Objektno orijentirano programiranje C++

Predavanje 01 - ponavljanje i proširenja

Matej Mihelčić

Prirodoslovno-matematički fakultet
Matematički odsjek

04. ožujka 2024.

const

```
1 const double PI = 3.14;//PI je konstantna
2 //varijabla, vrijednost joj se ne moze mijenjati
3 const double e; //Greska, vrijednost konstantne
4 //varijable mora biti pridijeljena pri
5 //inicijalizaciji
6 PI = 3;//Greska, PI je konstanta koja
7 //sadrzi vrijednost!
```

Prevodioc uglavnom mijenja pojavljivanja konstantnih varijabli u kodu odgovarajućim pridruženim konstantama.

Konstantne varijable su vidljive samo lokalno, unutar izvorne datoteke koja sadrži definiciju i inicijalizator. Da bi konstantna varijabla bila vidljiva u više izvornih datoteka, trebamo koristiti ključnu riječ `extern` (`extern const double PI = 3.14;` unutar datoteke s inicijalizatorom, `extern const double PI;` u svim drugim datotekama gdje želimo koristiti PI).

const

```
1  const int broj = 50;
2  const int ci = 4;
3  const int &r1 = ci; //OK, referenca i objekt koji
4  //reprezentira su const
5  r1 = 60; //greska jer je r1 referenca na const
6  int &r2 = broj; //greska, ne const referenca
7  //na const objekt
8  int k = 70;
9  const int &r3 = k; //OK, ne mozemo mijenjati k
10 //preko r3
11 const int &r4 = 3; //OK, r4 je referenca na const
12 const int &r5 = r1*2; //OK, r5 je
13 //referenca na const
14 int &r6 = r1*2; //greska, r6
15 //nije const referenca
```

const

```
1 double d = 1.23;
2 const int &r = d; //OK, prevodioc stvori
3 //privremenu varijablu tipa int, int tmp = d;
4 //i asocira ju s referencom r,
5 //const int &r = tmp;
```

const

```
1  const double pi = 3.14;
2  double *p =  &pi; // greska, p je
3  //obican pokazivac
4  const double *p1 = &pi; //OK, p1 je pokazivac
5  //na const
6  *p1 = 10; //greska, ne mozemo pridijeliti
7  //vrijednost preko pokazivaca p1
8  double k = 12.4;
9  p1 = &k; //OK, ne mozemo mijenjati
10 //vrijednost preko p1
```

const

```
1  int broj = 10;
2  int *const p = &broj; //p je const pokazivac
3  //na broj
4  *p = 35; //OK, p nije pokazivac
5  //na const
6  int broj1 = 20;
7  p = &broj1; //greska, p je const
8  //pokazivac na broj
9  int const broj2 = 45;
10 const int *const p1 = &broj2; //p1 je
11 //const pokazivac na const int
```

constexpr

Da bi osigurali da const varijablu inicijaliziramo konstantnim izrazom, možemo koristiti constexpr. U tom slučaju prevodioc provjerava izraz koji se pridjeljuje varijabli.

```
1 constexpr int a = 10; //OK, 10 je int konstanta
2 constexpr int b = a+1; //OK, a+1 je konstantan
3 //izraz
4 constexpr int sz = size(); //OK, samo ako je size
5 //constexpr funkcija
6 int c = 45;
7 constexpr int d = c;// greska, c nije
8 //konstantna varijabla
```

Konverzije

Uz standardnu implicitnu i eksplicitnu konverziju (kao u C-u), C++ uvodi 4 nove vrste eksplicitne konverzije.

```
1  int x = 5;
2  double y;
3  y = x; //implicitna konverzija
4  //int-a u double
5  double z = 12.34;
6  int g = (int) z; //eksplicitna konverzija
7  //double-a u int
8  static_cast<T> // eksplicitna konverzija
9  //za konverziju obrnutog smjera od implicitne
10 void *pv = &x;
11 int *pi;
12 pi = pv; //greska, nije dozvoljena implicitna
13 //konverzija iz uzeg tipa int* u siri void*
14 pi = static_cast<int*>(pv);
```

Konverzije

```
1  const_cast<T>(izraz); //uklanja konstantnost
2  //reference ili pokazivaca na
3  //nekonstantan objekt
4  dynamic_cast<T>(izraz); //konverzija pokazivaca
5  //ili reference na baznu klasu u pokazivac
6  //ili referencu na izvedenu klasu
7  reinterpret_cast<T>(izraz); //vrsi konverzije
8  //izmedu nepovezanih tipova, npr. int* u int.
9  //Vrsi se reinterpretacija bitova - ovisno o
10 sustavu. Nije prenosivo s racunala na racunalo
```

Reference i funkcije

U C++-u je moguće definirati funkciju koja kao formalni argument prima referencu na polje. Međutim, za razliku od korištenja pokazivača za prijenos polja funkciji, kada se koristi referenca, dimenzije polja moraju biti obavezno navedene (polja različitih dimenzija su različitog tipa).

```
1  int max(int (&polje)[4])
2  {
3      int m = polje[0];
4      for(int i=1; i< 4; ++i)
5          if(polje[i]>m) m = polje[i];
6      return m;
7  }
```

Reference i funkcije

Funkcije mogu vraćati referencu. Dolazi do uštede memorije i vremena jer se ne kopira cijeli objekt već samo referenca. Dodatna prednost je što se tako vraćene reference smiju koristiti kao lijevi operand operacije pridruživanja. Funkcije koje vraćaju referencu **ne smiju vraćati referencu na lokalne varijable**

```
1 int& dohvati(int *polje, int n, int k)
2 {
3     if(k < n)
4         return polje[k];
5     else exit(1);
6 }
```

