

A PROGAMOZÁS ALAPJAI 1

4. előadás

Vitéz András
egyetemi adjunktus
BME Híradástechnikai Tanszék
vitez@hit.bme.hu

2012. február 28.

Miről lesz ma szó?

- Programtervezési stratégiák
  - Top-down
  - Bottom-up
- Programszegmensek
- Függvények
  - Szintaxis (deklaráció - definíció)
  - Formális paraméterek
  - Aktuális paraméterek
  - Élettartam - láthatóság
- A C nyelv típusai
  - Egész (fixpontos ábrázolású) típusok
  - Egészek, mint bitminták tárolói
  - Lebegőpontos típusok
- A számábrázolás pontossága

Programtervezési stratégiák

Top-down tervezés
(lépésenkénti finomítás)

Bottom-up tervezés
(téglánkénti építkezés)

Top-down tervezés

A lépésenkénti finomítás elve:

A feladat megoldását elodázom.
Minden finomítási lépésben olyan építőelemeket használok, amelyek majd egy későbbi lépés eredményeként készülnek el.
Végül már csak elemi tevékenységek maradnak.
Ezeket kell kódolni.

Top-down tervezés

Előnyei:

Lehetőség van a belső megoldások elrejtésére.

Lehetetlen elhibázni.

Bottom-up tervezés

A téglánkénti építkezés elve:

A legegyszerűbbtől kiindulva egyre bonyolultabb eszközöket hozok létre.
Ezeket az újabb szinteken már építő-kockaként használhatom.

Bottom-up tervezés

Látszólagos előnyei:

Lehetőség van ugyanazt az elemet több helyen felhasználni.

Ha mégis hiányzik valami, utólag még meg lehet tervezni.

Programszegmensek

Program

Eljárás

Függvény

Programszegmensek

A C nyelvben minden szegmens

függvény

Programszegmensek

A felhasználás célja:

Program
Mindazon tevékenységek összessége, amit a végre akarunk hajtani.

Programszegmensek

Program megvalósítása:

a **main()** függvény

Aktiválása:

az operációs rendszer hívja

Programszegmensek

A felhasználás célja:

Eljárás
Összetartozó utasítások sorozata, amit többször akarunk végrehajtani.

Programszegmensek

Eljárás megvalósítása:

void típusú vagy fel nem használt visszatérési értékű függvény

Aktiválása:

kifejezés utasítással

Programszegmensek

A felhasználás célja:

Függvény
Ugyanaz, mint az eljárás,
de visszatérési érték is keletkezik.

Programszegmensek

Függvény megvalósítása:

tetszőleges (nem void) visszatérési értékű függvény

Aktiválása:

kifejezésben szerepeltethető

Megjegyzés:
a mellékhatás fontosabb lehet az értéknél!

Függvény definíciója

<visszatérési érték típusa>
<függvény azonosítója>
(<formális paraméterek listája>)
<blokk>

Függvény definíciója

<visszatérési érték típusa>
<függvény azonosítója>
(<formális paraméterek listája>)
<blokk>

- a függvény által kiszámított érték típusa
- alapértelmezésben **int** (ekkor elhagyható)

Függvény definíciója

<visszatérési érték típusa>
<függvény azonosítója>
(<formális paraméterek listája>)
<blokk>

- a függvény **egyedi** azonosító neve

Függvény definíciója

<visszatérési érték típusa>
<függvény azonosítója>
(<formális paraméterek listája>)
<blokk>

- a paraméterek formális deklarációja, hogy a függvényben hivatkozni lehessen rájuk
- egyenként, vesszővel elválasztva
- a zárójelek üres lista esetén is kellenek

Függvény definíciója

<visszatérési érték típusa>
<függvény azonosítója>
(<formális paraméterek listája>)
<blokk>

- **{ }** jelek között deklarációk és utasítások
- ha a típus nem void, **return** utasítás kell

Függvény deklarációja

Később látni fogjuk, hogy a függvény deklarációja elválasztható a definíciótól.

Ha a feljécet különválasztjuk,
a blokk helyére ;-t kell írnunk .

Függvény definíciója

Példa:

```
double tavolsag (double a,double b) {
   if(a>b)
      return a-b;
   return b-a;
}
```


Függvény definíciója

Megjegyzés:

Ha holnap is érteni akarjuk, akkor

```
double tavolsag (
            double a, /* Mi is ez? */
            double b  /*  Hát ez?  */){
   if(a>b)
      return a-b;
   return b-a;
   /* A nagyból vonjuk ki a kicsit */
}
```


Függvény definíciója

Még egy példa:

```
int prim(int p){
    int i;
    for(i=2;i*i<p+1;i++)
        if (p%i==0)
            return 0;
    return 1;
}
```

Aktuális paraméterek

A C nyelvben csak
érték szerinti
paraméter átadás van!

