LINEAR SEARCH – THE BREAKS IN TEACHING-LEARNING PRACTICE

SZALAYNÉ TAHY, Zsuzsanna – CZIRKOS, Zoltán HU

Abstract: Linear search is one of the well-known algorithms used in everyday practice but its coding is difficult for many. Students' learning practices break on linear search. There are advanced coding tools, proven methods and common tricks, but using any of these causes conflict on learning programming between teacher and students. Typical conflict is given in context of using the `break` instruction while coding linear search. Why do secondary school students love and why do teachers hate the "`break;`"? Is the `break` instruction really "to be considered harmful" when learning programming? How can teachers resolve conflicts considering the aims of learning the algorithms, coding and structured programming, improving problem solving and computational thinking skills?

Key words: linear search, learning methods, teaching methods, computational thinking.

1 Introduction

There are eight methods for teaching programming according to [1]: methodical/algorithmic oriented; data oriented; specification oriented; problem type-oriented; language oriented; instruction oriented; mathematics oriented; hardware oriented; model oriented. In teaching practice, these methods are combined considering the curriculum and – sometimes – the students' preparatory skills. Having been teaching programming for years, and having tried many combinations of these methods, a critical topic was recognized that determines the students' progress in programming. This topic is the linear search in coding practice.

In a typical class, there are students with different levels of knowledge.

1. Beginners, who have learnt linear search algorithms but cannot apply it. (They say they need some `if()` or they would like to use spreadsheets and built in MATCH() function.)
2. "Programmers" who developed software on iPhone, Android or web environment.
3. "Programmers" who are "perfect" in writing PHP or using script languages.
4. Talented students who solve the problem in special ways.
5. Advanced students who understand the problem, identify the algorithm and try to transform the general solution to the actual code.

Giving a problem with search as a sub-problem, there are only a few students who code the expected solution. The next group writes working code but the methods, or the applied algorithm is not correct. Furthermore, there is the third, usually the biggest group, who cannot implement the solution. After being shown the right code, most of the students can reproduce the solution but the next assignment raises the same problem again. Given more assignments, some students give up learning programming saying they will never be able to understand it. Another group of students give up learning in class because they think they can do it better on their own. In addition, there is the teacher with many questions: How to motivate students to learn, to think? How to motivate them to write clean code and structured programs?

Most teachers in Hungary, who teach programming, learnt programming and methodology at ELTE IK[1]. They learnt structured programming using the methodical/algorithm oriented method described in [1] first. They were adults and they had to have strong mathematical abstraction and logical skills. They teach mainly the way they learnt. One of the reasons why students cannot learn the same way is that they do not have advanced abstraction and logical skills.

[1] Eötvös Loránd University Faculty of Informatics (**E**ötvös **L**oránd **T**udomány**e**gyetem **I**nformatikai **K**ar)

Moreover, there is a hidden problem: what is the aim of teaching programming? Are students expected to write any working code or do they have to learn structured programming? The aim of public education is not to teach a profession but it cannot be independent of it. Two of the most prominent articles on this topic were written by Dijkstra [2] and Robert C. Martin [3]. Although, many famous programmers and scientists expressed their opinions, the learning path of linear search is different from practicing a profession in the didactical point of view. The problem of teaching linear search arises on the mailing lists and forums like the topic on CAS Community of teachers in UK: *Using break and continue in loops* [4], or threads on IT teachers' mailing list in Hungary: *Jumping out from loop with condition* [5], and *Programming decision* [6].

Seeing that the exact aim of teaching programming is not defined clearly in Hungary, we analyse the practice and contradictions and suggest methodological solutions.

2 Educators' Practice in Programming Linear Search

Experiencing problems in teaching linear search programming, we studied the taught coding practice. We explore the teaching method of basics of programming courses at ELTE IK [7] and BME-VIK [8]. They are the most important courses in Hungary, both having 500-600 students each year. The teaching practice of programming methods is very different in the two courses. The published comparative analysis [9] detects many differences. ELTE IK follows methodical/algorithmic- and specification-oriented method and develops a theoretical, mathematical knowledge. The Computer Science degree program focuses on programs and modelling. BME-VIK[2] combines more methods as it develops practical programming knowledge. The main methods are data-, problem-, language- and instruction-oriented. The Engineering Information Technology degree program develops programming skills.

