ΨΗΦΙΑΚΟΙ ΥΠΟΛΟΓΙΣΤΕΣ

ΑΝΑΦΟΡΑ ΕΡΓΑΣΤΗΡΙΟΥ 1

ΜΙΧΑΛΗΣ ΓΑΛΑΝΗΣ 2016030036
ΓΙΩΡΓΟΣ ΒΙΡΙΡΑΚΗΣ 2016030035

Προεργασία

Η άσκηση 1 προαπαιτούσε βασικές γνώσεις διαχείρησης μνήμης και γλώσσας προγραμματισμού C. Έπρεπε να κατανοήσουμε πόσο χώρο καταλαμβάνει η κάθε μεταβλητή, δείκτης ή πίνακας μεταβλητών καθώς και πως διαβάζουμε ή μεταβάλλουμε μία μεταβλητή μέσω δείκτη.

Περιγραφή Ζητούμενων

Αυτή η άσκηση αποσκοπούσε στην μελέτη της συμπεριφοράς της γλώσσας C ανάλογα με το υλικό (hardware) στο οποίο εκτελείται. Βοήθησε επίσης στην εξάσκηση μας με τις διάφορες περιοχές της μνήμης (heap, static, stack) και στη δεκαεξαδική αναπαράσταση των μεταβλητών στις θέσεις αυτές.

- Στο πρώτο ερώτημα μας ζητήθηκαν γενικές παρατηρήσεις σχετικά με την αποθήκευση διαδοχικών μεταβλητών, διαφορές σε εκφράσεις σε κάποιο στοιχείο του πίνακα, και τη δεκαεξαδική αναπαράσταση του στη μνήμη.
- Το δεύτερο ζητούμενο ήταν η κατανόηση μας σχετικά με το χώρο που καταλαμβάνουν διαδοχικές μεταβλητές ανάλογα με το είδος τους.
- Η τρίτη ερώτηση εξέταζε μια ενδιαφέρουσα συμπεριφορά της C που είχε να κάνει με τον τρόπο που αποθήκευε τις μεταβλητές στη μνήμη ανάλογα με τη σειρά δήλωσής τους.
- Στο τέταρτο ερώτημα μας ζητούσε να παρατηρήσουμε τι γίνεται όταν δεσμεύσουμε δυναμικά μνήμη για 4 διαφορετικές περιπτώσεις.
- Η τελευταία περίπτωση είχε να κάνει με ποιές περιοχές μνήμης δεσμεύονται ανάλογα με το αν η μεταβλητή είναι τοπική, καθολική, σταθερή ή αν υπάρχει δυναμική δέσμευση μνήμης. Μας ζητούσε επιπλέον να κατασκευάσουμε ενα χάρτη με όλες τις μεταβλητές σε αύξουσα σειρά.

Περιγραφή της Εκτέλεσης

ΕΡΩΤΗΜΑ 1

- **Κώδικας:**

```
    printf("A : (%#010x) %5d\n", A,A);
    printf("A+2 : (%#010x) %5d\n", A+2,A+2);
    printf("(int)A + 2 : (%#010x) %5d\n", (*A + 2),(*A+2));
    printf("A: (%#010x) %5d\n", &A[2],&A[2]);

    int elementSize = sizeof(int);
    printf("The size of one integer is %d bytes.\n", elementSize);

    int tableSize = (elementSize * TABLE_SIZE);
    printf("The size of the table is %d bytes.\n", tableSize);

    return 0;
}
```

- **Αποτέλεσμα:**

```
A : (0xffffcbc0) -13376
A+2 : (0xffffcbc8) -13368
(int)A + 2 : (0xffffcbc2) -13374
A: (0xffffcbc8) -13368
The size of one integer is 4 bytes.
The size of the table is 40 bytes.
```

Το **A** είναι δείκτης που «δείχνει» στο πρώτο στοιχείο του πίνακα **(&A[0])**. Το **A+2** αναφέρεται στο τρίτο στοιχείο του πίνακα **(&A[2])** ενώ το **(int)A + 2** βρίσκεται 2 bytes πιο κάτω απο τη διεύθυνση του **(&A[0])**.

ΕΡΩΤΗΜΑ 2

- **Κώδικας**

```c
#include <stdio.h>
#include <stdlib.h>

long double var1 = 124858568854866353;
char var2 = 'b';
int var3 = 67;

int main(void) {

    printf("Address of var1: %5d\n", &var1);
    printf("Address of var2: %5d\n", &var2);
    printf("Address of var3: %5d\n", &var3);

    return 0;
}
```

- **Αποτέλεσμα**

```
Address of var1: 4202512
Address of var2: 4202528
Address of var3: 4202532
```

- **Παρατήρηση**

Οι μεταβλητές **var1** , **var2**, **var3** είναι διαδοχικές και ανάλογα με το είδος της μεταβλητής καταλαμβάνουν τον αντίστοιχο χώρο. Παρατηρούμε ότι η **long double** καταλαμβάνει **16 bytes**, ο **χαρακτήρας 1 byte** αλλά λόγω ***padding**, η επόμενη μεταβλητή **(int)** ξεκινάει 4 bytes παρακάτω και καταλαμβάνει άλλα 4 bytes. Σημειώνεται ότι η λειτουργία **padding** αναλύεται παιρετέρω στο επόμενο ερώτημα μιας και εκει είναι πιο σαφής η λειτουργία του.

