Προγραμματισμός σε C - Δείκτες

Ιωάννης Γ. Τσούλος

Τμήμα Πληροφορικής και τηλεπικοινωνιών
Πανεπιστήμιο Ιωαννίνων

2023

Περίληψη

Θέσεις μνήμης

- Κάθε μεταβλητή στην γλώσσα C++ βρίσκεται σε συγκεκριμένη θέση στην μνήμη του υπολογιστή.
- Αυτή η θέση είναι ένας ακέραιος αριθμός αυστηρά θετικός.
- Κάθε αναφορά σε μεταβλητή πάντοτε μεταφράζεται σε διευθύνσεις στην μνήμη.
- Η θέση αυτή ονομάζεται διεύθυνση
- Υπάρχει δυνατότητα να εντοπιστεί η θέση αυτή με την χρήση του τελεστή διεύθυνσης &

Εντοπισμός διεθύνσεων

```
# include <stdio.h>
int main()
{
        int x=100;
        int *px=&x;
        printf("X= %d PX = %d \n",x,px);
        printf("X= %d PX = %x \n",x,px);
        return 0;
}
```

Εντοπισμός διευθύνσεων

- Από την εκτέλεση του προγράμματος φαίνεται πως και οι διευθύνσεις είναι ακέραιοι αριθμοί.
- Ένα ενδιαφέρον στοιχείο αυτού του μικρού προγράμματος είναι ότι δεν εκτυπώνει πάντα τις ίδιες διευθύνσεις.
- Αυτό συμβαίνει καθώς ο μεταγλωττιστής δεν δεσμεύει πάντοτε την ίδια διεύθυνση μνήμης για το ίδιο όνομα μεταβλητής.

Μεταβλητές δείκτη

- Οι διευθύνσεις μνήμης μπορούν να εκχωρηθούν σε ειδικές μεταβλητές.
- Οι μεταβλητές αυτές ονομάζονται δείκτες
- Παράδειγμα δήλωσης: TYPE *VARIABLE
- Ανάλογα με τον τύπο TYPE η μεταβλητή δείκτη μπορεί να δείξει σε διαφορετικού τύπου τιμές
- Οι μεταβλητές δείκτη ανεξάρτητα με το που δείχνουν είναι ακέραιες μεταβλητές

Εμφάνιση μεγέθους δεικτών

```
#include <stdio.h>
int main()
{
 int *x;
 double *z;
 int a=100;
 double b=200;
 printf("Size of variables:%d %d\n",
  sizeof(a),sizeof(b));
 printf("Size of pointers :%d %d\n",
  sizeof(x),sizeof(z));
 return 0;
}
```

Μέγεθος δεικτών

- Ο τελεστής sizeof() επιστρέφει το μέγεθος σε bytes ενός τύπου δεδομένων ή μιας μεταβλητής.
- Το μέγεθος αυτό πιθανόν να διαφέρει από την μια αρχιτεκτονική υπολογιστών σε άλλη.
- Σε 64bit μηχάνημα θέλουν 8 bytes αποθηκευτικού χώρου
- Μια συνήθης εκτύπωση είναι η:
  Size of variables: 4 8
  Size of pointers: 8 8

Αποαναφοροποίηση

- Οι δείκτες μπορούν να χρησιμοποιηθούν για την απόδοση τιμών στις μεταβλητές στις οποίες δείχνουν.
- Αυτό σημαίνει έμμεση αλλαγή τιμής σε μεταβλητή.
- Η διαδικασία αυτή ονομάζεται αποαναφοροποίηση.

Αλλαγή τιμών με δείκτη

```
#include <stdio.h>
int main()
{
 double x=1900;
 double *px=&x;
 *px=1950;
 printf("First print %lf %lf %x\n",x,*px,px);
 x=1970;
 printf("Second print %lf %lf %x\n",x,*px,px);
 return 0;
}
```

Αλλαγή τιμών με δείκτη

- Η πράξη *px=value δεν αλλάζει την μεταβλητή στην οποία δείχνει ο δείκτης px αλλά την τιμη που περιέχεται εκεί.
- Ανάλογα με το είδος της μεταβλητής που δείχνει ο δείκτης στα δεξιά της ανάθεσης η τιμή value θα έχει διαφορετικό τύπο
- Μπορεί να γίνει και εκτέλεση πράξεων με την χρήση δεικτών.

Πρόσθεση αριθμών με χρήση δεικτων

```
# include <stdio.h>
int main()
{
 double x=100;
 double y=200;
 double sum=0.0;
 double *px=&x;
 sum+=*px;
 px=&y;
 sum+=*px;
 printf("Final result: %lf\n",sum);
 return 0;
}
```

Ορίσματα δείκτη

- Τυπικά οι συναρτήσεις στην C δέχονται ορίσματα με τιμή, άρα δεν μπορούν να τα αλλάξουν.
- Με την χρήση δεικτών μπορεί να γίνει αλλαγή τιμής στα ορίσματα.
- 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 συνάρτηση.

