Ψηφιακοί Υπολογιστές - Αναφορά Εργαστηρίου 1

Ομάδα LAB20138010
- **Χρήστος Μιχαλόπουλος (2016030088)**
- **Αντώνης Ανδρεαδάκης (2013030059)**

Προεργασία

Για την πρώτη εργαστηριακή άσκηση, οι προαπαιτούμενες γνώσεις βασίζονταν πάνω στις υπάρχουσες γνώσεις μας σε θέματα κατανομής μνήμης στην γλώσσα C, από τη λειτουργία των δεικτών και των τύπων των μεταβλητών. Υλοποιήθηκαν τα ερωτήματα και επιβεβαιώσαμε τα συμπεράσματά μας απ' τη θεωρία.

Ζητούμενα και Εκτέλεση της άσκησης

Ερώτημα 1: Ζητούμενο του πρώτου ερωτήματος είναι η εκτύπωση σε δεκαδικό και σε δεκαεξαδικό των τιμών των παρακάτω: A, A+1, (int)A+1, &A. Επιπλέον θα πρέπει να τυπωθεί και το συνολικό μέγεθος του πίνακα A, καθώς επίσης και το μέγεθος ενός στοιχείου του πίνακα. Αρχικά, τυπώνονται τα μεγέθη σε δεκαεξαδικό και στη συνέχεια σε δεκαδικό. Τυπώνονται κατά σειρά το μέγεθος του πίνακα A, το μέγεθος του στοιχείου A+1 και η διεύθυνση του A[0]+1. Το τελευταίο γίνεται, γιατί ουσιαστικά το (int)A+1 αντιστοιχεί στην διεύθυνση του δείκτη του A, αυξημένο κατά 1. Ζητείται επίσης και η εκτύπωση της διεύθυνσης του A[1], αλλά η τιμή αυτή είναι ίδια με την τιμή της διεύθυνσης του A+1, αφού και τα δύο δείχνουν στην ίδια διεύθυνση (A[1]).

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

int var1 = 31;
int var2 = -2;

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("\n--//--\t--//--\t--//--\n");

    int *arr;
    int int_arr;
    arr = A;
    int_arr = ((int) A) + 1 ;

    printf("\nPrint at hex:\n");
    printf("Element A = %x,\n A + 1 = %x\n (((int) A) + 1) = %x\n address of 1st element = %x\n sizeof(A): %d\n", arr, (arr+1), int_arr, &A[1],sizeof(A));
    printf("\nPrint at dec:\n");
    printf("Element A = %d,\n A + 1 = %d\n (((int) A) + 1) = %d\n address of 1st element = %d\n sizeof(A[i]): %d\n", arr, (arr+1), int_arr, &A[1],sizeof(A[i]));
}
```

Ερώτηση 2: Για το δεύτερο πρόγραμμα δηλώσαμε 3 integer μεταβλητές (Var1, Var2, Var3) και στην συνέχεια τυπώσαμε τις διευθύνσεις τους σε δεκαδικό. Τυπώθηκαν σε αντίθετη σειρά από αυτή που δηλώθηκαν, δηλαδή πρώτα η διεύθυνση της Var3 μετά αυτή της Var2 και τέλος αυτή της Var1. Αυτό γίνεται, γιατί κάθε local μεταβλητή αποθηκεύεται στο stack και όπως είναι λογικό, η πρώτη μεταβλητή που θα δηλωθεί, θα εκτυπωθεί τελευταία και η τελευταία μεταβλητή θα εκτυπωθεί πρώτη (οι διευθύνσεις τους).

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

int main() {

    int Var1, Var2, Var3;

    Var1 = 2;
    Var2 = Var1 + 5;
    Var3 = Var1 * Var2;

    printf("\nVar values :  \n\t%d\n \t%d\n \t%d\n", Var1, Var2, Var3);
    printf("\nVar addresses :  \n\t%d\n  \t%d\n  \t%d\n", &Var1, &Var2, &Var3);
    printf("\nSize of variables: %d bytes\n", sizeof(Var1));

    return 0;
    //einai standar ari8moi kai diaferoun (meiwsh) kata 4 sto teleytaio dekadiko
}
```

