Instrucțiuni

30 octombrie 2003

Instrucțiunile limbajului C

Instrucțiunea *expresie* *expresie*$_{opt}$;
– *orice* expresie, evaluată pentru efectele ei laterale; în particular:
expresii de *atribuire*: `x = y + 1; y *= 2; --z;`
apel de funcție (ignorând valoarea returnată): `printf("salut!\n");`
instrucțiunea *vidă* `;` (expresia lipsește)
Exemplu: ciclu cu corp vid `while (s[i++]);`
Obs: în C `;` nu e separator, ci *face parte* din anumite instrucțiuni

Instrucțiunea *compusă* (bloc) `{` *lista-declarații lista-instrucțiuni* `}`
grupează declarațiile/instrucțiunile din listă sintactic într-o instrucțiune
poate fi încuibată (conține alte blocuri); poate fi vidă `{ }`
lista-declarații nu poate conține *definiții* de funcții

În C99 (și în C++) un bloc poate conține declarații și instrucțiuni în orice ordine.

Instrucțiuni etichetate

identificator : *instrucțiune*
`case` *expresie-constantă-int* : *instrucțiune*
`default :` *instrucțiune*

– permit referirea la instrucțiune pentru un salt (explicit sau implicit).
Etichetele au *spațiu de nume* separat de cel al identificatorilor obișnuiți (putem avea variabile/funcții/etc. și etichete cu același nume)

Domeniul de vizibilitate al etichetei e corpul funcției în care se află (numele de etichete trebuie să fie unice în cadrul unei funcții)

Etichetele `case` și `default` pot apare doar în instrucțiunea `switch`.
Într-o instrucțiune `switch` poate exista cel mult o etichetă `default`
iar constantele întregi din etichetele `case` vor fi distincte.

Instrucțiuni de selecție

Instrucțiunea `if` `if (` *expresie* `)` *instrucțiune1*
sau `if (` *expresie* `)` *instrucțiune1* `else` *instrucțiune2*
– expresia trebuie să fie de tip scalar (întreg, real, enumerare)
– dacă expresia e nenulă se execută *instrucțiune1*, altfel *instrucțiune2*
– un `else` e asociat întotdeauna cu cel mai apropiat `if`

Instrucțiunea `switch` `switch (` *expresie-intreagă* `)` *instrucțiune*
– se evaluează expresia (de tip întreg, posibil limitată la 1023 valori)
– dacă în corpul *instrucțiune* (compusă) există o etichetă `case`
cu valoarea întreagă obținută, se sare la instrucțiunea respectivă
– dacă nu, și există o etichetă `default`, se sare la acea instrucțiune
– altfel nu se execută nimic (se trece la instrucțiunea următoare)
– pt. același cod la mai multe etichete: `case` *val1*`:` `case` *val2*`:` *șir-instr*
Obs: Execuția nu se oprește la următorul `case` (e doar o etichetă);
ieșirea din `switch`: doar cu instruct. `break` sau la sfârșitul corpului!
⇒ permite utilizarea de cod comun pe ramuri, dar cu mare atenție!

Instrucțiunea `switch`: exemplu

```
char c; int a, b, r;
printf("Scrieti o operatie intre doi intregi: ");
if (scanf("%d %c %d", &a, &c, &b) == 3) { /* toate 3 corect */
  switch (c) {
    case '+': r = a + b; break;  /* iese din corpul switch */
    case '-': r = a - b; break;  /* idem */
    default: c = '\0'; break;    /* fanion caracter eronat */
    case 'x': c = '*';    /* 'x' e tot înmulțire, continuă */
    case '*': r = a * b; break;  /* ca și pt.'*' apoi iese */
    case '/': r = a / b;    /* la sfârșit nu trebuie break */
  }
  if (c) printf("Rezultatul: %d %c %d = %d\n", a, c, b, r);
  else printf("Operație necunoscută\n");
} else printf("Format eronat\n");
```


Ciclurile cu test inițial și final

Instrucțiunile *while* și *do* (ciclurile cu test inițial și final)

`while (` *expresie* `)` *instrucțiune*
`do` *instrucțiune* `while (` *expresie* `);`

– ambele execută *instrucțiunea* atât timp cât valoarea expresiei
(de tip scalar) e nenulă (adevărată).
– diferă momentul de evaluare a expresiei (înainte/după fiecare iterație)

Obs: În Pascal, din `repeat ... until` se iese pe condiție *true* (invers!)

