A Training Software Tool for Square Root Calculation

Anelia Vasileva, Angel Smrikarov, Teodora Toteva

Abstract: *The paper justifies the necessity to introduce the students from the 'Computer Systems and Technologies' degree course to algorithms, used for square root calculation in computers. Two of the most widely used iterative algorithms for fixed and floating point numbers square root approximation are discussed and the steps of development of the training environment are described.*

Key words: *Square root calculation, Training software environment, Virtual laboratory.*

INTRODUCTION

The square root calculation is a comparatively rarely performed operation in computers and in most cases it is executed via subroutines. These subroutines work according algorithms, which are based on some of the well-known numerical methods for square root calculation. For the students, taught in "Computer Organization" course is important to become familiar with these methods and the way they are applied in computers. This could be achieved through development of a software environment which will enable students to explore the algorithms for square root calculation. The environment should be included in the Virtual Laboratory on Computer Organization, described in [1].

LAYOUT

The main goal of the environment is to illustrate the algorithms for square root calculation of fixed and floating point numbers, therefore the first task is selection of two of the well-known numerical analysis methods [2] to be laid as a ground for the algorithms, which will be introduced to the learners.

1. Square Root Calculation of Fixed Point Numbers.

The simplest iterative method for square root approximation is the Babylonian method [3, 4]. Here is presented the algorithm, based on it. Let $y = \sqrt{x}$, where x is a fixed point number. For y calculation could be used the following iterative formula:

$$y_{i+1} = \frac{y_i + \frac{x}{y_i}}{2}.$$

It is proved that when i tends to infinity, y_{i+1} tends to a square root of x. In practice, the value of the maximal relative error δ_m is preliminary assigned and the iterations are repeated while

$$\delta = \frac{\mid y_{i+1} - y_i \mid}{y_i} \cdot 100 \le \delta_m .$$

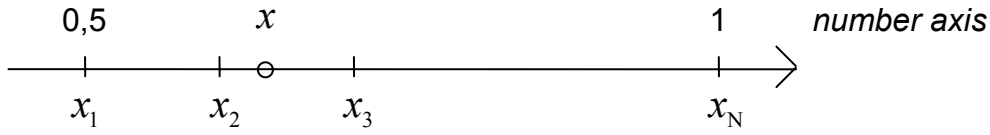
It is obvious that the number of iterations will depend on δ_m and y_0 (the first approximation to the square root value).

2. Square Root Calculation of Floating Point Numbers.

The algorithm is based on a modification of the considered above method. Let $Y = \sqrt{X}$, where X is a floating point number. $X = 2^{Px} x$ and since the mantissa x is normalized in the floating point machine, the following inequality holds:

$$1 > x \geq 0{,}1_2 = 0{,}5_{10},$$

Y should be obtained in the form $2^{Py}y$, which means that the solving the problem is reduced to calculation of the exponent and the mantissa of the square root. To this purpose are selected N points, evenly distributed in the interval [0,5 ÷ 1], i.e. x_1=0,5;…; x_N=1



and for each point the value of $\sqrt{x_i}$ is calculated preliminary. The array of these values is saved in ROM memory cells with addresses *A+1, A+2, ... , A+N*. The function Y is represented in the following form:

$Y = \sqrt{X} = \sqrt{2^{Px}x} = \sqrt{2^{Px}}\sqrt{x}$, $x = x_i + \Delta x$, where x_i is the point with closer to x value from the nearest points of x in the sequence $x_1, x_2, \cdots, x_N$, and $\Delta x = x - x_i$. It is obvious that Δx could have negative or positive value.

$$Y = \sqrt{2^{Px}}\sqrt{x_i + \Delta x} = \sqrt{2^{Px}}\sqrt{x_i\left(1 + \frac{\Delta x}{x_i}\right)} = \sqrt{2^{Px}}\sqrt{x_i}\sqrt{1 + \frac{\Delta x}{x_i}} \quad .$$

It could be seen that

when P_x is an even number $\sqrt{2^{Px}} = 2^{\frac{Px}{2}}$, and

when P_x is an odd number $\sqrt{2^{Px}} = \sqrt{\frac{2^{Px+1}}{2}} = \frac{\sqrt{2^{Px+1}}}{\sqrt{2}} = \frac{2^{\frac{Px+1}{2}}}{\sqrt{2}} = 2^{\frac{Px+1}{2}}\frac{\sqrt{2}}{2}$

Consequently,

when P_x is even – $P_y = \frac{P_x}{2}$, and when P_x is odd – $P_y = \frac{P_x + 1}{2}$.

