Proceedings of the 9th International Conference on Applied Informatics
Eger, Hungary, January 29–February 1, 2014. Vol. 2. pp. 9–16
doi: 10.14794/ICAI.9.2014.2.9

PFW as generator

László Menyhárt, Gáborné dr. Pap

Department of Media & Educational Informatics
Eötvös Loránd University, Budapest, Hungary
menyhart@inf.elte.hu
papne@inf.elte.hu

Abstract

Some months ago we presented on another conference the PFW: Programming Fundamentals Wizard, which is the improved version of helper tool for the subject Programming fundamentals. Our web-based application gives place to collect the information from the tasks and it contains an editor for Nassi-Shneiderman diagram. This guide helps the students to keep the expected order of problem solving namely at first thinking and secondly coding or even collecting the information, creating algorithm independently from program languages then creating source code. Unfortunately deep testing is not specific and documentation is required only as homework based on an example.

This time we present two new functions of PFW. One of the new functions is good to generate the first, raw version of the documentation which must be edited even. Second function generates C++-like source code. We would like to emphasize that the primary aim of this application is not the generation, it was not developed because of this but this beta version of C++-like source code helps intensively the beginner programmers in the implementation and understanding the code. Generating a compilable and runnable source code is possible with using program-language-specific expressions in the Nassi-Shneiderman diagram.

Keywords: teaching programming, code generation, documentation

MSC: 97B40, 97R99, 97Q60

1. Introduction

Computer science is a young and a diverse science. That is why education of computer science is very difficult. It is very important to clarify that who will learn it and on which level, because it defines the area and the depth. In this article we present some possibilities of a guide application (PFW) to the Programming Fundamentals course at University. There are more methods to teach programming

[1]. We use methodological and algorithm oriented programming on the foundation courses of students of informatics. This method takes in the whole process of implementation from creating specification to writing documentation. In an earlier paper we presented an own developed application, PFW, which helps the students in this systematic learning. We presented the PFW application and the method how it changes up the process of creating algorithm, helps the implementation and how it can be used in education. In this article we present two new functions of PFW. User is able to generate source code and documentation with these.

2. Short story of PFW

PFW is a guide on web platform. It helps to the students analyzing and thinking over their task in our programming courses. Data the tasks and their correlation can be collected on these pages so students are guided through the specification.

They can edit on a webpage the image representation of Nassi-Shneiderman diagram of the algorithm with mouse clicking. They can store the test cases and test results on this platform.

3. Implementation

This own developed, web-based application [14] was implemented in Java and it runs on our server. User must login for using it. Three possible ways are the Gmail identifier, user id in INF.ELTE domain or after a registration in the system with unique user identifier and password. Every user has an own workspace in the database where tasks are stored. The storage format is XML [6] from which the information on the screen and codes are generated.

3.1. Surface of the application

Figure 1 shows the initial screen of the application which contains introduction, link to the registration and login.

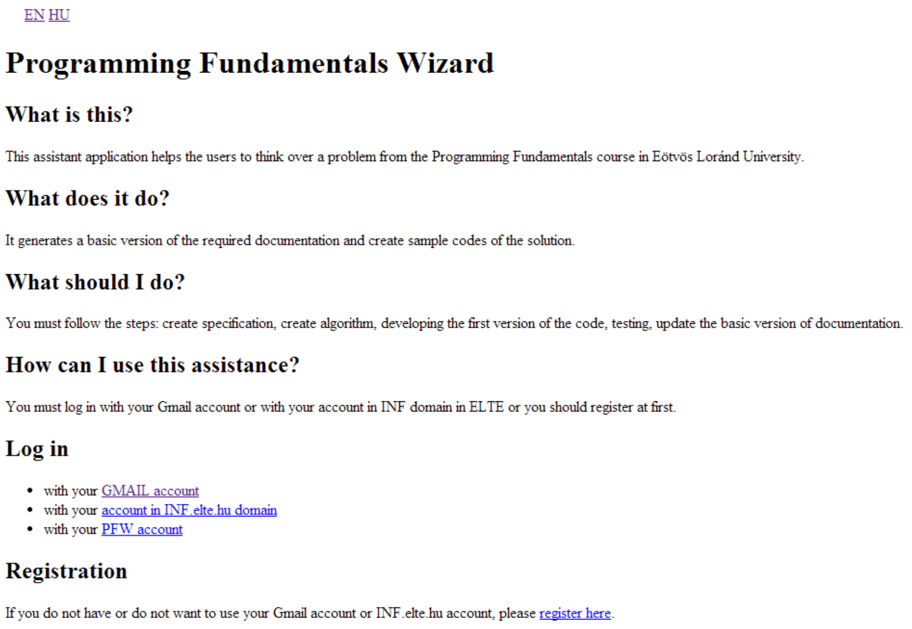
The first menu has menu items for opening a task or creating a new task. Figure 2 shows the first page after the successful login.

