Imperatív programozás

Tömbök és mutatók

Kozsik Tamás és mások

Eötvös Loránd Tudományegyetem

2022. szeptember 26.

Tartalomjegyzék

Tömb fogalma

- Azonos típusú (méretű) objektumok egymás után a memóriában
- Bármelyik eleme hatékonyan elérhető
- Rögzített számú objektum

```
int vector[4];
int matrix[5][3];      /* 15 elem sorfolytonosan */
```

Indexelés 0-tól

- `vector[i]` címe: vector címe + i * sizeof(int)
- `matrix[i][j]` címe: matrix címe + (i * 3 + j) * sizeof(int)

Tömbök indexelése

- `int t[] = {1, 2, 3, 4}`
- 0-tól indexelünk
- Hossz futás közben ismeretlen
- fordítás közben: `sizeof(t) / sizeof(t[0])`
- Hibás index: definiálatlanság

Példák

```
int t[5] = {2, 6, 5, 9, 1};

int sum = 0;
for (int i = 0; i < 5; ++i)
  sum += t[i];

printf("Az elemek összege: %d\n", sum);

int max = t[0];
for (int i = 0; i < 5; ++i)
  if (t[i] > max)
    max = t[i];

printf("A legnagyobb elem: %d\n", max);
```

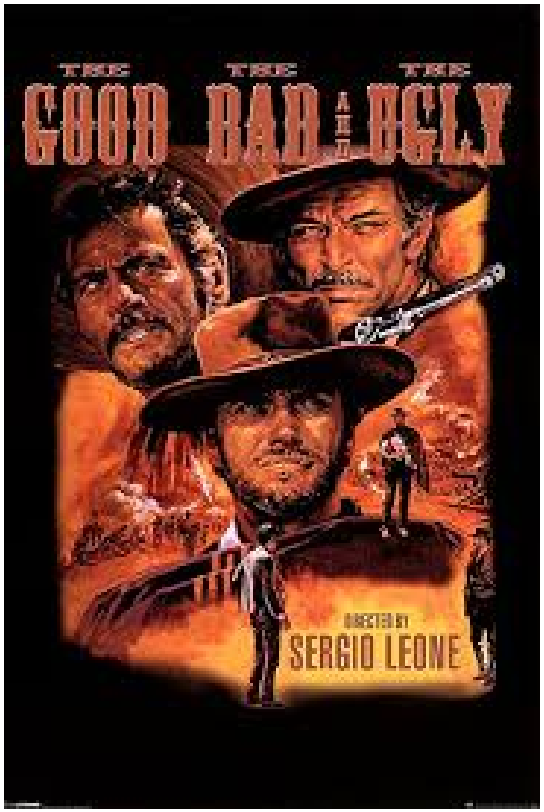
C tömbök deklarációja

```
int a[4];                          /* 4 elemű, inicializálatlan */
int b[] = {1, 5, 2, 8};            /* 4 elemű */
int c[8] = {1, 5, 2, 8};           /* 8 elemű, 0-kkal feltöltve */
int d[3] = {1, 5, 2, 8};           /* 3 elemű, felesleg eldobva */

int m[5][3];                       /* 15 elem, sorfolytonosan */
int n[][3] = {{1,2,3},{2,3,4}}; /* méret nem elhagyható! */
int q[3][3][4][3];                 /* 108 elem */

char s[] = "alma";
char z[] = {'a','l','m','a','\0'};
```

Szöveg




```
char good[] = "good";
char  bad[] = {'b', 'a', 'd'};
char ugly[] = {'u', 'g', 'l', 'y', '\0'};
printf("%s %s %s", good, bad, ugly);
```

Mutatók

- Más objektumok címét tárolja: „mutat rájuk” (indirekció)
- Típusbiztos

```
int i = 42;
int* p = &i;
printf("%d %d", i, *p);      /* 42 42 */
*p = 5;
printf("%d %d", i, *p);      /* 5 5 */
int j;                       /* Unspecified value */
p = &j;
*p = 10;
printf("%d", j);             /* 10 */
p = NULL;                    /* Sehová sem mutat */
```

