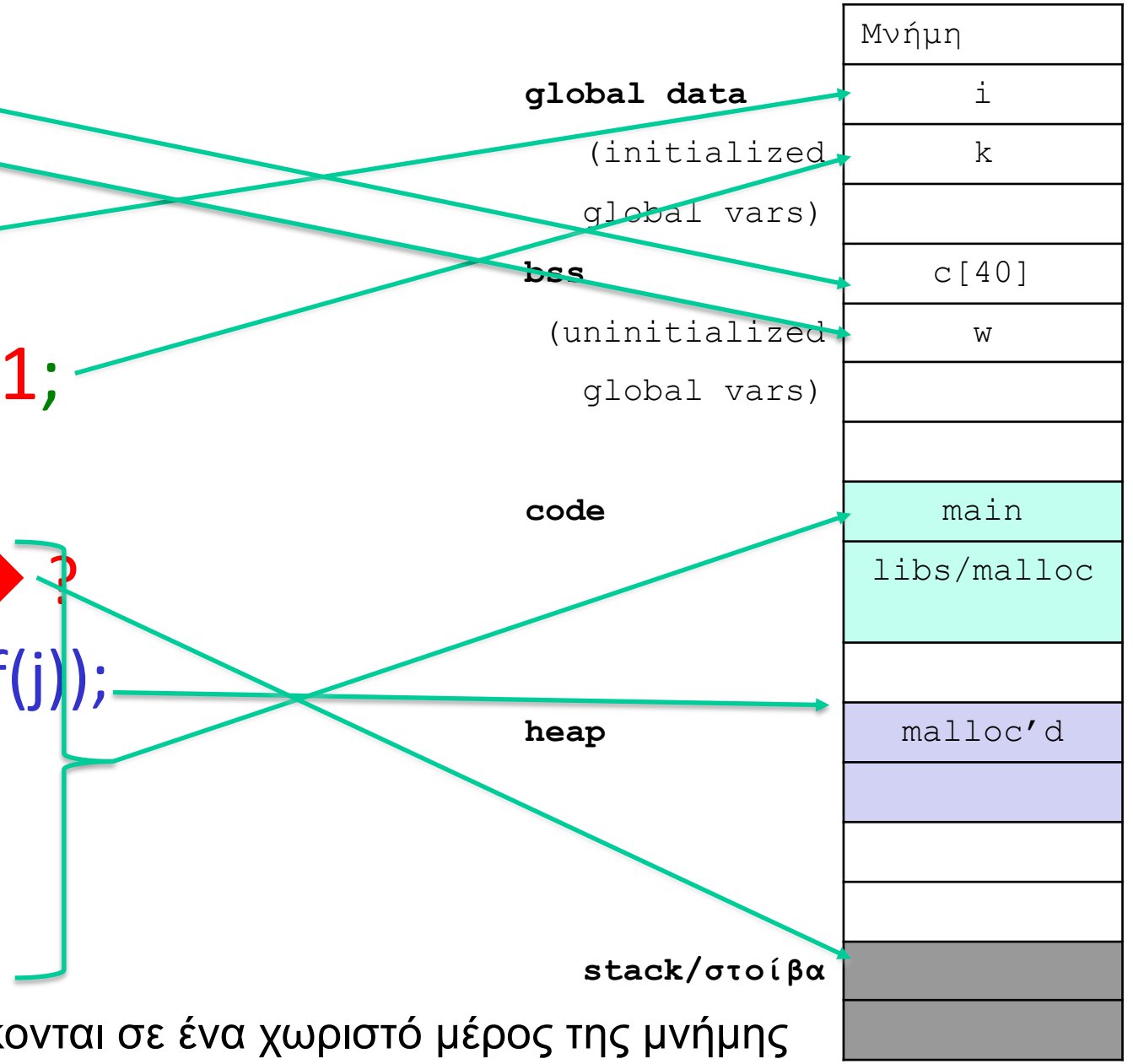
HY255 Εργαστήριο Λογισμικού – L12

(a) Program Stack
(b) Assignment 4 - sudoku

HY255 Εργαστήριο Λογισμικού
Άνοιξη 2021
Άγγελος Μπίλας

Μνήμη Προγράμματος

```
char c[40];
static double w;
int i=13;
static long k=2001;
int main(void){
    int j=3,m,*jp;
    jp=malloc(sizeof(j));
    *jp = j;
    w = 2.0 * k;
}
```



