

Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Γ' Ανακεφαλαίωση

Πέμπτη, 2 Ιουνίου 2022

Παναγιώτης Παύλου

c-programming-22@allos.gr

Χρήση Debugger

Τι είναι ο debugger και ποιες είναι οι πιο κοινές λειτουργίες του

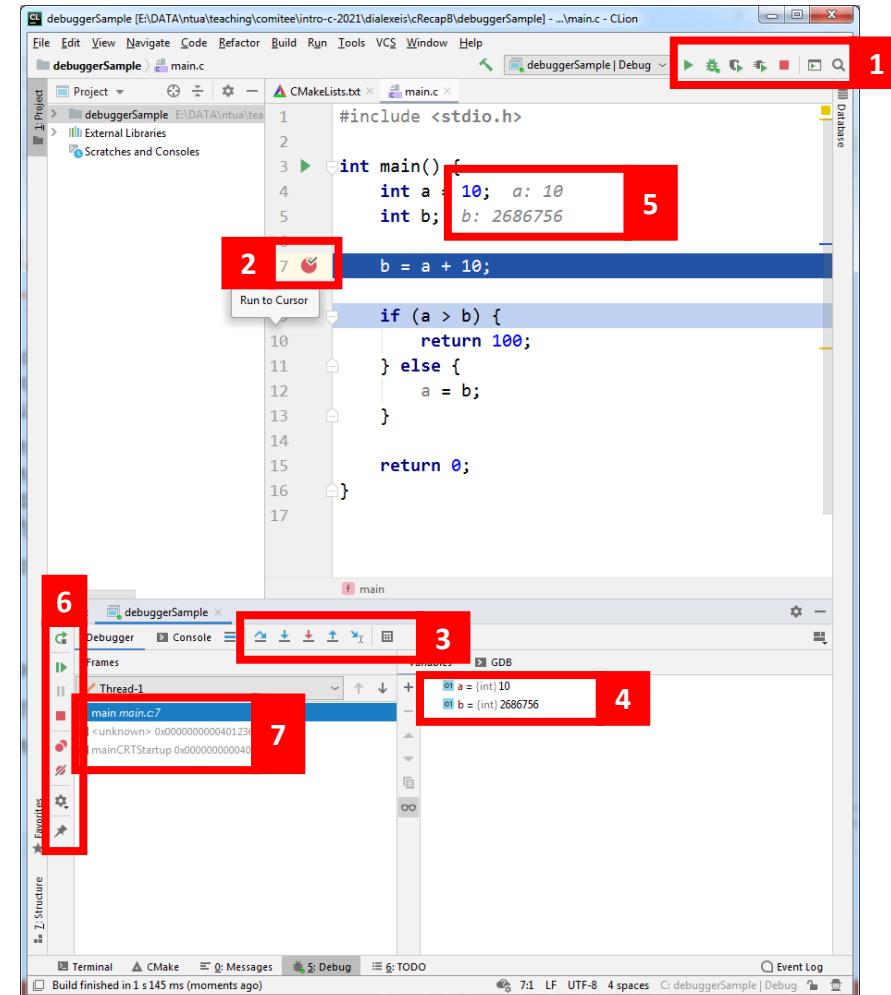
Τι είναι ο debugger

Ο debugger επιτρέπει την ελεγχόμενη εκτέλεση ενός κώδικα από τον προγραμματιστή προκειμένου να εντοπιστούν σφάλματα (bugs) του κώδικα.

Υπάρχουν διάφορα συστήματα για το debugging της C, με τα οποία συνεργάζεται το CLion και για τα οποία παρέχει στον χρήστη του μία ενιαία διεπαφή χρήστη.

Οι κύριες λειτουργίες που παρέχονται και θα παρουσιαστούν είναι:

- τα breakpoints
- ελεγχόμενη εκτέλεση εντολή προς εντολή
- παρακολούθηση τιμών μεταβλητών
- παρακολούθηση τιμών εκφράσεων



Έναρξη της διαδικασίας debugging

Η διαδικασία του debugging ξεκινά όταν ο χρήστης επιλέξει αντί του play, το αμέσως επόμενο (2^o) εικονίδιο.



