Why Do We Need the C language in Programming Courses?

Katsuhiko Gondow and Yoshitaka Arahori
Department of Computing Science, Tokyo Institute of Technology, Tokyo, Japan

Keywords: C Language, Programming Courses, Embedded/System Programming, Operating Systems, Language Pitfalls.

Abstract: The C language is still one of the important programming languages both in development and in education, since C has several positive characteristics like good abstraction for low-level programming, and fast execution speed with less footprint, although C has several drawbacks and pitfalls like buffer overruns. So there are several research studies to support C programming educations to compensate the C's drawbacks and pitfalls, but there is a skeptical view about these research direction: "C is a bad language, so it is better to stop teaching C, instead of supporting C education". In this position paper, we argue against this skeptical view, mainly because C is very important in upper-level courses like embedded/system programming and operating systems, so worth teaching.

1 INTRODUCTION

The C language is still one of the important programming languages both in development and in education, since C has several positive characteristics like good abstraction for low-level programming, and fast execution speed with less footprint, although C has several drawbacks and pitfalls like buffer overruns.

So there are several research studies to support C programming educations to compensate the C's drawbacks and pitfalls (Uchida and Gondow, 2016)(Freund and Roberts, 1996)(Song et al., 1997) (Osuka et al., 2012)(Kummerfeld and Kay, 2003), but there is a skeptical view about these research direction: "C is a bad language, so it is better to stop teaching C, instead of supporting C education". In this position paper, we argue against this skeptical view, mainly because C is very important in upper-level courses like embedded/system programming and operating systems, so worth teaching, although we admit that the C language may not be suitable for a first language.

This paper is organized as follows. Section 2 presents the background of this position paper: the reason why C is still used (Sec. 2.1), previous work to support C programming education (Sec. 2.2), previous work to choose first programming languages (Sec. 2.3). Section 3 provides four C code examples to show good abstraction in low-level programming. In Section 4, our position "the C language is worth teaching" is given. Section 5 provides the conclusion.

2 BACKGROUND

2.1 C Language and Its Uses in Development

C is a general-purpose procedural programming language, developed by D. Richie in 1973 (Ritchie, 1993). Over the 40 years, C has been widely used as a system implementation language, being revised through the three standards (C90/C99/C11). For example, operating systems like Linux, language processors like GCC and JVM, embedded systems, device drivers, Web servers like Apache are written in C. In the TIOBE index (TIOBE, 2017) (a measure of popularity of programming language), C has been ranked first or second from 2002 to 2017. Thus C is not a legacy, but still an active programming language.

As is well known, C has a serious drawback; C programmers are very likely to write insecure code like buffer overruns since the compile/run-time checking of C is very weak. Despite this drawback, why so many developers use the C language instead of a more modern, safer language like Java? The answer is that C has several advantages as below:

- C provides better abstraction than assembly languages. For example, in C, it is easy to program low-level access to system or memory using inline-assembler or pointer arithmetic. This is the reason why operating systems, device drivers, embedded systems are written in C. The

Gondow, K. and Arahori, Y.
Why Do We Need the C language in Programming Courses?.
DOI: 10.5220/0006837205490556
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 549-556
ISBN: 978-989-758-320-9

compile/run-time checking of C is weak on purpose to allow this kind of low-level description in C.

- Most C programs run fast in less memory footprint. This is closely related to C's weak run-time checking. This is the reason why Web servers like Apache or language processors like GCC are written in C.

Thus the C language is indispensable to write efficient and portable code, where assembly languages are required or execution speed and less footprint are important.

2.2 Previous Work to Support C Programming Education

There are several previous works to support C programming education as follows, to compensate C's weak compile-time/runt-time checking. One important issue is that there is a skeptical view about this research direction; "C is a bad language, so it is better to stop teaching C, instead of supporting C education."[1] We argue against this skeptical view in Sec. 4.

- C-Helper (Uchida and Gondow, 2016) is a C static checker of our previous work, that aims to emit more direct error messages understandable for novices to correct wrong programs, and also aims to handle latent errors. As our another previous work (Kojima et al., 2015) indicated, warning messages in commercial-level compilers like GCC are often difficult and misleading for novice programmers. Our C-Helper tries to (partially) solve this problem.