Ερώτημα 3: Ζητείται η δημιουργία 2 δομών, στις οποίες θα δηλωθούν με διαφορετική σειρά οι τρεις μεταβλητές. Στην συνέχεια, θα πρέπει να τυπωθεί το μέγεθος της κάθε δομής. Δεν πήραμε το αναμενόμενο αποτέλεσμα, δηλαδή 6 bytes = (sizeof(int) + 2*sizeof(char)). Παρατηρήσαμε ότι δεσμεύτηκαν 12 και 8 bytes αντίστοιχα. Στην 1η περίπτωση, δεσμεύονται 4 bytes για τον πρώτο char (που δηλώσαμε) και χρησιμοποιείται το 1. Περισσεύουν 3 τα οποία δεν είναι ικανά να "καλύψουν" τον int που έχει δηλωθεί ακολούθως, οπότε δεσμεύονται άλλα 4 bytes και επειδή έχουμε δηλώσει άλλο 1 char, θα δεσμευθούν ακόμα 4 bytes, απ' τα οποία το 1 χρησιμοποιείται. Σύνολο: 3 μεταβλητές * 4 bytes για την κάθε μεταβλητή = 12 bytes.
Στην 2η περίπτωση, πάλι δεσμεύονται 4 για τον πρώτο char. Επειδή όμως, υπάρχει δηλωμένος άλλος ένας char και χωράει στα 3 ελεύθερα (λόγω δέσμευσης για τον 1ο char), δε χρειάζεται να δεσμευτεί extra χώρος. οπότε περισσεύουν 2 bytes. Αυτά δεν είναι ικανά να "καλύψουν" τον int που δηλώνεται στην συνέχεια, οπότε θα δεσμευτούν 4 ακόμα bytes. Έτσι έχουμε, 2 μεταβλητές * 4 bytes για κάθε μεταβλητή = 8 bytes.
Να σημειώσουμε ότι, η δέσμευση σε bytes αφορά πάντα τη μεταβλητή που καταλαμβάνει τον περισσότερο χώρο (στην περίπτωσή μας ο int). Δηλαδή αν είχαμε κάποιον double int (ο οποίος απαιτεί 8 bytes), τότε με βάση αυτόν θα υπολογίζαμε τα bytes που θα δεσμεύοταν.

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

typedef struct { //ka8e metavliti apaitei 4 bytes
    char X;  //pianei 1 byte ap ta 4 kai perisseuoun 3, den mporei na apo8ikeusei ton int pou dilonetai apo
        katw
    int C;   //pianei kai ta 4 bytes kai den exei keno
    char Y; //pianei 1 byte ap ta 4
}packOne;   //synolo 12 bytes (3 * 4)

typedef struct { //ka8e metavliti apaitei 4 bytes
    char X;  //pianei 1 byte ap ta 4 kai perisseuoun 3, opote xwraei kai ton epomeno char
    char Y;  //efoson dilonetai amesws meta ap to X
    int C;  //pianei 4 bytes
}packTwo;   //synolo 8 bytes (2 * 4)

int main(){
    //    struct Package1;
    //    struct Package2;
    printf("\nSize of struct 1: %d bytes\n", sizeof(packOne));
    printf("\nSize of struct 2: %d bytes\n", sizeof(packTwo));
}
```

Ερώτημα 4: Ζητούμενο για το τέταρτο υποερώτημα είναι η δυναμική δέσμευση, με την χρήση της malloc για 1, 10, 16, 32 bytes και η εκτύπωση των διευθύνσεων τους σε δεκαεξαδικό. Αρχικά, δηλώσαμε τέσσερις pointers m1, m2, m3, m4 οι οποίοι χρησιμοποιήθηκαν για να κάνει η malloc την δυναμική δέσμευση μνήμης. Στην συνέχεια, τυπώθηκαν οι διευθύνσεις που επιστράφηκαν για κάθε pointer και παρατηρήθηκε ότι η διεύθυνση του m1 απέχει από το m2 κατά 32 byte, το m2 από το m3 κατά 24 byte και το m3 από το m4 κατά 24 byte. Το τελικό συμπέρασμα, είναι ότι η διαδικασία δέσμευσης μνήμης διαφέρει και εξαρτάται από το λειτουργικός σύστημα του υπολογιστή, το μεταγλωττιστή (compiler) του προγράμματος που χρησιμοποιούμε αλλά και από τους πόρους οι οποίοι διατίθενται σε hardware.

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

int main(){

 void *m1,*m2,*m3,*m4;
   m1=malloc(1);
   m2=malloc(10);
   m3=malloc(16);
   m4=malloc(32);

   printf("\nAdress of m1 in memory is: %d\n",m1); //diaferei ap to m2 kata 32
   printf("\nAdress of m2 in memory is: %d\n",m2); //diaferei ap to m3 kata 24
   printf("\nAdress of m3 in memory is: %d\n",m3); //diaferei ap to m4 kata 24
   printf("\nAdress of m4 in memory is: %d\n",m4);

 return 0;
}
```

