Basics of Indirect Programming

Olegs Verhodubs
oleg.verhodub@inbox.lv

Abstract. The human worldview is firmly based on the notion that every action leads to a certain result. That is, in order to achieve a certain result, you need to perform some action. However, in the real world, which is always different from the model of the world in a person's head, each action is followed by not only a result, but also some side effects. Practice shows that side effects can be used to achieve the goal no less effectively than the main actions. Often, using side effects, you can achieve even more than using the main actions. Side effects are a special case of indirect actions. It is indirect actions applicable to programming that will be considered in this paper. That is, indirect programming, namely the use of roundabout ways to achieve the goal of the program, is considered in this paper.

Keywords: Programming, Indirect Programming, Strategy

I. Introduction

A human cannot live outside of society, and society forms the basic ideas in humans about the world around them. Society is the bearer of collective experience, and it is primarily concerned with its own survival, therefore it is extremely interested in the opportunity to influence the world around. The idea that a certain action leads to a certain result is a traditional idea, repeatedly tested by time. Based on this idea, it is believed that any goal can be achieved if a series of intermediate steps is made from the initial state to the goal. This idea is relentlessly exploited in programming, where every operator produces a strictly defined result. However, life is immeasurably richer than if it were based only on the principle that an action leads to one and strictly defined result. In real life, any action has a list of consequences, some of which are immediately detected, while others are not.

The prevailing idea of the role of a certain action to obtain a certain result is traditional for our society. More precisely, this is a traditional representation for European society. And to be even more precise, this is characteristic of a society nurtured in the European tradition. However, there is also an opposite point of view, which does not reduce efficiency to purposefulness and effectiveness [1]. Obviously, the idea stems from an ancient Chinese source that describes strategies for winning military operations [2]. The fact that goals can be achieved in a different way has also become clear to European societies as experience accumulates and science develops. B. H. Liddell Hart was one of the scientists that introduced the concept of indirect actions to achieve goals (mainly in war) and fixed this in his work [3]. Probably, the question of indirect actions to achieve a result was investigated in more detail in systems theory. For example, some sources associated with systems theory mention the usefulness of side effects and even that side effects may be the main result [4].

In programming, indirect actions (commands, operators, operations) are not formally defined, although they are widely used in practice. There are many ways to achieve a certain result by implementing some algorithm using indirect actions, although the same can be implemented traditionally, through ordinary direct actions or commands. It is important to understand that indirect actions in programming are not an end in themselves, but on the contrary, they are often more effective than traditional, direct actions and therefore are used. Since indirect actions in programming, as well as indirect programming in general, has not yet been investigated, this is done in this paper. To be honest and objective, it should be clarified that the very concept of indirect programming was not defined, and its types were not described, although the methods of indirect programming themselves were used almost from the very beginning of the advent of computer technology. The merit of this paper is that it fills a gap in terminology, and also outlines the boundaries of an area that has great prospects as the volume and, as a consequence, the functionality of computer programs increases. In addition, the rich functionality of computer programs provides many opportunities for indirect programming.

This paper is organized into several sections. The first one provides the necessary information on terminology in indirect programming, the next one gives specific examples of indirect actions in programming and the last one provides a classification of indirect actions in programming.

II. Terminology

It is necessary to introduce clear and precise definitions of concepts in order to be able to operate with them. Clear, precise and unambiguous definitions are generally accepted space i.e. a space of consensus, serving as confirmation that we are talking about the same thing. First, it is necessary to define indirect actions in programming and indirect programming in general. The concept of indirect actions in programming semantically follows from the concept of indirect actions in general, and these definitions are related as the particular to the whole. In turn, in order to understand and define what indirect actions are, you must first understand and define what direct actions are.

So, direct actions in general are actions addressed directly to an object. Indirect actions in general are actions that are not directly addressed to the object, but the object receives the impact. Speaking about direct and indirect actions in general, we mean all possible actions available in real life. When we use the word "actions" in relation to programming, we mean commands, instructions, or expressions. Here, commands, instructions or expressions are synonyms. So, direct actions in programming are actions addressed directly to an object too, where the object is not material, but digital (digital object is a variable of any type). Indirect actions in programming are such actions that are not directly addressed to the digital object, but it receives some impact, or these are actions that are directly addressed to the digital object in order to obtain not the result of direct impact, but some unobvious consequences. Here unobvious consequences are side effects, that is, secondary (not main) or additional consequences. The definition of side effects here is different from the generally accepted concept of side effects in programming. Generally accepted concept of side effects in programming implies any changes in the state of the execution environment [5]. A typical example of this is an expression in the C programming language like this [6]:

```
x[i] = i++;                                  (1)
```

In this example (1), the value of x that is modified is unpredictable [6]. The value of the subscript could be either the new or the old value of i. The result can vary under different compilers or different optimization levels [6].

