

Source Code

Joasia Krysa & Grzesiek Sedek (2006)

(for *Software Studies: A Lexicon*, edited by Matthew Fuller, 2008, MIT Press)

```
/ Barszcz C recipe
*
* string based cooking
*
* Copyleft (C) 2006 Denis "Jaromil" Rojo
* for the barszcz project (currently unfinished)

#include <stdio.h>

#define ingredient char

#define tool char

#define few 3
#define some 5
#define pinch 1
#define plenty 8
#define one 1

#define soft_cooked 5

ingredient **take(int quantity, ingredient **ingr) {
    int c;
    int len = strlen(ingr) + 10;

    ingredient = malloc( (quantity+1) * sizeof(*ingredient));

    for(c = 0; c < quantity; c++)
        ingredient[c] = malloc(len * sizeof(ingredient));

    ingredient[c+1] = NULL;

    return ingredient;
}
```

In *The Art of Computer Programming* Donald Knuth suggests an analogy between programming and recipes in a cookbook as a set of instructions to follow. Algorithms, much like cooking recipes, provide a (computational) method; a set of defined formal procedures to be performed in order to accomplish a task in a finite number of steps.

(1) Examining the source code of a particular program reveals information about the

software in much the same way as the ingredients and set of instructions of a recipe reveals information about the dish to be prepared. The analogy is rather straightforward perhaps but reveals something of the interests involved in the preparation, execution and consumption of the (software) work. (2) The importance of source code for the description of software is that, alongside computer commands, it also usually provides programmers' comments – a documentation of the program including a detailed description of its functionality and user instructions. (3) Furthermore, the importance of source code is that any modifications (improvements, optimizations, customizing, or fixes) are not carried out on compiled binary code (object code or machine code) but on the source code itself. The significance of this is that the source code is where change and influence can be exerted by the programmer. In the example of recipes, further descriptions are provided in the accompanying narrative and explanation with modifications to the instructions implemented by human intervention. Although recipes are clearly not reducible to code – and vice versa – the analogy emphasizes that both programming and cooking can express intentionality and style.

Source code (usually referred to as simply 'source' or 'code') is the un-compiled, non-executable code of a computer program stored in source files. It is a set of human readable computer commands written in higher level programming languages such as C, C++, Java, Pascal, BASIC, Ada, Algol, FORTRAN, or COBOL amongst many others. Defined by a higher level of abstraction from machine language they share some of the characteristics of natural language, for instance rules of syntax. When compiled, the source code is converted into machine executable code (binary) – a series of simple processor commands that operate on bits and bytes. The process of compiling is twofold: the source code is converted into an executable file either automatically by a compiler (i.e. the GNU C Compiler) for a particular computer architecture and then stored on the computer, or executed on the fly from the human readable form with the aid of an interpreter. In principle, any language can be compiled or interpreted and there are many languages such as Lisp, C, BASIC, Python or Perl that incorporate elements of both compilation and interpretation. (4) In the history of computation, programs were first written and circulated on paper before being compiled in the same way as recipes were written and shared before being compiled in cookbooks. The first case of an algorithm written for a computer is credited to Ada Lovelace that interpreted Charles Babbage's Analytical Engine (of 1835) not merely as a calculator but, as a logic machine capable of arranging and combining letters and other symbolic systems. (5) The source code of a modern digital computer derives from the further adaptation (in the 1940s) of Babbage's ideas. (6) What became to be known as 'the von Neumann

architecture' is important as it presented a single structure to hold both the set of instructions on how to perform the computation and the data required or generated by the computation; it demonstrated the stored-program principle that has led to development of 'programming' as separate from hardware design. For example, Remington Rand's UNIVAC (Universal Automatic Computer, 1951) was one of the first machines to combine electronic computation with a stored program and capable of operating on its own instructions as data. (7) With a stored-program computer, a sequence of instructions that might be needed more than once could be stored. The computer could store the sequence in memory and insert the sequence into the proper place in the program as required. By building up a library of frequently used sequences, a programmer could write a complex program more efficiently. (8) In *A History of Modern Computing*, Paul E. Ceruzzi explains this development, from building up libraries of subroutines and then getting the computer to call them up and link them together to solve a specific problem, to a more general notion of a high-level computer language with the computer generating fresh machine code from the programmer's specifications. (9)

The principle of re-using or sharing code is reliant on storing collections of code lines or functions in 'libraries'. The subroutine, often collected into libraries, is a portion of code within a larger program, which performs a specific task and is relatively independent of the remaining code. A subroutine is often coded so that it can be executed several times or from several places during a single execution of the program. It can be adapted for writing more complex code sequences, and is thereby a labour-saving programming tool and an important mechanism for sharing and reusing code. (10) An early example of a community-based library of subroutines was SHARE, of 1955, where a group of IBM users developed their own repository for shared use. More contemporarily, the principle of sharing source code is instantiated in online repositories (for instance, SourceForge, Freshmeat or Code Snippets) as well as source code search engines that index programming code and documentation from open-source repositories (for instance, Koders, Krugle, Codefetch and Codase). (11) Online code repositories are often used by multi-developer projects to handle various versions and to enable developers to submit various patches of code in an organized fashion. CVS, a version control system commonly used in open source projects, is an important management mechanism that allows several developers to work on the same files both simultaneously and remotely. It allows the recording of individual histories of sources files and documents while storing it on a single central server. (12) There are other examples that extend the online repository model to the cultural realm. For instance,

