2036. július 22. A Merkúr, a Mars, és a Szaturnusz hármas együttállása, egy 1 fokos körben, 20 fokra a Naptól.

2036. augusztus 7. Teljes holdfogyatkozás 02:02-03:38 között, 96 percig teljes (145,4 %).

2036. augusztus 21. Részleges napfogyatkozás a Föld északi részén, 86 %-os maximális fázissal. Hazánkban 26 %-os fázis látható napnyugtakor.

2037. január 16. Részleges napfogyatkozás a Föld északi részén 71 %-os maximális fázissal. Hazánkban 48 %-os részleges fogyatkozás 09:27 körül.

2037. szeptember 15. A Merkúr és a Szaturnusz igen szoros látszó közelségben, a két bolygó korongja között csak 9 ívmásodperc távolság lesz! Szabad szemmel egy csillagnak látszanak, csak távcsovel bonthatók ketté.

2038–39. A Szaturnusz gyururendszere három alkalommal is élérol látszik, a bolygót gyuru nélkülinek látjuk.

2038. január 5. 97,3 %-os gyurus napfogyatkozás Nigéria, Csád, Szudán, Egyiptom területén a déli órákban. A maximális sáv 2700 km-re lesz tolünk. Legközelebbi ország: Egyiptom. Idotartama 3m19s. Szélessége 107 km. Hazánkban a részleges napfogyatkozás 14:02-kor kezdodik, és a nálunk 20 %-os maximális fázisában nyugszik a Nap.

2038. július 2. 99,1 %-os gyurus napfogyatkozás Marokkó, Mauritánia, Mali, Algéria, Niger vonalon a délutáni órákban. A maximális fázis 2900 km-re lesz tolünk. Legközelebbi ország: Marokkó. Idotartama 1m00s. Szélessége 31 km. Nálunk 5-10 %-os részleges fogyatkozás 14.26 körül.

2039. június 21. 94,5%-os gyurus napfogyatkozás Norvégia, Svédország, Finnország területén a délutáni-esti órákban. A maximális fogyatkozás sávja 900 km-re tolünk, Minszknél végzodik. Idotartama 4m05s. Szélessége 365 km. Hazánkban részleges fogyatkozás 17:33-tól, a Nap 72 %-os maximális fázissal nyugszik.

2039. november 7. A Merkúr átvonulása a Nap elott, annak déli peremén 07:18-10:18 között. Hazánkból végig megfigyelheto.

2040. szeptember 1. A Vénusz, a Mars, a Jupiter, és a Szaturnusz négyes együttállása, egy 6 fokos körben, 29 fokra a Naptól.

2040. szeptember 7. A Merkúr, a Vénusz, a Jupiter, és a Szaturnusz négyes együttállása, egy 6 fokos körben, 24 fokra a Naptól.

2040. szeptember 11. A Merkúr, a Vénusz, a Mars, és a Szaturnusz négyes együttállása, egy 7 fokos körben, 25 fokra a Naptól.

2040. november 18. Teljes holdfogyatkozás 18:18-19:46 között, 88 percig teljes (139,6 %).

Összeállította**: Keszthelyi Sándor,** Pécs

A programozási nyelvek elemei

II. rész

Típusok

Egy adat típusa definiálja azt a halmazt, amelybol az adat mint változó, értékeket vehet fel, az adat által a memóriában lefoglalt helyet, méretét, és ugyanakkor definiálja azokat a muveleteket is, melyek az adattal elvégezhetok.

Típus = értékhalmaz + muvelethalmaz

Minden programozási nyelv definiál egy alap típushalmazt, amely rendszerint a számítógép típushalmazával egyezik meg. Az alaptípusok három csoportba oszthatók: ***aritmetikai*** (egész és valós) típus nagyon közel áll a fizikai géphez és egy ilyen típusú

változóval aritmetikai muveleteket lehet végezni. A ***logikai*** típusú változó két értéket vehet fel: **igaz** és **hamis** és logikai muveleteket lehet vele végezni. A ***karakter*** típusú változó értékét általában az ASCII táblázatból veheti fel.

Az alaptípusokon kívül a magas szintu programozási nyelvek megengedik az ún. ***felhasználói típus*** (*user type*) deklarálását is. Ezek a típusok az alaptípusokra épülnek, de továbbfejlesztik azokat.

