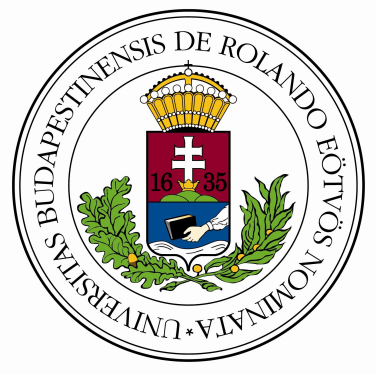
# Imperatív programozás

## Típusok




**Kozsik Tamás és mások**

ELTE Eötvös Loránd Tudományegyetem

# Outline

# Konstansok C-ben

## Konstansok

```
const int i = 3;
int const j = 3;

const int t[] = {1,2,3};


const int *p = &i;

int v = 3;
int * const q = &v;
```

## Fordítási hiba

```
i = 4;
j = 4;

t[2] = 4;
t = {1,2,4};

*p = 4;


q = (int *)malloc(sizeof(int));
```

# Nem teljes a biztonság

```
const int i = 3;
const int *r = &i;      /* korrekt */
```

# Nem teljes a biztonság

```
const int i = 3;
const int *r = &i;      /* korrekt */

int j = 2;
r = &j;                 /* korrekt */
```

# Nem teljes a biztonság

```
const int i = 3;
const int *r = &i;      /* korrekt */

int j = 2;
r = &j;                 /* korrekt */

r = &i;
int * p = r;            /* csak warning */
int * const q = &i;     /* csak warning */

*p = *q = 4;            /* i megváltozik :-( */
```

# Karakter első előfordulása szövegben

```
char *strfind( char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}
```

```
char p[] = "Hello";
char *q = strfind(p,'e');
*q = 'o';                          /* ok: p-ben "Hollo", q-ban "ollo" */
```

# Karakter első előfordulása szövegben

```
char *strfind( char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}
```

```
char p[] = "Hello";
char *q = strfind(p,'e');
*q = 'o';                          /* ok: p-ben "Hollo", q-ban "ollo" */

char *r = strfind("Hello",'e');    /* módosíthatatlan tárhelyen! */
*r = 'o';                          /* Segmentation fault :-( */
```

# Szövegliterálok módosíthatatlan tárhelyen

```
char *r = "Hello";
*r = 'h';                       /* segmentation fault :-( */
```

# Szövegliterálok módosíthatatlan tárhelyen

```
char *r = "Hello";
*r = 'h';                       /* segmentation fault :-( */

const char *cr = "Hello";
*cr = 'h';                      /* fordítási hiba :-) */
```

# Zárjuk ki az illegális memóriamódosítást!

```
const char *strfind( char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}

const char *r = strfind("Hello",'e');
*r = 'o';                          /* fordítási hiba :-) */
```

# Zárjuk ki az illegális memóriamódosítást!

```
const char *strfind( char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}

const char *r = strfind("Hello",'e');
*r = 'o';                       /* fordítási hiba :-) */

char p[] = "Hello";
char *q = strfind(p,'e');    /* fordítási hiba :-( */
*q = 'o';
```

# Zárjuk ki az illegális memóriamódosítást!

```
const char *strfind( char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}

const char *r = strfind("Hello",'e');
*r = 'o';                       /* fordítási hiba :-) */

char p[] = "Hello";
char *q = strfind(p,'e');       /* fordítási hiba :-( */
*q = 'o';

const char *r = "Hello";
*r = strfind(r,'e');            /* fordítási hiba :-( */
```

# Működjön const-ra?

```
const char *strfind( const char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}

const char *r = "Hello";
r = strfind(r,'e');
*r = 'o';                    /* fordítási hiba :-) */
```

# Működjön const-ra?

```
const char *strfind( const char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}

const char *r = "Hello";
r = strfind(r,'e');
*r = 'o';                    /* fordítási hiba :-) */

char p[] = "Hello";
char *q = strfind(p,'e');
*q = 'o';                    /* fordítási hiba :-( */
```

# const-ra polimorf megoldás nincs

```
char *strfind( char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}

const char *conststrfind( const char *str, int c ){
    int i = 0;
    while( str[i] != 0 && str[i] != c ) ++i;
    return &str[i];
}
```

