

ΕΠΛ232 – Προγραμματιστικές Τεχνικές και Εργαλεία

Δυναμική Δέσμευση Μνήμης (Κεφάλαιο 17.1-17.4, ΚΝΚ-2ΕΔ)

Τμήμα Πληροφορικής, Πανεπιστήμιο Κύπρου

<http://www.cs.ucy.ac.cy/courses/EPL232>



Περιεχόμενο Διάλεξης 9

- **Δυναμική Δέσμευση Μνήμης**
 - Συναρτήσεις Δέσμευσης Μνήμης
 - Δείκτες **NULL**
- **Δυναμικά Δεσμευμένες Συμβολοσειρές**
 - Συμβολοσειρές με Δυναμική Χορήγηση Μνήμης
 - Πίνακες Δυναμικά Δεσμευμένων Συμβολοσειρών
- **Αποδέσμευση Μνήμης**
 - Η συνάρτηση **free**
 - Το πρόβλημα του αιρούμενου δείκτη (dangling pointer)
- **Δυναμικά Δεσμευμένοι Πίνακες**
 - Συναρτήσεις **malloc**, **calloc** και **realloc**



Dynamic Storage Allocation

Δυναμική Δέσμευση Μνήμης

- Οι δομές δεδομένων σε C, συμπεριλαμβανομένων και των πινάκων, έχουν σταθερό μέγεθος.
- Οι δομές δεδομένων σταθερού μεγέθους αποτελούν πρόβλημα, δεδομένου ότι είμαστε αναγκασμένοι να επιλέξουμε τα μεγέθη τους κατά τη σύνταξη του προγράμματος.
- Η C υποστηρίζει τη **Δυναμική Δέσμευση Μνήμης**: τη δυνατότητα δέσμευσης μνήμης κατά την εκτέλεση του προγράμματος.
- Χρησιμοποιώντας τη δυναμική δέσμευση μνήμης, μπορούμε να σχεδιάσουμε δομές δεδομένων που μεγαλώνουν (και συρρικνώνονται) κατά επιλογή του χρήστη.



Dynamic Storage Allocation

Δυναμική Δέσμευση Μνήμης

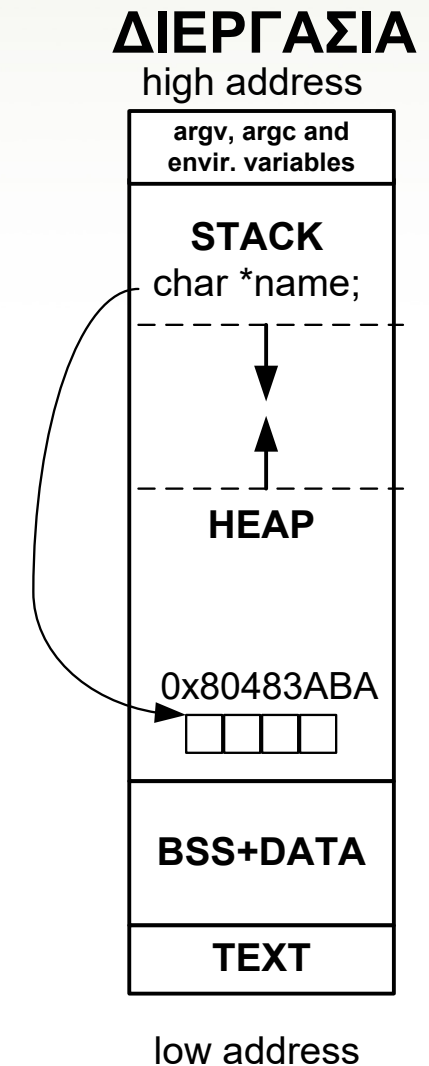
- **Δυναμική Δέσμευση Μνήμης:** Η διαδικασία κατά την οποία εκχωρείται επιπλέον χώρος σε μια διεργασία (πρόγραμμα) από τον Πυρήνα
- **Εφαρμογές:**
 - Δέσμευση Μνήμης όταν απαιτείται και αποδέσμευση όταν δεν απαιτείται.
 - Δημιουργία Πίνακα Μη-προκαθορισμένων διαστάσεων
 - Δημιουργία Συμβολοσειράς αυθαίρετου μεγέθους.
 - Δημιουργία σύνθετων δομών δεδομένων στη μνήμη (όπως λίστες, δέντρα, στοίβες, ουρές, κτλ.)
 -



Dynamic Storage Allocation

Δυναμική Δέσμευση Μνήμης

- Η μνήμη που εκχωρείται σε ένα πρόγραμμα (μέσω π.χ., malloc) είναι προσπελάσιμη μέσω της **σωρού (heap)** της.
- Η γενική φιλοσοφία είναι ότι οι **μεταβλητές τύπου δείκτη** από τη **στοίβα** του προγράμματος δείχνουν στο δεσμευμένο χώρο που εκχωρείται στο πρόγραμμα
 - Π.χ., char *name;
- Τα αντικείμενα μέσα στη σωρό είναι **προσπελάσιμα** από οποιοδήποτε συνάρτηση (σε οποιοδήποτε επίπεδο)
 - Κάνοντας χρήση της διεύθυνσης του αντικείμενου, π.χ., 0x80483AB η οποία αποθηκεύεται μέσα στο char *name;



