C: My Approach

George Russell
george.russell@clara.net

Jan 18, 2002

In the 1st issue of the **FREE SOFTWARE** magazine, was the article **"Why C is here to stay"**. In its initial paragraph, the question is raised : *"is the C language, which is over 30 years old, still relevant?"* It then goes on to propose that the answer is a resounding *"Yes"*.

I propose, that while the C programming language is still relevant to Free Software, that this is largely a historical accident, and that much, if not most, free software projects could be better served with a higher level language. While I do not propose any one language as suitable for all, I will provide references to various alternative languages which address various of C's deficiencies.

Much Free Software is written using the C programming language. This includes such notable projects as the Linux kernel, the GNU Compiler Collection, The XFree86 implementation of X11, a multitude of server daemons and desktop programs. Perhaps even the majority of programs written for GNU systems are written in C. The C programming language was designed as a systems programming language, in which to implement the Unix operating system. As such, it was targeted at expert programmers, implementing an entire operating system. Efficiency was a paramount consideration, and many low level features were included to allow for interfacing easily with hardware devices. Flexibility and simplicity were important, but safety was considered of less importance — the programmer was assumed to know what he was doing.

C rose to success in tandem with the Unix operating system, one assisting the other. To write programs on Unix, it was natural to write those in C, the language supplied with Unix. To extend Unix, again C was appropriate. To run Unix applications on other platforms, first you needed a C implementation and then you could port the C program. So when GNU in 1984 decided to clone Unix to produce the GNU System, it was naturally C that was chosen as the implementation language. GCC has for many years been one of the most portable and widely used C compilers available.

Since C was introduced, many other programming languages have been introduced. These have looked at the weaknesses and strengths of C, and learnt from them.

The weaknesses of C, in my opinion, make it largely inappropriate as a language for developing new software projects. The weaknesses of C are detailed below. Of course, these are subjective, and may well provoke some flamage from ardent fans.

Manual Memory Management

C requires that you manually allocate heap memory using the malloc, calloc, realloc functions, and when you are done with that memory, that you "free" it. In any non trivial program, tracking all the memory allocations is tedious, error prone and time consuming. This leads to programs which over their lifetime acquire memory but never free it, causing the memory usage of the program to increase and denying use of the memory to other processes.

Garbage collection is a well understood technique, which removes the need to manually deallocate memory. Data in memory is freed when no references to it are kept. The technique has been used for years in Lisp, the various functional programming languages such as ML, Haskell, the pioneering object based language Smalltalk, and fairly recently in Java. Bjarne Strustroup, creator of C++, has indicated garbage collection will be an optional part of the next C++ standard.

Bounds Checking, Strings as Arrays of Characters

This is best summed up Commandment 5 in the C programmers 10 commandments. "for surely where thou typest foo someone someday shall type supercalifragilisticexpialidocious."

In other languages, arrays are bounds checked at runtime. This eliminates an entire class of common programming errors due to buffer overflows, which are not only major causes of bugs but also security holes in software such as network daemons. C represents strings as arrays of characters with a NULL terminator. Operations involving storing these place the responsibility on the programmer to check that there is enough space in the buffer to hold the data.

Bjarne Stroustrup shows how the need to correctly handle C style strings greatly increases the complexity of even simple programs, presenting greater possibilities for error on the part of the programmer. This can be seen in his paper, Learning Standard C++ as a New Language.

A Broken Type System

The type system of C is broken by design. Other languages provide sophisticated checks at compile time, or perform sanity checks at run time on the types of variables.

C will let you treat any type as you will, allowing you to omit function prototypes, the number and types of arguments and the return type of functions. These omissions result in more work for the programmer who has to debug the problems that result. While some feel providing type information is too much of a hassle, languages such as Python allow its omission, and OCaml will infer the types of your data. Other languages detect at compile time problems that only become evident at run time in C. There is no loss of power in the use of type information, all that is required is that the programmer make explicit his intentions for how the data is to be treated.

Error Handling and a Lack of Exceptions

C lacks support for exceptions to indicate when an error has occurred. The programmer calling a function, such as malloc, must explicitly check the return value for error details. This damages readability, since the purpose of the code is obscured in error checking code. The alternative, too often chosen, is to omit the error checking and ignore failures.

Exceptions allow for grouping of error handling in one place through the use of try ... catch blocks,

which can allow all exceptions to be caught, and handled where they occur or propagated back up the handling chain to a suitable handler.