Beside the written and taught programming principles, we explored the educators' programming practice. All educators were asked to do a programming assignment as they would do it for private use and write down every thought, the decisions and mistakes they have made while coding as a comment.

There were four groups:

A. educators who teach Basics of Programming at ELTE Computer Science faculty;
B. educators who teach Programming Fundamentals at BME-VIK Engineering Information Technology;
C. educators who teach System Modelling and Formal Methods at BME-VIK Engineering Information Technology;
D. secondary school teachers involved in the final exam of informatics on advanced level.

Knowing that the task enquired about the personal experiences of the programmes and the aim of the task was hidden, each answer is of high importance. They gave a valuable insight into programming and coding practice of which the programming methods of linear search are highlighted in the following.

The task was the same for groups A, B and C. It was a part of an earlier final exam about music channels[3]: 1. Read data of songs from file. 2. Determine the number of data records. 3. Find the first and the last occurrence of "Eric Clapton" and count the time elapsed. 4. Find the last occurrence of "Omega Legend" and tell the title of music on other channels played at the same time.

[2] Budapest University of Technology and Economics Faculty of Electrical Engineering and Informatics (**B**udapesti **M**űszaki és Gazdaságtudományi **E**gyetem **V**illamosmérnöki és **I**nformatikai **K**ar)

[3] http://dload.oktatas.educatio.hu/erettsegi/feladatok2006osz/emelt/e_info_06okt_fl.pdf Problem: 4. Zenei adók, parts: 1–4 [Accessed on: 28 June 2016]

Because the task was known for secondary school teachers, they got a similar but new task. Fortunately, the most recent task of final exam was usable. There were tasks about shopping baskets[4]: 1. Read data from file. 2. Determine the number of baskets. 3. List the products in the first basket. 4. Find a given product; give the first, the last basket and the number of baskets you find it in.

In the following two typical observations are given from every group (remarks and names of variables are translated):

A1: "I do not plan in advance but write code immediately as I read the tasks." However, the name of a variable "searched" shows the programming theorem was detected.

```
while (i < z && !(Music[i].channel == 1 &&
       Music[i].track.substr(0,sLength) == searched))
```

A2: "I choose C# because I like this language."… "I want to use LINQ to sum lengths"

```
char[] sep = { ':' };
int id = ch1.FindIndex(x=>x.id.Split(sep)[0] == "Eric Clapton");
```

B1: "I implemented calculating the song lengths in a separate function, however, to find the artist, there is a built-in which can be used"

```
for (n=0;strncmp(m[ch[0][n]].track,"Eric Clapton:",13)!=0; ++n);
```

B2: "Thinking about the optimal use of memory, I read the file sequentially again and again, because the problems given can be solved like this."

```
for (i=0; i < n; i++){
  fgets(b,78,f);
  /* … */
  if(!strcmp(tunes[x-1],"Omega:Legenda"))
  break;
}
```

C1: "It seems to me that R language would be the best choice"

```
first <- min(which((dat$perf=="Omega") & (dat$song=="Legenda")))
```

C2: "Is it required to test if the input is correct?" "I started to solve the problem in PowerShell." "Filter the list of songs for Eric Clapton."

D1:

```
for (int i = 0; i < custList.Count; i++){
  if (first == -1 && custList[i].Prod(prodname)) first = i;
  if (custList[i].Prod(prodname)){last = i; count++;}
}
```

D2a: First solution

```
int j = 0;
while (baskets[i,j] != null && baskets[i,j] != productName) j++;
```

D2b: "But this is more elegant:"

```
if (baskets[i].Contains(productName))
```

We can see that there are many solutions using the advanced methods of the programming language instead of writing search algorithms. Other solutions were based on selection, which derives from the fact that the task asked for both first and last occurrence. Educators of BME mostly wrote `for()` loops for search and sometimes used `break` statements. Educators of ELTE and teachers who searched for first and last occurrence separately used `while()` loops. The latter two constructs highlight the difference in the way of thinking of the two groups:

[4] http://dload.oktatas.educatio.hu/erettsegi/feladatok_2016tavasz_emelt/e_inf_16maj_fl.pdf
Problem 4. Ötszáz" [Accessed on: 28 June 2016]

simplified pragmatic and algorithm-driven. The third one deducts the problem to the selection of advanced tools.

