Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Διάλεξη #7

Παναγιώτης Παύλου

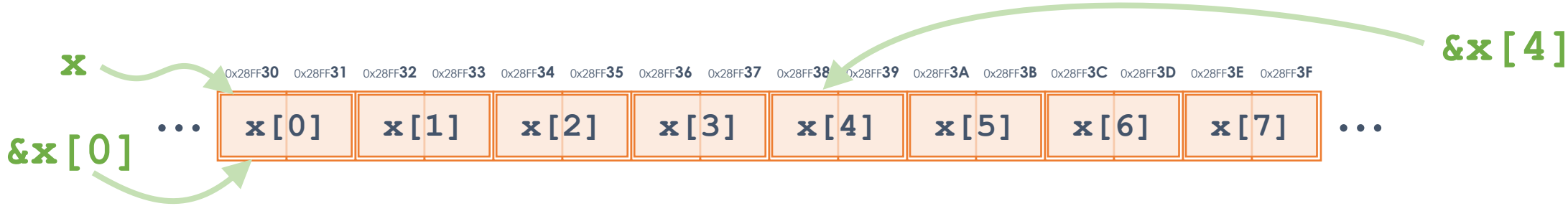
intro-hy-24@allos.gr

Δείκτες σε πίνακες

Πως εφαρμόζονται οι δείκτες στους πίνακες - Μια ιδιαίτερη σχέση

Αποθήκευση πινάκων στη μνήμη

Οι πίνακες, όπως έχει ήδη αναφερθεί από το σχετικό μάθημα, έχουν τα στοιχεία τους αποθηκευμένα στη μνήμη διαδοχικά. Δείτε το παράδειγμα του `short int x[8]` στο σχήμα:



Οι δείκτες στα μεμονωμένα στοιχεία, τα οποία λειτουργούν (όπως ξέρουμε) ως μεταβλητές, λαμβάνονται κατά τον γνωστό τρόπο.

Επίσης ως δείκτης σε πίνακα ορίζεται - κατ' αναλογία με τα προηγούμενα - η πρώτη διεύθυνση στην οποία είναι αποθηκευμένο το πρώτο στοιχείο του πίνακα, δηλαδή ο δείκτης στο πρώτο στοιχείο του πίνακα.

Προσοχή όμως, το όνομα του πίνακα **είναι** ο δείκτης στον πίνακα, έτσι δεν πρέπει να χρησιμοποιηθεί ο τελεστής & στο όνομα του πίνακα και κατά συνέπεια ισχύει πάντα:

```
x == &x[0]
```

Σημείο προσοχής!

Αφού το όνομα του πίνακα ταυτίζεται με τον δείκτη του πίνακα, αυτό σημαίνει ότι και ο τύπος δεδομένων του ταυτίζεται με δείκτη σε κατάλληλο τύπο δεδομένων. Π.χ. για έναν πίνακα ακεραίων:

```
int a[10];
int *b;

b = a;

b[4] = 888;
printf("%d\n", a[4]); // εμφανίζει 888
```

άρα ο δείκτης σε κάποιο τύπο δεδομένων συμπίπτει ως χρήση με πίνακα στον ίδιο τύπο δεδομένων.

Προσοχή όμως! Είναι ευθύνη του προγραμματιστή να μην τα χρησιμοποιήσει λανθασμένα.

Ως συνεπακόλουθο είναι οι δύο παρακάτω δηλώσεις να είναι (σχεδόν) ισοδύναμες:

```
int a[];
int *a;
```

Αριθμητική Pointers 1/2

Μέχρι αυτό το σημείο δεν έχουμε αναφερθεί παρά μόνο σε εκχωρήσεις τιμών σε pointers και χρήση τους.

Μια μεταβλητή τύπου pointer δεν διαφέρει στη λογική της από άλλες μεταβλητές, είναι δηλαδή αρχικά ακαθόριστη, άρα δείχνει δηλαδή σε άγνωστο κάθε φορά σημείο της μνήμης.