Ερώτημα 5: Ζητείται η εκτύπωση σε δεκαεξαδικό των διευθύνσεων των εξής: της συνάρτησης main, μιας καθολικής μεταβλητής, μιας τοπικής μεταβλητής και μιας malloc. Παρατηρούμε, ότι η διεύθυνση της main και της καθολικής μεταβλητής είναι "κοντά". Αυτό συμβαίνει γιατί είναι και οι δύο αποθηκευμένες στο static μέρος της μνήμης. Από την άλλη η διεύθυνση της τοπικής μεταβλητής είναι αποθηκευμένη στο stack. Τέλος, η διεύθυνση της malloc είναι αποθηκευμένη στο heap μέρος της μνήμης.

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

int i;   //ka8oliki metavliti

int main(){

    int x, *y; //topikh metavliti kai pointer o opoios xreiazetai gia to malloc
    y = (int *)malloc;
    //ektupwsi twn zitoumenwn me th seira
    printf("\n%p\n", &main);
    printf("\n%p\n", &i);
    printf("\n%p\n", &x);
    printf("\n%p\n", y);


    /*
     oi dieu8unseis ths main kai ths global (i) einai poly konta
     se sxesh me ayth ths local kai toy pointer, gt h main kai h
     global apo8ikeuontai se upsiles 8eseis mnhmhs (static memory)
     anti8etws oi local kai oi pointers se xamiloterh kai apexoun
     perissotero (heap memory)
     */


    system("PAUSE");
    return 0;
}
```

Συμπεράσματα

Ολοκληρώνοντας την προεργασία εξοικειωθήκαμε με θέματα κατανομής μνήμης στην C, καθώς επίσης κατανοήσαμε την διαφορά μεταξύ της static μνήμης, της heap και της stack. Ακολουθεί ο χάρτης μνήμης:

	MEMORY
Static LOW ADDRESS	(global variables, main)
Heap	malloc()
Stack HIGH ADDRESS	(local variables)

Ψηφιακοί Υπολογιστές - Αναφορά Εργαστηρίου 2

Προεργασία

Για την δεύτερη εργαστηριακή άσκηση, προαπαιτούμενες γνώσεις ήταν η σωστή χρήση της C για την επίλυση προβλημάτων με λίστες και παράλληλα, η δυνατότητα μετατροπής των εντολών της C σε CLANG.

Ζητούμενα και Εκτέλεση της άσκησης

Αρχικά έπρεπε να γράψουμε δύο προγράμματα, ένα σε C και ένα σε CLANG.
Για το πρόγραμμα της C, έπρεπε να δημιουργήσουμε ένα μενού 10 επιλογών και ανάλογα με το τι θα εισήγαγε ο χρήστης, θα γινόταν και η κατάλληλη εργασία. Στην CLANG θα έπρεπε να υλοποιήσουμε τις επιλογές 1-5, την έξοδο και το μενού επιλογών. Παραθέτουμε παρακάτω, στις αριστερές εικόνες σε γλώσσα C τον κώδικα και δεξιά σε CLang.
Συγκεκριμένα:
Επιλογή 1: Σε C δημιουργούμε μια λίστα/ CLang: Θέτουμε έναν δείκτη head ίσο με NULL, για να δημιουργηθεί ουσιαστικά η λίστα.

