Εισαγωγή στην Πληροφορική & στον Προγραμματισμό

Αρχές Προγραμματισμού Η/Υ (με τη γλώσσα C)

Διάλεξη #8

31 Μαΐου 2024

Παναγιώτης Παύλου

c-programming-24@allos.gr

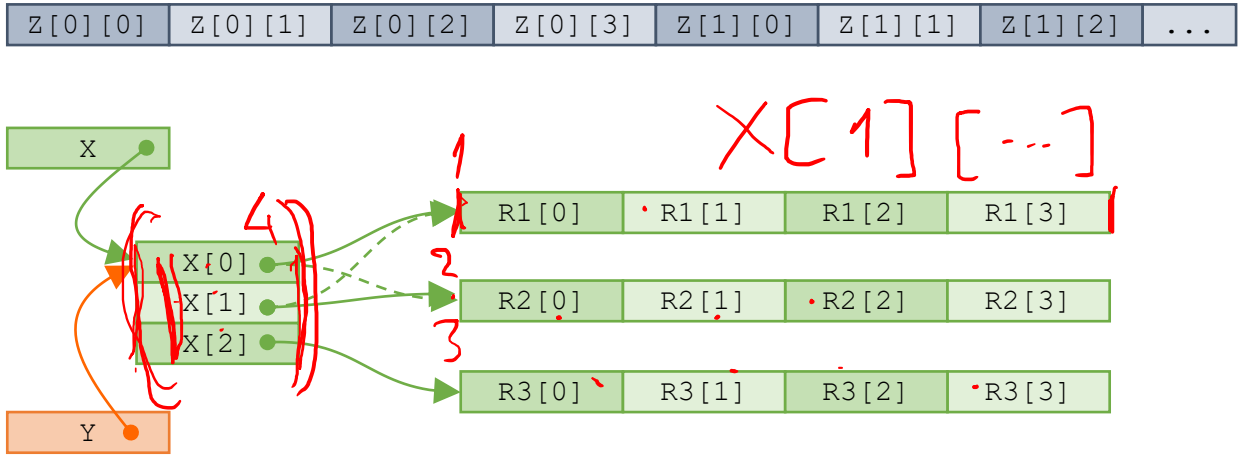
Εφαρμογές δεικτών - 2ο μέρος

Ποιός ο λόγος που μας ενδιαφέρουν οι δείκτες;

Πολυδιάστατοι πίνακες και pointers

Η δυνατότητα να γραφτούν δείκτες σε άλλους δείκτες, αλλά και ο δυϊσμός πίνακα και δείκτη, μας επιτρέπουν να γραφτεί ο δίπλα κώδικας.

Στο διπλό βρόχο στο κάτω μέρος οι X, Y και Z μπορούν να χρησιμοποιηθούν εναλλάξιμα, όμως η δήλωσή τους και η εσωτερική τους δομή δεν είναι η ίδια!




```
double Z[3][4] = {
    { 1,2,3,4 },
    { 5,6,7,8 },
    { 9,10,11,12 }
};

double R1[] = { 1.,2.,3.,4. };
double R2[] = { 5.,6.,7.,8. };
double R3[] = { 9.,10.,11.,12. };
double *X[] = { R1, R2, R3 };
double **Y = X;

for (int row = 0; row < 3; ++row) {
    for (int col = 0; col < 4; ++col) {
        printf("%5.2lf ", Z[row][col]);
    }
    printf("\n");
}
```

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

  c-programming-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!

