Hogyan bírjuk gondolkodásra hallgatóinkat, miközben segítjük is munkájukat? Dokumentáció alapú programfejlesztés

Menyhárt László – Dr Pap Gáborné

{menyhart, papne}@inf.elte.hu
ELTE IK

Absztrakt. Az Egyetemi képzésben az informatika szakosok számára a programozás oktatásának a teljes programkészítési folyamatot fel kell ölelnie. Tapasztalataink azt mutatják, hogy a hallgatók többségét ebből a folyamatból egyedül az érdekli, hogy a feladat megfogalmazásából kiindulva egy jól-rosszul összeállított, de működő kódot elő tudjon állítani. Jelen cikkünkben egy olyan törekvést, próbálkozást igyekszünk bemutatni, ami a fenn vázolt problémának a feloldásában kíván segítséget nyújtani. Próbáljuk a hallgató munkájában legfontosabbként a specifikáció és az algoritmus elkészítését kiemelni, a többi lépéshez pedig segítséget nyújtani. A segítség lényege egy algoritmusból kiinduló automatikus kódgenerálás, továbbá, ha egy előkészített Excel táblában a dokumentáció legfontosabb elemeit rögzíti a hallgató, akkor a program használati leírásának kivételével a teljes dokumentáció automatikus generálása.

1 Bevezetés

Az Eötvös Loránd Tudományegyetemen a Programozó Informatikusok és Informatika tanárok képzésében a programozás oktatásának alapozó tárgyában a cél nem egy programozási nyelv, hanem a programozás módszertanának megtanítása. Ez alatt a teljes programkészítési folyamatot értjük. A feladat megfogalmazásától kezdve a specifikáción és algoritmuson át a kódolás, tesztelés, hibakeresés lépésein keresztül egy működő és jól dokumentált program előállítása a cél. [3], [4]

Otthonra kiadott feladatainkban (beadandók) megköveteljük ugyan a teljes dokumentáció, ezen belül a specifikáció és algoritmus elkészítését is, de ezek megírása érdekesen zajlik. Először is a specifikációk (ha egyáltalán vannak) nagyon hiányosak. Az algoritmusokról látszik, hogy valamilyen módon a működő kódot próbálják visszafejteni. A dokumentáció pedig a kapott mintadokumentáció felületes átszerkesztése, sokszor benne hagyva a mintabeli feladathoz kötődő megfogalmazásokat.

Ezek a tapasztalatok adták a cikkünk ötletét. A programozási alapismeretek tantárgy keretében a hallgatók dolgozatokat is írnak, program és teljes dokumentáció készítését elváró beadandó feladatokat is készítenek. Úgy gondoltuk, hogy ha megkönnyítjük a munkájukat azzal, hogy kevesebb időt kell a kódolásra és a formázott dokumentáció elkészítésére fordítaniuk, akkor több energiájuk marad a gondolkodás igényesebb munka elvégzésére, nevezetesen a specifikáció, algoritmus és a jó tesztesetek elkészítésére.

Voltak már korábban is olyan próbálkozások, hogy a hallgatókat rávezessük arra, hogy a feladat megoldása nem a kódolással kezdődik, hanem a feladat alapos átgondolásával. A hangsúlyt a specifikációra helyezték és egy specifikációs nyelvet definiáltak, amivel tömören le lehet írni a felismert programozási tételek konkrét feladatra illesztését. A specifikációból aztán egy értelmező segítségével végre is hajtották a programot, tehát a kódolási lépés (sőt az algoritmuskészítés is) ebben az elképzelésben is kimaradt. [5]

Mi a hangsúlyt az algoritmusra helyezzük, ebből készül el a működő kód. Természetesen a specifikáció sem maradhat el, hiszen az adatok hozzáférésének (be-, kimenő adatok) megállapítása és a teljes dokumentáció elkészítése sem lehetséges enélkül.

2 ProgAlapCppVarázsló

Ez az új alkalmazáscsomag próbál segítséget nyújtani a hallgatók számára, hogy könnyebben el tudják készíteni a beadandó vagy házi feladatukat és akár az összes probléma-megoldásuk vonalvezetőjeként szolgálhat.

2.1 Első ránézés

Az [1] URL-ről letöltött és kicsomagolt könyvtárban lévő fájlok négy csoportba sorolhatóak.

```
ProgAlapCppVarazslo_v1.0.20121017:
|   algoritmus_HU_20121017.xls (2)
|   general.bat (3)
|   sablon.doc (4)
|   XLS2CppConverter.jar (3)
|   _!_feladatmegoldas_algoritmusa.txt (1)
|   _!_Olvass_el.pdf (1)
|
|---lib (3)
|       commons-beanutils-1.8.3.jar (3)
|       commons-beanutils-bean-collections-1.8.3.jar (3)
|       commons-beanutils-core-1.8.3.jar (3)
|       commons-collections-3.2.1.jar (3)
|       commons-digester-2.1.jar (3)
|       commons-io-2.4.jar (3)
|       commons-jexl-2.1.1.jar (3)
|       commons-logging-1.1.1.jar (3)
|       dom4j-1.7-20060614.jar (3)
|       jxls-core-1.0.jar (3)
|       poi-3.8-20120326.jar (3)
|       poi-ooxml-3.8-20120326.jar (3)
|       poi-ooxml-schemas-3.8-20120326.jar (3)
|       xmlbeans-2.4.0.jar (3)
|
|---sablon (3)
        generated.h (3)
        main.cpp (3)
        PROJECTNAME.cbp (3)
```

A varázsló használatát segíti a rövid ismertetés (`_!_Olvass_el.pdf`) és egy „algoritmus leíró nyelven” elkészített szöveges állomány (`_!_feladatmegoldas_algoritmusa.txt`).