Borland Pascal

Érdekes az, ahogy a Pascal megoldja a típuskezelést. A Pascalban tulajdonképpen tíz alaptípus van, az összes többi típus ezekbol származik. Ezek az alaptípusok a következok:

felsorolt: `()` közé írt azonosító-lista; **intervallum**: alsóhatár`..`felsohatár; **karaktersorozat**: `string`; **tömb**: `array`; **halmaz**: `set`; rekord (**bejegyzés)**: `record`; **referencia**: `^`; **eljárás**, **függvény**: `procedure`, `function`; **objektum**: `object`; **állomány**: `file`, `text`.

Az összes többi típust a Pascal ezekbol a típusokból származtatja, kivéve a valós típusokat, amelyeket a koprocesszort vezérlo egységben implementálja. Ezek elsorendu származtatott típusok és az alaptípusoktól, felhasználói szinten, nem szoktuk oket megkülönböztetni. Ezek a típusok a következo típusosztályokat képezik: *egyszeru*; *karaktersorozat*; *strukturált* (összetett); *eljárás/függvény*; *mutató*.

A felhasználó által definiált típusokat a `type` deklarációban (cikkely) kell leírni. Ez a deklaráció új típusokat értelmez, amelyeknek változóit késobb a `var` deklarációban lehet leírni.

Egyszeru típusok

? Sorszámozott típusok

Típus	*Ábrázolás*	*Muveletek*	*Eljárások, függvények*
`Boolean`	1 byte (`False`, `True`)	`and`,`or`,`xor`,`not`,`:=`, `<`,`>`,`<=`,`>=`,`<>`	Succ(), Ord(), Pred()
`WordBool`	2 byte (`False`, `True`)	`and`,`or`,`xor`,`not`,`:=`, `<`,`>`,`<=`,`>=`,`<>`	Succ(), Ord(), Pred()
`LongBool`	4 byte (`False`, `True`)	`and`,`or`,`xor`,`not`,`:=`, `<`,`>`,`<=`,`>=`,`<>`	Succ(), Ord(), Pred()
`ByteBool`	1 byte (`False`, `True`)	`and`,`or`,`xor`,`not`,`:=`, `<`,`>`,`<=`,`>=`,`<>`	Succ(), Ord(), Pred()
`Char`	1 byte `#0..#255`	`:=`,`<`,`>`,`<=`,`>=`,`<>`	Ord(), Chr(), Pred(), Succ()
`Byte`	1 byte `0..255`	`+`, `-`, `*`, `div`, `mod`, `/`, `and`,`or`, `xor`, `shl`, `shr`, `not`, `:=`,`<`,`>`,`<=`,`>=`,`<>`	Abs(), Sqrt(), Sqr(), Val()
`ShortInt`	1 byte `-128..127`	`+`, `-`, `*`, `div`, `mod`, `/`, `and`,`or`, `xor`, `shl`, `shr`, `not`, `:=`,`<`,`>`,`<=`,`>=`,`<>`	Abs(), Sqrt(), Sqr(), Val()
`Integer`	2 byte `-32768 .. 32767`	`+`, `-`, `*`, `div`, `mod`, `/`, `and`,`or`, `xor`, `shl`, `shr`, `not`, `:=`,`<`,`>`,`<=`,`>=`,`<>`	Abs(), Sqrt(), Sqr(), Val()
`Word`	2 byte `0..65535`	`+`, `-`, `*`, `div`, `mod`, `/`, `and`,`or`, `xor`, `shl`, `shr`, `not`, `:=`,`<`,`>`,`<=`,`>=`,`<>`	Abs(), Sqrt(), Sqr(), Val()
`LongInt`	4 byte `-2147483648 ..`	`+`, `-`, `*`, `div`, `mod`, `/`, `and`,`or`, `xor`,	Abs(), Sqrt(), Sqr(), Val()

Típus	*Ábrázolás*	*Muveletek*	*Eljárások, függvények*
	2147483647	shl, shr, not, :=,<,>,<=,>=,<>	
Felsorolt	( ) közé írt azonosítólista. Pl. szin = (piros, kek, feher)	:=,<,>,<=,>=,<>, in	Succ(), Ord(), Pred()
Intervallum	Egy gazda típus értékei. Pl. szam = 0..200	:=,<,>,<=,>=,<>, in	Succ(), Ord(), Pred()

