfunktioner, som deklareras som *public*:

```
using namespace std;          //Övriga inkluderingsfiler utelämnas här
//-------------------------------------------------------------------------------------------------------------
class Person                                    //Definition av klassen Person (med versal)
{
public:                                         //Anger att medlemsfunktionerna syns utanför klassen
void inmatning(char fn[],char en[],int ag,int le,int vi); //Deklaration av medlemsfunktioner
void skriv_namn();              //Deklaration av funktion för utskrift av för- och efternamn
int ge_alder() {return age;}     //Korta funktioner definieras i klassdefinitionen
int ge_langd() {return lengd;}   //Funktionen returnerar objektets längd
 int ge_vikt() {return vikt;}        //Funktionen returnerar objektets vikt
private:                        //Anger att datamedlemmarna ej syns utanför klassen
char fnamn[20], enamn[20];    //Deklaration av datamedlemmar (klassvariabler)
int age, lengd, vikt;
};     //Definition av klassen slutar
//-------------------------------------------------------------------------------------------------------------
void Person::inmatning(char fn[],char en[],int ag,int le,int vi)
{  //Definition av medlemsfunktion för inläsning till klassvariablerna (datamedlemmarna)
 age=ag;
lengd=le;
 vikt=vi;                          //Klassvariablerna är åtkomliga i klassfunktionerna
strcpy(fnamn,fn);               //stringcopy
strcpy(enamn,en);
}
//-------------------------------------------------------------------------------------------------------------
void Person::skriv_namn()    //Definition av medlemsfunktion för utskrift av för- och efternamn
{
                cout << setw(20) << fnamn << setw(20) << enamn << endl;
}
//-------------------------------------------------------------------------------------------------------------
```

```
int main()
{
		Person p;	//Instans av klassen Person, d v s p är ett objekt av klassen
		char f[20], e[20];
		int a,l,v;
		cout << "Ange förnamn: ";
		cin >> f;
		cout << "Ange efternamn: ";
		cin >> e;
		cout << "Ange ålder, längd och vikt: ";
		cin >> a >> l >> v;
		p.inmatning(f,e,a,l,v);	//Anrop av objektet p:s funktion inmatning
		p.skriv_namn();		//Skriver ut inmatade för- och efternamn
		cout << "Ålder:"<<p.ge_alder()<<"Längd:"<<p.ge_langd()<<"Vikt:"<<p.ge_vikt()
<<endl;
		return 0;
}
//-----------------------------------------------------------------------------------------------------------------
```

Klassen Person innehåller nu fem medlemsfunktioner och fem datamedlemmar (klassvariabler), d v s totalt tio *attribut* som beskriver klassens egenskaper. Observera att klassens namn anges med dubbla kolon (::) när klassfunktionerna (medlemsfunktionerna) definieras separat, samt att klassvariablerna är åtkomliga i klassfunktionerna (de tillhör ju klassen). Observera även att korta klassfunktioner (*ge_alder* t ex) kan definieras direkt i klassdefinitionen. Alternativet är att skriva en separat funktionsdefinition (som för *inmatning* och *skriv_namn* ovan):

```
int Person::ge_alder()		//Definition av medlemsfunktion som returnerar klassvariabln age
{
		return age;
}
```

Många menar att klassdefinitionen skall hållas ren från funktionsdefinitioner, d v s alla medlemsfunktioner (även korta) bör definieras separat utanför klassdefinitionen. Det ger en renare klassdefinition och ökar läsbarheten.

Konstruktor

Vanliga variabler kan som bekant initieras vid deklarationen:

```
int i=0;
```

Detta är emellertid inte tillåtet för klassvariabler, d v s klassens datamedlemmar. De kan dock initieras med en *konstruktor*, som liksom en funktion anropas vid deklarationen (instansen) av variabeln (objektet). Konstruktorn är en typlös (även utan *void*) medlemsfunktion med samma namn som klassen. Klassen kan dock innehålla flera konstruktorer, men med olika antal parametrar, d v s de är då överlagrade. Följande klassdefinition innehåller två konstruktorer:

```
class Person
{
public:
        Person (int ag,int le,int vi)       //Konstruktor med parametrar
        {
                age=ag;
                lengd=le;
                vikt=vi;
        }
        Person()                        //Konstruktor utan parametrar (default-konstruktor)
        {age=lengd=vikt=0;}
        void inmatning(char fn[],char en[],int ag,int le,int vi);
        void skriv_namn();
int ge_alder() {return age;}
        int ge_langd() {return lengd;}
        int ge_vikt() {return vikt;}
private:
        char fnamn[20], enamn[20];
        int age, lengd, vikt;
};
```

```
int main()
{
        Person p1, p2(45,181,73);  //Objekten p1 och p2 deklareras.
        cout<<"Ålder:"<<p1.ge_alder()<<"Längd:"<<p1.ge_langd()<<"Vikt:"<<p1.ge_vikt()
    <<endl;
        cout<<"Ålder:"<< p2.ge_alder()<<"Längd:"<<p2.ge_langd()<<"Vikt:"<<p2.ge_vikt()
   <<endl;
return 0;
}
```