- Kód duplikációja?
- Karakterliterállal az első még mindig meghívható

# Ugyanez a `string` szabványos könyvtárban

```
char *strchr( const char *str, int c ){
    while( *str != 0 && *str != c ) ++str;
    return str;
}
```

# Ugyanez a `string` szabványos könyvtárban

```
char *strchr( const char *str, int c ){
    while( *str != 0 && *str != c ) ++str;
    return str;
}
```

- Általánosan meghívható
- Karakterliterállal meghívható

# Ugyanez a string szabványos könyvtárban

```
char *strchr( const char *str, int c ){ ... }

const char *p = "Hello";
char *r = "Hello";
char q[] = "Hello";

p = strchr(p,'e');
r = strchr(q,'e');

*p = 'o';              /* fordítási hiba :-) */
*r = 'o';              /* ok :-) */

r = strchr(p,'o');     /* vagy strchr("Hello",'e') */
*r = 'e';              /* segmentation fault :-( */
```

# Outline

# typedef a C-ben

```
typedef double Element;
```

# typedef a C-ben

```
typedef double Element;

Element max( Element array[], int size ){
    Element m = array[0];
    while( size > 0 ){
        --size;
        if( array[size] > m )
            m = array[size];
    }
    return m;
}
```

# typedef a C-ben

```
typedef double Element;

Element max( Element array[], int size ){
    Element m = array[0];
    while( size > 0 ){
        --size;
        if( array[size] > m )
            m = array[size];
    }
    return m;
}
```

- Mint a Haskellben a `type`
- Típusszinoníma: ugyanannak a típusnak több neve is van
- Esztétikai szerepe van
- Karbantarthatóság (olvashatóság, módosíthatóság)

# Outline

# Típusok, típuskonstrukciók

- Egyszerű típusok
  - Szám típusok
    - Egész típusok (és karakter típusok)
    - Lebegőpontos típusok
  - Felsorolási típusok
  - Mutató típusok
- Összetett típusok
  - Tömb
  - Lista (Python, Haskell)
  - Rendezett n-es (Python, Haskell)
  - Halmaz (Python `set` és `frozenset`)
  - Leképezés (Python `dictionary`)
  - Rekord (C `struct`, Haskell record)
  - Unió típus (C `union`, Haskell algebrai adattípus)
  - Osztály

# Felsorolási típus

## Haskell: algebrai adattípussal

```
data Color = White | Green | Yellow | Red | Black
```

## C: enum

```
enum color { WHITE, GREEN, YELLOW, RED, BLACK };
```

# Felsorolási típus

## Haskell: algebrai adattípussal

```haskell
data Color = White | Green | Yellow | Red | Black
```

## C: enum

```c
enum color { WHITE, GREEN, YELLOW, RED, BLACK };

const char* property( enum color code ){
    switch( code ){
        case WHITE:  return "pure";
        case GREEN:  return "jealous";
        case YELLOW: return "envy";
        case RED:    return "angry";
        case BLACK:  return "sad";
        default:     return "?";
    }
}
```

# enum és typedef

```
enum color { WHITE, GREEN, YELLOW, RED, BLACK };

typedef enum color Color;

const char* property( Color code ){
    switch( code ){
        case WHITE:  return "pure";
        case GREEN:  return "jealous";
        case YELLOW: return "envy";
        case RED:    return "angry";
        case BLACK:  return "sad";
        default:     return "?";
    }
}
```

# enum: valójában egy egész szám típusra képződik le

```
enum color { WHITE, GREEN, YELLOW, RED, BLACK };

const char * const properties[] =
    { "pure","jealous","envy","angry","sad" };

const char* property( enum color code ){
    return properties[code];
}
```

# Trükközés a típusértékekkel

```
enum color { WHITE = 1, GREEN, YELLOW, RED = 6, BLACK };

typedef enum color Color;
const char* property( Color code ){ ... }
```