The value of $\sqrt{x_i}$ is taken from the ROM as the index i is calculated with the formula

$$i = \mathrm{int}\left\{\left[x + \frac{0{,}5}{2(N-1)}\right] \cdot \frac{N-1}{0{,}5}\right\}.$$

This index is used for finding the address of the memory cell where the requested value is saved i.e. A+i.

The function $\sqrt{1 + \frac{\Delta x}{x_i}} = \sqrt{1 + z_i}$ could be decomposed in the following convergent power series:

$$\sqrt{1 + z_i} = 1 + \frac{1}{2}z_i - \frac{1}{2.4}z^2{}_i + \frac{1.3}{2.4.6}z^3{}_i - \frac{1.3.5}{2.4.6.8}z^4{}_i + \cdots$$

The convergence condition is:

$$|z_i| = \left|\frac{\Delta x}{x_i}\right| < 1 .$$

Since $1 > x \geq 0{,}5_{10}$, if N = 3 ($\Delta x_{\max} = 0{,}25$) the condition will be fulfilled. However, in practice N = 5 ÷ 6, as only the first 4 ÷ 5 terms of the power series are taken. Hence when P_x is even – $y = \sqrt{x_i}\sqrt{1+z_i}$, and when P_x is odd – $y = \frac{\sqrt{2}}{2}\sqrt{x_i}\sqrt{1+z_i}$.

It is obvious that in case of fixed number of the points in the interval and fixed exponent of the power series, the value of the error will depend only on the proximity of x to some of the points x_i.

3. Design and Implementation.

The second task includes specifying the functional requirements, which will determine the functionality and structure of the developed training environment. They can be described as follows:

The environment should:

- enable the learner to select the algorithm, that is to be represented;
- allow entering initial data, required for the selected algorithm calculation;
- illustrate step by step the execution of the algorithm;
- allow the learner to control the algorithm execution;
- allow repeatedly execution of the algorithm with different input data for each repetition;
- save intermediate calculations and present the output after algorithm completion;
- provide graphical representation of the results and printing the graphics.

The analysis of the requirements is used for basic modelling of the environment structure (fig.1).