Εάν κατά τη δήλωση ενός pointer δεν είναι γνωστή κάποια τιμή για να εκχωρηθεί, τότε θα πρέπει να δίνεται η μηδενική.

```
int *A = NULL;
```

Η ειδική τιμή NULL είναι το μηδέν μαζί με έναν τύπο δεδομένων pointer. Αυτή ορίζεται στο stddef.h (το οποίο γίνεται έμμεσα include από σχεδόν όλα τα άλλα γνωστά includes) και στο σχετικό σημείο του γράφει:

```
#define NULL ((void *)0)
```

Η πρακτική αυτή βοηθά με τον εξής τρόπο:

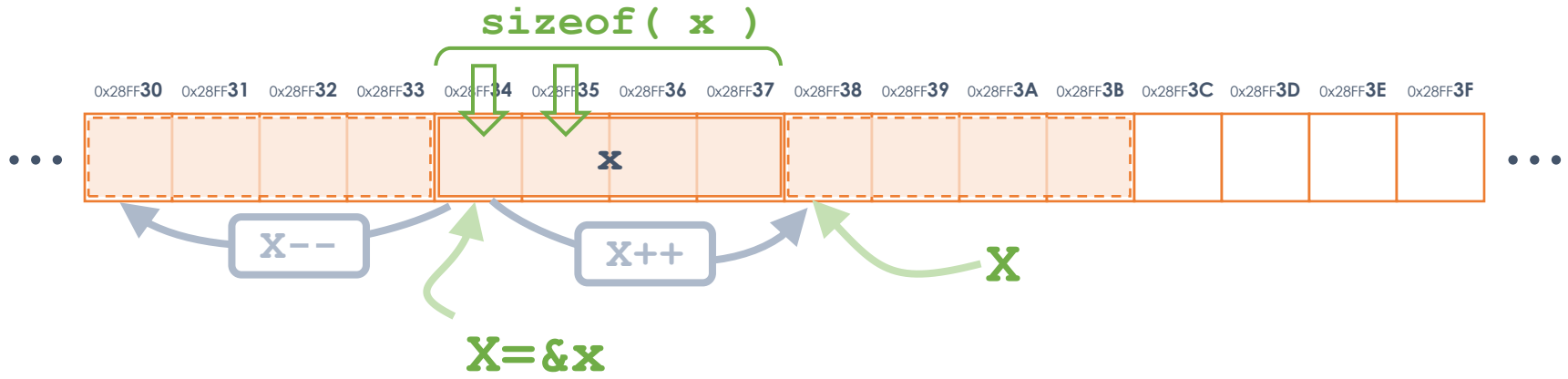
- Εάν χρησιμοποιηθεί η αρχική, ακαθόριστη τιμή του δείκτη τότε θα προκύψει ένα επίσης ακαθόριστο αποτέλεσμα, που στην **καλύτερη** περίπτωση θα είναι να τερματιστεί ανώμαλα το πρόγραμμα (να crashάρει όπως λέμε). Χειρότερα όμως μπορεί να λειτουργεί παράγοντας ασυνάρτητα αποτελέσματα.
- Ενώ με την ειδική τιμή **NULL** είναι δεδομένο ότι θα προκύψει ανώμαλος τερματισμός και ενδεχομένως με την ένδειξη *NULL pointer exception*. Άρα τουλάχιστον το πρόγραμμα δεν θα συμπεριφέρεται αλλοπρόσαλλα παράγοντας τυχαία, διαφορετικά σφάλματα σε κάθε εκτέλεση.

Αριθμητική Pointers 2/2

Επιτρέπονται αριθμητικές πράξεις με τους δείκτες; Ναι, αλλά μόνο οι προσθετικές. Όμως από αυτές οι σχεδόν αποκλειστικά χρησιμοποιούμενες είναι οι ++ και −− και αυτές μόνο θα δούμε παρακάτω.

Η αύξηση (ή η μείωση) της τιμής ενός δείκτη κατά μία θέση μνήμης δεν θα είχε πρακτική χρήση, καθώς (με την εξαίρεση του char) θα έδειχνε πλέον ο δείκτης σε θέση μνήμης που δεν έχει κάποια χρήσιμη πληροφορία.

Έτσι στην πραγματικότητα οι τελεστές ++ και −− μεταβάλουν την τιμή του δείκτη κατάλληλα, ώστε πλέον να δείχνει στην αμέσως επόμενη ή προηγούμενη θέση μιας τέτοιας μεταβλητής.



