ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Τμήμα Πληροφορικής και Τηλεπικοινωνιών • Εισαγωγή στον Προγραμματισμό

Περιεχόμενα

Κεφάλαιο 3: Η ροή του ελέγχου

Η ροή του ελέγχου στην C

- Όπως σε κάθε γλώσσα προγραμματισμού, έτσι και στην C, υπάρχουν εντολές που επηρεάζουν τη ροή του ελέγχου στα προγράμματα (`if`, `switch`, `while`, `for`, κλπ.).
- Στις εντολές αυτές είναι πολύ συχνή η χρήση ενός μπλοκ εντολών. Ένα μπλοκ εντολών αποτελείται από εντολές που είναι κλεισμένες μέσα σε άγκιστρα ({ και }). Αν σ' ένα μπλοκ εντολών περιέχεται μία μόνο εντολή, αυτή δεν είναι απαραίτητο να περικλείεται μέσα σε άγκιστρα.
- Είναι εξαιρετικά χρήσιμο να μελετηθεί όλο το Κεφάλαιο 3 από το [KR] (σελ. 85–100), στο οποίο περιγράφονται αναλυτικά όλες οι δομές ελέγχου που προσφέρει η C. Ακολουθεί μία πιο συνοπτική περιγραφή των δομών αυτών.

Εντολή `if`

- Η σύνταξη της εντολής `if` είναι:
  `if (⟨παράσταση⟩)`

⟨μπλοκ εντολών⟩$_1$
`else`
⟨μπλοκ εντολών⟩$_2$

- Αν η ⟨παράσταση⟩ είναι αληθής (έχει τιμή διάφορη από το μηδέν), θα εκτελεσθεί το ⟨μπλοκ εντολών⟩$_1$, αλλιώς το ⟨μπλοκ εντολών⟩$_2$.
- Το τμήμα από το `else` και μετά είναι προαιρετικό. Αν δεν υπάρχει και η ⟨παράσταση⟩ είναι ψευδής, ο έλεγχος θα πάει στην επόμενη εντολή του προγράμματος, δηλαδή στην πρώτη εντολή μετά το `if`.
- Η προαιρετικότητα του `else` μπορεί να προκαλέσει προβλήματα στην αναγνωσιμότητα προγραμμάτων με εμφωλευμένες εντολές `if`, αν δεν υπάρχει σαφής ομαδοποίηση με άγκιστρα. Αν δεν υπάρχουν άγκιστρα που να επιβάλλουν συγκεκριμένη δομή, κάθε `else` αντιστοιχεί στην αμέσως προηγούμενη `if` που δεν έχει `else`.
- Τι ακριβώς σημαίνει το παρακάτω;

```
if (n > 0)
   if (a > b)
      z = a;
else
   z = b;
```

Προσοχή! Ο μεταγλωττιστής δεν ασχολείται με τη στοίχιση.

- Αυτό:

```
if (n > 0) {
   if (a > b)
      z = a;
   else
      z = b;
}
```

- Αν θέλαμε το `else` να αντιστοιχεί στο πρώτο `if`, έπρεπε να το γράψουμε έτσι:

```
if (n > 0) {
   if (a > b)
      z = a;
}
else
   z = b;
```

- Μία συνηθισμένη χρήση της εντολής `if` είναι όταν θέλουμε να ελέγξουμε μία σειρά από επάλληλες συνθήκες, και ανάλογα με το ποια θα βρεθεί ότι ισχύει πρώτη, να εκτελέσουμε ένα μπλοκ εντολών. Αυτό γίνεται ως εξής:
`if (`⟨παράσταση⟩$_1$`)`

⟨μπλοκ εντολών⟩$_1$
`else if (`⟨παράσταση⟩$_2$`)`
⟨μπλοκ εντολών⟩$_2$
……
`else if (`⟨παράσταση⟩$_{n-1}$`)`
⟨μπλοκ εντολών⟩$_{n-1}$
`else` ⟨μπλοκ εντολών⟩$_n$

- Η προηγούμενη εντολή `if` θα μπορούσε να γραφεί σε μία εκδοχή με στοίχιση ως εξής:
  `if (`⟨παράσταση⟩$_1$`)`
  ⟨μπλοκ εντολών⟩$_1$
  `else`
  `if (`⟨παράσταση⟩$_2$`)`
  ⟨μπλοκ εντολών⟩$_2$
  ……
  `else`
  `if (`⟨παράσταση⟩$_{n-1}$`)`
  ⟨μπλοκ εντολών⟩$_{n-1}$
  `else`
  ⟨μπλοκ εντολών⟩$_n$
  Η συγκεκριμένη στοίχιση, όμως, μάλλον κάνει πιο δυσανάγνωστη την εντολή, οπότε είναι καλό να μην την ακολουθήσει κάποιος.

- Ένα παράδειγμα:

```c
if (x > 0) {
   sign = 1;
   printf("Number is positive\n");
}
else if (x < 0) {
   sign = -1;
   printf("Number is negative\n");
}
else {
   sign = 0;
   printf("Number is zero\n");
}
```

Εντολή `switch`

- Η σύνταξη της εντολής `switch` είναι:
  `switch (`⟨παράσταση⟩`) {`
  `case` ⟨σταθερά⟩$_1$ `:`
  ⟨εντολές⟩$_1$

```
    case ⟨σταθερά⟩_2 :
    ⟨εντολές⟩_2
    ..........................
    default:
    ⟨εντολές⟩
}
```

Η `switch` είναι μία διακλαδωμένη εντολή απόφασης που λειτουργεί όπως περιγράφεται στη συνέχεια.

- Η ⟨παράσταση⟩ πρέπει να είναι ακέραια. Υπολογίζεται η τιμή της.
- Κάθε $\langle$σταθερά$\rangle_i$ πρέπει να είναι ακέραια σταθερά και δεν πρέπει να υπάρχουν δύο `case` με την ίδια σταθερά.
- Αν η τιμή που έχει η ⟨παράσταση⟩ ισούται με κάποια $\langle$σταθερά$\rangle_i$, τότε ο έλεγχος μεταφέρεται στις $\langle$εντολές$\rangle_i$. Αν όχι, ο έλεγχος μεταφέρεται στις ⟨εντολές⟩ (μετά το `default`).
- Οι $\langle$εντολές$\rangle_i$ (και ⟨εντολές⟩) δεν είναι απαραίτητο να συνιστούν μπλοκ εντολών, δηλαδή να είναι κλεισμένες μέσα σε { και }.
- Μετά την εκτέλεση των εντολών σε μία `case`, ο έλεγχος μεταφέρεται στις εντολές της επόμενης `case`, εκτός αν τελευταία εντολή της προηγούμενης `case` είναι η `break`. Δεν είναι καλή πρακτική οι εντολές μίας `case` να συνεχίζουν με αυτές της επόμενης, εκτός αν ισχύει το επόμενο.
- Είναι δυνατόν για περισσότερες τιμές της μίας για την ⟨παράσταση⟩ να θέλουμε να εκτελεσθεί η ίδια ομάδα εντολών. Σ' αυτήν την περίπτωση, τοποθετούμε τα αντίστοιχα `case` συνεχόμενα και βάζουμε την ομάδα εντολών στο τελευταίο απ' αυτά.
- Η περίπτωση `default` είναι προαιρετική. Αν δεν υπάρχει και δεν ταιριάζει καμία `case`, ο έλεγχος μεταφέρεται μετά την εντολή `switch`.
- Παράδειγμα:

```c
switch (x) {
   case 1:  printf("one\n");
            break;
   case 2:
   case 3:  printf("two or three\n");
            break;
   case 4:  printf("four\n");
            break;
   default: printf("other\n");
            break;
}
```

Τι θα γινόταν αν έλειπαν κάποιες `break`; Γιατί βάλαμε `break` και στην `default` περίπτωση;

Εντολές βρόχου `while`

- Μία πιθανή σύνταξη της εντολής `while` είναι:
  `while (`⟨παράσταση⟩`)`
  ⟨μπλοκ εντολών⟩
- Αρχικά υπολογίζεται η ⟨παράσταση⟩. Αν έχει τιμή διάφορη του μηδενός (είναι αληθής) εκτελείται το ⟨μπλοκ εντολών⟩. Υπολογίζεται πάλι η ⟨παράσταση⟩ και ενόσω είναι αληθής, θα γίνεται ο κύκλος εκτέλεσης του ⟨μπλοκ εντολών⟩, μέχρι να γίνει η ⟨παράσταση⟩ ψευδής (δηλαδή ίση με το μηδέν).
- Παράδειγμα:

  ```
  f = 1;
  while (--n)
     f = f*(n+1);
  ```

  Τι υπολογίζει αυτό το `while`;
- Επίσης, θυμηθείτε και την εντολή `while` στο `http://www.di.uoa.gr/~ip/cprogs/capitalize.c`.
- Άλλη πιθανή σύνταξη της εντολής `while` είναι:
  `do`
  ⟨μπλοκ εντολών⟩
  `while (`⟨παράσταση⟩`)`
- Αρχικά εκτελείται το ⟨μπλοκ εντολών⟩. Μετά, υπολογίζεται η ⟨παράσταση⟩. Αν έχει τιμή διάφορη του μηδενός (είναι αληθής) εκτελείται πάλι το ⟨μπλοκ εντολών⟩ και συνεχίζεται ο κύκλος εκτέλεσής του, μέχρι να γίνει η ⟨παράσταση⟩ ψευδής (δηλαδή ίση με το μηδέν).
- Για λόγους καλύτερης αναγνωσιμότητας του προγράμματος, ακόμα και αν το ⟨μπλοκ εντολών⟩ αποτελείται από μία μόνο εντολή, συνήθως τη βάζουμε μέσα σε { και }.
- Παράδειγμα:

  ```
  f = 1;
  do {
     f = f*n;
  } while (--n);
  ```

  Τι υπολογίζει αυτό το `while`;
- Επίσης, θυμηθείτε και την εντολή `do/while` στο `http://www.di.uoa.gr/~ip/cprogs/picomp.c`.
- Η βασική διαφορά των δύο εκδοχών της εντολής `while` είναι ότι στην πρώτη περίπτωση το ⟨μπλοκ εντολών⟩ μπορεί και να μην εκτελεσθεί καμία φορά, αν η ⟨παράσταση⟩ είναι αρχικά ψευδής, ενώ στην εκδοχή `do/while`, το ⟨μπλοκ εντολών⟩ θα εκτελεσθεί τουλάχιστον μία φορά, αφού η ⟨παράσταση⟩ ελέγχεται στο τέλος.

Εντολή βρόχου `for`

- Η σύνταξη της εντολής `for` είναι:
  `for (`⟨παράσταση⟩$_1$ `;` ⟨παράσταση⟩$_2$ `;` ⟨παράσταση⟩$_3$`)`
  ⟨μπλοκ εντολών⟩
- Διαδικαστικά, ισοδυναμεί με τις εξής εντολές:
  ⟨παράσταση⟩$_1$`;`
  `while (`⟨παράσταση⟩$_2$`) {`
  ⟨μπλοκ εντολών⟩
  ⟨παράσταση⟩$_3$`;`
  `}`
- Δηλαδή, αρχικά υπολογίζεται η ⟨παράσταση⟩$_1$. Συνήθως, πρόκειται για την αρχικοποίηση ενός δείκτη που ελέγχει μία επαναληπτική διαδικασία. Στη συνέχεια, εκτελείται επαναλαμβανόμενα το ⟨μπλοκ εντολών⟩, που αποτελεί το σώμα της διαδικασίας, ενόσω η ⟨παράσταση⟩$_2$, που συνήθως ελέγχει την τιμή του δείκτη ελέγχου, είναι αληθής. Πριν τη διεξαγωγή νέας επανάληψης, εκτελείται και η ⟨παράσταση⟩$_3$, που συνήθως μεταβάλλει τον δείκτη ελέγχου.
- Παράδειγμα:

  ```
  f = 1;
  for (i=1 ; i <= n ; i++)
     f = f*i;
  ```

