# 4.
# CGI programozás és PERL

BODÓ ZALÁN
Matematika és Informatika Kar,
BBTE, Kolozsvár

# 1. CGI + C/C++

## CGI programozás

- szerveroldali programozás (pl. CGI, PHP stb.)
  - a szerveren lévő (ált. nagy adathalmazt, adatbázist) éri el könnyebben
- (kliensoldali programozás (pl. Javascript)
  - a böngészőben levő tartalmat/adatokat éri el könnyebben
  - a böngészőn kívüli „teret” nem érheti el

  )
- dinamikus HTML = az oldal tartalma dinamikus (nem statikus), biz. adatoktól függ a kimenet

# CGI

- = Common Gateway Interface (1993)
- a webszerver és a szerveroldali kommunikáció megvalósításának egy módja
- *CGI-szkript*: bármilyen – a szerver által „értelmezhető” – nyelven megírt program
  - a webszerver futtatja a szkriptet
  - általában a „cgi-bin” könyvtárban
  - lehet pl. az összes „.cgi” kiterjesztésű fájl

# Formok küldése

- = adatok átvitele
- GET: az URL-ből (URL felépítése)
  - <URL eleje/szkript neve> ?p1=v1&p2=v2&…&pn=vn
  - (gyakran) URL kódolást használunk
- POST: a kérés tartalmazza a paramétereket; ezek a kérés törzsében lesznek
- általában: POST – permanens változtatás (pl. megváltozik egy lista/tábla tartalma); GET – ellenkező esetben, és ha nem túl hosszú az URL

# URL kódolás

- ASCII (American Standard Code for Information Interchange)
- 7-bites kód: 0-127
  - 0-31 és 127: kontroll karakterek
  - 32-126: nyomtatható karakterek
- URI karakterek:
  - speciális karakterek:
    !*'();:@&=+$,/?#[]
  - hagyományos karakterek:
    ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_.~

- URL kódolás:
  - %-jel
  - két számjegyű hexadecimális szám

Pl.:

! -> %21

* -> %2A

( -> %28

@ -> %40

$ -> %24

á -> %E1

…

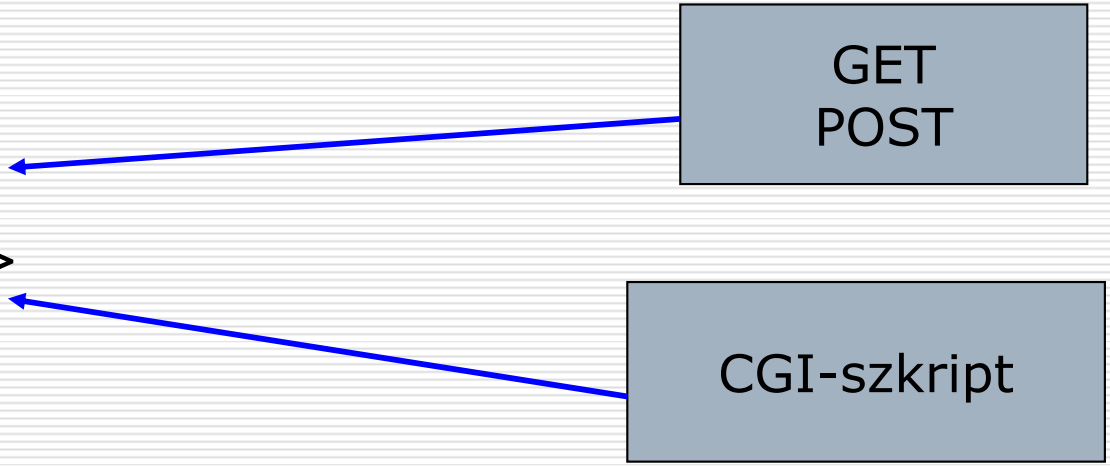
  - átalakítás: URL(x) = % + $ASCII_{16}(x)$

http://www.ascii-code.com/

# HTML formok ismét

- **űrlap**:

```
<form
    method="mod"
    action="cimzett">
...
</form>
```




- **űrlapmezők**:

```
<input name="név" type="típus" align="hely">
<textarea name="név" rows="magaság" cols="szélesség" value=
    "szöveg"> ... </textarea>
<select name="név" size="sor"> ... </select>
...
```

- fejléc küldése: “Content-Type: text/plain;\n\n”
- törzs (adatok) küldése: “Hello, vilag!”
- környezeti változókon és a STDIN-en történő kommunikáció
  - GET: környezeti változók
  - POST: környezeti változók + stdin

- GET: a paramétereket (x=2&y=3) a QUERY_STRING környezeti változóból kérjük le
- POST: a paramétereket (x=2&y=3) a stdin-ről olvassuk le

- Környezeti változók:

SERVER_NAME
REQUEST_METHOD
PATH_INFO
**QUERY_STRING**
REMOTE_HOST
REMOTE_IP
CONTENT_TYPE
**CONTENT_LENGTH**
HTTP_ACCEPT
HTTP_USER_AGENT

# Hello, vilag!

- **Program**:

```
#include <stdio.h>

int main() {
    printf("Content-Type: text/plain;\n\n");
    printf("Hello, vilag!\n");
    return 0;
}
```

www.mingw.org
http://sourceforge.net/projects/mingw/

- Teendők:
  - program lefordítása (ajánlott: MinGW)

    g++ fajl.cpp –o cgi.exe
  - másolás: .../cgi-bin/
  - Apache konfigurálás: .../apache/conf/http.conf
    - Options … ExecCGI (OK)
    - ScriptAlias /cgi-bin/ "…/cgi-bin/" (OK)
    - AddHandler cgi-script **.cgi** .pl .asp **.exe .bin**
  - futtatás: http://localhost/cgi-bin/cgi.exe

127.0.0.1

# 2. Perl [+ CGI]

## A Perl programozási nyelv

- 1987 – Larry Wall
- hasonló a következő nyelvekhez: *C*, *sed*, *awk*
- elterjedt használata: szkriptek készítése (ált. *Linux*) – nagyon erős a *reguláris kifejezések* támogatása, *CGI programozás*
- PERL = ***P****ractical* ***E****xtraction and* ***R****eport* ***L****anguage*

# A legegyszerűbb Perl program

Parancssorból:

perl –e “print ‘Hello, world!’”

- *interpretált* nyelv, de nem *valódi*: bájtkóddá alakítás + optimalizálás + értelmezés (futtatás)

# Adattípusok Perl-ben

- **skalár** ($) Pl. $a = 2; $b = “valami”;
- **tömb** (@) Pl.
  @a = (1, “ketto”, 3.0);
  print $a[0]; $a[1] = “k3tt0”;
- **hash** (%) Pl.
  %a = (“egy” => 1, “ketto” => 2);
  print $a{“egy”};
- **szubrutin** (&) Pl. print &faktorialis(10);
- **globális** (*) – minden más adattípust képvisel; Pl. *lista = *arulista;

# Operátorok

- aritmetikai:
  - hasonló a C-hez
  - + hatványozás: n**k #=n^k
  - + x operátor:

    print “a” x 4; #kiírja, hogy aaaa

    vagy

    my @a = (‘b’) x 3; #@a = (‘b’, ‘b’, ‘b’);

- összehasonító operátorok:
  - >, >=, <, <=, ==, !=, <=>
  - gt, ge, lt, le, eq, ne, cmp
- logikai operátorok: &&, ||, !
  - Pl. print “OK” if $a;
- bitműveletek: &, |, ^, <<, >>
- kötő (binding) operátorok: =~, =!
- tartományoperátor: ..
  - Pl. print (10..20);<br>@a = (1..5);<br>print @b[0..3];
- vessző operátor: ua. mint C-ben
- feltételes operátor: ua. mint C-ben

# Tömbök

- tömbszelet: print @szamok[0,1];
  print @szamok[0..2];
- push: push(@szamok, 4); #utolsó elemként beteszi
- pop: $teto = pop(@szamok); #az utolsó elemet kiveszi
- shift: @szamok = (1,2,3);
  unshift(@szamok, 0); #(0,1,2,3)
  $elso = shift(@szamok); #$elso = 0; @szamok = (1,2,3);
- tömb mérete: $#tomb #=utolsó index

# Asszociatív tömbök (hashek)

- %naplo = (
  Maria => 100,
  Janos => 93,
  Andrea => 88,
  Peter => 97,
  );
- **keys**: kulcsok lekérése:
  print keys(%naplo);
  Pl.:
  foreach $kulcs (keys %naplo) {
  …
  }

- **values**: értékek lekérése

```
@pontok = values(%naplo);
foreach $pont (@pontok) {
    print $pont, “\n”;
}
```

- **each**: bejárás párok lekérdezésével

```
while (($nev, $pont) = each(%naplo)) {
    print “$nev: $pont\n”;
}
```

- **delete**: törlés

```
delete $naplo{“Janos”};
```

# Névtelen és többdimenziós tömbök

- **névtelen tömbök**:

```
$tanulok = [“Maria”, “Janos”, “Andrea”, “Peter”];
hivatkozás:
print $tanulok->[0]; #Maria
print $$tanulok[0]; #Maria
print ${$tanulok}[0]; #Maria
```

- **kétdimenziós tömbök**:

```
@tomb = (
    [“egy”, 2, 3],
    [“negy”, 5, 6, 7],
    [“nyolc”, 9],
);
print $naplo[1][3]; #7
print $naplo[1]->[3]; #7
```

**Feladat**: Írjuk ki a fenti tömb elemeit sorokba rendezve, ahogyan a tömbben szerepel.

```
for (my $i=0; $i<=$#tomb; $i++) {
   for (my $j=0; $j<=$#{$tomb[$i]}; $j++) {
         print $tomb[$i][$j], " ";
   }
   print "\n";
}
```

# Vezérlési szerkezetek

- if-else (C++)
- unless-else (if ellentettje)
- while (C++)
- until (while ellentettje)
  - addig ismétlődik, amíg a feltétel igaz nem lesz
- do-while és do-until (C++)
  - ua. mint a while és until, de egyszer mindenképpen lefut
- for (C++)
- foreach – szintaxis:
  - foreach $x (@lista) {
    …
    }

# Next, redo, last és a SWITCH

- next: a ciklus elejére ugrik
- last: kilép a ciklusból (C++ break)
- redo: ciklusszerkezet végrehajtása a teszfeltétel kiértékelése nélkül
- **switch** szimulálása:

```
SWITCH: {
    if (felt1) { … ; last SWITH; }
    …
    if (feltn) {… ; last SWITCH; }
    #default:
    …
}
```

# Mintaillesztés és reguláris kifejezések

- mintaillesztés: m/regex1/opc
- helyettesítés: s/regex1/regex2/opc
- fordítás: tr/kars1/kars2/opc

opciók: (legfontosabbak)

- g: a minta összes előfordulását figyelembe veszi
- i: nagybetűk és kisbetűk ekvivalensek

- metakarakterek:
  \ a következő karakter levédése
  ^ sor eleje
  . bármely karakterre illeszkedik, kivéve az újsort (\n)
  $ sor vége
  | vagy
  () csoportosítás
  [] karakterosztály
- módosítók:
  * az előző kifejezés zéró- vagy többszöri illesztése (0, 1, 2, ...)
  + az előző kifejezés egy- vagy többszöri illesztése (1, 2, .)
  ? az előző kifejezés zéró- vagy egyszeri illesztése, azaz *opcionalitása* (0, 1)
  {n} az előző kifejezés pontosan n számú illesztése
  {n, } az előző kifejezés n- vagy többszöri illesztése (n, n+1, .)
  {n, m} az előző kifejezés n és m közötti illesztése (n, n+1, ., m-1, m)

- speciális karakterek
  \t *tab* karakter
  \n újsor (*newline*)
  \r kocsivissza karakter (*carriage return*)
  \w szókarakter (alfanumerikus karakterek és a "_")
  \s fehér (*whitespace*) karakter
  \S nem fehér karakter
  \d számjegy
  \D nem számjegy

- **mintaillesztés**:

```
while ($input = <STDIN>){
    if ($input =~ /^vege$/){last;}
    print $input;
}
```

- **helyettesítés**:

```
$s = “abbaaababbba”;
$s =~ s/ab+/c/g;
print $s;
```

- **fordítás**:

```
$s = “12345678”;
$s =~ tr/5678/123/;
print $s;
```

# Fájlműveletek

- beolvasás standard bemenetről:

  $a = <STDIN>;
- beolvasás fájlból:

  $a = <F>;
- fájlok megnyitása/bezárása:

  open(F, “<fajlnev”);

  open(F, “>fajlnev”);

  open(F, “>>fajlnev”);

  …

  close(F);

- kiírás:

  print $a;
  print STDOUT $a;
  print F $a;
  printf F “%d” $a;
- fájlvizsgálatok:
  - -e a fájl létezik
  - -r a fájl olvasható
  - -w írható
  - -z létezik, de mérete 0
  - ...

# Függvények és változók

- sub NÉV {…}
- definició: sub NÉV;
- lokális és lexikus változók:
  - local VÁLT; - megváltoztatja egy (létező) változó értékét
  - my VÁLT; - új (temporális) változót hoz be
- érték visszaadása:

  return $valami;
- aktuális paraméterek lekérése:

  a @_ tömb elemei lesznek sorrendben

# Referenciák és névtelen adatstruktúrák

- **valós referenciák** – a változó az adat címét tartalmazza
- **szimbolikus referenciák** – tetszőleges nevű változó létrehozása

  Pl.:

```
$szam = 20;
$vnev = “szam”;
$$vnev += 3;
print $szam; #ki: 23
```

- valós referenciák:
  - skalárok: **$a = \$b;**
    - visszahivatkozás/dereferencia: **$$a**
  - tömbök: **\@**
    - dereferencia: $a = \@b; **$$a[0]; @$a;**
  - hashek: **\%**
    - dereferencia: $a = \%b; **$$a{“valami”}; %$a;**
  - szubrutionok: **\&**
    - dereferencia: **&$fnev;**

- névtelen adatstruktúrák: nincs nevük, csak egy referencia a struktúrára:
  - tömb/névtelen tömb: (…) és **[…]**
  - hash/névtelen hash: (…) és **{…}**
  - Pl.:

    többdimenziós tömbök (2):
    @a = (["a",2], ["b", 3], …);
    $a = [["a",2], ["b",3], …];
    hashekből álló tömbök:
    @a = ({"a"=>2}, {"b" => 3}, …);
    $a = [{"a"=>2}, {"b" => 3}, …];
    …

# Modulok és OOP

```
package modul;
use Exporter;
@ISA = qw(Exporter);

@EXPORT = qw(v1 v2 v3);
@EXPORT_OK = qw(v4 v5);
```

Exporter osztálytól való származtatás

szimbólumok automatikus exportálása

szimbólumok exportálása kérésre

- konstruktor: sub new { … } (nem kötelező new-nak lennie)
- destruktor: sub DESTROY {…}
- objektum létrehozása: bless:
  - utolsó sor a konstruktorban:
    return bless $self, $class
    ahol:
    $class = első paraméter
    $self = az objektum adattagjai (általában anonim hashek)

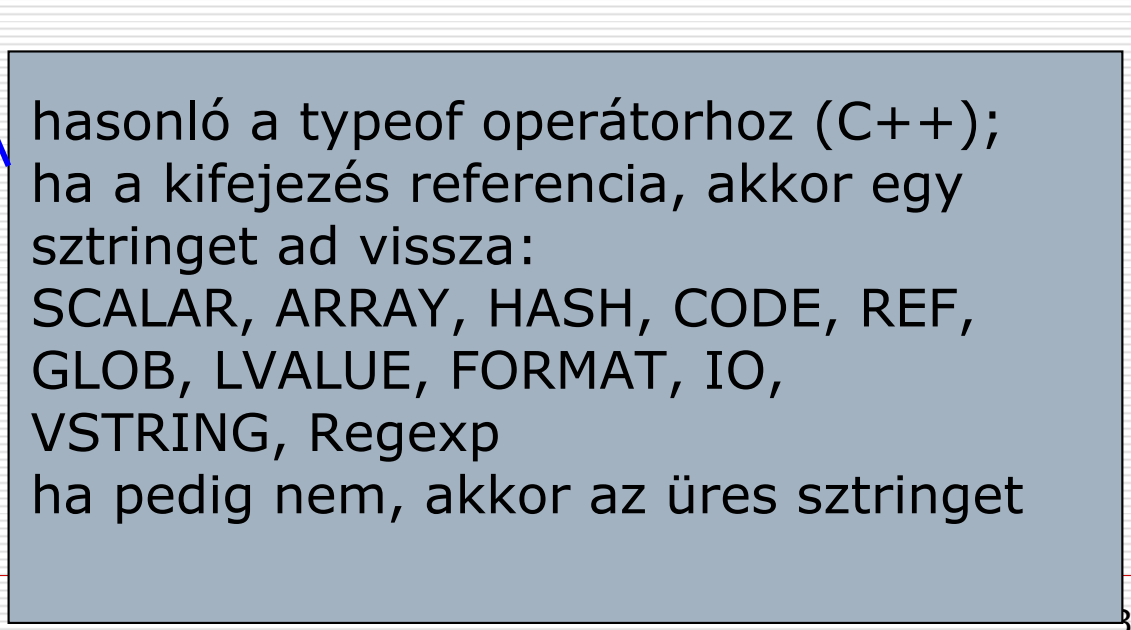
- Pl.

```
sub new {
    my $class = shift;
    my $self = {};
    $$self{“ID”} = undef;
    $$self{“TITLE”} = undef;
    my ($id, $title) = @_;
    $$self{“ID”} = $id;
    $$self{“TITLE”} = $title;
    return bless $self, ref($class) || $class;
}
```

vagy:

```
my %self = ();
$self{“ID”} = undef;
…
return bless \%self,
ref($class) || $class;
```

- objektum létrehozása:
  - my $obj = new OsztNev(…);
  - my $obj = $robj->new(…); #ezért kellett a ref($class)



- öröklődés:

  use Szulo;

  @ISA = (“Szulo”);
- szülő metódus meghívása:

  $self->SUPER::szubrutin(…);

# Perl és CGI

- hasonlóan a C/C++hoz
- javasolt a CGI.pm használata (minden Perl disztribúcióban megtalálható)
- egyszerüsített kezelés:

```
use CGI;
...
my $query = new CGI;
print $query->header(-type=‘text/html’);
…
print $query->server_name, “\n”;
print $query->param(‘valami’);
…
```

- 2 lehetőség:
  - @params = $query->param();
  - $query->param(“valami”);