Διάλεξη 11η: Δείκτες, μέρος 1

Τμήμα Επιστήμης Υπολογιστών, Πανεπιστήμιο Κρήτης

Εισαγωγή στην Επιστήμη Υπολογιστών

Βασίζεται σε διαφάνειες του Κ. Παναγιωτάκη

Δείκτες

- Τι είναι:
  - Τύπος μεταβλητών (όπως `int`, `float`, κλπ)
- Αποθηκεύουν τη διεύθυνση στη μνήμη άλλων μεταβλητών
- Χρήσεις
  - Δυναμική διαχείριση μνήμης και δυναμικές δομές δεδομένων
  - Call-by-reference
  - Στενή σχέση με τους πίνακες
- Πηγή πολλών λαθών που δύσκολα ανιχνεύονται

Παράδειγμα

Παράδειγμα δείκτη

```
int counter = 42;
int *pointer_to_counter = &counter;
```

Όνομα	Θέση μνήμης	Περιεχόμενα
	...	...
counter	1073741824	42
pointer_to_counter	2147483648	1073741824
	...	...

Δήλωση Δεικτών

- Χρησιμοποιούμε το * για δήλωση δείκτη

```
int *ptr;
```

  - Ορίζει δείκτη σε ακέραιο (`int`)
- Δήλωση πολλών μεταβλητών τύπου δείκτη απαιτεί τη χρήση ενός * μπροστά από κάθε μεταβλητή

```
int *ptr1, *ptr2;
```

- Μπορούμε να δηλώσουμε δείκτη σε οποιοδήποτε είδος μεταβλητής (ακόμα και άλλο δείκτη)

```
double *ptr2dbl;
char **ptr2ptr2char;
```

  - Δείκτης σε `double`
  - Δείκτης σε *δείκτη* σε `char`

Τελεστές Δεικτών

- Τελεστής &: η διεύθυνση μιας μεταβλητής

Τελεστής

```
q = &a;
```

  - Εφαρμόζεται σε οποιαδήποτε μεταβλητή
  - Επιστρέφει τη διεύθυνση στην οποία είναι αποθηκευμένη η μεταβλητή
- Τελεστής *: αναφορά (dereference) σε δείκτη

Τελεστής

```
a = *q;
```

  - Εφαρμόζεται σε δείκτη
  - Επιστρέφει τα περιεχόμενα της διεύθυνσης στην οποία δείχνει ο δείκτης

Παράδειγμα

pointer.c

```
int y = 5;
int *ptr;
ptr = &y;
*ptr = *ptr * 2;
```

- Μπορεί να αλλάξει η τιμή των μεταβλητών με τη βοήθεια δεικτών
  - Η τιμή του `y` γίνεται ίση με 10
  - Η μεταβλητή `y` αλλάζει τιμή χωρίς να υπάρχει εντολή που να αποθηκεύει κάτι άμεσα στο `y`
  - Η έκφραση `*ptr` είναι “ψευδώνυμο” (alias) της μεταβλητής `y`
- Τα `&` και `*` είναι ανάποδα και αλληλοακυρώνονται
  - `*(&y) == y`
  - `&(*ptr) == ptr` /* Εκτός αν ο δείκτης είναι κενός */

Κλήση με δείκτη

callbyref.c

```
#include <stdio.h>
void cube(int *);

int main(void) {
  int number = 5;
  printf("The original value of number is %d\n", number);
  cube(&number);
  printf("The new value of number is %d\n", number);
  return 0;
}
void cube(int *ptr) {
  *ptr = *ptr * *ptr * *ptr;
}
```

Έξοδος

```
The original value of number is 5
The new value of number is 125
```

Παράδειγμα

bubblesort.c

```c
#include<stdio.h>
#define SIZE 10
void bubblesort(int *, const int);
void swap(int *, int *);

void bubblesort(int *array, const int size)
{
  int pass, j;
  for(pass = 0; pass < size -1; pass++) {
    for(j = 0; j < size - 1; j++) {
      if(array[j] > array[j + 1]) {
        swap(&array[j], &array[j + 1]);
      }
    }
} }

void swap(int *ptr1, int *ptr2) {
  int hold = *ptr1;
  *ptr1 = *ptr2;
  *ptr2 = hold;
}
```

```c
int main(void)
{
  int i, a[SIZE] =
    { 2, 6, 4, 8, 10, 12, 89, 68, 42, 37 };

printf("Data items in original order\n");
  for(i = 0; i < SIZE; i++) {
    printf("%4d", a[i]);
  }
  printf("\n");

  bubblesort(a, SIZE);

  printf("Data items in ascending order\n");
  for(i = 0; i < SIZE; i++) {
    printf("%4d", a[i]);
  }
  printf("\n");

  return 0;
}
```