Perlmonks.org is a repository, discussion forum and a learning resource for the Perl community, that also provides an online platform for presenting Perl poetry and obfuscated code. Another example is Sweetcode.org that presents a themed and contextualized (reviewed) systematic selection of links to innovative free software. (13)

In *Free Software, Free Society*, Richard Stallman suggests that the sharing of software is as old as computing, just as the sharing of recipes is as old as cooking. (14) However, the reverse of this analogy holds too. Just as much as recipes might be shared (open) they also might be kept secret (closed recipes) in the same way as software licensing reinforces two radically opposite models of production, distribution and use of software – ‘open source’ and ‘closed source’. In general terms, under open source conditions, source code is included with a particular software package to allow its viewing and further modifications by the user (i.e. source code distributed under the terms of licenses such as BSD, GNU/GPL, MIT), whereas a proprietary model of closed source prevents its free distribution (and indeed modification) and software is released as already compiled binary program (i.e. software distributed under the Microsoft EULA (End User License Agreement)) (15). However, the politics of open source are much more complex. A further distinction is made between Open Source Software and Free Software within the free software community to articulate different ideological positions in relation to open source – emphasizing respectively either its development methodology or the ethical and social aspect of the ‘freedom’ of software. (16) More currently, the term FLOSS has been used as a more generic term to refer to Free, Libre and Open Source Software.

The idea of source code, and indeed the open source model, extends beyond programming and software. For instance, Knuth points to creative aspects of programming alongside technical, scientific or economic and suggests that writing a program ‘can be an aesthetic experience much like composing poetry or music’. (17) Following from this, source code can too be considered to possess aesthetic properties, as something that can be ‘displayed’ and ‘viewed’. (18) An even more radical assumption is to consider source code not only as a recipe for an artwork that is on public display but as the artwork – as an expressive artistic form that can be curated and exhibited or otherwise circulated. (19) For example, the activity of obfuscating code (making source code deliberately hard to read and understand), while in more general usage serves the purpose of protecting software applications from reverse engineering, might be also seen as creative practice in itself. It attempts to combine an executable function with an aesthetic quality of the source code through ‘simple keyword substitution, use or non-

use of white space to create artistic effects, to clever self-generating or heavily compressed programs' (20). The software art repository Runme.org lists obfuscated code under the category of 'code art' alongside code poetry, programming languages, and minimal code. (21) In the context of programming, the creative aspects are also registered in competitions such as the International Obfuscated C Code Contest: 'The aims of the contest are to present the most obscure and obfuscated C program, to demonstrate the importance of ironic programming style, to give prominence to compilers with unusual code and to illustrate the subtleties of the C language.' (22)

The excerpt of source code at the beginning of this entry is from a longer program and part of the Barszcz.net project. An online repository and a platform for presenting and sharing barszcz soup recipes in the form of source code written in number of programming languages, the project brings together cooking recipes and source code in quite a literal sense. (23) In a wider cultural context, this exemplifies a general line of thinking about source code as an open model for creative practice – encouraging collaboration and further development of existing work on the level of contribution, manipulation and recombination, and its further release under the same conditions in the public domain.

```
/* reminder about things we can do in the kitchen:  
* peel, wash, chop, cook */
```

```
beetroots = wash( beetroots );  
cabbage = chop( cabbage );  
  
cooking = 0;  
do {  
  
    cook( beetroots );  
    cook( cabbage );  
    cook( carrots );  
    cook( parsnips );  
  
} while( cooking < soft_cooked);  
  
carrots = cook(beetroots);  
exit(1);  
}
```