Első lépésként az Excel fájlt kell megnyitni (`algoritmus_HU_20121017.xls`), mely a megoldandó probléma analizálását segíti elő és ebbe a megoldás algoritmusát is el lehet készíteni algoritmus leíró nyelven.

Ezt követően (ha elkészültünk az analizálással és algoritmuskészítéssel és kiléptünk az Excelből) `C++` kódot tudunk generálni az algoritmusból. Ehhez meg kell hívni a `general.bat` -ot két paraméterrel, ahol az első paraméter az Excel fájl neve, a második pedig egy könyvtár elérési útja, ahova a generált `Code::Blocks` projekt létrejöhet. Az ebben található forráskódokat igény esetén módosíthatjuk, sőt a bemenő adatok konstansként történő megadása vagy beolvasásának implementálása szükséges is.

Az újonnan keletkező könyvtárban `algoritmus.xls` néven létrejön egy új Excel fájl, melynek Alap nevű lapján a „Dokumentum kitöltése” gomb megnyomása után a `sablon.doc` -ot alapul véve elkészül a dokumentáció első verziója, melyet testre kell szabni. A makrók használatát engedélyezni kell!

Az egy hónapos fejlesztés után, a tanulmány jelen fázisában a típusok és az algoritmusban definiálható különböző kifejezések kezelése még eléggé limitált és további fejlesztést igényel. Így ezeket a hiányosságokat nem hibának, hanem szükséges fejlesztési lépéseknek tekintjük. Az előforduló szükségletek listáját szívesen fogadjuk, hogy az igényeknek megfelelően folytathassuk a fejlesztést. Most első lépésben az ötlet támogatottságát és a megvalósíthatóság felmérését próbáljuk elvégezni.

2.2 A használat lépései részletesebben

A `_!_feladatmegoldas_algoritmusa.txt` nevű fájlban olvasható, hogy milyen lépésekkel kell haladni a feladat értelmezésétől a dokumentáció elkészítéséig.

Az egész feladatmegoldást addig folytatjuk, ismételjük a benne lévő lépéseket, amíg nem tartjuk eléggé kielégítőnek a készültségét, vagyis a kódot illetve a dokumentációt.

Első lépés a feladat értelmezése, vagyis a specifikálás, a második lépés pedig az algoritmus elkészítése. Ehhez segítségül adunk a hallgatóknak egy Excel sablont. Ennek részletesebb leírása a következő, 2.3-as fejezetben olvasható. Miután a hallgató kitöltötte a nyolc munkalapból az első hatot, már átfogó képpel kell rendelkeznie a problémáról és annak megoldásáról.

Ekkor eldöntheti, hogy saját maga lekódolja az algoritmust vagy használja a segítségül kapott generátort. Ennek leírása a 2.4-es fejezetben olvasható. A saját maga előállította kód vagy a generált kód testre szabása után ismét vissza lehet térni az Excel-hez és kitöltheti a maradék két munkalapra a hiányzó információkat.

Ezt követően a dokumentáció előállítását is segítjük egy előre elkészített sablonnal, és az Excelben lévő információk átmásolásával, hogy ne kelljen duplamunkát végezniük. Végül a dokumentáció testreszabása következik, hiszen a futtatási információkat, képernyőképeket, a legvégén a tartalomjegyzék frissítését nem tudjuk automatizálni.

2.3 Az Excel tábla részletesebb bemutatása

A munkafüzet nyolc munkalapján kell megadni az egyes témakörökhöz az információkat.

Az `Alap` nevű lapon találhatóak a feladatról, környezetről, készítőről és a dokumentációról ismert információk.

A feladat	
Címe	Páros számok összege
Rövid azonosító	parosak
Leírás	A feladat az, hogy olvassunk be egy számot (n), és adjuk össze a nulla és n közötti páros számokat.
Környezet	
Operációs rendszer	Windows 7
Fejlesztő környezet	CodeBlocks
Szerző	
Név	Minta Attila
Azonosító	MIAFENE.ELTE
Drótposta-cím	miafene@elte.hu
Kurzuskód	IP-01PAEG/1
Gyakorlatvezető neve	Vezér Gyula
Feladatsorszám	42
Dokumentáció	
Fájl	miafene_42_parosak

1. ábra: Az `Alap` munkalapon lévő információk

A feladatról meg kell adni a címét, rövid azonosítóját és a leírását vagyis a feladat szövegét. A rövid azonosítóból képződnek a gépen fájlnevek, így itt érdemes betartani, hogy csupa kisbetűből és számokból álljon.

A környezet alatt az operációs rendszer információit és a fejlesztő környezetet értjük.

