```
#include <cstdlib> // för slumptal

template<class T>
T mix(T t1, T t2) { return (rand() & 1) ? t1 : t2;}

std::cout
  << mix(1, 2)
  << mix(1.0, 2.0)
  << mix(1, 2.0)          // fungerar ej!
  << mix<double>(1, 2.0)
  << std::endl;
```

```
template<class T>
class SillyPair {
  T a;
  T b;
public:
  SillyPair(T t1, T t2) : a(t1), b(t2) {}
  void print_a();
};

template<class T>
void SillyPair<T>::print_a() {
  std::cout << a;
}
```

```
template<class AnimalType>
class Pet : public AnimalType {
public:
  std::string owner;
};

Pet<Dog> the_dog;
Pet<Fish> the_fish;

template<class T>
class B {
    typename T::Type *p; // deklarera pekare
    T::Value * f; // multiplikation
};
```

```
template<int N>
struct Fact {
  static const int value = N*Fact<N-1>::value;
}

template<>
struct Fact<0> {
  static const int value = 1;
}

std::cout << Fact<5>::value << std::endl;
```

```
template <int N, int Low = 1, int High = N> struct Root;

template <int N, int Mid>
struct Root<N,Mid,Mid> {
  static const int root = Mid;
};

template <int N, int Low, int High>
struct Root {
  static const int mean = (Low + High)/2;
  static const bool down = (mean * mean >= N);
  static const int root = Root<N,
                               (down ? Low : mean+1),
                               (down ? mean : High) >::root;
};

std::cout << Root<36>::root << std::endl;
```

```cpp
template<class T>
T max(T t1, T t2) {
  return (t1 > t2 ? t1 : t2);
}

template<>
char* max<char*>(char* c1, char* c2) {
  return (strcmp(c1,c2) > 0 ? c1 : c2);
}
```

```
#include <vector>

std::vector<int> fives(10, 5); // 10 element med värdet 5
std::vector<int> zeros(10); // 10 element med värdet int() (dvs 0)
std::vector<std::string> v; // 0 element
v.push_back("hello");
v.push_back("world");
v[0] += v[1]; // sammanslagning av strängar
std::copy(v.begin(),
          v.end(),
          std::ostream_iterator<std::string>(std::cout,"\n"));
v.size();     // antal element i v
v.capacity(); // minne allokerat för v (mätt i element)
v.resize(10); // tar bort element/lägger till nya default-element
v.reserve(10);// ordna plats för nya element i v
```

```cpp
#include <map>

std::map<std::string, int> m;
m["one"] = 1;
m["two"] = 2;
m["three"] = m["one"] + m["two"];

std::vector<std::pair<std::string, int> > v2; // notera > >
std::copy(m.begin(), m.end(), std::back_inserter(v2));

std::cout << m["four"]; // sätter m["four"] till int(), dvs 0
if(m.find("five") == m.end())
  std::cout << "five finns inte" << std::endl;
```

```
std::list<int> v;
for(int i = 0; i < 10; i++)
  v.push_back(i);

for(std::list<int>::iterator i = v.begin();
    i != v.end();                                        // notera !=
    ++i)                                                 // ++i bättre än i++
  std::cout << *i << std::endl;
```

```
#include <algorithm>

int triple(int i) { return 3*i; }

std::vector<int> v, v2;
for(int i = 0; i < 10; i++)
  v.push_back(i);
std::vector<int>::iterator it = std::find(v.begin(), v.end(), 4);
std::reverse(it, v.end());
std::sort(v.begin(), it);
int mx = *std::max_element(v.begin(), v.end());
std::transform(v.begin(), v.end(),
               std::back_inserter(v2),
               triple);
```

```
#include <memory>

std::auto_ptr<BigObj> a(new BigObj());
std::auto_ptr<BigObj> b;

b.get(); // NULL
b = a;
a.get(); // NULL
a = std::auto_ptr<BigObj>(new BigObj());
a->doBigObjStuff();
std::cout << *a;
*a = *b; // ändra a:s objekts värde
a = b;   // ge a b:s objekt
```

```
#include <bitset>
std::bitset<13> flags(17);
std::bitset<13> more_flags(std::string("0100"));

more_flags[0] = 1;    // more_flags == "0000000000101"
more_flags.set(10);   // more_flags == "0010000000101"
more_flags.reset(2);  // more_flags == "0010000000001"
more_flags[12].flip();// more_flags == "1010000000001"

flags &= more_flags; // must have same size
flags.flip(); // flip all bits
flags.size(); // template size arg
flags.count();// # of bits set
flags.any();  // any bit set ?
flags.none(); // no bit set ?

std::cout << (flags << 1) << " is 2 * " << flags.to_ulong();
```