Διάλεξη 4 - Live Coding, Git και Τελεστές

Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Εισαγωγή στον Προγραμματισμό

Θανάσης Αυγερινός / Τάκης Σταματόπουλος

Ανακοινώσεις / Διευκρινίσεις

- Η Εργασία θα βγει εντός της ημέρας 

Την Προηγούμενη Φορά

- Τύπωμα μεταβλητών continued
- Συνέχεια του Hello World που αφήσαμε την άλλη φορά
- Συναρτήσεις και Ακέραιοι (Μαθηματικά vs C)

Σήμερα

- Pair Programming
- Git
- Τελεστές

Pair Programming

Pair programming is a software development technique in which two programmers work together at one workstation. One, the *driver*, writes code while the other, the *observer* or *navigator*, reviews each line of code as it is typed in. The two programmers switch roles frequently.

You need to find someone that you're gonna pair-program with who's compatible with your way of thinking, so that the two of you together are a complementary force.

Jeff Dean

Χρειάζομαι 2-3 εθελοντές/ντριες

...

Υπολογισμός Βαθμολογίας Πρωτοετών

- 50% * Τελική Εξέταση + 30% * Ασκήσεις + 20% * Εργαστήριο
- Έστω ότι χρησιμοποιούμε 3 ακεραίους για να αναπαραστήσουμε τα αποτελέσματα [0 - 100].
  - int final_exam
  - int homework
  - int lab
- Στα μαθηματικά:
  - grade(final_exam, homework, lab) = final_exam x 50% + homework x 30% + lab x 20%
- Έλεγχος ορθότητας: grade(70, 80, 100) == 79?
- Σε C;

Κανόνας Αναπροσαρμογής: Αν ο βαθμός από τις Ασκήσεις είναι πάνω από 3 μονάδες μεγαλύτερος του βαθμού από την Τελική Εξέταση, τότε ο βαθμός για τις Ασκήσεις αναπροσαρμόζεται σε Τελική Εξέταση + 3. Πως θα το εκφράσουμε;

Πρόγραμμα Υπολογισμού Βαθμολογίας 1/2

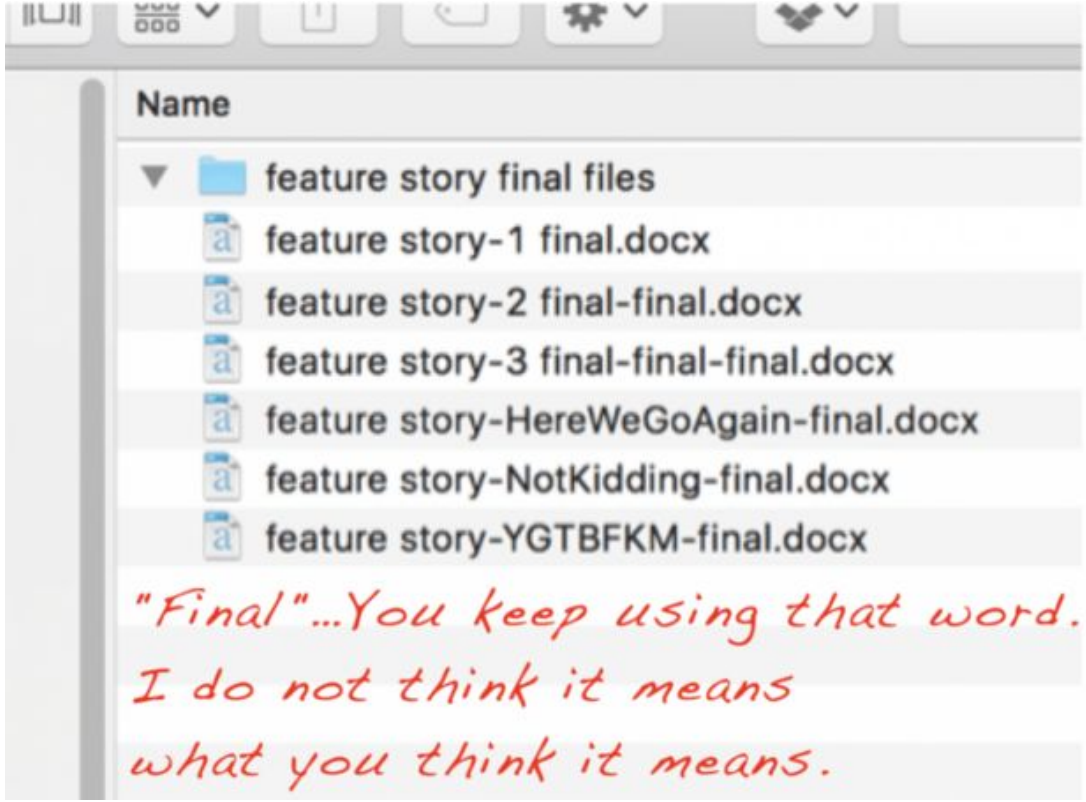
```c
#include <stdio.h>

#include <stdlib.h>

// The grade function computes the final grade for first year students according to

// https://progintro.github.io

int grade(int final_exam, int homework, int lab){

  return final_exam*50/100+homework*30/100+lab*20/100;

}

int main(int argc, char ** argv) {

  if (argc != 4) {

              printf("Run as: grade final_exam homework lab\n");

              return 1;

  }

  int final_exam = atoi(argv[1]);

  int homework = atoi(argv[2]);

  int lab = atoi(argv[3]);

  int final_grade = grade(final_exam, homework, lab);

  printf("My grade is: %d\n", final_grade);

  return 0;

}
```

