https://doi.org/1071066/VARECZA.2025.1.03

Dr. Falucskai János

CIKLUSOK OKTATÁSA

TEACHING OF LOOPS

ÖSSZEFOGLALÓ

A ciklusok oktatása fontos területe a programozás oktatásának, tekintettel arra, hogy megkerülhetetlen a képzés folyamatában. Jól ismert eredmény, hogy bármilyen feladat három programozási eszközzel megoldható: ezek a szekvencia, szelekció és az értekezés alapjául szolgáló iteráció, azaz a ciklusok. Az oktatás során figyelmet kell fordítani a fokozatos bevezetésnek, a nem túl gyors előrehaladásnak, valamint annak, hogy igazából elegendő egy ciklus ismerete, a többi csupán a kényelmet fogja szolgálni. Célunk az, hogy bemutassuk egy lehetséges bevezetését és oktatását a ciklusoknak, ami azt jelenti, hogy megmutatjuk az egyetemünkön alkalmazott módszertant. Természetes dolog, hogy más lehetőségek is vannak, a hatékonyságot nem célunk vizsgálni ebben a cikkben. Az ismertetés célja az, hogy mások is megismerhessék a módszerünket, ezzel akár vitát, egyeztetést generáljunk.

Kulcsszavak: programozás oktatása, ciklusok, iterációk

ABSTRACT

Teaching loops is an important area of programming education, given that it is unavoidable in the training process. It is a well-known fact that any task can be solved using three programming tools: sequence, selection, and iteration, which forms the basis of the article. During teaching, attention must be paid to gradual introduction, not progressing too quickly, and the fact that knowledge of one loop actually sufficient, with the others serving only for convenience. Our goal is to present a possible introduction to loops, which means showing the methodology used at our university. Naturally, there are other possibilities, but it is not our goal to examine their effectiveness in this article. The purpose of this presentation is to familiarize others with our method, thereby generating discussion and agreement.

Keywords: programming education, loops, iterations

1. Bevezetés

A programozás oktatása nem egyszerű, ahogy más terület oktatása sem, viszont az egyetemünkön különös hangsúlyt fektetünk arra, hogy az - úgymond - nulla tudással rendelkező hallgatók számára is követhető és megtanulható legyen a programozás. Természetesen, akik úgy jönnek egyetemünkre, hogy birtokában vannak a fontosabb ismereteknek a programozás területén, azok számára unalmasnak tűnhetnek az alapvető algoritmusok ismertetése. Volt, és van erre javaslatom, de a jelenlegi létszámmal (mind a hallgatói, mind az oktatói oldalon) nem kivitelezhető a differenciálás, pedig nagyon szerencsés lenne. Különösebb ismertetés nélkül hivatkozom a BME IMSc programjára, ami megér egy kérdést: Vajon nálunk is működne-e egy olyan modell, ahol az „okosabb”, „motiváltabb” hallgatók egy külön csoportban emeltebb oktatást kapnának, azzal a veszéllyel szembenézve, hogy gyengébb a jegyük? Avagy legyen egy ilyen csoport, s csökkentsük úgy a követelményeket, hogy megkapják ugyanazon jegyet, mintha a gyengébb csoportba tartoznának? Ennek kidolgozása külön vizsgálatot érdemel. Jelenleg nincs differenciálás, minden hallgató ugyanazt a képzést kapja. A fő gond az, hogyha bekerül egy igazán okos/motivált hallgató, akkor látja azt, hogy gyengébb teljesítménnyel is képes elérni a maximális érdemjegyet, idővel csökkenti a teljesítményét, úgymond bele fog