? Valós típusok

Típus	*Ábrázolás*	*Muveletek*	*Eljárások, függvények*
Real	6 byte -2.9E-39 .. 1.7E38	+, -, *, /, :=, <,>,<=,>=,<>	Abs(), Sqrt(), Sqr(), Val(), Trunc(), Int(), Round(), Fract()
Single	4 byte -1.5E-45 .. 3.4E38	+, -, *, /, :=, <,>,<=,>=,<>	Abs(), Sqrt(), Sqr(), Val(), Trunc(), Int(), Round(), Fract()
Double	8 byte -5E-324 .. 1.7E308	+, -, *, /, :=, <,>,<=,>=,<>	Abs(), Sqrt(), Sqr(), Val(), Trunc(), Int(), Round(), Fract()
Extended	10 byte 3.4E-4932 .. 1.1E4932	+, -, *, /, :=, <,>,<=,>=,<>	Abs(), Sqrt(), Sqr(), Val(), Trunc(), Int(), Round(), Fract()
Comp	8 byte -9.2E18 .. 9.2E18 (tört rész nélküli)	+, -, *, /, :=, <,>,<=,>=,<>	Abs(), Sqrt(), Sqr(), Val(), Trunc(), Int(), Round(), Fract()

Karaktersorozat típus

Típus	*Ábrázolás*	*Muveletek*	*Eljárások, függvények*
String	256 * 1 byte A 0. byte a string hossza.	+,:=,<,>,<=, >=,<>	Length(),Copy(), Str(), Concat(), Pos(), Insert(), Delete()
String [hossz]	(hossz + 1) byte A 0. byte a string hossza.	+,:=,<,>,<=, >=,<>	Length(),Copy(), Str(), Concat(), Pos(), Insert(), Delete()

Strukturált típusok

? Tömb

Típus	*Ábrázolás*	*Muveletek*	*Eljárások, függvények*
array[alsó .. felso] of alaptípus	(felso - alsó + 1) * alaptípushossz byte	:=,=,<>, Indexelés: tomb[index]	FillChar(), SizeOf()

? Halmaz

Típus	*Ábrázolás*	*Muveletek*	*Eljárások, függvények*
set of alaptípus *az alaptípus mérete nem lehet több mint 1 byte*	32 byte *A megfelelo bit 0 ha az elem nincs benne, 1 ha benne van*	[]: üreshalmaz +,-,*,=,<>, <=,>=,in	Insert(), Exclude()

? Állomány

Adatok lineárisan rendezett szekvenciáját ***állomány***nak nevezzük. Minden állomány megnyitás után rendelkezik egy logikai ***állománymutató***val, amely az aktuális bejeg y-

zésre mutat. Az állomány típus valósítja meg a perifériákkal való kapcsolatot. Turbo Pascalban három féle állománytípus ismeretes:

a.) **Szövegállomány (`text`)**

Sorokba rendezett, a sorokat a **CR/LF** karakterek zárják, az állományt pedig **Ctrl-Z**. A hozzáférés szekvenciálisan történik, az írás és az olvasás csak külön-külön történhet.

Pascalban léteznek még **eszközállományok** (input, output, lst), ezek a szövegállomány típus különleges esetei.

Standard eljárások és függvények, amelyek szövegállományokkal kapcsolatos muveleteket valósítanak meg: Eoln(), Eof(), Assign(), Rewrite(), Reset(), Close(), Append(), Write(), Read(), WriteLn(), ReadLn(), Rename(), SetTextBuf(), Erase()

b.) **Típusos állomány (`file of` típus)**

Azonos típusú adatok összessége. Pl. `f: file of integer; g: file of char;` stb. A hozzáférés szekvenciálisan vagy direkt módon is történhet, az adat sorszámának megfeleloen, a számozás 0-tól kezdodik. Az írás és az olvasás váltakozva is történhet. Minden muvelet után az állománymutató elmozdul.

Muveletek: Assign(), Eof(), Rewrite(), Close(), Reset(), Write(), Read(), Seek(), Truncate(), Flush(), Erase(), Rename()

c.) **Típus nélküli állomány (`file`)**

A típus nélküli állomány rögzített hosszúságú bejegyzésekre szervezett, egy bejegyzés hosszát a megnyitásnál kell definiálni (ez a hossz alapértelmezésben 128 byte). A hozzáférés szekvenciális vagy direkt. Az írás és az olvasás váltakozva is történhet. Minden muvelet után az állománymutató elmozdul.