Μια περσινή λύση

Πρόγραμμα Υπολογισμού Βαθμολογίας 1/2

- Η συνάρτηση `main` έχει μεταβλητό αριθμό ορισμάτων. Η παράμετρος `argc` λέει πόσα ορίσματα περάσαμε στο πρόγραμμα, ενώ η παράμετρος `argv` περιέχει το κείμενο που αντιστοιχεί σε κάθε όρισμα.
    - `argv[1]` περιέχει το 1ο όρισμα του προγράμματος, `argv[2]` το 2ο, κ.ο.κ.
- Η συνάρτηση βιβλιοθήκης (`stdlib.h`) `atoi` μετατρέπει ένα όρισμα από αλφαριθμητικό (ASCII) σε έναν ακέραιο (integer).
- Εάν σας ενοχλεί το πόσο "μαγική" φαίνεται η `main` και τα ορίσματά της, μην εξάπτεστε! Είναι μια από τις δυσκολότερες συναρτήσεις που θα συναντήσουμε στην C και τις επόμενες εβδομάδες θα χτίσουμε το υπόβαθρο για να γίνει πιο κατανοητή.

Version Control & Git



Version Control

- Ένα σύστημα που επιτρέπει να παρακολουθούμε όλες τις αλλαγές που γίνονται στον κώδικα και να κρατάμε ιστορικό αρχείο.

Γιατί να θέλουμε κάτι τέτοιο;;

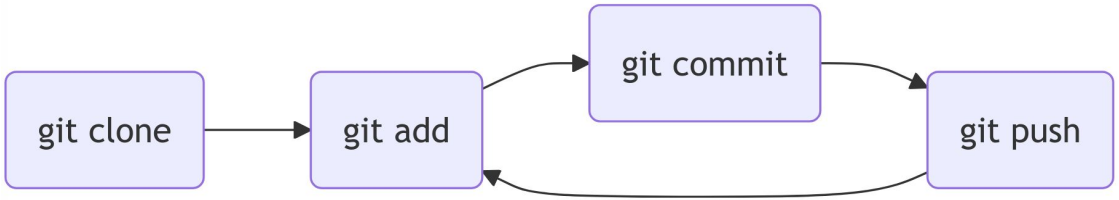
- Τα περισσότερα προγράμματα αλλάζουν με τον χρόνο - δεν υπάρχει "τελική μορφή"
- Κάνουμε αλλαγές και κρατάμε μόνο τα αρχεία που χρειαζόμαστε χωρίς να χάνουμε πληροφορία
- Μπορούμε να ανατρέξουμε σε οποιαδήποτε αλλαγή έγινε ανά πάσα στιγμή

Git

- Είναι το δημοφιλέστερο πρόγραμμα Version Control σήμερα (~90% των χρηστών/project)
- Πρώτη έκδοση γράφτηκε από τον Linus Torvalds το 2005
- Μια δημοφιλής και δωρεάν (για την ώρα) πλατφόρμα στην οποία μπορούμε να αποθηκεύσουμε τον κώδικά μας είναι το https://github.com/
- Θα το χρησιμοποιήσουμε για την γραφή και υποβολή των εργασιών μας