```c
struct list *createList (struct list *head)
{
    head = NULL;
    printf("LIST CREATED.");
    return head;
}
```

```c
long long createList (long long R4)
{
    R4 =(long long)NULL;
    printf("LIST CREATED.\n");
    return R4;
}
```

Επιλογή 2: C: Γίνεται εισαγωγή ενός στοιχείου σε αύξουσα σειρά στην λίστα / CLang: Η συνάρτηση δέχεται ως ορίσματα το head και το node. Αρχικά εισάγει και τοποθετεί έναν δείκτη να “κοιτάει” στο head. Έπειτα, ελέγχεται αν ένας καινούργιος κόμβος πρέπει να μπει στην κεφαλή της λίστας, είτε επειδη η λίστα είναι άδεια, είτε επειδή η τιμή (value) του head είναι μεγαλύτερη από αυτή του node και αν ισχύει εισάγεται εκεί. Σε άλλη περίπτωση μέσω της εντολής while, ο κόμβος (node) τρέχει μέσα στην λίστα μέχρι να βρει κόμβο με τιμή μεγαλύτερη από του node ή, αν δεν γίνει αυτό, τρέχει μέχρι το τέλος της λίστας. Σε κάθε μια από τις 2 περιπτώσεις εισάγεται κατάλληλα ο node στην λίστα. Η συνάρτηση (insertElement) επιστρέφει τον δείκτη head.

```c
struct list *insertElement (struct list *head, struct list *node){

    struct list *ptr = head;


    if ((head == NULL) || (head ->value) >= (node->value))
    {
        node->next = head;
        return node;
    }
    while((ptr->next != NULL) && (ptr->next->value)<=(node->value))
        ptr = ptr->next;
    node->next = ptr ->next;
    ptr->next = node;

    return head;
}
```

```c
long long insertElement (long long R4, long long R5)
{
    struct list *ptr, *head, *node;
    head = (struct list *)R4;
    node = (struct list *)R5;

    if(R4 == R31) goto ie_if1; //ie_if1: insertElement_if1 (gia na ksexorizontai ta labels twn allwn functions)
    R9 =(long long)head->value;
    R10 =(long long)node->value;
    if(R9 >= R10) goto ie_if1;
    goto ie_if1end;
    ie_if1:
        node->next =(struct list *)R4;
        R2 = R5;
        return R2;
    ie_if1end:

    ptr = (struct list *)R4;
    ie_while1:
    R8 =(long long)ptr->next;
    if(R8 == R31) goto ie_while1end;
    R9 =(long long)ptr->next->value;
    R10=(long long)node->value;
    if(R9>R10) goto ie_while1end;
        ptr = (struct list *)R8; //ptr = ptr->next
    goto ie_while1;
    ie_while1end:

    node->next = (struct list *)R8;
    ptr->next = (struct list *)R5;

    return R4;

}
```

<u>Επιλογή 3:</u> C: Διαγραφή ενός στοιχείου της λίστας που θα επέλεγε ο χρήστης / CLang:
Η συνάρτηση δέχεται ως ορίσματα τον δείκτη head και την τιμή value του κόμβου που θέλουμε να διαγράψουμε. Αρχικά γίνεται ο έλεγχος αν η λίστα είναι άδεια. Αν δεν είναι εισάγονται δύο δείκτες (ptr, node) οι οποιοι δείχνουν στην κεφαλή της λίστας. Σε περίπτωση που ο χρήστης ζητήσει να διαγραφεί το head, τότε ο δείκτης ptr θα δείξει στον αμέσως επόμενο κόμβο (ο οποίος θα γίνει το νέο head) και θα διαγραφεί το προηγούμενο του στοιχείο. Σε άλλη περίπτωση, θα "τρέξει" μέσα στην λίστα ο δείκτης, έως ότου να βρεθεί τιμή ίση με την δοθείσα από τον χρήστη. Στην συνέχεια ανάλογα με την περίπτωση πράττει ο κόμβος αναλόγως.

```
struct list *deleteElement(struct list *head, int value)
{
    struct list *ptr, *node;
    int numOfNodes=0;
    ptr = head;
    node = head;

    if (head == NULL)
    {
        printf("THE LIST IS EMPTY. \n");
        return head;
    }
    if(head->value == value)
    {
        ptr=ptr->next;
        free(node);
        printf("NODE HAS BEEN DELETED. \n");
        return ptr;
    }

    while (((ptr->next) != NULL) && ((ptr->next->value)!=value))
    {
        ptr=ptr->next;
    }

    if ((ptr->next) == NULL)
        printf("THERE IS NO ELEMENT WITH THIS VALUE: %d \n", value);

    else if (ptr->next->value == value)
    {
        node = ptr;
        node = node->next;
        ptr->next = node->next;
        free(node);
        numOfNodes--;
    }
    else
        printf("ERROR\n");
    return head;
}
```