szürkülni a mezőnybe. Meglátásom szerint ezzel küzd az egész felsőoktatás. A kérdés nem az, hogy van-e értelme az elit-képzésnek, csak az, hogyan valósuljon meg.
A „beszürkülés” ellen azzal védekezünk, hogy igyekszünk már a legegyszerűbb esetekben is megmutatni olyan megoldásokat, amire egy általános helyzetben nem gondol a fejlesztő. Sokszor elő fog fordulni a szövegben az „egyszerűbben”, „rövidebben” kifejezés. Ilyenkor arról van szó, hogy megoldunk egy problémát, de igyekszünk a hallgatókat további gondolkozásra ösztönözni. Attól, hogy működik egy megoldás, egy program, még nem biztos, hogy kihasználtuk az összes gondolkozást fejlesztő lehetőséget, ami a feladatban benne rejlik. Szeretnénk őket további kihívások elé állítani, hogy megismerjék önmaguk korlátait, és a kódolásban rejlő további utakat. A „jó” programozó sosem elégedett a saját kódjával, mindig kísérti annak a lehetősége, hogy tudna jobbat írni. Néha előfordul, hogy téved ebben és tökéletes, a lehető legjobb kódot hozta létre. Mindenkit biztatunk arra, hogyha kész egy program, próbálja megírni szebben, rövidebben. A kettő nem ugyanaz…
A programozás oktatása során igyekszünk hangsúlyt fektetni egyrészt az oktatás eredményességére, másrészt az élményszerű tanulásra. Utóbbi számunkra –oktatók- azt jelenti, hogy a hallgatók nem unják az órát, érdeklődnek a feladatok iránt és figyelemmel kísérik magát az órát. Tapasztalatom szerint ez nem egyszerűen értetődik abból a helyzetből, hogy valaki programozással releváns szakos hallgató. Sajnos a modern technológiák megjelenésével a hallgató figyelme könnyen elterelhető, amit igyekszem megértéssel fogadni. Az okos telefonok megjelenésével a hallgatók koncentrációja csökkent, viszont el kell fogadni azt a tényt, hogy ez egy létező dolog. Tudatosan nem problémáról beszélek, ugyanis ez oktatói oldalról is megjelenik, azaz az e-mail-ek, messenger üzenetek nem csak a hallgatók esetén kívánnak azonnali reakciót, hanem többnyire az oktatók oldaláról is. Én személy szerint nem tiltom az az eszközök használatát, viszont kérem, hogy többnyire az órára figyeljenek. Tapasztalataim szerint ez korrekt viszonyt tud indukálni, nem élnek vele vissza a hallgatók, ellenben nincsenek frusztrálva a mobiltelefon által.
Hasonlóan kihívások elé állít a mesterséges intelligencia. Itt nem azzal van gond, hogy a hallgatók mással foglalkoznak, hanem a „királyi” utat választják. Napjainkban teljes bizonyossággal kijelenthető, hogy az MI képes nem csak az egyszerűbb, hanem a bonyolultabb feladatokat is megoldani ebben a témakörben. A helyzet rohamosan változott a múltban, három évvel ezelőtt elegendő volt bonyolultabb feladatot adni, két éve elegendő volt bonyolultabban megfogalmazni a bonyolultabb feladatot, de közel egy éve egyik sem jelent akadályt egyik MI rendszernek sem. Érdekességképpen említem meg, hogy 2024 november elején, amit tévesen oldott meg az MI hétfőn, ugyanazon héten szerdán már helyesen, s a prompt karakterről karakterre megegyezett. A feladat a következő volt: „Kérj be egy pozitív egészszámot, majd írasd ki a törzstényezős felbontását!”
Függvényként nem tudta még megoldani ekkor maradéktalanul helyesen (első híváskor jól működött, a továbbiakban nem), idén, 2025-ben már ezzel is megbirkózott.
Érthető, hogy elakadás esetén használja a hallgató az MI-t, de szerencsés lenne, ha kritikusan és csak ötletmerítésre. Ennek ellenkezőjét tapasztalom, többnyire megelégszenek a megoldás megismerésével, de nem céljuk, hogy utána önállóan lekódolják a kérdéses feladatot. Számonkérésen engedtük az úgynevezett „statikus” Internet használatát régebben: szöveget bevinni csak a „google” keresőbe lehet. Ezt nem tartotta be minden hallgató, emiatt visszatértünk a papíralapú számonkéréshez: tiltjuk az elektronikus eszközök használatát, de papíralapú segédeszköz korlátlanul használható

2. Előzmények

Ahogy az összefoglalóban említettük, bármilyen feladat három programozási eszközzel megoldható: szekvencia, szelekció, iteráció. Ebből adódik, hogy a ciklusok bevezetése előtt foglalkozni kell a másik két eszközzel.

