Τμήμα Μηχανικών Η/Υ & Πληροφορικής
Πανεπιστήμιο Πατρών

Εισαγωγή στον Προγραμματισμό

Η γλώσσα προγραμματισμού C

Τελεστές και Τελεστέοι

Τελεστές και Τελεστέοι

- **Τελεστής** (**operator**)
  - Σύμβολο που αναπαριστά μία συγκεκριμένη στοιχειώδη διεργασία η οποία ενεργεί πάνω σε κάποια δεδομένα και παράγει κάποιο αποτέλεσμα
  - Κοινώς: μια «πράξη» (+, -, *, /)
  - Αλλά και μια σύγκριση (<, <=, == , >=, >), μια ανάθεση (=)
  - Και πολλά άλλα

- **Τελεστέος** (**operand**)
  - Τα δεδομένα (σταθερές, μεταβλητές, κλήσεις συναρτήσεων)
  - Π.χ., το **α** και το **β** στην έκφραση **α+β**

3

Κατηγορίες Τελεστών

- Αριθμητικοί
  - +, -, *, /, %
- Λογικοί
  - &&, ||, !
- Σύγκρισης ή Συσχετιστικοί
  - >, >=, ==, !=, <, <=
- Διαχείρισης ψηφίων
  - >>, <<, &, |, ^, ~
- Διαχείρισης μνήμης
  - &, [], ., ->
- Ειδικοί
  - ++, --, +=, -=, *=, /=, %=, &=, |=, <<=, >>=

Τελεστές διαχείρισης ψηφίων

- & AND ανά bit
- | OR ανά bit
- ^ αποκλειστικό OR ανά bit
- << ολίσθηση αριστερά ανά bit
- >> ολίσθηση δεξιά ανά bit
- ~ συμπλήρωμα ως προς 1 (δηλ, αντιστροφή ανά bit)

- Ο τελεστής AND (&) χρησιμοποιείται και για εφαρμογή μάσκας σε ένα σύνολο από bits, ενώ ο τελεστής OR (|) για να θέσει ένα σύνολο bits σε 1. Π.χ.:
  - η εντολή **k=k & 7** θέτει σε 0 όλα τα bit του **k** εκτός από τα 3 δεξιότερα, τα οποία αφήνει ανέπαφα
  - η εντολή **k=k | 7** θέτει τα 3 δεξιότερα bit του **k** σε 1, αφήνοντας ανέπαφα όλα τα υπόλοιπα

Παράδειγμα

```
#include <stdio.h>
main(){
 unsigned int a = 61; /* 61 = 00000000 00000000 00000000 00111101 */
 unsigned int b = 13; /* 13 = 00000000 00000000 00000000 00001101 */
 int c = 0;
 c = a & b; /* 13 = 00000000 00000000 00000000 00001101 */
 printf("c = a & b = %d\n", c );
 c = a | b; /* 61 = 00000000 00000000 00000000 00111101 */
 printf("c = a | b = %d\n", c );
 c = a ^ b; /* 48 = 00000000 00000000 00000000 00110000 */
 printf("c = a ^ b = %d\n", c );
 c = ~a; /*= 11111111 11111111 11111111 11000010 */
 printf("c = ~a = %d\n", c );
 c = a << ; /* 248 = 00000000 00000000 00000000 11110100 */
 printf("c = a << 2 = %d\n", c );
 c = a >> 2; /* 15 = 00000000 00000000 00000000 00001111 */
 printf("c = a >> 2 = %d\n", c );
}
```

Ειδικοί Τελεστές

- Μοναδιαίοι τελεστές αύξησης/μείωσης (++, --)
  - **++x** αυξάνει το x κατά ένα, και επιστρέφει την αυξημένη τιμή
  - **x++** αυξάνει το x κατά ένα, και επιστρέφει την τιμή προ της αύξησης

- Σύνθετοι τελεστές ανάθεσης (+=, -=, *=, /=)
  - **x += 10;** ισοδύναμο με το **x = x + 10;**
  - **x *= y+1;** ισοδύναμο με το **x = x*(y+1);**
  - **x <<= 2;** ισοδύναμο με το **x = x << 2;** (αριστ. ολίσθ. κατά 2 ψηφία)

- Τριαδικός τελεστής (ternary operator) (? :)
  - <expr1> ? <expr2> : <expr3>
  - **a > b ? a : b** "Αν a>b, τότε a, αλλιώς b", κοινώς max(a,b)
  - **printf("Το x %s μηδέν\n" , x==0 ? "είναι" : "δεν είναι");**

Παράδειγμα - Άσκηση

- Να βρεθούν οι τιμές των x και y μετά την εκτέλεση καθεμιάς από τις παρακάτω εντολές, ξεκινώντας **πάντα** με τις αρχικές τιμές
  - int x=10;
  - int y=20;

Εντολή	x	y
`++x;`	11	20
`y = x--;`	9	10
`y = --x;`	9	9
`y *= x--;`	9	200
`y += ++x;`	11	31
`y %= --x;`	9	2
`x *= 5`	50	20
`y = x *= 5`	50	50

Εκφράσεις

Εκφράσεις

- Έκφραση
  - Συνδυασμός ενός ή περισσότερων τελεστέων και ενός ή περισσότερων τελεστών
  - π.χ., **x = a + b**

- Σημειογραφία Εκφράσεων
  - **Infix notation** (ενδοθεματική): **x + y**
  - **Prefix notation** (προθεματική): **+ x y**
  - **Postfix notation** (μεταθεματική): **x y +**