Muveletek: Assign(), Eof(), Reset(), Rewrite(), Close(), BlockWrite(), BlockRead(), Erase(), Rename()

? Rekord

A rekord (`record`) mezokre tagolt adatstruktúra. Lehet rögzített formájú, de lehet változó is. Egy rekordnak csak egy változó része lehet, a rögzített rész mögött. Deklaráció:

```
record
        [mezolista;]
  [case [szelektor:] típus of érték: (mezolista) [;érték: (mezolista);...]]
  end;
```

Pl.
```
type TMyRec = record
       kor: integer;
       név: string;
       case nos: boolean of
       true:(feleség: string; gyerekszám: integer)
  end;
```

A rekord hossza a rögzített rész hossza plusz a legnagyobb változó rész hossza. Hivatkozás a mezokre: **RekordNev.mezo** vagy a `with` utasítás segítségével direkt is lehet hivatkozni: `with` RekordNev `do` mezo `:=` érték`;`

Ha a `case` szelektor nevét nem tüntetjük fel, csak típusát, akkor ennek csak szintaktikai jelentosége van, hivatkozni nem tudunk rá.

Pl.
```
type  TMyCaseRec = record
        kor: integer;
        név: string;
        case boolean of
        true:(feleség: string; gyerekszám: integer)
  end;
```

? Objektum

Ezt a típust az *Objektumorientált programozás* (OOP) című paragrafusban fogjuk részletesen tárgyalni.

Eljárás/Függvény típus

Az eljárás és függvény típusoknak köszönhetoen a Pascal a szubrutinokat olyan programrészekként tudja kezelni, amelyek változóknak megfeleltethetok, paraméterekként átadhatók. A típus deklarálása azonos az azonosító nélküli fejléc megadásával és az ilyen típusú függvények vagy eljárások `far` típusúak kell, hogy legyenek (`{$F+}`).

```
Pl:type
   proc = procedure;
   XProc = procedure(var x,y: byte);
   Func = function(a,b: real): real;
```

Példaprogram, amely paraméterként függvénytípusú változót ad át:

```
{$F+}
program FPTipus;
type      TFuggveny = function(x,y: integer): integer;

procedure Kiir(fugg: TFuggveny; x,y: integer);
begin
   writeln(fugg(x,y):5);
end;
function Osszeg(x,y: integer): integer;
begin
   Osszeg := x + y;
end;

function Szorzat(x,y: integer): integer;
begin
   Szorzat := x * y;
end;

var       Fuggveny: TFuggveny;

begin
   Kiir(Osszeg,2,3);
   Kiir(Szorzat,2,4);
   Fuggveny := Osszeg;
   Kiir(Fuggveny,1,2);
   Fuggveny := Szorzat;
   Kiir(Fuggveny,4,5);
end.
```

Mutató típus

Kétféle lehet: típus nélküli (`pointer`) és típusos - dinamikus (^Típus).
Minden mutatónak 4 byte van fenntartva, ez egy címet tartalmaz.
Pl. `$FFFF:$0000`.
-Muveletek: Ptr(), Addr(), @, Ofs(), Seg()

Dinamikus helyfoglalás

A változókat dinamikusan is kezelhetjük. Ez azt jelenti, hogy a változó nem ***statikusan*** van jelen a memóriában, hanem valamilyen mutató mutat egy ***dinamikusan*** lefoglalt helyre a heap-ben. A **heap** egy, a **stack**-tol független memóriaterület, ahova a Pascal a dinamikusan deklarált változókat tárolja és a **HeapPtr** mutató mutat rá. A statikusan

deklarált változók a **Stack**-ben foglalnak helyet. A heap és a memória méretét a `{$M memóriaméret, heapminimum, heapmaximum}` direktívával állíthatjuk be.