An alternative definition of side effects in programming, or more precisely in relation to functions, can be as follows: “A side effect is when a function relies on, or modifies, something outside its parameters to do something. For example, a function which reads or writes from a variable outside its own arguments, a database, a file, or the console can be described as having side effects" [7]. This second definition can be illustrated with the following figure (Fig.1.):

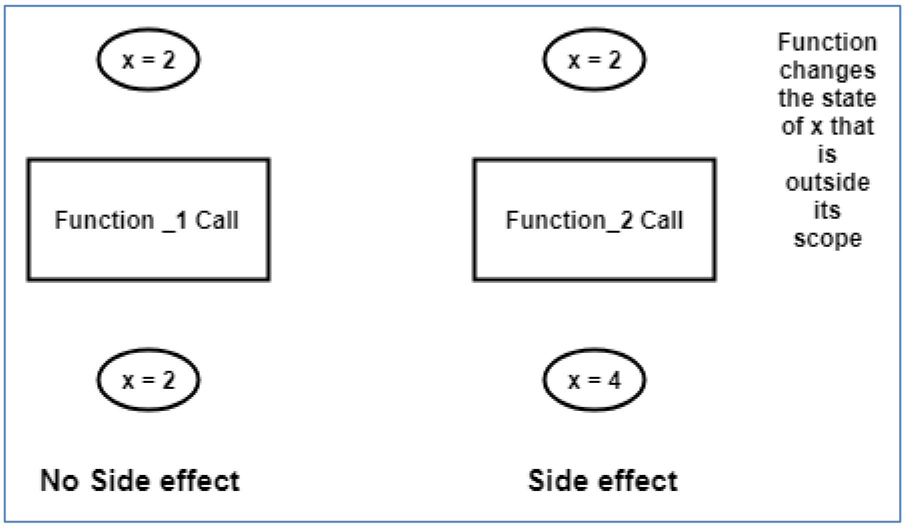


Fig.1. Side effect produced by the function [8]

Thus, this research proposes to expand the use of the term side effects in programming, extending this term to non-main (non-obvious), unusual and rarely used reults in programming languages.

It is possible to formulate what are indirect programming as soon as it is known what indirect actions in programming are. Indirect programming is programming by means of indirect actions. Or in other words, indirect programming is the use of means that are not directly aimed at digital objects of interest in order to exert influence, but for this purpose other objects are used through which the influence is exercised, or they are directed at the digital objects themselves of interest, but they are not primarily interested in the result of the influence, but in the side effects of this influence. These definitions of indirect programming do not have quantitative measure of the use of indirect actions, i.e. to what extent indirect actions are used for programming to be called indirect, but here it is all a matter of proportion: if there are more indirect actions, then such programming can safely be called indirect.

III. Examples

There are a lot of programming languages. They began to develop since the advent of computers, which happened many decades ago. A large number of programming languages has led to the emergence of different classifications, which conditionally group them according to different criteria. Programming languages can be classified according to the purpose for which they were created, according to programming paradigm that is used in a given programming language, and programming languages are also classified into interpreters and compilers. Here the examples will be carried out in high and low level languages. The C++ and Javascript programming languages are used as a high-level language for examples, and Assembler for x86 processor is used as a low-level language. The division of programming languages into high- and low-level languages is another classification. Classification of programming languages into high-level and low-level languages is preferable to demonstrate that both high-level and low-level programming languages have opportunities for indirect programming.

Even a low-level programming language like Assembler has built-in capabilities for indirect programming (Table I).

TABLE I. Comparison of direct and indirect programming in assembler.