  Listing 1: Example of assigning string to `char` array variable.

```
1 char arr[20];
2 arr[20] = "This is illegal";
```

  For example, for Listing 1, GCC-4.7.2 emits the message "warning: assignment makes integer from pointer without a cast", which is very confusing for novices. Instead, our C-Helper provides a more direct message "String cannot be stored in an element of `char` array variable. Consider to use `strcpy`".

- Thetis (Freund and Roberts, 1996) is an integrated development environment for C with C interpreter, understandable error reporting, run-time error detection, debugging/visualization tools. A special feature of Thetis is *strengthened syntactic restrictions*, where common code fragments often mistakenly used by novices are regarded as errors by Thetis, even though they are perfectly legal in the C standard. The most common example is `if(i=0)`$\cdots$, where the assignment operator = is often misused instead of the relational operator == by novices.

- C-Tutor (Song et al., 1997) is a C program analysis tool, which provides novices with understandable messages to correct their wrong programs. C-Tutor combines static and dynamic analysis techniques, and extracts novice's intentions using sample programs.

- CX-checker (Osuka et al., 2012) is a C coding style checker. While the current C-Helper only checks unbalanced indentation as a common coding style, CX-checker copes with various styles, which are highly customizable using XPath, DOM and wrapper API.

- Kummerfeld's method (Kummerfeld and Kay, 2003) catalogs some common C/C++ compiler error messages, typical code examples for them, and their possible corrections as a Web-based reference guide. This method might be effective, but its maintenance cost is very high since the catalog needs to be updated whenever compiler messages are changed.

2.3 Previous Work to Choose First Programming Languages

There are so many discussions about what language to teach as a first programming language (Kaplan, 2010) as below. There are many factors to decide this: e.g., student's ability, interest, background, prospective career, the levels of courses, and so on, thus making this discussion harder. Although this discussion is very common in our community, as far as we know, not restricted to the first language, what programming languages should be taught in the entire computer science courses has not been studied sufficiently.

- Early work suggested the C language was well introduced in introductory programming courses. For example, (Gilberg and Forouzan, 1996) reported the number of students in C courses dramatically increased compared to Pascal courses. Also, (Hosch, 1996) reported the combination of C and UNIX was well taught. Of course, these may not apply to today's situation since new programming

[1] Actually there was this question in our presentation in (Uchida and Gondow, 2016).

languages have emerged like Java, Python, C# and JavaScript.

- Java is one of the most popular first language nowadays; for example, Java is ranked first in (Gaspar et al., 2007), and ranked second in (Guo, 2014). As positive characteristics, Java is (relatively) small, robust, portable (Hosch, 1996)(King, 1997). Furthermore, Java is attractive to students since Java is a programming language used to develop Android applications. Of course, Java has several minor problems; e.g., uncaught exceptions must be declared explicitly, which is troublesome for introductory students (Hosch, 1996)(King, 1997).
- C# is another candidate of today's first programming language. As (Bates, 2004) mentioned, C# is easier to learn the fundamental concepts in the programming, since C# is a very sophisticated object-oriented language.
- Python is yet another most popular first language nowadays; for example, Python is ranked first in (Guo, 2014). As positive characteristics, Python is simple, dynamic, readable and Python has fewer syntactic exceptions. (Sanders and Langford, 2008) pointed out another interesting issue; Python is a language that attracts both experienced and inexperienced students. Like Scheme, Python is easily accessible to inexperienced students. At the same time, Python is a real-world popular language, so Python is positively accepted to experienced students, too.
- JavaScript is a today's very popular language, but JavaScript seems not to be considered as a good candidate of first programming languages. For example, (Mombrea, 2014) pointed out that JavaScript is a poor example of Object-Oriented programming. Another issue is that JavaScript's callback hells make it difficult for novices to learn JavaScript programming. The characteristic of "run-to-completion" in JavaScript prohibits programmers from blocking like simply calling `sleep(10)` in C. Instead, JavaScript always forces programmers to use callbacks (e.g., using `setTimeout()`).

