Imperatív programozás

Utasítások

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. augusztus 21.

Tartalomjegyzék

Utasítások

- Egyszerű utasítások
  - Változódeklaráció
  - Értékadás
  - Alprogramhívás
  - Visszatérés függvényből
- Vezérlési szerkezetek
  - Szekvencia
  - Elágazás
  - Ciklus

Változódeklaráció

- Minden változót az első használat előtt létrehozunk
- Érdemes már itt inicializálni

```
double m;
int n = 3;
char cr = '\r', lf = '\n';
int i = 1, j;
int u, v = 3;
```

Kifejezés-utasítás

- (Mellékhatásos) kifejezés kiértékelése
- Tipikus példa: értékadások

```
<statement> ::= <expression>;
              | ...
```

```
n = 1;
x *= y;
c++;
n > 0 ? --n : ++n;      /* Nem idiomatikus! */
```

Visszatérés

- Egy függvényben akár több `return` utasítás is lehet
- Nincs `return` ≡ üres `return` (`void`)

```
int twice(int x) {
  return 2 * x;
}

bool isPrime(int x) {
  for (int i = 2; i <= x / 2; ++i)
    if (x % i == 0)
      return false;
  return true;
}

printf("42 duplája: %d\n", twice(42));

if (isPrime(11))
  printf("A 11 prím szám\n");
```

Függvényhívás

- Deklarált visszatérési típus, megfelelő `return` utasítás(ok)
- Csak mellékhatás: `void` visszatérési érték, üres `return`

Tiszta függvény

```
unsigned long fact(int n)
{
  unsigned long result = 1L;
  int i;
  for (i = 2; i <= n; ++i)
    result *= i;
  return result;
}
```

Csak mellékhatás

```
void printSquares(int n)
{
  int i;
  for (i = 1; i <= n; ++i) {
    printf("%d\n",i*i);
  }
  return;   /* Elhagyható */
}
```

Kevert viselkedés

```
printf("%d\n", printf("%d\n", 42));
```

Üres utasítás

Semmit sem csináló utasítás

```
;
```

Végtelen ciklus

```
int i = 0;
while (i < 10);
  printf("%d\n");
```

Első negatív szám a tömbben

```
int nums[] = {3, 6, 1, 45, -1, 4};

for (int i = 0; i < 6 && nums[i] < 0; ++i);

for (int i = 0; i < 6 && nums[i] < 0; ++i) {
}
```

Vezérlési szerkezetek

- Szekvencia
- Elágazás
- Ciklus
  - Tesztelő – Elöltesztelő – Hátultesztelő
  - Léptető
- Nem strukturált vezérlésátadás
  - `return`
  - `break`
  - `continue`
  - `goto`

Strukturált programozás

- Szekvencia, elágázás, ciklus
- Minden algoritmus leírható ezekkel
- Olvashatóbb, könnyebb érvelni a helyességéről
- Csak nagyon alapos indokkal térjünk el tőle!

Szekvencia

- Utasítások egymás után írásával
- Pontosvessző
- Blokk utasítás

```
<statement>      ::= {<statement-list>}
                   | ...
<statement-list> ::= ""
                   | <statement> <statement-list>
```

Vezérlési szerkezetek belseje

- Egy utasítás
- Lehet blokk utasítás is

```
int arr[10];

for (int i = 0; i < 10; ++i)
  arr[i] = i + 1;

int pos = 0;
while (pos < 10)
{
  printf("%d\n", arr[pos]);
  ++pos;
}
```

Elágazás

- `if - else` szerkezet
- Az `else` ág opcionális

Idióma

```
if (x > 0)
  y = x;
else if (y > 0)
  x = y;
else
  x = y = x * y;
```

Konvencionális tördelés

```
if (x > 0)
  y = x;
else
  if (y > 0)
    x = y;
  else
    x = y = x * y;
```

A kapcsos zárójelek nem ártanak

Idióma

```
if (x > 0) {
  y = x;
} else if (y > 0) {
  x = y;
} else {
  x = y = x * y;
}
```

Konvencionális tördelés

```
if (x > 0) {
  y = x;
} else {
  if (y > 0) {
    x = y;
  } else {
    x = y = x * y;
  }
}
```

Csellengő else (dangling else)

Ezt írtam

```
if (x == 1)
  if (y == 2)
    printf("hello");
else
  printf("world");
```

Ezt jelenti

```
if (x == 1)
  if (y == 2)
    printf("hello");
  else
    printf("world");
```

Ezt akartam

```
if (x == 1) {
  if (y == 2)
    printf("hello");
} else
  printf("world");
```

Lásd még...

goto-fail (Apple) link!

switch-case-break utasítás

Egész típusú, fordítási idejű konstansok alapján

```
switch (dayOf(date()))
{
  case 0:  strcpy(name, "Sunday");    break;
  case 1:  strcpy(name, "Monday");    break;
  case 2:  strcpy(name, "Tuesday");   break;
  case 3:  strcpy(name, "Wednesday"); break;
  case 4:  strcpy(name, "Thursday");  break;
  case 5:  strcpy(name, "Friday");    break;
  case 6:  strcpy(name, "Saturday");  break;
  default: strcpy(name, "illegal value");
}
```