Πέρασμα δείκτη σε συνάρτηση

```
# include <stdio.h>
void increase1(int x)
{
 x=x+1;
}
void increase2(int *x)
{
 *x=*x+1;
}
int main()
{
   int value=100;
   increase1(value);
   printf("Now value is %d \n",value);
   increase2(&value);
   printf("Now value is %d \n",value);
   return 0;
}
```

Πέρασμα δεικτών σε συνάρτηση

- Η συνάρτηση increase1 δεν μπορεί να αλλάξει το όρισμα που δέχεται, αφού αυτό περνάει με τιμή.
- Κάθε αλλαγή στο τοπικό της όρισμα αλλάζει μόνον την τοπική μεταβλητή
- Η συνάρτηση increase2 μπορεί να αλλάξει το όρισμα της, αφού γίνεται πέρασμα τιμής με αναφορά.

Αντιστροφή τιμών χωρίς δείκτες

```
void swap1(int a,int b)
{
   int t = a;
   a = b;
   b = t;
}
```

Αντιστροφή τιμής χωρίς δείκτες

- Η συνάρτηση swap1() αντιστρέφει μόνον τα τοπικά ορίσματα
- Απαιτείται χρήση δεικτών ώστε να γίνει αντιστροφή των ορισμάτων

Αντιστροφή τιμών με δείκτες

```
void swap2(int *a,int *b)
{
 int t = *a;
 *a=*b;
 *b = t;
}
```

Αντιστροφή τιμών με δείκτες

- Η συνάρτηση swap2() κάνει χρήση της αποαναφοροποίησης
- Γίνεται αντιστροφή του περιεχομένου των δεικτων και όχι των τοπικών ορισμάτων
- Οι δείκτες μπορούν να χρησιμοποιηθούν και για την επιστροφή πολλαπλών τιμών από συναρτήσεις.

Εύρεση ελαχίστου και μεγίστου με δείκτες

```
void findMinMax(int N,int *min,int *max)
{
  int i;
  int x;
  for(i=0;i<N;i++)
  {
    printf("Enter value ?");
    scanf("%d",&x);
    if(i==0 || x<*min) *min=x;
    if(i==0 || x>*max) *max=x;
  }
}
```

Δείκτης σε συνάρτηση

- Ένας δείκτης μπορεί να δείξει και σε συνάρτηση
- Ο χρήστης με ανάθεση μπορεί να αλλάξει την συνάρτηση που θα κληθεί

Παράδειγμα δείκτη σε συνάρτηση

```
# include <stdio.h>
int add(int a,int b){ return a+b; }
int sub(int a,int b){ return a-b; }
int main()
{
  int (*fpointer)(int,int) int v1,v2;
 int a=200,b=100;
 fpointer = add;
 v1=fpointer(a,b);
 fpointer=sub;
 v2=fpointer(a,b);
 printf("values: %d %d \n",v1,v2);
 return 0;
}
```

Ορισμοί

- ΣΓενικά κάθε αναφορά σε δείκτη μπορεί να θεωρηθεί και αναφορά σε πίνακα όπως και το αντίστροφο.
- Για παράδειγμα οι δηλώσεις: int x[5]; int *px=&x[0]; δημιουργούν έναν πίνακα ακεραίων με 5 συνεχόμενα στοιχεία στην μνήμη του υπολογιστή και αναθέτουν στον δείκτη px την διεύθυνση του πρώτου στοιχείου στον πίνακα.
- Στα στοιχεία του πίνακα x μπορεί να γίνει αναφορά με τον γνωστό τρόπο, δηλαδή x[0], x[1], x[2], x[3], x[4] αλλά και με την χρήση του δείκτη px

Αναφορά σε στοιχεία πίνακα με δείκτη

```
#include <stdio.h>
int main()
{
        int x[5],i;
        int *px=&x[0];
        x[0]=1;
        *(px+1)=10;
        *(px+2)=100;
        x[3]=1000;
        x[4]=10000;
        for(i=0;i<5;i++)
                printf("x=%d \n",x[i]);
        return 0;
}
```

Αναφορά σε τιμές πίνακα με δείκτη

- Η αναφορά *px=&x[0] αναθέτει στον δείκτη px την διεύθυνση της πρώτης τιμής του πίνακα x.
- Ο δείκτης px και ο πίνακας x είναι ισοδύναμοι
- Οι αναφορές *(px+i) και x[i] είναι ισοδύναμες.