Ο τερματισμός του, γίνεται με το ίδιο εικονίδιο που σταματά και η εκτέλεση (αυτό του stop – 5^o).



The screenshot shows the CLion IDE interface during a debug session. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, VCS, Window, Help. The toolbar has several icons, with the first four highlighted by a red box and labeled with the number '1'. The code editor displays a C file named main.c with the following content:

```
#include <stdio.h>
int main() {
    int a = 10; a: 10
    int b; b: 2686756
    b = a + 10;
    if (a > b) {
        return 100;
    } else {
        a = b;
    }
    return 0;
}
```

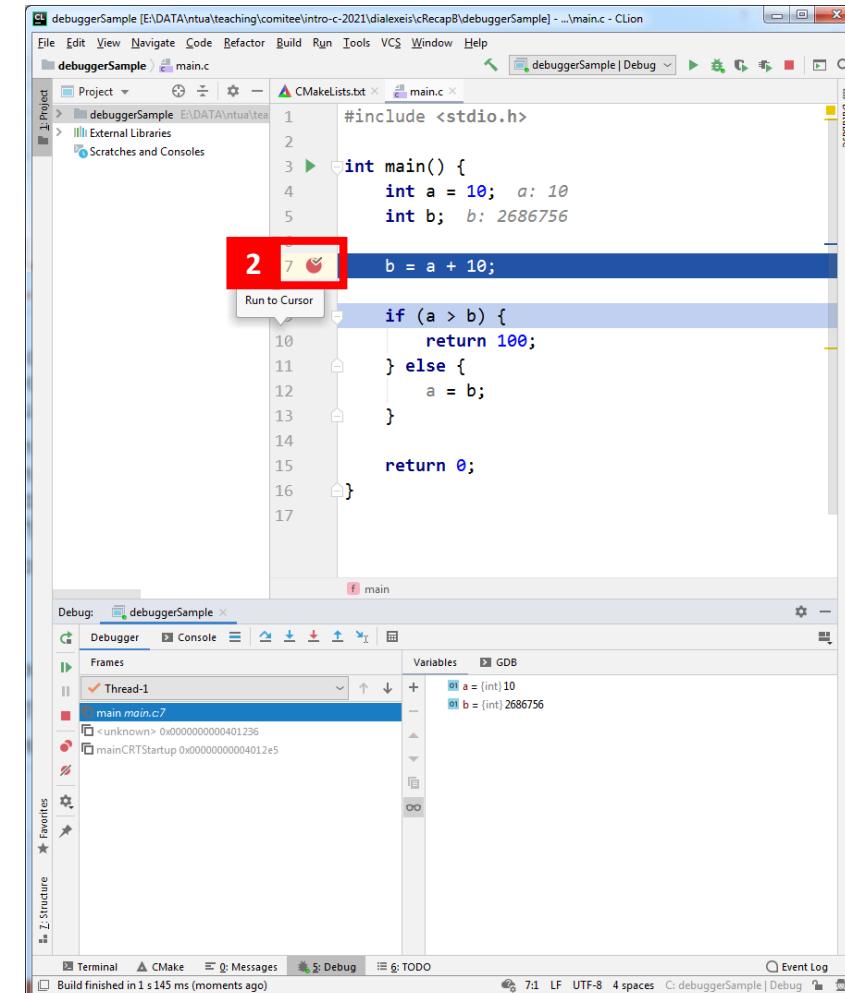
The debugger panel at the bottom shows the current thread is Thread-1, and the variables a and b are both set to 10. The status bar at the bottom indicates the build finished in 1 s 145 ms (moments ago), and the file C:debuggerSample | Debug is open.