References

1. Donald Knuth *The Art of Computer Programming*, volume 1 'Fundamental Algorithms', (Reading, Mass.: Addison – Wesley Publishing Company, 1981 [1968]), p.8.
2. The metaphor is also used by the Belgian artists group Constant in their project Cuisine Interne Keuken (2004) to examine the economics of the internal organisation of the cultural system and the workplace – a system that consists of components (ingredients), tools (utensils) and work and creation processes (recipes).
<http://www.constantvzw.com/cn_core/cuisine/>
3. In this connection Paul E. Ceruzzi in *A History of Modern Computing* (Cambridge, Mass.: MIT Press, 2003 [1998]) points to some earlier examples of programs such as COBOL that had 'the ability to use long character names that made the resulting language look like ordinary English'. Thus the program was self-documenting – instructions were sufficient descriptions for both machine and humans and programmer's comments were not required (92–93).
4. 'For instance 'the "compiler" for a bytecode-based language translates the source code into a partially compiled intermediate format, which is later run by a fast interpreter called a virtual machine. Some "interpreters" actually use a just-in-time compiler, which compiles the code to machine language immediately before running it'. (http://en.wikipedia.org/wiki/Programming_language)
5. J. David Bolter, *Turing's Man*, University of North California Press, 1984, p.33.
6. Although the stored-program principle is commonly credited to John von Neumann for his 'First Draft of a Report on the EDVAC' (1945), he was not the sole creator of 'von Neumann Architecture'. According to Paul E. Ceruzzi (1998, p.21–22), it was J. Presper Eckert and John Mauchly who first conceived of the similar idea (in 1944).
7. UNIVAC was designed by Eckert and Mauchly (Ceruzzi, 2003 [1998], p.20).
8. Paul E. Ceruzzi, *A History of Modern Computing*, Cambridge, Mass.: MIT Press, 2003 [1998], p.84.
9. Ibid. p.108.
10. Knuth, op cit, p. 182.
11. Examples cited: SourceForge <http://sourceforge.net/>, Freshmeat <http://freshmeat.net/>, Code Snippets <http://www.bigbold.com/snippets/>, Koders <http://www.koders.com/>, Krugle <http://www.krugle.com/>, Codefetch <http://www.codefetch.com/>, Codase <http://www.codase.com/>.
12. CVS developed from an earlier versioning system RCS and is similar to other packages such as PRCS, and Aegis. <http://www.nongnu.org/cvs/>
13. Examples cited are PerlMonks.org (<http://perlmonks.org>) and Sweetcode.org (<http://www.sweetcode.org/>). One of the developers of the Runme software art repository, Alex McLean, described it as 'perhaps the closest thing to an art gallery for the free software community, and indeed one of the inspirations for Runme.org'. <http://runme.org/feature/read/+sweetcode/+45/>
14. Richard Stallman 'GNU Project', in Joshua Gay (ed.), *Free Software, Free Society: Selected Essays of Richard M. Stallman*, Boston, MA: GNU Press, Free Software Foundation, 2002, p. 31–39.

15. Examples cited: BSD (Berkeley Software Distribution) <http://www.bsd.org/>, GPL (General Public License) <http://www.gnu.org/>, GNU/Linux project <http://www.kernel.org/>, and MIT (Massachusetts Institute of Technology) OpenCourseWare <http://ocw.mit.edu/>, Microsoft End User License Agreement <http://msdnaa.oit.umass.edu/Neula.asp>. For an extensive list of licences see: <http://www.opensource.org/licenses/> or <http://www.fsf.org/licensing/licenses/>
16. Richard Stallman 'Why Free Software is Better than Open Source', in Joshua Gay (ed.), *Free Software, Free Society: Selected Essays of Richard M. Stallman*, Boston, MA: GNU Press, Free Software Foundation, 2002, p. 55–60. Also see: Free Software Foundation <http://www.fsf.org/> and Open Source Initiative <http://www.opensource.org>
17. Donald Knuth *The Art of Computer Programming*, volume 1 'Fundamental Algorithms', (Reading, Mass.: Addison – Wesley Publishing Company, 1981 [1968]), p.v.
18. In relation to artistic practice, the idea of making source code of work public is evident for instance in live programming: 'In this area of software arts practice programmers make music in keeping with the expressive qualities of live performance, by using interpreted scripting languages (such as perl) and coding in real-time with the source code on public display.' Geoff Cox 'Software Actions', in Joasia Krysa (ed.), *Curating Immateriality*, DATA Browser 03 (New York: Autonomedia, 2006), p.76.
19. The phenomenon of computer viruses demonstrates the aesthetisation of code quite explicitly. For the purpose of art context, usually harmful properties of viruses are typically removed and viruses are exhibited as aesthetic systems. As an example, the notorious work 'biennale.py', a computer virus programmed in Python by the artist collective [epidemiC] and net art group 0100101110101101.org, operated with the sole purpose 'to survive' (13) by acting upon its exhibition context of the 49th Venice Biennale. Subsequently it was included along with other examples in I Love You [rev.eng] (2002), a larger show dedicated to phenomena of computer viruses in artistic context. See Alessandro Ludovico 'Virus charms and self-creating codes', in Franziska Nori (ed.), *I love you: computerviren, hacker, kultur*, exhibition catalogue (Museum für Angewandte Kunst: Frankfurt 2002), p. 40.
20. http://www.wikipedia.org/wiki/Obfuscated_code/
21. http://www.runme.org/categories/+code_art/
22. http://www.digitalcraft.org/iloveyou/c_code.htm
23. In culinary terminology 'Barszcz' [English: Borscht] refers to a traditional Eastern European speciality soup of red beetroot that comes in many regional varieties, <http://www.barszcz.net>