3 C CODE EXAMPLES TO SHOW GOOD ABSTRACTION FOR LOW-LEVEL PROGRAMMING

It is often said that one big advantage of the C language is good abstraction for low-level programming, e.g., in (TechLiebe, 2014)(Noordergraaf et al., 2015)(Viswa, 2014). But few of them present concrete code examples, which makes it difficult to discuss the reason why the C language is required in embedded/system programming courses, and thus how the C language is useful for students to understand how the system works inside the computer. In this section, we present several concrete code examples to show that the C language is good abstraction for low-level programming, but Java is not.

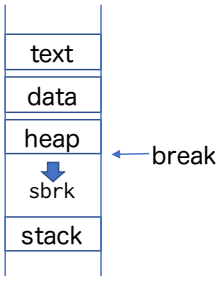

Figure 1: `sbrk` allocates memory by moving down the break position.

Listing 2: Outline of custome memory allocator.

```
1  struct memory_buckets *freelist;
2  void *my_malloc (size_t size) {
3      void *p, *s;
4      // first search freelist for an unused
5      //  bucket with 'size' bytes
6      p = search_freelist (size);
7      if (p != NULL) return p;
8
9      // then call sbrk, and divide the obtained
10     // memory into 'p' and 'freelist'
11     s = sbrk (BULK_MEMORY_SIZE);
12     divide_mem (s, &p, &freelist);
13     return p;
14 }
```

3.1 Custom Memory Allocator

To learn the mechanism of memory management in a hands-on way, it is a good way to implement a simple custom memory allocator, garbage collector, buffer overrun detector, and so on. In C, it is fairly easy to implement a custom memory allocator (e.g., reimplementation of `malloc` in Unix) using `sbrk` and/or `mmap`. As shown in Fig. 1, `sbrk` is a system call in Unix to allocate memory by moving down the bottom of heap (called the break position). `malloc` is implemented, considering the following requirements:

- To avoid system call's overhead, the number of `sbrk` calls should be reduced. `malloc` achieves this by allocating a large memory block with one `sbrk` call, and dividing it into several memory buckets.

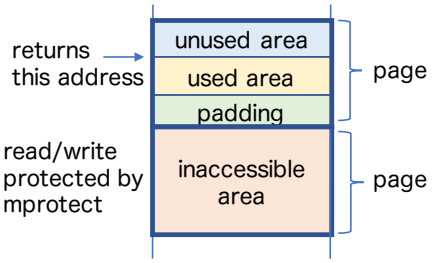

Figure 2: Inaccessible memory area detects buffer overrun.

Listing 3: Outline of buffer overrun detector.

```
void *ef_malloc (size_t size) {
    void *p, *p2;
    size_t size2 = ROUNDUP_PAGE (size);
    p = mmap (size2, PROT_WRITE|PROT_READ);
    p2 = p + size2 - PAGE_SIZE;
    mprotect (p2, PAGE_SIZE, PROT_NONE);
    return p + size2 - PAGE_SIZE - size;
}
```

- To achieve high-speed memory management, `malloc` manages the memory buckets using the data structure like hash or balanced tree.

So the outline of a custom memory allocator becomes like Listing 2. Thus, students see how two abstraction layers (`malloc` and `sbrk`) are bridged with actual running code in C. On the other hand, it is difficult in Java, since the API corresponds to `sbrk` is not provided in Java, and if provided, it is difficult to replace Java's `new` with the custom memory allocator.

By extending this custom memory allocator in C, it is also fairly easy to implement a garbage collector, and buffer overrun detector. For example, Fig. 2 shows the idea of Electric Fence, which is one of the buffer overrun detectors, where two pages are used at least; the last page is used to protect invalid read/write (inaccessible area), and the other pages are used for the user (used area). In actual implementation, `mmap` is used to allocate memory on continuous pages, and `mprotect` is used to prohibit read/write on the last page (Listing 3). If a read/write is attempted to the inaccessible area as a result of buffer overrun, it is notified as a signal delivery like `SIGSEGV`. Thus the students understand how the memory management works using actual running code in C, while it is difficult in Java, since `mmap` and `mprotect` are not available in Java.