Πως προσθέτω/αφαιρώ breakpoints

Ένα breakpoint (BP) εφαρμόζεται σε μία γραμμή που μας ενδιαφέρει να σταματήσει προσωρινά η εκτέλεση του κώδικα. Το προσθέτω ή το αφαιρώ κάνοντας κλικ στο σημείο που φαίνεται στην εικόνα , αριστερά από τη γραμμή που με ενδιαφέρει.

To break point μπορεί να ισχύει:

- πάντα
- υπό συνθήκες
- μία φορά
- μόνο αφού συναντηθεί άλλο BP



Εκτέλεση βήμα προς βήμα

Εφόσον έχει διακοπή η εκτέλεση του κώδικα εξ' αιτίας ενός BP, μπορεί να συνεχιστεί γραμμή προς γραμμή με τα εργαλεία της μπάρας (#3).



Εκτελεί την τρέχουσα γραμμή κώδικα χωρίς να εισέρχεται αναλυτικά στις εμπλεκόμενες συναρτήσεις.

Εκτελεί την τρέχουσα γραμμή κώδικα εκτελώντας αναλυτικά τις εμπλεκόμενες συναρτήσεις.

Απρόσκοπτη συνέχιση της εκτέλεσης της τρέχουσας συνάρτησης μέχρι την ολοκλήρωσή της.

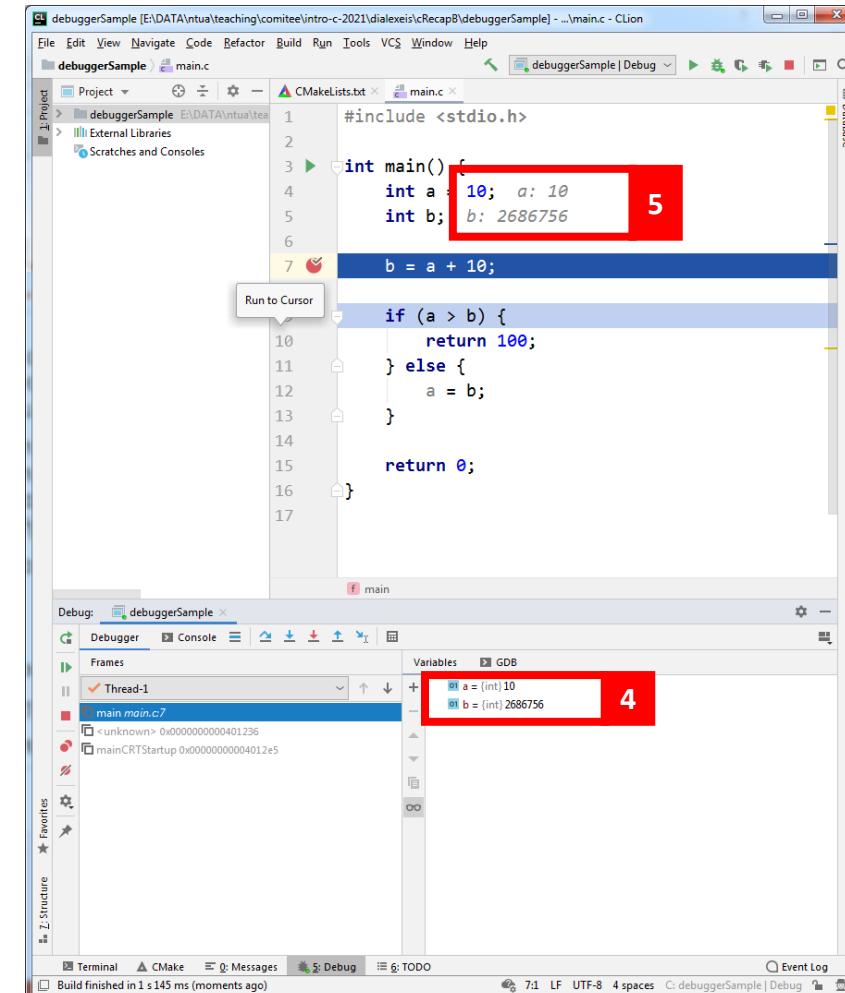
A screenshot of the CLion IDE showing the debugger interface. The code editor shows a C program with variable values for 'a' and 'b'. The debugger window shows a stack trace with 'Thread-1' and the current frame 'main main.c:7'. A red box highlights the third icon in the toolbar, labeled '3', which corresponds to the 'Step Into' function. The status bar at the bottom indicates the build was finished in 1 second.