Αυτό έχει ιδιαίτερο νόημα όταν ο δείκτης χρησιμοποιείται στα πλαίσια ενός πίνακα, όπου με την κάθε αύξηση (++) ο δείκτης δείχνει στο επόμενο στοιχείο του πίνακα και αντίθετα με τη μείωση (−−) στο προηγούμενο.

Η ποσότητα της μεταβολής αυτής σε bytes είναι όσο το μέγεθος του τύπου δεδομένων. Το μέγεθος ενός τύπου δεδομένων δίνεται με τον τελεστή **`sizeof`**`( )` ο οποίος εφαρμόζεται είτε σε μεταβλητές αυτού του τύπου, είτε στον ίδιο τον τύπο δεδομένων.

`sizeof``( x )` ή **`sizeof`**`( int )`

Προτεραιότητα των νέων τελεστών

Στον πίνακα προτεραιότητας των τελεστών μπορείτε να δείτε τις προτεραιότητες όλων των τελεστών. Εδώ βλέπουμε τις σχετικές προτεραιότητες των τελεστών αυτής της ενότητας.

Οι προτεραιότητες συχνά μας μπερδεύουν έτσι καλό είναι να χρησιμοποιούμε παρενθέσεις αφού δεν κοστίζουν υπολογιστικά τίποτα.

Τελεστής	Προτεραιότητα
++ -- (ως επίθεμα)	1
[ ]	
++ -- (ως πρόθεμα)	2
*	
&	
sizeof	

π.χ. η έκφραση `*somePointer++` μπορεί να μας προβληματίζει, οπότε ανάλογα το τι θέλουμε καλύτερα να γράφουμε:

```
(*somePointer)++
```

ή

```
*(somePointer++)
```

Ερωτήσεις?

- Διαβάστε τις σημειώσεις, διαβάστε τις διαφάνειες και δείτε τα videos **πριν** ρωτήσετε
- **Συμβουλευτείτε** τη σελίδα ερωταποκρίσεων του μαθήματος

  https://qna.c-programming.allos.gr
- **Στείλτε** τις ερωτήσεις σας πριν και μετά το μάθημα στο

  intro-hy-24@allos.gr
- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε τον κώδικα ως κείμενο με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Εφαρμογές δεικτών - 1ο μέρος

Ποιός ο λόγος που μας ενδιαφέρουν οι δείκτες;

Κείμενα και pointers

Καθώς η αναπαράσταση των κειμένων γίνεται μέσω πινάκων στη C, αλλά και οι πίνακες είναι ισοδύναμοι με τους pointers, όλες οι συναρτήσεις που αφορούν κείμενα έχουν τις παραμέτρους τους δηλωμένες ως `char *` και όχι ως `char []` που βλέπαμε στο προηγούμενο μάθημα.

Δείτε για παράδειγμα πόσο πιο απλός είναι ο κώδικας στη συνάρτηση που επιστρέφει το κείμενο με τους περισσότερους χαρακτήρες.

Δείτε επίσης την `stringLength` που είναι παραλλαγή της `calculateLength` του προηγούμενου μαθήματος, αλλά με pointer.

