```c
/*
   echos.c
   echos.exe 1.0.1
     C.L.Distefano 2025-05-27T17:05
   Enhanced ECHO command for Windows
*/
#include <stdio.h>
#include <string.h>
#include <ctype.h>

unsigned int get_offset(unsigned int ch) {
    int offset = 0;
    if (ch >= '0' && ch <= '9') {
        offset = '0';
    }
    else if (ch >= 'A' && ch <= 'F') {
        offset = 'A' - 10;
    }
    else if (ch >= 'a' && ch <= 'f') {
        offset = 'a' - 10;
    }
    else {
        offset = 0;
    };
    return offset;
}

unsigned int is_decimal(char ch) {
    unsigned int rc = 1;
    if (!(ch >= '0' && ch <= '9')) rc = 0;
    return rc;
}

unsigned int is_hex(char ch) {
    unsigned int rc = 1;
    if (!(ch >= '0' && ch <= '9' || ch >= 'A' && \
            ch <= 'F' || ch >= 'a' && ch <= 'f')) rc = 0;
    return rc;
}

unsigned int is_octal(char ch) {
    unsigned int rc = 1;
    if (!(ch >= '0' && ch <= '7')) rc = 0;
    return rc;
}

void show_help() {
```

```
    printf("Usage: ECHOS [OPTION]... [STRING]...\nEcho STRING to standard output.\n\n
-n                do not output the trailing newline\n   \
-e                enable interpretation of backslash escapes\n   \
-E                disable interpretation of backslash escapes (default)\n   \
-U                force UTF-16 output for characters in range 128 - 255\n\
                      (implies -e)\n   -h /?  --help     display this help and exit\n
-v    --version  display version information and exit\n\nIf -e is in effect, \
the following sequences are recognized:\n\n   \\\\     backslash\n   \\\"     double \
quote (also works with other printable characters)\n   \\a     alert (BEL)\n   \
\\b     backspace\n   \\c     produce no further output\n   \\e     escape\n   \
\\f     form feed\n   \\n     newline\n   \\r     carriage return\n   \
\\t     horizontal tab\n   \\v     vertical tab\n   \
\\0NNN  byte with octal value NNN (1 to 3 digits)\n   \
\\dNNN  byte with decimal value NNN (1 to 3 digits)\n   \
\\xHH   byte with hexadecimal value HH (1 or 2 digits)\n");
    return;
}

void show_version() {
    printf("echos.exe 1.0.0  [C.L.Distefano]\n\
Enhanced ECHO command for Windows\n\"ECHOS --help\" gives usage details.\n");
    return;
}

int main(int argc, char *argv[]) {
    if (argc < 2) {
        system("echo");
        return 0;
    }
    long unsigned int i, j, k;
    char word1[12]; word1[0] = '\0';
    unsigned int escapes = 0, trailn = 1;
    char argstr[16384]; argstr[0] = '\0';
    char outstr[16300]; outstr[0] = '\0';
    char tnum[4]; tnum[0] = '\0';
    unsigned int uval = 0, utf16 = 0;
    char command[16500]; command[0] = '\0';
    if (argc == 2 && (strcmp(argv[1], "/?") == 0 || \
        strcmp(argv[1], "-h") == 0 || strcmp(argv[1], "--help") == 0)) {
            show_help();
            return 0;
    }
    if (argc == 2 && (strcmp(argv[1], "-v") == 0 || strcmp(argv[1], "--version") == 0))
        show_version();
        return 0;
    }
    strcat(word1, argv[1]);
```

```
    for (i = 0; i < strlen(argv[1]); i++) {
        word1[i] = toupper(word1[i]);
    }
    if (argc == 2 && (strcmp(word1, "ON") == 0 || strcmp(word1, "OFF") == 0)) {
        strcpy(command, "echo ");
        strcat(command, word1);
        system(command);
        return 0;
    }
    while (strcmp(argv[1], "-n") == 0 || strcmp(argv[1], "-e") == 0 ||
        strcmp(argv[1], "-E") == 0 || strcmp(argv[1], "-U") == 0) {
        if (strcmp(argv[1], "-n") == 0) {
            trailn = 0;
        }
        else if (strcmp(argv[1], "-e") == 0) {
            escapes = 1;
        }
        else if (strcmp(argv[1], "-U") == 0) {
            escapes = 1;
            utf16 = 1;
        }
        else {
            escapes = 0;
        }