3.2 System Calls v.s. Function Calls

In system programming, it is essential for the students to understand the difference between a library function call and a system call (Fig. 3). Both of library functions (e.g., `printf`) and system calls (e.g., `write`) have the same interface to programmers in the sense that programmers can call both of them as a function call in C (e.g., `printf();` and `write();`). The reason why the difference is essential is because a library function is executed in the user space, while a system call is executed in the kernel space. All system calls are implemented inside the kernel, since they requires the CPU privilege, or some arbitration in the kernel (e.g., to avoid data races in the shared data among all processes).

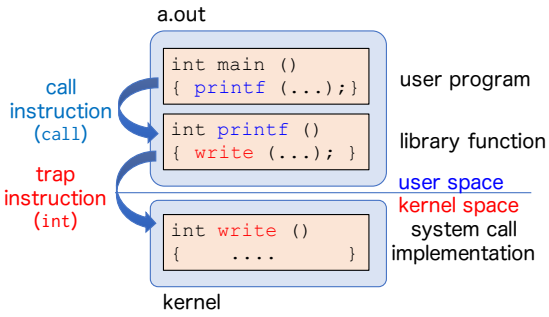

Figure 3: a library function is called with `call`, while a system call is called with `int` in x86-32.

Listing 4: The difference between library function and system call can be seen in the compiled assembly code.

```
% cat foo.c
int main (void)
{
    printf ("1\n");
    write (1, "2\n", 2);
}
% gcc -S foo.c; cat foo.s
...
call printf
...
int  0x80 // write
```

The easiest way for the students to see the difference is to see the assembly code compiled from C (Listing 4)[2]. This method using C is easy and straightforward for students, but not in Java, since we cannot see the difference in the Java bytecode compiled from Java.

3.3 Setting a Breakpoint in Debuggers

Suppose a system programming course where students implement a simple debugger which natively runs on their PCs. A key feature of debuggers is a breakpoint, and a typical way to implement (software) breakpoints is to overwrite the instruction at the bre-

[2]In the recent Unix(-like) OSes, a system call is wrapped as a library function. In this case, we need to use, for example, process monitor tools like `ltrace` and `strace` in Linux.

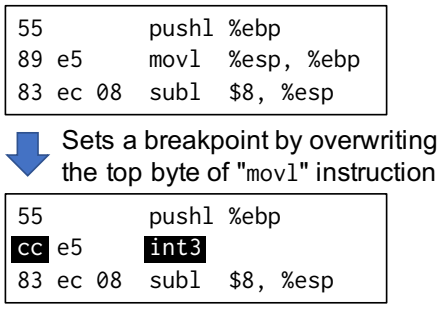

Figure 4: Setting a breakpoint on x86-32.

Listing 5: int3 transfers the control to the debugger.

```
% lldb ./a.out
(lldb) run
Process 7445 stopped
   2    int main ()
   3    {
   4        printf ("1\n");
-> 5        asm ("int3");
   6        printf ("2\n");
   7    }
```

akpoint address with a software interrupt instruction[3]. In x86-32, for example, we can set a breakpoint to overwrite the top byte of instruction (`movl` in Fig. 4) with the software interrupt instruction `int3` (`0xCC` in hex). The students easily see `int3` transfers the control to the debugger using inline assembly code like Listing 5.

Most Unix(-like) OSes (e.g., Linux) provide an process file system API (procfs) to allow programmers to access the debuggee's memory address space as file I/O. So the C code to set a breakpoint becomes like Listing 6, where first the byte at the breakpoint address (`addr`) is saved to `orig`, and then `int3` instruction (`0xCC`) is overwritten. Thus, the students easily understand how the breakpoint works in C.

On the other hand, it is not easy for students to understand it in Java, since the notion of software interruption (`int3` in x86-32) is completely hidden under the Java API. The Java Virtual Machine Tool Interface (JVM TI) provides a debugger interface including the following method to set a breakpoint.

```
jvmtiError SetBreakpoint (jvmtiEnv* env,
  jmethodID method, jlocation location);
```

But the students cannot see the inside of the Java method `SetBreakpoint` as Java code, since it is implemented in C using Java Native Interface (JNI).