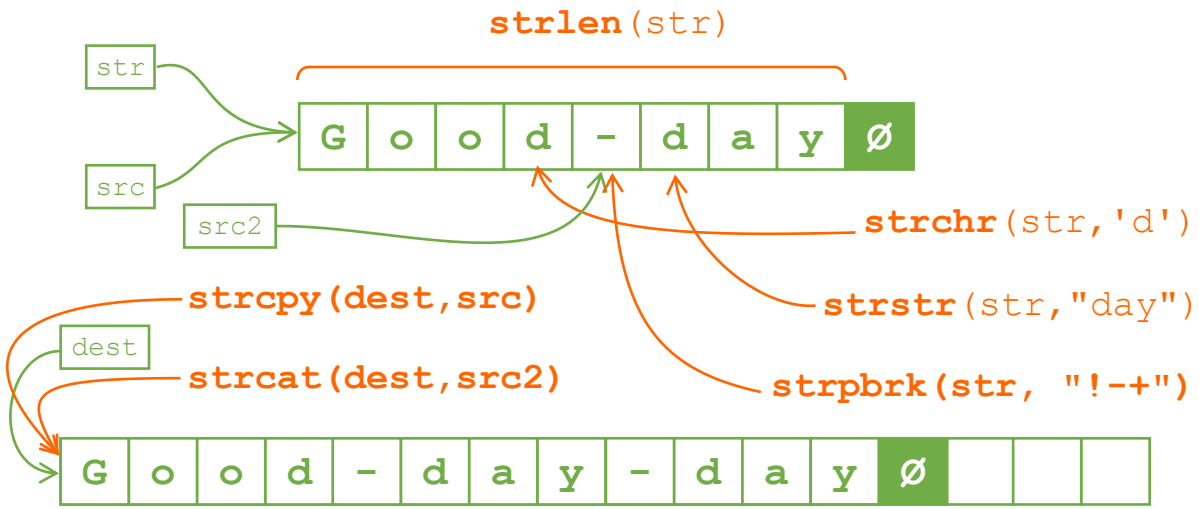
```
int stringLength(char *text) {
    int len=0;
    while (*text != 0) {
        text++;
        len++;
    }
    return len;
}

char *longestText(char *text1, char *text2) {
    if (stringLength(text2) > stringLength(text1))
        return text2;
    return text1;
}

int main() {
    char *t1 = "Goodday!", *t2 = "Goodnight!";
    printf("A. %s %d\n", t1, stringLength(t1));
    printf("B. %s %d\n", t2, stringLength(t2));
    printf("Longer : %s\n", longestText(t1,t2));
    return 0;
}
```

string.h

Η βιβλιοθήκη string έχει αρκετές έτοιμες συναρτήσεις για τη διαχείριση των κειμένων.

Πλήρη λίστα μπορείτε να δείτε εδώ και τις πιο σημαντικές θα τις βρείτε στο σχετικό παράρτημα των σημειώσεων.

Πολύ συχνά χρησιμοποιούμενες είναι αυτές που παραθέτουμε στα δεξιά.



`size_t strlen(char *str)`	Επιστρέφει το μήκος της συμβολοσειράς που δίνεται ως παράμετρος.
`char *strchr(char *str, int c)`	Βρίσκει τον χαρακτήρα c στο κείμενο str ή επιστρέφει NULL εάν δεν βρεθεί
`char *strstr( char *haystack, char *needle)`	Βρίσκει την πρώτη εμφάνιση του δεύτερου κειμένου μέσα στο πρώτο και επιστρέφει τον δείκτη σε εκείνο το σημείο ή NULL εάν δεν υπάρχει.
`char *strpbrk( char *str1, char *str2)`	Βρίσκει τον πρώτο χαρακτήρα στο πρώτο κείμενο (str1) από αυτούς που υπάρχουν στο δεύτερο κείμενο και επιστρέφει τον δείκτη σε εκείνο το σημείο ή NULL αν δεν υπάρχει κανένας.
`char *strcpy(char *dest, char *src)`	Αντιγράφει το 2ο κείμενο πάνω στο 1ο (υποθέτοντας ότι υπάρχει επαρκής χώρος)
`char *strcat(char *dest, char *src)`	Ενώνει δύο κείμενα στη θέση του πρώτου. Δηλαδή μετά το τέλος του 1ου κειμένου αντιγράφει το 2ο κείμενο.
`int strcmp(char *str1, char *str2)`	Συγκρίνει τα δύο κείμενα και επιστρέφει 0 εάν είναι ίσα, θετική τιμή εάν το 1ο κείμενο είναι λεξικογραφικά μεγαλύτερο από το 2ο και αρνητική εάν είναι μικρότερο.

`strcmp(str, dest)` ⟶ `-1`

Επόμενη Λέξη

Το ζητούμενο είναι να γράψουμε μία συνάρτηση, την:

```
char *nextWord(char *text)
```

η οποία να επιστρέφει τον pointer στο σημείο του κειμένου text που ξεκινάει η πρώτη λέξη. Εάν δεν βρεθεί κάποια λέξη τότε πρέπει να επιστρέφεται η τιμή NULL.

Ως λέξη θεωρούμε κάθε ομάδα διαδοχικών αλφαβητικών χαρακτήρων, όπως τους θεωρεί και η `isalpha`

Κατόπιν τροποποιήστε την παραπάνω συνάρτηση ώστε όταν αντί ως κείμενο text δίνεται η τιμή NULL να συνεχίζει την αναζήτηση στο προηγούμενο κείμενο που είχε δοθεί, για να βρει την επόμενη λέξη.