Ha a mutatóra akarunk hivatkozni, akkor ezt az azonosítójával tehetjük meg: **`p :=` kifejezés`;`,** ha pedig arra a memóriaterületre, amelyre mutat, akkor az **azonosító`^`** konstrukciót alkalmazzuk: **`p^ :=` kifejezés`;`**. Nézzük a következo példát:

```
program ByteMutato;

type
    PByte = ^Byte;
var
   a: byte;
   b: PByte;

begin
   a := 10;
   new(b);
   b^ := 10;
   writeln(a);
   writeln(b^);
   dispose(b);
end.
```

Így létrehoztunk egy `byte`-ra mutató típust. A programnak két statikus változója van, amely a **Stack**-ben foglal helyet, ez az `a` byte típusú változó, amely 1 byte-ot foglal le és a `b` mutató típusú változó, amely 4 byte-ot foglal le. Ezeken kívül a `new(b);` programsor végrehajtása után a program dinamikusan lefoglal a **heap**-bol még egy byte-ot, erre mutat a `b` mutató típusú változó. Ez a hely dinamikusan van fenntartva, egy byte – a mutató típus alaptípusa – fér bele és bármikor felszabadítható a `dispose` eljárással. Eddig tehát egy eljáráspárt ismertünk meg: a **new** helyet lefoglal, a **dispose** helyet felszabadít típusos mutatók számára. Még két ilyen eljáráspár van: a **GetMem** és **FreeMem**, illetve a **Mark** és **Release**.

A **GetMem** ugyanúgy muködik mint a **new**, csak típustalan mutatók számára. Mivel a mutatónak nincs egy elore meghatározott típusa, a **GetMem**-nek meg kell mondani, hogy mekkora helyet foglaljon le, a **FreeMem**-nek pedig, hogy mekkora helyet szabadítson fel. A mutatóval végzett muveletek során pedig típuskonverziót kell alkalmazni. Az elobbi program tehát így nézne ki:

```
program ByteMutato;
var
   a: byte;
   b: pointer;
begin
   a := 10;
   GetMem(b,SizeOf(byte));
   byte(b^) := 10;
   writeln(a);
   writeln(byte(b^));
   FreeMem(b,SizeOf(byte));
end.
```

A **Mark** a heap egy bizonyos helyzetét regisztrálja, és a **Release** ezt állítja később vissza, vagyis felszabadít minden, az utolsó **Mark** hívás utáni, dinamikusan lefoglalt változót. A példaprogramunk így módosul:

```
program ByteMutato;
        type
   PByte = ^Byte;
          var
      a: byte;
 b1, b2, b3: PByte;
     p: pointer;
        begin
      a := 10;
      new(b1);
      b1^ := 10;
       Mark(p);
       new(b2);
      b2^ := 20;
       new(b3);
      b3^ := 30;
      writeln(a);
writeln(b1^,' ',b2^,' ',b3^);
       Release(p);
      dispose(b1);
          end.
```

Tehát miután helyet foglaltunk a `b1` mutatónak, regisztráljuk a heap helyzetét a `p` pointerben. A `b2`, `b3` helyfoglalások és a kívánt muveletek elvégzése után visszaállítjuk a regisztrált heapet, ez felszabadítja a `b2`, `b3` helyfoglalásokat, így nekünk csak a `b1` -et kell felszabadítanunk.

Ha meg akarjuk tudni, hogy mennyi szabad memória áll még rendelkezésünkre, akkor a **MaxAvail** függvényt kell alkalmazunk: `if` **MaxAvail < SizeOf(**változó**)** `then` hiba; A **MaxAvail** a legnagyobb szabad blokk méretével tér vissza. Ha pedig a szabadmemória-összméret érdekel, akkor a **MemAvail** függvényt hívjuk. Szintaxisok:

```
procedure New(var p: pointer);
procedure Dispose(var p: pointer);
procedure GetMem(var p: pointer; Size: word);
procedure FreeMem(var p: pointer; Size: word);
procedure Mark(var p: pointer);
procedure Release(var p: pointer);
function MaxAvail: longint;
function MemAvail: longint;
```

Típuskonverzió Pascalban

Két típus ***azonos***, ha ugyanolyan nevueknek deklaráltuk, vagy ha az egyiket a másik nevének felhasználásával deklaráltuk.

Pl. `type`

```
egesz = integer;
int = integer;
```

Ebben az esetben az `egesz`, `int` és `integer` azonos típusok.