```
long long deleteElement(long long R4, long long R5){
    struct list *ptr, *node, *head;
    int numOfNodes=0,value;

    head = (struct list *)R4;
    value = (int)R5;
    node = (struct list *)R6;
    ptr = (struct list *)R7;

    R7 = R4;
    R6 = R4;

    R11 =(long long)head->value;
    R12 =(long long)ptr->next;
    if (R4 != R31) goto else_label_1;
        printf("THE LIST IS EMPTY. \n");
        return R4;
else_label_1:
    if (R11 != R12) goto else_label_2;
        R4 = R12;
        free((struct list *)R6);
        printf("NODE HAS BEEN DELETED. \n");
        return R7;

else_label_2:

while_label:
    R12 =(long long)ptr->next;
    if(R12 == R31) goto while_end1;
    R13 =(long long)ptr->next->value;
    R14=(long long)node->value;
    if(R13==R14) goto while_end1;
    R7 = R12; //ptr = ptr->next
    goto while_label;
while_end1:
    R13 =(long long)ptr->next;
    if (R13 != R31) goto if_label_2;
    printf("THERE IS NO ELEMENT WITH THIS VALUE: %lld \n", R5);

    R14 =(long long)ptr->next->value;
    R15 =(long long)node->next;
    R16 =(long long)ptr->next;
    R17 =(long long)numOfNodes;
if_label_2:
    if(R14 != R5) goto else_label_3;
        R6 = R7;
        R6 = R15;
        R16 = R15;
        free((struct list *)R6);
        R17--;
else_label_3:
    printf("ERROR \n");
return R4;
```

<u>Επιλογή 4:</u> C: Εκτύπωση ενός στοιχείου της λίστας που θα επέλεγε ο χρήστης (αν υπήρχε) / CLang:
Η συνάρτηση δέχεται ως όρισμα το head και το value του δείκτη που θέλει ο χρήστης να εκτυπώσει. Επιπλέον χρησιμοποιείται ένας βοηθητικός δείκτης που τρέχει μέσα στην λίστα μέχρι να βρει τον κατάλληλο κόμβο ή μέχρι να φτάσει στο τέλος της λίστας. Αν βρεθεί η τιμή που εισήγαγε ο χρήστης, με μια if/else, τυπώνεται την τιμή και το ID του κομβου, αλλιώς τυπώνεται το κατάλληλο μήνυμα λάθους.

```
void printNode(long long R4, long long R5)
{
    struct list *ptr, *head;
    int value;

    head = (struct list *)R4;
    value = (int)R5;
    ptr = (struct list *)R6;

while_label:
    R10 = (long long)ptr->value;
    if(R10 == R5) goto while_end_1;
    R11 = (long long)ptr->next;
    if(R11 ==  R31) goto while_end_1;
    R6 = R11;

    goto while_label;
while_end_1:
    R12 = (long long)ptr->value;
    R13 = (long long)ptr->id;

    if(R12 != R5) goto if_end_1;
    printf("THE FIRST NODE THAT HAS THE VALUE OF %lld HAS THE ID: %3lld", R5, R13);
if_end_1:
    printf("THERE WAS NO ELEMENT WITH THE VALUE YOU ENTERED.");
```

```c
void  printNode(struct list *head, int value)
{
    struct list *ptr;
    ptr = head;

    while (((ptr->value) != value) && ((ptr->next) != NULL))
        ptr = ptr->next;
    if((ptr->value) == value)
        printf("THE FIRST NODE THAT HAS THE VALUE OF %d HAS ID: %3i \n", value, ptr->id);
    else
        printf("THERE WAS NO ELEMENT WITH THE VALUE YOU ENTERED");
}
```

<u>Επιλογη 5:</u> C: Εκτύπωση του αριθμού των στοιχείων της λίστας / CLang:H συνάρτηση δέχεται ως όρισμα το head. Στην αρχή εισάγει την numOfNodes και την αρχικοποιει στο μηδέν. Έπειτα, ελέγχει αν είναι άδεια η λίστα και αν ισχύει, τυπώνει το κατάλληλο μήνυμα. Αλλιώς διασχίζει την λίστα μέχρι το τέλος και ταυτόχρονα σε κάθε κόμβο αυξάνει και την τιμή numOfNodes. Τέλος τυπώνεται ο συνολικός αριθμός κόμβων στην λίστα.

