Προγραμματισμός Ι

Έλεγχος

Δημήτρης Μιχαήλ

Τμήμα Πληροφορικής και Τηλεματικής
Χαροκόπειο Πανεπιστήμιο

Σχεσιακοί Τελεστές και Ισότητας

Ένα πρόγραμμα εκτός από αριθμητικές πράξεις πρέπει να μπορεί να παίρνει και αποφάσεις.

Για παράδειγμα ένα πρόγραμμα που εκτυπώνει τους επιτυχόντες τους μαθήματος σε μία λίστα πρέπει να διαβάζει τον βαθμό κάθε φοιτητή και να μπορεί να ελέγξει εάν αυτός ο βαθμός είναι μεγαλύτερος ή ίσος του 5.

Σχεσιακοί Τελεστές και Ισότητας

Η γλώσσα C δεν παρέχει κάποιο συγκεκριμένο τύπο που να παίρνει τις τιμές TRUE ή FALSE αλλά χρησιμοποιεί τις τιμές $\neq 0$ και 0 για να αναπαραστήσει αυτές τις τιμές.

Σχεσιακοί Τελεστές και Ισότητας

Οι τελεστές ισότητας και οι σχεσιακοί τελεστές μας επιτρέπουν να κάνουμε συγκρίσεις και να παίρνουμε ως αποτέλεσμα μία τιμή 1 ή 0 (TRUE ή FALSE) την οποία μπορούμε μετά να χρησιμοποιήσουμε για να εκτελέσουμε διαφορετικές εντολές.

Σχεσιακοί Τελεστές και Ισότητας

τελεστής	κώδικας σε C	αποτέλεσμα συνθήκης
>	`x > y`	1 (TRUE) εάν $x > y$, αλλιώς 0 (FALSE)
<	`x < y`	1 (TRUE) εάν $x < y$, αλλιώς 0 (FALSE)
>=	`x >= y`	1 (TRUE) εάν $x \geq y$, αλλιώς 0 (FALSE)
<=	`x <= y`	1 (TRUE) εάν $x \leq y$, αλλιώς 0 (FALSE)
==	`x == y`	1 (TRUE) εάν $x = y$, αλλιώς 0 (FALSE)
!=	`x != y`	0 (FALSE) εάν $x = y$, αλλιώς 1 (TRUE)

Οι τελεστές αυτοί εκτελούνται από αριστερά προς τα δεξιά και έχουν μικρότερη προτεραιότητα από τους αριθμητικούς.

Προτεραιότητα Τελεστών

Ο τρόπος υπολογισμού μιας έκφρασης εξαρτάται από την προτεραιότητα των τελεστών:

1. **παρενθέσεις:** $()$, **`expr`**$++$, **`expr`**$--$
   Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Εάν υπάρχουν ένθετες υπολογίζονται πρώτα οι εσωτερικές.
2. **μοναδιαίοι αριθμητικοί τελεστές:** $+$, $-$, $++$**`expr`** ή $--$**`expr`**
   Υπολογίζονται από δεξιά προς τα αριστερά.
3. **πολλαπλασιασμός, διαίρεση και υπόλοιπο:** $*$, $/$, ή $\%$
   Υπολογίζονται δεύτερα από αριστερά προς τα δεξιά.
4. **πρόσθεση, αφαίρεση:** $+$ ή $-$
   Εάν υπάρχουν πολλοί, υπολογίζονται από τα αριστερά προς τα δεξιά.
5. **Σχεσιακοί:** $<$, $>$, $<=$, $>=$
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
6. **Ισότητας:** $==$, $!=$
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
7. **εκχώρησης:** $=$, $+=$, $-=$, $*=$, $/=$, $\%=$
   Από δεξιά προς τα αριστερά.

Απόφαση στην C

Η δομή `if`

```
1  int main() {
2      int grade;
3
4      grade = 8;
5
6      if (grade >= 5)
7          printf("PASS\n");
8  }
```

Μας επιτρέπει να εκτελούμε κάποιες εντολές ή όχι ανάλογα με μία συνθήκη που εξαρτάται από τις τιμές κάποιων σταθερών και κάποιων μεταβλητών.

Απόφαση στην C

Η δομή `if`

Η δομή `if` δέχεται ως συνθήκη οποιαδήποτε παράσταση επιστρέφει έναν ακέραιο. Εάν αυτός ο ακέραιος είναι διάφορος του 0 τότε εκτελεί τις γραμμές που ακολουθούν, εάν είναι 0 δεν τις εκτελεί.

```
1   int main() {
2       int grade;
3
4       grade = 8;
5
6       if (grade >= 5) {
7           printf("grade = %d ", grade);
8           printf("(PASS)\n");
9       }
10  }
```

Μπορούμε να εκτελέσουμε παραπάνω από μία εντολές.

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      int num1, num2;
5
6      printf("Enter two integers\n");
7      scanf("%d%d", &num1, &num2);
8
9      if (num1 == num2)
10         printf("%d is equal to %d\n", num1, num2);
11
12     if (num1 != num2)
13         printf("%d is not equal to %d\n", num1, num2);
14
15     if (num1 < num2)
16         printf("%d is less than %d\n", num1, num2);
17
18     if (num1 > num2)
19         printf("%d is greater than %d\n", num1, num2);
20
21     if (num1 <= num2)
22         printf("%d is less than or equal to %d\n", num1, num2);
23
24     if (num1 >= num2)
25         printf("%d is greater than or equal to %d\n", num1, num2);
26 }
```

Προσοχή: Τελεστής Εκχώρησης σε `if`

Προσοχή θέλει όταν χρησιμοποιούμε τον τελεστή ισότητας, ώστε να μην γράψουμε καταλάθος την ίδια έκφραση αλλά με τον τελεστή εκχώρησης.

```
1  if (x == 5) {
2      /* code to be executed */
3  }
```

και όχι

```
1  if (x = 5) {
2      /* code to be executed */
3  }
```

Αφού όπως είπαμε το **x**=5 επιστρέφει την τιμή 5 η οποία είναι διάφορη του μηδενός και άρα αληθής. Άρα το δεύτερο πρόγραμμα εκτελεί το **if** πάντα ανεξάρτητα από την τιμή του **x**.

Αλγόριθμοι και Ψευδοκώδικας

Ένας αλγόριθμος είναι μία μέθοδος επίλυσης κάποιου προβλήματος χρησιμοποιώντας ένα πεπερασμένο αριθμό βημάτων.

Δομές Ελέγχου

Σε μία δομημένη γλώσσα προγραμματισμού χρησιμοποιούμε τους παρακάτω μηχανισμούς για να υλοποιήσουμε αλγορίθμους:

1. σειριακή δομή
2. δομή επιλογής
3. δομή επανάληψης

Δομές Ελέγχου

Σε μία δομημένη γλώσσα προγραμματισμού χρησιμοποιούμε τους παρακάτω μηχανισμούς για να υλοποιήσουμε αλγορίθμους:

1. σειριακή δομή
   στην γλώσσα C οι εντολές εκτελούνται η μία μετά την άλλη (σειριακά) όπως δίνονται στον πηγαίο κώδικα.
2. δομή επιλογής
3. δομή επανάληψης

Δομές Ελέγχου

Σε μία δομημένη γλώσσα προγραμματισμού χρησιμοποιούμε τους παρακάτω μηχανισμούς για να υλοποιήσουμε αλγορίθμους:

1. σειριακή δομή
2. δομή επιλογής
   πολλές φορές όμως θέλουμε να μπορούμε να επιλέξουμε ποιος κώδικας θα εκτελεστεί, για αυτό η γλώσσα C μας παρέχει την λειτουργία **if-then-else**.
3. δομή επανάληψης

Δομές Ελέγχου

Σε μία δομημένη γλώσσα προγραμματισμού χρησιμοποιούμε τους παρακάτω μηχανισμούς για να υλοποιήσουμε αλγορίθμους:

1. σειριακή δομή
2. δομή επιλογής
3. δομή επανάληψης
   είναι άλλες φορές που θέλουμε να επαναλάβουμε τα ίδια βήματα πολλές φορές. Η γλώσσα C παρέχει διάφορες δομές επανάληψης, όπως **for, do..while**.

if/then/else

```
1  if (/* boolean expression */) {
2      /* code to be executed if true */
3  }
4  else {
5      /* code to be executed if false */
6  }
```

Το else είναι προαιρετικό...

Παράδειγμα if/then/else

```
1  int main() {
2      float grade = 6.5;
3
4      if (grade >= 5.0) {
5          printf("PASSED!");
6      } else {
7          printf("FAILED!");
8      }
9  }
```

Ένθετα if/then/else

Οι δομές επιλογής μπορούν φυσικά να είναι και ένθετες πράγμα που μας επιτρέπει να έχουμε μεγαλύτερο έλεγχο πάνω στις εντολές που θα εκτελεστούν.

```
1  int main() {
2      float grade = 6.5;
3
4      if (grade >= 5.0) {
5          printf("PASSED!");
6          if (grade >= 8.5) {
7              printf(" with distinction!!");
8          }
9          printf("\n");
10     } else {
11         printf("FAILED!\n");
12     }
13
14     return 0;
15 }
```

Τελεστής Συνθηκών

Ο μοναδικός τριαδικός τελεστής στη C σχετίζεται στενά με την δομή if/then/else.

`boolean expression ? value if true : value if false`

Έχει 3 τελεστές:

1. μία συνθήκη
2. η τιμή για την έκφραση εάν η συνθήκη είναι αληθής
3. η τιμή για την έκφραση εάν η συνθήκη είναι ψευδής

Τελεστής Συνθηκών

```c
int main() {
    float grade;

    printf("Enter your grade\n");
    scanf("%f", &grade);

    printf("%s\n", grade >= 5.0 ? "Passed" : "Failed");
}
```

ή

```c
int main() {
    float grade;

    printf("Enter your grade\n");
    scanf("%f", &grade);

    if (grade >= 5.0)
       printf("Passed\n");
    else
       printf("Failed\n");
}
```

Δομή Επανάληψης `while`

Η δομή επανάληψης **`while`** επιτρέπει στον προγραμματιστή να εκτελέσει κάποιες γραμμές κώδικα επαναληπτικά όσο μία συγκεκριμένη συνθήκη παραμένει αληθής.

```
1 while(condition) {
2     /* code to be executed if condition is true */
3 }
```

Παράδειγμα `while`

Εκτύπωση των αριθμών 0 έως 9

Θέλουμε να εκτυπώσουμε τους αριθμούς από 0 έως και 9 ένα αριθμό ανά γραμμή.

```
1  #include <stdio.h>
2
3  int main() {
4      int i = 0;
5
6      while(i < 10) {
7          printf("%d\n", i);
8          i = i + 1;
9      }
10 }
```

Ατέρμων Βρόχος

```
1  #include <stdio.h>
2
3  int main() {
4      int i = 0;
5
6      while(i < 10) {
7          printf("%d\n", i);
8      }
9  }
```

Τι θα κάνει το παραπάνω πρόγραμμα;

Δομή Επανάληψης `do/while`

Η δομή επανάληψης **`do/while`** είναι παρόμοια με την **`while`** αλλά εκτελείται πάντα τουλάχιστον μία φορά.

```
1 do {
2     /* code to be executed */
3 } while(condition);
```

Προσοχή στο ερωτηματικό στο τέλος, είναι υποχρεωτικό!

Δομή Επανάληψης `do/while`

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      int i = 0;
5
6      do {
7          printf("%d\n", i);
8      } while(++i < 10);
9  }
```

Δομή Επανάληψης `do/while`

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      const int SECRETCODE = 3313;
5      int code;
6
7      do {
8          printf("Type the secret code to enter.\n");
9          scanf("%d", &code);
10     } while (code!=SECRETCODE);
11
12     printf("Well done, you can now enter\n");
13     return 0;
14 }
```

Δομή Επανάληψης `for`

Η δομή επανάληψης **`for`** μας επιτρέπει να γράψουμε εύκολα επαναλήψεις με μετρητές, μιας και παρέχει χώρο για να γράψουμε την αρχικοποίηση και την αύξηση των μετρητών.

```
1 for(initializations; test conditions; increment value) {
2     /* block of code to be repeated */
3 }
```

Δομή Επανάληψης `for`

Παράδειγμα

```
1  for(i = 0; i < 10; i++) {
2      printf("%d\n", i);
3  }
```

Δομή Επανάληψης `for`

Παράδειγμα

Η συνθήκη μπορεί να είναι ανεξάρτητη από τις μεταβλητές του **`for`**.

```
1  #include <stdio.h>
2
3  int main() {
4      int i, j;
5
6      j = 0;
7      for(i = 0; j < 10; i += 2) {
8          printf("i = %d, j = %d\n", i, j);
9          j++;
10     }
11 }
```

Δομή Επανάληψης `for`

Παράδειγμα

Μπορούμε να έχουμε και πολλαπλές αρχικοποιήσεις, κ.τ.λ

```
1  #include <stdio.h>
2
3  int main() {
4      int i, j;
5
6      for(i = 0, j = 0; j < 10; i += 2, j++)
7          printf("i = %d, j = %d\n", i, j);
8  }
```

break

Πολλές φορές μπορεί να χρειαστεί να σταματήσουμε πρόωρα την εκτέλεση ενός βρόγχου.

```
1  #include <stdio.h>
2
3  int main() {
4      int i = 0;
5
6      while(i < 100) {
7          printf("%d\n", i);
8          if (i >= 9)
9              break;
10         i++;
11     }
12
13     printf("Due to break i should be 9: %s\n",(i==9)?"YES":"NO");
14 }
```

Μπορούμε να χρησιμοποιήσουμε την **break** σε διάφορους βρόγχους.

continue

Άλλες φορές θέλουμε να παραλείψουμε τις υπόλοιπες εντολές του βρόγχου και να συνεχίσουμε με την επόμενη επανάληψη.

```
1  #include <stdio.h>
2
3  int main() {
4      int i,j;
5
6      for(i = 0, j = 0; i < 10; i++, j++) {
7          printf("j = %d\n", j);
8          if (i == 5)
9              continue;
10         printf("i = %d\n", i);
11     }
12 }
```

Διπλό Loop

Πολλές φορές χρειάζεται να κάνουμε διπλό loop.

```
1  int main() {
2      int i,j;
3      for(i = 0; i < 10; i++) {
4          for(j = 0; j < 5; j++) {
5              printf("(%d,%d) ", i, j);
6          }
7          printf("\n");
8      }
9  }
```

Τι τυπώνει ο παραπάνω κώδικας;

Δομή Πολλών Επιλογών `switch`

Υπάρχουν περιπτώσεις που το **`if`**/**`then`**/**`else`** και οι δύο περιπτώσεις που μας παρέχει δεν είναι αρκετές.

