```
/*
Template is a mechanism that allows a programmer to use types as parameter
for a class or a function. The compiler generates a specific class or func
when we later provide specific type as arguments.
The C++ notation for introducing a type parameter T is template<class T> p
"for all types T".
*/

/*
Generic Programming: Writing code that works with a variety of types
presented as arguments, as long as those argument types meet specific
syntactic and semantic requirements. Parametric polimorphism.

Use templates where performance is essential (e.g., numerics and hard real
time).
Use templates where flexiibility in combining information from several typ
is essential.
*/

// Template functions
template <class T>
T soma(T a, T b)
{
    T result;
    result = a+b;
    return result;
}

template <class T, class U>
bool are_equal(T a, U b)
{
    return (a==b);
}

// Template class
template <class T>
class mypair {
    T a, b;
  public:

    mypair (T first, T second)
      {a=first; b=second;}

    T getmax ();
};

template <class T>
T mypair<T>::getmax()
```

```
49 {
50   T retval;
51   retval = a>b? a : b;
52   return retval;
53 }
54
55 // Template specialization
56
57
58 // class template:
59 template <class T>
60 class mycontainer {
61     T element;
62   public:
63     mycontainer (T arg) {element=arg;}
64     T increase () {return ++element;}
65 };
66
67 // class template specialization:
68 template <>
69 class mycontainer <char> {
70     char element;
71   public:
72     mycontainer (char arg) {element=arg;}
73     char uppercase ()
74     {
75       if ((element>='a')&&(element<='z'))
76       element+='A'-'a';
77       return element;
78     }
79 };
80
81 template<class T, int N> struct array
82 {
83     T elem[N];
84
85     T& operator[] (int n) { return elem[n]; }
86     const T& operator[](int n) const { return elem[n]; }
87
88     int size() const { return N; }
89 }
90
91 int main()
92 {
93     int x = sum<int>(10,20);
94     std::cout << x;
95
96     double xx = sum<double>(10.2,11.1);
```

```
97      std::cout << xx;
98
99      are_equal<int,double>(10,10.0);
100
101     mypair<int> myints (115, 36);
102
103     mypair<double> myfloats (3.0, 2.18);
104
105     mypair <int> myobject (100, 75);
106
107     std::cout << myobject.getmax();
108
109     array<int,256> gb;
110 }
111
```