Εύρεση τιμής σε πίνακα με χρήση δείκτη

```
int find(int value,int *px,int size)
{
        int pos = 0;
        while(pos!=size && *px!=value)
        {
                px++;
                pos++;
        }
        if(pos!=size) return pos;
        else return -1;
}
```

Εύρεση τιμής σε πίνακα με την χρήση δείκτη

- Κάθε μοναδίαια αύξηση στον δείκτη px τον μετακινεί στο επόμενο στοιχείο του πίνακα
- Η αναζήτηση με δείκτες ειναι πολύ πιο γρήγορη από ότι με στοιχεία πίνακα.

Υποπίνακας με την χρήση δείκτη

```
# include <stdio.h>
int main()
{
        int x[]={10,20,200,300,400};
        int startPos=3;
        int *subX=&x[startPos];
        int i;
        for(i=startPos;i<5;i++)
        {
          printf("subelement␣%d␣\n",
           subX[i-startPos]);
        }
        return 0;
}
```

Υποπίνακες

- Ένας δείκτης μπορεί να δείξει όχι μόνο σε απλές μεταβλητές αλλά και σε πίνακες ή και στοιχεία πινάκων
- Με την χρήση τέτοιων δεικτών δημιουργούνται τμήματα πινάκων
- Χρειάζεται προσοχή στον χειρισμό τους, καθώς εύκολα μπορούμε να βγούμε εκτός ορίων του αρχικού πίνακα.

Αλλαγή πίνακα από συνάρτηση

```
void alterArray(int *x,int size)
{
        int pos=0;
        while(pos!=size)
        {
          if(pos%2==0)
            *x++=100;
          else
            *x++=200;
          pos++;
        }
}
```

Αλλαγή πίνακα

- Η συνάρτηση alterArray αλλάζει τα στοιχεία σε ζυγές θέσεις σε 100 και τα στοιχεία σε μονές θέσεις σε 200
- Η αλλαγή πίνακα με δείκτες είναι ταχύτερη μέθοδος από ότι η πρόβαση με [ ], καθώς απαιτείται η χρήση μιας μόνο μεταβλητής

Υπολογισμός μήκους αλφαριθμητικού

```
int mylen(char *s)
{
        int count=0;
        while(*s++) count++;
        return count;
}
```

Ορισμοί

- Οι δείκτες βρίσκουν εφαρμογή όχι μόνο στο πέρασμα τιμών σε συναρτήσεις αλλά και στην δυναμική κατανομή μνήμης, την δημιουργία δηλαδή μεταβλητών κατά την διάρκεια εκτέλεσης του προγράμματος και όχι με την εκκίνησή του.
- Έτσι ο προγραμματιστής μπορεί να δημιουργήσει μεταβλητές όταν τις χρειάζεται και να τις διαγράψει όταν πλέον δεν θα του είναι χρήσιμες.

Συναρτήσεις μνήμης

- Η γλώσσα C διαθέτει στην βιβλιοθήκη stdlib.h τις συναρτήσεις malloc() και free() για την διαχείριση δυναμικών μεταβλητών.
- Η συνάρτηση malloc() δεσμεύει μνήμη
- Η συνάρτηση free() διαγράφει την μνήμη που έχει κατανεμηθεί με την malloc()

Δημιουργία απλής μεταβλητής

```
# include <stdio.h>
# include <stdlib.h>
int main()
{
        int *x=NULL;
        printf("X pointer %x \n",x);
        x=(int*)malloc(sizeof(int));
        *x=100;
        printf("Pointer: %x Value: %d\n",x,*x);
        free(x);
        return 0;
}
```

Δημιουργία απλής μεταβλητής

- Η συνάρτηση malloc() δέχεται σαν όρισμα το πλήθος των bytes που θα δημιουργήσει και επιστρέφει σαν αποτέλεσμα ένα δείκτη σε αυτά τα bytes.
- Είναι χρήσιμη η συνάρτηση sizeof() για την εύρεση των bytes που απαιτεί μια μεταβλητή.
- Από την στιγμή αυτή και μετά έχει δημιουργηθεί μια νέα μεταβλητή και ο χρήστης μπορεί να την χρησιμοποιήσει όπως και κάθε άλλη στατική μεταβλητή.
- Όταν η μεταβλητή δεν θα είναι πλέον χρήσιμη γίνεται αποδέσμευση της μνήμης με την συνάρτηση free().

Δημιουργία μονοδιάστατου πίνακα

```
# include <stdio.h>
# include <stdlib.h>
int main()
{
        int n,i;
        int *x=NULL;
        printf("Dimension?\n");
        scanf("%d",&n);
        x=(int*)malloc(sizeof(int) * n);
        for(i=0;i<n;i++)  x[i]=i+1;
        for(i=0;i<n;i++)
                printf("%d \n",x[i]);
        free(x);
        return 0;
}
```

Σύνοψη

- Διευθύνσεις μεταβλητων
- Μεταβλητές δείκτη
- Δείκτες και συναρτήσεις
- Δείκτες και πίνακες
- Δυναμική δημιουργία μεταβλητών