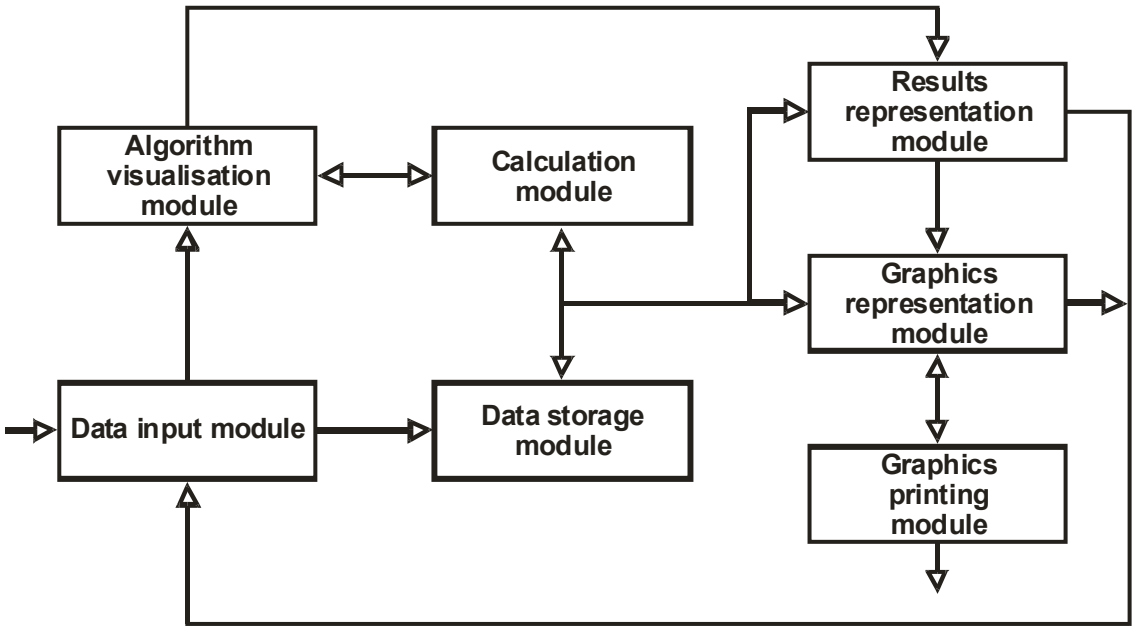


Figure1. A structural model of the training environment

After the requirements analysis a user tasks specification is produced. It is conducive for learner behavior modelling (fig.2), which allows outlining the functionality of the developed environment.

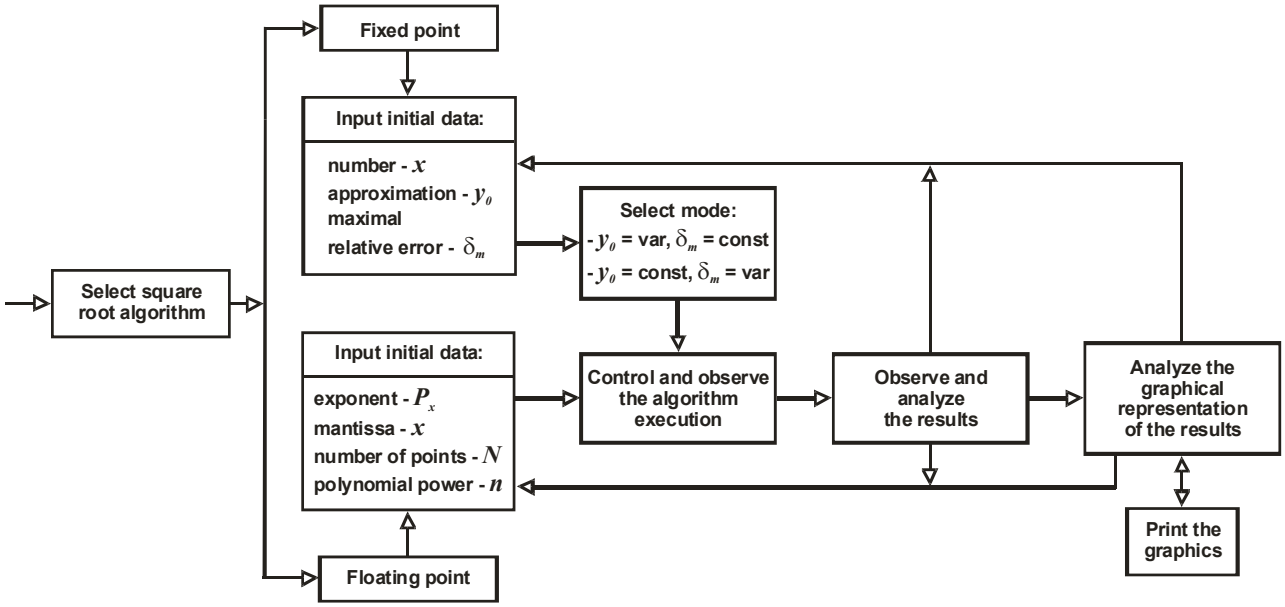


Figure2. A model of learner behavior in the training environment

The implemented environment is a Windows application and runs on Windows 9x, 2000, ME and XP. The welcome window contains a menu where the desired algorithm could be selected. When fixed point square root algorithm is selected a data input window is opened (fig.3) and the learner should enter a fixed point number x, the first approximation y_0 and the maximal relative error δ_m. "Continue" button opens the next window (fig.4) where the algorithm is performed as the learner controls each step execution by clicking a button and can observe the intermediate calculation results. When the algorithm is completed the learner is prompted to set the types of parameters, necessary for the graphical representation. The graphical representation illustrates the number of iterations as a function of y_0 or δ_m. There are two options: y_0 = constant, δ_m = variable and y_0 = variable, δ_m = constant. The next window (fig.5) contains a table, where the registered number of iterations is saved after each algorithm repetition with different initial data. When the table contains more than one result, a graphic could be produced and printed (fig.6). Via clicking a button the learner can go back from the results representation window to the data input window for changing the initial data and performing the algorithm with new initial values of x, y_0 and δ_m. The exploration of floating point square root algorithm is organized the same way as the discussed above one. The input data window (fig.7) contains form for entering the exponent P_x, the mantissa x, the number of points N and the power of the approximate polynomial n. Figures 8 and 9 show the algorithm visualization window and results representation window. The graphical representation, produced as an output, illustrates δ_m as a function of x and could be printed from the print management window (fig.10). When the tables, containing results are full of data, the learner is prompted to clear them and begin new exploration. Clearing the tables returns to the data input window for entering new data.

