Podrazumevani argumenti

- Vrlo često je potrebno da funkcija ima nekoliko varijanti, odnosno da se može pozvati sa nekim argumentom ili bez njega, pri čemu izostvaljeni argument treba da uzme neku *podrazumevanu vrednost* (*default argument*); na primer:

Podrazumevani argument

```
complex::complex(double re=0.0, double im=0.0);
```

Podrazumevani argument, sa značenjem da se podrazumeva početak ako se izostavi

```
template <typename T>
list& list::insert (T t, int at=0);
```

- Ako se neki stvarni argument u pozivu ovakve funkcije izostavi, taj argument dobiće podrazumevanu vrednost navedenu u deklaraciji funkcije:

Inicijalizacija: *c1(1.,1.), c2(1.,0.0), c3(0.0,0.0)*

```
complex c1(1.,1.), c2(1.), c3;
```

insertAt(c1,0).insertAt(c2,1).insertAt(c3,0)

```
list<complex> lst;
lst.insertAt(c1).insertAt(c2,1).insertAt(c3);
```

Podrazumevani argumenti

- Podrazumevani argumenti nisu deo tipa funkcije. Ako se deklaracije funkcije ponavljaju, ne smeju ponovo navoditi podrazumevani argument za isti parametar, čak i ako je identičan. Na mestu poziva funkcije, podrazumevani argumenti predstavljaju uniju svih do tada deklarisanih podrazumevanih argumenata, s tim da ne sme postojati parametar koji nema podrazumevani argument iza parametra koji ima podrazumevani argument:

```
void f(int p1, int p2 = 2,int p3);
void f(int p1 = 1, int p2, int p3);
f(3);
void f(int p1, int p2,int p3=3);
f(0);
```

- U principu, ovaj koncept predstavlja notacionu pogodnost: da ga nema, bilo bi potrebno pisati više varijanti funkcija sa različitim parametrima; na primer, umesto:

```
double log (double x, double base=10.0);
```

moralo bi da se piše:

```
double log (double x, double base);
double log (double x) { return log(x,10.0); }
```