Git Development Cycle

Σε επίπεδο βασικής χρήσης (αυτό που χρειαζόμαστε για το μάθημα), η γραφή Κώδικα και η αποθήκευση με git περιλαμβάνει 4 βήματα:



Clone Repository

`git clone`: Αντιγράφει το repository (αποθετήριο) με όλα τα αρχεία σε έναν τοπικό φάκελο.

```
git clone git@github.com:progintro/hw0-barbouni-2005.git
```

- Αν ελέγξουμε τον φάκελο παρατηρούμε ότι έχει ένα README.md αρχείο.
- Τρέχοντας `git status` μέσα στον φάκελο μπορούμε να δούμε αν υπάρχουν καθόλου αλλαγές.
- Η παραπάνω εντολή θα δημιουργήσει έναν τοπικό φάκελο (αν δεν υπάρχει) με το όνομα hw0-barbouni-2005
- Τα βήματα που ακολουθούμε εδώ, προϋποθέτουν ότι έχετε φτιάξει ένα κλειδί για τον Github λογαριασμό σας και έχετε τελειώσει configuration όπως περιγράφεται στο Φυλλάδιο 1 του εργαστηρίου.

Προσθήκη Αρχείου

`git add`: Προσθέτει τα αρχεία στο staging area (προσωρινή λίστα) για μελλοντική αποθήκευση στο repository.

- Δημιούργησε ένα καινούριο αρχείο, π.χ., `touch command/solution.txt`.
- Τρέξε `git status` - θα παρατηρήσεις ότι το καινούριο αρχείο καταγράφεται ως untracked.
- Για να το προσθέσουμε: `git add command/solution.txt`
- Τρέξε `git status` ξανά - θα παρατηρήσεις ότι το αρχείο είναι έτοιμο να γίνει commit (καταχωρηθεί).

Καταχώρηση Αλλαγών

`git commit`: Αποθηκεύει τις αλλαγές στο τοπικό repository.

- Τρέξε `git commit` και δώσε ένα μήνυμα που περιγράφει την αλλαγή σου.
- Τρέξε `git status` και πάλι - τώρα το μόνο που μένει είναι να σώσουμε τις αλλαγές μας και εκτός του υπολογιστή μας.

Ανέβασμα Αλλαγών

`git push`: Ανεβάζει τις τοπικές αλλαγές στον απομακρυσμένο server (εξυπηρετητή).

- Έλεγξε ότι οι αλλαγές σου εμφανίζονται στο Github!

Κατέβασμα Αλλαγών

`git pull`: Κατεβάζει αλλαγές από τον απομακρυσμένο server (αν υπάρχουν).

- Χρήσιμο όταν έχεις πολλούς προγραμματιστές ή πολλούς υπολογιστές και δεν θέλεις να κάνεις clone κάθε φορά.
- Κάνε κάποιες αλλαγές στο repository από το GitHub UI και μετά τρέξε `git pull`.

Don't forget to add your files before committing & pushing!

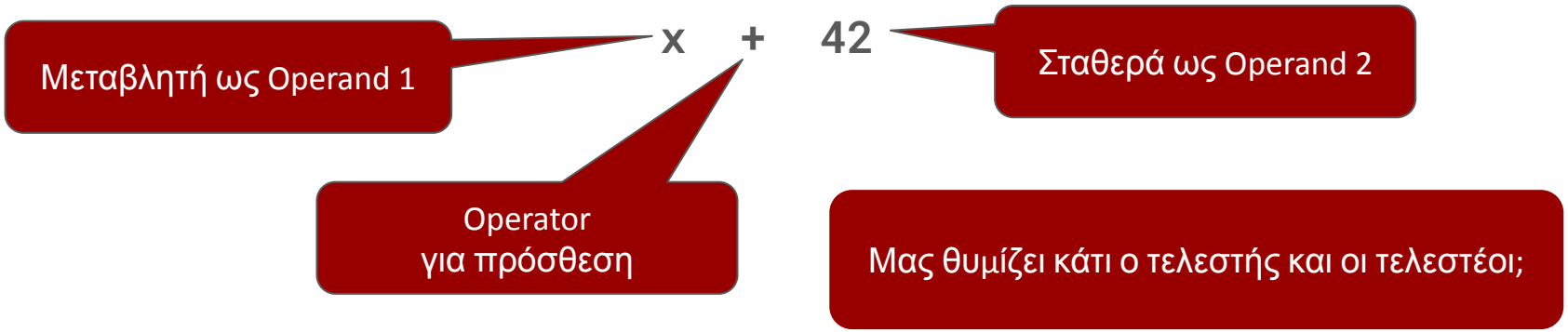
Τελεστές (Operators)