Tömbök és mutatók kapcsolata

- Tömb konvertálódik mutatóvá
- Nem ekvivalensek!

```
int arr[] = {1, 2, 3};
arr = {1, 2, 4};    /* Fordítási hiba */

int* ptr = arr;
int* q = &arr[0];

arr[1] = 5;
ptr[1] = 5;

int t[] = {3, 2, 1};
ptr = t;    /* ok */
arr = t;    /* fordítási hiba */

printf("%d %d\n", sizeof(arr), sizeof(ptr));
```

Példa

Feltételes keresés

```
int t[] = {6, 2, 8, 7, 3};
int* p = NULL;

for (int i = 0; i < 5; ++i)
  if (t[i] % 2 == 1)
    p = &t[i];

if (p)
  printf("Első páratlan szám: %d\n", *p);
else
  printf("Nincs páratlan szám\n");
```

Mutató-aritmetika

Léptetések

```
int v[] = {6, 2, 8, 7, 3};
int* p = v;          /* v: 6, 2, 8, 7, 3 */
int* q = p + 3;      /*    p        q    */

q = v + 3;           /* v konvertálódik  */

++p;                 /* v: 6, 5, 8, 7, 3 */
*p = 5;              /*       p     q    */

p += 2;              /* v: 6, 5, 8, 1, 3 */
*q = 1;              /* v:          pq   */

q -= 2;              /* v: 6, 2, 8, 1, 3 */
*q = 2;              /*       q     p    */
```

Mutató-aritmetika

Összehasonlítások

```
int v[] = {6, 2, 8, 7, 3};
int* p = v;
int* q = v + 3;

if (p == q) { ... }
if (p != q) { ... }
if (p <  q) { ... }
if (p <= q) { ... }
if (p >  q) { ... }
if (p >= q) { ... }
```

Mutató-aritmetika

Indexelés

```
char str[] = "hello";

str[1] = 'o';
*(str + 1) = 'o';
*(1 + str) = 'o';

printf("%s\n", str + 3);
printf("%c\n", 3[str]);
```

Mutató-aritmetika

Kivonás

```
int v[] = {6, 2, 8, 7, 3};

int* p = v;
int* q = v + 3;

int i = q - p;      /* 3 */
```

Példa

Szöveg hossza

```
char str[] = "hello";
char* p = str;
char* q = str;

while (*q != '\0')
  ++q;

printf("Szöveg hossza: %d\n", q - p);
```

Tömbök átadása paraméterként

Beégetett érték

```
double distance(double a[3], double b[3]) {
  double sum = 0.0;
  unsigned int i;
  for (i = 0; i < 3; ++i) {   /* beégetett érték :-( */
    double delta = a[i] - b[i];
    sum += delta*delta;
  }
  return sqrt(sum);
}

int main() {
  double p[3] = {36, 8, 3}, q[3] = {0, 0, 0};
  printf("%f\n", distance(p, q));
  return 0;
}
```

Tömbök átadása paraméterként

Fordítási időben rögzített méret

```
#define DIMENSION 3

double distance(double a[DIMENSION], double b[DIMENSION]) {
  double sum = 0.0;
  unsigned int i;
  for (i = 0; i < DIMENSION; ++i) {
    double delta = a[i] - b[i];
    sum += delta * delta;
  }
  return sqrt(sum);
  }

int main() {
  double p[DIMENSION] = {36, 8, 3}, q[DIMENSION] = {0, 0, 0};
  printf("%f\n", distance(p, q));
  return 0;
}
```

Tömbök átadása paraméterként

Futási időben rögzített méret

```
double distance(double a[], double b[]) {
  double sum = 0.0;
  unsigned int i;
  for (i = 0; i < ???; ++i) {
    /* nem tudjuk a méretét */
    double delta = a[i] - b[i];
    sum += delta * delta;
  }
  return sqrt(sum);
}

int main() {
  double p[] = {3.0, 4.0}, q[] = {0.0, 0.0};
  printf("%f\n", distance(p, q));
  return 0;
}
```