- Οι local μεταβλητές βρίσκονται σε ένα χωριστό μέρος της μνήμης που ονομάζεται στοίβα

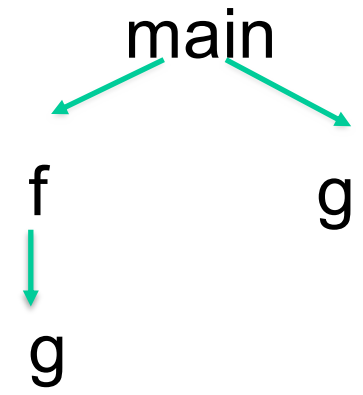
Στοίβα

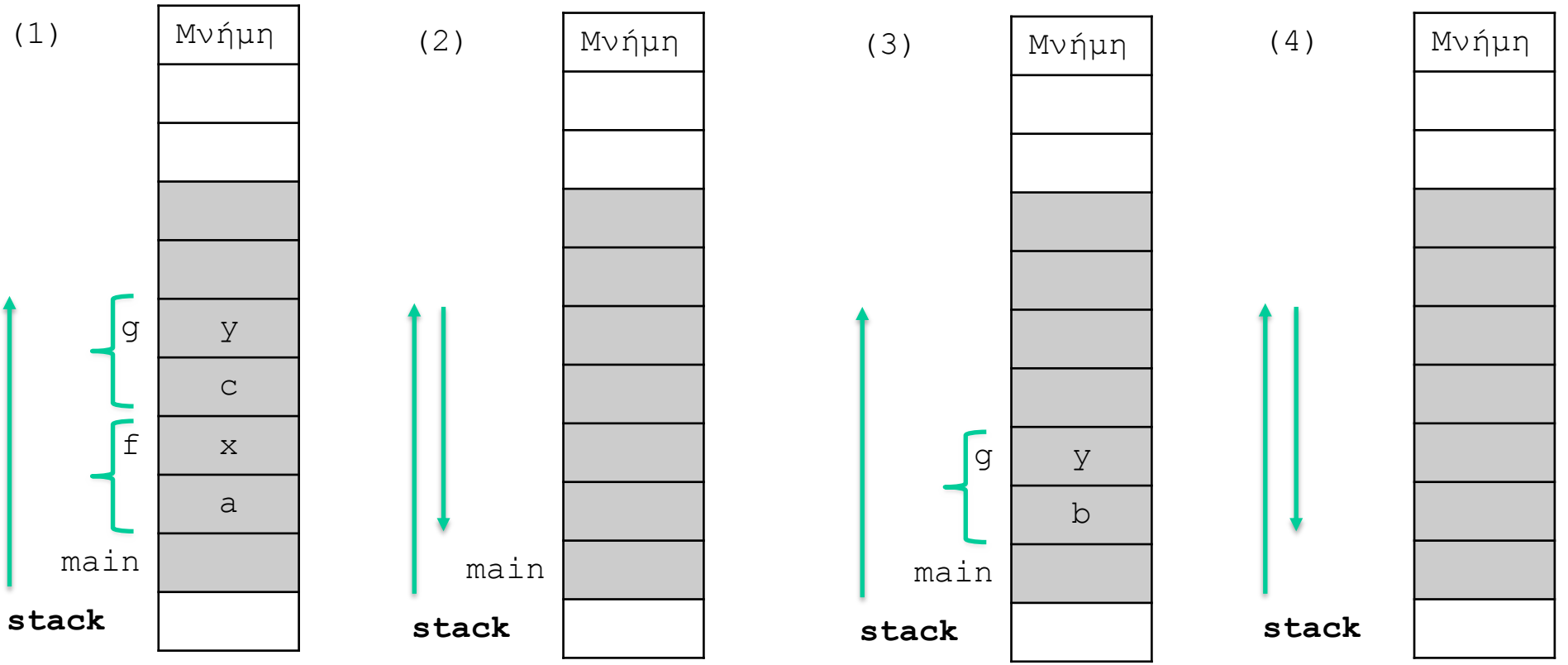
- Η στοίβα είναι ένα τμήμα της μνήμης του προγράμματος που χρησιμοποιείται κατά την κλήση των συναρτήσεων
  - Η στοίβα υλοποιεί την ιδιότητα των τοπικών μεταβλητών να είναι ορατές μόνο από τον κώδικα της συνάρτησης που ορίζονται
  - Η στοίβα επιτρέπει να δημιουργούμε ένα νέο αντίγραφο των τοπικών μεταβλητών για κάθε κλήση μιας συνάρτησης
- Κάθε συνάρτηση όταν ξεκινά, ”παίρνει” κάποιο χώρο στη στοίβα. Ο χώρος αυτός ονομάζεται stack frame (ή μερικές φορές activation record) της κλήσης της συνάρτησης
- Στο stack frame υπάρχει χώρος για όλες τις τοπικές μεταβλητές και input parameters
- Πριν επιστρέψει η κλήση μια συνάρτησης ελευθερώνει το stack frame της και ο χώρος μπορεί να χρησιμοποιηθεί από επόμενη κλήση της ίδιας ή άλλης συνάρτησης