Átcsorgás

```
switch (month)
{
  case 1:
  case 3:
  case 5:
  case 7:
  case 8:
  case 10:
  case 12: days = 31;
           break;
  case 2:  days = 28 + (isLeapYear(year) ? 1 : 0);
           break;
  default: days = 30;
}
```

Nem triviális átcsorgás

```
switch (getKey())
{
  case 'q': jump = 1;
  case 'a': moveLeft();
            break;
  case 'e': jump = 1;
  case 's': moveRight();
            break;
  case ' ': openDoor();
}
```

A `switch` és a strukturált programozás

Strukturáltnak tekinthető

- Minden ág végén `break`
- Ugyanaz az utasítássorozat többághoz

Nem felel meg a strukturált programozásnak

- Nem triviális átcsorgások
- Pl. ha egyáltalán nincs `break`

Elöltesztelő ciklus

```
<while-stmt> ::= while (<expression>) <statement>
```

```
while (i > 0)
{
  printf("%d\n", i);
  --i;
}
```

```
while (i > 0)
  printf("%d\n", i--);
```

Hátultesztelő ciklus

<do-while-stmt> ::= do <statement> while (<expression>);

Jellemző példa

```
char command[LENGTH];
do {
  read_data(command);
  if (strcmp(command, "START") == 0) {
    printf("start\n");
  } else if (strcmp(command, "STOP") == 0) {
    printf("stop\n");
  }
} while (strcmp(command, "QUIT") != 0);
```

Átírások

Milyen feltétel mellett igaz ez?

`do` σ `while (`ε`);` $\equiv$ σ `while (`ε`)` σ

Milyen feltétel mellett igaz ez?

`do` σ `while (`ε`);`

$\equiv$

`int new_var = 1; ... while (new_var) {` σ`; new_var =` ε`; }`

Az előző példa átírva

```
char command[LENGTH];
int new_var = 1;
...
while (new_var) {
  read_data(command);
  if (strcmp(command, "START") == 0 ) {
    ...
  } else if (strcmp(command, "STOP") == 0 ) {
    ...
  }
  new_var = (strcmp(command, "QUIT") != 0);
}
```

Refaktorálva

```
char command[LENGTH];
int stay_in_loop = 1;
...
while (stay_in_loop) {
  read_data(command);
  if (strcmp(command, "START") == 0) {
    ...
  } else if (strcmp(command, "STOP") == 0) {
    ...
  } else if (strcmp(command, "QUIT") == 0) {
    stay_in_loop = 0;
  }
}
```

Léptető ciklus

```
<for-stmt> ::= for (<optional-expression>;
                    <optional-expression>;
                    <optional-expression>)
                <statement>
<optional-expression> ::= "" | <expression>
```

(inicializáció; feltétel; léptetés)

Példa

```
unsigned char c;
for (c = 0; c < 256; ++c)
{
  printf("%d\t%c\n", c, c);
}
```

Végtelen ciklus

```
while (1) ...

for (;;) ...
```

Átírások

Mindig megtehető

`while (`ε`)` $\Rightarrow$ `for (;` ε`;)` σ

Milyen feltétel mellett igaz ez?

`for (`ι`;` ε`;` λ`)` σ $\Rightarrow$ ι`; while (`ε`) {` σ`;` λ`; }`

Strukturált programozás vezérlési szerkezetei

- Blokk utasítás
- Elágazások
  - `if-else`
  - `switch-case-break`
- Ciklusok
  - Tesztelő ciklusok
    - Elöltesztelő (`while`)
    - Hátultesztelő (`do-while`)
  - Léptető ciklus (`for`)

Nem strukturált vezérlésátadás

- `return`
- `break`
- `continue`
- `goto`

break utasítás

- Kilép a legbelső ciklusból (vagy `switch`-ből)

```
for (int i = 0; i < 10; ++i)
{
  if (i == 5)
    break;

  printf("%d", i);      /* 0 1 2 3 4 */
}
```

continue utasítás

- Befejezi a legbelső ciklusmag végrehajtását
- for-ciklusnál végrehajtja a léptetést

```
for (int i = 0; i < 10; ++i)
{
  if (i == 5)
    continue;

  printf("%d", i);      /* 0 1 2 3 4 6 7 8 9 */
}
```

goto utasítás

- Egy függvényen belül a megadott címkéjű utasításra ugrik

```
<statement> ::= ...
              | goto <label>
              | <label>: <statement>
<label> ::= <identifier>
```

Keressünk nulla elemet egy mátrixban

goto-val

```
int matrix[SIZE][SIZE];
...
int found = 0;
int i, j;
for (i=0; i<SIZE; ++i) {
  for (j=0; j<SIZE; ++j) {
    if (matrix[i][j] == 0) {
      found = 1;
      goto end_of_search;
    }
  }
}
/* --i; --j; */
end_of_search:;
```

Szabályosan

```
int matrix[SIZE][SIZE];
...
int found = 0;
int i= -1, j;
while (i < SIZE - 1 && !found) {
  j = -1;
  while (j < SIZE - 1 && !found) {
    if (matrix[i + 1][j + 1] == 0) {
      found = 1;
    }
    j++;
  }
  i++;
}
```