Nr	Direct programming	Indirect programming	Explanation
1	mov ax, 0	xor ax, ax	clearing (zeroing) a processor register ax
2	mov cx, ax mov ax, bx mov bx, cx	xor ax, bx xor bx, ax xor ax, bx	XOR swap algorithm [9]
3	mov ax, 5	mov bx, 5 push bx pop ax	put value 5 into register ax
4	mov bl, 2 mul bl	mov ax, bx shl ax, 1	ax = bl * 2
5	a dw 123h mov ax, a	a dw 123h mov bx, offset a mov ax, [bx]	loads ax register with contents of a; “mov bx, offset a” is possible to replace with “lea bx,a” [10]

All these capabilities without working with stack of the program (nr.3, Table I) are available in high-level languages like C/C++, too. High-level programming languages provide greater indirect programming capabilities due to wider syntax possibilities than Assembler has. For example, some indirect programming cases expressed in C/C++ programming language are listed below (Table II):

TABLE II. Comparison of direct and indirect programming in C++.

Nr	Direct programming	Indirect programming	Explanation

1	int a=5, b=7, c=0;
	c=a+b;
		int a=5, b=7, c=0;
		c=a - (-b);
			sum of a and b

2	int Add(int x, int y)
	{
	if (y == 0)
	return x;
	else
	return (x + y);
	}
		int Add(int x, int y)
		{
		if (y == 0)
		return x;
		else
		return Add(x ^ y, (x & y) << 1);
		}
			sum of x and y

3	#include <stdio.h>
	int g=0, i=1;
	main(){
	printf("g = %d,
	i = %d \n", g, i);
	}
		#include <stdio.h>
		
		int g, i;
		main(i){
		printf("g = %d, i = %d \n",g,i);
		}
			Output is: “g = 0, i = 1”

4	#include <stdio.h>
	main(){
	int x = 10;
	while(x > 0){
	x--;
	printf("%d ", x);
	}
	}
		#include <stdio.h>
		main(){
		int x = 10;
		while(x --> 0) {
		printf("%d ", x);
		}
		}
			Output is: “9 8 7 6 5 4 3 2 1 0”

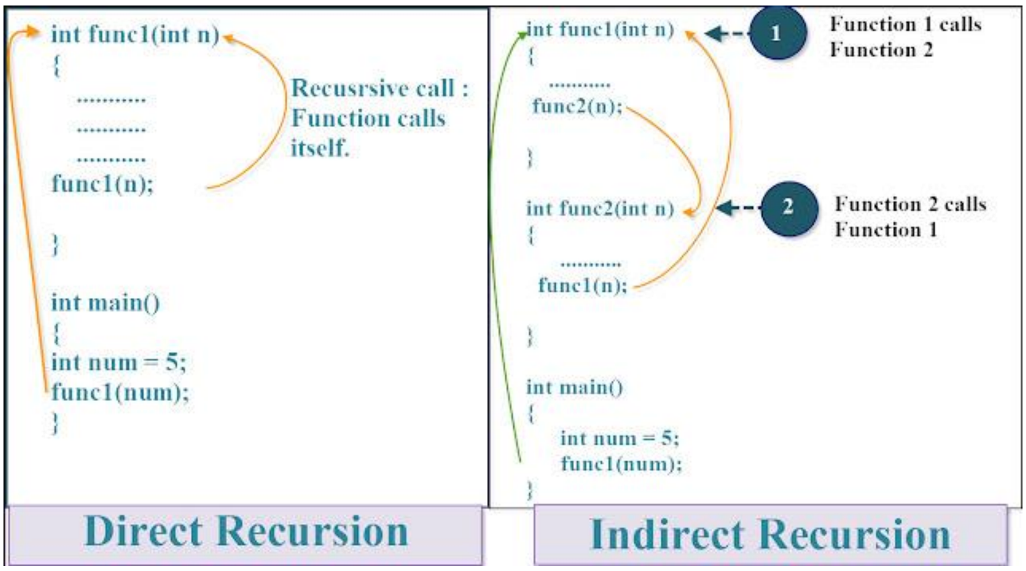
5	#include <stdio.h>
	main() {
	int a[3]; int b=5;
	for(int i=0;i<3;i++) {
	a[i]=777;
	}
	printf("%d", b=777);
	}
		#include <stdio.h>
		main() {
		int a[3]; int b=5;
		for(int i=0;i<4;i++) {
		a[i]=777;
		}
		printf("%d", b);
		}
			Setting b variable in to value 777
			
			Output is “777”