Memory Allocation Functions

Συναρτήσεις Δέσμευσης Μνήμης

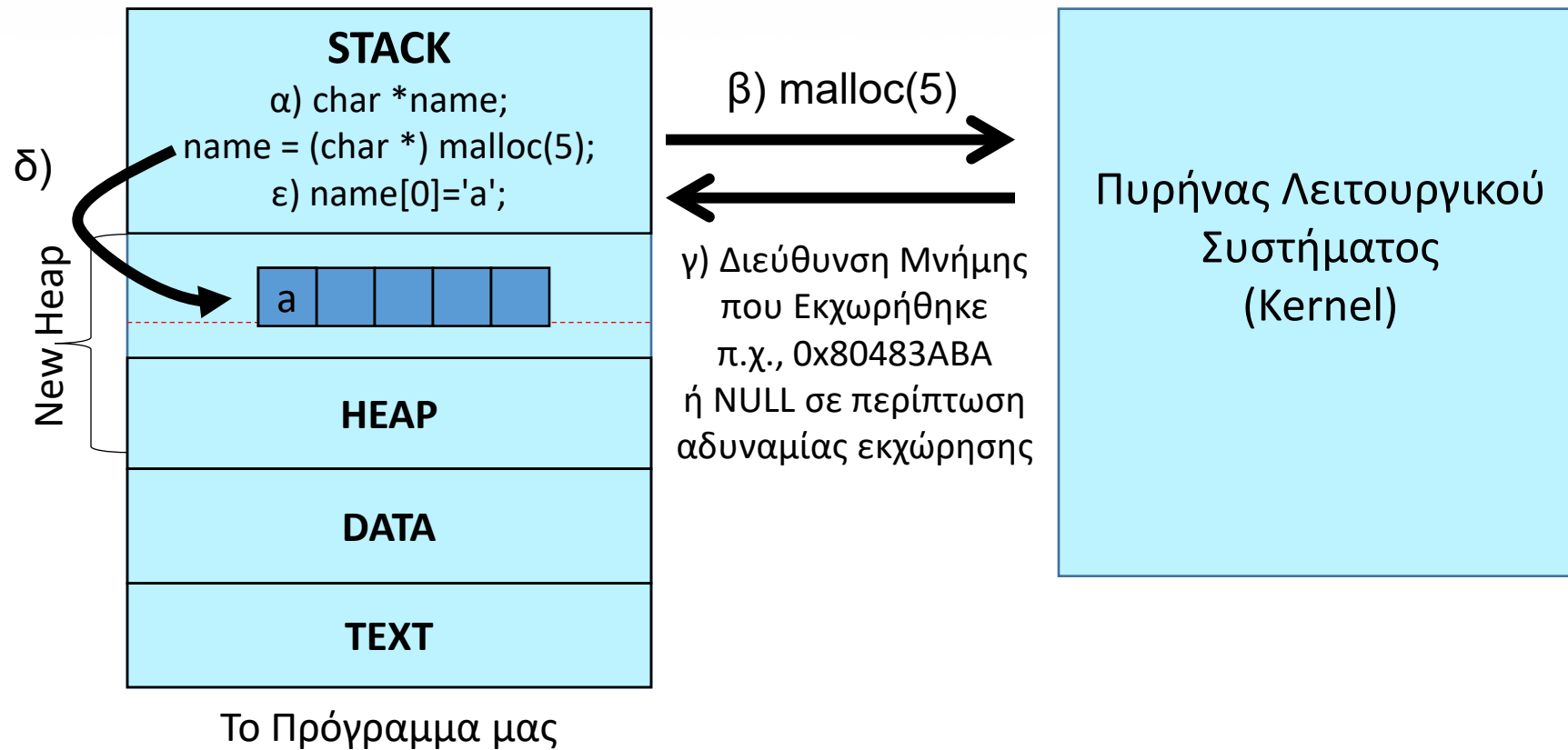
- Η βιβλιοθήκη `<stdlib.h>` ορίζει **τρεις συναρτήσεις (κλήσεις συστήματος)** για να **εκχωρηθεί** μνήμη από τον πυρήνα του **Λειτουργικού Συστήματος** :
 - **malloc**: Δεσμεύει ένα block μνήμης (μετριέται σε bytes) αλλά **ΔΕΝ** τον αρχικοποιεί.
 - **calloc**: Δεσμεύει ένα block μνήμης και το **αρχικοποιεί** (με τον χαρακτήρα `'\0'`).
 - **realloc**: Αλλάζει το μέγεθος ενός προηγούμενου δεσμευμένου μπλοκ μνήμης.
- Αυτές οι συναρτήσεις επιστρέφουν μια τιμή τύπου **void *** (ένα "γενικό" δείκτη δηλαδή, εφόσον κατά τη δέσμευση δεν είναι γνωστό τι τελικό τύπο θα έχει η εν λόγω μνήμη).



Memory Allocation Functions

Συναρτήσεις Δέσμησης Μνήμης

Αναπαράσταση της Διαδικασίας Δέσμησης Μνήμης από ένα Πρόγραμμα



Συναρτήσεις Δέσμησης Μνήμης (malloc)

- Η δυναμική δέσμηση μνήμης είναι συχνά χρήσιμη για συμβολοσειρές.
- Οι συμβολοσειρές αποθηκεύονται σε πίνακες χαρακτήρων και μπορεί να είναι δύσκολο να προβλέψετε για πόσο χρόνο πρέπει να είναι δεσμευμένες αυτές οι συστοιχίες.
- Δεσμεύοντας τις συμβολοσειρές δυναμικά, μπορούμε να αναβάλουμε αυτή την απόφαση μέχρι να εκτελεστεί το πρόγραμμα.