Κατηγορίες Εκφράσεων

- **Αριθμητικές**
  - **Τελεστές:** αριθμητικοί
  - **Τελεστέοι:** αριθμητικές τιμές, μεταβλητές, άλλες εκφράσεις
  - **Αποτέλεσμα:** αριθμητικό
  - Παράδειγμα: (5*x+y/4)*8
- **Συγκριτικές**
  - **Τελεστές:** σύγκρισης
  - **Τελεστέοι:** αριθμητικές τιμές, μεταβλητές, άλλες εκφράσεις
  - **Αποτέλεσμα:** λογικού τύπου, ουσιαστικά int με τιμή 0 ή 1
  - Παραδείγματα: x>5, a!=b, x==3, x+y >= 4, x-y > x+z
- **Λογικές**
  - **Τελεστές:** λογικοί
  - **Τελεστέοι:** παραστάσεις με τιμές λογικές
  - **Αποτέλεσμα:** λογικού τύπου (0 ή 1)
  - Παραδείγματα: (x<5) && (x>=1), (x==0) || (y==0)

Υπολογισμός Εκφράσεων [1/3]

Η **εφαρμοστική σειρά** (**applicative order**) υπολογισμού των εκφράσεων βασίζεται σε δύο κανόνες:

1. **Προτεραιότητα**
   - Πρώτα οι τελεστές ταξινομούνται ανά επίπεδο προτεραιότητας. Π.χ.:
     - στο **5+3*8**, το * έχει προφανώς υψηλότερη προτεραιότητα από το +

2. **Προσεταιριστικότητα**
   - Μετά, καθορίζεται η κατεύθυνσης εφαρμογής μεταξύ των τελεστών ίδιας προτεραιότητας. Π.χ.:
     - στο **a+b+c** θα υπολογιστεί πρώτα το **a+b** και στη συνέχεια το **(a+b)+c** (προσεταιριστικότητα από αριστερά προς δεξιά)
     - στο **a=b=c** θα υπολογιστεί πρώτα το **b=c** και στη συνέχεια το **a=(b=c)** (προσεταιριστικότητα από δεξιά προς αριστερά)

Υπολογισμός Εκφράσεων [2/3]

ΤΕΛΕΣΤΕΣ	ΠΡΟΣΕΤΑΙΡΙΣΤΙΚΟΤΗΤΑ
() [ ] ->	Από αριστερά προς τα δεξιά
! ~ ++ -- - $*_{(pointer)}$ & (τύπος) sizeof	Από δεξιά προς τα αριστερά
$*_{(πολλαπλασιασμός)}$ / %	Από αριστερά προς τα δεξιά
+ -	Από αριστερά προς τα δεξιά
<< >>	Από αριστερά προς τα δεξιά
< <= > >=	Από αριστερά προς τα δεξιά
== !=	Από αριστερά προς τα δεξιά
&	Από αριστερά προς τα δεξιά
^	Από αριστερά προς τα δεξιά
|	Από αριστερά προς τα δεξιά
&&	Από αριστερά προς τα δεξιά
||	Από αριστερά προς τα δεξιά
?:	Από δεξιά προ τα αριστερά
= += -= *= %= &= ^= |= <<= >>=	Από δεξιά προς τα αριστερά
, (κόμμα)	Από αριστερά προς δεξιά

Από: Η ΓΛΩΣΣΑ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ C KERNIGHAN W. BRIAN, RITCHIE M. DENNIS, εκδόσεις Κλειδάριθμος, δεύτερη έκδοση

Υπολογισμός Εκφράσεων [3/3]

- Στη C ακολουθείται **υπολογισμός περιορισμένης έκτασης** (**short circuit evaluation**)
  - Δηλαδή όταν η τιμή μια έκφρασης είναι «προδικασμένη» δεν εκτελούνται όλοι οι υπολογισμοί μέχρι τέλους
  - Υπολογίζονται μόνο όσοι τελεστές απαιτούνται
  - Π.χ., στο **x && y**, αν το **x** βγει FALSE δεν θα γίνει ο υπολογισμός του **y**
  - Αντίστοιχα στο **x || y** αν το **x** βγει TRUE, μιας και το αποτέλεσμα θα είναι ούτως ή άλλως TRUE ανεξάρτητα από το y. Αν βέβαια το x βγει FALSE, θα υπολογιστεί και το **y** που θα κρίνει την τιμή της έκφρασης.

- Προσοχή: Πιθανές παρενέργειες!!
  - Το δεύτερο (ή τρίτο, κ.ο.κ.) σκέλος μιας λογικής έκφρασης μπορεί και να μην εκτελεστεί
  - Αν κάνει κάποια αλλαγή ή ανάθεση τιμής, *ίσως* να διαπιστώσουμε συμπεριφορά και αποτελέσματα που δεν περιμέναμε
  - π.χ. έστω ότι θέλουμε να μειώσουμε και το **x** και το **y** κατά 1, και να ελέγξουμε αν έχουν γίνει και τα δύο 0
    - Τότε, η έκφραση **(--x == 0) && (--y == 0)** περιλαμβάνει ένα καλά κρυμμένο ΛΑΘΟΣ. Γιατί;;;;;;
    - Διότι το **y** θα μειωθεί μόνο αν το **x** μειούμενο γίνει 0, οπότε το αριστερό σκέλος υπολογίζεται σε TRUE, και έτσι η υπο-έκφραση μετά το && πρέπει να υπολογιστεί. Αλλιώς το δεξιό σκέλος θα παρακάμπτεται και το **y** θα παραμένει σταθερό.

- Γενικά, δεν εκτελείται η έκφραση expr στα παρακάτω σενάρια:
  - **FALSE && expr**
  - **TRUE || expr**