Templates und generische Programme

Aufgabe 7.1 Führen Sie folgendes Codefragment in Ihrem Kopf aus. Welche Definition der Funktion `f` wird auf den einzelnen Zeilen jeweils aufgerufen? Für diese Aufgabe ist es wichtig, zu wissen, dass ein Ganzzahl-Literal wie beispielsweise `1` den Typ `int` hat (entsprechend hat `5.0` den Typ `double`).

```
template <typename T>
void f(T x)    { std::cout << x << " Generic"     << std::endl; }

template <>
void f(int x) { std::cout << x << " Specialized" << std::endl; }

void f(int x) { std::cout << x << " Overloaded"  << std::endl; }

f(1);
f<double>(2);
f<int>(3);
f<>(4);
f(5.0);
f<double>(6.0);
f<int>(7.0);
f<>(8.0);
```

Aufgabe 7.2 Im Gegensatz zu Funktionen-Templates erlauben Klassen-Templates (oder auch Struct-Templates) eine partielle Spezialisierung, bei der nur eine Teilmenge der Template-Parameter vorgegeben werden kann. Welche Ausgaben erwarten Sie auf den untersten fünf Programmzeilen?

```
template <typename T1, typename T2, typename T3 = T2>
struct S {
  void f() { std::cout << "Generic"     << std::endl; }
};
template <typename T2>
struct S<int, T2, char> {
  void f() { std::cout << "Specialized" << std::endl; }
};

S<char, int,  int>  s1; s1.f();
S<int,  char, int>  s2; s2.f();
S<int,  int,  char> s3; s3.f();
S<char, int>        s4; s4.f();
S<int,  char>       s5; s5.f();
```

Aufgabe 7.3 Betrachten Sie das folgende Codefragment. Überlegen Sie sich zuerst, was höchst wahrscheinlich die Absicht des Programmierers war, welche Ausgabe das Programm idealerweise produzieren sollte, und was stattdessen tatsächlich passiert.

```
template <typename Number, typename ... Numbers>
double sum(Number first, Numbers ... rest) {
  return static_cast<double>(first) + sum(rest ...);
}

int   a = 1;  float    b = 2.5F; double      c = 0.05;
short d = 42; unsigned e = 30U;  long double f = 2.22L;
std::cout << sum(a, b, c, d, e, f) << std::endl;
```

Passen Sie nun den Code an, so dass er sich wie vom Programmierer gewünscht verhält.