Néhány STL függvény

v0.000021

2010. november 2.

Tartalomjegyzék

1. Vector

```
#include <vector>
```

A `vector` könyvtár definiálja a `vector` típust. A klasszikus C tömbökhöz képest a vektor előnye, hogy könnyű és kényelmes dinamikusan átméretezni.

1.1. Deklaráció és kezdőérték

Vektor típusú változót így deklarálunk:

```
vector<T> x;
```

ahol `T` bármilyen típus lehet, pl. `int`, `string`, `double`, stb. Az általunk definiált típusokat is rendezhetjük vektorba.

Vektort létrehozhatunk úgy is, hogy megadjuk, hány elemből álljon:

```
vector<T> x(10);
```

A fenti kód létrehoz egy 10 darab, T típusú elemből álló vektort.

A második paraméterben megadhatjuk az elemek kezdőértékét, pl:

```
vector<int> x(10,56);
```

A fenti kód létrehoz egy 10 darab, integer típusú elemből álló vektort, melynek minden elemének értéke '56'. Ügyeljünk rá, hogy a kezdőérték típusa azonos legyen a vektor típusával.

Egy vektor tartalmát könnyen átmásolhatjuk egy másik vektorba az = operátorral:

```
vector<int> x(10,56);
vector<int> y(5,10);

x=y;
```

A kód futása után x egy öt darab, integer típusú elemet tartalmazó vektor lesz, melynek minden eleme '10'. A vektorok típusának azonosnak kell lennie; méretük különbözhet.

1.2. Elemek elérése

1.2.1. T x[int i]

Paraméterek

`i`	`unsigned int`	A lekérni kívánt elem indexszáma. Ne feledjük, hogy az első elem indexe 0, nem pedig 1.

Visszatérési érték

`T`	A vektor `i` indexű eleme.

Egy vektor elemeit a legkönnyebben az `[ ]` operátorral érhetjük el:

```
vector<int> x(10,2)
int y=x[0];
```

A kód futása után y értéke '2' lesz.

VIGYÁZAT!

Ha olyan elemet próbálunk az `[]` operátorral lekérni, ami nem létezik, a programunk jó eséllyel meg fog halni. Ezért nagyon fontos, hogy sose próbáljunk olyan elemet lekérni, ami nem létezik! Mindig tartsuk észben azt is, hogy az elemek számozása 0-val kezdődik: az első elem indexe 0. Vagyis egy tízelemű vektor elemei 0-tól 9-ig számozódnak:

```
vector<int> x(10,56);
int y=x[10];
```

A fenti kód nagy eséllyel 'segmentation fault'-ot fog okozni, vagyis le fog állni; de ha nem is, komoly működési zavarokat okozhatunk.
Szóval: csak óvatosan.

A `[ ]` operátort értékadásra is használhatjuk (`x[0]=25;`, x első elemének értéke 25 lesz). Ha olyan indexű tagnak akarunk értéket adni, ami nem létezik, akkor a vektor előzékenyen megnyúlik:

```
vector<int> x(10,56);
x[50]=12;
```

A fenti kód eredményeként lesz egy 51 tagú vektorunk, melynek az első tíz eleme '56', az utolsó eleme 12, a többi pedig nulla.

Ugyan ez egy nagyon hasznos tulajdonsága a vektor osztálynak, ne éljünk vissza vele, rossz szokásokhoz vezet.

1.2.2. T x.at(int i)

Paraméterek

`i`	`unsigned int`	A lekérni kívánt elem indexszáma. Ne feledjük, hogy az első elem indexe 0, nem pedig 1. Ha `i` indexű elem nem létezik, `exception`t lök.

Visszatérési érték

`T`	A vektor `i` indexű eleme.

Általában ugyanúgy működik, mint az `x[i]` operátor, azzal a különbséggel, hogy `exception`t lök ha `i` indexszámú elem nem létezik. *Exception handling*et, azaz kivételkezelést még nem tanultunk, és kivételkezelés nélkül a programunk ugyanúgy meghal, de így is sokszor többre jutunk egy exceptionös hibaüzenettel, mint azzal, ha a program benyögi, hogy 'Segmentation fault'.

Merészebbeknek itt egy példa a használatára:

```
vector<int> x(10,5);
try {
        int i=x.at(664);
}
catch (exception& e) {
        cout << "Hiba: " << e.what() << endl;
}
cout << "... de a program fut!";
```

1.2.3. T x.front()

Visszatérési érték

`T`	A vektor első, azaz 0. indexszámú eleme.

Visszaadja a vektor első elemét. Ugyanazt jelenti, mintha azt mondanánk, hogy `x[0]`.

1.2.4. T x.back()

Visszatérési érték

`T`	A vektor utolsó eleme.

Visszaadja a vektor utolsó elemét. Ugyanazt jelenti, mintha azt mondanánk, hogy `x[x.size()-1]`.

