Operátorok, precedencia szabályok

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2023 szeptember 12

Kifejezések kiértékelési sorrendje

Egy kifejezésben szerepelhet:

- függvényhívás
- zárójelek
- előjelek (`+`, `-`)
- aritmetikai operátorok (`+`, `-`, `*`, `/`, `%`)
- logikai operátorok (`==`, `!=`, `<`, `>`, `<=`, `>=`)
- beolvasás (`»`), kiírása (`«`), névtér feloldás (`::`) operátora

és még további operátorok, amikről később tanulunk.

A végrehajtás sorrendjét az ún. **precedencia** szabja meg:

1. függvényparaméterek kiértékelése
2. függvények meghívása
3. zárójelek
4. előjelek
5. szorzás és osztás műveletek
6. összeadás és kivonás műveletek

Néhány fontosabb operátor

Aritmetikai operátorok

- `+`, `-`, `*`, `/` szokásos jelentés
- `%`: maradék képzés (modulo), `int` típusú operandusokat fogad el: `intv1 % intv2`

Összehasonlító operátorok

- `<`; `>`; `==` (egyenlő); `<=`, `>=` (kisebb/nagyobb v egyenlő); `!=` (nem egyenlő)

Logikai operátorok: logikai kifejezések között értelmezett operátorok

- és (`&&`), vagy (`||`), nem (`!`) és a zárójelek

A logikai operátorokkal összekapcsolt kifejezések kiértékelése

- a `||` $\prec$ `&&` $\prec$ `!` precedencia szerint
- de a kiértékelés csak addig tart, amíg az eredmény nem egyértelmű!

Logikai kifejezések kiértékelése

A logikai operátorokkal összekapcsolt kifejezések kiértékelése csak addig tart, amíg az eredmény nem egyértelmű

```
1  int x, y, z;
2  if ( x < y && y < z )
3    printf("x is less than z\n");
```

A fenti példában

- ha $x < y$ teljesül, még $y < z$-t is le kell ellenőrizni
- ha $x < y$ nem teljesül, akkor $y < z$ már mindegy

Számok összehasonlítása

`Integer` típusú számok esetén:

- a változók értéke bitenként kerül összehasonlításra
- azonos integer típusok esetében tehát nincsen probléma
- különböző integer típusok esetén implicit konverzió történik

A lebegőpontos számok esetében már lehetnek gondok

- a műveletek nem végtelenül pontosak: kerekítési hibák léphetnek fel
- a konstansokat 10-es számrendszerben adjuk meg, de a gépben minden bináris
- lásd Számábrázolás fóliák

Lebegőpontos számok összehasonlítása

Példaprogram:

```
int main() {
  double a = 0.1 + 0.2;
  if (a == 0.3) { cout << "egyenlo" << endl;
  }
  else {cout << "nem egyenlo" << endl;
    printf("     %.17f\n", 0.3);
    printf("a = %.17f\n", a);
  }
  return 0;
}
```

Figyelem! itt most vegyesen használjuk a C++ outstream-jét (`cout`) és a C-ből ismert kiíratást (`printf` függvény)

- néha kényelmes C függvényekhez visszanyúlni

Lebegőpontos számok összehasonlítása

```
int main() {
  double a = 0.1 + 0.2;
  if (a == 0.3) { cout << "egyenlo" << endl;
  }
  else {cout << "nem egyenlo" << endl;
    printf("     %.17f\n", 0.3);
    printf("a = %.17f\n", a);
  }
  return 0;
}
```

Fordító üzenet:

```
warning:  comparing floating with == or != is unsafe
[-Wfloat equal]
```

Futtatás:

```
1 nem egyenlo
2      0.29999999999999999
3 a =  0.30000000000000004
```

Megoldás lebegőpontos számok összehasonlítására

Be kell vezetnünk egy ϵ precizitást

- minden összehasonlításnál ekkora eltérést engedünk
- használjuk az abs függvényt az eltérés kiszámítására

```
int main() {
  double epsilon = 1e-7;
  double a = 0.1 + 0.2;
  if (abs(a - 0.3) < epsilon) {
  cout << "majdnem egyenlo" << endl;}
  else {
  cout << "nem egyenlo" << endl; }
  return 0;
}
```

Még néhány aritmetikai operátor

Inkrementáló operátorok

- `i++` – post-increment
  a változó értékét eggyel növeli, de csak miután a kifejezés kiértékelődött
- `++i` – pre-increment
  a változó értékét eggyel növeli, és ezt az értéket adja vissza
- pl: 6 + (++i) = 7 + (i++);
  de a következő utasításban i értéke már eggyel nagyobb lesz, mint volt
- hasonlóan `i- -`, `- -i` post/pre decrement

Ezekkel gyakran találkozhatunk különböző helyzetekben

Értékadó operátorok

- `a += 5` – a változó értékét 5-tel növeli: $a = a + 5$
- `b *= 2` – a változó érékét megszorozza 2-vel: $a = 5 * a$