BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **16**, No 4
Sofia • 2016 Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.1515/cait-2016-0068

Iteration vs Recursion in Introduction to Programming Classes: An Empirical Study

Vladimir Sulov

Department of Computer Science, University of Economics – Varna, 9002 Varna, Bulgaria
Email: vsulov@ue-varna.bg

Abstract: *In this article we have presented the results of an empirical study which was carried out on 130 students in Introduction to programming classes. Their initial preference, success rate, comprehension and subsequent preference were studied when dealing with programming tasks which could be solved using either iteration or recursion.*

Keywords: *Recursion, iteration, introduction to programming, education, students.*

1. Introduction

In computer programming the expression of computations that are not finite and determined statically (for example finding the sum of a vector consisting of *n* elements, where *n* is not known at the time of writing the program) is necessary in order for the programming language to be Turing complete and to be able to express all possible algorithms. In today's high-level programming languages this is achieved by two basic mechanisms: Iteration and recursion [2].

Iteration is "the repetition of a sequence of computer instructions a specified number of times or until a condition is met" [10]. The mechanism of iteration is used when there's need for some action to be performed a number of times based on certain conditions [5]. The classic mechanism of constructing iterations in programming code is done through so called cycles or loops.

On the other hand, recursion is a "computer programming technique involving the use of a procedure, subroutine, function, or algorithm that calls itself one or more times until a specified condition is met at which time the rest of each repetition is processed from the last one called to the first" [10].

As noted iteration and/or recursion are necessary mechanisms for programming languages and virtually all applicable computer programs use one or both of these mechanisms inside their code. Therefore, a student studying programming must be able to devise algorithms using iteration and/or recursion. On the other hand, devising algorithms, especially requiring complex cycles, turns out to be one of the most difficult tasks for the novice programmer.

In this paper we present an empirical study on students' preferences and abilities to devise, write and comprehend algorithms and programming code based on iteration and recursion, as this is essential for developing their professional skills.

2. Methodology

The experiment was carried out during three years. The experiment test subjects were 130 students taking their first programming course called "Introduction to programming". The programming language used was C. As it is an imperative and also a structured and procedural language, C supports both concepts of iteration and recursion [4, 8], and could be successfully used for comparing them, while "in many functional and logic languages there does not, in fact, exist any iterative construct" [2].

Previous studies showed mixed results but also had different target groups and questions. Generally, students have difficulties with recursion [3, 6, 7], especially identifying base cases [3] and thus some researchers argue that "it is sensible pedagogical practice to base understanding of recursive flow of control on understanding iterative flow of control" [6]. Some other researchers, though, reach the opposite results [11] and one study even concluded that students favor recursion when doing a task such as searching a linked list for a given value and comprehend better recursive code for this and a similar task [1] although the latter was in turn contradicted by a similar more recent study [9].

Our experiment was conducted during the final days of the course so all students were already familiar with the basics of programming in C, data types, flow control statements, arrays, structures, user functions, etc. and also with the concepts of iteration and recursion doing several practical tasks. On the other hand, as this was strictly an introductory course, the students did not deal with more complex data structures and algorithms as for example managing a linked list.

The students were given four practical tasks, all of them requiring some repetitive code ranging from a very simple one to a moderately, or even quite complex task, given their lack of experience. The students were free to choose how to write the code, and no mention of either iteration or recursion was made. Therefore, our first goal was to study their initial preferences when doing tasks of different difficulty. Our second goal was to find how successful they were with their choice and generally the given task. After all the tasks were completed, students were shown both iterative and recursive solutions for each task which were explained and discussed and were asked whether they comprehend the solutions and once again what their preference would be in a similar future situation/task, i.e. they had the chance to change their opinion afterwards. As not all 130 students were present during all tasks, the data was filtered and only the results of 103 students that were participating in each exercise will be given and used as to avoid error.

3. Tasks and solutions used

All tasks (and the course itself) targeted console applications with input from the keyboard and output to the console (text screen). Apart from sequential numbers, the tasks were also given “code names” as to distinguish them semantically.

The first task was an exceptionally basic and easy one.