Τιμές μεταβλητών

Όταν η εκτέλεση του κώδικα είναι σταματημένη σε κάποιο σημείο, οι τιμές των μεταβλητών εμφανίζονται σε δύο σημεία:

- Κάτω δεξιά όπου εμφανίζεται το όνομα και η τιμή των μεταβλητών (watch window)
- Μέσα στον κώδικα, δίπλα στην κάθε δήλωση της μεταβλητής με αχνά γκρι γράμματα

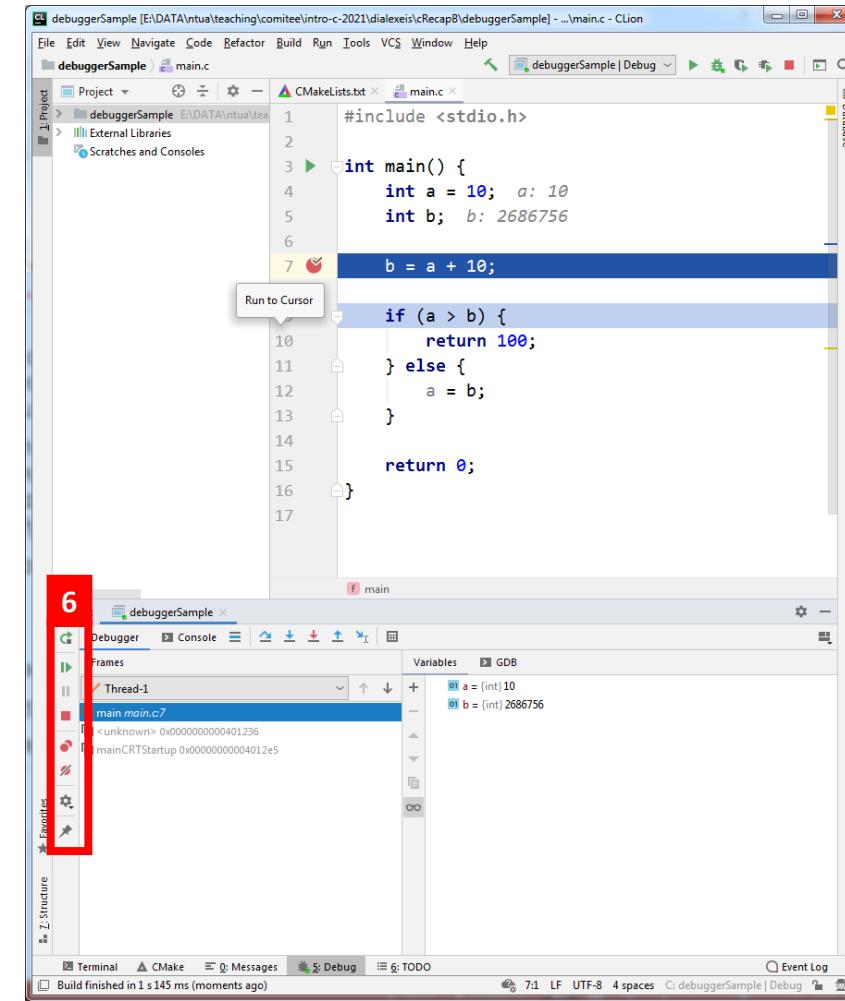
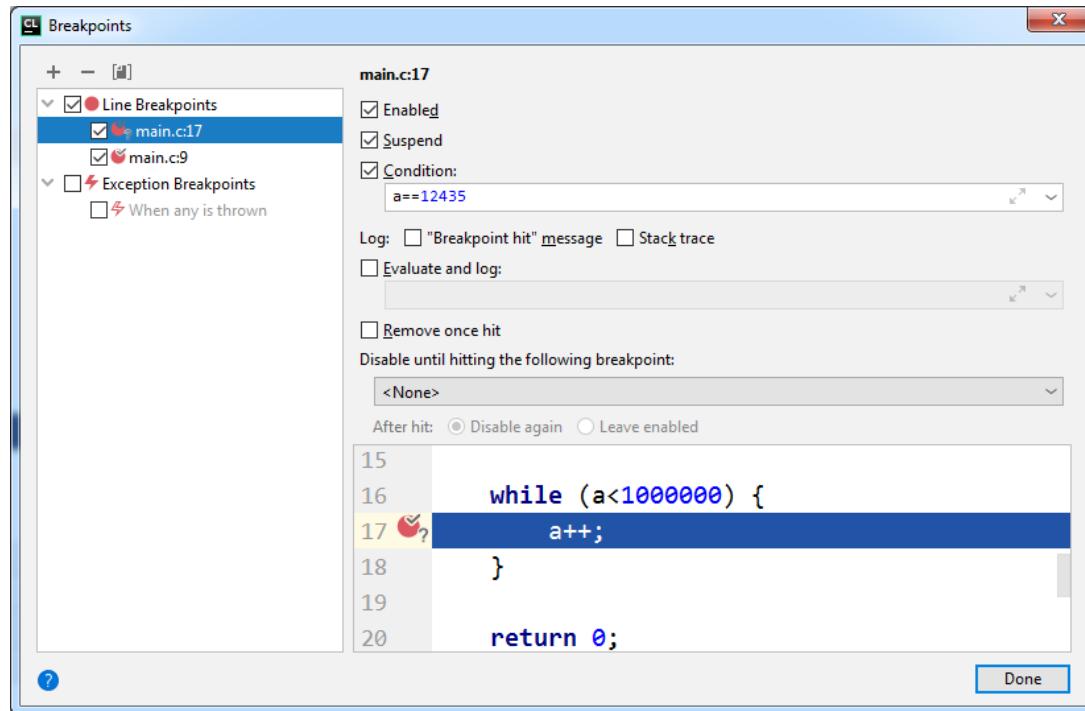
Εκτός από τις τοπικές μεταβλητές που εν γένει εμφανίζονται αυτόματα, μπορούμε να προσθέσουμε και άλλες μεταβλητές ή παραστάσεις με δεξιή κλικ στο watch window.