# Trükközés a típusértékekkel

```
enum color { WHITE = 1, GREEN, YELLOW, RED = 6, BLACK };

typedef enum color Color;
const char* property( Color code ){ ... }

int main( int argc, char *argv[] )
{
    for( --argc; argc>0; --argc )
    {
       printf("%s\n", property( atoi(argv[argc]) ));
    }
    return 0;
}
```

# Direktszorzat típusok

(Potenciálisan) különböző típusú elemekből konstruált összetett típus

- tuple
- rekord, struct

## C struct

```c
struct month { char *name; int days; };  /* típus létrehozása */

struct month jan = {"January", 31};      /* változó létrehozása */

/* three-way comparison */
int compare_days_of_month( struct month left, struct month right )
{
    return left.days - right.days;
}
```

# C struct

```c
struct month { char *name; int days; };
struct month jan = {"January", 31};

struct date { int year; struct month *month; char day; };
struct person { char *name; struct date birthdate; };

typedef struct person Person;

int main(){
   Person pete = {"Pete", {1970,&jan,28}};
   printf("%d\n", pete.birthdate.month->days);
   return 0;
}
```

# Paraméterátadás

```
void one_day_forward( struct date *d ){
    if( d->day < d->month->days ) ++(d->day);
    else { ... }
}

struct date next_day( struct date d ){
    one_day_forward(&d);
    return d;
}

int main(){
   struct date new_year = {2019, &jan, 1};
   struct date sober;
   sober = next_day(new_year);
   return ( sober.day != 2 );
}
```

# Unió típus

Típusértékei több típus valamelyikéből

## C: union

```c
struct       month { char *name; int days; }; /* name and days */
union name_or_days { char *name; int days; }; /* either of them */

union name_or_days brrr = {"Pete"};  /* now it contains a name */
printf("%s\n", brrr.name);   /* fine */
printf("%d\n", brrr.days);   /* prints rubbish */
brrr.days = 42;              /* now it contains an int */
printf("%d\n", brrr.days);   /* fine */
printf("%s\n", brrr.name);   /* probably segmentation fault */
```

# Tömb fogalma

Azonos típusú (méretű) objektumok egymás után a memóriában.

- Bármelyik 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)`

# 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 */

extern int e[];
extern int f[10];               /* méret ignorálva */

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

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 */
```

# Tömbök indexelése

## C

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

## Python

- `t = [1,2,3,4]`
- 0-tól indexelünk
- hossz futás közben ismert
- negatív index is értelmezett
  - utolsó előtti elem: `t[-2]`
- hibás index: futási hiba

# C mutatók

- Más változókra mutat(hat): indirekció
  - dinamikus
  - automatikus vagy statikus
- Típusbiztos

```
int i;
int t[4];
int *p = NULL;  /* sehova sem mutat */

/* dinamikus tárolású változóra mutat */
p = (int*)malloc( sizeof(int) * i ); ... free(p);

/* statikusra vagy automatikusra mutat */
p = &i;    p = t;

*p = 5;          /* dereferálás */
```

# C deklarációk mutatókkal

```
int i = 42;
int *p = &i;
int **pp = &p;          /* mutató mutatóra */
int *ps[10];            /* mutatók tömbje */
int (*pt)[10];          /* mutató tömbre */

char *str = "Hello!";

void *foo = str;        /* akármire mutathat */

int* p,q;               /* mutató és int */
int  s,t[5];            /* int és tömb */
int *f(void);           /* int* eredményű függvény */
int (*f)(void);         /* mutató int eredményű függvényre */
```

# Tömbök és mutatók kapcsolata

- Tömb: *second-class citizen*
- Tömb $\rightarrow$ mutató
- Nem ekvivalensek!

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

int *p = t;
int *q = &t[0];

int (*r)[3] = &t;

printf( "%d%d%d%d\n", t[0], *p, *q, (*r)[0] );
```

# Tömb átadása paraméterként?

Valójában mutató típusú a paraméter!

```
double distance( double a[], double *b )
{
   double dx = a[0] - b[0],
          dy = a[1] - b[1],
          dz = a[2] - b[2];
   return sqrt(dx*dx + dy*dy + dz*dz);
}
```