2.1. Szekvencia

Szekvencia alatt az egymás után jövő utasítások végrehajtását értjük. Programnyelv függő az utasítás, például az értékadás egyes nyelveken utasítás, másokban operátor. Közös jellemzőjük, hogy olyan sorrendben hajtódnak végre, ahogy egymásután leírjuk őket a forráskódban. Ahhoz, hogy érdekesebb programokat írjunk, ez ritkán elég. Jellemző megjelenési formája a lineáris program, nincs benne elágazás, ciklus. Bár nagyon hasznos bemutató jelleggel a „Hello World!” program, ez kissé kelti fel a hallgatók és mindenki más érdeklődését.
Elsősorban tisztázni kell a program lényegét: input adatokból output adatok létrehozása. Az adatok valamilyen típusúak, emiatt szükséges legalább egy típus ismerete, kézenfekvő a legegyszerűbb típus, mely egész számot képes tartalmazni. A típushoz szorosan kapcsolódik, hogy miként tudunk ilyen típusú változót létrehozni, esetleg inicializálni, milyen műveleteket tudunk vele végezni, meg kell ismerni az operátorait, pl. összeadás, kivonás, szorzás, osztás, maradékképzés, hatványozás, s még lehetne sorolni a különböző lehetőségeket programnyelvtől függően, de ezek a legfontosabbak. Értéket kell tudni adni egy ilyen típusú változónak, ami lehet közvetlenül egy kifejezés által, vagy más módon, például beolvasással. Oktatjuk a kifejezés fogalmát, az operátorok prioritását, a kiértékelés irányát, a „kerek zárójelet”, mint a prioritást befolyásoló lehetőséget.
Ahogy fentebb említettem, szükség van a billentyűzetről való beolvasás megismerésére, és a teljes élmény megkívánja a képernyőre való kiírást is. A teljesség igénye nélkül az alábbiakban felsoroljuk a C nyelv esetében a fontosabb ismereteket sillabusz formában:

```
int a;
int b,c,d;
int e=23,f=1,g;
b=34;
a=a+2;
d=f=45;
Kifejezés: amit ki tudunk számolni, pl:
((int)sin(2)*56+fgv(14)%56-b)
+ - * / %
9/2->4                9%2->1
scanf(”%d”,&a);
scanf(”%d alma %d”,&c,&d);
printf(”%d”,a);
printf(”A szam erteke: %d egyeb:%d”,a,4*b);
printf(”Alma\nKorte”);
\t \”….
printf(”%3d”,g);
```

Ennyi ismerettel már jó néhány „izgalmasabb” programot meg tudunk írni, pl:

- Kérj be egy egész számot, írasd ki a kétszeresét!
- Kérd be egy téglalap oldalait, írasd ki a területét, kerületét!
- Kérj be egy számot (min 16, max 31), írasd ki a bináris átváltásnál kapott sorokat!

- Kérj be egy számot (min 16, max 31), írasd ki a bináris alakját balról jobbra helyesen!
- Cseréld fel két egész változó értékét segédváltozó nélkül!
- Írj programot, mellyel kideríthető az egész típusú legnagyobb és legkisebb szám (többször lehet módosítani és futtatni a kódot).
- Kérj be egy pénzösszeget, fizesd ki minimális db címlettel!