```c
void printNumOfNodes(struct list *head)
{
    int numOfNodes=0;
    if (head != NULL)
        numOfNodes++;
    while(head->next != NULL)
    {
        head=head->next;
        numOfNodes++;
    }
    printf("THE NUMBER OF NODES IN THE LIST IS: %d", numOfNodes);
}
```

```c
void printNumOfNodes(long long R4)
{
    struct list *head;
    head = (struct list *)R4;
    int numOfNodes = 0;
    R10 = (long long)numOfNodes;
    pnn_if1:
    if(R4 == R31) goto pnn_if1end;
        R10++;
        pnn_if1end:
    R11 = (long long)head->next;
    while(R11 == R31) goto pnn_while1end;
        R4 = R11;
        R10++;

    pnn_while1end:
    printf("THE NUMBER OF NODES IN THE LIST IS: %lld", R10);
}
```

<u>Επιλογές 6-9:</u> Στις επιλογές 6-7, γίνεται εκτύπωση των διευθύνσεων αρχικά ενός συγκεκριμένου στοιχείου και στην συνέχεια ολόκληρης της λίστας, με βάση την ίδια λογική όπως και στο πρώτο εργαστήριο. Με παρόμοιο τρόπο λύνονται και οι εργασίες των 8-9, αλλά με την χρήση ενός sizeof. Δηλαδή, στις επιλογές 8 και 9 θα εκτυπωνόταν το μέγεθος ενός συγκεκριμένου στοιχείου και της λίστας, αντίστοιχα, σε byte. Τέλος, με την επιλογή 10 θα γινόταν έξοδος από το μενού και θα τερματιζόταν το πρόγραμμα. Οι κώδικες στην C είναι παρόμοιοι με αυτούς της πρώτης εργαστηριακής άσκησης. Παρακάτω παρατίθενται οι κώδικες για τις συναρτήσεις 6 έως 9 από αριστερά προς δεξιά:

```c
void printListAddr(struct list *head)
{
    printf("THE ADDRESS OF THE LIST IS: %p \n", head);
}
```

```c
void printNodeAddr(struct list *head)
{
    int j;
    if (head == NULL)
    {
        printf("THE LIST IS EMPTY.\n");
        return;
    }

    printf("PLEASE GIVE THE VALUE OF THE NODE: \n");
    scanf ("%d", &j);

    while (head != NULL)
    {
        if (head->value == j)
        {
            printf("THE ADDRESS OF THE NODE YOU SEARCHED IS: %p \n", head);
            return;
        }
        else
        {
            head = head->next;
        }
    }
    printf("THERE IS NO NODE WITH THE VALUE YOU HAVE ENTERED.\n");
    return;
}
```

```c
void printSizeOfList(struct list *head)
{
    int j=0;
    if (head == NULL)
    {
        printf("THE LIST IS EMPTY.\n");
        return;
    }
    while (head != NULL)
    {
        j++;
        head = head->next;
    }
    printf("THE SIZE OF THE LIST IS: %ld \n", (sizeof(struct list)*j));
}
```

```c
void printSizeOfNode(struct list *head)
{
    int j;
    if (head == NULL)
    {
        printf("THE LIST IS EMPTY.\n");
        return;
    }

    printf("PLEASE GIVE THE VALUE OF THE NODE: \n");
    scanf ("%d", &j);

    while (head != NULL)
    {
        if(head->value == j)
        {
            printf("THE SIZE OF THE ELEMENT IS: %ld", sizeof(*head));
            return;
        }
        else
        {
            head = head->next;
        }
    }
    printf("THERE IS NO NODE WITH THE VALUE YOU HAVE ENTERED.\n");
}
```

Συμπεράσματα

Ολοκληρώνοντας την δεύτερη εργαστηριακή άσκηση, μας έγινε πλέον κατανοητή η σχέση που έχουν C και CLANG. Παράλληλα μας δόθηκε μια πολύ καλή ευκαιρία να προγραμματίσουμε στην καινούργια, για εμάς, γλώσσα προγραμματισμού CLANG.

FlowChart: