Δείκτες και Λίστες

Ιωάννης Γ. Τσούλος

2020

1 Δείκτες

1.1 Εισαγωγή στην χρήση δεικτών

Κάθε μεταβλητή στην γλώσσα C βρίσκεται σε συγκεκριμένη θέση στην μνήμη του υπολογιστή. Αυτή η θέση ονομάζεται και διεύθυνση και υπάρχει δυνατότητα να την εντοπίσουμε με την χρήση του τελεστή & όπως παρουσιάζεται και στο παράδειγμα 1. Οι διευθύνσεις μνήμης μπορούν να εκχωρηθούν σε ειδικές μεταβλητές που ονομάζονται δείκτες. Οι μεταβλητές αυτές έχουν δηλώνονται σαν TYPE *VARIABLE, όπου TYPE ένας οποιοσδήποτε τύπος της γλώσσας (πχ. int, double κτλ) Στο παράδειγμα 2 γίνεται ανάθεση των διευθύνσεων απλών μεταβλητών σε δείκτες και στην συνέχεια γίνεται εμφάνιση αυτών των τιμών. Ωστόσο οι δείκτες μπορούν να χρησιμοποιηθούν για την απόδοση τιμών στις μεταβλητές στις οποίες δείχνουν, δηλαδή κάποιος μπορεί έμμεσα να αλλάξει την τιμή μιας μεταβλητής μέσω του δείκτη. Αυτό παρουσιάζεται στον αλγόριθμο 3.

1.2 Χρήση δεικτών σε συναρτήσεις

Με την χρήση των δεικτών και την δυνατότητα που δίνουν για έμμεση αναφορά σε μεταβλητές είναι εφικτό κανείς να επιστρέψει παραπάνω από μια τίμες από μια

Algorithm 1 Εμφάνιση διεθύνσεων μεταβλητών.

```
1  # include <stdio.h>
2
3  int main()
4  {
5          int x=100;
6          double y=200;
7          printf("Variables are %d %lf\n",x,y);
8          printf("Addresses are %d %d \n",&x,&y);
9          return 0;
10 }
```

Algorithm 2 Ανάθεση διευθύνσεων σε δείκτες.

```
1  # include <stdio.h>
2
3  int main()
4  {
5          int x=100;
6          double y=200;
7          int *px=&x;
8          double *py=&y;
9          printf("Variables are %d %lf \n",x,y);
10         printf("Addresses are %d %d \n",px,py);
11         return 0;
12 }
```

Algorithm 3 Αλλαγή μεταβλητών με την χρήση δεικτών.

```
1  # include <stdio.h>
2
3  int main()
4  {
5          int x=100;
6          double y=200;
7          int *px=&x;
8          double *py=&y;
9          printf("Variables are %d %lf \n",x,y);
10         *px=5;
11         *py=20;
12         printf("Variables are %d %lf \n",x,y);
13         return 0;
14 }
```

Algorithm 4 Μια πρώτη προσπάθεια υλοποίησης συνάρτησης αντιμετάθεσης μεταβλητών.

```
1  # include <stdio.h>
2
3  void myswap(int a,int b)
4  {
5          int t=a;
6          a=b;
7          b=t;
8  }
9
10 int main()
11 {
12         int x=100;
13         int y=200;
14         printf("Before swap %d %d\n",x,y);
15         myswap(x,y);
16         printf("After swap %d %d\n",x,y);
17         return 0;
18 }
```

συνάρτηση. Για παράδειγμα έστω ότι χρειάζεται να υλοποιηθεί μια συνάρτηση που να αντιμεταθέτει δύο ακέραιες μεταβλητές. Μια πρώτη προσπάθεια να γίνει αυτό παρουσιάζεται στον αλγόριθμο 4. Προφανώς αυτός ο αλγόριθμος δεν μπορεί να κάνει την αντίστροφη, καθώς οι μεταβλητές περνούν στην συνάρτηση με τιμή και το μόνο που κάνει η συνάρτηση είναι να αντιμεταθέσει τις τοπικές μεταβλητές και όχι τα πραγματικά ορίσματα. Αυτό μπορεί να επιλυθεί με την χρήση δεικτών όπως παρουσιάζεται και στον αλγόριθμο 5.

2 Δυναμική κατανομή μνήμης

2.1 Δυναμικές μεταβλητές

Με την χρήση των δεικτών μπορεί κανείς εύκολα να δημιουργήσει και να διαγράψει μεταβλητές όταν αυτό χρειάζεται. Για να επιτευχθεί αυτό γίνεται χρήση των τελεστών new και delete. Ο τελεστής new χρησιμοποιείται για να δώσει μνήμη και ο τελεστής delete χρησιμοποιείται για να διαγράψει (αποδεσμεύσει) την μνήμη αυτή. Πρέπει πάντοτε να γίνεται αποδέσμευση αυτής της μνήμης και αυτό πρέπει να γίνεται με τον τελεστή delete. Η γλώσσα C δεν διαγράφει ποτέ αυτόματα την μνήμη που έχει δεσμευθεί. Μια ενδεικτική χρήση των παραπάνω τελεστών παρουσιάζεται στον αλγόριθμο 6. Η σταθερά NULL είναι ίδια με τον αριθμό 0, χρησιμοποιείται αρκετά συχνά προκειμένου να αρχικοποποιήσει δείκτες σε κενή θέση μνήμης.

Algorithm 5 Αντιμετάθεση μεταβλητών με την χρήση συνάρτησης.

```
1  # include <stdio.h>
2  void myswap(int *a,int *b)
3  {
4          int t=*a;
5          *a=*b;
6          *b=t;
7  }
8
9  int main()
10 {
11         int x=100;
12         int y=200;
13         printf("Before swap %d %d\n",x,y);
14         myswap(&x,&y);
15         printf("After swap %d %d\n",x,y);
16         return 0;
17 }
```

Algorithm 6 Δημιουργία δυναμικής μεταβλητής.

```
# include <stdio.h>
int main()
{
        double *xpointer=NULL;
        printf("Starts with address %d\n",xpointer);
        xpointer=new double;
        printf("Now the address is %d\n",xpointer);
        *xpointer=100;
        printf("The value of the memory
is %lf\n",*xpointer);
        *xpointer=*xpointer+100;
        printf("Now the value of the memory
is %lf\n",*xpointer);
        delete xpointer;
        return 0;
}
```

Algorithm 7 Δημιουργία μονοδιάστατου πίνακα ακεραίων αριθμών.

```
1  # include <stdio.h>
2  # include <stdlib.h>
3  int main()
4  {
5          int n;
6          int *x=NULL;
7          int i;
8          printf("Please provide array's dimension ");
9          scanf("%d",&n);
10         x=(int*)malloc(sizeof(int) * n);
11         for(i=0;i<n;i++) x[i]=i+1;
12         for(i=0;i<n;i++)
13         {
14                 printf("%d \n",x[i]);
15         }
16         free(x);
17         return 0;
18 }
```

2.2 Δυναμικοί πίνακες

Οι τελεστές new και delete συνήθως χρησιμοποιούνται για την δημιουργία δυναμικών πινάκων και δομών και όχι τόσο για την δημιουργία δυναμικών απλών μεταβλητών. Στο παράδειγμα του αλγορίθμου 7 ο χρήστης εισάγει τον επιθυμητό αριθμό στοιχείων ενός πίνακα και στην συνέχεια δημιουργείται ένας πίνακας ακεραίων με τον επιθυμητό αριθμό στοιχείων. Η δημιουργία δυναμικών πινάκων πρέπει να γίνεται με την χρήση του σχήματος mallon(n * sizeof(TYPE)), όπου TYPE ο επιθυμητός τύπος δεδομένων του πίνακα. Η διαγραφή του δυναμικού πίνακα γίνεται πάντοτε με το σχήμα free(array).

Στην περίπτωση που ο χρήστης θέλει να δημιουργήσει δυναμικό πίνακα δύο διαστάσεων, ακολουθείται μια διαδικασία που είναι πιο περίπλοκη, καθώς ένας πίνακας δύο διαστάσεων θεωρείται και πίνακας από πίνακες. Ένα παράδειγμα δημιουργίας και εμφάνισης πίνακα δύο διαστάσεων παρουσιάζεται στον αλγόριθμο 8.

3 Λίστες

Με τις λίστες κανείς μπορεί να δημιουργήσει δυναμικές δομές, οι οποίες μπορούν να αλλάξουν δυναμικά σε μέγεθος ανάλογα με τις απαιτήσεις της εφαρμογής.

Algorithm 8 Δημιουργία πίνακα δύο διαστάσεων.

```
1  # include <stdio.h>
2  # include <stdlib.h>
3  int main()
4  {
5          int rows,cols;
6          int i,j;
7          double **table;
8          printf("Provide rows for the table ");
9          scanf("%d",&rows);
10         printf("Provide columns for the table ");
11         scanf("%d",&cols);
12         table=(double **)malloc(sizeof(double*)*rows);
13         for(i=0;i<rows;i++)
14                 table[i]=(double *)malloc(sizeof(double)*cols);
15         for(i=0;i<rows;i++)
16         {
17                 for(j=0;j<cols;j++)
18                 {
19                         table[i][j]=(i+1)*(j+1);
20                 }
21         }
22         for(i=0;i<rows;i++)
23         {
24                 for(j=0;j<cols;j++)
25                 {
26                         printf("%lf\t",table[i][j]);
27                 }
28                 printf("\n");
29         }
30         for(i=0;i<rows;i++) free(table[i]);
31         free(table);
32         return 0;
33 }
```

3.1 Εισαγωγή κόμβου

Στο παράδειγμα του αλγορίθμου 9 δημιουργείται μια δομή για την αποθήκευση της λίστας, γίνεται εισαγωγή δύο στοιχείων και στην συνέχεια εμφανίζεται το μήκος της λίστας.

3.2 Εμφάνιση λίστας

Στο παράδειγμα του αλγορίθμου 10 η λίστα από το προηγούμενο παράδειγμα εμφανίζεται με την χρήση μιας συνάρτησης που εκτελεί μια επανάληψη μέχρι να βρεθεί κενό στοιχείο στην λίστα.

3.3 Προσθήκη στοιχείου στην αρχή της λίστας

Στο παράδειγμα του αλγορίθμου 11 παρουσιάζεται μια συνάρτηση για την εισαγωγή δεδομένων στην αρχή της λίστας.

3.4 Διαγραφή κόμβου

Στο παράδειγμα του αλγορίθμου 12 παρουσιάζεται μια συνάρτηση διαγραφής στοιχείων από λίστα.

Algorithm 9 Εισαγωγή κόμβων σε λίστα και εμφάνιση μήκους.

```
1  # include <stdio.h>
2  # include <stdlib.h>
3
4  typedef struct Node
5  {
6          int value;
7          struct Node *next;
8  }List;
9
10 int listSize(List *t)
11 {
12         int count=0;
13         while(t!=NULL)
14         {
15                 t=t->next;
16                 count++;
17         }
18         return count;
19 }
20
21 int main()
22 {
23         List *head=NULL;
24         head=(List *)malloc(sizeof(List));
25         head->value=100;
26         head->next=(Node *)malloc(sizeof(List));
27         head->next->value=200;
28         head->next->next=NULL;
29         printf("List size is %d\n",listSize(head));
30         return 0;
31 }
```

Algorithm 10 Εμφάνιση στοιχείων λίστας.

```
1  # include <stdio.h>
2  # include <stdlib.h>
3
4  typedef struct Node
5  {
6          int value;
7          struct Node *next;
8  }List;
9
10
11 void printList(List *t)
12 {
13         while(t!=NULL)
14         {
15                 printf("Element is %d\n",t->value);
16                 t=t->next;
17         }
18 }
19
20 int main()
21 {
22         List *head=NULL;
23         head=(List *)malloc(sizeof(List));
24         head->value=100;
25         head->next=(Node *)malloc(sizeof(List));
26         head->next->value=200;
27         head->next->next=NULL;
28         printList(head);
29         return 0;
30 }
```

Algorithm 11 Εισαγωγή στοιχείων στην αρχή της λίστας.

```
1  # include <stdio.h>
2  # include <stdlib.h>
3
4  typedef struct Node
5  {
6          int value;
7          struct Node *next;
8  }List;
9
10 void addElement(List ** head, int x)
11 {
12     List  *new_node;
13     new_node = (List *)malloc(sizeof(List));
14     new_node->value = x;
15     new_node->next = *head;
16     *head = new_node;
17 }
18
19
20 void printList(List *t)
21 {
22         while(t!=NULL)
23         {
24                 printf("Element is %d\n",t->value);
25                 t=t->next;
26         }
27 }
28
29 int main()
30 {
31         List *head=NULL;
32         int x;
33         do
34         {
35                 printf("Enter positive x ");
36                 scanf("%d",&x);
37                 addElement(&head,x);
38         }while(x>0);
39         printList(head);
40         return 0;
41 }
```

Algorithm 12 Διαγραφή κόμβου από λίστα.

```
1  # include <stdio.h>
2  # include <stdlib.h>
3
4  typedef struct Node
5  {
6          int value;
7          struct Node *next;
8  }List;
9
10 void addElement(List ** head, int x)
11 {
12     List  *new_node;
13     new_node = (List *)malloc(sizeof(List));
14     new_node->value = x;
15     new_node->next = *head;
16     *head = new_node;
17 }
18
19
20 int      removeElement(List **head)
21 {
22         int retval = -1;
23         List *next_node = NULL;
24         if (*head == NULL)
25         {
26                 return -1;
27         }
28         next_node = (*head)->next;
29         retval = (*head)->value;
30         free(*head);
31         *head = next_node;
32         return retval;
33 }
34
35 void printList(List *t)
36 {
37         while(t!=NULL)
38         {
39                 printf("Element is %d\n",t->value);
40                 t=t->next;
41         }
42 }
```