2.2. Szelekció

C típusú nyelvek esetén tovább bővíti a lehetőségeinket a szelekciós (három operandusú) operátor megismerése és használata, de ez nem minden esetben alkalmas igazi szelekcióra, viszont használatához feltétlenül szükség van a logikai típus és operátorainak ismeretére. A szelekció segítségével már nem csak lineáris programokat írhatunk, egy döntés alapján más, és más hajtódhat végre. Amennyiben a programozási nyelv tartalmaz közvetlen vezérlésátadási lehetőséget (általában `goto/GOTO`), ciklusokra sincs szükségünk, viszont ebben az esetben sajnos átláthatatlan, nehezen kezelhető és fejleszthető forráskódot kaphatunk. Ilyen volt régen a BASIC nyelvben írt programok többsége, mivel csak számlálóciklust tartalmazott a nyelv, s csupán a ciklusváltozó cikluson belüli átírásával lehetett tetszőleges ciklust létrehozni, ami sok hibára adott lehetőséget. Alternatívát jelentett a `GOSUB` utasítás, mely szubrutint tudott hívni, ami a `RETURN` után visszatért a hívás utáni sorban, de ez sem oldotta meg a program olvashatóságát. Ugyan a legtöbb nyelv tartalmaz a közvetlen vezérlés átadásra lehetőséget (`goto, continue, exit, break`) meglátásunk szerint nem szabad oktatni, főleg nem gyakoroltatni, mivel erősen befolyásolja a strukturális programozáshoz szükséges képességek kialakulását. Ebben egy kivételt támogatunk a `switch` szelekció miatt, a `break` utasítást. A teljesség igénye nélkül az alábbiakban felsoroljuk a C jellegű nyelvek esetében a fontosabb ismereteket a szelekciókhoz kapcsolódóan szintén sillabusz formájában, nem törekedve a pontos szintaktikára minden nyelvnek megfelelően:

```
kif1?kif2:kif3
nulla: hamis
nem nulla: igaz
int a;
a=0?13:6;
== != ! < <= > >=
! && ||
if (l.kif) utasitas;
if (l.kif) utasitas1; else utasitas2;
{ u1;u2;…. ;}
switch (szelektor kif.){
case eset11:case eset12:…case eset1n: [ut11;[…;[ut1m;]]][break;]
…
case esetn1:case esetn2:…case esetnn: [utn1;[…;[utnm;]]][break;]
[default: [utm1;[…;[utmn;]]]
}
```

Az ezzel járó feladatok, melyek némelyike megoldható háromoperandusú operátorral is:

- Kérd be a saját és a főnököd fizetését, majd háromoperandusú operátort használva írasd ki a véleményed!
- Kérj be három számot, írasd ki a legnagyobbat!
- Kérd be egy háromszög oldalait, írasd ki, hogy derékszögű-e!

- Kérj be három egész számot, írasd ki, hány darab egyforma van köztük!

3. Ciklusok

Amennyiben túljutottunk a szekvencia és a szelekció oktatásán, sokkal többrétűbb feladatokat tudunk megoldani. A tapasztalatunk szerint jóval nagyobb ugrást jelent a hallgatók számára ez az új algoritmizálási eszköz, mint az eddigiek, emiatt óvatosabb, lassabb bevezetést igényel. Az eredetileg meglévő feladatsorunkat kénytelenek voltunk megváltoztatni, s egy lényegesen egyszerűbbel helyettesíteni, ami kiállta az idők próbáját, hasznosnak bizonyult.
Elsődlegesen (C nyelven) a `while` ciklust vezetjük be, s ezzel oldunk meg minden feladatot. Választását az indokolja, hogy számláló ciklusoknál kénytelen az alkalmazó programozó/hallgató mind a kezdőérték beállítására, mind a számláló változtatására figyelmet fordítani a határérték ellenőrzésén kívül. Ez egyértelműbb a `for` ciklus esetén, ott nem merülhetnek fel ezek a hibák ilyen hangsúllyal. A sillabusz nem bonyolult:

```
while (l.kif) utasitas;
```

Ettől függetlenül már rengeteg gondolkozásra késztető feladat adható, a következő egyszerűbb feladatokkal kezdjük:

- Írasd ki a számokat az [1;100] intervallumban!
- Írasd ki a számokat a [200;100] intervallumban!
- Írasd ki a hárommal osztható számokat az [1000;600] intervallumban!
- Írasd ki a hárommal és héttel osztható számokat az [1;1000] intervallumban!
- Írasd ki azokat a számokat az [1;1000] intervallumban, melyeknek jobbról a második számjegyük 7!
- Kérj be egy számot, írasd ki az osztóit!
- Kérj be egy számot, írasd ki, hogy hány db osztója van!
- Kérj be egy számot, írasd ki az osztói összegét!
- Kérj be egy számot, írasd ki a számjegyei összegét!
- Írasd ki 100-ig a prímszámokat!

A feladatok során igyekeztünk cserélgetni a növekvő és csökkentő számláló ciklusokat, valamint gondolkoztató feladatokat is beilleszteni (például, ha hárommal és héttel osztható, akkor huszoneggyel, s elegendő minden huszonegyediket kiírni, tehát huszoneggyel lehet növelni a számlálót, nem kell folyton vizsgálni). A feladatsor utolsóelőtti feladata nem számláló ciklust kíván.

Ezt követik a több gondolkodást igénylő példák, ahol a karakter típus ismeretét is elvárjuk:

- Kérj be két karaktert, majd írasd ki az általuk meghatározott zárt intervallumban lévő karaktereket. Figyelj arra, hogy [b;f] más eredményt ad, mint az [f;b]!
- Kérj be három számot, írasd ki a legnagyobbat!
- Írasd ki nullától ezerig azokat a prímszámokat, melyekben a számjegyek összege nyolc!
- Írasd ki százig a bővelkedő számokat!
- Írasd ki táblázatos alakban az összes kétbetűs szót, mely előállhat az angol kisbetűs ábécé első tíz betűjéből!
- Kérd be egy háromszög oldalait, írasd ki, hogy derékszögű-e!
- Írasd ki ezerig azokat a számokat, melyek utolsó két számjegye prímszámot alkot!
- Írasd ki százig azokat a számokat, melyeknek páratlan számú osztója van!

- Írasd ki száztól ezerig azokat a számokat, melyek utolsó két számjegye által alkotott szám az első két számjegy által alkotott szám fele!
- Írasd ki százig azokat a számokat, melyek fele nem prímszám!
- Kérj be három egész számot, írasd ki, hány darab egyforma van köztük!
- Írasd ki ezerig azokat a háromjegyű számokat, melyek középső számjegye kisebb, mint az első számjegy!
- Kérj be egy számot, s írasd ki a tőle nagyobb négyzetszámok közül a legkisebbet!

Ugyanezen feladatokat oldjuk meg később az újabb ciklusok használatával. Bevezetjük a `do while` és a `for` ciklusokat:

```
do ut; while(l.kif);
for(kif1;l.kif;kif2) ut;
```

A megoldandó feladatok körében újdonságot nem hoznak, inkább könnyebbséget. Azért kerül későbbre a bevezetésük, hogy a hallgatók gondolkozása ne „kényelmesedjen” el. A továbbiakban a megismert ciklusok felváltva való alkalmazását erőltetjük, akár keverve is. Célunk az, hogy a hallgató felismerje, bármelyik ciklus ismeretében bármilyen feladatot meg tud oldani, amennyiben az ciklussal megoldható (C nyelven).
Végezetül arra szoktuk felhívni a figyelmet, hogy a szokatlan ciklusoknak is van létjogosultsága, különösebb értelmezés nélkül legyen itt egy erre szolgáló példa: `for(;;);`

Felhasznált szakirodalom

- B. W. Kernighan – D. M. Ritchie (1978):The C Programming Language, Second Edition, Original English language edition published by Copyright © 1988, 1978 by Bell Telephone Laboratories, Incorporated

További források

- Budapesti Műszaki és Gazdaságtudományi Egyetem, Villamosmérnöki és Informatikai kar weboldala URL: https://imsc.vik.bme.hu/

SZERZŐI ADATOK

Dr. Falucskai János PhD
Főiskolai docens
Nyíregyházi Egyetem
Matematika és Informatika Intézet
falucskai.janos@nye.hu