```
1  switch(expression) {
2  case value1:
3      /* code if value1 */
4      break;
5  case value2:
6      /* code if value2 */
7      ...
8  default:
9      /* execute default action */
10     break;
11 }
```

- η τιμή της έκφρασης πρέπει να είναι ακέραιος
- οι τιμές στο **`case`** πρέπει να είναι σταθερές

Δομή Πολλών Επιλογών `switch`

Παράδειγμα

```
1  #include <stdio.h>
2
3  int main() {
4      int acount = 0, bcount = 0;
5      char c;
6
7      while ((c = getchar()) != EOF) {
8          switch(c) {
9          case 'a':
10         case 'A':
11             acount++;
12             break;
13         case 'b':
14         case 'B':
15             bcount++;
16             break;
17         default:
18             break;
19         }
20     }
21     printf("number of a: %d, number of b: %d\n", acount, bcount);
22 }
```

Πολύπλοκες Συνθήκες (AND OR NOT)

Η C μας παρέχει τους λογικούς τελεστές &&, || και ! για να μπορούμε να γράψουμε πιο πολύπλοκες συνθήκες ελέγχου.

τελεστής	σημασία	παράδειγμα
&&	λογικό AND	`x` > 10 && `x` < 20
||	λογικό OR	`x` <= 10 || `x` >= 20
!	λογικό NOT	!(`x`>10 && `x` < 20)

Παράδειγμα Πολύπλοκης Συνθήκης

```
1  /*
2   * A program to count letters in input.
3   */
4  #include <stdio.h>
5
6  int main() {
7      int c ;
8      int count = 0;
9
10     while ((c = getchar()) != EOF) {
11         if ((c >= 'A') && (c <= 'Z') ||
12             (c >= 'a') && (c <= 'z')) {
13             count++;
14         }
15     }
16
17     printf("%d letters\n" , count) ;
18 }
```

Προτεραιότητα Τελεστών

Ο τρόπος υπολογισμού μιας έκφρασης εξαρτάται από την προτεραιότητα των τελεστών:

1. **παρενθέσεις:** $()$, **`expr`**$++$ ή **`expr`**$--$
   Υπολογίζονται πρώτα, από τα αριστερά προς τα δεξιά. Εάν υπάρχουν ένθετες υπολογίζονται πρώτα οι εσωτερικές.
2. **μοναδιαίοι τελεστές:** $+$, $-$, $++$**`expr`**, $--$**`expr`** και $!$
   Υπολογίζονται από δεξιά προς τα αριστερά.
3. **πολλαπλασιασμός, διαίρεση και υπόλοιπο:** $*$, $/$, ή $\%$
   Υπολογίζονται δεύτερα από αριστερά προς τα δεξιά.
4. **πρόσθεση, αφαίρεση:** $+$ ή $-$
   Εάν υπάρχουν πολλοί, υπολογίζονται από τα αριστερά προς τα δεξιά.
5. **Σχεσιακοί:** $<$, $>$, $<=$, $>=$
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
6. **Ισότητας:** $==$, $!=$
   Υπολογίζονται από τα αριστερά προς τα δεξιά.
7. **λογικό AND:** && Από αριστερά προς τα δεξιά.
8. **λογικό OR:** || Από αριστερά προς τα δεξιά.
9. **εκχώρησης:** $=$, $+=$, $-=$, $*=$, $/=$, $\%=$
   Από δεξιά προς τα αριστερά.

goto

Η γλώσσα C μας παρέχει και μία δομή ελέγχου η οποία οδηγεί σε ”spaghetti code”, δηλαδή σε κώδικα που είναι δύσκολος να διαβαστεί.

```
1  #include <stdio.h>
2
3  int main() {
4      int a = 1;
5
6      if (a == 0)
7          goto test0;
8      else
9          goto test1;
10
11 test0:
12     printf("a = 0\n");
13     return 0;
14
15 test1:
16     printf("a = 1\n");
17     return 0;
18
19 }
```

Ότι μπορούμε να υλοποιήσουμε με την χρήση του `goto` μπορούμε να το υλοποιήσουμε και χωρίς, με την χρήση των άλλων δομών ελέγχου.

Η ΧΡΗΣΗ ΤΟΥ `goto` ΑΠΑΓΟΡΕΥΤΑΙ

http://xkcd.com/292

Για περισσότερες πληροφορίες διαβάστε εδώ.