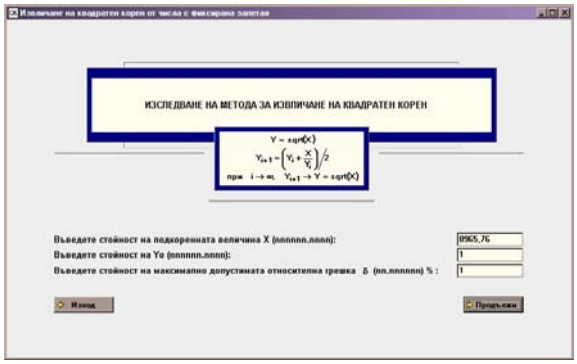


Figure 3. Square root algorithm for fixed point numbers – data input window

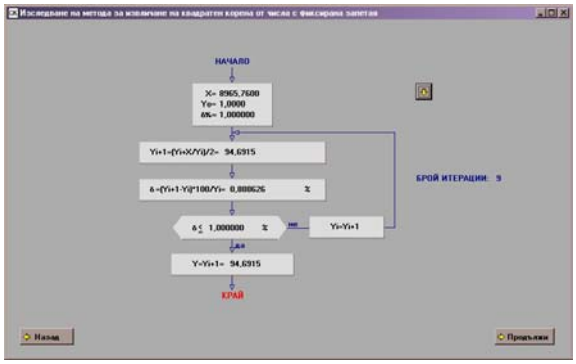


Figure 4. Square root algorithm for fixed point numbers – algorithm visualisation window

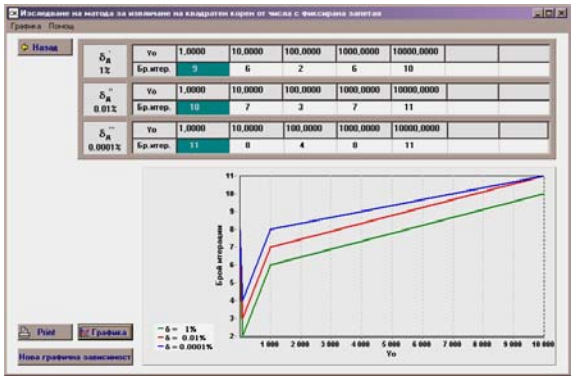


Figure 5. Square root algorithm for fixed point numbers – results representation window