1.3. Értékek módosítása

1.3.1. void x.assign(int n, T)

Paraméterek

n	unsigned int	Ennyi eleme legyen a vektornak
T	T	Ezt az értéket adja a vektor összes elemének. Típusának értelemszerűen azonosnak kell lennie a vektor típusával.

Értékadó funkció.

```
vector<int> x(10,56);
x.assign(5,20);
```

A fenti kód eredményeként a vektornak 5 eleme lesz, mindegyik értéke 20. A vektor eredeti tartalma elvész.

1.3.2. void x.push_back(T)

Paraméterek

T	T	A vektor típusának megfelelő típusú érték, amit szeretnénk betenni.

A T elemet beteszi a vektorba, az utolsó elem után. A vektor mérete értelemszerűen eggyel nő.

```
vector<int> x(5,56);
x.push_back(84);
```

A fenti kód eredményeként a vektornak 6 eleme lesz, az első öt eleme ’56’, az utolsó ’84’.

1.3.3. void x.pop_back(T)

Törli a vektor utolsó elemét.

```
vector<int> x(5,56);
x.pop_back();
```

A fenti kód eredményeként a vektornak 4 eleme lesz, mindegyik ’56’.

1.3.4. void x.swap(vector<T> y)

Paraméterek

y	vector<T>	Egy másik, ugyanolyan típusú vektor.

Két vektor tartalmának cseréje. A vektorok típusának azonosnak kell lennie, de méretük eltérhet.

```
vector<int> x(5,56);
vector<int> y(10,84);
x.swap(y); //'cseréld ki x-et és y-t'
```

A kód eredményeként x-nek 10 eleme lesz, mind '84'; y-nak pedig 5 eleme lesz, mind '56'.

1.4. Méret

1.4.1. int x.size()

Visszatérési érték

unsigned int	A vektor mérete.

Megadja a vektor méretét.

VIGYÁZAT!

Sose feledjük, hogy a vektor elemei nullától számozódnak! 'For' ciklusban tehát mindig legyen $<$, és nem $\leq$ a ciklusfeltételben:

```
vector<int> x(5,56);
for(unsigned int i=0; i < x.size(); i++)
        {
        x[i]=x[i]+1;
        }
```

A fenti kód eredményeként az öt elemből álló vektor minden eleme 57 lesz.

1.4.2. void x.resize(i, T)

Paraméterek

i	unsigned int	A vektor új mérete
T	T, opcionális	Kezdőérték. Ha a vektor méretét *növeljük*, ezzel az elemmel tölti fel az új helyeket. A típusa legyen ugyanaz, mint a vektor típusa. Ha nem adjuk meg, a típus alapértelmezett konstruktora hívódik meg.

Átméretezi a vektort. Ha a vektor méretét *csökkentjük*, akkor az első `i` elem változatlan marad, a többi elvész; ha a vektor méretét *növeljük*, akkor az eredeti elemek változatlanul megmaradnak, a maradék helyekre pedig adott `T` objektumok kerülnek. Ha nem adjuk meg `T`-t, a típus alapértelmezett konstruktora hívódik meg.

```
vector<int> x(5,56);
x.resize(10);
x.resize(2);
x.resize(4,84)
```

A fenti kódban x-et először deklaráltuk, lett öt eleme, mindegyik '56'; majd méretét 10-re növeltük (az első öt '56', a többi '0' - az alapértelmezett konstruktor hívódott meg); majd lecsökkentettük a méretét kettőre, mindkét megmaradt eleme '56' volt; végezetül megnöveltük a méretét 4-re, az első kettő eleme '56', az utolsó kettő eleme pedig '84' lett.

1.4.3. void x.clear()

Kiürít egy vektort. A vektor mérete nullára csökken, tartalma elvész.

```
vector<int> x(5,56);
x.clear();
```

1.4.4. bool x.empty()

Visszatérési érték	
`bool`	`true` ha a vektor üres, különben `false`

Megadja, hogy üres-e az adott vektor, azaz elemeinek száma kisebb-e egynél.

2. String

#include <string>

A string könyvtár definiálja a string típust. A stringre, azaz a szöveg típusú változóra szeretünk úgy gondolni, mint elemi típusra, de ha jobban belegondolunk, rájövünk, hogy egyáltalán nem az: a string nem más, mint karakterek, azaz char típusú változók sorozata, láncolata. Régen, a C nyelvben, bizony még char változókból álló tömbök voltak a szöveg típusú változók. A stringek ugyan sokkal kényelmesebben kezelhetők (ezért használjuk), mégis célszerű észben tartani, hogy itt tulajdonképpen tömbökről van szó, és ennek megfelelően sok olyan tagfüggvény van rájuk definiálva, mint amilyeneket a vektorokra használunk.