Besides C++, other high-level programming languages have their own indirect programming capabilities, too. The more flexible a programming language is, the wider is the field for indirect programming. For example, Javascript is richer for indirect programming than Pascal. One of the features in the Javascript programming language is, that it has a reduced calculation of logical operations. This means that only the second expression (3) will be executed from the following two expressions (2), (3):

```
true  || alert("will NEVER work");                                    (2)
false || alert("will work");                                          (3)
```

This feature is exploited, when it is necessary to execute instructions if the condition on the left side is false [11]. There is a standard “IF” operator in the Javascript programming language, which is designed to execute instructions regardless of what the value in the condition is equal to, i.e. true or false. Thus, the “||” operation has some incomplete similarity to the “if” operator that is it has a beneficial side effect, and this is an example of indirect programming, too.

An indirect recursion is one more example of indirect programming. Recursion is a programming approach in which a function calls itself intending to solve a problem [12]. There are two types of recursion: direct and indirect (Fig.2).



```
int func1(int n)
{
    ...........
    ...........
    ...........
func1(n);

}

int main()
{
int num = 5;
func1(num);
}
```

```
int func1(int n)
{
    ..........
func2(n);

}

int func2(int n)
{
    ..........
func1(n);

}

int main()
{
    int num = 5;
    func1(num);
}
```

Fig. 2. Direct and indirect recursion schemes [13].

Direct recursion is the type of recursion in which a function directly calls itself within its own block of code [12]. In indirect recursion, a function calls another function which then calls the first function again [12].

Self-modifying code is one of the promising feature in programming now. A self-modifying code is a program or an algorithm that changes itself namely changes instructions to be executed. Probably, one of the first appearances of self-modifying code occurred in the era of computer viruses infecting executable files of the MS-DOS operating system. Such computer viruses encrypted its code to make it difficult to be detected and destroyed [14]. At that time, computer viruses were written mainly in Assembler programming language. Nowadays, computer programs are not written entirely in Assembler programming language. Basically, high-level languages are used to write programs. Some programming languages, for example C/C++, make it possible to create self-modifying code [15]. It is necessary to test this possibility using modern compilers, because they may prevent you from changing the code directly, but in general it is at least theoretically possible. The use of a self-modifying algorithm is an example of indirect programming, although you can also get by only with direct programming, because almost everything that is needed can be implemented using only direct programming.

IV. Classification

Now it makes sense to consider the classification of indirect programming, when many examples of such programming have been considered. In general, the presence of a classification indicates a long period of development of an object or field of study. In addition, classification is possible if there are many objects. In other words, one object excludes the possibility of classification. For example, one masterpiece cannot be classified, but several masterpieces can already be classified. In the case of indirect programming, classification is possible since there are quite a lot of examples classified as indirect programming.

So, based on the indirect programming examples given earlier, there are two main types of indirect programming. They are:

1) programming by side effects,
2) programming by influencing one object to produce a result in another object.

Programming by side effects is programming in which the emphasis is on secondary rather than primary results of actions (operations, functions or operators). Among the examples given earlier, this type includes the following: in Table I (cases 1, 2, 4), in Table II (cases 1, 2, 3, 5) and case (3). The second type of indirect programming uses an intermediate object to achieve the result of another object. The following cases should be classified as this type: in Table I (cases 3, 5), indirect recursion and self-modifying code.

The case 4 in Table II remains still unclassified, and this is because this case is special. In full, it cannot be attributed to either one or another type of indirect programming. In this case, syntax features of the C/C++ programming language are used namely the expression “x --> 0” or arrow that shows x tends to 0, however in reality this arrow operation composed of two operations: “--” or decrement (postfix decrement) operation and “>” or operation more. This is not the only case of peculiarities of the syntax of a programming language; you can find other similar examples. Moreover, it is high-level languages, not low-level ones, that have similar features. Moreover, the more flexible the programming language, the more such possibilities there are. For example, the C++ programming language has more such capabilities than the BASIC programming language. It is possible to give an example to demontrate this. All programmers know such an expression in C++ programming language (4):

```
x = (y < 0) ? 10 : 20;                    (4)
```

But not all programmers know that such a construction (5) is also acceptable [16]:

```
(a == 0 ? a : b) = 1;                    (5)
```

Such a construction/expression (5) is equivalent to the “if” operator (6):

```
if (a == 0) a = 1; else b = 1;                    (6)
```

There are many other capabilities that are provided by the rich syntax of flexible high-level programming languages, but they will not be presented in this research, because this is not a reference book. The main idea, according to what principle the described cases with special syntax are grouped, is clear, so let us summarize the results of this research.