The menu will be changed when the user open or create a task. The new menu supports the saving and closing of task and it's solving steps. Figure 3 shows the main page.

Figure 4 shows the pages on which the user creates the specification.

User fabricates the algorithm in NSD format using a graphical view of it. Figure 5 shows the floating menu of editing the NSD.

After these steps user can fill out the data on the testing and the possible future improvements pages.



EN HU

Programming Fundamentals Wizard

What is this?

This assistant application helps the users to think over a problem from the Programming Fundamentals course in Eötvös Loránd University.

What does it do?

It generates a basic version of the required documentation and create sample codes of the solution.

What should I do?

You must follow the steps: create specification, create algorithm, developing the first version of the code, testing, update the basic version of documentation.

How can I use this assistance?

You must log in with your Gmail account or with your account in INF domain in ELTE or you should register at first.

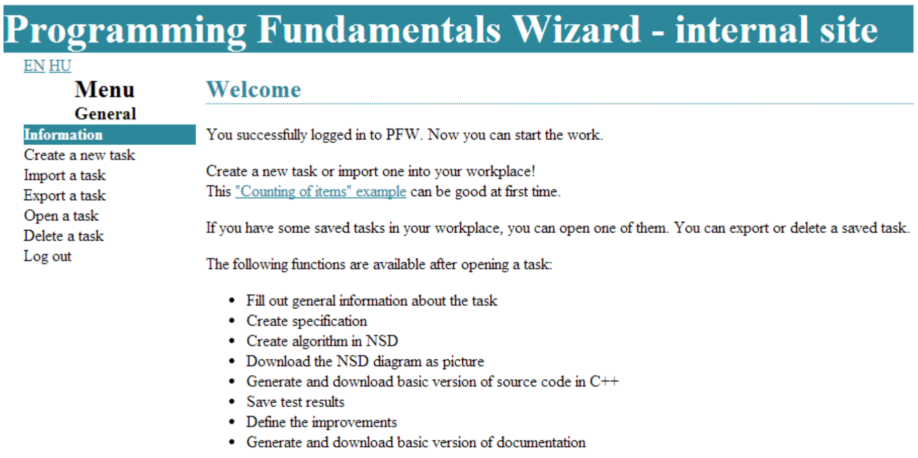
Log in

- with your GMAIL account
- with your account in INF.elte.hu domain
- with your PFW account

Registration

If you do not have or do not want to use your Gmail account or INF.elte.hu account, please register here.

Figure 1: Initial screen of PFW



Programming Fundamentals Wizard - internal site

EN HU

Welcome

You successfully logged in to PFW. Now you can start the work.

Create a new task or import one into your workplace!
This "Counting of items" example can be good at first time.

If you have some saved tasks in your workplace, you can open one of them. You can export or delete a saved task.

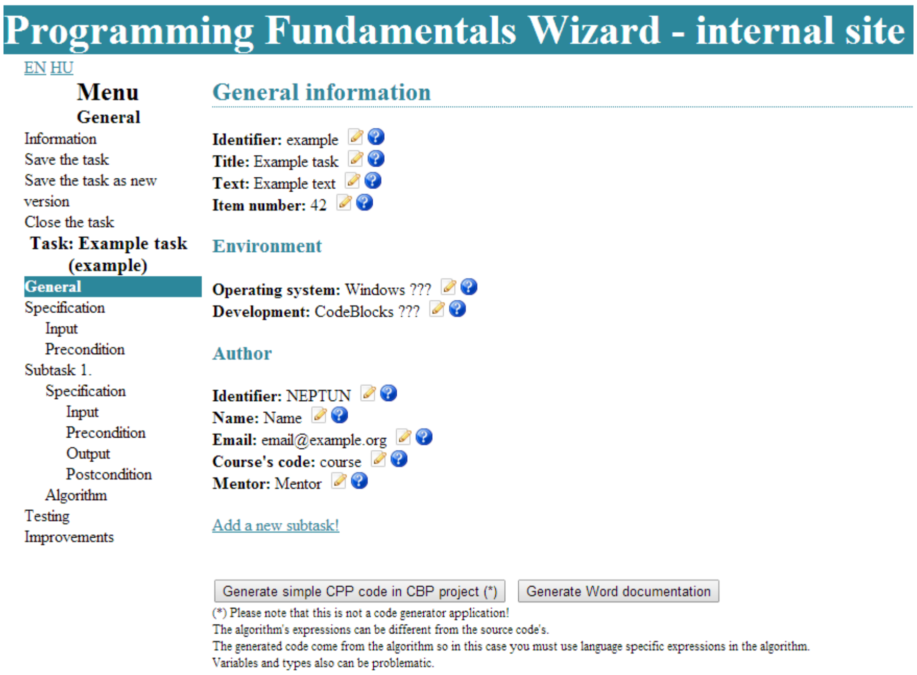
The following functions are available after opening a task:

- Fill out general information about the task
- Create specification
- Create algorithm in NSD
- Download the NSD diagram as picture
- Generate and download basic version of source code in C++
- Save test results
- Define the improvements
- Generate and download basic version of documentation

Figure 2: After the successful login

3.2. Functions of generation

The user performed the design part of solving the task a la the previous paragraph. This is the most important step. After this the coding and creating the documen-


Programming Fundamentals Wizard - internal site

EN HU

Menu

General

Information
Save the task
Save the task as new version
Close the task

Task: Example task (example)

General
Specification
  Input
  Precondition
Subtask 1.
  Specification
    Input
    Precondition
    Output
    Postcondition
  Algorithm
Testing
Improvements

General information

Identifier: example
Title: Example task
Text: Example text
Item number: 42

Environment

Operating system: Windows ???
Development: CodeBlocks ???

Author

Identifier: NEPTUN
Name: Name
Email: email@example.org
Course's code: course
Mentor: Mentor

Add a new subtask!

Generate simple CPP code in CBP project (*)   Generate Word documentation

(*) Please note that this is not a code generator application!
The algorithm's expressions can be different from the source code's.
The generated code come from the algorithm so in this case you must use language specific expressions in the algorithm.
Variables and types also can be problematic.

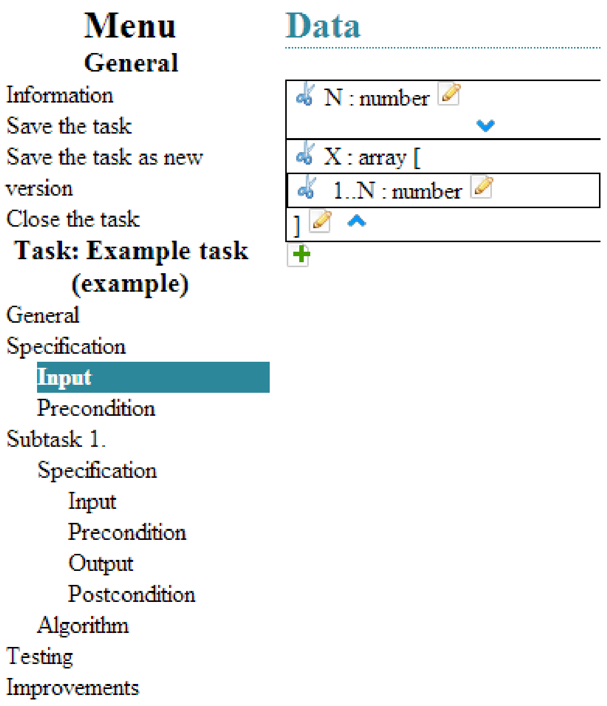
Figure 3: Example task

tation demand less thinking. Several parts of these can be mechanically produced. PFW's new functions grab these parts and help the user with generation. They have not to double work so they save time. It is not our intention to offer correct source code and documentation. Students must correct and update both stuffs.

3.2.1. Generation of source code

Generation of C++ source code starts with a click on a button (See Figure 6) on the page General.

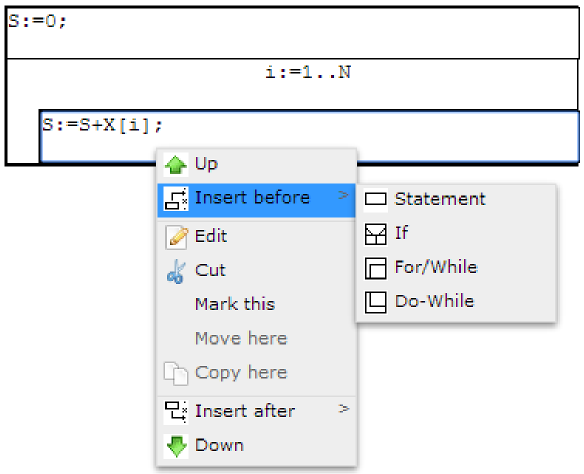
A compressed file is downloaded which contains a Code::Blocks project. In good case generation is required only one times. If the algorithm was modified, the new generation has a unique name as a version. Three files are in the project because the source code is taken apart to the main part and generated functions of subtasks. Main part contains the input and output handling, so the user must perform more work on this. NSD is very flexible and the boxes can contain anything because it is for human and not for the computer. But using the symbol system learned on the course is useful because the initiative C++ code is generated from this. The algorithm does not contain the reading input, writing output and other language specific items. For example operators must be rewrite in the source code. It is mechanical work, but sometimes it is difficult for the students. For instance