'Άλεις δυνατότητες

Διάφορες εργασίες και δυνατότητες παρέχονται στη μπάρα εργαλειών #6.

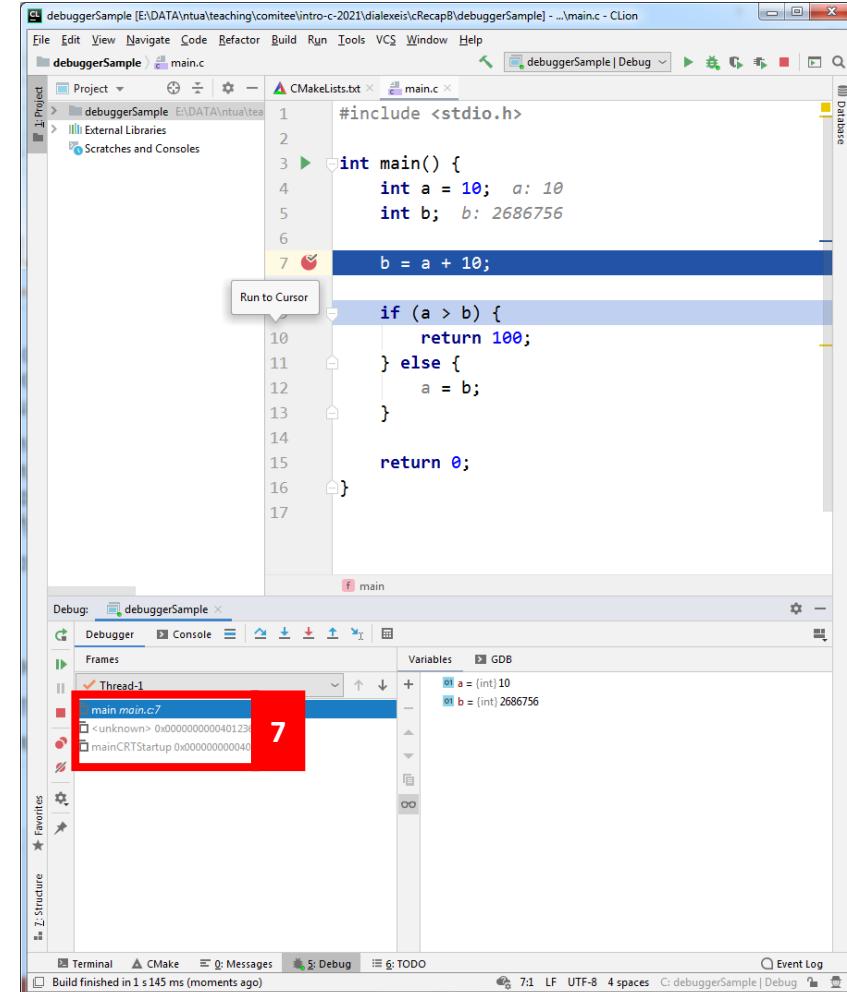
Αυτή που μας ενδιαφέρει για την παρουσίαση είναι η αναλυτική διαχείριση των BP.



To call stack

Όταν η εκτέλεση του κώδικα βρίσκεται μέσα σε κάποια συνάρτηση, την οποία έχει καλέσει κάποια άλλη, Κ.Ο.Κ., τότε στο call stack (#7) εμφανίζονται και οι συναρτήσεις, μέσω των οποίων έχει φτάσει η εκτέλεση στο τρέχον σημείο.

Έτσι δίνεται η δυνατότητα ο χρήστης να δει με ποιες συνθήκες έχει γίνει η εκτέλεση αυτών των συναρτήσεων, επιλέγοντας την κάθε συνάρτηση στον χώρο του call stack.



Εφαρμογή 1 - Infinite digits math

char *sumOfIntegers(char *A, char *B)

