egesz_valos.c

```
#include <stdio.h>
int main(){
  float a, b, c, d, e, f; double g;
  a = 1*2+4*5/3;          a = 8.000000
  b = 1*2.0+4*5/3;        b = 8.000000
  c = 1*2+4*5.0/3;        c = 8.666667
  d = 1*2+(int)4*5.0/3;   d = 8.666667
```


egesz_valos.c

```
e = 1*2+(int)(4*5.0/3);   e = 8.000000
f = 1*2+4*(int)5.0/3;     f = 8.000000
int ia, ib, ic, id;
ia = 1*2+4*5/3;           ia = 8
ib = 1*2.0+4*5/3;         ib = 8
ic = 1*2+4.0*5/3;         ic = 8
id = d;                   id = 8
```


egesz_valos.c

```
/* Kerekítés és csonkítás */
a = -2.0; ia=a; a = a+1.0/3.0;
ia=a;          a = -1.666667, ia = -1
a = a+1.0/3.0;
ia=a;          a = -1.333333, ia = -1
a = a+1.0/3.0;
ia=a;          a = -1.000000, ia = 0
```


egesz_valos.c

```
printf("ia=%d\n", ia);   ia=0
printf("ia=%f\n", ia);   ia=-1.000000
g = 3.4E38; printf("g=%f\n", g);
g=340000000000000000000000000000000000000.000000
printf("ia=%f\n", ia);
ia=3399998735199788000000000000000000000.000000
printf("\n"); system("PAUSE"); return 0; }
```


unsigned.c

```
#include <stdio.h>
int main(){  unsigned i; i = -1;
  printf("i=%u\n",i);   i=4294967295
```

- A signed -1 kódja:

11111111 11111111 11111111 11111111

- ez unsiged-ként értelmezve:

2^32 -1 = 4294967295

unsigned.c

```
unsigned j; int k, m, p;
 j = 3; k = 2; m = 3; p = -1;
 if(j>k){ printf("j>k\n");}
 else { printf("j<=k\n");}          j>k
 if(j>m){ printf("j>m\n");}
 else if(j==m){ printf("j==m\n");}
 else { printf("j<m\n");}           j==m
```


unsigned.c

```
  if(j>p){ printf("j>p\n");}        j=3; k=2;
  else { printf("j<=p\n");}         p = -1;
  if(j>-1){ printf("j>-1\n");}      j<=p
  else { printf("j<=-1\n");}        j<=-1
  if(k>p){ printf("k>p\n");}        k>p
  else { printf("k<=p\n");}
printf("\n"); system("PAUSE"); return 0; }
```

I. Valós és egész számok
II. Mutatók
III. Karakterek és ASCII kódok
IV. Függvények és eljárások
V. Tárolási osztályok
VI. Rekurzió.

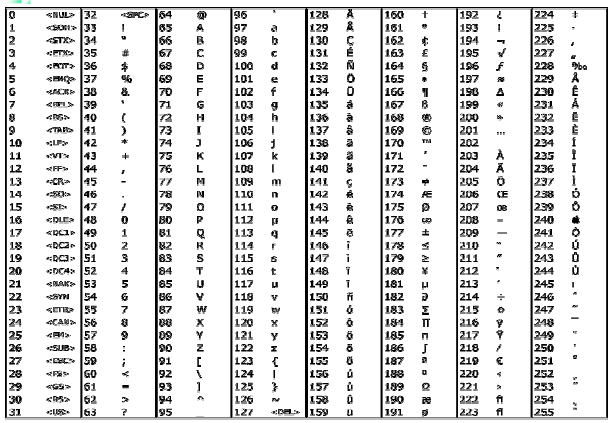
mutatok.c

```
#include <stdio.h>
int main(){ int a, b, *p;
  p = &a; *p = 2; a = 3; *p *= 4;
  b = *p+1; p = &b; *p += 4;
  printf("a=%d, b=%d\n",a,b);
```

a=12, b=17

```
printf("\n"); system("PAUSE"); return 0; }
```


I. Valós és egész számok
II. Mutatók
III. Karakterek és ASCII kódok
IV. Függvények és eljárások
V. Tárolási osztályok
VI. Rekurzió.