Task 1. Numbers. Write a program that takes a positive integer *n* and displays all numbers from 1 up to *n*.

Example: 5 → 12345

The following iterative solution was given to the students (in order to make code shorter, no input validation was performed, the lines with the necessary headers were removed and only actual code was left for the paper):

```
int main(void)
{
    int i, n;
    scanf("%d", &n);
    for(i=1; i<=n; i++)
        printf("%d", i);
    return 0;
}
```

The following recursive solution was given:

```
void loop(int current, int total)
{
    printf("%d", current);
    if(current==total) return;
    else loop(current+1, total);
}
int main(void)
{
    int n;
    scanf("%d", &n);
    loop(1, n);
    return 0;
}
```

Task 2. Factorial. Write a program that takes a positive integer *n* then calculates and displays its factorial.

Example: 5 -> 120

Iterative solution:

```
int main(void)
{
    int i, n, factorial=1;
    scanf("%d", &n);
    for(i=1; i<=n; i++)
        factorial*=i;
    printf("%d", factorial);
    return 0;
}
```
Recursive solution:
```
int factorial(int n)
{
   if(n==0) return 1;
   else return n*factorial(n-1);
}
int main(void)
{
   int n;
   scanf("%d", &n);
   printf("%d", factorial(n));
   return 0;
}
```
Task 3. Fibonacci. Write a program that takes a positive integer *n* then calculates and displays the *n*-th number of Fibonacci. Note: we'll define the first two numbers of the Fibonacci sequence as 1 and 1 and each subsequent number is the sum of the previous two, so the sequence is 1, 1, 2, 3, 5, 8, 13 …

Example: 6 -> 8

Iterative solution:
```
int main(void)
{
   int i, n, a=1, b=1, fib=1;
   scanf("%d", &n);
   for (i = 3; i <= n; i++) {
            fib = a + b;
            a = b;
            b = fib;
   }
   printf("%d", fib);
   return 0;
}
```
Recursive solution:
```
int fib(int n)
{
   if (n == 1 || n == 2) return 1;
   else return fib(n - 1)+fib(n - 2);
}
int main(void)
{
   int n;
   scanf("%d", &n);
   printf("%d", fib(n));
   return 0;
}
```

Task 4. Permutations. Write a program that takes two positive integers *length* and *n* then generates and displays all possible permutations each on a new line with repetition of the numbers from 1 to *n* of size *length*.

```
Example: 2, 3 ->   1 1
                   1 2
                   1 3
                   2 1
                   2 2
                   2 3
                   3 3
```

Iterative solution:

```
int main()
{
   int length, n, i;
   scanf("%d%d", &length, &n);
   int *combo = new int[length];
   for (i = 0; i < length; i++) combo[i] = 0;
   while (combo[0] < n) {
         for (int i = 0; i < length; i++) {
               printf("%d ", combo[i]+1);
         }
         printf("\n");
         combo[length - 1]++;
         for (int i = length - 1; combo[i] == n && i > 0; i--) {
               combo[i - 1]++;
               combo[i] = 0;
         }
   }
   delete[]combo;
   return 0;
}
```

Recursive solution:

```
void generatecombo(int* combo, int pos, int length, int n) {
   int i, j;
   for (i = 0; i < n; i++) {
         int *newCombo = new int[length];
         for (j = 0; j < length; j++)
               newCombo[j] = combo[j];
         newCombo[pos] = i+1;
         if (pos < length) {
               generatecombo(newCombo, pos + 1, length, n);
         }
         else {
               for (j = 0; j < length; j++)
                     printf("%i ", combo[j]);
```

```
                printf("\n");
                delete []combo;
                return;
            }
        }
    }
    int main()
    {
        int length, n;
        scanf("%d%d", &length, &n);
        generatecombo(new int[length], 0, length, n);
        return 0;
    }
```

4. Results

As previously noted, the results are based on the answers and actions of 103 students that participated in *each* exercise. The summary of results is shown on Table 1. Tasks are ordered the same way they were given to students, which is from the simplest to the most difficult one (as also observed and confirmed by the success rate).