Συναρτήσεις Δέσμευσης Μνήμης (malloc)

- Πρότυπο Συνάρτησης:

```
void *malloc(size_t size);
```

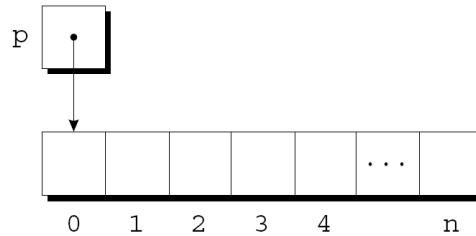
- Δεσμεύει ένα block μνήμης (μεγέθους size bytes) αλλά **ΔΕΝ τον αρχικοποιεί**.
- Η συνάρτηση επιστρέφει **ένα δείκτη** στον χώρο που δεσμεύτηκε.
- Βιβλιοθήκη: `<stdlib.h>`
- `size_t` είναι τύπου `unsigned integer` το οποίο ορίζεται στη βιβλιοθήκη.



Συναρτήσεις Δέσμησης Μνήμης (malloc)

- Παράδειγμα κλήσης `malloc` το οποίο δεσμεύει χώρο στη σωρό προγράμματος για μια συμβολοσειρά `n` χαρακτήρων (+1 για το επιπλέον `'\0'` χαρακτήρα):

```
p = malloc(n + 1);
```



- Εφόσον ο δείκτης `p` αναφέρεται ουσιαστικά σε `char *`, είναι καλή πρακτική (εάν και όχι απαραίτητη) να γίνεται το ανάλογο casting:

```
p = (char *) malloc(n + 1);
```

Άλλο Παράδειγμα δέσμησης 10 PERSON

```
p = (PERSON *) malloc (sizeof(PERSON) * 10);
```



NULL Δείκτες (Null Pointers)

- Εάν ο πυρήνας δεν μπορεί να παραχωρήσει τον αιτούμενο χώρο, επιστρέφεται στη κλήση δέσμευσης μνήμης ένας **Κενός Δείκτης (Null Pointer)**.

```
p = malloc(10000);  
if (p == NULL) {  
    /* allocation failed; take action */  
}  
ή if ((p = malloc(10000)) == NULL) { ... }
```

- **Κενός Δείκτης:** Μια ειδική τιμή (δηλ., 0) η οποία μπορεί να διαχωριστεί από τις υπόλοιπες τιμές που δύναται να έχει ένας ορθός δείκτης (σε διεύθυνση μνήμης).
 - `printf("%ld", (unsigned long int) NULL);` Τυπώνει "0"



NULL Δείκτες (Null Pointers)

- Οι δείκτες ελέγχουν εάν κάτι είναι TRUE ή FALSE με τον ίδιο τρόπο όπως οι αριθμοί.
- Όλοι οι μη null δείκτες επιστρέφουν True, μόνο οι μηδενικοί δείκτες NULL είναι FALSE.
- Οπότε, αντί για
 - `if (p == NULL)` ← μπορούμε να γράψουμε → `if (!p)`
 - `if (p != NULL)` ← μπορούμε να γράψουμε → `if (p)`

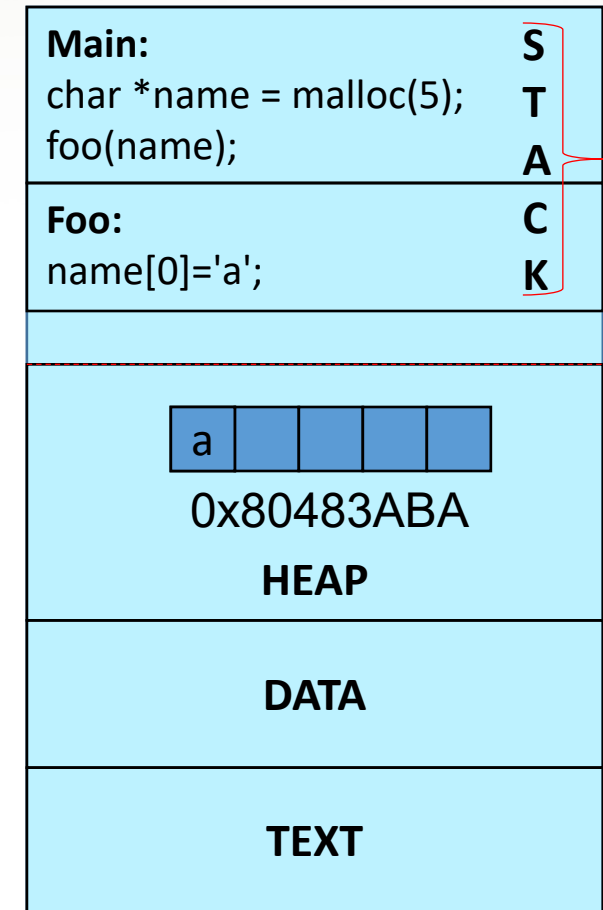
NULL Δείκτες (Null Pointers)

- Εάν και το **NULL** μοιάζει με την τιμή **0**, αυτό είναι **διαφορετικό** από τις επί μέρους **αναπαραστάσεις του 0** που γνωρίζουμε:
 - 0 (τύπου ακεραίου, π.χ., `int`)
 - `'\0'` (τύπου χαρακτήρα, δηλ., `char`)
 - Σε πολλούς αρέσει: `#define NUL '\0'`
 - `"0"` (τύπου συμβολοσειράς, δηλ `char[]`)
 - `NULL` (τύπου δείκτη, δηλ., `void *`, `char *`, `int *`, ..., κτλ.)
 - `int *p = (void *) 0; /* Άσχημο Στυλ */`
 - `int *p = NULL /* Καλό Στυλ */`



Δεδομένα στη Σωρό (Data in Program Heap)

- Ένα **αντικείμενο** στη σωρό του προγράμματος είναι **προσπελάσιμο** μέσω της **διεύθυνσης του**
- Συνεπώς, εάν δημιουργηθεί μια δομή δεδομένων στη σωρό τότε **οποιαδήποτε συνάρτηση** μπορεί να επεξεργάζεται τα στοιχεία τα οποία περνάνε **δια-αναφορές**.