- Μοναδιαίοι (unary), Δυαδικοί (binary), Τριαδικοί (ternary)
- Αριθμητικοί, Συγκριτικοί, Λογικοί, Συνθήκης, Bitwise, Μετατροπής, Ανάθεσης
- Προτεραιότητα & Προσεταιριστικότητα

Τελεστής (Operator) και Τελεστέος (Operand)

Οι τελεστές στην C είναι σύμβολα που πραγματοποιούν μαθηματικούς, συγκριτικούς, δυαδικούς, υποθετικούς ή λογικούς υπολογισμούς.

Οι τελεστέοι στην C είναι τα αντικείμενα (μεταβλητές, σταθερές) πάνω στα οποία πραγματοποιούν τους υπολογισμούς οι τελεστές.



Πρόγραμμα - Συνάρτηση - Τελεστές

Το πρόγραμμα παίρνει κάποια δεδομένα εισόδου και υπολογίζει μια έξοδο

Το πρόγραμμα αποτελείται από συναρτήσεις που παίρνουν ορίσματα και υπολογίζουν μια έξοδο

Οι συναρτήσεις χρησιμοποιούν τελεστές και τελεστέους προκειμένου να υπολογίσουν μια έξοδο

It's turtles all the way down!

Κατηγοριοποίηση Τελεστών #1

Κατηγοριοποίηση με βάση το arity - πόσους τελεστέους έχουν:

- Μοναδιαίοι Τελεστές (Unary Operators): Ένα τελεστέο (operand)
- Δυαδικοί Τελεστές (Binary Operators): Δύο τελεστέους
- Τριαδικοί Τελεστές (Ternary Operators): Τρεις τελεστέους

Στην συνέχεια θα δούμε κατηγοριοποίηση με βάση την χρήση του τελεστή και στην σειρά υπολογισμού

Παραδείγματα Τελεστών

Αριθμητικοί Τελεστές

Αριθμητικός Τελεστής	Ερμηνεία	Παραδείγματα
+	πρόσθεση	40 + 2 επιστρέφει 42 40.0 + 2.0 επιστρέφει 42.0
−	αφαίρεση	44 − 2 επιστρέφει 42 44.0 − 2.0 επιστρέφει 42.0
*	πολλαπλασιασμός	42 * 2 επιστρέφει 84 42.0 * 2.0 επιστρέφει 84.0
/	διαίρεση	85.0 / 2.0 επιστρέφει 42.5 85 / 2 επιστρέφει 42
%	υπόλοιπο	7 % 2 επιστρέφει 1

1. Σε τι κατηγορία θα εντάσσατε τους αριθμητικούς τελεστές;
2. Τι τύπο θα είχαν οι τελεστές αν ήταν συναρτήσεις;

Συγκριτικούς Τελεστές (Comparison Operators)

Οι συγκριτικοί τελεστές επιτρέπουν την σύγκριση τελεστέων. Το αποτέλεσμα είναι 0 (ψευδές) ή 1 (αληθές). Τι τύπου τελεστές είναι;

Σχεσιακός Τελεστής	Ερμηνεία	Παραδείγματα
==	Ίσοι τελεστέοι;	42 == 0 επιστρέφει 0 42 == 42 επιστρέφει 1
!=	Διαφορετικοί τελεστέοι;	42 != 0 επιστρέφει 1 42 != 42 επιστρέφει 0
>	Αριστερός τελεστέος μεγαλύτερος του δεξιού;	42 > 0 επιστρέφει 1 42 > 42 επιστρέφει 0
<	Αριστερός τελεστέος μικρότερος του δεξιού;	42 < 0 επιστρέφει 0 42 < 44 επιστρέφει 1
>=	Αριστερός τελεστέος μεγαλύτερος ή ίσος του δεξιού;	42 >= 42 επιστρέφει 1 42 >= 43 επιστρέφει 0
<=	Αριστερός τελεστέος μικρότερος ή ίσος του δεξιού;	42 <= 42 επιστρέφει 1 42 <= 41 επιστρέφει 0

Λογικοί Τελεστές (Logical Operators)

