Programozás I. gyakorlat

Összetett adatszerkezetek

Emlékeztető

Írjatok programot, amely lefoglal egy n méretű int tömböt, beállítja az elemeit, kiírja azokat, majd felszabadítja!

Feladat

- Írj programot, amely bekér és kiír egy 2D koordinátát, használj csak struktúrát!

Megoldás

```c
#include <stdio.h>

struct pont {
   int x, y;
};
int main() {
   struct pont p;
   printf("x: ");
   scanf("%d", &p.x);
   printf("y: ");
   scanf("%d", &p.y);
   printf("(%d; %d)\n", p.x, p.y);
   return 0;
}
```

Feladat

- Módosítsd az előző programot úgy, hogy a koordinátákat külön függvény állítsa be!

Megoldás

```c
#include <stdio.h>

struct pont {
   int x, y;
};
void beallit(struct pont * p) {
   p->x = 40;
   (*p).y = 60;
}
int main() {
   struct pont p;
   beallit(&p);
   printf("(%d; %d)\n", p.x, p.y);
   return 0;
}
```

Feladat

- Foglaljunk le dinamikusan egy n méretű tömböt amely típusú elemeket tartalmaz! Töltsd fel az elemeket értékekkel, majd szabadítsd fel a tömböt!

Megoldás

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

struct pont {
   int x, y;
};

int main() {
   struct pont * tomb;
   int i;
   int n = 10;
   tomb = (struct pont*)malloc(
      sizeof(struct pont) * n);
```

Megoldás

```
for (i = 0; i < n; i++) {
     tomb[i].x = 10*i;
     (tomb + i)->y = 20*i;
   }
   for (i = 0; i < n; i++)
     printf("(%d; %d)\n", tomb[i].x,
tomb[i].y);
   free(tomb);
   tomb = 0;
   return 0;
}
```

Feladat

- Egészítsd ki az előző programot úgy, hogy a struktúra egy stringet is tartalmazzon. A string hosszát a felhasználó adja meg. Adj értéket a stringnek, és szabadítsd fel a program végén!

Megoldás

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

struct pont {
   int x, y;
   char * szoveg;
};

int main() {
   struct pont * tomb;
   int i;
   int n = 10;
   int hossz;
```

Megoldás

```
printf("Szoveg maximalis hossza: ");
scanf("%d", &hossz);
hossz++;
tomb = (struct pont*)malloc(
   sizeof(struct pont) * n);
for (i = 0; i < n; i++) {
   tomb[i].x = 10*i;
   (tomb + i)->y = 20*i;
   tomb[i].szoveg = (char*)malloc(
      sizeof(char) * hossz);
   printf("Szoveg: ");
   scanf("%s", tomb[i].szoveg);
}
```

Megoldás

```
    for (i = 0; i < n; i++)
      printf("%s: (%d; %d)\n", tomb[i].szoveg,
    tomb[i].x, tomb[i].y);

    for (i = 0; i < n; i++)
      free( tomb[i].szoveg );
    free(tomb);
    tomb = 0;
    return 0;
}
```

Feladat

- Írj programot, amely emberek nevét, fizetését, korát, magasságát és testsúlyát tárolja. Írj függvényt, amely bekéri hogy hány ember van, valamint hogy egy név legfeljebb milyen hosszú lehet, egy másik foglaljon le egy megfelelő méretű tömböt. Egy harmadik töltse fel az elemeket billentyűzetről. Írj függvényt, amely rendezi az elemeket az egyes adattagok értékei alapján, a függvény kapja meg paraméterként, hogy melyik legyen ez az adattag.

Megoldás

Megoldás

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

struct ember {
   char * neve;
   int kor, magassag;
   int testsuly, fizetes;
};

enum szempont { kor, magassag,
   suly, fizetes, nev };
```

Megoldás

```
int main() {
   int n, nevhossz;
   struct ember * emberek;
   n = beker(&nevhossz);
   lefoglal(&emberek, n, nevhossz);
   feltolt(emberek, n);
   kiir(emberek, n);
   rendez(emberek, n, kor);
   kiir(emberek, n);
   felszabadit(&emberek, n);
   return 0;
}
```

Megoldás

```
int beker(int * nevhossz) {
   int mennyi;
   printf("Emberek szama: ");
   scanf("%d", &mennyi);
   printf("Nev maximalis hossza: ");
   scanf("%d", nevhossz);
   return mennyi;
}
```

Megoldás

```
void lefoglal(struct ember ** tomb, int meret,
  int max) {
  int i;
  *tomb = (struct ember*)malloc(sizeof(
    struct ember) * meret);
  max++;
  for (i = 0; i < meret; i++)
    (*tomb)[i].neve = (char*)malloc(
      sizeof(char) * max);
}
```

Megoldás

```
void feltolt(struct ember * tomb, int meret) {
   int i;
   for (i = 0; i < meret; i++) {
      printf("%d. neve: ", i+1);
      scanf("%s", tomb[i].neve);
      printf("%d. kora: ", i+1);
      scanf("%d", &tomb[i].kor);
      printf("%d. magassaga: ", i+1);
      scanf("%d", &tomb[i].magassag );
      printf("%d. testsulya: ", i+1);
      scanf("%d", &tomb[i].testsuly );
      printf("%d. fizetese: ", i+1);
      scanf("%d", &tomb[i].fizetes );
   }
}
```

Megoldás

```
void kiir(struct ember * tomb, int meret) {
   int i;
   for (i = 0; i < meret; i++) {
      printf("%d. neve: %s\n", i+1, tomb[i].neve);
      printf("%d. kora: %d\n", i+1, tomb[i].kor);
      printf("%d. magassaga: %d\n", i+1,
         tomb[i].magassag);
      printf("%d. testsulya: %d\n", i+1,
         tomb[i].testsuly);
      printf("%d. fizetese: %d\n\n", i+1,
         tomb[i].fizetes);
   }
   printf("**********************\n");
}
```

Megoldás

```
void felszabadit(struct ember ** tomb,
   int meret) {
   int i;
   for (i = 0; i < meret; i++)
      free( (*tomb)[i].neve );
   free(*tomb);
   tomb = 0;
}
```

Megoldás

```
void rendez(struct ember * tomb, int n,
   enum szempont sz) {
   int mini;
   int i, j;
   struct ember temp;
   for(i = 0; i < n - 1; i++) {
      mini = i;
      for (j = i; j < n; j++) {
```

Megoldás

```
   if (sz == nev) {
      if (strcmp(tomb[mini].neve, tomb[j].neve) > 0)
         mini = j;
   } else {
      int elso = *(&tomb[mini].kor + sz);
      int masodik = *(&tomb[j].kor + sz);
      if ( elso >  masodik)
         mini = j;
      }
   }
   temp = tomb[mini];
   tomb[mini] = tomb[i];
   tomb[i] = temp;
   }
}
```