1. Feladat: Olvasson be változó számú neveket és érdemjegyeket egy struktúrába és írassa ki az eredményt.

```
#include<stdio.h>  // printf() és scanf() miatt kell
#include<stdlib.h> // malloc() miatt kell

struct Kurzus
{
    char nev[30];
    int jegy;
};

int main()
{
    struct Kurzus *ptr;
    int i, db;
    printf("Hany hallgato van? ");
    scanf("%d", &db);

    ptr = (struct Kurzus*) malloc (db * sizeof(struct Kurzus));

    for(i = 0; i < db; ++i)
    {
        printf("Kerem a nevet es a jegyet:\n");
        scanf("%s %d", (ptr + i)->nev, &ptr[i].jegy);
    }

    printf("Osszesen:\n");
    for(i = 0; i < db ; ++i)
    {
        printf("%s\t%d\n", ptr[i].nev, (ptr + i)->jegy);
    }
}
```

2. Feladat: Láncolt lista: készítsünk egy *int szam*-ot tartalmazó struktúrát, ami tartalmaz egy mutatót egy önmagával megegyező típusra *next* néven.

```
#include <stdio.h>
struct Elem
{
    int szam;
    struct Elem *next;
};

int main()
{
```

```
}
```

3. Feladat: Láncolt lista: az előzőből hozzunk éltre két darabot dinamikusan, az első a másodikra mutasson.

```
#include <stdio.h>
#include <stdlib.h>
struct Elem
{
    int szam;
    struct Elem *next;
};

int main()
{
    struct Elem* e1 = (struct Elem*) malloc(sizeof(struct Elem));
    struct Elem* e2 = (struct Elem*) malloc(sizeof(struct Elem));
    // osszekotes első variacio
    e1->next = &e2[0];
    // osszekotes 2. variacio
    e1[0].next = e2;
}
```

4. Feladat: Láncolt lista: Készítsünk egy függvényt, ami kiírja a lista összes elemét. Majd kérjünk be a felhasználótól tetszőleges számú dinamikusan létrehozott listaelemet.

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

typedef struct Elem  // azért lett typedef, hogy kényelmes függvényparaméter
lehessen
{
    int szam;
    struct Elem *next;  // a 'struct Elem' nem mas mint onmaga
} Elem;

void printList(Elem * head)
{
    Elem * current = head;

    while (current != NULL)
    {
        printf("%d\n", current->szam);
        current = current->next;
    }
```

```
}

int main()
{
    int db;
    printf("Hany elemet ad meg?");
    scanf("%d", &db);

    Elem *last = NULL;
    Elem *first = NULL;
    for(int i=0; i < db; i++)
    {
        Elem* e = (Elem*) malloc(sizeof(Elem));
        e->next = NULL;  // probaljuk ki úgy is, hogy ezt a sort kitoroljuk.
Magyarazzuk meg miert tortenik?
        if(last != NULL) // ha nem az elsőt töltjük fel
        {
            last->next = e;
        }
        else // ha az elsőt töltjük fel, az elsőt eltároljuk, mert e nélkül
elveszne az első eleme a listának
        {
            first = e;
        }
        printf("Kerem az %d.-t:", i);
        scanf("%d", &e->szam);
        last = e; // osszekotjuk a ket strukturat
    }
    printList(first);
}
```

5. Feladat: Értsük meg jobban a typedef használatát a _types.h kódrészlete alapján:

```
/* sys/x86/include/_types.h in FreeBSD */
/* this is machine dependent */
#ifdef  __LP64__
typedef unsigned long       __uint64_t;
#else
__extension__
typedef unsigned long long  __uint64_t;
#endif
...
...
typedef __uint64_t  __size_t;
```

A fenti kódpélda az ***unsigned long long*** ot rövidíti ***__size_t***-nek (64 bites rendszereken). A következő kódrészlet 3 féle struktúradefiníciót alkalmaz:

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

/* uj tipust hozunk létre Item neven.
   A struct elem_t helyett Item-et irhatunk kesobb
*/
typedef struct elem_t
{
    char *text;
    int count;
    struct elem_t *next; /* itt meg nem lehet roviditeni */
} Item;

/* normal struktura definicio */
struct item
{
    char *text;
    int count;
    struct item *next;
};

/*
* ez valojaban egy nevtelen struktura
*/
struct
{
    int i;
    char c;
} nevtelen;

int main(void)
{
    // mutato a "struct item"-re
    struct item *pi = (struct item *)malloc(sizeof(struct item));;
    pi->count = 10;

    // Roviditett hasznalata a "struct item_t"-nek
    Item *iI = (Item *)malloc(sizeof(Item));
    iI->count = 5;
    iI->next = iI; // onmagara mutat (csak a pelda kedveert)

    // nevtelen structura
    nevtelen.i = 9;
    nevtelen.c = 'x';
}
```

6. Feladat: A 4. példához készítsünk egy ***Elem* pop()*** függvényt, ami az első elemet eltávolítja a listából

```
Elem* pop(Elem *first)
{
    Elem *ret = first->next;
    free(first);
    return ret;
}
```

7. Feladat: A 4. példához készítsünk egy ***void sort(Elem*)***, ami sorbarendezi növekvő sorrendbe a lista elemeit, az értékek módosításával.

```
void sort(Elem *first)
{
    Elem *e1, *e2;
    e1 = first;
    while(e1->next != NULL)
    {
        e2 = e1->next;
        while(e2 != NULL)
        {
            if(e2->szam < e1->szam)
            {
                int tmp = e2->szam;
                e2->szam = e1->szam;
                e1->szam = tmp;
            }
            e2 = e2->next;
        }
        e1 = e1->next;
    }
}
```