Analysis

As far as I can tell, this program does just what it appears to do, which is absolutely nothing. Nonetheless, the fact that it even compiles is a shock. This program was the first one I ever saw that demonstrated such odd practices as radical changes of semantics before and after macro substitution, playing games with comment delimiters, and totally meaningless redefinitions of library and system call names.

In case you are wondering, no, this is not legal Standard C. In fact, none of the programs so far have been. Don't hold your breath for the next ones, either.

First Place Award

Sjoerd Mullender <sjoerd@cwi.nl>
Centrum voor Wiskunde en Informatica (Inst. for Math. and CS)
Amsterdam, Netherlands

Robbert van Renesse <rvr@cs.cornell.edu>
Cornell University
Ithaca, New York

```
/* Portable between VAX11 && PDP11 */
short main[] = {
        277, 04735, -4129, 25, 0, 477, 1019, 0xbef, 0, 12800,
        -113, 21119, 0x52d7, -1006, -7151, 0, 0x4bc, 020004,
        14880, 10541, 2056, 04010, 4548, 3044, -6716, 0x9,
        4407, 6, 5568, 1, -30460, 0, 0x9, 5570, 512, -30419,
        0x7e82, 0760, 6, 0, 4, 02400, 15, 0, 4, 1280, 4, 0,
        4, 0, 0, 0, 0x8, 0, 4, 0, ',', 0, 12, 0, 4, 0, '#',
        0, 020, 0, 4, 0, 30, 0, 026, 0, 0x6176, 120, 25712,
        'p', 072163, 'r', 29303, 29801, 'e' };
```

This is one of my favorite contest winners of all time. It was submitted by both authors when they were students (both held bachelor's degrees in mathematics at the time) at Vrije Universitait (Free University) in Amsterdam.

Author's Analysis

"When this program is compiled, the compiler places the array somewhere in memory, just like it places any compiled code somewhere in memory. Usually, the C startup code (`crt0.o`) calls a routine named `main`. The loader fills in the address in the startup code, but, at least on the old systems where this program ran, it doesn't know that the `main` in this program isn't code but data!

"When the program is run, the C startup code transfers control to the location `main`. The contents of the array just happen to be machine instructions for both a PDP-11 and a VAX.

"On the VAX, the routine `main` is called with the `calls` instruction. This instruction uses the first (2-byte) word of the called routine as a mask of registers that are

to be saved on the stack. In other words, on the VAX the first word can be anything. On the PDP, the first word is a branch instruction that branches over the VAX code. The PDP and VAX codes are thus completely separate.

"The PDP and VAX codes implement the same algorithm:

```
for (;;) {
        write(1, "  :-)\b\b\b\b", 9);
        delay();
}
```

"The result is that the symbols `:-)` move over the screen. `delay` is implemented differently on the PDP, where we used a nonexistent system call (`sys 55`), and on the VAX where we used a delay loop.

"My co-author, Robbert, and I had earlier written a similar program for an assignment on the PDP-11. Along came the first Obfuscated C Code Contest, and we decided that we should write a program like this, but make it run on two different architectures. We didn't think long about what the program should do, so it does something very simple.

"We started with writing the PDP code in assembler. We both knew PDP-11 assembler, so that was no problem. The assembler code we came up with is as follows:

```
pdp:
        mov     pc,r4
        tst     -(r4)
        sub     $9,r4
        mov     r4,0f
        mov     $1, r0
        sys     4; 0:0; 9
        mov     $1000, r2
1:
        sys     55
        sob     r2, 1b
        br      pdp
```

"This is not the code we originally wrote, but it is the code that we ultimately used in the program. The string to be printed is shared by the VAX and the PDP code and is located between the two sections. First, the program deals with figuring out the address of the string. Then the program counter is saved in a scratch register. Since the program counter points at the second instruction, we subtract 2 from the scratch register in the second instruction. Then we subtract the length of the string and store the result in the location with label 0. This has to do with the calling sequence of system calls on the PDP. Following the `sys` instruction is the system call number (4 for `write`), the address of the buffer (pointed to by label 0), and the length of the buffer (9). The file descriptor is in register `r0`. The rest of the code implements a delay loop. In each iteration, a nonexisting system call (55) slows things down.

