Zaawansowane metody programowania

Dr Zbigniew Kozioł - wykład
Mgr Mariusz Woźny - laboratorium

Wykład II

Elementy struktury kodu

- - dyrektywy
- - biblioteki i moduły zewnętrzne
- - kompilacja kodu

- - pętle FOR
- - WHILE
- - IF
- - LABEL i GOTO
- - Funkcje i Procedury

Języki programowania

- **assambler** (MOVE, POP; elementarne operacje na bajtach i rejestrach procesora; dla człowieka niemal nieczytelny)
- **C/C++**
- **C#**
- **Fortran**
- **Java** (Sun Microsystems)
- **PHP** (Pretty Home Page, http://www.php.net)
- **Perl** (wspaniały do obróbki tekstów i danych w formacie ASCII) http://www.perl.com
- **Pascal / Free Pascal** http://www.freepascal.org
- **Basic / Free Basic** http://www.freebasic.org
- **Python** http://www.python.org
- **JavaScript**
- **SQL**
- **bash** (Linux i inne systemy uniksowe, do pracy z terminala i dla skryptów systemowych)

Structura kodu programu

Struktura programu (Pascal)

```
PROGRAM dyfuzja; ←-- Na początku nazwa

Nagłówek programu:
Deklaracje bibliotek, stałych, zmiennych, ich typów
Deklaracje funkcji i procedur

BEGIN
… ←-- program główny
END. ←-- END z kropką oznacza koniec programu
```

Nagłówek programu

Nagłówek programu (Pascal)

```
PROGRAM dyfuzja;

{ Podłączamy zewnętrzne biblioteki:}
uses crt, dos, keyboard, math, sysutils;

{ Definiujemy stałe, zmienne i ich typy: }
CONST  maxSTEP=10000;
       kappa=1.5;
       maxXsteps = 1000;
       maxTsteps = 100;
VAR xs, ts: Integer;

{ Definiujemy nowy typ: }
TYPE   data=array[0..maxXsteps,0..maxTsteps] of Double;
Var f: data;
```

Nagłówek programu (Perl)

```
#!/usr/bin/perl ←-- pierwsza linijka kodu
# Znak # jest znakiem komentarza. Jest BARDZO dobrym zwyczajem
# dodawanie w kodzie komentarzy!
# use like this:
# ./derivative.pl < profile_0.200_0005000.dat
# Deklarujemy używanie zewnętrznego modułu (biblioteki):
use Math::Derivative qw(Derivative1 Derivative2);
#           @dydx=Derivative1(\@x,\@y);
#           @d2ydx2=Derivative2(\@x,\@y);
#           @d2ydx2=Derivative2(\@x,\@y,$yp0,$ypn);
# Deklarujemy zmienne @x i @y jako tablice (znak @)
my (@x, @y); # ← Deklaracja my ma specjalne znaczenie w Perl.
# Oznacza zakres "istnienia" danej zmiennej, blok w którym
# jest widziana, scope po angielsku.
my (@row);
```

Nagłówek programu (Python)

```
#!/usr/bin/env python ← pierwsza linijka kodu
# -*- coding: utf-8 -*-

from pylab import *
import numpy

# Python jest bardzo liberalny gdy chodzi o deklarację typów:
a=0.5;
kappa = 1.0;
```

Nagłówek programu (Free Basic)

```
'*****************************************************
'
' Hard-Locker, Transport Measurment Program,
' (c) J. Wróbel 2012, 2013, 2014
'
'*****************************************************
#Include ".\inc\dislin.bi"
#Include "GL\gl.bi"
#Include "crt\stdio.bi"
#Include "fbmath.bi"
#INCLUDE "linux-gpib.bi"
'
' variables and subroutines
'
#Include ".\src\GlobalVars-iv.inc"
#Include ".\src\SetUpWidgets.inc"
#Include ".\src\FileSubs.inc"
```

Definicje funkcji i procedur

Definicja funkcji (Pascal)

```
FUNCTION PWR(x,p: real): Double;
var ab: Double;
BEGIN

    ab :=x;
    IF(x <0) THEN ab := -x;

    IF ab<=1E-12 THEN
        BEGIN
            Pwr:=0; ← wynik zwracany przez funkcję
            exit;
        END
    ELSE
        BEGIN
            pwr := x*exp((p-1)*ln(ab)); ← wynik zwracany przez
funkcję
        END;
END;
```

Definicja Procedury (Pascal)

```
{ W Pascalu, procedury defniuje się tak samo jak i funkcje.
Procedura nie zwraca wyniku operacji.}

Procedure SaveData(x,p: real);
var ab: Double;
BEGIN
   …
END;
```