# Mutató-aritmetika – léptetések

```
int v[] = {6, 2, 8, 7, 3};
int *p = v;
int *q = v + 3;    /* v konvertálódik */
++p;
*p = 5;            /* v: 6, 5, 8, 7, 3 */
p += 2;
*q = 1;            /* v: 6, 5, 8, 1, 3 */
q -= 2;
*q = 2;            /* v: 6, 2, 8, 1, 3 */
```

# 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';

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

# Mutató-aritmetika: példa

```
int strlen( char* s )
{
    char* p = s;
    while( *p != '\0' )
    {
        ++p;
    }
    return p - s;
}
```

# Mutatók és tömbök közötti különbségek

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

v[ 1 ] = 5;
p[ 1 ] = 8;

int w[] = {1,2,3};
p = w;  /* ok */
v = w;  /* fordítási hiba */

printf( "%d %d\n", sizeof( v ), sizeof( p ) );
```

Lásd még az utolsó példát itt:
http://gsd.web.elte.hu/lectures/imper/imper-lecture-5/.

# Outline

# Tömbök átadása paraméterként: általánosítás?

```
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 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 paraméterként: futási időben rögzített méret?

```
double distance( double a[], double b[] ){
    double sum = 0.0;
    unsigned int i;
    for( i=0; i<???; ++i ){    /* ez itt nem Python */
        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 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 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;
}
```

# Bonyolult struktúra átadása paraméterként

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

- argc: pozitív szám
- argv[0]: program neve
- argv[i]: parancssori argumentum ($1 \leq i < \text{argc}$)
  - karaktertömb, végén NUL (`'\0'`)
- argv[argc]: NULL

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

# 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");
   }
}
```

# Túl merev megoldás

```
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);
```

# Sorfolytonos ábrázolás: egybefüggő memóriaterület

```
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 ) */
```

# Alternatív reprezentáció: 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);
```

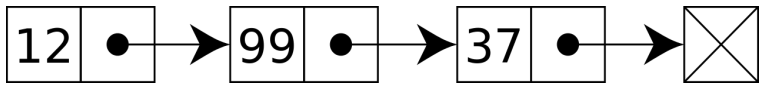
# Outline

# Adatszerkezetek

- „Sok” adat szervezése
- Hatékony elérés, manipulálás
- Alapvető módszerek
  - Tömb alapú ábrázolás (indexelés)
  - Láncolt adatszerkezet
  - Hasítás

# Láncolt adatszerkezetek

- Sorozat: láncolt lista
- Fa, pl. keresőfák
- Gráf




# Sorozatok ábrázolása

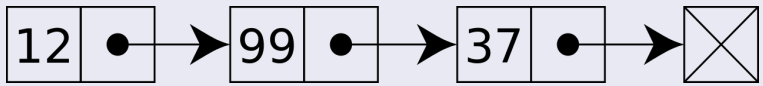
## Tömb

- Akárhányadik elem előkeresése, felülírása
- Beszúrás/törlés?
  - Adatmozgatás
  - Újraallokálás

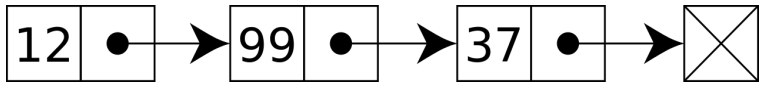
(egy példa: http://gsd.web.elte.hu/lectures/imper/imper-lecture-10/)

## Láncolt lista

- Elemek előkeresése és felülírása bejárással
- Beszúrás/törlés bejárás során
- Akárhányadik elem előkeresése, felülírása?



# Láncolt lista

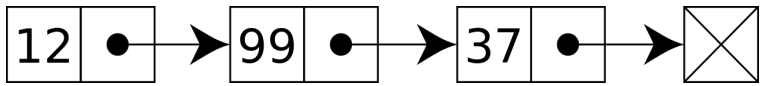



```c
struct node
{
    int data;
    struct node *next;
};
```

```haskell
nums :: [Int]
nums = [12, 99, 37]
```