A készítőről a nevét, azonosítóját és elérhetőségét lehet megadni.

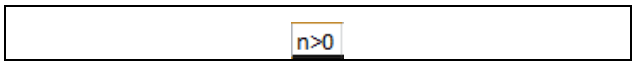
A dokumentációhoz pedig a generálandó dokumentum fájl azonosítóját tárolhatjuk, ahol ismét érdemes betartani, hogy csupa kisbetűből és számokból álljon.

A feladat megértését segíti a következő négy munkalap, melyek a specifikáció négy részének felelnek meg. A `Bemenet`-en a bemeneti adatokat tartalmazó változó neveket és azok típusait kell megadni. Felkészültünk, hogy kezdőértéket is definiálni lehessen és jelezhető, hogy konstansról van-e szó.

Név	Típus	Kezdőérték	Konstans
n	Egész		
a	Szöveg		

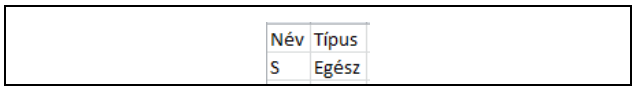
2. ábra: A `Bemenet` munkalapon lévő adatok

Az `Elofeltetel` nevű lapon az első oszlop egyes soraiba írhatjuk a bemeneti adatokat megszorító információkat. Ezt a generátor nem veszi figyelembe, ezért a kód testre szabásakor erre figyelni kell. Viszont a dokumentációba be fog kerülni.



3. ábra: Az `Elofeltetel` munkalap

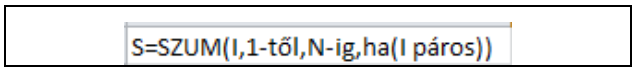
A `Kimenet` nevű lapon a kimeneti változók és azok típusai adhatóak meg.



Név	Típus
S	Egész

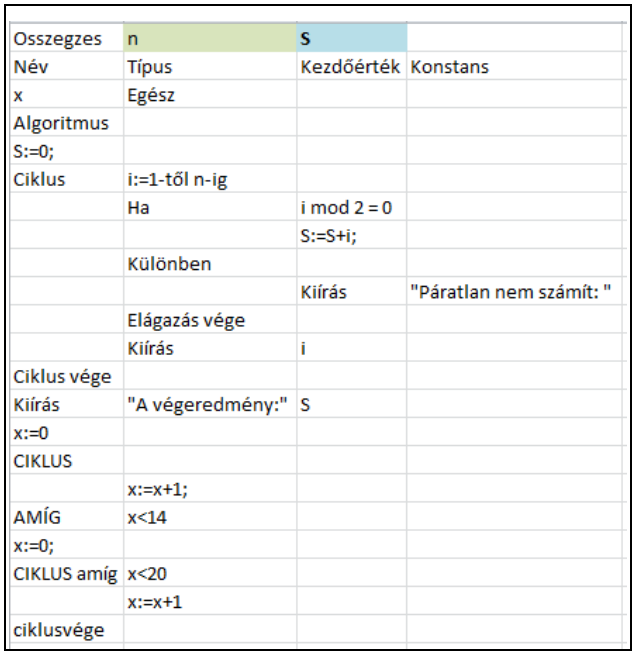
4. ábra: A `Kimenet` munkalap

Az `Utofeltetel` lap első oszlopának egyes sorai fogják tartalmazni azokat az információkat, melyek definiálják a kimeneti változókra érvényes, az algoritmus futása után igaz megállapításokat. Ezt sem veszi figyelembe a generátor, az algoritmusnak kell gondoskodnia erről. Az itteni sorok is bekerülnek a dokumentációba.



5. ábra: Az `Utofeltetel` munkalap

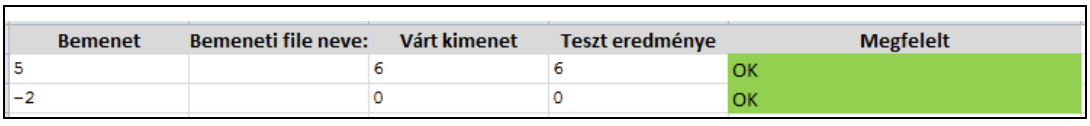
A következő `Algoritmus` nevű lapon kell definiálni a generálandó eljárás nevét, paramétereit a 6. ábrán látható módon. Egy paraméter neve egy cella, a típusát és hogy bemeneti vagy kimeneti adat nem kell megadni, ezt a generátor a specifikációból, vagyis a `Bemenet` illetve `Kimenet` lapokról felismeri. Magunknak jelölhetjük akár színekkel, hogy melyik a bemeneti és melyek a kimeneti változók, például zöld (`green, [gri:n], INput`) és kék (`blue, [blu:], oUtput`) színekkel. Szükség esetén a segédváltozókat is meg lehet adni, ugyanúgy mint a `Bemenet` lapon. Ezt követi maga az algoritmus. A jobb olvashatóság és a generálás helyes működése miatt a behúzások kezelése kötelező! Például „ciklus vége” információt nem szükséges megadni, mert a behúzásból egyértelműen kiderül, hogy a következő parancs nem a cikluson belül van, hanem azzal egy szinten, vagyis le kell zárni előtte a ciklust. Ezt a generátor meg is teszi.



