

Programmering i C++

EDAF30

De sista pusselbitarna

Innehåll

- Namnrymder
- struct
- union
- Bit-operatorer
- Bit-fält

Namnrymder

Avgränsning av deklARATIONERS giltighetsområde kan göras mha *namnrymder* (namespace).

Exempel

```
// Filen abib.h
namespace Abib {
    void initiera();
    double resultat(double x);
    const int max_antal = 20;
    void skriv_ut(double x);
}

// Filen bbib.h
namespace Bbib {
    void initiera();
    double resultat(double x);
    const int max_antal = 100;
    const int minsta_tal = 2;
}
```

Definitioner av funktioner

Variant 1

```
// Filen abib.cpp
#include "abib.h"

void Abib::initiera() {
    // ...
}

double Abib::resultat(double x) {
    // ...
}

void Abib::skriv_ut(double x) {
    // ...
}
```

Enklare sätt att definiera funktioner

Variant 2

```
// Filen bbib.cpp
#include "bbib.h"

namespace Bbib {
    void initiera() {
        // ...
    }

    double resultat(double x) {
        // ...
    }
}

// Går bra att utöka namnrymder efter hand!
```

Namnlösa namnrymder – Synliga bara inom filen

Ny version

```
// Filen bbib.cpp (ny version)
#include "bbib.h"

namespace {
    void intern1() {//...}
    void intern2() {//...}
}

namespace Bbib {
    void initiera() {
        // ... intern1(); ...
    }
    double resultat(double x) {
        // ... intern2(); ...
    }
}
```

Användning av namnrymder

```
//  
#include "abib.h"  
#include "bbib.h"  
//...  
skriv_ut(1.23); // Fel, ty ej entydig  
Abib::skriv_ut(1.23); // Ok  
//...  
using Bbib::minsta_tal; // Användningsdekl.  
cout << minsta_tal << endl;  
//...  
using namespace Abib;  
skriv_ut(1.23); // Ok nu  
using namespace Bbib;  
initiera(); // Fel ty ej entydig  
Bbib::initiera(); // Ok nu
```

- Arv från C – struct: Sammansatt datastruktur där komponenterna kan vara av olika typ
- I C++ är struct samma som class förutom när det gäller underförståddhet (default) av synlighet:
 - För class gäller `private` om inget anges
 - För struct gäller `public` om inget anges
- Ovanstående gäller även för synlighet vid arv

Exempel

```
struct S1 {  
  // public: /* Underförstått dvs behövs ej anges */  
  S1(int n=0) : i(n) {}  
  void skriv() {cout << i << endl;}  
  int i;  
};  
  
struct S2 : S1 { // eg. S2 : public S1  
  // public: /* Underförstått */  
  S2(int n, double x) : S1(n), d(x) {}  
  double kvad() { return d*d;}  
private:  
  double d;  
};
```

I en "vanlig" struct (class) allokeras utrymme motsvarande *summan* av de ingående delarna

```
struct DataS {  
    int nr;  
    double v;  
    char txt[6];  
};
```

I en union allokeras utrymme motsvarande *maxstorleken* av de ingående delarna

```
union DataU {  
    int nr;  
    double v;  
    char txt[6];  
};
```

Med definitionen av DataU enligt ovan kan följande utföras

```
//  
DataU a;  
a.nr = 57;  
cout << a.nr << endl;  
a.v = 12.345;  
cout << a.v << endl;  
strcpy(a.txt, "Tjo");  
cout << a.txt << endl;
```

En union kan kapslas in i en klass för att minska risken för fel

```
enum Slag {nummer, varde, text, odef};

struct Kod {
    Slag s; // diskriminant
    DataU d; // union (kallas också variant)
};

// Test av diskriminanten innan användning
// (m är av typen Kod)
m.s = text; strcpy(m.d.txt, "Kod");
if (m.s == text)
    cout << "Text: " << m.d.txt << endl;
```

Ett mer direkt alternativ är följande:

```
struct Kod {  
    Slag s;  
    union {  
        int nr;  
        double v;  
        char txt[6];  
    };  
};
```

```
// Även här är m av typen Kod  
// Observera att en "nivå" eliminerats  
m.s = text; strcpy(m.txt, "Kod2");  
if (m.s == text)  
    cout << "Text: " << m.txt << endl;
```

Operationer på låg nivå: Bit-operatorer

Alla variabler antas vara av typen `unsigned short int` vilket innebär 16 bitars positiva heltal

```
a = 77;           // a = 0000 0000 0100 1101
b = 22;           // b = 0000 0000 0001 0110
c = ~a;           // c = 1111 1111 1011 0010
d = a & b;        // d = 0000 0000 0000 0100
e = a | b;        // e = 0000 0000 0101 1111
f = a ^ b;        // f = 0000 0000 0101 1011
g = a << 3;       // g = 0000 0010 0110 1000
h = c >> 5;       // h = 0000 0111 1111 1101
i = h & 0x0f0f;   // i = 0000 0111 0000 1101
j = h | 0x0f0f;   // j = 0000 1111 1111 1111
```

Snålt om minnesutrymme? Använd bit-fält!

Ange explicit antalet bitar med *var* : *antalbitar*

```
struct Bil { // post i ett bilregister
    char reg_nr[6];
    unsigned int arsmod : 7;
    unsigned int skatt_betalld : 1;
    unsigned int besiktigad : 1;
    unsigned int avstalld : 1;
};
```

Åtkomst av bit-fält

```
Bil b;  
strncpy(b.reg_nr, "ABC123", 6);  
b.arsmod = 97;  
b.skatt_betalld = true;  
b.besiktigad = true;  
b.avstalld = false;  
cout << "Årsmodell: " << b.arsmod << endl;  
if (b.skatt_bet && b.besiktigad)  
    cout << "Bilen är OK";
```


Register med 16 bitar:

```
struct Register {  
    unsigned int enable :1;  
                  :4;  
    unsigned int ready  :1;  
                  :2;  
    unsigned int data   :8;  
};
```