$$'a' = \underline{\underline{g_2}}$$

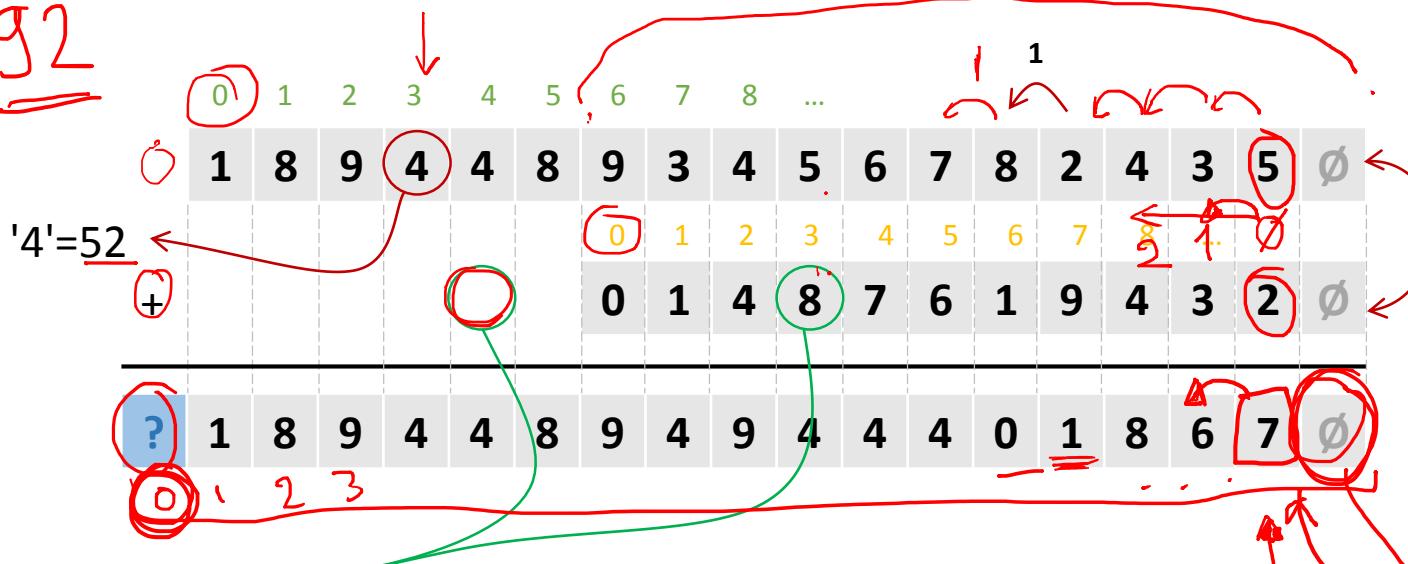
' ϕ ' \rightarrow 48

'1' → 49

'2' → 50

~~getDigitValue(char *x, int dig)~~

'a' 'b' 'c')



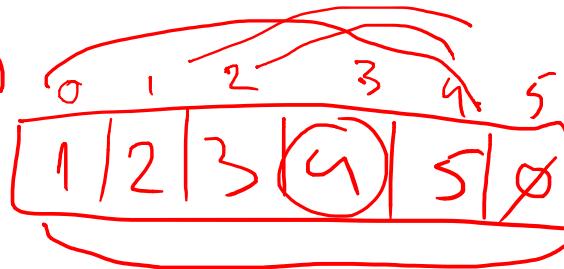
9	9	9	Ø
+	0	0	1
<hr/>			
1	0	0	Ø

$$\begin{array}{r} \cancel{x} - '0' \\ \hline \end{array} \rightarrow \begin{array}{r} 0 \\ \rightarrow 1 \\ \rightarrow 2 \end{array}$$

dig

10^{dig}

$\leftarrow b * X$



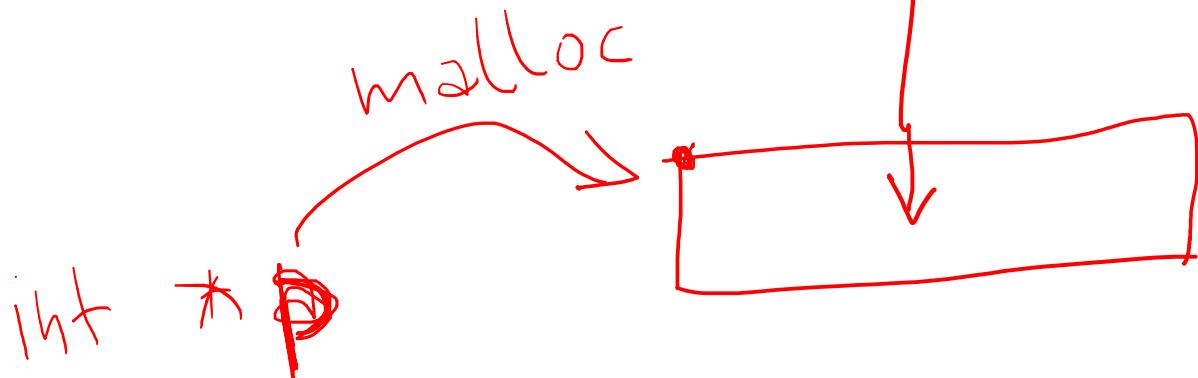
$$\begin{aligned} 0 &\rightarrow 10^0 = 1 \\ 1 &\rightarrow 10^1 \\ 2 &\rightarrow 10^2 \end{aligned}$$

'4' \rightarrow 52

$X [\cancel{4} - 1 - \text{dig}] - '0'$

0 1 2 3 4 ... g

48 49 ...