Δείκτες σε δομές δεδομένων

Πως εφαρμόζονται οι δείκτες στις δομές δεδομένων

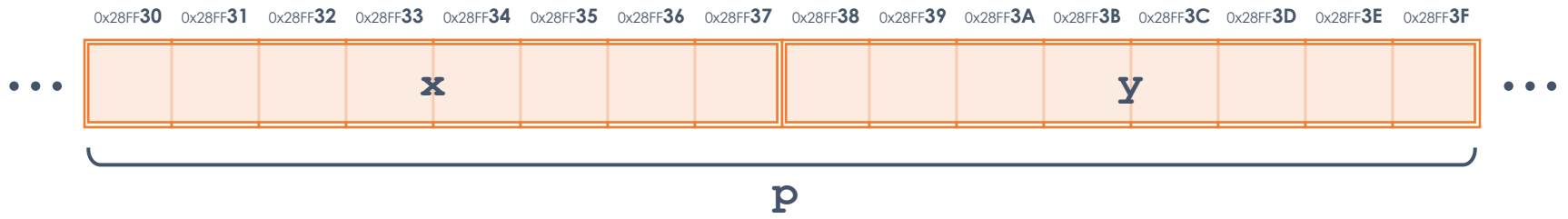
Αποθήκευση δομών δεδομένων στη μνήμη

Οι δομές δεδομένων (struct) αποτελούν ομαδοποιήσεις μεταβλητών σχετικές μεταξύ τους. Έτσι όταν δηλώνεται μια μεταβλητή τέτοιου τύπου, τα πεδία της βρίσκονται αποθηκευμένα συνεχόμενα στη μνήμη.

Δείτε το παράδειγμα:

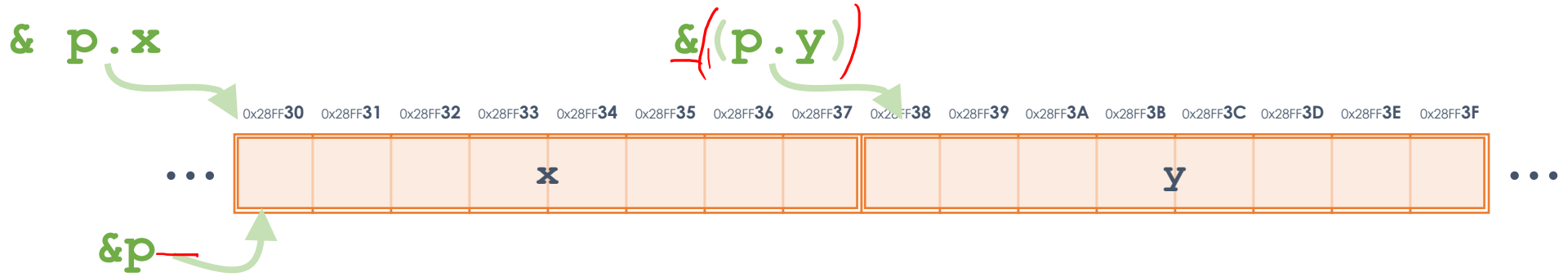
```
typedef struct _point2d {
    double x;
    double y;
} Point2D ;
...
Point2D p;
```

Η σειρά με την οποία αποθηκεύονται συμπίπτει με τη σειρά την οποία δηλώνονται.



Pointers σε δομές δεδομένων

Τα πεδία μιας δομής δεδομένων λειτουργούν ως απλές μεταβλητές. Άρα μπορεί με τη χρήση του τελεστή & να ληφθεί ο pointer σε αυτές.



Με βάση το παράδειγμα της προηγούμενης διαφάνειας, οι pointers στα πεδία x και y γράφονται απλά όπως παραπάνω. Λόγω της υψηλότερης προτεραιότητας που έχει ο τελεστής . ως προς τον & οι παρενθέσεις που παρουσιάζονται περισσεύουν.