2.1. Operátorok

A string típusú változókra minden szokásos operátor definiálva van. Csak futólag, és a teljesség igénye nélkül:

1. Az = operátor az értékadás jele.
2. Az == operátor az egyenlőségvizsgálat jele. Két string akkor egyenlő, ha egymásnak megfelelő karaktereik egyenlőek.
3. A relációs operátorok (vagyis <, >, <= és >=) úgy működnek, mint a szótári rendezés: ami előbb szerepelne a szótárban, az a kisebb. A "macska" > "macsek" > "egérfelderítési bizottmány a közös jólétért"; vagyis két string relációját nem a hosszuk befolyásolja.
4. A + operátor összekapcsol két stringet; vagy egy string és egy char típusú változót.
5. A += operátor hozzácsatolja a stringhez az utána következő string vagy char típusú változót.

2.2. Karakterek elérése

2.2.1. char x[i]

Paraméterek

i	unsigned int	A lekérni kívánt karakter indexszáma. Ne feledjük, hogy az első karakter indexe 0, nem pedig 1.

Visszatérési érték

char	A string i indexű karaktere.

```
string x="Macska";
for(unsigned int i=0; i < x.length(); i++)
        {
        cout << "Az " << i << ". karakter:" << x[i] << "\n";
        }
```

VIGYÁZAT!
Itt még fokozottabban érvényes, amit a vektoroknál elmondtunk: próbáljunk SOHA nem hivatkozni olyan elemre, ami nem létezik.

Az `[]` operátor persze értékadásra is használható:

```
string x="Macska";
x[0]='k';
```

A fenti kód eredményeképp a string tartalma "kacska" lesz.

VIGYÁZAT!
Ha nem létező elemnek próbálunk értéket adni, a programunk látványosan meg fog halni. A vektor ilyen esetben megpróbálja átméretezni magát, de a string már nem ilyen előzékeny.

VIGYÁZAT!
Amikor a string egyes karaktereinek új értéket adunk, a `char` típusú változót mindig aposztrófok (`'`), és ne idézőjelek (`"`) közé tegyük! Az aposztrófok közé tett karakterek karakterek, vagy C-stringek; az idézőjelek közé tett karakterek C++ stílusú stringek. A két típus között van átjárás ugyan, de nem felcserélhetőek.

2.2.2. char x.at(i)

Paraméterek

`i`	`unsigned int`	A lekérni kívánt karakter indexszáma. Ha `i` indexszámú karakter nem létezik, `exception`t lök.

Visszatérési érték

`char`	A vektor `i` indexszámú karaktere.

2.3. Beolvasás

2.3.1. istream& getline(istream& is, string str, char delim)

Paraméterek

`is`	`istream&`	Referencia szerint átadott `istream` objektum (pl `ifstream`, `cin`), amelyből ki akarjuk szedni az adatot.
`str`	`string`	A string, amelybe bepakolja a streamből kiszedett adatot.
`delim`	`char`	Opcionális. A 'delimiter', az a karakter, amedig ki akarunk szedni adatot. Opcionális, alapértelmezetten '\n'.

Visszatérési érték

`istream&`	A referencia szerint átadott objektum. Nyugodtan elvethető, kezelhető `void`ként.

Fogja `is` stream objektumot, és kiszed belőle minden adatot egészen a `delim` karakterig. Ha a `delim` karakter nem található, a beolvasás a stream végéig tart. A `delim` karakter maga is beolvasásra kerül, de nem kerül bele a stringbe, elvetjük.

VIGYÁZAT!

A sorvégjellel nem lehet viccelni.

Ha streamből olvasunk be, nagyon vigyázzunk a sorvégjellel, és vagy a getline()-t használjuk, vagy a >> operátort, de ne keverjük a két technikát.

```
string x, y;
cin >> x;
getline(cin,y);
```

A fenti kis kódtól azt várnánk, hogy előbb beolvas egy stringet a konzolról, majd ismét user inputot kér, és beolvas egy egész sort. A valóságban viszont csak egyszer fog user inputot kérni, beolvas egy stringet, majd y-ba bekerül a user input maradéka – a >> operátor ugyanis otthagyja a sorvégjelet! Persze meg lehet próbálni kikerülni a csapdát:

```
string x, y;
cin >> x >> ws;
getline(cin,y);
```

A fenti kód a vártnak megfelelően fog működni, mert a >> ws operátor beolvassa és elveti a sorvégjelet. De ha a user az első inputban egynél több szót írt be, a >> csak az első szóközig fog beolvasni, a getline pedig beolvassa a maradékot...
Kétségkívül meg lehet oldani a dolgot, de a hibalehetőségek száma nagyon magas. Válasszuk ki a feladathoz illő technikát, és csak azt használjuk.

2.4. Méret

2.4.1. int x.size()

Visszatérési érték	
`unsigned int`	A `string` hossza.

Megadja a string hosszát. Azonos a `x.length()`-szel.