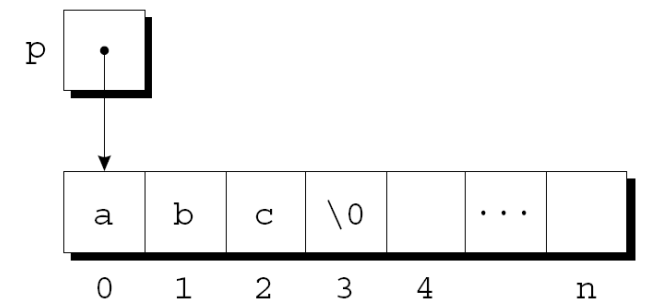
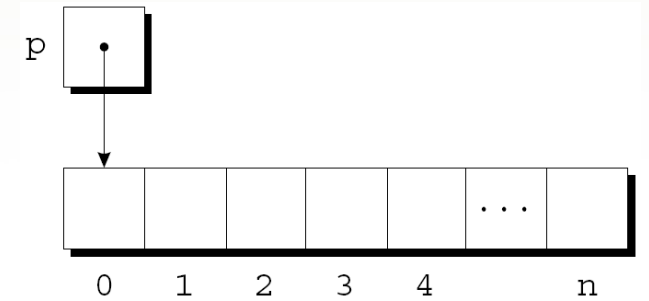


Το Πρόγραμμα μας



Συναρτήσεις & Δυναμική Δέσμευση Μνήμης

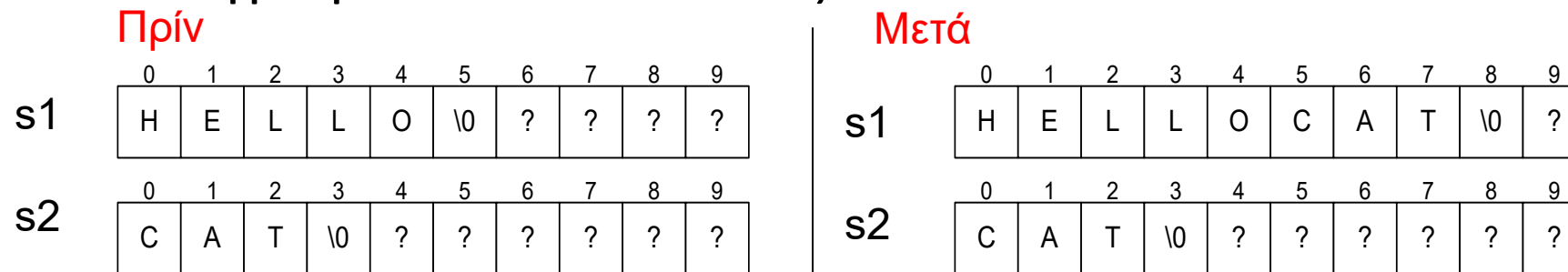
- Η μνήμη που δεσμεύετε από τη `malloc` δεν είναι ξεκάθαρη, οπότε το `p` θα δείχνει σε έναν μη αρχικοποιημένο πίνακα με $n + 1$ χαρακτήρες:
- Καλώντας την `strcpy` (δες διάλεξη 7) μπορούμε να την αρχικοποιήσουμε:
`strcpy(p, "abc");`
- Οι πρώτοι τέσσερις χαρακτήρες στον πίνακα θα είναι τώρα `a`, `b`, `c`, και `\0`:



Συναρτήσεις & Δυναμική Δέσμευση Μνήμης

Η Συνάρτηση `strcat`

- Ας θυμηθούμε ξανά τη συνάρτηση `strcat(s1, s2)`, από τη διάλεξη 7, η οποία αντιγράφει το `s2` στο τέλος του `s1`.



- Θυμάστε ότι η υλοποίηση που είχαμε δώσει μπορούσε να προκαλέσει υπερχείλιση του `s1`
- Άσκηση:** Δώστε μια συνάρτηση η οποία επιτελεί την ίδια εργασία αλλά δεν προκαλεί υπερχείλιση του `s1` με τη χρήση της `malloc()`
 - `char *mystrcat(const char *s1, const char *s2)`

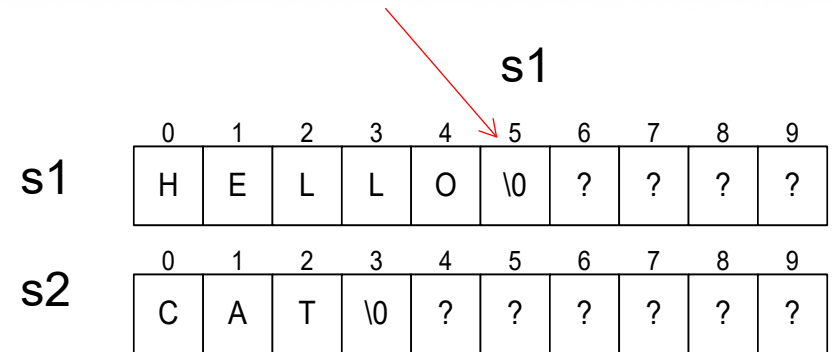
→ επιστροφή



Η Συνάρτηση `strcat` με Υπερχείλιση του `s1`

- **Επανάληψη:** Μια συμπαγής έκδοση της `strcat`:

```
char *strcat(const char *s1, const char *s2)
{
    if (s1==NULL) return s2;
    else if (s2==NULL) return s1;
    while (*s1 != '\0') {
        s1++;
    }
    while (*s1 = *s2) { // ανάθεση, όχι σύγκριση
        s1++; s2++;
    }
    return s1;
}
```