Όσον αφορά την ίδια τη μεταβλητή `p`, όπως και στις απλές μεταβλητές, ο δείκτης σε αυτή ταυτίζεται με την πρώτη διεύθυνση της μνήμης η οποία χρησιμοποιείται. Οπότε η παρακάτω ισότητα ισχύει πάντα για το πρώτο πεδίο της (εδώ το x).

```
&p == &p.x
```

Χρήση pointers δομών δεδομένων

Ο δείκτης στη δομή δεδομένων δηλώνεται όπως και αυτός των απλών μεταβλητών. Άρα μπορούμε να γράψουμε:

```
Point2D p;
Point2D *Pp = &p;
```

Όμως η αναφορά σε ένα στοιχείο με δεδομένο των pointer θέλει **προσοχή** στη γραφή της:

~~`*Pp.x`~~ `(*Pp).x`

καθώς ο τελεστής **.** έχει υψηλότερη προτεραιότητα από τον * επειδή όμως αυτό δυσκολεύει την αναγνωσιμότητα του κώδικα, γι'αυτό υπάρχει ο ισοδύναμος τελεστής ->

`Pp->x` *ή* `Pp->y`

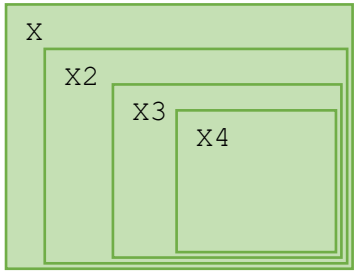
Παράδειγμα χρήσης

Όταν γίνεται κλήση μιας συνάρτησης με όρισμα μία δομή, επειδή η κλήση γίνεται με call by value, όλα τα πεδία της δομής του ορίσματος αντιγράφονται στα αντίστοιχα πεδία της παραμέτρου και (επειδή τα δεδομένα μιας δομής μπορεί να είναι πάρα πολλά) αυτό μπορεί να προκαλέσει μη αμελητέα καθυστέρηση.

Πέρα από αυτόν τον λόγο υπάρχουν και άλλοι σπουδαιότεροι που θα δούμε αργότερα, όμως όταν μία συνάρτηση χρειάζεται πρόσβαση στα στοιχεία μιας δομής συνηθίζεται να δίνεται στη συνάρτηση ο δείκτης στη δομή.

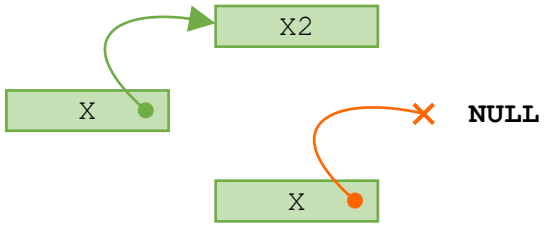
```
double distance(
    Point2D *A,
    Point2D *B
) {
    double dx = B->x - A->x;
    double dy = B->y - A->y;
    return sqrt(
        dx*dx + dy*dy
    );
}
```

Αυτοαναφορική δομή



Όταν συναντήσαμε τις δομές είχαμε αναφερθεί στον περιορισμό που υπάρχει ότι μία δομή δεν πρέπει να περιέχει πεδίο με τύπο δεδομένων αυτής της ίδιας.

Όμως με τη χρήση των pointers ο περιορισμός αυτός παρακάμπτεται. Αυτό συμβαίνει επειδή ο pointer δεν δείχνει υποχρεωτικά σε πληροφορία (μεταβλητή, πίνακα ή δομή δεδομένων) καθώς μπορεί να έχει και την τιμή NULL. Αλλά και όταν δείχνει, αναφέρεται σε υπάρχουσα πληροφορία «έξω» από το συγκεκριμένο στιγμιότυπο της δομής.



Δηλαδή ο pointer μας παρέχει <u>τη **δυνατότητα** να υπάρχει η σύνδεση με την πληροφορία</u>, αλλά <u>δεν εξασφαλίζει, ούτε απαιτεί</u> την ύπαρξη της πληροφορίας.

