**16/10/2016**

# Αναφορά Εργαστηρίου 1

Ομάδα

|  |
| --- |
| *Κολομβάκης Χρήστος (Α.Μ.: 2013030103)* |
| *Ζαχαριουδάκης Χρήστος (Α.Μ.: 2014030056)* |

## Προεργασία

Κατα την προεργασία μας ζητήθηκε να κάνουμε επανάληψη στην γλώσσα προγραμματισμού C και ιδιαίτερα στην χρήση της στην διαχείριση της μνήμης . Επίσης μας ζητήθηκε κώδικας C για την περαιτέρω κατανόηση του τρόπου αποθήκευσης διάφορων μεταβλητών (δείκτες , δομές , δυναμικά δεσμευμένες , καθολικές , τοπικές κτλ) .

## Περιγραφή Ζητούμενων

Σκοπός της άσκησης είναι η εξοικείωση με τον τρόπο οργάνωσης των δεδομένων στην μνήμη, καθώς και η εξάσκηση στην χρήση δεικτών.

## Περιγραφή της Εκτέλεσης

Η άσκηση έγινε με χρήση του περιβάλλοντος Netbeans. Συγκεκριμένα δημιουργήσαμε 5 νέα project της C , από το ‘File →NewProject → C/C++ → C/C++ Application - >Finish’

Έπειτα από το ‘Run →Βuild Project’ κάναμε compileτο πρόγραμμα μας και απο το ‘Run →RunProject’το εκτελέσαμε .

**Ερώτημα Α**

H παράσταση Α + 1 είναι πράξη δεικτών (διευθύνσεων) και Α είναι δείκτης ακεραίων, οποτε το αποτέλεσμα της παραπάνω πράξης θα έιναι: &Α [0] + sizeof (int). Η παράσταση (((int) A) + 1) μετατρέπει την τιμή του Α , την διεύθυνση του πρώτου στοιχείου του πίνακα σε ακέραιο μεσω casting (int) και προσθέτει το 1 , δηλαδή μια πρόσθεση ακεραίων . Άρα προκύπτει το αποτέλεσμα &Α[0] + 1 . Αυτό φαίνεται τόσο σε δεκαδική όσο και σε δεκαεξαδική μορφη .Toμεγέθος του πίνακα είνα 10 \* sizeof (int) και ενος στοιχείου sizeof (int), καθώς ο πίνακας είναι τύπου intκαι έχει 10 θέσεις (int A [10]).

Capture

**Ερώτημα B**

Παρατηρούμε οτι οι μεταβλητές αποθηκεύονται σε φθίνουσα σειρά αρχίζοντας με αυτή που δηλώθηκε πρώτη και με διαφορά sizeof(int) (εδώ 4bytes) . Και αυτό διότι ανήκουν στο stackmemoryως localvariablesτης. Μια τέτοια μεταβλητή, σύμφωνα και με την εικόνα στην τρίτη σελίδα της αναφοράς, ονομάζεται autovariable.

Capture

**Ερώτημα C**

Παρόλο που η θεωρία της C αναφέρει ότι το μέγεθος μιας μεταβλητής – δομής είναι ίσος με το μέγεθος των επιμέρους μεταβλητών της , παρατηρούμε ότι δεν ισχύει πάντα .Αυτό φαίνεται ιδιαίτερα στην εκτέλεση του παρακάτω κώδικα . Έχοντας δύο χαρακτήρες (sizeof(char)= 1 byte)και έναν ακέραιο (sizeof(int)) = 4 bytes) θα αναμέναμε μέγεθος (2\*1 + 4) 6 bytes.