"We assembled this program and extracted the machine code from the resulting object file. We used this code in the VAX part. Since neither of us was fluent in VAX assembly, we wrote the VAX code in C and massaged the compiler output. The VAX assembly program that we came up with is as follows:

```
vax:    .word 0400 + (pdp - vax) / 2 - 1
1:
        pushl   $9
        pushal  str
        pushl   $1
        calls   $3,write
        cvtwl   $32767,r2
2:
        decl    r2
        jneq    2b
        jbr     1b

write:  .word 0
        chmk    $4
        ret

str:    .ascii "  :-)\b\b\b\b"

pdp:    .word 4548, 3044, 58820, 9, 4407, 6, 5568, 1, 35076,
0, 9, 5570\, 512, 35117, 32386, 496
```

"The first word (after the label `vax`) is the PDP branch instruction. PDP branch instructions are octal 400 + the distance divided by 2. The string that both the PDP and VAX programs use is after the `str` label, and the PDP code is after the `pdp` label.

"On the VAX, the program pushes 9 (the length of the string), the address of the string and 1 (the file descriptor) on the stack and calls `write`. Since we didn't know the exact calling sequence for system calls, we just copied the source for the `write` system call stub into our program. After `write` finishes, the program executes a delay loop, after which it jumps back to the start of the program.

"We assembled this program, and extracted the machine code from the object file. After this we only had to convert the machine code to ASCII and write a little bit of C to glue everything together. We wanted to use different formats for each constant in the resulting array, and we wanted to choose the format randomly. So we wrote a program to choose an appropriate format at random. The program we wrote for that follows. This program actually also extracted the machine code from the object file.

```
#include <stdio.h>
#include <a.out.h>

main(argc, argv)
char **argv;
```

```
{
        register FILE *fp;
        register short pos = 0, c, n;
        register char *fmt;

        if (argc != 2) {
                fprintf(stderr, "Usage: %s file\n");
                exit(1);
        }
        if ((fp = fopen(argv[1], "r")) == NULL) {
                fprintf(stderr, "%s: can't open %s\n", argv[0],
argv[1]\);
                exit(2);
        }
        fseek(fp, (long) sizeof(struct exec), 0);
        printf("/* portable between VAX and PDP11 */\n\n");
        printf("short main[] = {\n");
        for (;;) {
                if (pos == 0) printf("\t");
                c = getc(fp) & 0377;
                if (feof(fp)) break;
                n = getc(fp) << 8 | c;
                switch (rand() % 5) {
                case 0:
                case 1:
                        fmt = "%d"; break;
                case 2:
                        fmt = "%u"; break;
                case 3:
                        fmt = "0%o"; break;
                case 4:
                        fmt = "0x%x"; break;
                }
        if (32 <= n && n < 127 && (rand() % 4)) fmt = "'%c'";
                printf(n < 8 ? "%d" : fmt, n);
                printf(",");
                if (pos++ == 8) {
                        printf("\n");
                        pos = 0;
                }
                else printf(" ");
        }
        printf("};\n");
}
```

"As can be seen, there is a slight preference for decimal, and also a character format is sometimes used, but only if the data is a printable ASCII character.

"When we ran this program, we were almost completely satisfied with the result. The only problem we had was that the program had chosen an octal representation for the first word. Since everybody knows what a PDP-11 branch instruction looks

like (everyone knows that the traditional magic word for an executable, 0407, is a PDP-11 branch), we changed that to decimal. After checking the size of the resulting program we saw that it was one byte too long. The limit was 512 bytes, and our program was 513 bytes. So we changed the word `and` in the comment to `&&`."