Τέλος τροποποιήστε την παραπάνω συνάρτηση η οποία να παίρνει και το όρισμα L όπως φαίνεται και μέσω αυτού να επιστρέφει το μήκος της λέξης που βρέθηκε. Εάν ως L δοθεί το NULL, τότε δεν χρειάζεται να υπολογίζει το μήκος της λέξης. Εάν όμως δοθεί το L και δεν βρεθεί λέξη, τότε ΔΕΝ θα πρέπει να αλλάζει η τιμή του.

```
char *nextWord(char *text, int *L)
```

Η παραπάνω διαδικασία της σταδιακής συγγραφής της συνάρτησης απεικονίζει την ανάπτυξη της επίλυσης των προβλημάτων. Επιλύοντάς τα ένα προς ένα μπορούμε να ελέγχουμε την ορθότητα του κώδικά μας.

Ερωτήσεις?

- Διαβάστε τις σημειώσεις, διαβάστε τις διαφάνειες και δείτε τα videos **πριν** ρωτήσετε
- **Συμβουλευτείτε** τη σελίδα ερωταποκρίσεων του μαθήματος

  https://qna.c-programming.allos.gr
- **Στείλτε** τις ερωτήσεις σας πριν και μετά το μάθημα στο

  intro-hy-24@allos.gr
- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε τον κώδικα ως κείμενο με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Δυναμική διαχείριση μνήμης

Πως συνεργαζόμαστε με το λειτουργικό σύστημα γι’αυτό

Δυναμική διαχείριση μνήμης – Stack και Heap

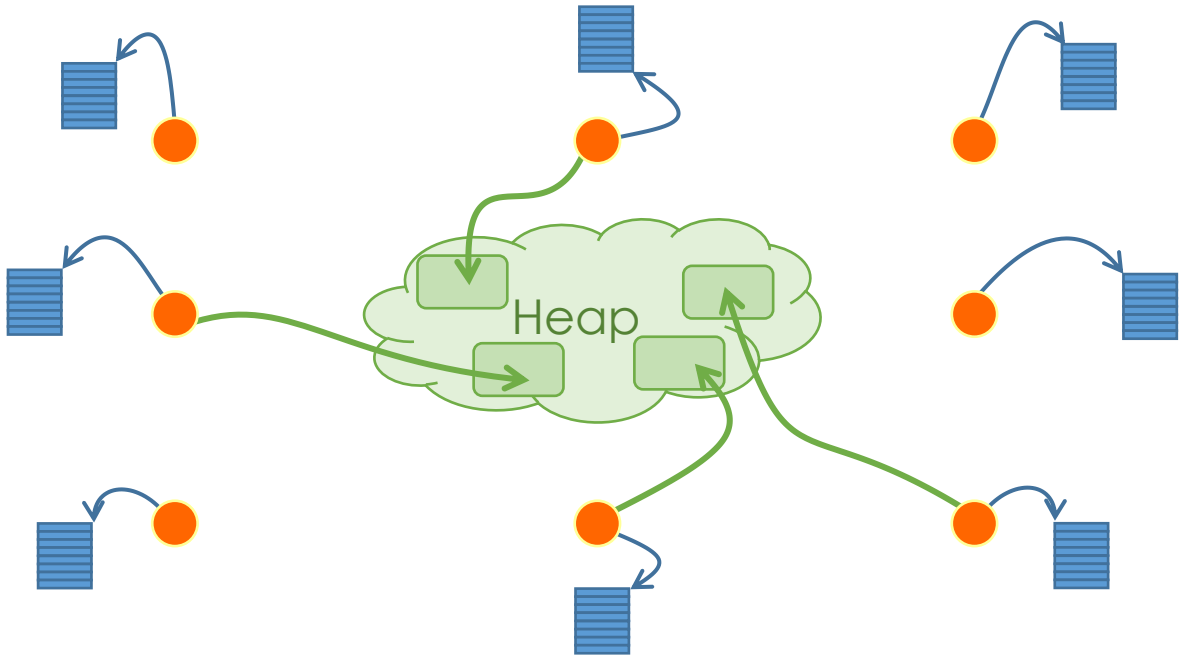
Μέχρι αυτό το σημείο έχουμε χρησιμοποιήσει τους pointers ώστε να δείχνουν σε πληροφορία, που αποθηκεύεται σε τμήματα της μνήμης του υπολογιστή, τα οποία προϋπάρχουν στη μνήμη επειδή έχουν δηλωθεί μέσα από τους γνωστούς μηχανισμούς της C.

