Operátorok, precedencia szabályok

Kormányos Andor

Komplex Rendszerek Fizikája Tanszék

2022. szeptember 13.

Kifejezések kiértékelési sorrendje

Egy kifejezésben szerepelhet:

- függvényhívás (minden argumentum egy további kifejezés!)
- zárójelek
- előjelek (`+`, `-`)
- aritmetikai operátorok (`+`, `-`, `*`, `/`, `%`)
- logikai operátorok (`==`, `!=`, `<`, `>`, `<=`, `>=`)

és még egy sor további operátor, 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

Logikai kifejezések

Összehasonlító operátorok

- `<`, `>`, `==` (egyenlő), `!=` (nem egyenlő)
- ezek eredménye 0 vagy 1, azaz egy integer szám!

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

A logikai operátorokkal összekapcsolt kifejezések kiértékelése

- a `||` $\prec$ `&&` $\prec$ `!` precedencia szerint
- de a kiértékelés 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

A processzor két változó értékét bitenként hasonlítja össze

- 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 hiba
- a konstansokat 10-es számrendszerben adjuk meg, de a gépben minden bináris

Lebegőpontos számok összehasonlítása

Példaprogram:

```
1  int main() {
2    double a = 0.1 + 0.2;
3    if (a == 0.3) {
4      printf("egyenlo\n");
5    } else {
6      printf("nem egyenlo\n");
7      printf("    %.17f\n", 0.3);
8      printf("a = %.17f\n", a);
9    }
10   return 0;
11 }
```

Lebegőpontos számok összehasonlítása

Példaprogram:

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

Kimenet:

```
nem egyenlo
    0.29999999999999999
a = 0.30000000000000004
```

Megoldás lebegőpontos számok összehasonlítására

Be kell vezetnünk egy ϵ precizitást

- minden összehasonlításnál ekkora eltérést engedünk

```c
int main() {
  double epsilon = 1e-7;
  double a = 0.1 + 0.2;
  if (fabs(a - 0.3) < epsilon) {
    printf("majdnem egyenlo\n");
  } else {
    printf("nem egyenlo\n");
    printf("    %.17f\n", 0.3);
    printf("a = %.17f\n", a);
  }
  return 0;
}
```

Kimenet:

```
majdnem egyenlo
```

Osztás nullával vagy nagyon kicsi számmal

Problémák:

- nullával nem szabad osztani
- nagyon kicsi számmal igen, de lehet, hogy az eredményt már nem képes ábrázolni a `double`

A `double` szabvány előír néhány speciális értéket

- `NaN`: not-a-number, nem szám
- `Inf`: infinity, végtelen

```
1 int main() {
2   printf("%.17f\n", 1.0 / 0.0);
3   return 0;
4 }
```

Kimenet:

```
1 1.#INF0000000000000
```

Még pár operátor

Inkrementáló operátorok

- `i++` – post-increment
  a változó értékét eggyel növeli, de csak miután a kifejezés kiértékelődött
- `-i` – pre-decrement
  a változó értékét eggyel csökkenti, de csak miután a kifejezés kiértékelődött

Értékadó operátorok

- `a += 5` – a változó értékét 5-tel növeli
- `b *= 2` – hasonló