2.4.2. int x.length()

Visszatérési érték	
`unsigned int`	A `string` hossza.

Megadja a string hosszát. Azonos a `x.size()`-zal.

VIGYÁZAT!
Nem lehet elégszer hangsúlyozni, hogy az elemek nullától számozódnak, és hogy nagy szarba kerülhetünk, ha nem létező elemre hivatkozunk.

2.4.3. void x.resize(int i, char c)

Paraméterek

`i`	`unsigned int`	A string új mérete
`c`	`char`	Kezdőérték. Ha a string méretét növeljük, ezzel a karakterrel töltődik fel.

Átméretezi a stringet. Ha a string mérete csökken, a levágott karakterek elvesznek; ha a string mérete nő, az adott `char` karakterrel töltődik fel.

```
string x="Csacska macska";
x.resize(8);
x.resize(14,'*');
```

A fenti kód eredményeként a string először "Csacska "-ra változik, majd "Csacska ******"-ra.

2.4.4. void x.clear()

Kiüríti a stringet. A string mérete nullára csökken, tartalma elvész.

2.4.5. bool x.empty()

Visszatérési érték

`bool`	`true` ha a string hossza nulla, különben `false`

Megadja, hogy 'üres'-e az adott string, azaz hossza kisebb-e egynél.

2.5. Módosítás

2.5.1. string& x.append(string& str)

Paraméterek

`str`	`string`	A string, amit hozzácsatolunk a stringhez.

Az `append` első verziója: hozzácsatol egy stringet a stringhez. Megegyezik a `+=` operátorral.

2.5.2. string& x.append(string& str, int pos1, int pos2)

Paraméterek

`str`	`string`	A string, aminek egy részét hozzácsatoljuk a stringhez.
`pos1`	`unsigned int`	A kezdőindex.
`pos2`	`unsigned int`	A végzőindex.

Az `append` második, izgalmasabb változata: `str` `pos1` és `pos2` közötti darabját csatolja a stringhez.

```
string x="kicsi";
string str="kicsi macska mocska";
x.append(str,5,11)
```

A fenti kód eredménye "kicsi macska".

2.5.3. string& x.append(int n, char c)

Paraméterek

`n`	`unsgined int`	Ennyiszer illesztjük hozzá a karaktert a stringhez.
`c`	`char`	A karakter, amit `n`-szer hozzáillesztünk a stringhez.

Az `append` harmadik változata: `n` darab `c` karaktert illeszt hozzá a stringhez.

```
string x="Hurrá";
char c='!'
x.append(c,3)
```

A kód eredménye "Hurrá!!!".

2.6. Stringmanipuláció

2.6.1. char* x.c_str()

Visszatérési érték

`char*` (C-string)	A string C-string változata.

Átalakítja a "modern" stringet régimódi C-stringgé. Több C függvényhez szükséges.

```
string x="myfile.txt";
istream f;
f.open(x.c_str());
```

Az `f.open(x);` nem fordulna.

2.6.2. int x.find(string& str, int pos)

Paraméterek

str	string&	Ezt a stringet keressük a stringben.
pos	unsigned int	Ennyiedik karaktertől keresünk. Opcionális, alapesetben 0.

Visszatérési érték

unsigned int	A keresett szövegrész elejének indexszáma a stringben. Ha nincs meg a string, a visszatérési érték string::npos.

Megkeresi a stringben str szövegrész első előfordulását.

```
string x="Kedves, kicsi csacska macska.";
string str="macska";
if(x.find(str)!=string::npos)
        {
        cout << "Megvan a macska!!!";
        }
else
        {
        cout << "Nincs benne macska :/";
        }
```

2.6.3. int x.find(char c, int pos)

Paraméterek

c	char	Ezt a karaktert keressük a stringben.
pos	unsigned int	Ennyiedik karaktertől keresünk. Opcionális, alapesetben 0.

Visszatérési érték

unsigned int	A keresett karakter indexszáma a stringben. Ha nincs meg a karakter, a visszatérési érték string::npos.

Megkeresi a stringben c karakter első előfordulását.

```
string x="Kedves, kicsi csacska macska.";
if(x.find(',', 9)!=string::npos)
        {
        cout << "Az elso vesszo a ";
        cout << x.find(',', 9) << ". helyen van.";
        }
else
        {
```

```
        cout << "Nincsen benne vesszo egy sem.";
        }
```

2.6.4. int x.rfind(string& str, int pos)

Paraméterek

str	string&	Ezt a stringet keressük a stringben.
pos	unsigned int	Ennyiedik karakterig keresünk. Opcionális, alapesetben x.size().

Visszatérési érték

unsigned int	A keresett szövegrész elejének indexszáma a stringben. Ha nincs meg a string, a visszatérési érték string::npos.

Megkeresi a stringben str szövegrész *utolsó* előfordulását.

2.6.5. int x.rfind(char c, int pos)

Paraméterek

c	char	Ezt a karaktert keressük a stringben.
pos	unsigned int	Ennyiedik karakterig keresünk. Opcionális, alapesetben x.size().

Visszatérési érték

unsigned int	A keresett karakter indexszáma a stringben. Ha nincs meg a karakter, a visszatérési érték string::npos.

Megkeresi a stringben c karakter *utolsó* előfordulását.

2.6.6. int x.find_first_of(string str, int pos)

Paraméterek

string	str	Az ebben a stringben lévő karakterek valamelyikét keressük.
pos	unsigned int	Ennyiedik karaktertől keresünk. Opcionális, alapesetben 0.

Visszatérési érték

unsigned int	A keresett karakterek valamelyikének indexszáma a stringben. Ha nincs meg a karakter, a visszatérési érték string::npos.

Megkeresi a stringben az str stringben lévő karakterek *bármelyikét*.

```
string x="Analízist tanulgat az én babám.";
if(x.find_first_of("0123456789")!=string::npos)
        {
        cout << "Van benne szamjegy."
        }
else
        {
        cout << "Nincsen benne szamjegy.";
        }
```

2.6.7. int x.find_last_of(string str, int pos)

Paraméterek

string	str	Az ebben a stringben lévő karakterek valamelyikét keressük.
pos	unsigned int	Ennyiedik karakterig keresünk. Opcionális, alapesetben x.size().

Visszatérési érték

unsigned int	A keresett karakterek valamelyikének indexszáma a stringben. Ha nincs meg a karakter, a visszatérési érték string::npos.

Megkeresi a stringben az str stringben lévő karakterek bármelyikének utolsó előfordulását.

2.6.8. string x.substr(int pos, int n)

Paraméterek

pos	unsigned int	Az ennyidik karaktertől vágjuk ki a szövegrészt.
n	unsigned int	Ennyi karaktert vágunk ki a szövegből. Opcionális.

Visszatérési érték

string	A kívánt szövegrész.

Kiszedi a stringből a pos indexszámú karaktertől n karakter hosszúságú szövegrészt, és visszaadja. Ha n nincs megadva, pos-tól a string végéig vágja ki a szövegrészt, és azt adja vissza.

3. Stringstream

#include <sstream>

A sstream könyvtár definiálja a stringstream típust. A stringstream lényege, hogy tetszőleges stringet streamként kezelhessünk, ugyanúgy, ahogy a konzolinputot (cin) vagy egy beolvasott file-t, stb, ezzel lényegesen leegyszerűsítve az adatfeldolgozást. Én kedvelem.

Így deklaráljuk:

```
stringstream x;
```

3.1. Értékadás

3.1.1. void x.str(string s)

Paraméterek

s	string	A szöveg, amit bele akarunk tenni a stringstreambe.

A x.str() első változata, értékadó függvény. Használata egyszerű:

```
stringstream x;
string t="Macska";
x.str(t);
```

3.1.2. void x << string s

Paraméterek

s	string	A szöveg, amit bele akarunk tenni a stringstreambe.

A standard << operátor, amit minden ostream objektumra (pl cout vagy ofstream) használunk:

```
stringstream x;
string s="Macska";
x << s;
```

Az << operátorral használhatjuk az iomanip könyvtárban definiált formázó függvényeket, melyek leírása egyszer talán bekerül ebbe a dokumentumba is.

3.2. Adatok kinyerése

3.2.1. Öröklött függvények és operátorok

A stringstreamből ugyanúgy nyerhetünk ki adatokat, mint bármely `istream` objektumból. Használhatjuk a `getline()` függvényt (ld. ??) és az `>>` operátort is, a megszokott módon.

```
stringstream x;
x.str("1986 kemény telén sok hó esett.");
int num;
string word, line;

x >> num >> word;
getline(x, line);
```

A kód futása után `num` tartalma "1986", `word` tartalma "kemény", míg `line` tartalma " telén sok hó esett.".

3.2.2. string x.str()

Visszatérési érték	
`string`	A stringstream tartalma.

A `x.str()` második változata, egyszerűen stringgé alakítja és visszaadja a stringstream teljes tartalmát.

```
stringstream x;
string t="Macskám";
int i=1986;
x << t << " fázott " << i << "-ban."
string res=x.str();
```

A kód futása után `res` tartalma "Macskám fázott 1986-ban." lesz.

Láthatjuk, hogy a stringstream kiválóan alkalmas stringek és számok összefűzűsére, ami különben problémás lehet.

VIGYÁZAT!

Fontos mindig emlékezni rá, hogy bár azt mondjuk "adatot szedünk ki a streamből", sem az `>>` operátor, sem a `getline()` függvény nem távolít el adatot a streamből! Például

```
stringstream x;
x.str("1986 kemény telén sok hó esett.");
int num;
string word, line;
x >> num >> word;

line=x.str();
```

`line` tartalma "1986 kemény telén sok hó esett." lesz, annak ellenére, hogy "kiszedtünk" már a stringstreamből adatot. Ha a stringstreamet üríteni, resetálni akarjuk, csináljuk így:

```
x.str("");
```

Vagyis egyszerűen tegyünk bele egy üres stringet.