Lack of High Level Language Features ...

Users of high level languages such as OCaml, Java, C++, Perl or Python often find C lacks useful features which are built into the other languages. OCaml provides type inference, higher order functions and built in garbage collection, Java provides standardised networking, windowing and threading libraries, Perl has built in regular expressions and pattern matching for string operations, Python has more sophisticated high level dynamic data types built in such as dictionary's, lists, tuples and sets. C is a small language — and as a result, functionality omitted in the core language requires third party libraries. The C standard library is much smaller that that of other languages in terms of functionality.

So, while this shows why I do not feel C is an appropriate language for Free Software development, what do I feel is appropriate? More importantly, what do many other Free Software authors use?

For those who like the syntax and structure of C, C++ is the obvious successor. The GNU Compiler Collection includes G++, the GNU C++ compiler. This is a compiler which compares favourably with commercial proprietary offerings in portability and standards compliance. Much free software is built using g++ . Some obvious examples of large C++ projects include the K Desktop Environment, with its sophisticated office suite, web browser, and integrated development environment. The Mozilla web browser is written in C++. The Open Office Suite, formerly Star Office, is also written in C++.

C++ is a superset of C, as standardised in 1989. Any well written C89 program is also a C++ program, and can be compiled with a C++ compiler. *"Thinking in C++* [1] describes how C++ can be used as "a better C". The C++ compiler is pickier than a C compiler and can reveal hidden errors in C programs, due to better type checking and compile time analysis. Its also possible to easily link in prewritten C libraries in C++. The low level features of C, such as the ability to mix assembly code, are preserved in C++.

C++ provides a choice of programming style and idiom. While C is procedural, C++ can be procedural, Object Oriented or generic. The C++ STL library provides powerful and efficient container classes, such as vector, algorithms, such as sort, and data types such as string. C++ can be used to express programs more succinctly and with greater clarity than C, by removing the need to manage every detail of memory allocation, string handling, and do this without sacrificing runtime efficiency.

The programming language Java is similar to C++ and C in terms of its syntax. Language features which were dangerous, such as pointers, have been removed. The GNU Compiler collection provides support for java in GCJ, which compiles to native code. Otherwise, Java can be compiled to portable byte code, that can run unmodified across several platforms, unlike natively compiled applications. Java is also used in a number of free software projects. These include Netbeans and Eclipse, flexible plug-in based integrated development environments supporting features such as incremental compilation, graphical interface design, code refactoring and auto completion, syntax highlighting. The text to speech synthesiser FreeTTS, which provides an emacspeak compatible server is written in Java. The Limewire

[1] This is a nice book, but it is not free. — **FSM**

gnutella client is also written in Java. In general, Java is suitable for any task requiring a portable GUI application, involving networking, connecting to databases and many other tasks supported in the extensive standard class library.

Perl and Python programs can also be run wherever interpreters are available. OCaml programs can compile to native code or portable byte code. Perl is a ubiquitous glue language, and Python is a high level scripting language suitable for prototyping and used in the Zope server software.

The GNU system provides a plethora of programming languages. While C has historically been the choice of free software hackers, today there are a large number of alternatives which can create equally efficient and elegant programs in less lines of code. Try a new language today.

About the Author George Russell is a graduate computer science student at Strathclyde University in Scotland who uses GNU/Linux daily at work and at home. He has used GNU/Linux since 1997 and written applications on it using Java and C++. He can be reached by `george.russell@clara.net`.

References

The Ten Commandments for C Programmers
http://www.lysator.liu.se/c/ten-commandments.html

Thinking in C++,
Bruce Eckel
http://www.bruce-eckel.com

Learning C++ as a new language
Bjarne Stroustrup
http://www.research.att.com/~bs/papers.html

The Ocaml Web page
http://www.ocaml.org

The Python Web page
http://www.python.org

The Java Web page
http://java.sun.com

Mozilla
http://www.mozilla.org

KDE
http://www.kde.org

Open Office
http://www.openoffice.org

Netbeans
http://www.netbeans.org

Eclipse
http://www.eclipse.org

FreeTTS
http://freetts.sourceforge.net

Errata:

On page 79, 80, Issue 01, the download URL of **"How to Think Like a Computer Scientist"** was wrong, the right one is *http://www.ibiblio.org/obp/thinkCSjav/*.

Notice:

Our planned article about ZMailman (How to use GNU Mailman mailing list inside the Zope web server) will be delayed to publish in a future issue. —**FSM**