Κατά την αποθήκευση μεταβλητών στην μνήμη , οι μεταβλητές τοποθετούνται με το πρώτο byte τους να έχει διεύθυνση που είναι πολλαπλάσιο του μεγέθους τους. Το φαινόμενο αυτό ονομάζεται στοίχιση μνήμης (**memory alignment)**. Αυτό έχει ως αποτέλεσμα να δημιουργούνται κενά ανάμεσα στις μεταβλητές τα οποία ωστόσο θεωρούνται κατειλημμένα (Padding).H μνήμη ανάλογα με την αρχιτεκτονική του υπολογιστή, χωρίζει την μνήμη σε λέξεις (words) των 4 bytes (x32) ή των 8 bytes (x64).

Κατα την δήλωση μεταβλητων σε δομές, όταν πλέον αποθηκευτεί όλες οι μεταβλητές καταλαμβάνεται λόγω στοιχίσης και η ΄ουρά ’ της τελευταίας λέξης που έχει χρησιμοποιθεί.

Η μνήμη μπορεί να παρουσιαστεί ως ένας πίνακας από bytes στοιχισμένος ως εξής (σε x32) (βλ κώδικα) :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Struct ‘ONE’** | | | | **Struct ‘SECOND’** | | | |
| **X** |  |  |  | **X** | **Y** |  |  |
| **C** | **C** | **C** | **C** | **C** | **C** | **C** | **C** |
| **Y** |  |  |  |  |  |  |  |

Η παραπάνω κατανομή έχει ως αποτέλεσμα να εμφανίζεται το ακόλουθo μήνυμα στον κώδικα .

C:\Users\Christos\AppData\Local\Microsoft\Windows\INetCache\Content.Word\Capture.png

**Ερώτημα D**

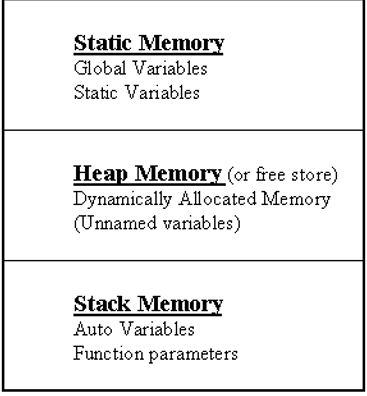
Στο ερώτημα αυτό μας ζητήθηκε να δεσμέυσουμε δυναμικά μνήμη με την χρήση της εντολής malloc .

CaptureΠαρατηρούμε οτι όλες οι διευθύνσεις έχουν απόσταση 32 και αυξάνονται με την σειρά που έχουν δεσμευτεί

Άρα αντίθετα απο ότι αναμέναμε όλες οι εντολές malloc φαίνεται να έχουν δεσμεύσει 32 byte μνήμης . Αυτό συμβαινει επείδή εκτός τις θέσεις μνήμης που χρειαζόμαστε , χρείαζονται κάποια bytesγια πληροφορίες όπως η αρχή και το τέλος των θέσεων μνήμης και το μεγεθός του δεσμευμένου χώρου μνήμης .

**Ερώτημα E**

Στη C και στην C++ χρησιμοποιούνται τρία είδη μνήμης ,όπως απεικονίζεται στην παρακάτω εικόνα :



Στον παρακάτω κώδικα δεσμεύουμε δυναμικά μνήμη και ορίζουμε μια καθολική και τοπική μεταβλητή . Έπειτα μας δείχνει τις διευθύνσεις της καθεμίας και της συνάρτησης main ().

Capture

|  |  |
| --- | --- |
| Μεταβλητή | Διεύθυνση |
| Local Variable | 24f344 |
| Main Function  Global Variable | 401530  407a20 |
| Dynamically Allocated Memory | 761430 |

Δηλαδή οι μεταβλητές και η συνάρτηση έχουν αποθηκευτεί με την εξής κατανομή στην μνήμη :

Παρατηρούμε ότι η ομαδοποιήση των μεταβλητων στην μνήμη είναι ίδια με αυτή της θεωρίας,ωστόσο η σειρά διαφέρει .