Η Συνάρτηση `mystrcat` ΧΩΡΙΣ Υπερχείλιση του `s1`

Παράδειγμα: Μια συνάρτηση που συνενώνει δύο συμβολοσειρές χωρίς να αλλάζει καμία από τις δύο.

```
char *mystrcat(const char *s1, const char *s2)
{
    if (s1==NULL) return s2;
    else if (s2==NULL) return s1;

    char *result = NULL;
    result = malloc(strlen(s1) + strlen(s2) + 1);
    if (result == NULL) {
        printf("Error: malloc failed in concat\n");
        return NULL;
    }
    strcpy(result, s1);
    strcat(result, s2);
    return result;
}
```



Η Συνάρτηση `mstrcat` ΧΩΡΙΣ Υπερχείλιση του `s1`

- Ένα κάλεσμα της συνάρτησης `mstrcat`:

```
p = mstrcat("abc", "def");
```

- Μετά το κάλεσμα, ο `p` θα δείχνει στη συμβολοσειρά `"abcdef"`, που αποθηκεύεται σε μια συστοιχία που η μνήμη της δεσμεύεται δυναμικά.
- Συναρτήσεις όπως η `mstrcat` πρέπει να χρησιμοποιούνται προσεκτικά.
- Όταν η συμβολοσειρά που επιστρέφει η `mstrcat` δεν χρειάζεται πλέον, θα θέλουμε να καλέσουμε την συνάρτηση `free` για να απελευθερώσουμε το χώρο που καταλαμβάνει η συμβολοσειρά.
- Αν δεν το κάνουμε, το πρόγραμμα μπορεί τελικά να καταλήξει να μην έχει άλλη διαθέσιμη μνήμη.



Deallocating Storage

Αποδέσμευση Μνήμης

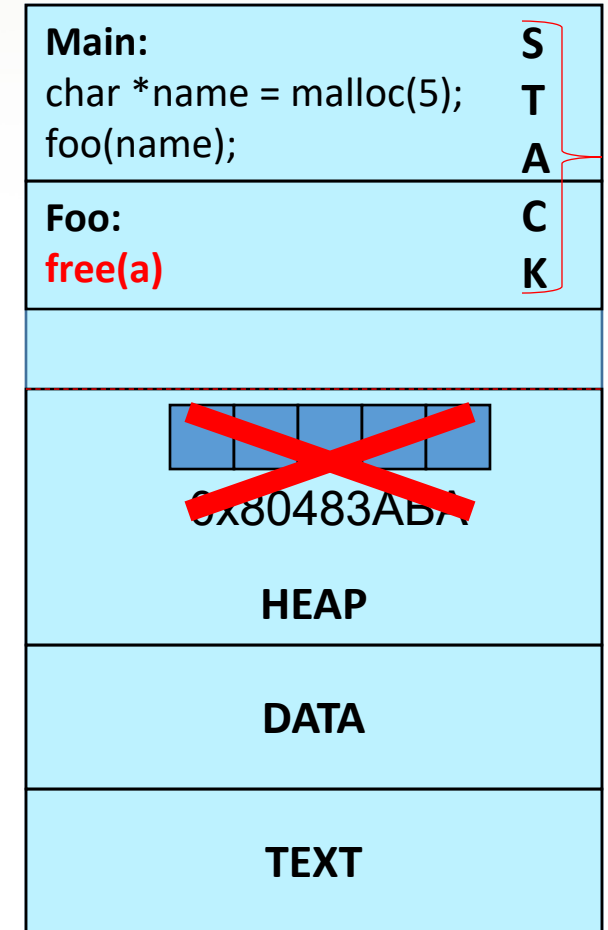
- Τα δεδομένα της σωρού έχουν **στατική αποθηκευτική διάρκεια**.
 - Συνεπώς, με την ολοκλήρωση μιας συνάρτησης αυτά **ΔΕΝ διαγράφονται αυτόματα**.
 - Η **αποδέσμευση** πρέπει να γίνεται από τον προγραμματιστή με την εντολή **free()**

- Πρότυπο Συνάρτησης `free`:

```
void free(void *ptr) ;
```

Κλήση στον Πυρήνα για αποδέσμευση χώρου που δεικνύεται από το `ptr`.

- το `ptr` παραμένει! Το `ptr` δημιουργήθηκε με `malloc` ή `calloc` ή `realloc` μόνο!
- Το `free` δεν μπορεί να γίνει στην μέση ενός δυναμικού πίνακα γιατί ο kernel θυμάται για κάθε πίνακα μόνο την διεύθυνση αρχής.