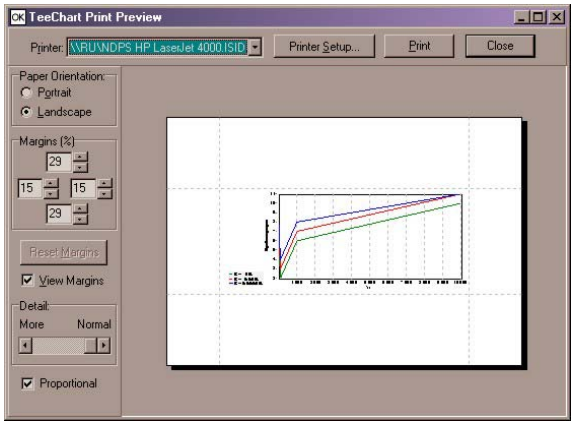


Figure 6. Square root algorithm for fixed point numbers – print preview window

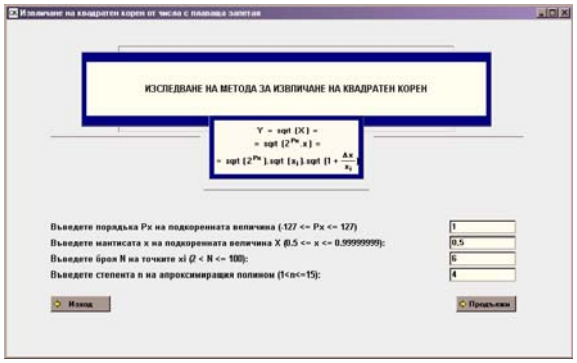


Figure 7. Square root algorithm for floating point numbers – data input window

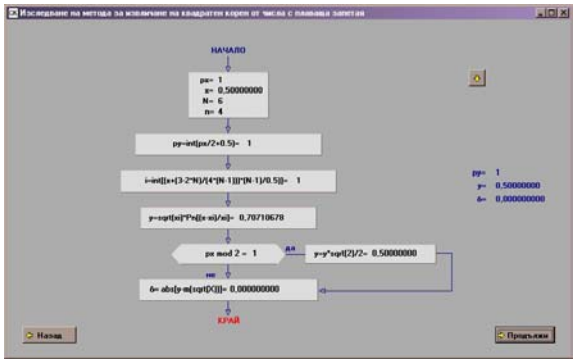


Figure 8. Square root algorithm for floating point numbers – algorithm visualisation window

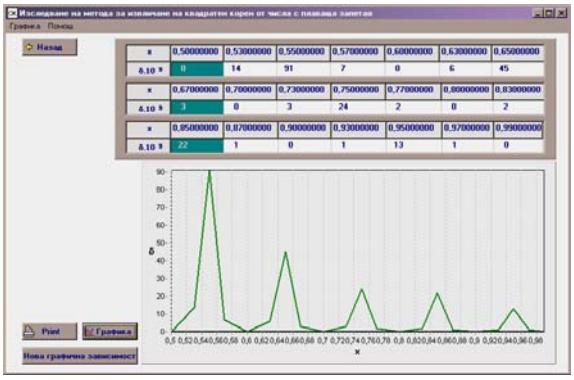


Figure 9. Square root algorithm for floating point numbers – results representation window

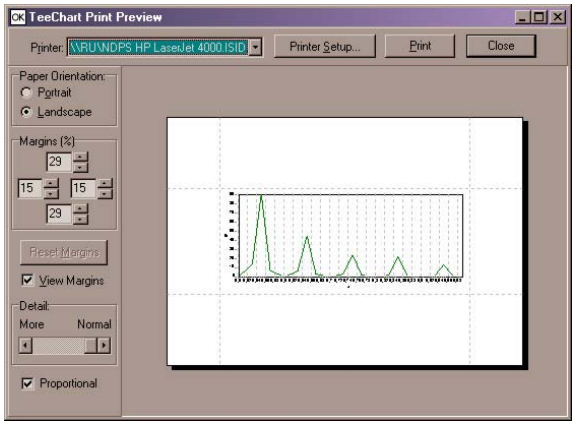


Figure 10. Square root algorithm for floating point numbers – print preview window

CONCLUSIONS AND FUTURE WORK

The implemented environment introduces the learners with the most used algorithms for square root approximation in computers. It has user friendly interface, provides help and allows observation, analysis, comparison of the obtained results and graphical representations and printing the latter. It expands the preliminary gained theoretical knowledge on the topic and is appropriate for self-preparation for the exam.

REFERENCES

[1] Vasileva, A., A. Smrikarov, T. Hristov, A Conceptual Model of a Virtual Laboratory on Computer Organization, Proceedings of the CompSysTech'2002 Sofia, 20-21 June 2002.

[2] Faires,J.D., R. Burden, Numerical Methods, Brooks/Cole Publishing Company, 1998.

[3] http://www.fact-index.com/s/sq/square_root.html

[4] http://personal.bgsu.edu/~carother/babylon/Babylon1.html

ABOUT THE AUTHORS

Anelia Vasileva, Assistant, Department of Computing, University of Rousse, Phone: +359 (0)82 888 276, E-mail: asvasileva@ecs.ru.acad.bg

Assoc.Prof. Angel Smrikarov, PhD, Department of Computing, University of Rousse, Phone: +359 (0)82 888 743, E-mail: ASmrikarov@ecs.ru.acad.bg

Teodora Toteva, MSc, Department of Computing, University of Rousse, E-mail: tedi_nik@yahoo.com.