Osszegzes	n	S	
Név	Típus	Kezdőérték	Konstans
x	Egész		
Algoritmus			
S:=0;			
Ciklus	i:=1-től n-ig		
	Ha	i mod 2 = 0	
		S:=S+i;	
	Különben		
		Kiírás	"Páratlan nem számít: "
	Elágazás vége		
	Kiírás	i	
Ciklus vége			
Kiírás	"A végeredmény:"	S	
x:=0			
CIKLUS			
	x:=x+1;		
AMÍG	x<14		
x:=0;			
CIKLUS amíg	x<20		
	x:=x+1		
ciklusvége			

6. ábra: Az `Algoritmus` munkalap

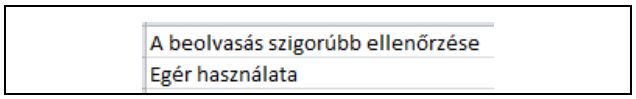
A `Teszteles` nevű lapra írhatja a felhasználó a tesztjeinek az eredményeit, melyek ugyancsak a dokumentáció részét fogják képezni.



Bemenet	Bemeneti file neve:	Várt kimenet	Teszt eredménye	Megfelelt
5		6	6	OK
-2		0	0	OK

7. ábra: A `Teszteles` munkalap

Végül a `Fejlesztes` lap első oszlopának soraiba lehet írni a fejlesztési lehetőségeket.



8. ábra: A `Fejlesztes` munkalap

2.4 A generátor és a generált kód leírása

A generátort a `general.bat` meghívásával lehet elindítani. Két paramétert kell neki átadni. Az első paraméter az Excel fájl neve, a második pedig az az alapkönyvtár, ahova a generált `Code::Blocks` projekt fog létrejönni.

Ennek a megvalósítása most Java-ban készült el és az Apache Group POI projektjében [7] elkészült függvénykönyvtárat használja. Ezek a szükséges függvénykönyvtárak a lib könyvtárban találhatóak.

Bemenetként a kitöltött Excel-t és a sablon könyvtárban lévő előkészített `Code::Blocks` projektet használja. A második paraméterben megkapott könyvtárba hoz létre egy új könyvtárat az Excel Alap munkalapján található rövid azonosító és az időbélyegből képzett néven. Így verziózza a generált forrásokat.

Az azonosító lesz a projekt neve is és így jön létre a `Code::Block` projekt fájl (`cbp`) is. Ugyanígy legenerálásra kerül a forráskód is.

```cpp
#include <iostream>
#include "generated.h"
#include <stdlib.h>
using namespace std;
int main()
{
    setlocale(LC_ALL,"");
    cout << "Páros számok összege" << endl;
    //Valtozok deklaralasa
        //Bemenet
    int n;
    string a;
        //Kimenet
    int S;
    //TODO beoolvasas, ha kell
        //0. kozelites: konstans adat(ok)
        cout << "A beolvasas HIANYZIK!";
        exit(1);
        //1. kozelites: egyszeru beolvasas
        //2. kozelites: beolvasas tipus ellenorzessel
        //3. kozelites: beolvasas tipus es elofeltetel ellenorzessel
    //Algoritmus lekodolasa
    Osszegzes(n,S);
    //TODO kiiras, ha kell
    return 0;
}
```

```cpp
#include <iostream>
using namespace std;
void Osszegzes(int n,int & S){
    int x;
    //S:=0;
    S=0;
    //Ciklus i:=1-től n-ig
     for ( int i=1; i<=n; i++ ){
        //Ha i mod 2 = 0
         if ( i % 2 == 0 ){
            //S:=S+i;
            S=S+i;
            }
    //Különben
         else {
            //Kiírás "Páratlan nem számít: "
            cout  << "Páratlan nem számít: " << endl;
            }
    //Elágazás vége
        //Kiírás i
        cout  << i << endl;
        }
//Ciklus vége
    //Kiírás "A végeredmény:" S
    cout  << "A végeredmény:" << S << endl;
    //x:=0
    x=0;
    //CIKLUS
     do {
        //x:=x+1;
        x=x+1;
        }
//AMÍG x<14
     while ( x<14 );
    //x:=0;
    x=0;
    //CIKLUS amíg x<20
     while ( x<20 ){
        //x:=x+1
        x=x+1;
        }
//ciklusvége
}
```

9. ábra: A generált kódok (`main.cpp` és `generated.cpp`)