Aktuális paraméterek

A hívás helyén a paramétereket a típusuknak megfelelő kifejezéssel kell aktualizálni.

Példa:

```
c=tavolsag(d+9.7,4.6);
```


Aktuális paraméterek

A paramétereket a függvényben típusuknak megfelelő változókként használhatjuk, melyek a hívás helyén kapott kezdeti értékkel rendelkeznek.

Példa:

```
unsigned factor(int n){
  unsigned f;
  for(f=1;n>1;n--)
    f=f*n;
  return f;
}
```


Aktuális paraméterek

Ha referencia szerinti paraméter átadásra volna szükségünk, akkor formális paraméterként mutatót használunk, amit a hívás helyén a hivatkozni kívánt objektum címével aktualizálunk.

Aktuális paraméterek

Tömb és függvény mutatójának aktualizálásakor csak a nevet kell megadni & operátor nélkül.

Változók élettartama - láthatósága

A formális paraméterek és a függvény blokkjának elején deklarált ú.n. automatikus változók csak a függvény végrehajtásának idején léteznek. Utána megsemmisülnek.

Ezeket a függvény lokális változóinak nevezzük. A függvényen kívülről nem láthatók.

A függvényeken kívül deklarált ú.n. fájl szintű változók a függvény globális változói.
Ha egy globális és egy lokális név megegyezik, a lokális értelmezés az érvényes.

Változók élettartama - láthatósága

A függvény nem látja a többi függvény lokális változóit, akkor sem, ha a másik függvény végrehajtása még folyamatban van.

A függvény a külvilággal a kapcsolatot csak paraméterein és a függvényértéken keresztül tartja. Ha csak nem kimondottan ez a feladata, nem lép interakcióba a felhasználóval.

A függvények közötti kölcsönhatások biztonságos kézben tarthatóságának érdekében kerüljük a globális változók használatát!

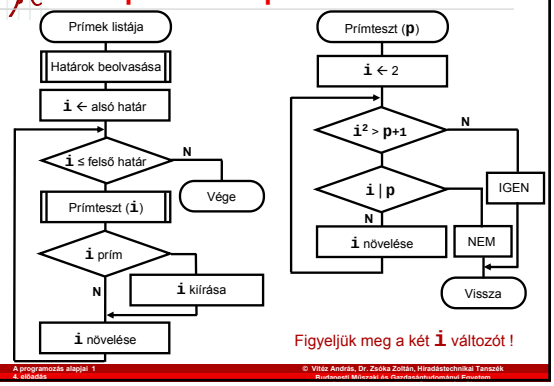
Komplex mintapélda

Írjunk programot két egész szám közé eső prímek listázására!

Olvassuk be az alsó és felső határt!

Lépkedjünk végig a két határ közé eső számokon!
Ha a vizsgált szám prím,
írjuk ki!

Komplex mintapélda



Figyeljük meg a két `i` változót !

Komplex mintapélda

```
#include <stdio.h>
int a,b;
void beolvas(){
    printf("Adj meg egy kisebb és egy nagyobb számot!\n");
    scanf("%d%d",&a,&b);
}
int prim(int p){
    int i;
    for(i=2;i*i<p+1;i++)
        if (p%i==0)
            return 0;
    return 1;
}
int main(){
    int i;
    beolvas();
    printf("Prímszámok %d és %d között:\n",a,b);
    for(i=a;i<b+1;i++)
       if(prim(i))
          printf("%d\n",i);
    return 0;
}
```


A C nyelv típusai

- void
- skalár:
  - aritmetikai:
    - egész:
      - integer
      - karakter
      - felsorolás
    - lebegőpontos
  - mutató
- függvény
- union
- összetett:
  - tömb
  - struktúra

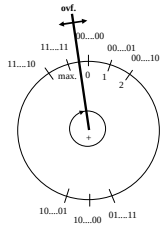
Egész (fixpontos ábrázolású) típusok

A szabvány nem írja elő, de a gyakorlatban

- előjeles egészeket
  kettes komplemens kódban
- előjel nélküli egészeket
  bináris alakban ábrázolunk

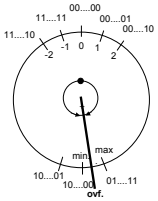
Egész (fixpontos ábrázolású) típusok

Előjel nélküli egészek bináris ábrázolása



Egész (fixpontos ábrázolású) típusok

Előjeles egészek kettes komplemens kódú ábrázolása



Az ábrázolás korlátai

Az adott implementációhoz az értékkészletet
a **<limits.h>** file adja meg.