# Láncolt lista felépítése

```
struct node
{
    int data;
    struct node *next;
};

struct node *head;
head = (struct node *)malloc(sizeof(struct node));
head->data = 12;
head->next = (struct node *)malloc(sizeof(struct node));
head->next->data = 99;
head->next->next = (struct node *)malloc(sizeof(struct node));
head->next->next->data = 37;
head->next->next->next = NULL;
```



# Beszúrás rendezett sorozatba (Haskell)

```
insert (x:xs) y
    | x < y     = x:(insert xs y)
    | otherwise = y:x:xs

insert [] y = [y]
```

# Beszúrás rendezett sorozatba (Python)

```
def insert(xs,y):
    if xs:
        z, *zs = xs
        if z < y:
            return [z] + insert(zs,y)
        else:
            return [y] + xs
    else:
        return [y]
```

# Beszúrás rendezett sorozatba (Python, ciklussal)

```
def insert(xs,y):
    def pos():
        i = 0
        for v in xs:
            if v > y:
                return i
            else:
                i += 1
        return i
    xs.insert(pos(),y)
```

# Beszúrás rendezett sorozatba (C)

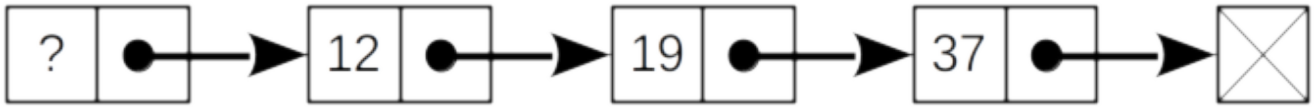
```c
struct node
{
    int data;
    struct node *next;
};

typedef struct node * list_t;

list_t insert( list_t list, int value );
```

# Beszúrás rendezett listába (C)

```
list_t insert( list_t list, int value ){
    list_t new = (list_t) malloc(sizeof(struct node));
    if( NULL == new ) return NULL;
    new->data = value;
    if( NULL == list || value < list->data ){
        new->next = list;
        return new;
    } else {
        list_t current = list, next = current->next;
        while( next != NULL && next->data < value ){
            current = next;
            next = current->next;
        }
        new->next = next;
        current->next = new;
        return list;
    }
}
```

# Fejelemes lista

- Plusz egy node a lista legelején
- A benne tárolt értéket nem használjuk
- Ne kelljen az üres lista esettel külön foglalkozni




# Beszúrás rendezett (fejelemes) listába

```
void insert( list_t list, struct node *new_node ){
    list_t current = list, next = current->next;
    while( next != NULL && next->data < new_node->data ){
        current = next;
        next = current->next;
    }
    new_node->next = next;
    current->next = new_node;
}
```

# Outline

# Egyenlőségvizsgálat és másolás elemi típusokon

```
int a = 5;
int b = 7;

if( a != b )
{
    a = b;
}
```

# Mutatókkal?

```
int n = 4;
int *a = (int*)malloc(sizeof(int));
int *b = &n;

if( a != b )
{
    a = b;
}
```

# Tömbökkel?

```
int a[] = {5,2};
int b[] = {5,2};

if( a != b )
{
    a = b;              /* fordítási hiba */
}
```

# Tömbökkel!

```
#define SIZE 3

int is_equal( int a[], int b[] ){
    for( int i=0; i<SIZE; ++i )
        if( a[i] != b[i] ) return 0;
    return 1;
}

void copy( int a[], int b[] ){
    for( int i=0; i<SIZE; ++i ) a[i] = b[i];
}

int a[SIZE] = {5,2}, b[SIZE] = {7,3,0};

...
if( ! is_equal(a,b) ) copy( a, b );
```

# Struktúrákkal?

```
struct pair { int x, y; };

struct pair a, b;
a.x = a.y = 1;
b.x = b.y = 2;

if( a != b )          /* fordítási hiba */
{
    a = b;
}
```

# Struktúrákkal!

```
struct pair { int x, y; };

int is_equal( struct pair a, struct pair b )
{
    return (a.x == b.x) && (a.y == b.y);
}

struct pair a, b;
a.x = a.y = 1;
b.x = b.y = 2;

if( is_equal(a,b) )
{
    a = b;
}
```

# Láncolt lista?

```
struct node
{
    int data;
    struct node *next;
};

struct node *a, *b;
...

if( a != b )
{
    a = b;
}
```