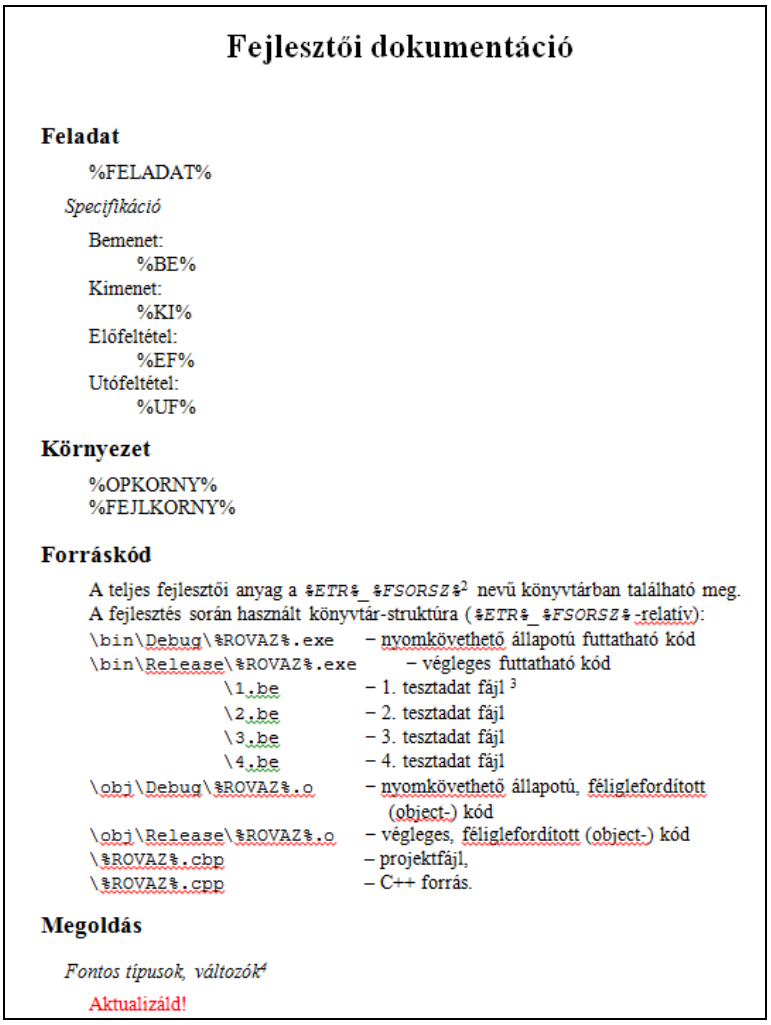
A `main.cpp`-be belegenerálódnak a változó deklarációk és az algoritmusból generált függvény hívása. Ezen kívül kommentekben figyelmeztetjük a hallgatókat, hogy a beolvasást és szükség esetén a kiírást még implementálniuk kell. Ha nem teszik meg hibaüzenettel le is áll a program.

A `generated.h` fájlba belekerül az algoritmust implementáló függvény fejléce. A `generated.cpp` fájlba pedig az algoritmus implementációja. Ez azért van így külön, hogy ha újra generáljuk a kódot egy módosított Excel algoritmusból, akkor elég ezt a `generated.cpp` fájlt átmásolni a projekt előző verziójába és nem kell újra kódolni a beolvasást a `main.cpp`-ben. A generált kódba belekerül kommentbe az algoritmus is, így a kód debuggolása vagyis a hibakeresés közben az algoritmus is olvasható. Tudnak belőle tanulni, illetve könnyebb megtalálni az algoritmusban a javítandó pozíciót.

A generálás közben az algoritmus és a kód előáll egy nagy szöveges adatként, így ezt vissza tudjuk írni az Excelbe egy rejtett munkalapra, hogy onnan könnyen át tudjuk másolni a dokumentációba. Ezért a dokumentum generálását már a verziózott könyvtárban lévő Excelből kell indítani.

2.5 A dokumentáció sablonja

A dokumentáció sablonja az alapozó tárgy beadandójának mintadokumentációján alapszik, ahol bevezettünk `%` (százalék) jellel jelölt változókat, melyek helyére az Excelből átmásolásra kerülnek az információk.



Fejlesztői dokumentáció

Feladat

%FELADAT%

Specifikáció

Bemenet:
%BE%
Kimenet:
%KI%
Előfeltétel:
%EF%
Utófeltétel:
%UF%

Környezet

%OPKORNY%
%FEJLKORNY%

Forráskód

A teljes fejlesztői anyag a *%ETR%_%FSORSZ%*[2] nevű könyvtárban található meg.
A fejlesztés során használt könyvtár-struktúra (*%ETR%_%FSORSZ%*-relatív):

\bin\Debug\%ROVAZ%.exe – nyomkövethető állapotú futtatható kód
\bin\Release\%ROVAZ%.exe – végleges futtatható kód
\1.be – 1. tesztadat fájl [3]
\2.be – 2. tesztadat fájl
\3.be – 3. tesztadat fájl
\4.be – 4. tesztadat fájl
\obj\Debug\%ROVAZ%.o – nyomkövethető állapotú, féliglefordított (object-) kód
\obj\Release\%ROVAZ%.o – végleges, féliglefordított (object-) kód
\%ROVAZ%.cbp – projektfájl,
\%ROVAZ%.cpp – C++ forrás.