Menu
General
Information
Save the task
Save the task as new version
Close the task
Task: Example task (example)
General
Specification
Input
Precondition
Subtask 1.
Specification
Input
Precondition
Output
Postcondition
Algorithm
Testing
Improvements

Data
N : number
X : array [
1..N : number
]

Figure 4: Editing the specification



```
S:=0;
i:=1..N
S:=S+X[i];
```

Figure 5: Floating menu at NSD editing

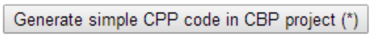
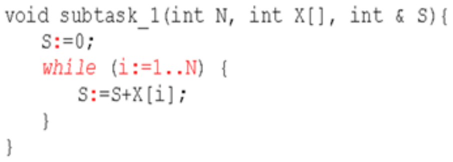


Figure 6: Button for generating source code

the indexes of arrays need a bit thinking because C-like languages' points to the beginning of an item, so brackets must contain one less value (0 instead of 1).

The generator produces a function to every subtask. The parameters of the function, with its name and type, are generated from the specification. That is why the specifications of subtasks are very important. The function body will be generated from the NSD diagram. User must translate this faulty source to the correct C++ specific expressions.



```
void subtask_1(int N, int X[], int & S){
    S:=0;
    while (i:=1..N) {
        S:=S+X[i];
    }
}
```

Figure 7: Generated source code

Figure 7 shows the generated C++ source code. We marked with red color the places where user must modify it for getting a runnable code with correct syntax.

3.2.2. Generation of documentation

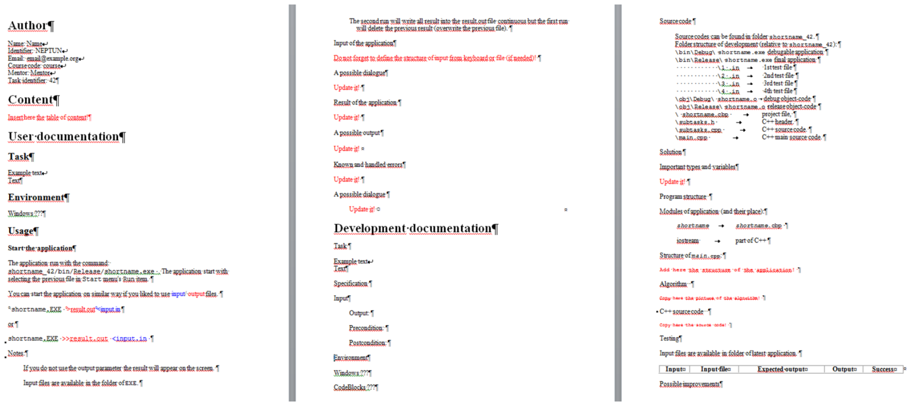
Generation of documentation starts with a click on a button (see. Figure 8) on the page General.

Figure 8: Button for generating documentation

This document is based on an example with the same content and structure. The content is filled with every known data which are stored in the PFW. The expected but unknown places contain red colored "Update it!" expression where students must edit it. For instance users must paste the screenshots into the user manual and diagram of project structure must be copied into the development documentation.

3.3. Technology

The basic idea of the generation is XSLT. The XML Stylesheet Language defines the output of the transformation from the XML. Fortunately PFW stores the data in XML format. C++ source code is text which can be generated from XML



Author

Name: Name
Identifier: NEPTUN
Email: email@example.org
Course code: course
Mentor: Mentor
Task identifier: 42

Content

Insert here the table of content!

User documentation

Task

Example text
Text

Environment

Windows ???

Usage

Start the application

The application run with the command: `shortname_42/bin/Release/shortname.exe`. The application start with selecting the previous file in `Start` menu's Run item.

You can start the application on similar way if you liked to use input/output files.

`shortname.EXE >result.out <input.in`

or

`shortname.EXE >>result.out <input.in`

Notes:

If you do not use the output parameter the result will appear on the screen.