# Sekély megoldás – nem jó ide

```
struct node
{
    int data;
    struct node *next;
};

int is_equal( struct node *a, struct node *b )
{
    return (a->data == b->data) && (a->next == b->next);
}

void copy( struct node *a, const struct node *b )
{
    *a = *b;
}
```

# Mély egyenlőségvizsgálat

```
struct node
{
    int data;
    struct node *next;
};

int is_equal( struct node *a, struct node *b )
{
    if( a == b ) return 1;
    if( (NULL == a) || (NULL == b) ) return 0;
    if( a->data != b->data ) return 0;
    return is_equal(a->next, b->next);
}
```

# Mély másolás

```
struct node {
    int data;
    struct node *next;
};

struct node *copy( const struct node *b ){
    if( NULL == b ) return NULL;
    struct node *a = (struct node*)malloc(sizeof(struct node));
    if( NULL != a ){
        a->data = b->data;
        a->next = copy(b->next);
    } /* else hibajelzés! */
    return a;
}
```

# Outline

# Magasabbrendű függvények

## Python

```
def foreach(fun,list):
    for item in list: fun(item)

foreach( print,                  [1,2,3,4,5] )
foreach( lambda v: print(v,v), [1,2,3,4,5] )
```

## C: függvénymutatók segítségével

```
/* mutató int eredményű, paraméter nélküli függvényre */
int (*fp)(void);

/* int->int függvényt és int-et váró függvény int eredménnyel */
int twice( int (*f)(int), int n );
```

# C: függvénymutatók

```
int twice( int (*f)(int), int n )
{
    n = (*f)(n);
    n = f(n);
    return n;
}

int inc( int n ){ return n+1; }


printf(  "%d\n",  twice( &inc, 5 )  );
```

# C: függvénymutatók - néhány észrevétel

```
int inc( int n ){ return n+1; }

int (*f)(int) = &inc;
f = inc;
f(3) + (*f)(3);

int (*g)() = inc;
g(3,4);  g();
```

# Outline

# Címkézett unió

```
enum shapes { CIRCLE, SQUARE, RECTANGLE };
struct    circle { double radius; };
struct    square { int side; };
struct rectangle { int a; int b; };

struct shape
{
    int x, y;
    enum shapes tag;
    union csr
    {
        struct    circle c;
        struct    square s;
        struct rectangle r;
    } variant;
};
```

# Egységes használat

```
struct shape
{
    int x, y;
    enum shapes tag;
    union csr
    {
        struct    circle c;
        struct    square s;
        struct rectangle r;
    } variant;
};

void move( struct shape *aShape, int dx, int dy ){
    aShape->x += dx;
    aShape->y += dy;
}
```

# Használat esetszétválasztással

```
struct shape {
    int x, y;
    enum shapes tag;
    union csr {
        struct    circle c;
        struct    square s;
        struct rectangle r;
    } variant;
};

double leftmost( struct shape aShape ){
   switch( aShape.tag ){
        case CIRCLE: return aShape.x - aShape.variant.c.radius;
        default:     return aShape.x;
   }
}
```

# Biztonságos létrehozás

```
struct shape {
    int x, y;
    enum shapes tag;
    union csr {
        struct    circle c;
        struct    square s;
        struct rectangle r;
    } variant;
};

struct shape make_circle( int cx, int cy, double radius ){
    struct shape c;
    c.x = cx; c.y = cy; c.tag = CIRCLE;
    c.variant.c.radius = radius;
    return c;
}
```

# A switch-nél rugalmasabb, bővíthetőbb megoldás?

```
double leftmost_default( struct shape *aShape ){
        return aShape->x;
}

double leftmost_in_circle( struct shape *aCircle ){
        return aCircle->x - aCircle->variant.c.radius;
}


...
```

# Műveletek és adatok egy egységbe zárása

```
struct shape {
    int x, y;
    enum shapes tag;
    union csr {
        struct    circle c;
        struct    square s;
        struct rectangle r;
    } variant;
    double (*leftmost)( struct shape *this );  /* függvénymutató */
};

double leftmost_default( struct shape *aShape ){ ... }
double leftmost_in_circle( struct shape *aCircle ){ ... }

double leftmost( struct shape aShape ){
    return ashape.leftmost( &aShape );
}
```