Megoldás

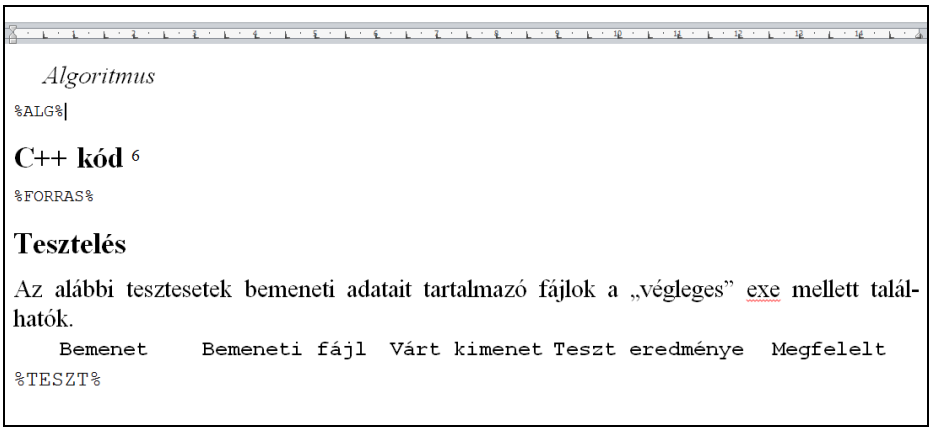
Fontos típusok, változók[4]

Aktualizáld!

10. ábra: Fejlesztői dokumentáció sablonja

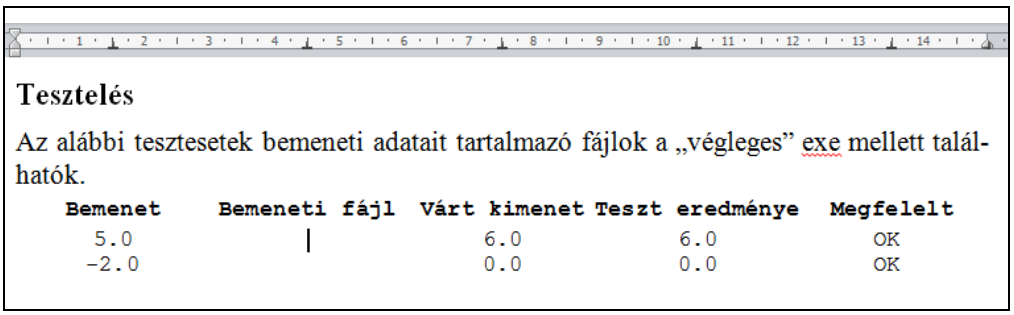
Amit nem tudunk automatikusan kitölteni, azt piros színű „Aktualizáld!” szöveggel jelezzük a hallgatóknak.

Az adatok másolásakor a formázás megmarad, így szerencsére jól olvasható, formázott dokumentáció generálódik, miután beállítottuk a `%ALG%`, a `%FORRAS%` és `%TESZT%` változók soraiban a tabulátorokat és a betűtípust illetve betűméretet.



11. ábra: Az algoritmus és forráskód formázása a sablonban.

Ugyanezt használtuk a tesztelés dokumentálásakor is.



12. ábra: A tesztelés formázása a generált dokumentumban

A dokumentáció sablonjából úgy készül el a kitöltött, félig kész dokumentáció, hogy a kitöltött Excelben az Alap munkalapon rákattintunk a „Dokumentum kitöltése” gombra.

Az első használatkor, ha sikertelen volt a generálás, szükség lehet a Makrók konfigurálására. A VisualMacro más néven Microsoft Visual Basic for Applications alkalmazásban be kell állítani, hogy az Excel makrója elérje a Word dokumentumokat. Ezt a Tools almenü References menüpontjában tudjuk megtenni. Ki kell választani az adott rendszeren legnagyobb verziójú függvényosztályt a Word dokumentumok kezeléséhez. Mi a Microsoft Word 12.0 Object Library-t állítottuk be, ami a magasabb verziókkal kompatibilis és elvileg nincs szükség konfigurálásra.

2.6 A megvalósítás közben felmerült problémák, limitációk, fejlesztési lehetőségek

Elképzelhető, hogy egy problémát egy specifikáció, de több függvény közös használata tud csak kielégíteni. Például a Backtrack esetében. Ezt jelenleg 3 táblázat 3 specifikáció és 3 algoritmussal lehet megtenni, hiszen visszavezethető egy másik (illetve 3 másik) egyszerűbb problémára. De fejlesztési lehetőség az egy projektben több Excel táblából generált függvények felhasználásának megvalósítása.

Jelenleg csak egyszerű típusokat és abból sem mindent kezel a generátor, így ennek fejlesztésére mindenképpen szükség van, hiszen nagyon sok feladat megoldásához tömböt használunk.

3 Kérdések, válaszok

Miközben végiggondoltuk és elkészítettük ezt a fajta segítségnyújtást, bennünk is számtalan kérdés merült fel, melyek közül néhányat itt meg is válaszolunk.

3.1 Miért Excelben kell megírni a dokumentáció kezdeményt, miért nem rögtön Wordben?

A hallgató szempontjából gyakorlatilag mindegy, hogy melyik alkalmazásba gépeli be a legfontosabb dokumentáció részleteket, a feldolgozás szempontjából viszont nem. Excelben egy jól tagolható szerkezetben, a munkalapokkal elkülönítjük a dokumentáció egyes részeit, sablont tudunk adni arra, hogy melyik munkalapra, annak melyik cellájába milyen információ kerüljön. Ebből aztán az Excel programozásával történhet az automatikus kód és dokumentáció generálása. Persze törekedni kell arra, hogy ne legyen túl sok korlátozás az

adatok beírására, mert az akkor megnehezíti a hallgató munkáját és nem foglalkozik az egésszel, hanem marad a régi „fapados” módszernél, amit eddig is követett.