3 Typical Solution of Linear Search Written by Students

There are different methods to teach and learn programming but every method and every learning path contains search tasks. We observed secondary school students who had learnt the basic control flow statements. They had been taught some programming theorems and solved a few tasks in group-work. Students had to solve some problem together and they were helped by the teacher. Those who were successful in self-employed coding were asked to work alone at the start of the search tasks. Thirty students were involved in this experiment.

Though some of them had learnt linear search algorithm and determination of the type of task before, none of them noticed that the task is an example of search.

The true beginners asked for some help on how to start coding. They hesitated whether `for()` or `while()` loop or maybe `if()` would be the best choice. Only two of them started with writing `if()` others started with a `for()` loop. Five of them selected or counted all appropriate items. Later they used a variable to store the answer. The main part of their solutions:

S1:

```
for (int i = 0; i < list.Count; i++ )
  if («logical_expr»)list2.Add(item);
```

S2:

```
for (int i = 0; i < list.Count; i++)
  if («logical_expr») counter++;
```

Though they got the result somehow, they understood their solutions were not optimal.

Other seven students were found who stopped at the body of the `if()` and asked what to do at this point. As they were not separated from classmates, three of them got help from advanced programmers. At this state, all classmates suggested the `break` instruction. "Wow, it works!" Like a miracle cheat code, the use of `break` instruction spread among other students. In this way, twelve students finished their code as follows:

S3:

```
for (int i = 0; i < list.Count; i++)
  if («logical_expr»){answer = i; break;}
```

There were only four students who got help from the teacher while thinking about what to write into the body of `if()`. They learnt how to transform the `«logical_expr»` into `for()` statement:

S4:

```
int i;
for (i = 0; i < list.Count && !(«logical_expr»); i++);
```

It has to be noted that these students did this transformation without any prior practice in negation of Boolean expression. They learnt it as formal transformation, which they proved in details. They understood the code, but could not express it in spoken words.

Two of them were open to more code transformation. Their final version was like this:

S5:

```
int i=0;
while (i < list.Count && !(«logical_expr»))
  i++;
```

However, as an extra advantage, the advanced programming students, who helped their classmates earlier, learnt this transformation as well.

Many students learn programming in their extra lessons. The most popular topics are developing games on a mobile or on the web. They learn how to use predefined tools. They solve search problems very fast, because they use the built-in tools of the language:

S6:

```
var solution = list.Contains(«logical_expr»);
```

Teachers need to make extra effort to explain to these students why it is important to know how the `Contains()` method works, why they should implement their own `Contains()` function.

Interviewing students who had learnt `while()` loop before, about why they do not use the `while()` loop instead of `for()`, they mentioned two main points:

1. The task was similar to selection
2. The condition of `while()` loop was too difficult, using `for()` loop is more handy to use and the condition of `if()` is easier to write separately.

Examining what “difficult” and “handy” mean in this context we found that the spoken language influences the programmer's thinking. When talking about searching in Hungarian, we “scan the elements to the end”. The same method in English is expressed as “sequentially check each element“. Both expressions include the meaning of “from the first to the last”. That is the reason why students feel more confident using `for()` rather than `while()`. The popularity of `break` instruction also has a linguistic reason. The checking is in progress “until a match is found”. The task in the body of `for()` loop is “if a match is found…”. Learners see that “is found” does not involve the negation of the condition, but must break the further checking, the further scanning. They ask for an escape, and the `break` instruction solves their problem. Although teachers and professionals know that `break` instruction is not advised in many situations, years of practice are needed for the students to understand this.

Eliminating the `break` instruction is a coding technique, a formal rewrite of code. It includes the negation of the “is found” condition. Inserting this condition into the head of `for()` loop, the form of logical expression is “not end AND not found”, where “not found” could be a complex logical expression. In order to understand the resulting expression one has to practice the use of De Morgan’s laws. Should they accept them blindly and use formal transformation without actually understanding what they do? We have to understand that implementing the linear search algorithm in a structured manner (ie. without the `break` instruction) requires high-level thinking skills in mathematical logic. At this point, we have to note that learning mathematical logic is part of Maths curriculum but typically later and more formal than it needs while learning programming. Therefore, the mathematical logic and the De Morgan’s laws should be a main part of Informatics curriculum.

The complexity of the condition could be reduced by implementing it in a separate function. This way the “not found” would mean an evaluation of the logic function. However, this solution expects the learning of function’s declaration earlier than practicing problem solving using linear search. This learning path also has many pitfalls related to function argument passing, references, global and local variables.