Két típus ***kompatibilis***, ha:

- azonosak vagy azonos leírásúak,
- mindketto valós vagy mindketto egész típusú,
- valós típusú változó egész típusú kifejezés által kap értéket,
- egyik a másik részintervalluma,
- kompatibilis típuson alapuló halmaztípusok,
- egyik string, a másik karakter vagy karaktertömb típusú,
- az egyik típusos, a másik típus nélküli mutató,
- ugyanolyan paraméter formátumú eljárás- vagy függvénytípusok.

Egy kifejezés típusa megváltoztatható a **`típusnév`**`(`**`kifejezés`**`);` konverziós muvelettel.

Pl.

```
var
  c: char;
  b: byte;
begin
  c := 'A';
  b := byte(c); {ezután a b értéke 65}
end.
```

C, C++

A C++ programozási nyelvben két nagy alaptípus-osztály van: az aritmetikai és a `void` (nem érdekel a visszatéro típus, típustalan) típusosztályok. Az aritmetikai típus-osztályba tartoznak az egész (`char`, `int`) és a valós (`float`) típusok. A típusok alsó és felso határai a **<limits.h>** header-állományban vannak leírva. A C++ teljesen átveszi a C alaptípusait, ezek mellett új típusok is jelennek meg pl. az objektumosztály (`class`) és a referencia típus (`&`).

A* C *nyelv alaptípusai:

A **C** nem annyira típusorientált nyelv, mint a **Pascal**. Egy pár alaptípust használ, amelyekre építve, a programozó új típusokat hozhat létre. Az alaptípusok azonosítói fenntartott szavak.

Típus	*Ábrázolás*	*Muveletek*
`int`	2 byte `-32768 .. 32767`	`&,|,~,=,<,>,<=,>=,!=,<<,>>,+,-,%,*,/,+=,-=,++,--` stb.
`short`	2 byte `-32768 .. 32767`	`&,|,~,=,<,>,<=,>=,!=,<<,>>,+,-,%,*,/,+=,-=,++,--` stb.
`long`	4 byte -2^{31} .. 2^{31}	`&,|,~,=,<,>,<=,>=,!=,<<,>>,+,-,%,*,/,+=,-=,++,--` stb.
`unsigned`	2 byte `0 .. 65535`	`&,|,~,=,<,>,<=,>=,!=,<<,>>,+,-,%,*,/,+=,-=,++,--` stb.
`char`	1 byte `0 .. 255` vagy `-128 .. 127`	`&,|,~,=,<,>,<=,>=,!=,<<,>>,+,-,%,*,/,+=,-=,++,--` stb.
`float`	4 byte $3.4*10^{-38}$.. $3.4*10^{38}$	`=,<,>,<=,>=,!=,+,-,*,/,+=,-=,++,--` stb.
`double`	8 byte $1.7*10^{-308}$..$1.7*10^{308}$	`=,<,>,<=,>=,!=,+,-,*,/,+=,-=,++,--` stb.
`long double`	10 byte $3.4*10^{-4932}$..$1.1*10^{4932}$	`=,<,>,<=,>=,!=,+,-,*,/,+=,-=,++,--` stb.

A **C** nyelv alaptípusainak azonosítóit, az ortogonalitás szellemében, komplexebb deklarációkra is használhatjuk, az azonosítók egymásután írásával. Például: `unsigned long, unsigned char` stb.

Mutató típus

A mutató olyan típus, amelynek változói értékként egy címet tartalmaznak. A **C** nyelvben, talán az alapértelmezett érték szerinti paraméterátadás miatt, nagyon jól ki van dolgozva a ***pointeraritmetika***.

A mutató (pointer) típust **C**-ben a `*` unáris operátor jelzi. Például, ha egy egész számra mutató pointert akarunk deklarálni, akkor ezt az `int *p;` változódeklarációval tehetjük meg. A `p` pointer típusú változó és az a cím, amely a `p` értéke, egy egész számot tartalmaz. Ha egy pointernek egy változó címét akarjuk megfeleltetni, akkor a cím (unáris `&`) operátort kell használnunk: `p = &i;`.

A `void *` deklaráció explicit típuskonverziót követel és ezt bármilyen mutatótípusra használhatjuk.