3.2 Struktogram vagy algoritmusleíró nyelv?

A Programozási alapismeretek tantárgyban az algoritmusaink leírására struktogramot tanítunk. Ennek oka, hogy a programozó matematikus és az informatikus képzésben a strukturált programok leírására kezdetektől ezt az eszközt használják tömörsége és egyértelműsége miatt. Mi most mégis algoritmusleíró nyelvet kérünk, de miért? A tanárképzésben és a középiskolai programozás oktatásban tradicionálisan a mondatszerű algoritmusleíró nyelvet használjuk, mert közel áll a magyar nyelvhez, könnyebben érthető a diákok számára, mint a struktogram, ugyanakkor a szabályok betartásával ez is egyértelműen írja le az algoritmust. Úgy gondoljuk, hogy egy leendő szakember számára nem szabadna, hogy gondot jelentsen a két algoritmusleíró eszköz közötti átjárás. A fontos az, hogy először gondolkodjon (tervezzen) majd utána kódoljon, ne pedig tervezés nélkül kódoljon és sikerül, ahogy sikerül. Legyen mindegy, hogy folyamatábra, struktogram, leírónyelv, stb. a tervezés eszköze.

Másrészt, ha valaki az algoritmusból „csak” kódot akar generáltatni, akkor használhatja s structorizer programot is erre a célra. [5] Ennek a programnak struktogrammal kell megadni az algoritmust és választhatunk, hogy milyen (többek között C és Pascal) nyelvű kódot generáljon. A ProgAlapCPPVarazslo célja azonban több az automatikus kódgenerálásnál, inkább vonalvezetőnek és kódolási, dokumentálási segítségnek készül.

3.3 Az automatikus kódgenerálás nem eredményezi-e azt, hogy nem tanulnak meg a hallgatók kódolni?

Természetesen nem szabad teljesen megfeledkezni az oktatásban a kódolásról sem, hiszen ezzel tudjuk kipróbálni elkészült algoritmusainkat és úgy jutnak sikerélményhez a hallgatók, ha látják, hogy munkájuk működik is. Ezért a segítségnyújtásunkat nem rögtön a félév elején kell felajánlani nekik, hanem akkor, amikor már jó néhány programot az órán és házi feladatban kódoltak. Kellenek kész algoritmusok kódolását gyakoroltató órák és feladatok, sőt, hogy a kódolás se szoruljon háttérbe, kell olyan ZH is, ami kész algoritmus kódolását kéri számon.

Később, amikor már képesek a kódolásra, bemutathatjuk Nekik az algoritmusból történő automatikus kódolást, ezzel erősítve bennük azt a felismerést, hogy mennyire fontos a jó algoritmus elkészítése. Persze a hibás algoritmusukból is készülhet kód, amikor látják, hogy nem működik a program, vélhetően bele fognak javítani. Mivel az értékelésnél nagy hangsúlyt fektetünk arra, hogy az algoritmus és a kód összhangban legyenek, kénytelenek visszanyúlni az algoritmushoz és abba is bele javítani. Lehetőségük van így algoritmusuk helyességének ellenőrzésére is, hiszen egy újabb automatikus kódgenerálással ezt megtehetik. Sőt, ezt a szoros kapcsolatot igyekeztünk azzal is bemutatni, hogy a kódba belegenerálódik az algoritmus is megjegyzésekbe.

4 Alkalmazási tapasztalatok

Mielőtt elkészítettük volna ezt a cikket, szerettük volna munkánkat elbíráltatni az érintett célcsoporttal. Első lépésben azok véleményére voltunk kíváncsiak, akik már elvégezték a programozási alapismeretek tantárgyat. Készítettünk egy kérdőívet, amit egy Google űrlapon, a [2] URL-en válaszolhattak meg az érintett hallgatók, majd kiértékeltük a válaszaikat. Nem vártuk el, hogy nevüket adják a válaszokhoz, bízván abban, hogy így objektív válaszokat adnak.

4.1 A kérdőív

A kérdőív kérdéseit 2 csoportba foglaltuk. Először a múltra kérdeztünk rá, vagyis arra, hogy a tantárgy elvégzésekor hogyan készítették el a beadandót. Majd odaadtuk nekik kipróbálásra a ProgAlapCppVarázslót a használati utasítással együtt és a kipróbálás közben szerzett tapasztalataikra vonatkozó kérdéseket tettünk fel.

4.1.1 A Programozási Alapismeretek tantárgy beadandó feladatára vonatkozó kérdések:

1. Mennyi időt töltött a beadandó elkészítésével? (órában)
2. Milyen százalékos megoszlás volt az algoritmus A= kódolás K= és dokumentáció D= között. Ha ez előzőek összege nem 100, akkor mire fordította a többi időt?
3. Milyen sorrendben készítette el az algoritmust (A), kódot (K) és dokumentációt (D)?