4. C Character Type

```
#include <cctype>
```

A `cctype` könyvtárban található funkciók a `char`, azaz karakter típusú változók jellegével foglalkoznak. Szerintem nagyon hasznos kis cuccok.

4.1. Karakter jellegének vizsgálata

4.1.1. bool isalnum(char c)

Paraméterek

`c`	`char`	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

`bool`	`true` ha a vizsgált karakter alfanumerikus, vagyis vagy számjegy, vagy az ábécé betűinek egyike (tehát nem kontrollkarakter, írásjel, stb), különben `false`.

4.1.2. bool isalpha(char c)

Paraméterek

`c`	`char`	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

`bool`	`true` ha a vizsgált karakter *alfabetikus*, vagyis az ábécé betűje (nem szám, nem kontrollkarakter, írásjel, stb), különben `false`.

4.1.3. bool iscntrl(char c)

Paraméterek

`c`	`char`	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

`bool`	`true` ha a vizsgált karakter kontrollkarakter (pl `\n`, `\a`, `\t` stb), különben `false`.

4.1.4. bool isdigit(char c)

Paraméterek

c	char	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

bool	true ha a vizsgált karakter számjegy (0, 1, 2, 3, 4, 5, 6, 7, 8 vagy 9), különben false.

4.1.5. bool islower(char c)

Paraméterek

c	char	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

bool	true ha a vizsgált karakter az ábécé kisbetűje, különben false.

4.1.6. bool ispunct(char c)

Paraméterek

c	char	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

bool	true ha a vizsgált karakter írásjel (a függvény szempontjából minden olyan karakter, ami nem kontrollkarakter, nem alfanumerikus, és nem whitespace), különben false.

4.1.7. bool isspace(char c)

Paraméterek

c	char	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

bool	true ha a vizsgált karakter whitespace (' ', '\n', '\t', '\v', '\f' vagy '\r'), különben false.

4.1.8. bool islower(char c)

Paraméterek

c	char	A vizsgált karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

bool	true ha a vizsgált karakter az ábécé nagybetűje, különben false.

4.2. Karakterek átalakítása

4.2.1. char tolower(char c)

Paraméterek

c	char	Az átalakítani kívánt karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

char	A karakter kisbetűs változata, ha ilyen létezik; ha nem létezik, változatlanul visszaadja a karaktert.

4.2.2. char toupper(char c)

Paraméterek

c	char	Az átalakítani kívánt karakter. A függvény akármilyen számot is elfogad, amit karakterré alakít.

Visszatérési érték

char	A karakter nagybetűs változata, ha ilyen létezik; ha nem létezik, változatlanul visszaadja a karaktert.

```
        string x="Kicsi Macska";
        for(unsigned int i=0; i<x.length(); i++)
                {
                if(islower(x[i]))
                        {
                        x[i]=toupper(x[i]);
                        }
                else if(isupper(x[i]))
                        {
                        x[i]=tolower(x[i]);
```

```
                    }
                }
        cout << x;
        // kICSI mACSKA
```

5. C Math

```
#include <cmath>
```

Matematikai műveletek.

5.1. Trigonometria

5.1.1. double cos(double x)

Paraméterek

x	double	A szög radiánban.

Visszatérési érték

double	A szög koszinusza.

5.1.2. double sin(double x)

Paraméterek

x	double	A szög radiánban.

Visszatérési érték

double	A szög szinusza.

5.1.3. double tan(double x)

Paraméterek

x	double	A szög radiánban.

Visszatérési érték

double	A szög tangense.

5.1.4. double acos(double x)

Paraméterek

x	double	Egy szög koszinusza, lebegőpontos szám a [-1, 1] intervallumban.

Visszatérési érték

double	A szám arkusz koszinusza radiánban. Ha a szám nem esik bele a [-1, 1] intervallumba, az eredmény "Not a number" lesz.

5.1.5. double asin(double x)

Paraméterek

x	double	Egy szög szinusza, lebegőpontos szám a [-1, 1] intervallumban.

Visszatérési érték

double	A szám arkusz szinusza radiánban. Ha a szám nem esik bele a [-1, 1] intervallumba, az eredmény "Not a number" lesz.

5.1.6. double atan(double x)

Paraméterek

x	double	Egy szög tangense.

Visszatérési érték

double	A szám arkusz tangense. Az előjelek kiesése miatt a függvény nem tudja megállapítani, melyik síknegyedbe esik a szög, ami gondokat okozhat pl komplex számok átalakításakor.