Reference i funkcije

```
1 int x[10] = {1,2,3,4,5,6,7,8,9,10};
2 int n = 10;
3
4 dohvati(x,10,4) = 12;
5 std::cout<<x[4];//ispis je: 12
```

lvalue i rvalue

Svi objekti s lokacijom u memoriji sadrže lijevu vrijednost (`lvalue`). Npr. varijable, referenca na element polja, dereferencirani pokazivač, poziv funkcije koji vraća referencu. Desnu vrijednost (`rvalue`) imaju izrazi koji nemaju lijevu vrijednost. Npr. konstante, povratne vrijednosti većine operatora, pozivi funkcija koji vraćaju vrijednost (a ne referencu).

```
1 int x; x = 3; //OK!
2 double y[5];
3 y[2] = 2.2; //OK!
4 "ab" = "cd"; // greska! "ab" je rvalue
```

Referenca na rvalue

Uvedena u standardu C++11. Označava se sa &&. Može se inicijalizirati samo s rvalue. Referenca na lijevu vrijednost se ne može inicijalizirati s rvalue.

```
1 int a = 4;
2 int &r1 = a*6; // greska: a*6 je rvalue
3 int &r2 = 0;  // greska: konstanta je rvalue
4 int &&r3 = 3; // OK!
5 int &&r4 = a*6; // OK!
6 int &&r5 = r4; //greska, varijabla je lvalue
7 int const &r6 = 3 //OK zbog stvaranja
8 //privremenog objekta!
```

Funkcijom std::move(lvalue) možemo napraviti eksplicitnu konverziju lijeve vrijednosti u desnu vrijednost.

```
1 int &&r7 = std::move(r4); //OK! r4 vise ne
2 //koristimo nakon ovog poziva.
```

Preopterećenje funkcije na osnovu konstantnosti

Kod prepoterećenja funkcije na osnovu konstantnosti jedna funkcija kao argument prima konstantnu referencu a druga ne konstantnu. Prevodioc uvijek može odrediti koju funkciju pozvati.

```
1 void f(int & x) {
2   std::cout << "f(int␣&)"<<std::endl;}
3 void f(const int & x) {
4   std::cout << "f(const␣int␣&)"<<std::endl;}
5
6 int x = 5;
7 f(6); // poziva  f(const int & )
8 f(x); // poziva  f(int & )
```

Preopterećenje funkcije na osnovu konstantnosti

Kod funkcija:

```
1 void f(int x) {
2     std::cout << "f(int)"<<std::endl;}
3 void f(const int x) {
4     std::cout << "f(const␣int)"<<std::endl;}
```

prevodioc ne može odrediti koju varijantu funkcije pozvati. Dolazi do greške pri prevođenju.

Preopterećenje funkcije na osnovu lijeve/desne reference

Kod funkcija:

```
1  void f(const int & x) {
2        std::cout << "f(const␣int␣&␣)\n"; }
3  void f(int & x) {
4       std::cout << "f(int␣&␣)\n"; }
5  void f(int && x) {
6      std::cout << "f(int␣&&␣)\n"; }
7
8  int x = 4;
9  const int j = 1;
10 f(3); // poziva  f(int && )
11 //izbjegava se konverzija
12 //rvalue u const int
13 f(x); //poziva f(int &)
14 f(x*x); // poziva  f(int && )
15 f(j);  // poziva  f(const int & )
```

Generičke funkcije - parametri predloška

- Tipovi: koristimo `typename ime_tipa`
- Vrijednosti:
  - Konstantni cjelobrojni izrazi: npr. `<int N>`
  - Pokazivaci na objekte/funkcije: npr. `<int *polje>`
  - Lijeve reference na objekte/funkcije: npr. `<char &slovo>`

```
1 template <std::size_t v>
2 void f(std::array<double, v> x){
3   std::cout << "zadnji = " << x[v-1]<< std::endl;
4 }
5
6 std::array<double, 3> x{1,2,3};
7 f(x);//automatska dedukcija tipa (v = 3)
```

Zbog kompatibilnosti sa starijim verzijama možemo koristiti i `template <class ime_tipa>`.

Generičke funkcije - parametri predloška

```
1  template <const char *tekst>
2  void f(int x){
3    std::cout << tekst << "␣:␣" << x <<  std::endl;
4  }
5
6  const char  nesto[] = "Neki␣tekst...";
7
8  int main(void){
9    f<nesto>(4); // dedukcija nije moguca
10   return 0;
```

Generičke funkcije - parametri predloška

```
1  template <char &znak>
2  void f(int x){
3    std::cout << znak << "␣:␣" << x <<  std::endl;
4  }
5
6  char d = 'c';
7
8  int main(void){
9    f<d>(4);//dedukcija nije moguca
10   return 0;
```

Dedukcija parametara

Prevodioc ispituje tip argumenata s kojim je pozvana generička funkcija kako bi odredio parametar predloška. Konverzije koje prevodioc radi u ovom koraku su ograničene.

```
1 template <typename T>
2 T Abs(T a)
3 {
4     return a > 0 ? a : -a;
5 }
6
7 Abs(10); //OK, T = int
8 Abs(5.4); //OK, T = double
```

Dedukcija parametara

```
1 template <typename T>
2 T Sum(T a, T b)
3 {
4     return a+b;
5 }
6
7 Sum(20,21.23); //greska, T se ne
8 //moze zakljuciti
```