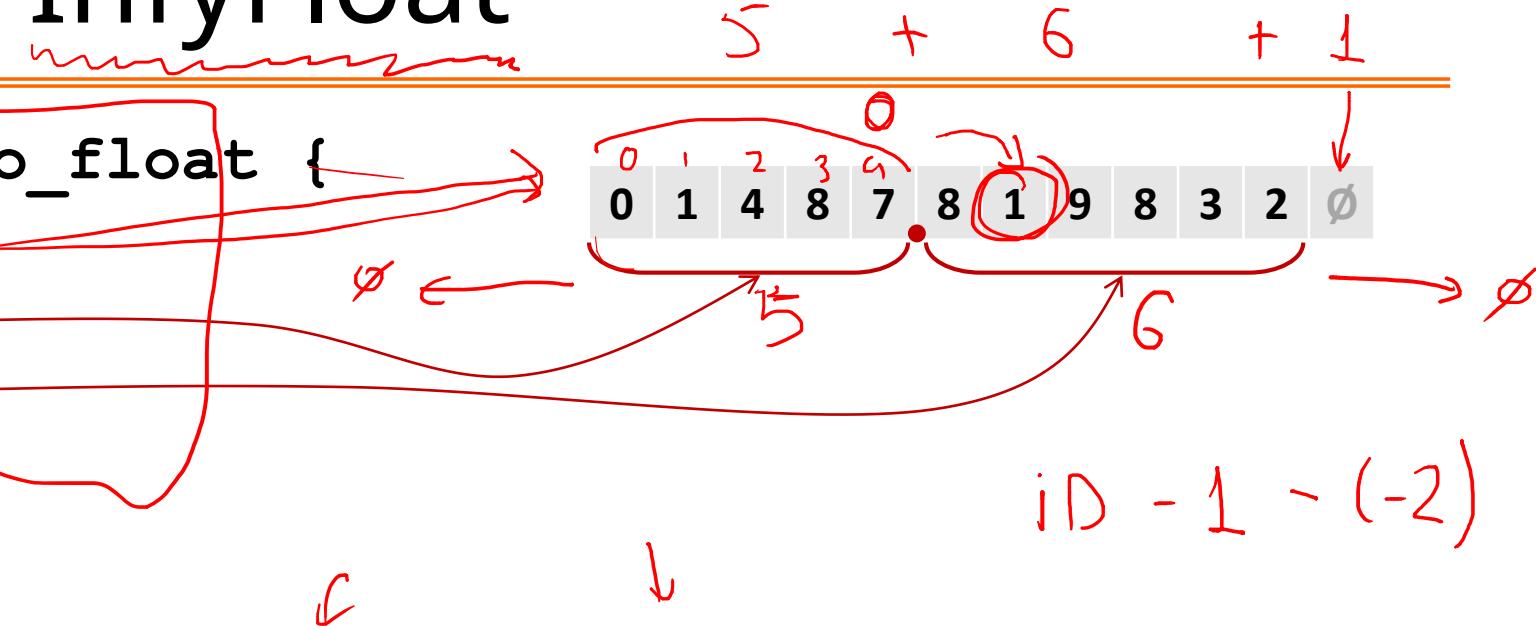
P += 10 ;
P --
P = &*P*[10] ;

) → free
realloc

~~free~~
~~realloc~~

Εφαρμογή 2 - InfyFloat

```
typedef struct _info_float {  
    char *digs;  
    int intDigs;  
    int decDigs;  
} InfyFloat;
```



```
InfyFloat *ifCreate (int iDig, int dDig)  
bool ifValidate(InfyFloat *iFlo)  
int ifGetDigitValue(InfyFloat *iFlo, int dig)
```