4.1.2 A ProgAlapCppVarazslo kipróbálására vonatkozó kérdések

4. Mennyire tartja jónak az ötletet? (Nem látom értelmét / Jó az ötlet, de nagyon munkaigényes az alkalmazás / Jónak tartom / Nagyon jónak tartom)

5. Ön szerint segíti-e az algoritmuskészítés fontosságának erősödését? (Nem / Részben / Igen)
6. Ön szerint segíti-e a dokumentációkészítést? (Nem / Részben / Igen)
7. Lát-e valamilyen veszélyt abban, ha ezt az alkalmazást használják a hallgatók? (szöveges válaszát kérjük)
8. Zavarja-e, hogy az algoritmust kell előbb elkészíteni? (Igen / Nem)
9. Használta volna-e ezt a segédeszközt, ha annak idején megkapja? (Igen / Nem)
10. Hogy érzi, tudott volna időt megspórolni a beadandó elkészítésekor ezzel az eszközzel? (Nem készült volna el / Hátráltatott volna / Semennyit / Keveset / Sokat)
11. Talált-e hibát az alkalmazásban, és ha igen, akkor mit? (szöveges válaszát kérjük)
12. Milyen fejlesztési javaslatai lennének? (szöveges válaszát kérjük)
13. Egyéb észrevételek: (szöveges válaszát kérjük)

4.1.3 A kérdőív kiértékelése

2-300 főnek, a Programozási alapismeretek tárgyat már elvégzett hallgatónak küldtünk értesítést, hogy tesztelje és értékelje oktatást segítő anyagunkat. A tesztelés és a válaszadás mindössze 20-30 percet vett volna igénybe, kb másfél hetes határidővel. A hallgatók inaktivitását jellemző volt a reakció. 13 embertől kaptunk értékelhető választ.

1. Átlagosan 17,5 óra alatt készítették el a beadandót. Volt, aki 90 órát szánt rá, de valaki csak fél órát.
2. Az idejük 26,41%-át fordították az algoritmusra (5 és 60% között), 40,56%-ot a kódolásra (28 és 85% között), 32,65%-ot pedig dokumentálásra (10 és 60% között). A fennmaradó időben történő cselekedetekhez a hibakeresést, dokumentáció formázását, utánajárást/internetezést említették.
3. A hallgatók 25%-a bevallottan a kódolással kezdte a munkát. 100% hagyta a végére a dokumentációt.
4. A válaszadók 15,4%-a nem látja értelmét, míg ugyanennyien nagyon jónak tartják. 30,8%-uk jónak tartja, 38,5%-uk pedig jónak, de munkaigényesnek.
5. Senki nem mondta, hogy nem segíti az algoritmuskészítés fontosságának erősödését, de 61,54% szerint csak részben.
6. A dokumentáció készítést 7,7% szerint nem segíti, míg a többiek 46,15%-46,15%-ban megosztottak, hogy segítség-e vagy csak részben.
7. A hallgatók a következőket említették: csak fejleszt; nem árt; még kevesebbet fog gondolkodni; kódolást nem tanulják meg; mindenre nem lehet excel táblával felkészülni.
8. Csak 15,4% mondta azt, hogy zavarja, hogy az algoritmust előbb kell elkészíteni.
9. A válaszadók 69%-a használta volna ezt a segédeszközt.
10. Az idő megspórolására vonatkozó kérdésre úgy válaszoltak, hogy 15,4% sok időt spórolt volna meg és ugyanennyit hátráltatott volna. A többiek kevés időt megspóroltak volna, de senkit nem hátráltatott volna.
11. Hibát nem jeleztek.
12. Az Excel helyett saját grafikus felületet, kevesebb lépést és a struktogramm integrálását említették fejlesztési lehetőségnek.
13. Hasznos lenne ez az új alkalmazás. A többi megjegyzést a tárgyhoz általánosságban írták, hogy ne egy tárgyban legyen az algoritmus és a kódolás oktatása.

Az örömteli, hogy a kevés válaszadó hasznosnak és az algoritmizálás fontosságát kiemelőnek találta munkánkat, bízunk benne, hogy nem a tárgy elvégzése után, hanem az oktatás aktuális pillanatában majd nagyobb érdeklődést kelt a ProgAlapCppVarázsló alkalmazása.

5 Irodalom

1. http://xml.inf.elte.hu/ProgAlapCppVarazslo_v1.0.20121017.zip
2. https://docs.google.com/spreadsheet/viewform?formkey=dGxHdldudWZDUElTTXNhdDlFWXpfc0E6MQ
3. Dijkstra E.W., A Discipline of Programming, Prentice-Hall, Englewood Cli_s, 1973.
4. Szlávi Péter, Zsakó László: Módszeres programozás: Programozási bevezető, 18. Mikrológia

5. http://structorizer.fisch.lu/

6. Sz. Csepregi, A. Dezső, T. Gregorics, S. Sike: Automatic Implementation of Service Required by Components, Workshop on Property Verification for Software Components and Services ,PROVECS 2007, http://lina.atlanstic.net/provecs/2007/provecs2007proceedings.pdf

7. http://poi.apache.org/