Εξέλιξη της στοίβας κατά την εκτέλεση

- Έστω το απλό πρόγραμμα:

```
main(){          f(int){        g(int){
   f(a);           int x;         int y;→(1),(3)
→(2)g(b);          g(c);          return;
→(4)return
}                }              }
```





Διαχείριση Στοίβας

- Όσο εκτελείται το πρόγραμμα μας η εικόνα της στοίβας αλλάζει διαρκώς
- Αυτό γίνεται ”αυτόματα” – εμείς δεν γράφουμε κώδικα για την στοίβα
  - Ο κώδικας που χειρίζεται την στοίβα παράγεται αυτόματα από τον compiler όταν μεταφράζει το πρόγραμμά μας και παράγει τον κώδικα των κλήσεων για τις συναρτήσεις
  - Ο κώδικας αυτός προστίθεται αυτόματα σε τέσσερα σημεία κάθε κλήσης μια συνάρτησης
    - (1) Πριν την κλήση της συνάρτησης, στο σημείο που γίνεται η κλήση της συνάρτησης
    - (2) Στην αρχή της συνάρτησης που κλήθηκε, αμέσως μετά την κλήση
    - (3) Στο τέλος της συνάρτησης που κλήθηκε, αμέσως πριν την επιστροφής της
    - (4) Μετά την επιστροφή της συνάρτησης, στο σημείο του κώδικα που έγινε η κλήση της συνάρτησης
- Τι συμβαίνει αν μια συνάρτηση καλέσει τον εαυτό της;
  - Μια τέτοια συνάρτηση ονομάζεται αναδρομική
  - Η αναδρομή ονομάζεται άμεση όταν η συνάρτηση f καλεί απ’ευθείας τον εαυτό της και έμμεση όταν καλεί πρώτα μια άλλη συνάρτηση g η οποία κάποια στιγμή οδηγεί στην κλήση της συνάρτησης f
  - Θα δούμε σε λίγο τι συμβαίνει κατά την αναδρομή
- Για ποιο λόγο να χρησιμοποιήσουμε αναδρομή;
  - Επειδή σε μερικά προβλήματα είναι ευκολότερο να εκφράσουμε την λύση χρησιμοποιώντας αναδρομή
  - Ας δούμε ένα τέτοιο παράδειγμα, την quicksort