Definicja funkcji (Python)

```
# W Python struktura programu jest określona poprzez
# formatowanie kodu: ilość znaków TAB lub spacji na początku
# każdej linii. Na początku denerwujące, ale szybko można
# przywyknąć, zaś kod jest łatwy do odczytania

def pwr(x,p):
    ab=abs(x);
    Power=0;  # Pętla if...else
    if (ab<0.000000000001):
        power=0;
    else:
        power= x *pow(ab,p-1);
    return power;
```

Definicja Funkcji lub Procedury (Perl)

```
# W perl nie istnieje różnica między funkcją a procedurą.
#
# Acz też... perl to bardzo zwariowany i magiczny język.

sub xf { # ← znak początku bloku (funkcji)
    my $t = shift; # ← pierwszy parametr funkcji
    my $kappa = shift; # ← drugi parametr funkcji
    my $Xf = 0.897*($t/100000.0) ** (1/($kappa+2));
    return $Xf; # ← wartość jaką zwraca funkcja
} # ← znak końca bloku (funkcji)

# W perl nie istnieje różnica między funkcją a procedurą.
# W perl możemy też zrezygnować z ładnego formatowania. Np.
Kod powyżej można zapisać w jednej linii:

sub xf{my $t=shift;my $kappa=shift;my $Xf=0.897*($t/100000.0)
**(1/($kappa+2));return $Xf;}
```

Pętla FOR (Pascal)

```
FOR ts:=0 to maxTsteps DO
BEGIN
    FOR xs:=0 to maxXsteps DO
    BEGIN
        f[xs,ts]:=0;
    END;
END;
```

Dwie pętle FOR, jedna w drugiej.

Pętla FOR (Perl)

```
my $size = $#x;

for(my $e=0; $e<$size; $e++) {
    $xRen[$e] = $Xf * $x[$e];
}
```

Pętle IF … ELSE

Pętla IF (Perl)

```
if ($i<=10000) {
    $deltaJ = $NewJ[20000 - $i];
}

if ($deltaJ >0 && $i<=10000) {;
    print $NewX[$i], "\t", $NewJ[$i], "\t", $deltaJ, "\n";
} else {
    print $NewX[$i], "\t", $NewJ[$i], "\t", 0, "\n";
}
```

Pętla IF (Free Basic)

```
If idemon=1 Then
    MeasType = "Demo"
End If
```

```
If hand_(ii)=0 Then
       dwgmsg(dv_name(ii)+" not present on bus !")
Else
    swgtxt (id_ltxt(ii,2), Str$(hand_(ii)))
    dv_flag(ii) = 1
    mmm=1
End If
```

Pętla IF (Free Pascal)

```
IF (step=19) OR (step=49) THEN
SaveBeta((step+1)*maxTsteps);
```

Pętla WHILE (Perl)

```
while(my $line = <>) {

        $line =~ s/\n//;
        $line =~ s/\r//g;

        while($line=~ s/^ //){}

        @row = split /\s+/,$line;

        push @x, @row[0];
        push @y, @row[1];
}
```

Pętla WHILE (PHP)

```
$i = 0;
while ($i <= 10) {
    echo $i++;  /* the printed value would be
                   $i before the increment
                   (post-increment) */
}
```

```
# Jaki będzie rezultat zapuszczenia
# następującego skryptu na serwerze?
$i=7;
while(1) {
     $a= PI*$i;
     print $a;
     $a = $a+1;
}
```

Kompilacja

Kompilacja (Pascal)

```
zbych@orion:~/dyfuzja$ fpc dyfuzja.pas
Free Pascal Compiler version 2.6.4 [2014/03/03] for x86_64
Copyright (c) 1993-2014 by Florian Klaempfl and others
Target OS: Linux for x86-64
Compiling dyfuzja.pas
Linking dyfuzja
/usr/bin/ld: warning: link.res contains output sections; did
you forget -T?
116 lines compiled, 0.7 sec
```

Kompilacja (C/C++)

W przypadku dobrze przygotowanego kodu na Linuksie kompilacja jest trywialnie prosta:

```
./configure
make
make install
```

Komenda make może czasem wykonywać się przez długi czas...

Kompilacja: warto dokumentować

```
./configure --prefix=/www/apache
make
make install
/www/apache/bin/apachectl start

apr-util
./configure --with-apr=/usr/local/apr

./configure --prefix=/www/apache --enable-module=so --with-
apr=/usr/local/apr
make
sudo make install

PHP:
./configure --with-apxs2=/www/apache/bin/apxs -l/usr/local/apr
make
sudo make install
cp php.ini-development /usr/local/lib/php.ini
```

A co, gdy są problemy? Praktycznie zawsze są...

```
zbych@orion:~/Hard-Locker-Matrix$ fbc Hard-Locker-Matrix.bas
ld: cannot find -llibgpib
```

1. Google!

2. TLUG

3. Szukać odpowiedniej frazy w dokumentacji

Dalej:

- operacje wejścia/wyjścia
- grep, przekierowywanie danych, perl i bash, gnuplot
- STD INPUT, STD OUTPUT, STD ERROR
- | (pipe)