# Rugalmasan beállítható implementáció

```
struct shape {
    int x, y;
    ...
    double (*leftmost)( struct shape *this );
};

double leftmost_default( struct shape *aShape ){ ... }
double leftmost_in_circle( struct shape *aCircle ){ ... }

struct shape make_circle( int cx, int cy, double radius ){
    struct shape c;
    c.x = cx; c.y = cy; c.tag = CIRCLE;
    c.variant.c.radius = radius;
    c.leftmost = leftmost_in_circle;
    return c;
}
```

# Osztály

- Objektum-orientált nyelvek
- Osztály: rekordszerű struktúra
  - Adattagok (mezők)
  - Műveletek (metódusok)
- Öröklődés: címkézett unió

## Python osztály: rekord típus megvalósítására

```
class Month: pass

jan = Month()
jan.name, jan.days = 'January', 31
print(jan.name, '(with', jan.days, 'days)')
```

# Biztonságos inicializáció

```
class Month:
    def __init__(self,name,days):
        self.name, self.days = name, days

jan = Month( 'January', 31 )
print(jan.name, '(with', jan.days, 'days)')
feb = Month()  # hibás!
```

# Műveletek

```
class Month:
    def __init__(self,name,days):
        self.name, self.days = name, days
    def isValid(self,day):
        return 0 < day <= self.days
    def __str__(self):
        return self.name + ' (with ' + str(self.days) + ' days)';

jan = Month( 'January', 31 )
print( jan.isValid(29) )
print(jan)
```

# C++ osztály

```cpp
#include <string>

struct Month
{
    std::string name;
    int days;
    bool isValid( int day ) const
    {
        return 0 < day && day <= days;
    }
};
```

# C++ objektum

```
#include <iostream>
#include <string>

struct Month {
    std::string name;
    int days;
    bool isValid(int day) const { return 0 < day && day <= days; }
};

int main() {
    Month jan;
    jan.name = "January";
    jan.days = 31;
    std::cout << jan.name << " (with " << jan.days << " days)"
              << std::endl;
}
```

# C++ konstruktor

```
struct Month {
    std::string name;
    int days;
    bool isValid(int day) const { return 0 < day && day <= days; }
    Month(std::string name, int days) {
        this->name = name;
        this->days = days;
    }
};

int main() {
    Month jan("January",31);
    std::cout << jan.name << " (with " << jan.days << " days)"
              << std::endl;
}
```

# Információelrejtés

```
class Month
{
public:
    bool isValid(int day) const { return 0 < day && day <= days; }
    Month(std::string name, int days)
    {
        this->name = name;
        this->days = days;
    }
    std::string getname() const { return this->name; }
    int getdays() const { return this->days; }
private:
    std::string name;
    int days;
};
```

# Külön fordítás

## Month.h

```
#ifndef MONTH_H
#define MONTH_H
#include <string>
class Month
{
public:
    bool isValid( int day ) const;
    Month(std::string name, int days);
    std::string getname() const;
    int getdays() const;
private:
    std::string name;
    int days;
};
#endif
```

# Külön fordítás

## Month.cpp

```
#include "Month.h"

bool Month::isValid(int day) const
{
    return 0 < day && day <= days;
}

Month::Month(std::string name, int days)
{
    this->name = name;
    this->days = days;
}

std::string Month::getname() const { return this->name; }
int Month::getdays() const { return this->days; }
```

# Külön fordítás

## main.cpp

```
#include "Month.h"
#include <iostream>

int main()
{
    Month jan("January",31);
    std::cout << jan.getname()
              << " (with "
              << jan.getdays()
              << " days)"
              << std::endl;
}
```

# Névterek

```
#ifndef MONTH_H
#define MONTH_H
#include <string>
namespace ktolib {
    class Month {
        public:
            bool isValid( int day ) const;
            Month(std::string name, int days);
            std::string getname() const;
            int getdays() const;
        private:
            std::string name;
            int days;
    };
}
#endif
```