Quicksort

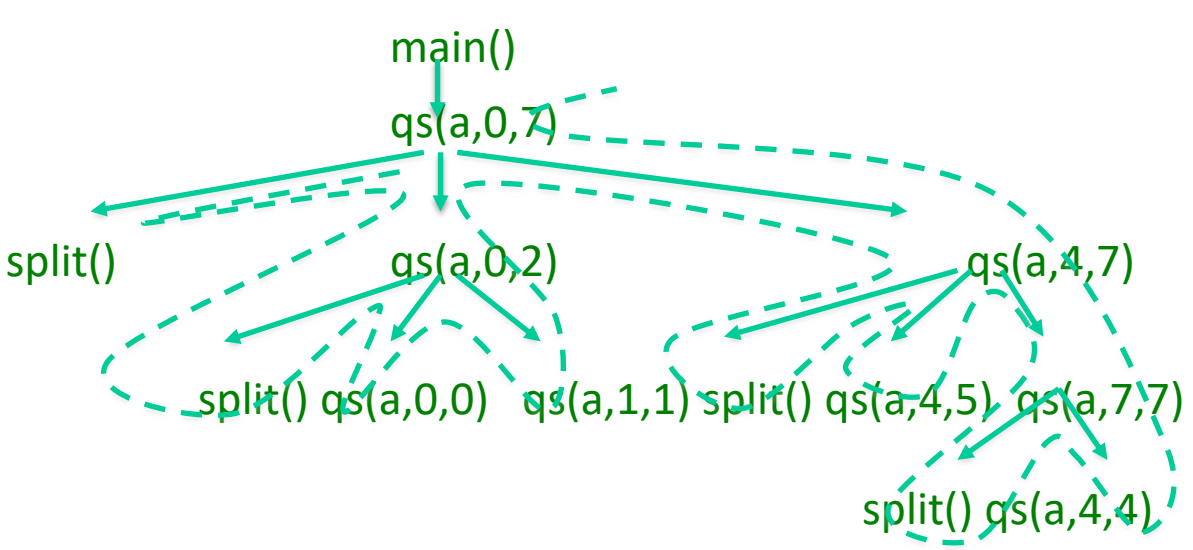
- Περιγραφή: Διάταξε ένα array A[N] από στοιχεία ως εξής:
  - (1) Διάλεξε ένα στοιχείο e. Αναδιάταξε το array A έτσι ώστε:
    - Τα στοιχεία στις θέσεις 0…i-1 να είναι <= e
    - Το στοιχείο στη θέση i να είναι το e
    - Τα στοιχεία στις θέσεις i+1…N-1 να είναι >= e
  - (2) Διάταξε το υπο-array 0…i-1
  - (3) Διάταξε το υπο-array i+1…N-1
- Με αυτή την περιγραφή, είναι εύκολο να ”πειστούμε” ότι μετά το βήμα (3) το A είναι διατεταγμένο
- Πως μετατρέπουμε αυτή την περιγραφή σε κώδικα;
  - Η αναδρομή κάνει αυτή την μετατροπή εύκολη, ουσιαστικά κάθε σκέλος της περιγραφής είναι μια κλήση συνάρτησης

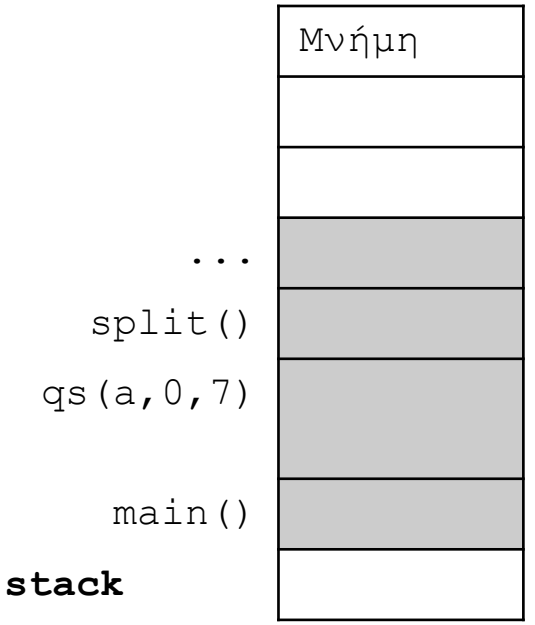
```
void quicksort(int a[], int low, int high){
    int i;
    if (low >= high) return;
    i = split(a, low, high); /*
                                step 1 – Ο κώδικας της split δεν είναι σημαντικός για τον
                                σκοπό μας. Η split διαλέγει το e και μετακινεί τα στοιχεία του
                                array a εσωτερικά με ένα pass      */
    quicksort(a, low, i-1);  /* step 2 */
    quicksort(a, i+1, high); /* step 3 */
    return;
}
```