5.1.7. double atan2(double y, double x)

Paraméterek

y	double	A tangens y koordinátája.
x	double	A tangens x koordinátája.

Visszatérési érték

double	Az y/x szám arkusz tangense. Az atan() függvénnyel ellentétben sikeresen megállapítja azt is, mely síknegyedbe esik a szög.

5.2. Hiperbolikus függvények

5.2.1. double cosh(double x)

Paraméterek

x	double	Lebegőpontos szám.

Visszatérési érték

double	x koszinusz hiperbolikusza.

5.2.2. double sinh(double x)

Paraméterek

x	double	Lebegőpontos szám.

Visszatérési érték

double	x szinusz hiperbolikusza.

5.2.3. double tanh(double x)

Paraméterek

x	double	Lebegőpontos szám.

Visszatérési érték

double	x tangens hiperbolikusza.

5.3. Exponenciális és logaritmusfüggvények

5.3.1. double exp(double x)

Paraméterek

x	double	Lebegőpontos szám.

Visszatérési érték

double	e^x - e az x-edik hatványra emelve.

5.3.2. double log(double x)

Paraméterek

x	double	Lebegőpontos szám, legyen pozitív!

Visszatérési érték

double	$\ln x$ - x természetes alapú logaritmusa.

5.3.3. double log10(double x)

Paraméterek

x	double	Lebegőpontos szám.

Visszatérési érték

double	$\log_{10} x$ - x tizes alapú logaritmusa.

5.3.4. double modf(double x, double* egeszresz)

Paraméterek

x	double	Lebegőpontos szám, aminek az egész- és törtrésze érdekel.
egeszresz	double*	Mutató ahhoz a double típusú változóhoz, ahová az egészrészt szeretnénk betenni.

Visszatérési érték

double	x törtrésze.

Előjeles maradékos osztás: a visszatérési érték a szám törtrésze; az egészrész pedig bekerül az egeszresz változóba. Így használjuk:

```
double x, egeszresz, tortresz;
x=2.71828;

tortresz=modf(x, &egeszresz);
```

A kód futása után egeszresz változó értéke 2.000000 lesz, a tortresz változóé pedig 0.71828.

Fontos a & jel az egeszresz változó neve előtt! Ezzel a jellel az egeszresz változóra mutató *mutatót* adjuk át a funkciónak, nem magát a változót.

5.4. Hatványfüggvények

5.4.1. double pow(double a, double x)

Paraméterek

a	double	Az alap.
x	double	A kitevő.

Visszatérési érték

double	a^x - az alap a kitevő hatványára emelve.

5.4.2. double sqrt(double x)

Paraméterek

x	double	A szám, amiből gyököt szeretnénk vonni.

Visszatérési érték

double	$\sqrt{x}$ - x négyzetgyöke.

5.5. Kerekítés és abszolútérték

5.5.1. double abs(double x)

Paraméterek		
`x`	`double`	Lebegőpontos szám.

Visszatérési érték	
`double`	`x` abszolútértéke.

5.5.2. double ceil(double x)

Paraméterek		
`x`	`double`	A kerekítendő tört.

Visszatérési érték	
`double`	A legkisebb, `x`-nél nagyobb egész szám (`x` felfele kerekítve).

5.5.3. double floor(double x)

Paraméterek		
`x`	`double`	A kerekítendő tört.

Visszatérési érték	
`double`	A legnagyobb, `x`-nél kisebb egész szám (`x` lefele kerekítve).

6. C Standard General Utilities Library

#include <cstdlib>

"C Standard Általános Eszközök Könyvtár"

6.1. String átalakítása számmá

6.1.1. double atof(char* str)

Paraméterek		
`str`	`char*` (C-string)	A C-string, amit számmá akarunk átalakítani.

Visszatérési érték	
`double`	A szöveg számértéke.

Ha a string whitespace-ekkel kezdődik, a függvény ezeket átugorja, ezután kezdődik a konverzió. Ezután beolvas minden olyan karaktert, amelyek együtt emlékeztetnek egy lebegőpontos számra, vagyis illeszkednek a következő mintába:

- opcionálisan egy darab `+` vagy `-` jel;
- számjegyek sorozata, ami tartalmazhat egy pontot ('.');
- opcionálisan az exponenciális rész, vagyis egy `e` vagy `E` karakter, amit egy `+` vagy `-` jel és számjegyek sorozata követ.

Amint a függvény olyan karakterre bukkan, ami ebbe a mintába nem illeszkedik, a függvény abbahagyja a konverziót. Ha a konverzió vége előtt nem talál érvényes mintát, nullát ad vissza.

Ha 'modern', c++ stringet szeretnénk átalakítani, konvertáljuk a `c_str()` függvénnyel.

```
        double res;
        string x="1984ad";
        res=atof(x.c_str());
        cout << res << "\n";
        //'1984'

        string y="kr.u.1984";
        res=atof(y.c_str());
        cout << res << "\n";
        //'0'
```

6.1.2. int atoi(char* str)

Paraméterek

`str`	`char*` (C-string)	A C-string, amit számmá akarunk átalakítani.