4 Developing Step by Step or Stop

Students who know and like to use built-in search methods (S6) say that using these methods is easier and faster. Professionals say built in methods are practical (A2, C1, C2), teachers say it is elegant (D2b). In the teaching context, this solution could be the first state of exploring (analysing) the linear search algorithm, or the last state when learners can implement their own searching algorithm.

The main steps and key stages of learning path of linear search are as follows:

1. The first step is to understand the task and the meaning of the algorithm. The solution of the search problem is to find an application, a tool that implements this algorithm and gives the proper answer. In this sense using MATCH() function of spreadsheets, write an SQL query, use built-in methods or dedicated software are similar.
2. The second step is to code an algorithm, which solves the search problem. In this learning phase, any algorithm is acceptable which gives the right answer. (B2, S1, S2, S3) The suboptimal selection or counting algorithms and using the `break` instruction are also acceptable.
3. The third step is to learn how to write structured and optimal implementation. Depending on the programming language, the usage of control statements could be different (A1, B1, B2, D1, S4, S5). The key task here is the development of logic skills. The students have to practice the negation of Boolean expression and the application of the De Morgan's laws.
4. The fourth step is the synthesis of linear search problems. Learners recognise and write general-purpose functions for linear search.

The learning process could start in the elementary school by analyses and end on professional level with synthesis. Unfortunately, having reached each level, students' answer to the question "Can you solve linear search problem" is "Yes, I can." Therefore the next step of development is not motivated, the learning process breaks. However, we believe that each step is necessary, and omitting any of them can result in shallower understanding and worse programming skills.

On the other hand, linear search is mentioned in public educations' curriculum but the level of knowledge is not defined. It means the curriculum's criteria are performed without knowing the required level. Therefore, the teaching-learning process of programming could stop on the first level in context of linear search, and that is a lower level than the required one for an educated person in a digital society.

We have to make it clear that learning linear search includes the improvement of the most important thinking skills, like logic thinking and computational thinking. Therefore, the step to the third level is highly recommended for every member of public education but it expects prepared logic skills, time and effort to practice thinking.

The stop at the second level could result in a huge crowd of home-coders, who write suboptimal and unreadable code, and that is really harmful for the programmers' profession. The next example is a snapshot of a program that was written by a home-coder student and demonstrates the unlimited using of `break`. The task is to write the distance travelled of a tour guide's first workday on a week to the console:

```
private static void first() {
  for (int j = 1; j < 8; j++) {
    for (int i = 0; i < works.Count; i++) {
      if (works[i].day == j && works[i].id == 1) {
        Console.WriteLine(works[i].distant);
        break;
      }
/* first try, but I can compact the code
      if (works[i].day == 2 && works[i].id == 1) {
        Console.WriteLine(works[i].distance);
        break;
      }
```

```
        if (works[i].day == 3 && works[i].id == 1) {
          Console.WriteLine(works[i].distance);
          break;
        }
*/
      }
    }
}
```

The first thoughts: "I check each element and if I find that the day equals 1, I finish. Maybe the tour guide had holidays; I have to do the same with the 2nd, 3rd… days. I can compact it into a loop. The loop of days should be the outermost one." After testing this code the student made more changes, added a new Boolean variable and one more `break`.

We must also note that the unreadable, messy nature of a code snippet, which contains the `break` instruction, is not always caused by the `break` instruction itself. Consider the example below. The next student wanted to implement some kind of a cache, in which elements are to be inserted if they are not already present:

```
for (int i = 0; i < cacheditems.size() or
(cacheditems.push_back(make_pair(item, 1)), false); i++) {
  if (cacheditems[i].first == item){
    cacheditems[i].second++;
    break;
  }
}
```

The code misuses the shortcut behaviour of the logical or operator, to make sure that the new element is inserted to the container only once, even though the `push_back()` call is inside the loop, moreover in the loop condition. This is a clear violation of the command-query separation principle [10], which can also be applied to this very simple algorithm as well. Any programmer expects a `for()` loop to look and work like this:

```
for (command; query; command)
```

Simply because by convention, the second expression should not have any side-effect. However, the `for()` loop of the student worked like this:

```
for (command; command; command)
```

This is the most prominent mistake here, not the usage of the `break` instruction. The code could easily be rewritten like this:

```
int i;
for (i=0; i < cacheditems.size(); i++)
  if (cacheditems[i].first == item)
    break;
if (i < cacheditems.size())
  cacheditems[i].second++;
else
  cacheditems.push_back(make_pair(item, 1));
```

The revised version still has a `break`, but it is now quite readable, and the subproblems (finding the item in the cache and then modifying it accordingly) are coded separately. We asked the student, what the idea behind this solution was:

"I don't like `if()` after the loop, I want to give the answer in the loop. I found a similar code snippet on the web:

```
fopen($file) or die("500 Internal Server Error");
```

I like it. What is the problem with this solution?"