V. Conclusion and perspectives

Any development is usually accompanied by a proliferation of forms, types, and modifications. Programming has been developing for quite a long time and we know many of its types. This is functional, object-oriented, logic programming and so on. Moreover, programming did not stop in its development, only to be reborn into something qualitatively new, or fixed in a stable state. Evidence of this is indirect programming, the basics of which were discussed in this paper. Indirect programming continues the line of various types of programming, which brings together the real and virtual world (the world of information technology). The convergence of the real and virtual worlds is due to the availability of those methods in the virtual world that were previously available only in the real world.

The merit of this research can be attributed to the development of the terminology of indirect programming, the demonstration of various examples of indirect programming, as well as the development of a classification of indirect programming. However, the main contribution should be considered a new perspective, a new point of view from which one can look at long-known and used techniques and methods in practical programming. An alternative point of view is one of the factors that opens up new possibilities.

Indirect programming is fraught with many undiscovered possibilities. This paper has covered only part of what indirect programming means. Rather, this paper only touched on the tip of the iceberg of what indirect programming conceals. Only the most basic structures were considered here, and not all of them. For example, pointers and references were not considered among the basic constructs, if we take the C++ programming language. It is especially interesting to consider pointers to functions within the context. Object-oriented programming also provides a lot of scope for indirect programming. Two of the four most important principles in the concept of object-oriented programming, namely inheritance and polymorphism, give hope for the possibility of indirect programming. Less fundamental features in object-oriented programming are not mentioned, but this does not mean that they do not exist. Another direction in the context of indirect programming is multithreading. Each thread and/or process can influence another thread and/or process not directly, but indirectly. There are many restrictions imposed here, guided by security considerations, but it is impossible to suppress a real, vital phenomenon, so loopholes for indirect programming can be found here, too. One more direction in indirect programming is is the use of a cellular automaton. By setting its configuration, you can program interesting visual and dynamic effects. It is possible that not only visual effects can be programmed using cellular automata. This is a large field for research. It would also be useful to introduce semantics for both direct and indirect programming. For this, it makes sense to use ontologies of the OWL (Web Ontology Language) language, but it is possible that the set of tools of the OWL language is insufficient for this purpose. That is why additions await OWL language too.

Acknowledgments

This work like most previous works has been supported by my family and my friends.

References

[1] François Jullien. Traite de l'efficacite. Paris, Bernard Grasset, 1996. (in Russian)

[2] Sun Tzu, "The Art of War"

[3] B. H. Liddell Hart, "The strategy of indirect approach", 1941

[4] Joseph O'Connor, Ian McDermott, "The Art of Systems Thinking: Essential Skills for Creativity and Problem Solving", 1997

[4.1] International standard. ISO/IEC 9899:201x. Information technology — Programming languages — C. Committee draft N1570

[4.11] https://learn.microsoft.com/en-us/cpp/c-language/side-effects?view=msvc-170 [Accessed: 05.10.2023]

[4.2] https://www.yld.io/blog/the-not-so-scary-guide-to-functional-programming/#:~:text=A%20side%20effect%20is%20when,described%20as%20having%20side%20effects [Accessed: 05.10.2023]

[4.3] https://theailearner.com/2018/09/26/side-effects-in-programming/ [Accessed: 05.10.2023]

[5] https://academickids.com/encyclopedia/index.php/Xor_swap_algorithm [Accessed: 02.10.2023]

[6] http://www.sce.carleton.ca/courses/sysc-3006/f11/Part8-IndirectAddressing.pdf [Accessed: 28/09/2023]

[7] https://learn.javascript.ru/logical-operators [Accessed: 25/09/2023]

[8] https://www.baeldung.com/cs/recursion-direct-vs-indirect#:~:text=4.,number%20is%20even%20or%20odd. [Accessed: 26/09/2023]

[9] https://www.computerbitsdaily.com/2020/10/recursion-data-structures.html [Accessed: 03.10.2023]

[10] Klimentjev, Kompjuternie virusi i antivirusi, 2013 (in Russian)

[11] Tropeano, G., Self modifying code, 2006.

[12] https://proglib.io/p/hiddencpp?ysclid=lnahmu3aue919281097 [Accessed: 10.10.2023]