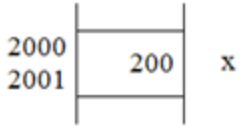
מבוא לתכנות בשפת C

מצביעים והקצאה דינאמית

Tzachi (Isaac) Rosen

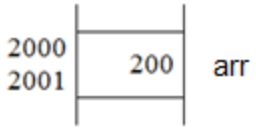
כתובות של משתנים



- לכל משתנה כתובת של המקום שלו בזיכרון
- כבר ראינו:
  - שם של מערך הוא למעשה הכתובת של התא הראשון (באינדקס 0) של המערך
  - להזכירכם: תא של מערך הינו משתנה לכל דבר ועניין
- בהינתן משתנה x, אנחנו יכולים בעזרת הפעולה &x לקבל את הכתובת של x
  - להזכירכם: scanf(“%d”, &x)
- שימו לב: הפעולה & יכולה להתבצע רק על משתנים
  - למשל אם x משתנה שלם, אז &(x +2) אינו חוקי, שכן x + 2 הוא ביטוי שנותן ערך שלם, ולערך שלם אין כתובת
- כיוון שאין משתנה ללא סוג (type), הכתובת היא תמיד של משתנה מסוג מסוים
  - כך, אם x הוא משתנה מסוג int, אז &x היא כתובת למשתנה מסוג int, או בקיצור כתובת ל- int
  - ב- scanf(“%d”, &x), ה- scanf מצפה לקבל כתובת של int, אם לא זו תהיה שגיאה

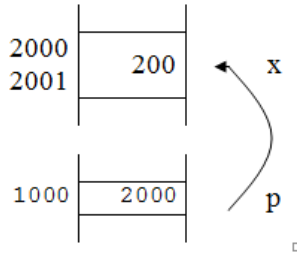
שחזור משתנים לפי כתובות

- בהינתן כתובת של משתנה אנחנו יכולים באמצעות הפעולה * לשחזר מהכתובת את המשתנה
  - כך, אם x הוא משתנה, &x הוא כתובת של x, ו- *&x הוא שוב x, ו- *&x = 3 מכניס ל- x את הערך 3
  - כך, אם arr הוא שם של מערך (כלומר כתובת של התא הראשון (באינדקס 0) של המערך), *arr הוא התא הראשון של המערך, ו- *arr = 200 מכניס לתא הראשון של המערך את הערך 200



מצביעים

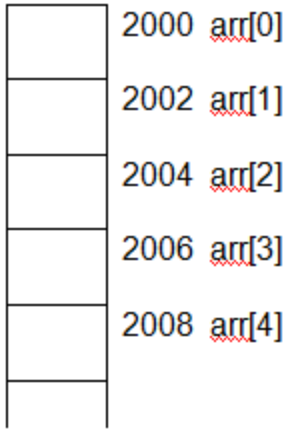
- את הכתובות אנחנו יכולים לשמור במשתנים (בדומה לאופן בו אנו שומרים מספרים)
- משתנה ששומר כתובת של משתנה אחר נקרא מצביע
- משתנה מסוג מצביע מוגדר באמצעות הוספת * לפני שמו
  - כך int x מגדיר משתנה שלם בשם x, בעוד ש- int * p מגדיר מצביע (משתנה שמכיל כתובת) ל- int בשם p



- מצביעים מקבלים כתובות
  - אם x הוא משתנה מסוג int אנחנו יכולים לכתוב p = &x ולשמור במשתנה p את הכתובת של x
  - שימו לב: ב- p ניתן לשמור רק מצביעים למשתנים מסוג int. למשל, אם y הוא משתנה מסוג double, ההשמה p = &y לא עוברת קומפילציה
- אם p מכיל את הכתובת של x אז p* הוא x, ו- p* = 200 מכניס ל- x את הערך 200
  - שימו לב: p = 200 לא בסדר משום ש- p מקבל כתובות ולא מספרים שלמים
- אנחנו גם משתמשים בצורה חופשית במילה מצביע כדי לציין כתובת