Az ábrázolás korlátai

Miért kell ismernünk?

Például ki akarjuk számolni $\binom{15}{3} = \frac{15!}{3!\cdot 12!}$ értékét.

Legyen 2-es komplemens számábrázolásunk
4 bájton, ami –2 147 483 648-tól 2 147 483 647-ig
teszi lehetővé a számok ábrázolását.

Az eredmény 455, ami gond nélkül ábrázolható.

Az ábrázolás korlátai

Miért kell ismernünk?

Például ki akarjuk számolni $\binom{15}{3} = \frac{15!}{3!\cdot 12!}$ értékét.

A számláló 1 307 674 368 000 Nem ábrázolható
A nevező 2 874 009 600 Nem ábrázolható

Az ábrázolás korlátai

Miért kell ismernünk?

Például ki akarjuk számolni $\binom{15}{3} = \frac{15!}{3!\cdot 12!}$ értékét.

Ha viszont először egyszerűsítünk

$$\frac{15!}{3!\cdot 12!} = \frac{15\cdot 14\cdot 13}{3!} = \frac{2730}{6}$$

A művelet elvégezhető!

Előjeles integer típusok

	minimális hossz [bit]	típus-megadás alakja	minimális érték <limits.h> -ban	maximális érték <limits.h> -ban
rövid	16	short short int signed short signed short int	SHRT_MIN	SHRT_MAX
normál	16	int signed int	INT_MIN	INT_MAX
hosszú	32	long long int signed long signed long int	LONG_MIN	LONG_MAX

Előjeles integer típusok

Szám-konstans alakja:

Típus	Számrendszer	Példák
int	decimális	**0 123 -33**
	oktális	**012 0177777**
	hexadeimális	**0x1a 0x7fff 0xAa1bB**
long		**-"-L -"-l** vagy, ha az érték olyan nagy

Előjel nélküli integer típusok

	minimális hossz [bit]	típus-megadás alakja	minimális érték	maximális érték <limits.h> -ban
rövid	16	unsigned short unsigned short int	0	USHRT_MAX
normál	16	unsigned unsigned int	0	UINT_MAX
hosszú	32	unsigned long unsigned long int	0	ULONG_MAX

Előjel nélküli integer típusok

Szám-konstans alakja:

Típus	Példák
unsigned	<mintha int lenne>**u** <mintha int lenne>**U**
unsigned long	<mintha int lenne>**lu** <mintha int lenne>**Ul** <mintha int lenne>**Lu** <mintha int lenne>**UL** vagy, ha az érték olyan nagy

Automatikus ábrázolási mód

ha nincs megadva U vagy L

Amelyikbe a szám-konstans előbb belefér:

1. **int**
2. **unsigned int** oktális és hexa esetén
3. **long int**
4. **unsigned long int**

A short típusmódosító

A …**short** típusok csak az adat tárolási hosszát írják elő.

Számításnál a ... **short** típusú értéket automatikusan ... **int** típusúvá alakítja, és azzal számol.

Karakter típusok

Ábrázolás: minimum 8 biten.

Ez lesz a tárolás alapegysége

típus-megadás alakja	minimális érték <limits.h> -ban	maximális érték <limits.h> -ban
signed char	SCHAR_MIN	SCHAR_MAX
unsigned char	0	UCHAR_MAX
char	CHAR_MIN	CHAR_MAX

Karakter típusok

Karakter-konstans alakja:

Egyszerű karakterek

Alakja	Jelentése
'a'	kis a betű
'A'	nagy A betű
':'	kettőspont

Karakter típusok

Karakter-konstans alakja:

Különleges karakterek

Alakja	Jelentése
'\''	aposztróf
'\"'	idézőjel
'\\'	backslash
'\?'	kérdőjel
'\a'	hangjelzés

Karakter típusok

Karakter-konstans alakja:

Különleges karakterek

Alakja	Jelentése
'\n'	új sor
'\f'	lapdobás
'\t'	tabulátor
'\v'	függőleges tabulátor
'\b'	visszatörlés

Karakter típusok

Karakter-konstans alakja:

Karakter megadása kódjával

Alakja	Jelentése
'\0'	nullás kódú karakter
'\10'	oktális szám
'\x10'	hexadecimális szám

Karakter típusok

A …**char** típusok gyakorlatilag
az egészekhez hasonlóan viselkednek.