Listing 6: Setting a breakpoint in C.

```
void
set_breakpoint (int fd, void *addr, char *orig)
{
    char int3 = 0xCC;
    llseek (fd, addr, SEEK_SET);
    read (fd, orig, sizeof (*orig));
    llseek (fd, addr, SEEK_SET);
    write (fd, &int3, sizeof (int3));
}
```

3.4 Direct Memory Access (DMA) in OSes

In the above discussion through Sec. 3.1 to Sec. 3.3, all example code runs in the user space. The same discussion holds for the code that runs in the kernel space. For example, Listing 7 is actual C code that sets up the DMA controller (Intel 8237) to transfer data between an I/O device (typically HDD) and memory without CPU. The textbooks of operating systems just explain how to do DMA transfer like the following simple description.

1. The kernel sets up the I/O device in DMA mode.
2. The kernel instructs the DMA controller to transfer data in DMA mode with the specified address, size and the transfer direction (i.e., read or write). The code example in Listing 7 does this step in PC/AT compatibles.
3. The DMA controller transfers the data.
4. When the transfer is finished, the DMA controller interrupts CPU to signal transfer completion.

But the students cannot understand concretely how DMA works with the above simple description, since the API of the DMA controller and the code using the API are not offered. Using the actual running code example like Listing 7, the students can understand DMA more concretely:

- The whole actual running code is not long as shown in Listing 7, although each byte of address, for example, needs to be transferred separately.
- Five I/O registers (DMAC_clear_byte_point_flipflop, DMAC_base_addr_reg_2, DMAC_page_reg_2, DMAC_base_count_reg_2, DMAC_mode_reg) are used in this setup, which are the APIs of the DMA controller.
- During this setup, the DMA must be disabled.

[3]For lack of space, the detailed mechanism is omitted here.

Listing 7: Setting up the DMA controller.

```
void
outb (uint8_t value, uint16_t port) {
    asm volatile ("outb %0,%1"::"a"(value),"Nd"(port));
}

static void
setup_DMAC (uint32_t physical_addr, int count, int is_read) {
    // disable DMA2 (mask set for DMA2)
    outb (0x06, DMAC_channel_mask_reg); // STCL=1, SEL1:SEL0=2

    /* set address and counter */
    outb (0xFF, DMAC_clear_byte_point_flipflop);
    outb (physical_addr       & 0xFF, DMAC_base_addr_reg_2);
    outb (physical_addr >> 8  & 0xFF, DMAC_base_addr_reg_2);
    outb (physical_addr >> 16 & 0xFF, DMAC_page_reg_2);
    count--;
    outb (count      & 0xFF, DMAC_base_count_reg_2);
    outb (count >> 8 & 0xFF, DMAC_base_count_reg_2);

    /* set mode */
    if (is_read) {
        outb (0x46, DMAC_mode_reg); // single, inc, noauto, write (I/O device -> memory), DMA2
    } else {
        outb (0x4A, DMAC_mode_reg); // single, inc, noauto, read (memory -> I/O device), DMA2
    }

    // enable DMA2 (mask clear for DMA2)
    outb (0x02, DMAC_channel_mask_reg); // STCL=0, SEL1:SEL0=2
}
```

4 OUR POSITION

4.1 The C Language is Worth Teaching

Our main position is that the C language is worth teaching since C is required to teach upper-class courses like embedded/system programming and operating systems. The C language does not have to be a first programming language, but it still be needed for the upper-class courses.

- The C language may not be the best first language in introductory courses themselves, especially just for teaching algorithms, or the fundamental concepts of programming languages like conditional branches, loops, variable assignments, (recursive) procedure calls, and so on. This is mainly due to C's weakness of compile-time/runtime checking, and the ambiguity of the C standards (e.g., the evaluation order of the subexpressions within an expression is unspecified in (C90/C99/C11)).
- Nevertheless, the C language is worth teaching in programming courses, since C is very required to teach upper-class courses like embedded/system programming and operating systems, as shown in Sec. 3. Furthermore, not only real operating systems, but also educational operating systems bootable in PC/AT compatibles like Xv6 (Cox et al., 2006) and udos (Gondow and Ohba, 2007) are written in C (and little assembly code). This is also supported by the survey analysis in (Gaspar et al., 2007).
- Native educational compilers (i.e., compilers that generate native assembly code like x86-32) have many practical and pedagogical advantages (Gondow, et al., 2010), since;
  - Generating native (i.e., real) assembly code motivates students to learn.
  - Students are getting better to read assembly code emitted by commercial-level compilers (e.g., GCC).
  - There are many high quality documents like (Intel, 2017) and tools (eg., GCC and GNU Binutils) for real machines.
  - Library functions (eg., `printf` in the C standard library) can be directly called from the source program to be compiled, which widens the expressiveness of the source program.