Tömbök átadása paraméterként

Hibás megközelítés

```
double distance(double a[], double b[]) {
  double sum = 0.0;
  unsigned int i;
  for (i = 0; i < sizeof(a) / sizeof(a[0]); ++i) {
    double delta = a[i] - b[i];
    sum += delta * delta;
  }
  return sqrt(sum);
}

int main() {
  double p[] = {3.0, 4.0}, q[] = {0.0, 0.0};
  printf("%f\n", distance(p, q));
  return 0;
}
```

Tömbök átadása paraméterként

Helyesen

```
double distance(double a[], double b[], int dim) {
  double sum = 0.0;
  unsigned int i;
  for (i = 0; i < dim; ++i) {
    double delta = a[i] - b[i];
    sum += delta * delta;
  }
  return sqrt(sum);
}

int main() {
  double p[] = {3.0, 4.0}, q[] = {0.0, 0.0};
  printf("%f\n", distance(p, q, sizeof(p) / sizeof(p[0])));
  return 0;
}
```

Több dimenziós tömbök paraméterként

```
double m[4][4] = {{1,2,3,4}, {1,2,3,4}, {1,2,3,4}, {1,2,3,4}};

transpose(m);

{
  int i, j;
  for (i = 0; i < 4; ++i) {
    for (j = 0; j < 4; ++j) {
      printf("%3.0f", m[i][j]);
    }
    printf("\n");
  }
}
```

Több dimenziós tömbök paraméterként

Beégetett méret

```
void transpose(double matrix[4][4]) {  /* double matrix[][4] */
  int size = sizeof(matrix[0]) / sizeof(matrix[0][0]);
  int i, j;
  for (i = 1; i < size; ++i) {
    for (j = 0; j < i; ++j) {
      double tmp = matrix[i][j];
      matrix[i][j] = matrix[j][i];
      matrix[j][i] = tmp;
    }
  }
}

double m[4][4] = {{1,2,3,4}, {1,2,3,4}, {1,2,3,4}, {1,2,3,4}};
transpose(m);
```

Több dimenziós tömbök paraméterként

Sorfolytonos ábrázolás

```
void transpose(double* matrix, int size) {  /* size*size double */
  int i, j;
  for (i = 1; i < size; ++i) {
    for (j = 0; j < i; ++j) {
      int idx1 = i * size + j,  /* matrix[i][j] helyett */
          idx2 = j * size + i;  /* matrix[j][i] helyett */
      double tmp = matrix[idx1];
      matrix[idx1] = matrix[idx2];
      matrix[idx2] = tmp;
    }
  }
}

double m[4][4] = {{1,2,3,4}, {1,2,3,4}, {1,2,3,4}, {1,2,3,4}};
transpose(&m[0][0], 4);  /* transpose((double*)m, 4) */
```

Több dimenziós tömbök paraméterként

Mutatók tömbje

```
void transpose(double* matrix[], int size) {
  int i, j;
  for (i = 1; i < size; ++i) {
    for (j = 0; j < i; ++j) {
      double tmp = matrix[i][j];
      matrix[i][j] = matrix[j][i];
      matrix[j][i] = tmp;
    }
  }
}

double m[4][4] = {{1,2,3,4}, {1,2,3,4}, {1,2,3,4}, {1,2,3,4}};
double* helper[4]; for (i = 0; i < 4; ++i) helper[i] = m[i];
transpose(helper, 4);
```

Parancssori argumentumok

```
int main(int argc, char* argv[]) { ... }
```

- `argc`: pozitív szám
- `argv[0]`: program neve
- `argv[i]`: parancssori argumentum ($1 \leq i <$ `argc`)
  - Karaktertömb, végén `\0`
- `argv[argc]`: `NULL`

Parancssori argumentumok

```
int main(int argc, char* argv[]) {
  for (int i = 0; i < argc; ++i)
    printf("%d -> %s\n", i, argv[i]);
}
```

```
$ ./a.out one two three
0 -> ./a.out
1 -> one
2 -> two
3 -> three
```