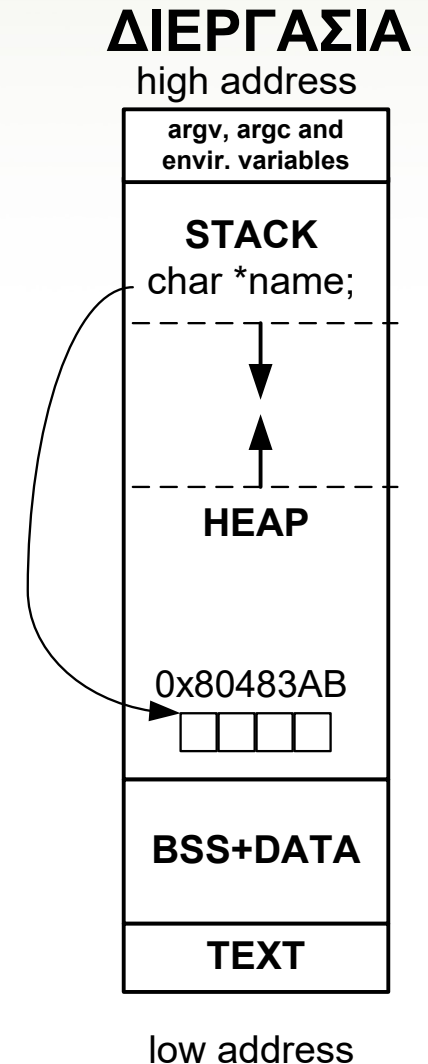
Το Πρόγραμμα μας



Deallocating Storage

Αποδέσμευση Μνήμης

- Η `malloc` και οι άλλες συναρτήσεις εκχώρησης μνήμης λαμβάνουν μπλοκ μνήμης από ένα χώρο αποθήκευσης, το *heap*.
- Πριν προχωρήσουμε να αναφέρουμε ότι κατά τον **τερματισμό** μιας διεργασίας, **ΟΛΗ** η μνήμη του προγράμματος **επιστρέφεται στον πυρήνα** του Λ.Σ.
 - TEXT, DATA, STACK, HEAP, κτλ.
- **Ωραία, τότε γιατί εκτελούμε τη `free()`;**
 - Τα προγράμματα (π.χ., υπηρεσίες συστήματος) σχεδιάζονται για να λειτουργούν χωρίς επανεκκίνηση.
 - Συνεπώς, εάν δεν γίνεται `free()`, τότε σύντομα θα αναλωθεί όλη η μνήμη και οι κλήσεις `malloc` θα παίρνουν NULL



Διαρροή Μνήμης (Memory Leak)

- **Διαρροή Μνήμης (Memory Leak):** Κατάσταση κατά την οποία μια διεργασία **δεσμεύει** μνήμη από το Λ.Σ. αλλά **ΔΕΝ μπορεί να το επιστρέψει** στο Λ.Σ.
 - Συμβαίνει όταν χάσουμε την **αναφορά** σε αντικείμενα δεσμευμένα στη Σωρό.

ΔΙΕΡΓΑΣΙΑ

high address

argv, argc and
envir. variables

STACK

char *name;

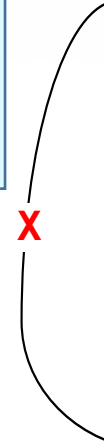
HEAP

BSS+DATA

TEXT

low address

X

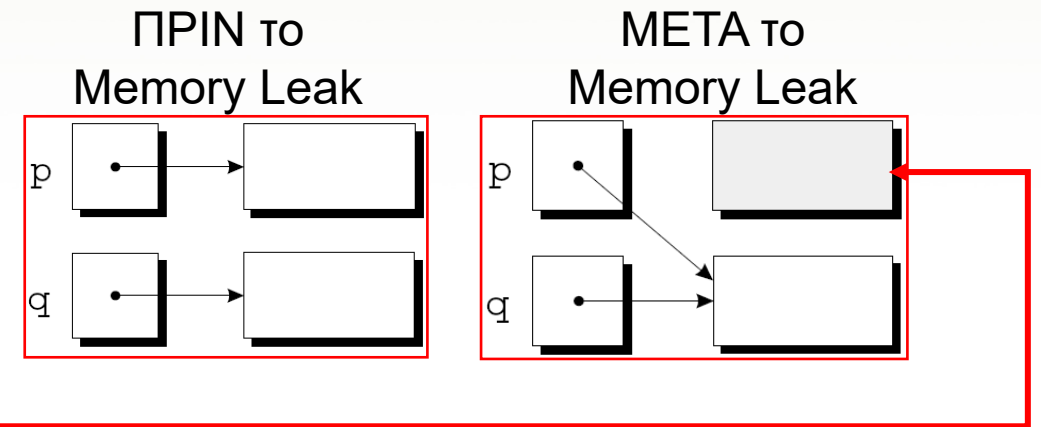


Διαρροή Μνήμης (Memory Leak)

- Παράδειγμα Memory Leak

```
p = malloc(...);  
q = malloc(...);  
// free(p);  
p = q;
```

Σκουπίδια!!!
(garbage)



- Δεν υπάρχουν δείκτες στο πρώτο μπλοκ, έτσι δεν θα μπορέσουμε ποτέ να το χρησιμοποιήσουμε ξανά.
- Ένα μπλοκ μνήμης που δεν είναι πλέον προσβάσιμο σε ένα πρόγραμμα λέγεται ότι είναι σκουπίδι (**garbage**).
- Ένα πρόγραμμα που αφήνει πίσω σκουπίδια λέγεται ότι έχει **memory leak**.
- Κάθε πρόγραμμα σε C είναι υπεύθυνο για την ανακύκλωση των δικών του σκουπιδιών με τη συνάρτηση `free` ώστε να απελευθερωθεί η μνήμη.



Προβλήματα με τη Free()

- Εάν προκύψουν προβλήματα στη free (crash), τότε αυτό οφείλεται μάλλον σε ένα από τους ακόλουθους λόγους
 - Η περιοχή έχει αποδεσμευτεί ήδη (από προηγούμενη κλήση)
 - Ο δείκτης που επιχειρείται να αποδεσμεύσετε δεν είναι δείκτης που επιστράφηκε από malloc/calloc/realloc.
- Αποφεύγετε τη κλήση `fp = malloc(...)` αμέσως μετά από `free(fp)`
 - Σε κάποια συστήματα έχουν παρατηρηθεί διάφορα προβλήματα με αυτή τη λογική.