  The C language or its moderate subset is a good candidate of the source program to be compiled in such native educational compilers, since the mapping between C and assembly code is quite simple and straightforward.
- C is worth teaching in other reasons:
  - C is needed to learn low-level security issues

like buffer overruns and the techniques to solve them.

– C is also needed to read open source code like Linux, Apache, TCP/IP protocol stacks, to learn a wide variety of programming techniques.

4.2 Every Language Has Its Drawbacks and Pitfalls

Another position of ours is as follows. As mentioned in Sec. 2.1 and in (Koenig, 1989), C has some drawbacks and pitfalls, but this cannot be a reason to stop teaching C, since they are tightly coupled to C's advantages, and C is very required in upper-level courses like embedded/system programming and operating systems.

Similarly, other languages have their drawbacks and pitfalls, e.g., Java (Biddle and Tempero, 1998)(Daconta et al., 2003), Python (Miller and Settle, 2016), JavaScript (Alimadadi, et al., 2016).

5 CONCLUSIONS

The C language is still one of the important programming languages both in development and in education, so there are several research studies to support C programming educations to compensate the C's drawbacks and pitfalls. This position paper argued that the C language is worth teaching even though C has the drawbacks like buffer overruns and mysterious compiler error messages.

Our future work is to create a better C language while preserving the C's advantages. The languages like C++, Java, Objective-C, C#, Go failed to be a better C in our observation.

REFERENCES

Ritchie, D. (1993). The development of the C language. *2nd ACM SIGPLAN conf. on History of programming languages (HOPL-II)* pp.201-208.

Programming languages–C: ISO/IEC 9899:1990, 9899:1999, 9899:2011 (C90/C99/C11).

TIOBE programming community index. (2017). http://www.tiobe.com/tiobe-index/, [Online; accessed 31-Jan-2017].

Gaspar, A., Boyer, N. and Ejnioui, A. (2007). Role of the C language in current computing curricula part 1: survey analysis. *J. Comput. Sci. Coll.* 23, 2, 120-127.

Gilberg, R. and Forouzan, B. (1996). Comparison of student success in Pascal and C-language curriculums. *Proc. 27th SIGCSE tech. sympo. on Computer science education (SIGCSE '96)* pp.252-255.

Hosch, F. (1996). Java as a first language: an evaluation. *SIGCSE Bull. 28, 3.* pp.45-50.

King, K. (1997). The case for Java as a first language. *Proc. 35th Annual Southeast Regional Conf. (ACM-SE 35).* pp.124-131.

Sanders, I. and Langford, S. (2008). Students' perceptions of python as a first programming language at wits. *SIGCSE Bull. 40, 3.* pp.365-365.

Bates, B. (2004). C# as a first language: a comparison with C++. *J. Comput. Sci. Coll. 19, 3.* pp.89-95.

Kaplan, R. (2010). Choosing a first programming language. *Proc. conf. on Information technology education (SIGITE '10).* pp.163-164.

Mannila, L. and Raadt, M. (2006). An objective comparison of languages for teaching introductory programming. *Proc. 6th Baltic Sea conf. on Computing education research: Koli Calling 2006.* pp.32-37.

Bosse, Y. and Gerosa, M. (2017). Why is programming so difficult to learn?: Patterns of Difficulties Related to Programming Learning Mid-Stage. *SIGSOFT Softw. Eng. Notes 41, 6.* pp.1-6.

Biddle, R. and Tempero, E. (1998). Java pitfalls for beginners. *SIGCSE Bull. 30, 2.* pp.48-52.

Miller, C. and Settle, A. (2016). Some Trouble with Transparency: An Analysis of Student Errors with Object-oriented Python. *Proc. ACM Conf. Int. Computing Education Research (ICER '16).* pp.133-141.