Instrucțiunea `for`

`for` (*exp-init* ; *exp-test* ; *exp-cont*)
  instrucțiune

e echivalentă* cu:

exp-init;
`while` (*exp-test*) {
  instrucțiune;
  exp-cont;
}

* excepție: instrucțiunea `continue`, vezi ulterior

– oricare din cele 3 expresii poate lipsi (dar cele două ; rămân)
– dacă *exp-test* lipsește, e tot timpul adevărată (ciclu infinit)

În C99 (ca și în C++) se permite ca expresia *exp-init* să fie înlocuită cu o *declarație* de variabile (evtl. inițializate) cu domeniu de vizibilitate întreaga instrucțiune.

```
for (int i = 0; i < 10; ++i) { /* corpul ciclului */ }
```


Instrucțiunea `return`

`return` $expresie_{opt}$;

– încheie execuția funcției curente
– returnează valoarea expresiei date (dacă este prezentă)
Obs: într-o funcție care nu are tipul `void`, fiecare cale prin cod trebuie să returneze o valoare; nu e voie să se atingă ultima acoladă `}`.

```
int pos(char s[], char c)      /* prima pozitie a lui c in s */
{
  int i = 0;
  do
    if (s[i] == c) return i;   /* returnează poziţia găsită */
  while (s[i++]);
  return -1;                   /* -1 ca fanion, nu s-a găsit */
}
```

– `main` returnează un `int` (succes/cod eroare) sistemului de operare; se declară `int main(void)`; implicit returnează 0 (succes)

Instrucțiunea `break`

– produce ieșirea din corpul instrucțiunii `while, do, for` sau `switch` *imediat înconjurătoare*; execuția continuă cu instrucțiunea următoare
– mai convenabilă decât testarea unei variabile boolene la ciclul următor
– mai lizibil, dacă codul peste care se sare e complex

```
const int MAX = 20;
int i, t[MAX], v;
/* ... aici dăm nişte valori lui v şi t */
for (i = MAX; --i >= 0; ) /* caută pe v în tabloul t */
  if (t[i] == v) break;
if (i == -1) printf("nu s-a găsit\n");
else printf("găsit la poziţia %d\n", i);
```


Instrucțiunea `continue`

– produce trecerea la sfârșitul iterației într-un ciclu `while, do` sau `for` începând cu testul pt. `while` și `do`, și cu *expr3* (actualizare) pt. `for` (controlul trece la punctul din ciclu de după ultima instrucțiune)
– la fel, cod mai lizibil, dacă partea neexecutată din iterație e complexă

```
for (d = 2; ; d++) {     /* descompune n > 1 în factori primi */
  if (n % d != 0) continue;  /* nu se împarte, următorul! */
  exp = 0;
  do                     /* repetă de câte ori d e factor */
    exp++;
  while ((n /= d) % d == 0);
  printf ("%d^%d ", d, exp);     /* scrie factorul curent */
  if (n == 1) break;             /* am terminat */
}
```


Instrucțiunea `goto`

Sintaxa: `goto` *eticheta* ;
Efectul: se sare la execuția instrucțiunii cu *eticheta* specificată
Obs: orice instrucțiune poate fi etichetată opțional *etichetă* : *instr*

– instrucțiunea `goto` nu corespunde principiilor programării structurate
– de evitat: duce ușor la programe dificil de înțeles și analizat
– orice program poate fi rescris fără folosirea lui `goto`
(eventual utilizând teste și/sau variabile boolene suplimentare)
– poate fi totuși utilă, ex. pentru ieșirea din mai multe cicluri încuibate

```
while (...) { /* scriem într-un fişier, linie cu linie */
  while (...) { /* prelucrăm cuvintele şi spaţiile din linie */
    if (eroare_la_scriere)
      goto eroare; /* abandonează ciclurile */
  }
}
eroare: /* cod pt. tratarea erorii */
```