In cases like the above, simply forbidding the students to use the `break` instruction will not help their understanding. The problem is not the misuse of the `break`, but the lack of knowledge of general code organizing methods and idioms, which are not related to the `break` instruction at all.

Conclusion

Exploring the teaching-learning process and using linear search, we found that the learning practice breaks along the quality of problem solving: to find a solution, to code a runnable program, to write optimised code, to code a general solution. The use of `break` statement seems to be necessary at a definite point of the learning process, because it expresses a novice programmer's way of thinking. However, over-using the `break` instruction is harmful, therefore learning (and teaching) higher level of programming skills is highly recommended not only for professional training but for public education, too. Coding linear search improves not only programming knowledge but also thinking skills, where the higher levels of solutions involve higher level of logical, mathematical and computational thinking. To get higher level of skills is a qualitative step, which requires the teaching of the additional concepts and principles. Therefore, the aim of public education in context of linear search should be that students are able to write a readable and optimized code, according to the rules of structured programming.

Bibliography

[1] SZLÁVI, P. and ZSAKÓ, L. *Methods of teaching programming.* In: Teaching mathematics and Computer Science, vol. 01(2), pp. 247–258., 2003. DOI: 10.5485/TMCS.2003

[2] DIJKSTRA, E. W. *Letters to the Editor: Go to Statement Considered Harmful.* In: Communications of the ACM, vol. 11, no. 3, pp. 147–148, 1968. DOI: 10.1145/362929.362947

[3] MARTIN, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship*. NJ, USA, Prentice Hall PTR, 2008. ISBN: 978 013235088 4

[4] CAS Community *Using break and continue in loops*. Computing at School, 06 2014. [Online]. Available: http://community.computingatschool.org.uk/forums/63/topics/2835 [Accessed on: 28 June 2016]

[5] Informatika tanári levelező lista *Kiugrás feltételes ciklusból*. Sulinet, 04 2016. [Online]. Available: http://lista.sulinet.hu/mailman/private/informatika/2016-April/021186.html. [Accessed on: 28 June 2016]

[6] Informatika tanári levelező lista *Programozás eldöntés*. Sulinet, 05 2016. [Online]. Available: http://lista.sulinet.hu/mailman/private/informatika/2016-May/021668.html. [Accessed on: 28 June 2016]

[7] PAPP, G., HORVÁTH, G., SZLÁVI, P. and ZSAKÓ, L. *Programozási alapismeretek 4. előadás*. ELTE IK, Budapest, 2015. [Online]: http://progalap.inf.elte.hu/downloads/eloadas/progalap_ea4.zip [Accessed on: 28 June 2016]

[8] CZIRKOS, Z. *Programozási tételek*. BME EET, Budapest, 2015. [Online]: https://infoc.eet.bme.hu/ea03/ [Accessed on: 28 June 2016]

[9] SZALAYNÉ TAHY, Zs. and CZIRKOS, Z., *"ProgAlap" és ami mögötte van.* 8. InfoDIDACT, Zamárdi: Webdidaktika Alapítvány, 2015. ISBN: 978 963123892 1

[10] MEYER, B. *Object-oriented Software Construction (2Nd Ed.),* NJ, USA: Prentice-Hall, Inc., 1887. ISBN: 0-13-629155-4

Lectured by: Sándor Király, PhD, associate professor

Contact address:

Zsuzsanna Szalayné Tahy

Department of Media and Educational Informatics, Faculty of Informatics, Eötvös Loránd University, H-1117 Budapest, Pázmány P. sétány 1/C, Hungary,

e-mail: sztzs@caesar.elte.hu

Zoltán Czirkos, Ph.D,

Department of Electron Devices, Faculty of Electrical Engineering and Informatics, Budapest University of Technology and Economics, H-1117 Budapest, Magyar tudósok körútja 2.

e-mail: czirkos@eet.bme.hu