  Τι υπολογίζει αυτό το `for`;
- Επίσης, θυμηθείτε και τις εντολές `for` στα `http://www.di.uoa.gr/~ip/cprogs/easter.c` και `http://www.di.uoa.gr/~ip/cprogs/magic.c`.
- Είναι δυνατόν κάποια από τις ⟨παράσταση⟩$_{1,2 \text{ ή } 3}$ να μην υπάρχει, συνήθως η πρώτη ή/και η τρίτη. Αν δεν υπάρχει η δεύτερη, τότε δεν γίνεται έλεγχος για τερματισμό του βρόχου, οπότε πρέπει αυτό να γίνει βιαίως με κάποιο τρόπο μέσα από το σώμα της επανάληψης. Σε κάθε περίπτωση, όποια ⟨παράσταση⟩$_i$ και να λείπει, τα `;` υπάρχουν κανονικά.
- Η εντολή
  `for ( ; ; )`
  ⟨μπλοκ εντολών⟩
  είναι η κλασική υλοποίηση ενός ατέρμονος βρόχου.
- Επίσης, στην C υπάρχει και ο τελεστής `,` (κόμμα), με τον οποίο μπορούμε να κατασκευάσουμε μία σύνθετη παράσταση που θα υπολογισθεί από αριστερά προς τα δεξιά. Η τιμή της σύνθετης παράστασης ισούται με την τιμή της δεξιότερης από τις απλές που την συνιστούν.
- Μερικές φορές, θέλουμε να γράψουμε μία εντολή `for` που θα ελέγχεται από περισσότερους του ενός δείκτες. Αυτό γίνεται χρησιμοποιώντας τον τελεστή `,` στην ⟨παράσταση⟩$_1$ και

στην ⟨παράσταση⟩$_3$. Παράδειγμα:

```
for (i=0, j=n ; i <= j ; i++, j--)
   printf("%d\n", i*j);
```

Εντολές `break` και `continue`

- Μία χρήση της εντολής `break` είναι αυτή που είδαμε στην εντολή `switch`, για τον τερματισμό της ακολουθίας εντολών που θα εκτελεσθούν όταν ταιριάξει κάποια συγκεκριμένη `case`.
- Επίσης, με την εντολή `break`, μπορούμε να τερματίσουμε βιαίως τις επαναλήψεις ενός βρόχου (`while`, `do/while` ή `for`), πριν ισχύσει η συνθήκη τερματισμού (ή όταν ο βρόχος είναι ατέρμων). Παράδειγμα:

```
while (!finished) {
   ......................
   if (bad_data)
      break;
   ......................
}
```

- Με την εντολή `continue`, ο έλεγχος στο εσωτερικό ενός βρόχου μεταφέρεται στο τέλος του, για να αρχίσει η επόμενη επανάληψη. Παράδειγμα:

```
for (i=0 ; i < n ; i++) {
   if (a[i] < 0)  /* Ignore negative elements */
      continue;
   .............  /* Process positive elements */
}
```

Εντολή `goto` και ετικέτες

- Κάθε γλώσσα προγραμματισμού, έτσι και η C, έχει μία εντολή για ρητή μεταφορά του ελέγχου σε κάποιο άλλο σημείο (ετικέτα) του προγράμματος. Στην C, η εντολή αυτή είναι η `goto`.
- Η `goto` μπορεί να χρησιμοποιηθεί σε πολύ εξειδικευμένες περιπτώσεις, για παράδειγμα όταν θέλουμε να γίνει βίαιη έξοδος από εμφωλευμένους βρόχους, όταν ο έλεγχος βρίσκεται σε κάποιο εσωτερικό βρόχο. Αυτό δεν μπορεί να γίνει με την εντολή `break`. Παράδειγμα:

```
for (i=0 ; i < n ; i++)
   for (j=0 ; j < m ; j++)
      if (a[i] == b[j])
         goto found;
```

```
    ..............   /* Didn't find common element */
    found:
    ..............   /* Found a[i] == b[j] */
```

- Η εντολή `goto` βλάπτει σοβαρά την ιδέα του δομημένου προγραμματισμού, γιατί μπορεί να οδηγήσει σε προγράμματα δυσανάγνωστα και δύσκολα συντηρήσιμα, και γι' αυτό πρέπει να χρησιμοποιείται σπάνια, αν όχι καθόλου. [1]

[1] Στα πλαίσια του συγκεκριμένου μαθήματος, απλά ΑΠΑΓΟΡΕΥΕΤΑΙ η χρήση της.