Visszatérési érték

`int`	A szöveg számértéke.

Az adott C-stringet integerré próbálja alakítani. A stringkezdő whitespace-eket átugorja, majd beolvas minden olyan karaktert, ami emlékeztet egy egész számra, vagyis:

- opcionálisan egy darab `+` vagy `-` jel;
- számjegyek sorozata.

Amint olyan karakterre bukkan, ami ebbe az egyszerű mintába nem illik, a függvény abbahagyja a konverziót. Ha egyáltalán nem talált a mintának megfelelő karaktereket, nullát ad vissza.

6.2. Véletlenszám

6.2.1. int rand()

Visszatérési érték

`int`	Egy szám 0 és a `RAND_MAX` konstans között.

Használata előtt mindig hívjuk meg az `srand()` függvényt!

6.2.2. void srand(int seed)

Paraméterek

`seed`	`unsigned int`	Az a nullánál nagyobb egész szám, amit a véletlenszám-generáló algoritmus magként fog használni.

Inicializálja a véletlenszám-generáló algoritmust.

```
srand( time(NULL) );
int res=rand() % 10 +1;
```

I. Függelék: adattípusok

A teljesség igénye nélkül.

Név	Értékkészlet	Méret (byte)
`bool`	{`true`, `false`}	1
`char`	[-128, 127]	1
`unsigned char`	[0, 255]	1
`int`	[-2147483648, 2147483647]	4
`unsigned int`	[0, 4294967295]	4
`short int`	[-32768, 32767]	2
`unsigned short int`	[0, 65535]	2
`long int`	[-2147483648, 2147483647]	4
`unsigned long int`	[0, 4294967295]	4
`float`	±3.4 e±38 (kb. 7 tizedesjegy)	4
`double`	±1.7 e±308 (kb. 15 tizedesjegy)	8

A fentieken kívül szót érdemel még a `void`: szó szerint "üresség". Azon funkciók visszatérési értéke, amelyek nem adnak vissza értéket. Az ilyen funkciókat *eljárás*nak nevezzük, és nem szükséges `return` parancsot tenni beléjük. `void` típusú változó nincs. Matekosoknak: az üres halmaz.

i. Típuskonverzió

Az elemi típusok között van átjárás. Még a `bool` típus is numerikus: a 0 szám `false`-nak számít, és minden $x > 0$ érték `true`-nak (ha nincs jel, az hamis, ha van jel, az igaz). Hasonlóképp, `int`-ek jelölhetnek kiíratható karaktereket is. Amikor valahogy két típus között átjárunk, típuskonverziót hajtunk végre. A típuskonverzió lehet *implicit* és *explicit*.

Implicit a típuskonverzió akkor, ha mi magunk nem írunk le semmit, és a fordítót kényszerítjük rá, hogy végezze el nekünk a konverziót:

```
double d=5.524;
int i=7;

if(i>d)
        {
        i=d;
        cout << i; //'5'
        }
```

A fenti kódban kétszer történik konverzió: egyszer összehasonlítunk egy `double` és egy `int` értéket, egyszer meg lebegőpontos értéket adunk egy `int` változónak. A compiler hiba nélkül elvégzi a dolgot, de óhatatlanul adatvesztés történik.

Explicit konverzióról akkor beszélünk, ha egy paranccsal megkérjük a compilert, hogy konvertáljon egyik adattípusból a másikba:

```
int i=90;
cout << static_cast<char>(i); //'Z'
```

A fenti példában a `static_cast<>()` funkció hatására a program `char` típusú változóként értékeli `i`-t a kifejezésben.

A `static_cast<>()` a legmodernebb és preferált módja a típuskonverziónak, de ha *nagyon* sietünk, súlyosabb büntetés nélkül használhatjuk a C stílusú típuskonverziót:

```
int i=90;
cout << (char)i; //'Z'
```

ii. Pár szó az előjelről

Egy típus előjeles és előjel nélküli változata két külön típusnak számít! Ennek megfelelően szigorúbb compilerek figyelmeztetést adnak, ha egy típus előjel nélküli (`unsigned`) és előjeles (`signed`) változatát próbáljuk összehasonlítani.

```
vector<int> x{10,25}
for(int i=0; i<x.size(); i++)
        {
        x[i]+=5;
        }
```

A fenti kód figyelmeztetést generál, mert az `x.size()` visszatérési értéke `unsigned int`, míg az `i signed int`-ként van deklarálva.

A program ettől még fordulni fog, és az esetek nagy többségében soha semmiféle galiba nem lesz belőle, a nagyon is létező hibalehetőségeket, a konzisztenciát és a, hm, szépséget észben tartva mégis hasznos lehet odafigyelni.