- Με την χρήση της αναδρομής είναι εύκολο να επιχειρηματολογήσουμε ότι αυτή η υλοποίηση είναι σωστή, αφού είναι ουσιαστικά μια αποτύπωση “ένα προς ένα” της παραπάνω περιγραφής
- Τι συμβαίνει όταν καλείτα η συνάρτηση quicksort, πχ

```
int main(void) {
    int A[N];
    quickqort(A, 0, N-1);
}
```

Εκτέλεση quicksort





- Κάθε χρονική στιγμή η στοίβα περιέχει τα stack frame κάποιων κλήσεων. Ποιών; Των κλήσεων που βρίσκονται στο μονοπάτι από την ρίζα (main) προς την συνάρτηση που εκτελείται αυτή τη στιγμή
- Από τι καθορίζεται το μέγεθος της στοίβας; Μέγεθος κάθε stack frame + αριθμός από stack frames
- Πόσο είναι το μέγεθος του κάθε stack frame; Περίπου όσο το μέγεθος των τοπικών μεταβλητών και των παραμέτρων της συνάρτησης - αργότερα θα δούμε σε λεπτομέρεια τα στοιχεία αυτά για τους επεξεργαστές x86.
- Πόσα stack frames μπορεί να περιέχει; Όσες είναι οι ενεργές συναρτήσεις… Άρα όσο το μεγαλύτερο μονοπάτι από την ρίζα σε κάποιο φύλλο. Στην περίπτωση της quicksort, πόσο μπορεί να είναι το μεγαλύτερο μονοπάτι στη χειρότερη περίπτωση (~N);
- Γενικά στην αναδρομή μπορεί να υπάρχουν πολλές συναρτήσεις ενεργές ανά πάσα στιγμή
  - Αν δεν έχουν αναδρομή, πόσες συναρτήσεις μπορεί να είναι ενεργές; Αντιδιαστείλετε τις δύο περιπτώσεις (με και χωρίς αναδρομή)

C blocks

- Η C σε κάποιες εκδόσεις της, επιτρέπει την χρήση των statement blocks

```
int main(){
    int x=1;
    int y;
    ...
    {
            int x = 2, z;
            ...
    }
    ...
}
```

- Τα x,z που δηλώνονται στο block έχουν scope μόνο το block.
- Το block έχει ένα δικό του stack frame στη στοίβα για όσο εκτελείται.
- Μια διαφορά με τις συναρτήσεις στη C είναι ότι μέσα στο statement block είναι ορατές και οι μεταβλητές x,y που δηλώνονται ως τοπικές μεταβλητές στην main. Αυτό απαιτεί κάποια διαφορετική διαχείριση στη στοίβα για το stack frame του block (nested scopes), ώστε να μπορεί ο κώδικας μέσα στο block να βρει την μεταβλητή π.χ. y στο stack frame της main (και όχι του block).

sudoku

	4	3	5	7	9		8	2
	2	7	3	1	8			
	5						3	7
9		2	6	5	7	4		3
		4		3		7		8
7	3	1			4			6
2		6	1	8		3		
			7		3	2	5	
3			4			8	6	

Εξάσκηση ελεύθερου χρόνου

- Μια υλοποίηση της split για arrays από integers είναι

```
int split(int a[], int low, int high){
     int e = a[low];
     for (;;){
       while (low<high && e<=a[high]) high--;
       if (low >= high) break;
       a[low++] = a[high];
       while (low<high && a[low]<=e) low++;
       if (low>=high) break;
       a[high--]=a[low];
     }
     a[high] = e;
     return high;
}
```

- Γράψτε μια υλοποίηση της qsort που είναι πολυμορφική, όπου το array a[] μπορεί να περιέχει στοιχεία οποιουδήποτε τύπου

Reading

- King 9.6, 26.2 (rand/srand)