Οι λογικοί τελεστές επιτρέπουν την αποτίμηση συνδυασμού αληθών και ψευδών τιμών.
Τι τύπου τελεστές είναι;

Λογικός Τελεστής	Ερμηνεία	Παραδείγματα
&&	Λογική σύζευξη (AND, $\wedge$) είναι και οι δύο τελεστέοι αληθείς;	42 > 0 && 2 > 3 επιστρέφει 0 42 > 0 && 3 > 2 επιστρέφει 1
||	Λογική διάζευξη (OR, $\vee$) είναι ένας εκ των δύο τελεστέων αληθής;	2 > 3 || 42 > 0 επιστρέφει 1 2 > 3 || 42 < 0 επιστρέφει 0
!	Λογική άρνηση (NOT, ¬) είναι ο τελεστέος ψευδής;	!(42 < 0) επιστρέφει 1 !(42 > 0) επιστρέφει 0

Σημαντικό: για τους λογικούς τελεστές στην C, το μηδέν (0) σημαίνει “ψευδές”. Οποιαδήποτε άλλη μη μηδενική τιμή σημαίνει “αληθές”. Το 1, το 42 και το -5 είναι όλα εξίσου “αληθή”.

Bitwise Τελεστές (Bitwise Operators)

Οι bitwise τελεστές κάνουν μετατροπές στα *bit* των ακέραιων αριθμών. Τι τύπου είναι οι τελεστές;

Bitwise Τελεστής	**Ερμηνεία**	**Παραδείγματα**
&	Σύζευξη bit (bitwise AND)	0xFF & 0xF0 επιστρέφει 0xF0 0xFF & 0x00 επιστρέφει 0x00
|	Διάζευξη bit (bitwise OR)	0xFF | 0xF0 επιστρέφει 0xFF 0xF0 | 0x0F επιστρέφει 0xFF
^	Αποκλειστική διάζευξη bit (bitwise XOR)	0xFF ^ 0xFF επιστρέφει 0x00 0x00 ^ 0xFF επιστρέφει 0xFF
~	Άρνηση bit (bitwise NOT)	~0xF0 επιστρέφει 0x0F ~0x05 επιστρέφει 0xFA
<<	Ολίσθηση bit αριστερά (shift left). Shift N bit ισοδύναμο με πολλαπλασιασμό με 2^N	2 << 4 επιστρέφει 32 4 << 1 επιστρέφει 8
>>	Ολίσθηση bit δεξιά (shift right). Shift N bit ισοδύναμο με διαίρεση με 2^N	8 >> 1 επιστρέφει 4 32 >> 4 επιστρέφει 2

Πως απομονώνουμε το πιο σημαντικό bit σε ένα byte;

A. byte & 0xFF

B. byte & 0x80

C .byte | 0xFF

D. byte | 0x80

Ποια η τιμή της παράστασης 0xbeef | 0xcafe0000;

A. 0xffffffff

B. 0xcafebeef

C. 0xbeefcafe

D. 0xfaceb00c

Τελεστής Συνθήκης (Conditional Operator)

Ο τελεστής συνθήκης ελέγχει την συνθήκη και αν είναι αληθής επιστρέφει την πρώτη τιμή, αλλιώς επιστρέφει την δεύτερη. Γενική μορφή:

```
συνθήκη ? τιμή1 : τιμή2
```

Για παράδειγμα:

```
(42 > 0) ? 8 : 16  => επιστρέφει 8
```

Αντίστοιχα:

```
(42 == 0) ? 8 : 16  => επιστρέφει 16
```

Τι τύπου είναι ο τελεστής συνθήκης; Πως θα φτιάχναμε μια συνάρτηση max;

Τελεστής Μετατροπής (Cast Operator)

Ο τελεστής μετατροπής αλλάζει τον τύπο ενός τελεστέου. Γενική μορφή:

```
(τύπος)τελεστέος
```

Παραδείγματα:

```
(int)42.67 επιστρέφει 42
(char)67.8 επιστρέφει 'C'
(double)3 επιστρέφει 3.0
(double)3/4 επιστρέφει ??
```

Ψηφία μετά την υποδιαστολή αποκόπτονται, δεν στρογγυλοποιούνται

Η παραπάνω μετατροπή λέγεται και explicit type conversion

Σιωπηρή Μετατροπή Τύπων (Implicit Type Conversion)

Σε περιπτώσεις μίξης τύπων σε έναν τελεστή, ο μεταγλωττιστής προσθέτει αυτόματα μετατροπές στον μεγαλύτερο των τελεστέων.