Érdekes, hogy hogyan oldja meg a **C** a tömbök kezelését. A tömbváltozó egy mutató, amely a tömb elso elemére mutat. Így egy szoros kapcsolat jön létre a tömbök és a mutatók között, azzal a megkötéssel, hogy míg egy mutató értékét meg lehet változtatni, a tömb mindig az elso elemére mutat, tehát a tömböt egy konstans pointernek tekinthetjük. Például legyen `p` egy mutató és `t` egy tömb. A `p = t` értékadás helyes, és eredményeként a `p` mutató is a `t` tömbre (pontosabban a tömb elso elemére) fog mutatni, viszont a `t = p` értékadás már helytelen, mert a `t` konstans mutató.

A **C** nyelv jól kidolgozott pointeraritmetikájának köszönhetoen mutatókkal számos muvelet végezheto: inkrementálás, dekrementálás, egész kifejezés hozzáadása, illetve kivonása egy pointerbol. Ezek a muveletek a címre hatnak, azokat módosítják, az eredményként kapott pointerek tehát más memóriazónára mutatnak. Mutatók között értelmezett az összehasonlítás muvelet is. A **C** a `NULL` fentartott szóval jelöli azokat a mutatókat, amelyek semmilyen zónára sem referálnak.

Mutatóra mutató mutatót a ** konstrukcióval lehet deklarálni.

Az inkrementálás (++) illetve a dekrementálás (--) muveleteket felhasználva mutatókkal tömbök elemeire is hivatkozhatunk, indexelés([...]) nélkül.

Strukturált típusok

A **C** nyelv strukturált típusai származtatott típusok. Mint már említettük, például a ***tömb*** típus egy konstans pointer, amelyet a **nev**[...] konstrukcióval lehet indexelni. A FILE típus nélküli ***állománytípus*** felhasználói szinten van deklarálva a **<stdio.h>** include állományban. Létezik azonban két elsorendu strukturált típus, a ***struct*** illetve a ***union*** típus. Mindketto a Pascal **record** típusának felel meg. A különbség köztük az, hogy a union típusú változó mezoi ugyanarra a memóriacímre kerülnek, a típus tehát a Pascal **case-record**-jához hasonlóan muködik. A struktúrák mezoire, a Pascalhoz hasonlóan, a **nev.mezo** szerkezettel lehet hivatkozni. Mutatók esetén, hogy elkerüljük a (***nev**).**mezo** hivatkozást, bevezették a -> hivatkozó operátort, amely segítségével **nev** -> **mezo** alakban hivatkozhatunk az adatokra.

Felhasználói szinten megváltoztathatjuk egy deklarált típus nevét, vagy valamilyen néven deklarálhatunk egy új típust. Ezt a typedef fenntartott szóval tehetjük meg.

Például a FILE típus **<stdio.h>** állományban levo deklarációja.:

```
typedef struct{
  short             level;
  unsigned      flags;
  char          fd;
  unsigned char      hold;
  short             bsize;
  unsigned char      *buffer, *curp;
  unsigned      istemp;
  short             token;
} FILE;
```

Típuskonverzió C-ben

C-ben a típuskonverzió implicit és explicit módon valósulhat meg. A **C** automatikus, implicit típuskonverziót használ kifejezésekben szereplo operátorok operandusaira. Nincs az operandusok típusára vonatkozó megkötés. A muvelet végrehajtódik, az eredmény típusát pedig az alábbi szabályok valamelyike határozza meg:

1. a char típusú operandusok int típusúvá konvertálódnak és az eredmény int lesz.
2. ha valamelyik operandus long double, akkor a másik is long double lesz és az eredmény típusa is long double.
3. hasonló a helyzet a double, float, long, unsigned típusokkal is.
4. a char, int és unsigned típusok egymással kompatibilisek.

Az explicit típuskonverziónál felkérjük a fordítóprogramot, hogy konvertálja át a típusokat. Ezt a (**újtípus**) **változó** szerkezettel tehetjük meg.

Kovács Lehel

A zsírok minoségének romlása használat során

A zsírok (vagy lipidek) és az olajok a szénhidrátok mellett táplálkozásunk azon legfontosabb energiatermelo tényezoi, melyeket a motorok muködésénél az üzemanyaghoz hasonlíthatunk. A szervezetben ugyanis ezek az anyagok elbomlanak, a szervezet „feldolgozza” oket, s eközben energiát termel, melyet munkavégzésre használ fel.