ΕΡΩΤΗΜΑ 3

- **Κώδικας**

```c
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    struct A { //Should be 6 bytes
        char X; //1 byte
        int C;  //4 bytes
        char Y; //1 byte
    };
```

```
    struct B { //Should be 6 bytes
        char X; //1 byte
        char Y; //1 byte
        int C;  //4 bytes
    };

    printf("Size of struct A (2 characters and 1 integer): %d bytes.\n", sizeof(struct A));
    printf("Size of struct B (2 characters and 1 integer): %d bytes.\n", sizeof(struct B));

    return 0;
}
```

- **Αποτέλεσμα**

```
Size of struct A (2 characters and 1 integer): 12 bytes.
Size of struct B (2 characters and 1 integer): 8 bytes.
```

- **Παρατήρηση**

  Παρατηρήσαμε οτι εάν αλλάξουμε τη σειρα με την οποία ορίζουμε τις μεταβλητές υπάρχει διαφορά στο χώρο που καταλαμβάνει το struct. Η εξήγηση σε αυτό το φαινόμενο (**padding**) είναι οτι η μνήμη είναι φτιαγμένη ωστε να διαβάζεται μία «λέξη» κάθε φορα (1 λέξη ειναι 4 bytes σε 32-bit σύστημα και 8 bytes σε 64-bit σύστημα). Για παράδειγμα, στη **struct A**, δεσμεύεται 1 byte για το **char X** αλλά η C θα δεσμεύσει 3 επιπλέον bytes έτσι ώστε να συμπληρωθεί η «τετράδα». Οι μεταβλητές integer καταλαμβάνουν 4 bytes χώρο οπότε διαβάζονται ολόκληρες. Στη **struct B** απ'την άλλη, δεσμεύεται 1 byte για το **char X**, 1 byte για το **char Y** και επιπλέον 2 bytes για να σχηματιστεί ολόκληρη η λέξη. Αυτός είναι ο λόγος που τα struct A και B καταλαμβάνουν διαφορετικό μέγεθος στη μνήμη.

ΕΡΩΤΗΜΑ 4

- **Κώδικας**

```
int main(void) {

    void *ptr1 = malloc(1);
    void *ptr2 = malloc(10);
    void *ptr3 = malloc(16);
    void *ptr4 = malloc(32);

    printf("Returned address: %p\n", ptr1);
    printf("Returned address: %p\n", ptr2);
    printf("Returned address: %p\n", ptr3);
    printf("Returned address: %p\n", ptr4);

    free(ptr1);
    free(ptr2);
    free(ptr3);
    free(ptr4);

    return 0;
}
```

- **Αποτέλεσμα**

```
Returned address: 0x600000330
Returned address: 0x600000350
Returned address: 0x600000370
Returned address: 0x600000390
```

- **Παρατήρηση**

  Όταν καλούμε τη malloc, δεσμεύεται το ελάχιστο πολλαπλάσιο των 8 bytes σε 32-bit σύστημα και 16 bytes σε 64-bit σύστημα, έτσι ώστε να ικανοποιείται η ζητούμενη δεύσμευση. Χρειάζεται όμως και κάποιος χώρος για πληροφορίες όπως το μέγεθος της malloc και τις διευθύνσεις των θέσεων μνήμης που καταλαμβάνει (**bookkeeping**). Για αυτό βλέπουμε ότι για τις πρώτες 3 δεσμεύσεις μνήμης χρειάζεται 32 bytes, ενώ στην τελευταία (32 + 16) = 48 bytes. Σημειώνεται ότι οι παραπάνω διευθύνσεις αναπαρίστανται στο δεκαεξαδικό σύστημα.

ΕΡΩΤΗΜΑ 5

- **Κώδικας**

```c
#include <stdio.h>
#include <stdlib.h>

int global = 5;

int main(void) {

    int local = 10;

    void *ptr = malloc(5);

    printf("Main adddress: %#010x\n", &main);
    printf("Global variable address: %#010x\n", &global);
    printf("Local variable address: %#010x\n", &local);
    printf("Malloc return address: %p\n", ptr);

    free(ptr);

    return 0;
}
```

- **Αποτέλεσμα**

```
Main adddress: 0x004010e0
Global variable address: 0x00402010
Local variable address: 0xffffcbf4
Malloc return address: 0x600000330
```

- **Παρατήρηση**

  Παρατηρούμε ότι η C δεσμεύει για καθε τύπου μεταβλητή χώρο σε ένα συγκεκριμένο κομμάτι της μνήμης. Η τοπική μεταβλητή έχει την υψηλότερη διεύθυνση μνήμης **(stack memory)**. Ακολουθεί η Malloc **(heap memory)** με πιο χαμηλή διεύθυνση και τέλος η καθολική μεταβλητή και η main αποτελούν τη στατική μνήμη (**static memory**) με τη χαμηλότερη διεύθυνση.

Συμπέρασμα

Στο πρώτο εργαστήριο εξοικειωθήκαμε κυρίως με τις ιδιαιτερότητες σε κάποια λεπτά σημεία του τρόπου αποθήκευσης μνήμης ο οποίος διακυμαίνεται απο σύστημα σε σύστημα.