Παράδειγμα:

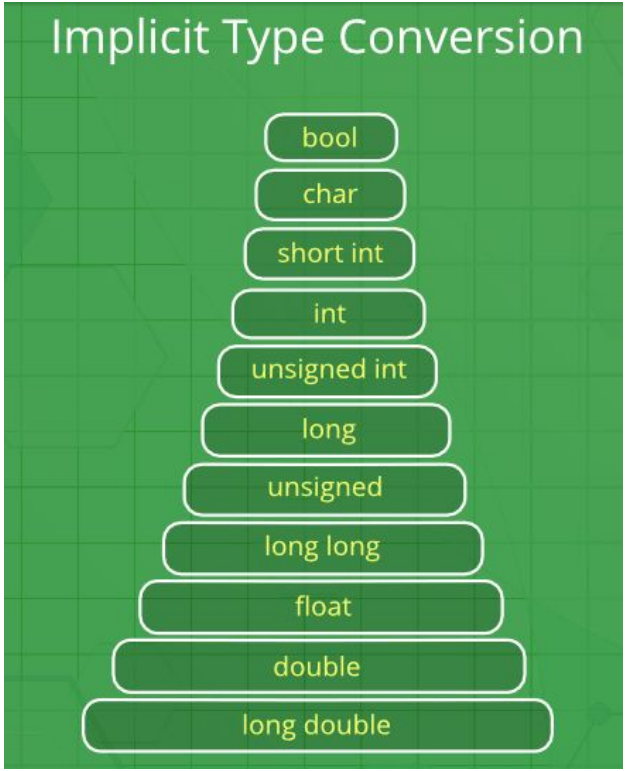
```
40 + 2.0
```

Είναι ισοδύναμο με:

```
(double)40 + 2.0
```

Ισοδύναμο με:

```
40.0 + 2.0
```



Τελεστής Ανάθεσης (Assignment Operator)

Ο τελεστής ανάθεσης παίρνει την τιμή του δεξιού τελεστέου (rvalue), την αναθέτει στην μεταβλητή του αριστερού τελεστέου (lvalue) και την επιστρέφει

Παράδειγμα:

```
x = 42 // αναθέτει 42 στην x και επιστρέφει 42
```

Μπορούμε να αναθέσουμε πολλές μεταβλητές ταυτόχρονα:

```
x = y = 42
```

Ή ισοδύναμα:

```
x = (y = 42)
```

Συνδυαστικοί Τελεστές Ανάθεσης

Οι συνδυαστικοί τελεστές ανάθεσης επιτρέπουν να κάνουμε αριθμητικές πράξεις και να σώσουμε το αποτέλεσμα με συντομογραφία. Γενική μορφή:

```
τελεστέος1 op= τελεστέος2
```

Όπου op είναι οποιοσδήποτε από τους τελεστές +, -, *, %, /, &, ^, |, <<, >>

Παράδειγμα:

```
a += b
```

Είναι ισοδύναμο με:

```
a = a + b
```

Αντίστοιχα, **a *= b** είναι ισοδύναμο με **a = a * b** και ομοίως για τους άλλους τελεστές

Τελεστές Αύξησης και Μείωσης

Ο τελεστής αύξησης **++** μπαίνει πριν ή μετά από το όνομα μίας μεταβλητής και την αυξάνει κατά 1. Ο τελεστής μείωσης **--** μπαίνει πριν ή μετά από το όνομα μίας μεταβλητής και την μειώνει κατά 1.

Επιθεματικά:

```
a++  επιστρέφει την τιμή του a πριν την αύξηση
a--  επιστρέφει την τιμή του a πριν την μείωση
```

Προθεματικά:

```
++a  επιστρέφει την τιμή του a μετά την αύξηση
--a  επιστρέφει την τιμή του a μετά την μείωση
```

Τελεστής Παράθεσης (Comma Operator)

Ο τελεστής παράθεσης υπολογίζει τον πρώτο τελεστέο, στην συνέχεια αγνοεί το αποτέλεσμα και επιστρέφει τον δεύτερο τελεστέο.

Παράδειγμα:

```
12, 42 επιστρέφει 42
```

Σχετικά περιορισμένες χρήσεις συνήθως χρησιμοποιείται με τελεστές ανάθεσης.