karakter_io.c

```
#include <stdio.h>
int main(){
  char c; int d;
  printf("c=");  scanf("%c",&c);
  printf("c1=%c, c2=%d \n", c, c);
```

c=B
c1=B, c2=66

Miért jó? Azért, mert amikor printf-nek átadjuk, akkor automatikus típuskonverzióval:

- char-ból és short-ból int vagy unsigned int lesz
- float-ból duoble lesz.

karakter_io.c

```
c = 'b';
printf("sizeof(c)=%d\n", sizeof(c));
printf("sizeof('b')=%d\n",
                        sizeof('b'));
```

sizeof(c)=1
sizeof('b')=4

karakter_io.c

```
  printf("d=");  scanf("%d",&d);
  printf("d1=%c, d2=%d\n", d, d);
  printf("\n"); system("PAUSE");
}
```

d=77
d1=M, d2=77

- a %c eleve int-et vár,

amit karakterként értelmez.

I. Valós és egész számok
II. Mutatók
III. Karakterek és ASCII kódok
IV. Függvények és eljárások
V. Tárolási osztályok
VI. Rekurzió.

fgv_eljaras.c

```
#include <stdio.h>
int f(int x, int *y){
  x=11; *y = 12; return 13;
}
void g(int x, int *y){
  x=21; *y = 22; /* nincs return */
}
```


fgv_eljaras.c

```
int main(){ int a =2,b,c;
 c = 1+2*f(a,&b);
 g(b,&a);
 printf("\n"); system("PAUSE"); return 0; }
```

a=2, b=12, c = 27

a=22, b=12, c = 27

- g(b,&a); nem helyettesítődik semmivel, ezért nem lehet sem 1+2*g(a,&b) sem c = g(b,&a);

Globális:

- függvényeken kívül van deklarálva
- bármelyik függvényben elérhető, amelyikben nem definiáltuk felül
- a teljes program futása alatt megőrzi a memóriafoglalását

Lokális:

- függvényen (blokkon) belül van deklarálva
- csak abban a függvényben (blokkban) elérhető, amelyikben definiáltuk

Lokális:

- a függvény megívásakor foglalódik számára memória
- a függvényből való visszatéréskor a számára foglalt memória felszabadítódik

globalis.c

```
#include <stdio.h>
int a, b;
void f(void){ int b, c; a = 2; b = 3; }
void g(void){ a += 5; b = 3; }
int main(){
  f(); printf("a=%d, b=%d\n",a,b);
```

a=2, b=0

globalis.c

```
/* c = 5; hiba, mert nem lathato */
  a = 3; g();
  printf("a=%d, b=%d\n",a,b);
```

a=8, b=3

```
  printf("\n"); system("PAUSE"); return 0;
}
```


Statikus:

- csak abban a függvényben elérhető, amelyikben definiáltuk
- memóriafoglalása a program teljes futása alatt megmarad
- a függvény ismételt meghívásakor a régi értékét visszakapjuk

I. Valós és egész számok
II. Mutatók
III. Karakterek és ASCII kódok
IV. Függvények és eljárások
V. Tárolási osztályok
VI. Rekurzió.


```
double fakt(int n){
  int i; double f = 1.0;
  if (n<0){ printf("Negativ szam!\n");
    return 0;
  } else {
     for (i=1; i<=n; i++) f *= i; return f;
  }                                    }
```



```
double faktr(int n){
  if (n<0) { printf("Negativ szam!\n");
    return 0;
  } else if (n==0) return 1;
    else if (n==1) return 1;
    else return faktr(n-1)*n;
}
```



```
int main(){
  int a;
  double faktorialis;
  printf("Hany faktorialist
  szamoljak? ");
  scanf("%d", &a);
  faktorialis = fakt(a);
```



```
  if ((int) faktorialis)
    printf("%d! = %d\n",
                    a, (int) faktorialis);
  faktorialis = faktr(a);
  if ((int) faktorialis)
    printf("%d! rekurzivan = %d\n",
                    a, (int) faktorialis);
printf("\n"); system("PAUSE"); return 0; }
```