Η συνάρτηση `free`

- Πρότυπο για `free`:

```
void free(void *ptr);
```

- Η `free` θα περάσει ένα δείκτη σε ένα περιττό μπλοκ μνήμης:

```
p = malloc(...);
```

```
q = malloc(...);
```

```
free(p);
```

```
p = q;
```

- Η κλήση της `free` απελευθερώνει το μπλοκ μνήμης που δείχνει η `p`.

Αιρούμενος Δείκτης (Dangling Pointer)

- Η `free(p)` αποδεσμεύει το μπλοκ μνήμης που δείχνει η `p`, αλλά δεν αλλάζει αυτό καθαυτό το `p`.
- Αν ξεχάσουμε ότι το `p` δεν δείχνει πλέον σε έγκυρο μπλοκ μνήμης, μπορεί να προκληθεί χάος:

```
char *p = malloc(4);  
...  
free(p);  
...  
strcpy(p, "abc");    /*** WRONG ***/
```

- Η αλλαγή της μνήμης που δείχνει το `p` είναι ένα σοβαρό λάθος.

Αιρούμενος Δείκτης (Dangling Pointer)

- Η χρήση της `free` οδηγεί ωστόσο σε ένα νέο πρόβλημα: τον **αιρούμενο δείκτη (*dangling pointer*)**.

- **Αιρούμενος Δείκτης (Dangling Pointer):** Δείκτης ο οποίος δείχνει σε αντικείμενο το οποίο δεν υπάρχει στην μνήμη.

- Π.χ., έχει αποδεσμευθεί από άλλη `free` το εν λόγω αντικείμενο.

- Παράδειγμα Αιρούμενου Δείκτη

```
char *p = malloc(...);
```

```
...
```

```
free(p);
```

```
... // Το p είναι dangling pointer
```

```
strcpy(p, "abc") // crash
```

- Η αλλαγή της μνήμης που δείχνει το `p` είναι ένα σοβαρό λάθος.



Dynamically Allocated Arrays

Δυναμικά Δεσμευμένοι Πίνακες

- Οι **δυναμικά δεσμευμένοι** πίνακες έχουν τα ίδια **πλεονεκτήματα** με τα **δυναμικά strings**.
- Η **συνυφασμένη έννοια** των **πινάκων / δεικτών** μας επιτρέπει να επεξεργαζόμαστε τους **δυναμικούς πίνακες** με την **ίδια ευκολία** που το πράττουμε με τους **σταθερούς πίνακες**.
- **Παραδείγματα που θα μελετήσουμε:**
 - Δημιουργία δυναμικού 1-d διανύσματος
 - Δημιουργία δυναμικού m-d πίνακα



Μονοδιάστατος Πίνακας

Δυναμικά Δεσμευμένοι Πίνακες

• Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *a = NULL, n;
    printf("Enter Array Size:");
    scanf("%d", &n);
    /* Δημιουργία Δυναμικού Πίνακα στη Σωρό*/
    a = (int *) malloc(n * sizeof(int));
    /* Αρχικοποίηση και Εκτύπωση Πίνακα */
    for (int i = 0; i < n; i++) a[i] = i;
    for (int i = 0; i < n; i++) printf("%3d,", a[i]);
    /* Απελευθέρωση Δυναμικού Πίνακα στη Σωρό */
    free(a);
    return 0;
}
```

Δημιουργία δυναμικού 1-d
διανύσματος



Συνεχόμενος Πολυδιάστατος Πίνακας

Δυναμικά Δεσμευμένοι Πίνακες

• Παράδειγμα

```
#include <stdio.h>
#include <stdlib.h>
int main() {
    int *a = NULL;
    int n;
    printf("Enter Matrix (nxn) Size:");
    scanf("%d", &n);
    int k = 0;
    a = (int *) malloc(n * n * sizeof(int));
    for (int i=0; i<n; i++)
        for (int j=0; j<n; j++)
            a[i*n + j] = ++k;

    free(a);
    return 0;
}
```

Δημιουργία δυναμικού m-d πίνακα

```
1, 2, 3, 4, 5, 6, 7, 8, 9, 10,
11, 12, 13, 14, 15, 16, 17, 18, 19, 20,
21, 22, 23, 24, 25, 26, 27, 28, 29, 30,
31, 32, 33, 34, 35, 36, 37, 38, 39, 40,
41, 42, 43, 44, 45, 46, 47, 48, 49, 50,
51, 52, 53, 54, 55, 56, 57, 58, 59, 60,
61, 62, 63, 64, 65, 66, 67, 68, 69, 70,
71, 72, 73, 74, 75, 76, 77, 78, 79, 80,
81, 82, 83, 84, 85, 86, 87, 88, 89, 90,
91, 92, 93, 94, 95, 96, 97, 98, 99, 100,
```

Η Λύση αυτή θα βελτιωθεί στη Διάλεξη-10 έτσι ώστε να αναφερόμαστε στα στοιχεία του πίνακα ως $a[i][j]$!



Δυναμικά Δεσμευμένοι Πίνακες (`malloc`)

- Ας υποθέσουμε ότι ένα πρόγραμμα χρειάζεται μια σειρά από n ακέραιους, όπου n υπολογίζεται κατά την εκτέλεση του προγράμματος.

- Πρώτα θα δηλώσουμε μια μεταβλητή δείκτη:

```
int *a;
```

- Μόλις η τιμή του n είναι γνωστό, το πρόγραμμα μπορεί να καλέσει τη `malloc` για να δεσμεύσει μνήμη για τον πίνακα:

```
a = malloc(n * sizeof(int));
```

- Χρησιμοποιείτε πάντα το `sizeof` για να υπολογίσει τον απαιτούμενο χώρο για κάθε στοιχείο.



Δυναμικά Δεσμευμένοι Πίνακες (**malloc**)

- Μπορούμε τώρα να αγνοήσουμε το γεγονός ότι ο `a` είναι ένας δείκτης και να το χρησιμοποιήσετε ως πίνακα, χάρη στη σχέση μεταξύ πινάκων και δεικτών στη C.
- Για παράδειγμα, θα μπορούσαμε να χρησιμοποιήσουμε τον ακόλουθο βρόχο για την προετοιμασία του πίνακα `a` που δείχνει στο:

```
for (i = 0; i < n; i++)  
    a[i] = 0;
```

- Έχουμε επίσης την επιλογή να χρησιμοποιούμε αριθμητική δείκτη αντί για subscripting για να έχουμε πρόσβαση στα στοιχεία του πίνακα.

Προχωρημένες Συναρτήσεις

Η Συναρτήσεις `calloc/realloc`

- Είχαμε πει ότι η βιβλιοθήκη `<stdlib.h>` ορίζει **τρεις συναρτήσεις** (κλήσεις συστήματος) για να **εκχωρηθεί** μνήμη από τον πυρήνα του **Λειτουργικού Συστήματος** :
 - **`malloc`**: Δεσμεύει ένα block μνήμης (μετριέται σε bytes) αλλά **ΔΕΝ** τον αρχικοποιεί.
 - **`calloc`**: Δεσμεύει ένα block μνήμης και το **αρχικοποιεί** (με τον χαρακτήρα `'\0'`).
 - **`realloc`**: Αλλάζει το μέγεθος ενός προηγούμενου δεσμευμένου μπλοκ μνήμης.
- Ας δούμε λίγο αναλυτικότερα τα πρότυπα των συνάρτησαν αυτών.



Η Συνάρτηση `calloc`

- Πρότυπο `malloc`:

```
void *malloc(size_t size);
```

- Πρότυπο `calloc`:

```
void *calloc(size_t nmemb, size_t size);
```

- `typedef struct { int x, y; } POINT; POINT *p = NULL;`
- `p = (POINT *) calloc(1, sizeof(POINT));`

- Χαρακτηριστικά `calloc`:

- Δεσμεύει χώρο για ένα πίνακα με `nmemb` στοιχεία, το κάθε ένα με εκ των οποίων είναι `size bytes` μακρύ.
- Επιστρέφει ένα `NULL` δείκτη εάν δεν υπάρχει ο αιτούμενος χώρος.
- Αρχικοποιεί τον δεσμευμένο χώρο θέτοντας όλα τα bits σε 0.



Η Συνάρτηση `realloc`

- Πρότυπο `realloc`:

```
void *realloc(void *ptr, size_t size);
```

- **ΛΑΘΟΣ ΧΡΗΣΗ** (αύξηση πίνακα κατά 5 θέσεις)

```
ip = realloc(ip, sizeof(int) * (SIZE + 5));
```

- **ΟΡΘΗ ΧΡΗΣΗ** (αύξηση πίνακα κατά 5 θέσεις)

```
int *tmp = realloc(ip, sizeof(int) * (SIZE + 5));  
if (tmp == NULL) {  
    fprintf(stderr, "ERROR: realloc failed");  
}  
ip = tmp;
```

Δημιουργία νέου δείκτη,
δυναμική καταχώρηση
μνήμης σε νέα θέση,
αντιγραφή περιεχομένου, free
της προηγούμενης μνήμης



Η Συνάρτηση `realloc`

- Χαρακτηριστικά `realloc`:

```
void *realloc(void *ptr, size_t size);
```

- **Αναπροσαρμόζει** το μέγεθος ενός προηγούμενα δεσμευμένου πίνακα (μετακινώντας τον πίνακα σε νέο σημείο της μνήμης εάν χρειάζεται).
 - Μόλις επιστραφεί η `realloc`, βεβαιωθείτε ότι έχετε ενημερώσει όλους τους δείκτες στο μπλοκ μνήμης σε περίπτωση που έχει μετακινηθεί.
- **`ptr`** πρέπει να δείχνει σε διεύθυνση που λήφθηκε από προηγούμενη κλήση `malloc`, `calloc`, ή `realloc`.
- Το `size` **μπορεί** να είναι **μεγαλύτερο** ή **μικρότερο** (με όποιες συνέπειες ακολουθούν) από το αρχικό μέγεθος



Επιπρόσθετες Συναρτήσεις

bzero, bcopy, memcpy, memset

- Συναρτήσεις για αρχικοποίηση ακολουθιών *bytes* στη μνήμη.

void bzero(char *buf, int count);

- Θέτει 0 σε *count* bytes αρχίζοντας από τη διεύθυνση *buf*.
- Ίδιο με την ANSI C εντολή:
 - `void * memset (void * ptr, int value, size_t num);`

void bcopy(char *buf1, char *buf2, int count);

- Αντιγράφει *count* bytes αρχίζοντας από τη διεύθυνση *buf1* στη διεύθυνση *buf2*.
- Ίδιο με την ANSI C εντολή:
 - `void * memcpy (void * dest, const void * src, size_t num);`.