Table 1. Experiment results

Task	Initial preference		Success rate		Comprehension		Subsequent preference	
	Iteration	Recursion	Iteration	Recursion	Iteration	Recursion	Iteration	Recursion
Numbers	100%	0%	97%	n/a	100%	88%	98%	2%
Factorial	79%	21%	83%	86%	90%	86%	73%	27%
Fibonacci	66%	34%	68%	74%	78%	81%	42%	58%
Permutations	68%	32%	18%	28%	44%	47%	53%	47%

4.1. Initial preference

Generally speaking, the majority of students initially always favor seeking of iterative solutions over recursive ones. Still, the more difficult the task is, the more probable they would prefer to find a recursive solution.

The lowest share of recursion preference (no students) was observed on the “Numbers” task, the iterative solution of which is just a straightforward loop. The greatest share of initial favoring recursion was observed when solving the “Fibonacci” task, which can probably be explained by the fact that the Fibonacci sequence can be described as a typical example of a recursive sequence: the first two members of the sequence are 1 and 1 (base cases) and each subsequent Fibonacci number is the sum of the previous two (recursive description). Still, the proportion of students favoring recursion was relatively too high so further study was carried out (see Section 5 below).

4.2. Success rate

Success rate and its trend across tasks were as expected.

The “Numbers” task is a very simple one, with almost all students successfully solving it.

The “Factorial” task is just a bit more difficult and even so, a trend started to emerge: the more difficult the task, the more relatively successful were students who preferred a recursive solution which is most evident in the last “Permutations” task.

4.3. Comprehension

After students (both successful or not in solving a specific task) were shown standard iterative and recursive solutions, which were also explained and discussed, they were questioned whether they comprehended the respective solutions.

The most interesting find in this regard is that students almost equally well comprehend iterative and recursive solutions although to some extent this is due to the fact that most tasks were relatively simple.

4.4. Subsequent preference

The final question was whether students would change their preference after seeing successful solutions of both kinds.

We observed a tendency of increased subsequent preference of recursion compared to the initial preference. This could probably be the result of the recursive solutions being shown and explained to students, who, in turn, comprehended them quite well. This also demonstrates that students do not always correctly identify the cases for using recursion.

5. Some thoughts and an additional experiment

As our initial experiment was carried out on students taking their first course in programming during their first year at the university, the results could be somewhat influenced by limiting factors such as:

- the difficulty of tasks: only relatively simple tasks could be tested which can be easier to identify as probable cases for iteration or recursion;
- although no recursion or iteration was mentioned explicitly during the tests, students had been introduced to the concept of recursion during the same course (approximately a month before the tests) which may have impacted their thoughts and consequently their initial preference.

In this regard, the relatively high percentage of students favoring recursion (between 21 and 34%), especially in an introductory course and using the C programming language, could be justified by the above given factors.

Therefore, afterwards, in order to further test the students' abilities in this sphere, and to try to explain the results, a new experiment was devised and carried out.

Out of the original 103 students that took part in all initial exercises, 57, who were later in their 3rd or 4th (final) year of their bachelor study, were tested again. This time the students had already passed all of their core programming courses (including "Introduction to programming", "Object-oriented programming", "Algorithms and data structures") so they were more experienced with tasks of greater difficulty and scope. The actual courses during which the new experiment was carried out were about developing desktop and web applications using C# and the .NET-platform so they were not directly connected to algorithms and recursion/iteration respectively.

The new tests were "disguised" as ordinary tasks so students would not suspect that any kind of special observation was made and there was no mention of either recursion or iteration. Thus, only initial preference and success rate were tested (and sample solutions using both approaches were not given to the students). Two tasks were selected which are amenable respectively to an iterative and to a recursive solution:

Task 1. Prime numbers. Write a program that takes a positive integer *n* from a text box and then generates all prime numbers from 1 to *n* which should be filled in a dropdown list.

This task was "disguised" as being a "refreshment" of students' basic skills working with the GUI controls of the specified language/platform. The prime number generation by itself is a typical iterative task.

Task 2. List the content of a directory and its subdirectories ("Directory traversal"). Write a program that makes the user select a directory by clicking on a button and then list all of its subdirectories and files in each directory/subdirectory in no specific order in a dropdown list.