Επίσης έχουμε δει ότι εάν έχουμε έναν pointer τότε μπορούμε να κάνουμε οτιδήποτε θα κάναμε και με τις απλές μεταβλητές (χωρίς τη χρήση των pointers).

Άρα αυτό που μας λείπει – με δεδομένους τους pointers – ώστε να χρησιμοποιούμε όλη τη λειτουργικότητα της γλώσσας, είναι να «δημιουργείται» ο χώρος στη μνήμη που καταλαμβάνει αυτόματα μία νέα μεταβλητή (είτε πίνακας, είτε δομή) όταν δημιουργείται.

Stack & Heap

Αυτό μας ενδιαφέρει για έναν επιπρόσθετο λόγο. Επειδή ο ίδιος ο επεξεργαστής υποστηρίζει μία δομή που λέγεται stack – ώστε να παρέχει πρόχειρη μνήμη στον κώδικα – όλες οι μεταβλητές της C δημιουργούνται μέσα σε αυτό. Όμως το μέγεθος του stack είναι αρκετά μικρό καθώς για κάθε ροή προγράμματος που ονομάζεται και thread (εύκολα σε έναν υπολογιστή ξεπερνούν τις 1000) υπάρχει ξεχωριστό stack, ενώ όλα τα stacks πρέπει να έχουν το ίδιο μέγεθος! Έτσι δεν μπορεί ο κώδικάς μας να αξιοποιήσει το μεγαλύτερο κομμάτι της μήνης του υπολογιστή.



Η υπόλοιπη μνήμη αποτελεί ένα ενιαίο κομμάτι που ονομάζεται heap και το οποίο το διαχειρίζεται το λειτουργικό σύστημα ώστε να παρέχει μνήμη στα προγράμματα που εκτελούνται. Ο μόνος περιορισμός είναι η μνήμη που έχει διαθέσιμη ο υπολογιστής.

Όλη η διαχείριση μνήμης αφορά τον χώρο του heap. Οι δύο βασικές λειτουργίες που μπορεί να κάνει το πρόγραμμά μας είναι είτε να ζητήσει (λέμε να **δεσμεύσει**) κάποια συγκεκριμένη ποσότητα μνήμης του heap, είτε να **απελευθερώσει** μνήμη την οποία είχε ήδη δεσμεύσει νωρίτερα.

Δέσμευση και απελευθέρωση μνήμης 1/4

Υπάρχουν 2+1 βασικές λειτουργίες που αφορούν τη διαχείριση της μνήμης. Είναι διαθέσιμα μέσω του stdlib.h:

- Αρχικά γίνεται η **δέσμευση**, η οποία βάσει του πλήθους των διαδοχικών bytes που απαιτούνται, επιστρέφει έναν pointer στην αρχή της περιοχής της μνήμης η οποία δεσμεύεται (ονομάζεται και block) ή εφόσον δεν υπάρχει διαθέσιμο κατάλληλο block επιστρέφει NULL. Αυτό γίνεται με την εντολή **`malloc`**:

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

- Όταν ολοκληρωθεί η χρήση ενός block το οποίο είχε δεσμευθεί, και εφόσον δεν πρόκειται να ξαναχρησιμοποιηθεί από τον κώδικα, πρέπει να αποδεσμευθεί (λέμε και **ελευθερωθεί**), ώστε να γίνει διαθέσιμο ξανά για δέσμευση. Αυτό γίνεται με τη χρήση της εντολής **`free`**:

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

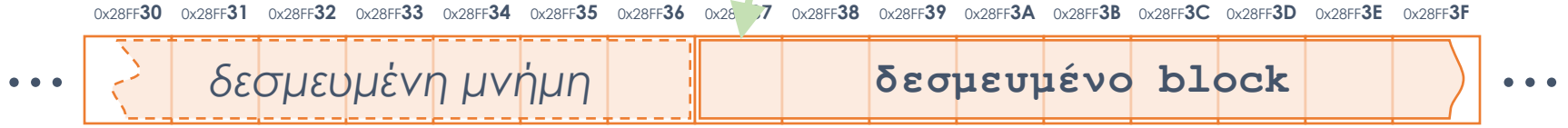
- Επιτρέπεται και η μεταβολή του μεγέθους για ήδη δεσμευμένη μνήμη, αυτό όμως θέλει προσοχή. Γίνεται με τη χρήση της εντολής **`realloc`**:

```
void *realloc(void *ptr, size_t new_size)
```