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

  c-programming-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

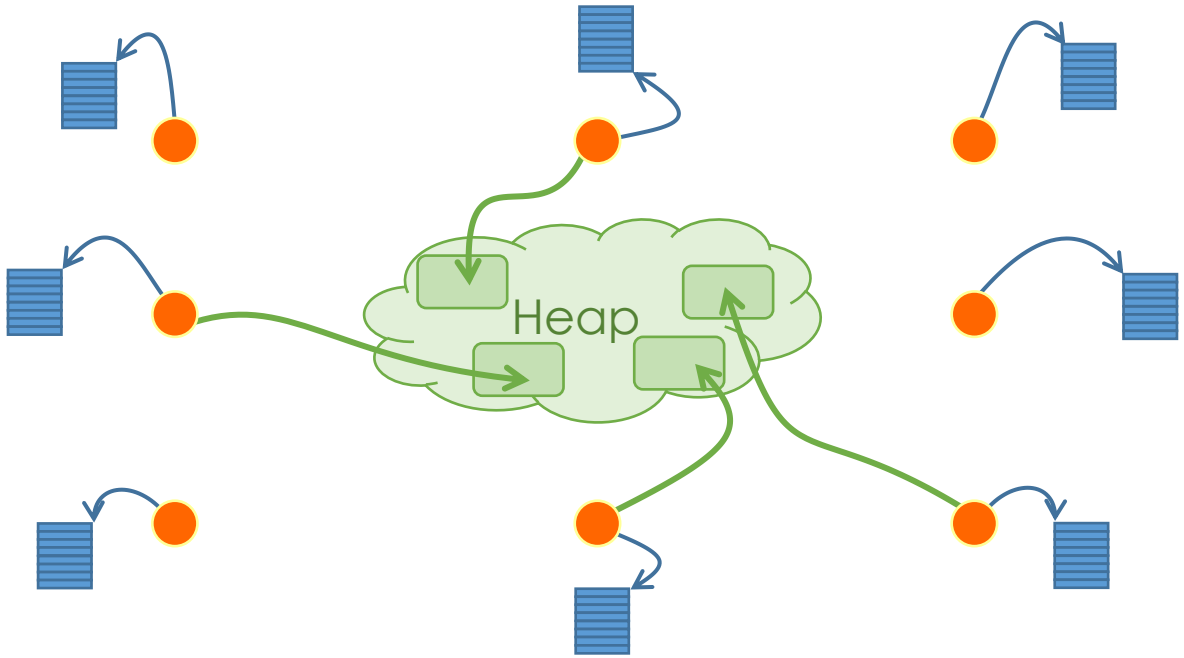
Μέχρι αυτό το σημείο έχουμε χρησιμοποιήσει τους pointers ώστε να δείχνουν σε πληροφορία, που αποθηκεύεται σε τμήματα της μνήμης του υπολογιστή, τα οποία προϋπάρχουν στη μνήμη επειδή έχουν δηλωθεί μέσα από τους γνωστούς μηχανισμούς της C.

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. Υλοποιούνται 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)
```

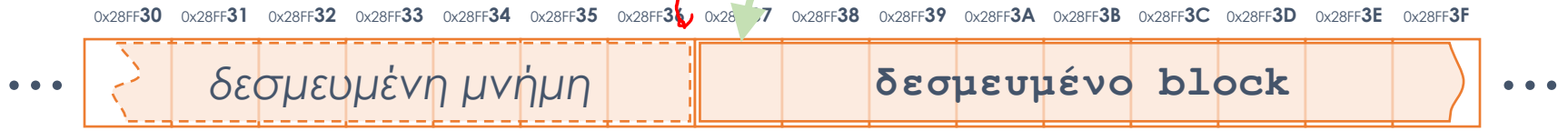
- Επιτρέπεται και η μεταβολή του μεγέθους για ήδη δεσμευμένη μνήμη, αυτό όμως θέλει προσοχή. Γίνεται με τη χρήση της εντολής **realloc**:

```
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 για το σύστημα 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 pointer.