This task was "disguised" as a practice for students to learn how to deal with directories and files using C#/.NET. Indeed, it is a typical recursive task, similar to the classical "depth-first search" traverse algorithm.

The observed results are presented in Table 2.

Table 2. The additional experiment results

Task	Initial preference		Success rate	
	Iteration	Recursion	Iteration	Recursion
Prime numbers	100%	0%	84%	n/a
Directory traversal	74%	26%	23%	64%

The results concerning the "Prime numbers" task are as expected. None of the students chose a recursive approach.

At the same time, the results on the "Directory traversal" task came in somewhat unexpected. A very large proportion of the students once again chose to solve the task iteratively. If we compare the proportion of students that chose the recursive approach (26%) to the results obtained from the initial experiments (between 32 and 34% for typical recursive tasks), we could note that although the students were now more experienced, without guidance or recent iteration/recursion/data structures discussion, they incorrectly chose the more difficult approach to the specific task. Unfortunately, this questions students' ability to identify situations/tasks amenable to recursion especially when they are not abstract scenarios but real-life application requirements. Moreover, when students choose a more difficult approach, their success rate is greatly reduced which may also indicate a correlation to students' general abilities (see the conclusion below).

6. Conclusion and further studies

We can conclude that students who are novice in programming prefer iteration over recursion in most cases and prefer recursion over iteration only when there are clear indications. General comprehension of recursion is good although introductory classes and tasks are relatively simple. The results show though, that as students gain more experience, and move from abstract theory to actual software applications, they do not always identify adequately the possible cases where recursion should be preferred. In this regard, perhaps extra effort should be made on not only teaching the principles of recursion but the principles of how to identify typical recursive scenarios.

Possible further studies would be to try to correlate the students' general abilities (not as a whole, but individually, for example based on their success rates of the specific tasks and/or final grade) to their initial and subsequent preference and success rate. Thus we could try and find out not only the results when studying iteration/recursion but also more of the reasons for the preference, comprehension and success of students. Another possible further study could be to try and analyze the influence of the programming language used (or first/predominantly taught) on students' abilities to correctly identify possible recursive scenarios.

References

1. B e n a n d e r, A., B. B e n a n d e r, H. P u. Recursion vs. Iteration: An Empirical Study of Comprehension. – Journal of Systems and Software, Vol. **32**, 1996, Issue 1, pp. 73-82.
2. G a b b r i e l l i, M., S. M a r t i n i. Programming Languages: Principles and Paradigms. Springer Science & Business Media, 2010.
3. H a b e r m a n, B., H. A v e r b u c h. The Case of Base Cases: Why Are They So Difficult to Recognize? Student Difficulties with Recursion. – In: Proc. of 7th Annual Conference on Innovation and Technology in Computer Science Education, ACM, New York, 2002, pp. 84-88.
4. K a m t h a n e, A. Introduction to Data Structures in C. Pearson Education, India, 2004.
5. K a m t h a n e, A. Programming and Data Structures. Pearson Education, India, 2003.
6. K e s s l e r, C., J. A n d e r s o n. Learning Flow of Control: Recursive and Iterative Procedures. – Human-Computer Interaction Archive, Vol. **2**, 1986, Issue 2, pp. 135-166.
7. L e w i s, C. Exploring Variation in Students' Correct Traces of Linear Recursion. – In: Proc. of 10th Annual Conference on International Computing Education Research, ACM, New York, 2014, pp. 67-74.

8. L o u d o n, K. Mastering Algorithms with C. O’Reilly Media, Inc., 1999.
9. M c C a u l e, R., B. H a n k s, S. F i t z g e r a l d, L. M u r p h y. Recursion vs Iteration: An Empirical Study of Comprehension Revisited. – In: Proc. of 46th ACM Technical Symposium on Computer Science Education, ACM, New York, 2015, pp. 350-355.
10. Meriam-Webster Dictionary. **http://www.merriam-webster.com.**
11. M i r o l o, C. Is Iteration Really Easier to Learn Than Recursion For CS1 Students? – In: Proc. of 9th Annual International Conference on International Computing Education Research, ACM, New York, 2012, pp. 99-104.