Δέσμευση και απελευθέρωση μνήμης 2/4

Η εντολή `malloc` είναι η πιο βασική εντολή από τις τρεις. Η μόνη παράμετρος που δέχεται είναι ο αριθμός των bytes που χρειάζεται ο κώδικας. Ο τύπος `size_t` δεν είναι εγγενής τύπος δεδομένων, στην πραγματικότητα είναι ένας απρόσημος ακέραιος κατάλληλος ώστε να χωρά αριθμό επαρκή για το address bus, του συστήματος για το οποίο προορίζεται. Η τιμή που επιστρέφει είναι είτε `NULL` εάν δεν είναι δυνατή η δέσμευση της ζητούμενης ποσότητας μνήμης, είτε ένας δείκτης στο πρώτο της byte. Τα περιεχόμενα της μνήμης είναι ακαθόριστα (ό,τι υπήρχε εκεί από προηγούμενη εκτέλεση). Ο δείκτης πρέπει να γίνεται **cast** στον επιθυμητό τύπο.

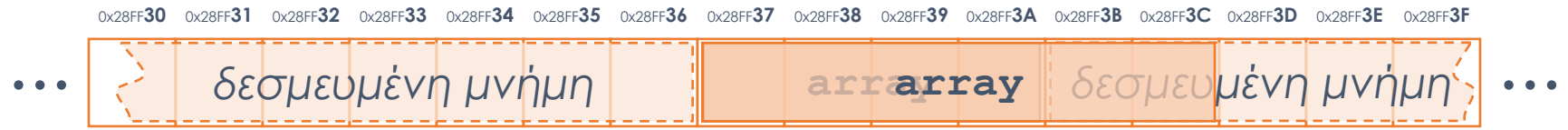
```
int *array = (int *)malloc(1000 * sizeof(int));
if (array == NULL) {
    // διαχείριση σφαλμάτων
}
```



Δέσμευση και απελευθέρωση μνήμης 3/4

Η εντολή `realloc` είναι παραλλαγή της `malloc` όπου δέχεται δύο ορίσματα. Το 2ο λειτουργεί όπως του `malloc`, το 1ο όμως είναι προϋπόθεση να είναι ένας δείκτης ο οποίος πρέπει να έχει επιστραφεί προηγουμένως από την `malloc` (ή προηγούμενη `realloc`). Επιστρέφει έναν pointer στη νέα περιοχή μνήμης.

- Εάν το ζητούμενο μέγεθος είναι μικρότερο από αυτό που είχε αρχικά δεσμευθεί ή εάν ζητηθεί μεγαλύτερο και υπάρχει η δυνατότητα για επέκτασή του στην ίδια θέση, τότε επιστρέφεται ο ίδιος δείκτης με τον δεδομένο.
- Αλλιώς εάν υπάρχει σε άλλο σημείο της μνήμης, τότε επιστρέφεται ένας δείκτης στη νέα θέση, αντιγράφονται τα δεδομένα της πρώτης περιοχής στη νέα και η παλιά μνήμη αποδεσμεύεται.
- Τέλος εάν δεν υπάρχει καν δυνατότητα δέσμευσης της ζητούμενης μνήμης, επιστρέφεται NULL και η προηγούμενη μνήμη παραμένει ανεπηρέαστη.



Δέσμευση και απελευθέρωση μνήμης 4/4

Τέλος η εντολή `free` αποδεσμεύει μια περιοχή μνήμης που είχε προηγουμένως δεσμευθεί από τις `malloc`/`realloc`. Και δέχεται ως όρισμα έναν δείκτη που έχει προηγουμένως επιστραφεί από τις `malloc` ή `realloc`. Η μνήμη που αποδεσμεύεται πρέπει πλέον να μην χρησιμοποιηθεί και με ευθύνη του προγραμματιστή να διαγραφούν οι τιμές των pointers που έδειχναν σε οποιοδήποτε σημείο αυτής της περιοχής.