Den första konstruktorn innehåller parametrar för ålder, längd och vikt. Variabeln (objektet) p2:s klassvariabler (datamedlemmar) tilldelas alltså åldern 47, längden 181 och vikten 73 vid anropet. Variabeln p1 deklareras utan parametrar, men eftersom vi har deklarerat en parameterlös konstruktor blir den *default-konstruktor* som anropas automatiskt vid deklarationen om inga parametrar anges. Variabeln p1 initieras alltså med värdet 0 på de tre datamedlemmarna.

Separat konstruktordefinition

Konstruktorns definition kan liksom för medlemsfunktioner göras separat utanför klassdefinitionen. Följande exempel visar hur detta ser ut för den parameterlösa konstruktorn *Person()*:

```
//-----------------------------------------------------------------------------------------
class Person
{
public:
        Person (int ag,int le,int vi)
        {
                age=ag;

                lengd=le;
                vikt=vi;
        }
        Person();                                   //Konstruktorns deklaration
        void inmatning(char fn[],char en[],int ag,int le,int vi);
        void skriv_namn();
        int ge_alder() {return age;}
        int ge_langd() {return lengd;}
        int ge_vikt() {return vikt;}
private:
        char fnamn[20], enamn[20];
        int age, lengd, vikt;
};
```

```
//-------------------------------------------------------------------------------------------
Person::Person()                                        //Konstruktorns definition utan void
{
        age=lengd=vikt=0;
}
//-------------------------------------------------------------------------------------------
```

Observera att konstruktorns definition saknar inledande *void*.

Initieringslista

En tredje rekommenderad variant (fungerar även för datamedlemmar som är konstanter) är att tilldela klassvariablerna via en *initieringslista*. Följande exempel illustrerar detta för konstruktorn med parametrar:

```
//-------------------------------------------------------------------------------------------
class Person
{
public:
        Person (int ag,int le,int vi);                  //Konstruktorns deklaration
        Person();
        void inmatning(char fn[],char en[],int ag,int le,int vi);
        void skriv_namn();
        int ge_alder() {return age;}
        int ge_langd() {return lengd;}
        int ge_vikt() {return vikt;}
private:
        char fnamn[20], enamn[20];
        int age, lengd, vikt;
};
//-------------------------------------------------------------------------------------------
Person::Person()
{
        age=lengd=vikt=0;
}
//-------------------------------------------------------------------------------------------
Person::Person (int a, int l, int v)            //Konstruktorns deklaration
:age(a), lengd(l), vikt(v) {}                   //Initieringslistan inleds med :, avslutas med tom {…}
//-------------------------------------------------------------------------------------------
```

Observera parameterlistans inledande kolon och de avslutande tomma klamrarna.

Destruktor

En *destruktor* är liksom en konstruktor en typlös medlemsfunktion. Men medan konstruktorn allokerar minnesutrymme när ett objekt skapas (variabeldeklareras), frigör destruktorn minne när objektet upphör att existera. I vårt program, där vi ännu arbetar med *statiska objekt* (d v s minnet allokeras vid kompileringen via variabeldeklarationer), anropas destrukorn automatisk när programmet avslutas. Snart kommer vi emellertid att arbeta med *dynamiska objekt* (d v s minnet allokeras dynamiskt under exekveringen) och då är det viktigt att kunna frigöra minne (d v s ta bort objekt) som inte längre används under exekveringen. Följande kod visar klassdefinitionen Person med en destruktor. Observera att klassdefinitionen får innehålla endast en destruktor:

```
class Person
{
public:
        Person (int ag,int le,int vi);
        Person();
        ~Person()                                   //Här är destruktorn deklarerad.
        {cout << "Objektet med åldern " << age << " tas bort." << endl;}
        void inmatning(char fn[],char en[],int ag,int le,int vi);
        void skriv_namn();
        int ge_alder() {return age;}
        int ge_langd() {return lengd;}
        int ge_vikt() {return vikt;}
private:
        char fnamn[20], enamn[20];
        int age, lengd, vikt;
};
```

Tecknet (~) framför destruktorns deklaration kallas *tilde*. Destruktorn ovan innehåller kod som talar om att att ett visst objekt tas bort. Vanligtvis innehåller destruktorn naturligtvis ingen kod.