Πρόγραμμα Υπολογισμού Βαθμολογίας

```c
int grade(int final_exam, int homework, int lab){
  return final_exam*50/100+homework*30/100+lab*20/100;
}
int main(int argc, char ** argv) {
  if (argc != 4) {
        printf("Run as: grade final_exam homework lab\n");
        return 1;
  }
  int final_exam = atoi(argv[1]);
  int homework = atoi(argv[2]);
  int lab = atoi(argv[3]);
  int final_grade = grade(final_exam, homework, lab);
  printf("My grade is: %d\n", final_grade);
  return 0;
}
```

Προτεραιότητα (Precedence) & Προσεταιριστικότητα (Associativity)

Η προτεραιότητα ενός τελεστή καθορίζει την σειρά με την οποία θα εκτελεστούν οι υπολογισμοί. Για παράδειγμα, ο πολλαπλασιασμός έχει υψηλότερη προτεραιότητα από την πρόσθεση:

`5 * 6 + 3 * 4 επιστρέφει ??`

Αν δύο τελεστές έχουν την ίδια προτεραιότητα, η προσεταιριστικότητα τους καθορίζει την σειρά των υπολογισμών (αριστερά προς τα δεξιά ή το αντίστροφο). Για παράδειγμα, ο πολλαπλασιασμός και η διαίρεση έχουν την ίδια προτεραιότητα, ενώ η προσεταιριστικότητα τους είναι από αριστερά προς τα δεξιά

`90 * 50 / 100 επιστρέφει ??`

Προτεραιότητα (Precedence) & Προσεταιριστικότητα (Associativity)

Η προτεραιότητα ενός τελεστή καθορίζει την σειρά με την οποία θα εκτελεστούν οι υπολογισμοί. Για παράδειγμα, ο πολλαπλασιασμός έχει υψηλότερη προτεραιότητα από την πρόσθεση:

```
5 * 6 + 3 * 4 επιστρέφει ??
```

Αν δύο τελεστές έχουν την ίδια προτεραιότητα, η προσεταιριστικότητα τους καθορίζει την σειρά των υπολογισμών (αριστερά προς τα δεξιά ή το αντίστροφο). Για παράδειγμα, ο πολλαπλασιασμός και η διαίρεση έχουν την ίδια προτεραιότητα, ενώ η προσεταιριστικότητα τους είναι από αριστερά προς τα δεξιά

```
90 * 50 / 100 επιστρέφει ??
```

Δεν είμαστε σίγουροι για την ακριβή σειρά; Χρησιμοποιούμε παρενθέσεις ()!

Θέση	Τελεστές	Προσεταιριστικότητα
1	() (παρενθέσεις-κλήση συνάρτησης) [] -> . ++ (επιθεματική αύξηση) -- (επιθεματική μείωση)	αριστερά προς δεξιά
2	++ (προθεματική αύξηση) -- (προθεματική μείωση) ! ~ * (έμμεση αναφορά) & (διεύθυνση) + (μοναδιαίο +) - (μοναδιαίο -) (προσαρμογή τύπου) sizeof	δεξιά προς αριστερά
3	* (πολλαπλασιασμός) / %	αριστερά προς δεξιά
4	+ (πρόσθεση) - (αφαίρεση)	αριστερά προς δεξιά
5	<< >>	αριστερά προς δεξιά
6	< <= > >=	αριστερά προς δεξιά
7	== !=	αριστερά προς δεξιά
8	&	αριστερά προς δεξιά
9	^	αριστερά προς δεξιά
10	|	αριστερά προς δεξιά
11	&&	αριστερά προς δεξιά
12	||	αριστερά προς δεξιά
13	?:	δεξιά προς αριστερά
14	= += -= *= /= %= &= ^= |= <<= >>=	δεξιά προς αριστερά
15	,	αριστερά προς δεξιά

Kahoot Time!

Για την Επόμενη Φορά

- Από τις σημειώσεις του κ. Σταματόπουλου συνιστώ να έχετε καλύψει τα πάντα μέχρι την σελίδα 43.
- Type conversion
- List of operators
- lvalues and rvalues
- Precedence and associativity
- Προαιρετικό: Git Cheat Sheet
- Προαιρετικό: Git Tutorial
- Jeff and Sanjay

Ευχαριστώ και καλή εβδομάδα εύχομαι!
Keep coding ;)