        --argc; ++argv;
    }

    // No escapes, just print the string
    if (!escapes) {
        for (i = 1; i < argc; i++) {
            printf("%s", argv[i]);
            if (i < argc - 1) printf(" ");
        }
        if (trailn) printf("\n");
        return 0;
    }

    // Option -e is in effect: Escape sequences
    i = 0; // Concatenate args into a single string
    for (j = 1; j < argc; ++j) {
        for (k = 0; k < strlen(argv[j]); k++) {
            argstr[i] = argv[j][k];
            i++;
        }
        if (j < argc - 1) {
            argstr[i] = ' ';
            i++;
```

```
    }
}
argstr[i + 1] = '\0';

// Character substitutions
for (i = 0, j = 0; i < strlen(argstr); i++, j++) {
    if ((int)argstr[i] != 92) {
        outstr[j] = argstr[i];
        continue;
    }
    i++;
    switch (argstr[i]) {
    // Note: default case covers "\any_printable"
    //  including "\\" and "\""
        case 'a':
            outstr[j] = (char)7;
            break;
        case 'b':
            outstr[j] = (char)8;
            outstr[j + 1] = ' ';
            outstr[j + 2] = (char)8;
            j += 2;
            break;
        case 'c':
            outstr[j] = '\0';
            break;
        case 'e':
            outstr[j] = (char)27;
            break;
        case 'f':
            outstr[j] = '\f';
            break;
        case 'n':
            outstr[j] = (char)13;
            outstr[j + 1] = (char)10;
            j += 1;
            break;
        case 'r':
            outstr[j] = (char)13;
            break;
        case 't':
            outstr[j] = (char)9;
            break;
        case 'v':
            outstr[j] = (char)11;
            break;
        case '0': // octal Ascii
```

```
            tnum[0] = argstr[i + 1];
            if (i + 2 < strlen(argstr)) {
                tnum[1] = argstr[i + 2];}
            else {
                tnum[1] = '\0';
            }
            if (i + 3 < strlen(argstr)) {
                tnum[2] = argstr[i + 3];}
            else {
                tnum[2] = '\0';
            }
            if (!is_octal(tnum[0])) {
                outstr[j] = argstr[i - 1];
                outstr[j + 1] = argstr[i];
                j += 1;
                break;
            }
            if (!is_octal(tnum[1])) {
                uval  = tnum[0] - get_offset(tnum[0]);
                if (uval > 255) {
                    outstr[j] = tnum[0];
                    break;
                }
                outstr[j] = (char)(uval);
                i += 1;
                break;
            }
            if (!is_octal(tnum[2])) {
                uval = 8 * (tnum[0] - get_offset(tnum[0]));
                uval = uval + tnum[1] - get_offset(tnum[1]);
                if (uval > 255) {
                    outstr[j] = argstr[i - 1];
                    outstr[j + 1] = argstr[i];
                    j += 1;
                    break;
                }
                outstr[j] = (char)uval;
                i += 2;
                break;
            }
            uval = 64 * (tnum[0] - get_offset(tnum[0]));
            uval = uval + (8 * (tnum[1] - get_offset(tnum[1])));
            uval = uval + tnum[2] - get_offset(tnum[2]);
            if (uval > 255) {
                outstr[j] = tnum[0];
                i += 1;
                break;
```

```
236             }
237             outstr[j] = (char)uval;
238             i += 3;
239             break;
240         case 'd': // decimal Ascii
241             tnum[0] = argstr[i + 1];
242             if (i + 2 < strlen(argstr)) {
243                 tnum[1] = argstr[i + 2];}
244             else {
245                 tnum[1] = '\0';
246             }
247             if (i + 3 < strlen(argstr)) {