Επειδή οι αποδεσμεύσεις μπορεί να γίνουν με οποιαδήποτε σειρά η διαθέσιμη μνήμη μπορεί να φαίνεται επαρκής σε κάποιο σύστημα, όμως οι `malloc`/`realloc` να αποτυγχάνουν επειδή δεν υπάρχει συνεχόμενο το ζητούμενο μέγεθος της μνήμης. Αυτό ονομάζεται fragmentation (ή κατακερματισμός) της μνήμης.

Ερωτήσεις?

- Διαβάστε τις σημειώσεις, διαβάστε τις διαφάνειες και δείτε τα videos **πριν** ρωτήσετε
- **Συμβουλευτείτε** τη σελίδα ερωταποκρίσεων του μαθήματος

  https://qna.c-programming.allos.gr
- **Στείλτε** τις ερωτήσεις σας πριν και μετά το μάθημα στο

  intro-hy-24@allos.gr
- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε τον κώδικα ως κείμενο με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Resources ή Πόροι

Ως πόρο του συστήματος ονομάζουμε κάθε στοιχείο του συστήματος το οποίο – συγκριτικά με τη ζήτηση – είναι περιορισμένο σε διαθεσιμότητα.

Τέτοιο μπορεί να είναι η επεξεργαστική ισχύς, η πρόσβαση σε έναν δίσκο ή η μνήμη του υπολογιστή, η πρόσβαση στο δίκτυο, κ.α.

Κάθε πόρος συνήθως συνοδεύεται από ένα σύστημα διαχείρισης. Για παράδειγμα ως προς την μνήμη είδαμε τον μηχανισμό δέσμευσης/αποδέσμευσης.

Το γενικό σχήμα που ακολουθείται είναι η **δέσμευση** ή **άνοιγμα** για χρήση, ενός μέρους του πόρου και η επιστροφή κάποιου ενδεικτικού μεγέθους (π.χ. ο pointer στη μνήμη).

Ακολουθεί η **χρήση** του πόρου (η οποία συνήθως είναι αποκλειστική από αυτό τον συγκεκριμένο πρόγραμμα) και όταν η χρήση ολοκληρωθεί, ακολουθεί η **αποδέσμευση**. Μετά από την αποδέσμευση απαγορεύεται η χρήση του πόρου ακόμα και αν το ενδεικτικό μέγεθος παραμένει γνωστό στον κώδικα.

Αυτό το σχήμα, χρησιμοποιείται σε κάθε περίπτωση πόρου. Επίσης μπορεί κάτι που δεν είναι άμεσα πόρος, αλλά χρησιμοποιεί πόρους, να ακολουθεί και αυτό το ίδιο σχήμα. Πχ ανοίγω/χρησιμοποιώ/κλείνω ένα:

- παράθυρο στην οθόνη
- session σε μια TCP σύνδεση δικτύου
- κλειδί (από)κρυπτογράφησης

Ερωτήσεις?

- Διαβάστε τις σημειώσεις, διαβάστε τις διαφάνειες και δείτε τα videos **πριν** ρωτήσετε
- **Συμβουλευτείτε** τη σελίδα ερωταποκρίσεων του μαθήματος

  https://qna.c-programming.allos.gr
- **Στείλτε** τις ερωτήσεις σας πριν και μετά το μάθημα στο

  intro-hy-24@allos.gr
- Εάν έχετε **πρόβλημα** με κάποιο κώδικα στείλτε τον κώδικα ως κείμενο με copy/paste. Εάν θεωρείτε ότι επιπλέον βοηθά και ένα στιγμιότυπο οθόνης, είναι καλοδεχούμενο.
- Επαναλαμβάνουμε : Μην στείλετε ποτέ κώδικα ως εικόνα μας είναι παντελώς άχρηστος!

Σημαντικά σημεία

Μετά από τη σημερινή διάλεξη θα πρέπει να γνωρίζετε:

- Δείκτες σε μονοδιάστατους πίνακες
- Βασική αριθμητική των δεικτών
- Τον τελεστή sizeof
- Τη χρήση των κειμένων μέσω pointers
- Το σύστημα διαχείρισης μνήμης
- Την έννοια του πόρου ενός συστήματος