Separat destruktordefinition

I följande klassdefinition definieras destruktorn utan kod separat utanför klassdefinitionen:

```
//-------------------------------------------------------------------------------------------
class Person
{
public:
        Person (int ag,int le,int vi);              //Konstruktorns deklaration
        Person();                                   //Konstruktorns deklaration
        ~Person();                                  //Destruktorns deklaration
        void inmatning(char fn[],char en[],int ag,int le,int vi);
```

```
        void skriv_namn();
        int ge_alder() {return age;}
        int ge_langd() {return lengd;}
        int ge_vikt() {return vikt;}
private:
        char fnamn[20], enamn[20];
        int age, lengd, vikt;
};
//-----------------------------------------------------------------------------------------
Person::~Person()                                   //Destruktorns definition
{
        cout << "Objektet med åldern " << age << " tas bort." << endl;
}
//-----------------------------------------------------------------------------------------
```

Övning 8.1:

Du skall skriva ett program som läser in en persons förnamn, efternamn, födelseår, längd (meter) och vikt. Programmet skall sedan skriva ut personens för- och efternamn, ålder , BMI-värde (se nedan) med en decimal och om personen är överviktig. Programmet skall ha följande klassdefinition:

```
class Person
{
public:
        void inmatning(char fn[],char en[],int f,double le,int vi);
        void skriv_namn();
        double bmi();
        int alder(int artal);
private:
        char fnamn[20], enamn[20];
        int vikt, bar;
        double lengd;
};
```

Klassen har följande medlemsfunktioner:

- *inmatning(…);* läser in personens förnamn, efternamn, födelseår, längd och vikt.
- *skriv_namn();* skriver ut för- och efternamn på en rad utan radbrytning.
- *bmi();* beräknar personens BMI (body mass index), d v s vikten (kg) dividerad med längden (meter) i kvadrat.
- *alder(ar);* beräknar personens ålder utifrån inmatat innevarande årtal.

Klassen har följande datamedlemmar (klassvariabler):

- *fnamn* (förnamn), *enamn* (efternamn), *vikt* (vikt), *bar* (födelseår) och *lengd* (längd).

Programmet innehåller dessutom en funktion *string overvikt(double bmi);* som skriver ut om personen är underviktig (bmi<20), överviktig (bmi>20) eller normalviktig.

Övning 8.2:

Ändra klassdefinitionen i föregående uppgift så att den får följande utseende:

```
class Person
{
public:
        double bmi();
        int alder(int artal);
        char fnamn[20], enamn[20];
        int vikt, bar;
        double lengd;
};
```

Observera att klassens samtliga attribut är synbara (public) utifrån. Programmet skall nu kunna lagra flera personer i ett fält (array) eller vector. Modifiera och använd funktionerna för inläsning, utskrift, lagring på och läsning från binär eller textfil jmfr med tidigare uppgifter du gjort t.ex *CD-Lista – Stad/ Land – m.fl*.

Inga globala variabler skall användas. **Lös uppgiften på följande sätt:**

- I main-funktionen deklareras variabeln *antal*, som håller reda på antalet personer, samt fältet *folk* för lagring av maximalt 10 personer:
  Person folk[10];

Följande funktioner skall kunna anropas frånn menyn i main-funktionen:

- *void fyll(Person f[], int& antal);* läser in och lagrar personens förnamn, efternamn, födelseår, längd och vikt i fältet.
- *void skriv(Person f[], int antal);* skriver ut personernas förnamn, efternamn, ålder BMI och status (se nedan) i en snygg tabell.
- *void medelvarde(Person f[],int antal);* skriver ut medelvärdet för samtligas vikt, ålder och BMI.
- *void spara(Person f[],int antal);* lagrar personerna på en binärfil.
- *void hamta(Person f[],int& antal);* läser in personerna från en binärfil.