248                 tnum[2] = argstr[i + 3];}
249             else {
250                 tnum[2] = '\0';
251             }
252             if (!is_decimal(tnum[0])) {
253                 outstr[j] = argstr[i - 1];
254                 outstr[j + 1] = argstr[i];
255                 j += 1;
256                 break;
257             }
258             if (!is_decimal(tnum[1])) {
259                 uval = tnum[0] - get_offset(tnum[0]);
260                 if (uval > 255) {
261                     outstr[j] = tnum[0];
262                     break;
263                 }
264                 outstr[j] = (char)uval;
265                 i += 1;
266                 break;
267             }
268             if (!is_decimal(tnum[2])) {
269                 uval = 10 * (tnum[0] - get_offset(tnum[0]));
270                 uval = uval + tnum[1] - get_offset(tnum[1]);
271                 if (uval > 255) {
272                     i += 1;
273                     outstr[j] = tnum[0];
274                     break;
275                 }
276                 outstr[j] = (char)uval;
277                 i += 2;
278                 break;
279             }
280             if (tnum[0] < '0' || tnum[0] > '2') {
281                 outstr[j] = argstr[i - 1];
282                 outstr[j + 1] = argstr[i];
```

```
283                 j += 1;
284                 break;
285             }
286             uval = 100 * (tnum[0] - get_offset(tnum[0]));
287             uval = uval + (10 * (tnum[1] - get_offset(tnum[1])));
288             uval = uval + tnum[2] - get_offset(tnum[2]);
289             if (uval > 255) {
290                 outstr[j] = argstr[i - 1];
291                 outstr[j + 1] = argstr[i];
292                 j += 1;
293                 break;
294             }
295             outstr[j] = (char)uval;
296             i += 3;
297             break;
298         case 'x': // hexadecimal Ascii
299             tnum[0] = argstr[i + 1];
300             if (i + 2 < strlen(argstr)) {
301                 tnum[1] = argstr[i + 2];}
302             else {
303                 tnum[1] = '\0';
304             }
305             if (!is_hex(tnum[0])) {
306                 outstr[j] = argstr[i - 1];
307                 outstr[j + 1] = argstr[i];
308                 j += 1;
309                 break;
310             }
311             if (!is_hex(tnum[1])) {
312                 uval = tnum[0] - get_offset(tnum[0]);
313                 if (uval > 255) {
314                     outstr[j] = tnum[0];
315                     i += 1;
316                     break;
317                 }
318                 outstr[j] = (char)uval;
319                 i += 1;
320                 break;
321             }
322             uval = 16 * (tnum[0] - get_offset(tnum[0]));
323             uval = uval + tnum[1] - get_offset(tnum[1]);
324             if (uval > 255) {
325                 outstr[j] = argstr[i - 1];
326                 outstr[j + 1] = argstr[i];
327                 j += 1;
328                 break;
329             }
```

```
            outstr[j] = (char)uval;
            i += 2;
            break;
        default: // covers "\any_printable", including "\\"
            outstr[j] = argstr[i];
            break;
    }
}
// The rubber meets the road:
outstr[j + 1] = '\0';
if (!utf16 && strstr(outstr, "\x1B[") == NULL && \
        strstr(outstr, "\x0B") == NULL) {
        printf("%s", outstr);
}
else {
    strcat(command, "@for /F \"tokens=*\" %A in \x28\"");
    if (utf16) strcat(command, "\x0B");
    strcat(command, outstr);
    strcat(command, "\"\x29 do @echo|set/p=%A");
    command[strlen(command) + 1] = '\0';
    system(command);
}
if (trailn) printf("\n");
return 0;
}

```