חישובי כתובות

- להזכירכם:
  - מערך הוא רצף של תאים (משתנים) מאותו סוג
  - שם של מערך הוא הכתובת של התא הראשון
- שאלה: מה הכתובת של התא השני?
  - הכתובת של התא השני היא הכתובת של התא הראשון ועוד מספר הבתים שהתא הראשון מכיל
- C מאפשר לנו לדלג בין כתובות של תאים במערך באופן מאוד פשוט
  - אם arr הוא שם של מערך (כלומר כתובת של התא הראשון של המערך) אז:
    - arr+1 היא הכתובת של התא השני
    - arr+7 היא הכתובת של התא השביעי
    - (&arr[3]) – 2 היא הכתובת של התא השני (באינדקס 2)
  - C יודע לפי סוג הכתובת כמה תאים לדלג
- שאלה: כמה תאים יש בין שתי כתובות של תאים של אותו מערך?
  - ההפרש בין הכתובת הגדולה לקטנה חלקי הגודל של התאים
- C מאפשר לנו לחשב את מספר התאים באמצעות ההפרש בין הכתובת הגדולה לכתובת הקטנה
  - כך, אם p = &arr[2] ו- q = &arr[5] אז q – p = 3
- שאלה: איזה תא בא אחרי איזה תא
- C מאפשר לנו לחשב להשוות כתובות
  - כך, אם p = &arr[2] ו- q = &arr[5] אז q > p
  - אפשר גם == ו- =! וכול השאר



דוגמה: 5 צורות לאתחל תאים של מערך

```
int main() {
        int arr[SIZE], *p = arr;
        int i;
        for (i = 0; i < SIZE; ++i)
                arr[i] = i;
        for (i = 0; i < SIZE; ++i)
                *(arr + i) = i;
        for (i = 0; i < SIZE; ++i)
                *(p + i) = i;
        for (i = 0; i < SIZE; ++i) {
                *p = i;
                ++p;
        }
        for (p = arr; p < arr+SIZE; ++p)
                *p = p - arr;
        return 0;
}
```

מצביעים כערכים מוחזרים וכפרמטרים של פונקציות

```
#include <stdio.h>
int concat(char * s1, char * s2);
int main() {
        char s1[7] = "abc";
        char * s2 = "def";
        int n = concat(s1, s2);
        printf("%s %d\n", s1, n);
        return 0;
}
int concat(char * s1, char * s2) {
        char * start = s1;
        while (*s1 != '\0')
                ++s1;
        *s1 = *s2;
        while (*s2 != '\0') {
                ++s1;
                ++s2;
                *s1 = *s2;
        }
        return s1 - start;
}
```

חיפוש בינארי

```
int * binSearch(int * left, int * right, int k) {
        int * mid;
        if (left > right)
                return NULL;
        mid = left + (right - left) / 2;
        if (*mid == k)
                return mid;
        if (*mid > k)
                return binSearch(left, mid - 1, k);
        return binSearch(mid + 1, right, k);
}
```

חיפוש בינארי

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#define SIZE 10
#define RANGE 20
#define RAND (int) ((double) rand() * RANGE / RAND_MAX)
int * binSearch(int * left, int * right, int k);
int main() {
        int arr[SIZE] = { 1, 3, 4, 6, 7, 11, 15, 16, 18, 19 };
        int i, k, *p;
        for (i = 0; i < SIZE; ++i)
                printf("[%d] = %d ", i, arr[i]);
        printf("\n");
        // random initialization
        srand(time(NULL));
        k = RAND;
        p = binSearch(arr, arr + SIZE - 1, k);
        if (p != NULL)
                printf("%d is found in arr[%d]\n", k, p - arr);
        else
                printf("%d doesn't found", k);
        return 0;
}
```

הקצאות ושחרור זיכרון דינמיים

- לפעמים יש צורך להקצות מקום לשמירת נתונים בזמן ריצה
  - למשל כאשר קולטים מספר לא ידוע מראש של נתונים מהמשתמש
- יתר על כן. לפעמים מערך יכול לשמש אותנו רק חלק מזמן ריצת התוכנית, ולכן שמירתו לאורך כל ריצת התוכנית מיותרת ומבזבזת זיכרון
- שימו לב: הגדרה של מערך היא תמיד עם מספר קבוע של תאים, ואם לא ניתן לחזות מראש את מספר התאים הרצוי, לא ניתן להגדיר מראש מערך שיכיל את הנתונים או שמגדירים מערך גדול מידי ובזבזני
- הפתרון: הקצאה ושחרור זיכרון דינאמיים

free -ו alloc

```
void *malloc ( size_t size );
```
Allocates a memory block of *size* bytes.

```
void *calloc ( size_t n , size_t size );
```
Allocates enough memory to hold an array of *n* elements, each of which has the size *size*, and initializes every byte with the value 0.