Input files are available in the folder of `EXE`.

The second run will write all result into the result.out file continuous but the first run will delete the previous result (overwrite the previous file).

Input of the application

Do not forget to define the structure of input from keyboard or file (if needed)!

A possible dialogue

Update it!

Result of the application

Update it!

A possible output

Update it!

Known and handled errors

Update it!

A possible dialogue

Update it!

Development documentation

Task

Example text
Text

Specification

Input

Output:

Precondition:

Postcondition:

Environment

Windows ???

CodeBlocks ???

Source code

Source codes can be found in folder `shortname_42`.
Folder structure of development (relative to `shortname_42`):

```
\bin\Debug\ shortname.exe debugable application
\bin\Release\ shortname.exe final application
            \1 .in    →    1st test file
            \2 .in    →    2nd test file
            \3 .in    →    3rd test file
            \4 .in    →    4th test file
\obj\Debug\ shortname.o → debug object-code
\obj\Release\ shortname.o release object-code
\ shortname.cbp      →    project file,
\subtasks.h          →    C++ header.
\subtasks.cpp        →    C++ source code.
\main.cpp            →    C++ main source code.
```

Solution

Important types and variables

Update it!

Program structure

Modules of application (and their place):

shortname → *shortname.cbp*

iostream → part of C++

Structure of `main.cpp`:

`Add here the structure of the application!`

Algorithm

`Copy here the picture of the algoritm!`

C++ source code

`Copy here the source code!`

Testing

Input files are available in folder of latest application.

Input	Input file	Expected output	Output	Success

Possible improvements

Figure 9: Generated documentation

with XSLT. We used HTML output for the document but the extension and the mimetype is set to signify to the Microsoft Windows system that the content is for Microsoft Word. MS Word opens the file correctly after this trick.

4. Conclusion

We presented two new functions in our web-based application. These generations are good for students and teachers, too. They can save time with these and the students can study the connection between the specification and data structure of the application and between the algorithm and source code.

References

[1] SZLÁVI, P., ZSAKÓ, L., Methods of Teaching Programming, Teaching Mathematics and Computer Science Vol. 1/2, (2003), 247–257

[2] NASSI, I., SHNEIDERMAN, B., Flowchart techniques for structured programming, ACM SIGPLAN Notices Vol. 8 Issue 8, (1973), 12-–26

[3] DIJKSTRA, E.W., A Discipline of Programming, Prentice-Hall, (1973),

[4] SZLÁVI, P., ZSAKÓ, L., Módszeres programozás, Programozási bevezető, 18. Mikrológia, (1996)

[5] CSEPREGI, Sz., DEZSŐ, A., GREGORICS, T., SIKE, S., Automatic Implementation of Service Required by Components, Workshop on Property Verification for Software Components and Services, (2007), `http://lina.atlanstic.net/provecs/2007/provecs2007proceedings.pdf`

[6] MENYHÁRT, L., Can a language be before “the first programming language”?, Teaching Mathematics and Computer Science Volume IX, Issue II, (2011), 209-224, ISSN: 1589-7389, `http://tmcs.math.klte.hu/Contents/2011-Vol-IX-Issue-II.html`

[7] MENYHÁRT, L., PAP, G.né, Dokumentáció alapú programfejlesztés, INFODIDACT 2012 conference, Zamárdi, Hungary (2012)

[8] MENYHÁRT, L., PAP, G.né, How can we get our students to think while we help their work too?: Document based development, Proceedings of the 7th International Multi-Conference on Society, Cybernetics and Informatics. (2013) pp. 97–102.

[9] MENYHÁRT, L., PA,P G.né, A programozási alapismeretek oktatását támogató eszköz továbbfejlesztett változatának bemutatása, INFODIDACT 2013 conference, Zamárdi, Hungary, (2013)

[10] SHNEIDERMAN, B., A short history of structured flowcharts (Nassi-Shneiderman Diagrams), University Maryland, `http://www.cs.umd.edu/hcil/members/bshneiderman/nsd/` (2003)

[11] FISCH, B., Structorizer, `http://structorizer.fisch.lu/`, (2009-2013)

[12] MOLNÁR, T., StukiMania, http://stukimania.hu, (2012-2014)

[13] MENYHÁRT, L., Short description of Programming Fundamentals Wizard, URL: `http://xml.inf.elte.hu/2013_14_1/progalap/anyagok/PFW_leiras.pdf`, (2013)

[14] MENYHÁRT, L., Programming Fundamentals Wizard, URL: `https://157.181.166.134:8181/PFW/index2_HU.jsp`, (2013)