## Συμπεράσματα

Η διαχείριση μνήμης είναι μια πολύπλοκη διαδικασία , η οποία εξαρτάται από πολλούς παράγοντες (αρχιτεκτονική επεξεργαστή , λειτουργικό σύστημα κτλ) και συνεπώς συχνά διαφέρει για κάποια χαρακτηριστικάσε διάφορους υπολογιστές , παρόλο που ο εκτελέσιμος κώδικας είναι ο ίδιος .

## Παράρτημα - Κώδικας

***A)***

*#include <stdio.h>*

*#include <stdlib.h>*

*int var1 = 42;*

*int var2 = -1;*

*int main()*

*{*

*int A[10], i; /\* A = array of 10 ints, i = scalar int variable \*/*

*int \* p; /\* p is a scalar variable that points to an int \*/*

*for (i = 0; i < 10; i++) {*

*A[i] = i;*

*}*

*for(i = 0; i < 10; i++) {*

*printf("Element A[%d] = %d is stored in address : %x\n", i, A[i], &A[i]);*

*}*

*p = & var1;*

*printf("Var addresses(hex): %x %x %x # p = %x, \*p = %d\n", &var1, &var2, &p, p, \*p);*

*printf("Var values 1: %d %d hex: %x %x\n", var1, var2, var1, var2);*

*\*p = 0xffff;*

*printf("Var values 2: %d %d hex: %x %x\n", var1, var2, var1, var2);*

*\*(p+1) = 1500;*

*printf("Var values 3: %d %d hex: %x %x\n", var1, var2, var1, var2);*

*printf("\nLAB EXERCISE :\nDec: %d %d %d %d \n", A, A + 1, (((int) A) + 1), &(A[1]));*

*printf("hex: %x %x %x %x\n",A , A + 1, (((int) A ) + 1), &(A[1]));*

*printf ("The size of the array is : %d", 10 \* sizeof(int));*

*printf ("The size of an array element is : %d", sizeof(int));*

*}*

**B)**

*#include <stdio.h>*

*#include <stdlib.h>*

*int main(){*

*inti , j , k ;*

*printf ("Variable Addresses:\ni: %d \nj: %d \nk: %d \n",&i,&j,&k);}*

**C)**

*#include <stdio.h>*

*#include <stdlib.h>*

*struct one {*

*char X ;*

*int C ;*

*char Y ;*

*};*

*struct second {*

*char X , Y ;*

*int C ;*

*};*

*int main(){*

*printf ("The size of struct 'one' is : %d\n", sizeof(struct one));*

*printf ("The size of struct 'second' is : %d\n", sizeof(struct second));*

*}*

**D)**

*#include <stdio.h>*

*#include <stdlib.h>*

*int main(){*

*shortint \* ten1;*

*shortint \* ten2 ;*

*double \* sixteen ;*

*double \* thirytwo ;*

*ten1 = (short int\* )malloc (5\*sizeof(short int)); // 10 bytes*

*ten2 = (short int\* )malloc (5\*sizeof(short int)); // 10 bytes*

*sixteen = (double \*) malloc (2\*sizeof (double)); // 16 bytes*

*thirytwo = (double \*) malloc (4\*sizeof (double)); // 32 bytes*

*printf ("The first 'ten' array address is : %d",ten1);*

*printf ("\nThe second 'ten' array address is : %d",ten2);*

*printf ("\nThe 'sixteen' array address is : %d",sixteen);*

*printf ("\nThe 'thirytwo' array address is : %d",thirytwo);*

*}*

***E)***

*#include <stdio.h>*

*#include <stdlib.h>*

*int glob;*

*int main(){*

*int local;*

*int \*ptr ;*

*ptr = (int \*)malloc (5\*sizeof(int)) ;*

*printf ("The main() function address is : %x" , &main);*

*printf ("\nThe global variable address is : %x" , &glob);*

*printf ("\nThe local variable address is : %x" , &local);*

*printf ("\nThe dynamically allocated memory address is : %x" , ptr);}*