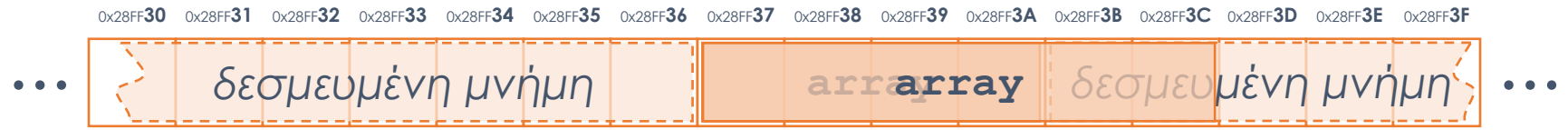
```
int *array = (int *)malloc(1000 * sizeof(int));
if (array == NULL) {
    // διαχείριση σφαλμάτων
}
```



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

  c-programming-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 κώδικα) 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.

Αυτή η λογική επεκτείνεται και αντικατοπτρίζεται και στη δομή του κώδικα όταν αυτός αφορά πληροφορία ή δράση που εξαρτάται από έναν πόρο.

Συναρτήσεις Constructor

Είχαμε δει ότι με τη χρήση των struct ομαδοποιούμε συναφείς μεταβλητές, οι οποίες πολλές φορές περιγράφουν κάποια έννοια ή οντότητα. Έτσι μπορεί να έχουμε μία δομή που να περιγράφει έναν αλγεβρικό πίνακα ή μία δομή που να περιγράφει ένα σωματίδιο ή μία που να περιγράφει έναν μιγαδικό αριθμό.

Συνδυάζοντας το παραπάνω με τους pointers και την δέσμευση μνήμης, συνηθίζεται – και είναι μία πολύ καλή πρακτική – το να γράφονται συναρτήσεις οι οποίες «δημιουργούν» δομές κατάλληλα αρχικοποιημένες και να επιστρέφουν τον pointer σε αυτές (ή NULL εάν δεν ήταν δυνατή η δημιουργία μιας τέτοιας δομής).

Αυτή η λογική αποτελεί το επόμενο βήμα (μετά τις δομές) προς τον αντικειμενοστραφή προγραμματισμό. Αυτές οι συναρτήσεις ονομάζονται Constructors.

Μέσα τους μπορεί να περιέχουν και τη δέσμευση άλλων πόρων εκτός από τη δέσμευση της μνήμης.

Συναρτήσεις Destructor

Μετά την ολοκλήρωση της χρήσης κάθε τέτοιας οντότητας, πρέπει να κληθεί μία συνάρτηση που «καταστρέφει» αυτή την «οντότητα». Αυτή ουσιαστικά απελευθερώνει όλους τους πόρους που είχαν δεσμευθεί από τον constructor και την υπόλοιπη χρήση της.

Οι συναρτήσεις αυτές ονομάζονται destructors και είναι πολύ σημαντικό να μην αμελεί να τις καλεί ο προγραμματιστής καθώς χωρίς αυτές δεν αποδεσμεύονται οι πόροι και το σύστημα μπορεί να σταματήσει να λειτουργεί από την έλλειψή τους.

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

  c-programming-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!

Εφαρμογή

Μία συγκεντρωτική εφαρμογή όλων των παραπάνω είναι η δημιουργία ενός constructor και ενός destructor για αλγεβρικούς πίνακες.

- Ορίστε μια δομή που να περιγράφει έναν αλγεβρικό πίνακα
- Δημιουργήστε έναν constructor για μηδενικό αλγεβρικό πίνακα
- Δημιουργήστε έναν destructor για έναν αλγεβρικό πίνακα
- Δημιουργήστε έναν constructor για μοναδιαίο αλγεβρικό πίνακα

Σημαντικά σημεία

Μετά από τη σημερινή διάλεξη θα πρέπει να γνωρίζετε:

- Τη χρήση των πολυδιάστατων πινάκων
- Το σύστημα διαχείρισης μνήμης
- Την έννοια του πόρου ενός συστήματος
- Την πρακτική χρήση constructors και destructors