Templates (mallar)

- Typparametrisering
- Ersätter macros
- Ersätter ofta arv
  - Snabbare men mer minneskrävande
- För Funktioner
- För Klasser
- Nya algoritmbiblioteket

För funktioner

```
i = max( j++, k);
```

```
#define max(x,y) (x>y?x:y)

// ersätts av (T är godtycklig datatyp):

template<class T> T max(T& x,T&y)
{
  return (x>y?x:y);
}
 int i,j=5,k=3;
 double a,b=23.78,c=5.6;
 i=max(j,k);
 a=max(b,c);
 cout << i << ’’ << a; // 5 23.78
```


För klasser

```
template<class T> class V
{ public:
     V(int sz) { vk=new T[size=sz];}
  private:
     T* vk;
     int size;
};

V<int> vi(23);
V<double> vd(12);
```


Parametrisera storlekar

```
class Wheel
{
  int sz;
};

template<int wheels> class Vehicle
{
   Wheel w[wheels];// undviker
                   // pekarhantering
};

 Vehicle<3> trehjuling;
```


Mer avancerad funktion

```
template<class T> T medel(T a[],unsigned storlek)
{
 T res=a[0];
 for (int i=1;i<storlek;i++)
         res += a[i];
 return res/storlek;
 }
```


Vi måste definiera operatorerna

```
class Bil
{
public:
  Bil(int m=200):mil(m){}
  Bil operator+= (Bil& b2) {  mil += b2.mil; return *this; }
  Bil operator / (int n) { mil /= n; return *this;}
  friend ostream& operator<<(ostream& os, const Bil& b);

private:
  int mil;
};
```

Användning

```
ostream& operator<<(ostream& os, const Bil& b)
{
 os << "Miltal " << b.mil;
 return os;
}
void main()
{
 Bil garage[10]={1000,0};
 Bil mb= medel(garage,sizeof(garage)/sizeof(Bil));
 cout << garage[0] << ’’<< mb;
 //Miltal 1000 Miltal 260
}
```


Men vad heter konstruktorn?

```
template<class T> class V
{ public:
  V(int sz);
  V& operator=(V& v);

private:
  T* vk;
  int size;
};

template<class T> V<T>::V(int sz)
{
  vk = new T[size=sz];
}
```


För att inte tala om tilldelningsoperatorn...

```
template<class T>
V<T>& V<T>::operator=(V<T>& v)
{
  delete [] vk;
  vk = new T[size=v.size];
  for (int i=0;i<size;i++)
  {
      vk[i] = v.vk[i];
  }
  return *this;
}
V<int> v1(12),v2(23);
v2=v1; // tilldelning
```


Några egenskaper hos templates

- ”Något” kryptiskt
- Ev problem med länkningen (dubbla def)
- Kraftfullt
- Behöver **inte** ha källkoden
- `V<int>` fungerar som klassnamn
- Kan använda arv i mallarna
- Kan ha flera olika klasser&parametrar

Vid tveksamheter

- Ofta felmeddelanden

```
int i,j,k; char c,d,e;
i=max(j,k); //ok
c=max(d,e); // ok
i=max(c,j);// Fel; max(int,int)/max(char,char)?
//definiera
int max(int,int);
i=max(c,j); // går bra nu max(int,int)
```


Anropsregler

1. Se om exakt parametermatchande funktion finns; anropa den
2. Se om en template finns som kan generera en funktion med exakt match
3. Försök hitta en (överlagrad) funktion

2

En specialiserad funktion

```
cout << max("stropp","kalle");
// skriver alltid ut den 2:a
//parametern..högst adress!

#include <string.h>
template<class T> T max(T& x,T&y)
{ // den allmänna funktionen
  return (x>y?x:y);

}
char* max ( char *s1,  char *s2)
{ // specialiserad funktion
  return (strcmp(s1,s2)>0? s1 :s2);
}
```


Flera parametrar är möjliga

```
template<class Key,class Value>
class Map {
 Value& operator[](const Key&); // m fl
};
void main()
{
  Map<String,int> count; // ”associationsklass”
  String word;
  while(cin >> word)
    count[word]++; // ord som index!
  for (Mapiter<String,int> p=count.first();p; p++)
    cout << p.value() << ’\t’ << p.key() << ’\n’;
}
```

Iterator

Defaultvärden är möjliga

```
template<class T=double, int storlek=10>
class V
{
 T vk[storlek];
};

V<> vekt; // En vektor med 10 doubles
```