Alimadadi, S. Mesbah, A. and Pattabiraman, K. (2016). Understanding asynchronous interactions in full-stack JavaScript. *Proc. Int. Conf. on Software Engineering (ICSE'16).* pp.1169-1180.

Freund, S. N. and Roberts, E. S. (1996). Thetis: An ansi c programming environment designed for introductory use. In *Proceedings of the Twenty-seventh SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '96, pages 300–304, New York, NY, USA. ACM.

Song, J. S., Hahn, S. H., Tak, K. Y., and Kim, J. H. (1997). An intelligent tutoring system for introductory c language course. *Comput. Educ.*, 28(2):93–102.

Osuka, T., Kobayashi, T., Atsumi, N., Mase, J., Yamamoto, S., Suzumura, N., and Agusa, K. (2012). CX-checker: A flexibly customizable coding checker for C. *Journal of Information Processing Society of Japan*, 53(2):590–600.

Kummerfeld, S. K. and Kay, J. (2003). The neglected battle fields of syntax errors. In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20*, ACE '03, pages 105–111, Darlinghurst, Australia, Australia. Australian Computer Society, Inc.

Kojima, Y. Arahori, Y., Gondow, K. (2015). Investigating the Difficulty of Commercial-level Compiler Warning Messages for Novice Programmers. *7th Int. Conf. on Computer Supported Education (CSEDU 2015).* pp.483–490.

Uchida, K. and Gondow, K. (2016). C-Helper: C Latent-error Static/Heuristic Checker for Novice Programmers. *8th Int. Conf. on Computer Supported Education (CSEDU 2016).* pp.321–329.

Gondow, K. Fukuyasu, N. and Arahori, Y. (2010). MieruCompiler: Integrated Visualization Tool with "Horizontal Slicing" for Educational Compilers. *41st ACM Technical Sympo. on Computer Science Education (SIGCSE 2010).* pp.7–11.

Gondow, K. and Ohba, M. (2007). Design and implementation of compact educational operating system udos by practice of middle-level abstraction, hhin middle layers and traceability. *The Transactions of the Institute of Electronics, Information and Communication Engineers.* J90-D[5], pp.1194-1208. (In Japanese) http://www.sde.cs.titech.ac.jp/∼gondow/udos [Online; accessed 31-Jan-2017].

Cox, R. Kaashoek, F. and Morris, R. (2006). Xv6, a simple Unix-like teaching operating system. https://pdos.csail.mit.edu/6.828/2012/xv6.html, [Online; accessed 31-Jan-2017].

Koenig, A. (1989). C Traps and Pitfalls. *Addison-Wesley Professional*. ISBN-10: 0201179288. 160 pages. https://www.amazon.co.jp/dp/0201179288/, [Online; accessed 31-Jan-2017].

Daconta, M. Smith, K. Avondolio, D. and Richardson, W. (2003). More Java Pitfalls: 50 New Time-Saving Solutions and Workarounds. *Wiley*. ISBN-10: 0471237515. 480 pages. https://www.amazon.co.jp/dp/0471237515/, [Online; accessed 31-Jan-2017].

Monbrea, M. (2014). Why I don't suggest JavaScript as a first programming language. *ITworld*, Jan. 9, 2014. http://www.itworld.com/article/2693386/why-i-don-t-suggest-javascript-as-a-first-programming-language.html, [Online; accessed 31-Jan-2017].

Guo, P. (2014). Python is now the most popular introductory teaching language at top U.S. universities. *BLOG@ACM*, July 7, 2014. http://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext [Online; accessed 31-Jan-2017].

Intel. (2017). Intel 64 and IA-32 Architectures Software Developer Manuals http://www.intel.com/products/processor/manuals/, [Online; accessed 31-Jan-2017].

TechLiebe (2014) Why to Learn C? https://techliebe.com/why-to-learn-c/ [Accessed: 24-Nov-2017]

Noordergraaf, L. Boldyshev, K. and Rideau, F. (2015) Assembly HOWTO, version 0.7 http://asm.sourceforge.net/howto/Assembly-HOWTO.html [Accessed: 24-Nov-2017]

Viswa (2014) Advantages and Disadvantages of C Language https://tekslate.com/advantages-and-disadvantages-of-c-language/ [Accessed: 24-Nov-2017]