Számításnál a ... **char** típusú értéket
automatikusan ... **int** típusúvá alakítja,
és azzal számol.

Karakter típusok

Számjegyeket ábrázoló karakterek kódolása

A számjegy értéke és kódja nem ugyanaz!

karakter	értéke	ASCII kódja			
		hexadecimálisan	decimálisan	oktálisan	binárisan
'0'	0	30	48	60	110000
'1'	1	31	49	61	110001
'2'	2	32	50	62	110010
…					
'8'	8	38	56	70	111000
'9'	9	39	57	71	111001

Nem érdemes kódokat megjegyezni!

Az értéket megkapjuk, ha a számjegy kódjából kivonjuk a '0' kódját.

Egészek, mint bitminták tárolói

Egy előjel nélküli egészben tárolt kódot nem kell feltétlenül számként értelmeznünk.

Tetszés szerinti bitmintát előállíthatunk, és manipulálhatunk.

A következő előadáson megismerkedünk az ezt támogató számos hatékony operátorral.

& | << >> ^ ~ !

Lebegőpontos típusok

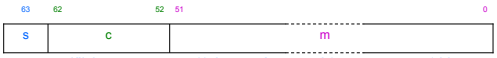
Típus	Konstans alakja	Min. abs. érték	Max. abs. érték	Pontosság [dec. jegy]
float	12.3f 0.12F 12.F .5f 1E-3f 1.8e5f	FLT_MIN <= 1e-37	FLT_MAX >= 1e37	FLT_DIG >= 6
double	12.3 0.12 12. .5 1E-3 1.8e5	DBL_MIN <= 1e-37	DBL_MAX >= 1e37	DBL_DIG >= 10
long double	12.3L 0.12l 12.l .5L 1E-3l 1.8e5L	<= mint double	>= mint double	>= mint double

Az ábrázolás korlátai

Lebegőpontos ábrázolás

Az általunk használt rendszerben a double típus ábrázolása:

előjel-abszolút értékes ábrázolás 64 biten



s	előjel	(1, ha a szám negatív)	1 biten
c	karakterisztika	eltolt zéruspontú ábrázolás	11 biten
m	mantissza	implicit bites ábrázolás	52(+1) biten

$$(1-2s)(1.m\cdot 2^{c-1023}) = (1-2s)(n\cdot 2^{c-1024}),\ \text{ahol}\quad 2 \le n < 4$$

Az ábrázolás korlátai

A véges számábrázolási pontosság következménye

Mivel az ábrázolás matematikai értelemben nem pontos, műveletek eredményét egyenlőségre összehasonlítani

NEM SZABAD

Például:

$$\frac{22}{7}+\frac{3}{7}\neq\frac{25}{7}$$

Az ábrázolás korlátai

A relatív számábrázolási pontosság következménye

Mivel a karakterisztika átfogása nagy, megfeleldkezhetünk a pontosságról.

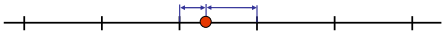
Ha A sokkal nagyobb, mint a és b, akkor

$$(a+A)-(b+A)\neq a-b$$

A számábrázolás pontossága

Egy egész típusban az ábrázolás korlátai között egész számokat teljes pontossággal ábrázolhatunk.

A számegyenes tetszőleges pontját ábrázolhatjuk egész típusban, csak tudomásul kell vennünk, hogy az ábrázolás nem lesz pontos.



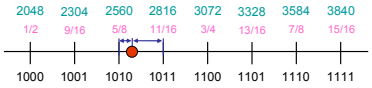
Ha mindig a legközelebbi egész értéket választjuk, az eltérés sohasem lesz nagyobb 0,5-nél.

Azt mondjuk, hogy az egész típus abszolút ábrázolási pontossága 0,5

A számábrázolás pontossága

A véges hosszúságú mantissza miatt a lebegőpontos ábrázolás pontossága sem lesz korlátlan.

A normalizálás miatt a legmagasabb helyiértékű bit és az utolsó ábrázolt bit távolsága rögzített.



Az ábrázolás pontossága ilyenkor az utolsó ábrázolt bit helyiértékének a fele. (példánkban **128** illetve **1/32**)

Ez az érték a szám nagyságrendjétől függ, ezért ekkor nem beszélhetünk abszolút, csak relatív ábrázolási pontosságról.

Azt mondjuk, hogy a lebegőpontos típus ábrázolási pontossága n bit