```
void *realloc ( void *ptr , size_t n );
```
Changes the length of the memory block referenced by *ptr* to the new length *n*.
If the memory block has to be moved in order to provide the new size, then its current contents are automatically copied to the new location.

```
void free ( void *ptr );
```
Releases the memory block referenced by *ptr*.

דוגמה

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
char * cpystr(char *s);
char * catstr(char *s1, char *s2);
int main() {
        char * s = "Hello";
        char * cs = cpystr(s);
        char * ccs = catstr( "Hello", " World");
        printf("s = \"%s\", cs = \"%s\", ccs = \"%s\"\n", s, cs, ccs);
        free(cs);
        free(ccs);
        return 0;
}
char * cpystr(char *s) {
        int n = strlen(s);
        char * r = (char *) malloc(n + 1);
        return strcpy(r, s);
}
char * catstr(char *s1, char *s2) {
        int n = strlen(s1) + strlen(s2);
        char * r = (char *) malloc(n + 1);
        strcpy(r, s1);
        return strcat(r, s2);
}
```

sizeof

sizeof unary-expression
sizeof(type-name)

```
#include <stdio.h>
#define  printsize(x) printf ("sizeof("#x")= %d\n", sizeof(x))
int main() {
        int tab[100];
        printsize (char);       // sizeof(char)= 1
        printsize (double);     // sizeof(double)= 8
        printsize (float);      // sizeof(float)= 4
        printsize (int);        // sizeof(int)= 4
        printsize (long);       // sizeof(long)= 4
        printsize (long long);  // sizeof(long long)= 8
        printsize (short);      // sizeof(short)= 2
        printsize (int *);      // sizeof(int *)= 4
        printsize (void *);     // sizeof(void *)= 4
        printsize (tab);        // sizeof(tab)= 400
        printsize (tab[0]);     // sizeof(tab[0])= 4
        printsize (10);         // sizeof(10)= 4
        printsize (10.0);       // sizeof(10.0)= 8
        printsize (*tab);       // sizeof(*tab)= 4
        printsize (*tab+10.0);  // sizeof(*tab+10.0)= 8
        return 0;
}
```

דוגמה: מערך דינאמי

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

int* read_data(int *);
void print_array(int *, int);

int main() {
        int n;
        int *array;
        array = read_data(&n);
        print_array(array, n);
        free(array);
        return 0;
}
```

דוגמה: מערך דינאמי

```
void print_array(int *a, int n) {
        int i;
        for (i = 0; i < n; ++i)
                printf("%d  ", a[i]);
        printf("\n");
}
Int *read_data(int *pn) {
        int num, size = 0;
        int *a;
        a = (int*) malloc(0);
        printf("Enter the series: ");
        while (scanf("%d", &num) == 1) {
                ++size;
                a = (int *) realloc(a, size * sizeof(int));
                a[size - 1] = num;
        }
        *pn = size;
        return a;
}
```

דוגמה: מערך של מערכים

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_SIZE 256
char **read_data(int *);
void print_array(char **, int);
void freeAll(char **, int);
int main() {
        int n;
        char **array;
        array = read_data(&n);
        print_array(array, n);
        freeAll(array, n);
        return 0;
}
```

```
Enter names: Hello World How are you
0: Hello
1: World
2: How
3: are
4: you
```

דוגמה: מערך של מערכים

```
char **read_data(int *pn) {
        int size = 0;
        char **a;
        char name[MAX_SIZE];
        printf("Enter the names: ");
        fflush(stdout);
        a = (char **) malloc(0);
        while (scanf("%s", name) == 1) {
                ++size;
                a = (char **) realloc(a, size * sizeof(char *));
                a[size - 1] = malloc((strlen(name) + 1) * sizeof(char));
                strcpy(a[size - 1], name);
        }
        *pn = size;
        return a;
}
```

דוגמה: מערך של מערכים

```
void print_array(char **a, int n) {
        int i;
        for (i = 0; i < n; ++i)
                printf("%d: %s\n", i, a[i]);
}
void freeAll(char **a, int n) {
        int i;
        for (i = 0; i < n; ++i)
                free(a[i]);
        free(a);
}
```

הערך NULL

- במקרים בהם צריך מצביע כתופס מקום משתמשים ב- NULL (הכתובת 0)

הטיפוס void *

- void * מציין מצביע כללי
- כל מצביע מומר אוטומטית ל- void *