Παράδειγμα (2)

Έξοδος

```
Data items in original order
   2   6   4   8  10  12  89  68  42  37
Data items in ascending order
   2   4   6   8  10  12  37  42  68  89
```

Αριθμητική Δεικτών

- Στους δείκτες μπορούμε να πραγματοποιούμε αριθμητικές πράξεις
- `++` ή `--`
  - Ο δείκτης δείχνει στο επόμενο ή προηγούμενο αντικείμενο
  - Π.χ., αν είναι δείκτης σε ακέραιο στον επόμενο ή προηγούμενο ακέραιο
- Αντίστοιχα για `+`, `+=`, `-` ή `-=`
- Οι δείκτες μπορούν να αφαιρούνται ο ένας από τον άλλο
  - Προκύπτει *απόσταση*
  - Π.χ.: πόσοι ακέραιοι υπάρχουν από το `*ptr1` μέχρι το `*ptr2`
- Αυτές οι πράξεις έχουν νόημα όταν γίνονται πάνω σε δείκτες που δείχνουν στα στοιχεία ενός πίνακα

Παράδειγμα

pointer-arith.c

```
int array[5] = {1, 2, 3, 4, 5};
int *ptr;

ptr = array;
ptr = &array[0];
ptr++;
ptr += 2;
```

- Ο δείκτης δείχνει στην αρχή του πίνακα
- Αυξάνοντας το δείκτη κατά 1 δείχνει στο επόμενο στοιχείο, `sizeof(int)` bytes μετά
- (όχι στην επόμενη διεύθυνση μνήμης)

Πίνακες και Δείκτες

pointer-arith.c

```
int array[5] = {1, 2, 3, 4, 5};
int *ptr;

ptr = array;
ptr = &array[0];
ptr++;
ptr += 2;
```

- Υπάρχουν πολλοί τρόποι να βρούμε το τελευταίο στοιχείο του πίνακα
  - `array[4]`
  - `*(array + 4)`
  - `ptr[4]`
  - `*(ptr + 4)`

Πίνακες και Δείκτες: Διαφορές

- `int *ptr;`
  - Η μεταβλητή `ptr` είναι δείκτης σε ακέραιο
  - Δεσμεύεται μνήμη μόνο για την `ptr` (όχι για αυτό που δείχνει)
  - Τα περιεχόμενα της μεταβλητής `ptr` (μια διεύθυνση μνήμης) μπορούν να αλλάξουν
- `int array[10];`
  - Η μεταβλητή `array` είναι σταθερός δείκτης σε ακέραιο
  - Δεσμεύεται μνήμη για 10 ακέραιους
  - Τα περιεχόμενα της μεταβλητής `array` (η διεύθυνση μνήμης του πρώτου στοιχείου) *δεν* μπορούν να αλλάξουν

Πίνακες και Δείκτες: Ορίσματα

- Αν μια συνάρτηση παίρνει δείκτη τότε δέχεται και πίνακα
- Αν μια συνάρτηση παίρνει πίνακα τότε δέχεται και δείκτη
- Και στις δύο περιπτώσεις περνά μια διεύθυνση στη μνήμη

Παράδειγμα

```
void get_array(int a[]);
void get_ptr(int *b);

int main()
{
  int *ptr, array[10];
  get_ptr(array);
  get_array(ptr);
}
```

- *Προσοχή με τη δέσμευση της μνήμης*
  - Αν το πρόγραμμα προσπαθήσει να διαβάσει διεύθυνση μνήμης που δεν είναι δεσμευμένη: crash!

Παράδειγμα

ptr-arith2.c

```
#include <stdio.h>

int main() {
  int array[10] = {2, 3, 4, 5, 6, 7, 8, 9, 10, 1};
  int *ptr, i;

  printf("First  way to traverse the array: indexing.\n");
  for (i  = 0; i < 10; i++) {
    printf("array[%d] = %d\n", i, array[i]);
  }

  printf("\nSecond way to traverse the array: pointer arithmetic.\n");
  for (i=0; i < 10; i++) {
    printf("array[%d] = %d\n", i, *(array + i));
  }
```

Παράδειγμα (2)

ptr-arith2.c

```
  printf("\nThird way to traverse the array: pointer arithmetic.\n");
  /* set the pointer to the beginning of the array */
  ptr = array;
  for (i = 0; i < 10; i++) {
    printf("array[%d] = %d\n", i, *(ptr + i));
  }

  printf("\nFourth way to traverse the array: pointer arithmetic.\n");
  ptr = array;
  for (i = 0 ; i < 10; i++) {
    printf("array[%d] = %d, %d\n", i, *(ptr++), ptr);
  }

  printf("Check\n");

  return 0;
}
```