```
/*
 * PURPOSE:
 *  Show how to convert from string to int and float,
 *  two common operations for processing argv[] arguments.
 *
 *  Also illusrate the use of stderr and the significant
 *  of the value returned from main(). (0 for OK, non−zero
 *  for errors).
 */
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
    int i;
    float f;

    if (argc < 3)
    {
            fprintf(stderr, "usage: %s <int> <float>\n", argv[0]);
            return 1;
    }

    i = atoi(argv[1]);
    f = atof(argv[2]);
    printf("i is %d and f is %f\n", i, f);

    return 0;
}
```

```
/*
 * PURPOSE:
 *   To illustrate:
 *   − C function supports call−by−value only.
 *   − Can simulate call−by−reference by using pointers.
 */

#include <stdio.h>
void change_i_wrong(int i)
{
    i = 10;
}

void change_i_right(int *i)
{
    *i = 10;
}

void change_ptr_right(int **i)
{
    *i = malloc(sizeof(int));
}

int main()
{
    int i = 1;
    printf("i initialized to %d\n", i);
    change_i_wrong(i);
    printf("after change_i_wrong i is %d\n", i);
    change_i_right(&i);
    printf("after change_i_right i is %d\n", i);

    int *p;
    p = &i;
    printf("pointer p initialized to %p\n", p);
    change_ptr_right(&p);
    printf("after change_ptr_right %p\n", p);
}
```

```
/*
 * PURPOSE:
 *  Illustrate dangling pointers −−cannot return address
 *  of local variables to outside, since after a function
 *  terminates, the local variables are deallocated.
 */

#include <stdio.h>

char *foo()
{
        char y = 'A';
        return &y;
}

char bar()
{
        char y = 'B';
}

int main()
{
    char *z = foo();
    bar();
    printf("printing %c\n", *z);
}
```

```
/*
 * PURPOSE:
 *  Illustrate the danger of using strcpy without
 *  checking for length.  This is an example of the
 *  notorious buffer overflow security flaw.
 */
#include <stdio.h>

int main(int argc, char *argv[])
{
        char buf[255];
        if (argc < 2)
        {
                fprintf(stderr, "usage: %s <input>\n");
                return 1;
        }
        strcpy(buf, argv[1]);
        return 0;
}
```

```
/*
 * PURPOSE:
 *   Introduce 2D array and #define constants
 *   Introduce preprocessor and "−E" option of gcc.
 */
#include <stdio.h>
#define X 0
#define Y 1
#define TOP_LEFT 0
#define LOWER_RIGHT 1
#define TOP_RIGHT 2
#define LOWER_LEFT 3

int main()
{
        int rect[4][2] = {{0,0}, {10,10}, {0,10}, {10,0}};
        printf("lower right corner is (%d,%d)\n",
            rect[LOWER_RIGHT][X], rect[LOWER_RIGHT][Y]);
        return 0;
}
```

```
#include <stdio.h>
#include <stdlib.h>

/*
 * PURPOSE:
 *  Shows array of pointers and its treatment as a 2D array.
 */

int main()
{
    float *f[10];
    int **a;
    int i, j;

    a = (int **)malloc(sizeof(int *)*10);
    for (i = 0; i < 10; i++)
    {
            a[i] = (int *)malloc(sizeof(int)*10);
            f[i] = (float *)malloc(sizeof(float)*10);
            for (j = 0; j < 10; j++)
            {
                    f[i][j] = i*10 + j;
                    a[i][j] = i*10 + j;
            }
    }

    printf("%.3f\n", f[1][4]);
    printf("%d\n", a[1][4]);
    return 0;
}
```

```
/*
 * PURPOSE
 *  Passing 1D and 2D array as function arguments.
 *  Note the differences between int (*a)[10] and int *a[10].
 *  int *a[10]
 *     => *a[10] is an int
 *     => a[10] is pointer to int
 *     => a is an array of pointer to int.
 *  int (*a)[10]
 *     => (*a)[10] is an int
 *     => *a is an array of int.
 *     => a is a pointer to an array of int.
 */

// void init_vector(char a[], int length) − OK
// void init_vector(char *a, int length) − OK
void init_vector(char a[10], int length)
{
    int i;
    for (i = 0; i < length; i++)
    {
        a[i] = 0;
    }
}

// void init_matrix(int (*a)[10], int w, int h) − OK
// void init_matrix(int **a, int w, int h) − NOT OK
void init_matrix(int a[][10], int w, int h)
{
    int i, j;
    for (i = 0; i < w; i++)
    {
        for (j = 0; j < h; j++)
        {
            a[i][j] = 0;
        }
    }
}

int main()
{
    char vector[10];
    int matrix[10][10];
    init_vector(vector, 10);
    init_matrix(matrix, 10, 10);
    return 0;
}
```

```
/*
 * PURPOSE
 *  Illustrate different methods from stdio.h (FILE, fopen,
 *  fgets, fclose, fflush, stdin, stdout, NULL) and the
 *  danger of not checking if fopen returns NULL.
 */

#include <stdio.h>
int main(int argc, char *argv[])
{
    FILE *in;
    char buf[255];

    if (argc < 2) {
        in = stdin;
    } else {
        in = fopen(argv[1], "r");
    }

    while (fgets(buf, 255, in) != NULL) {
        char *curr = buf;
        while (*curr) {
            if (*curr == 'Y') {
                *curr = 'K';
            }
            curr++;
        }
        fprintf(stdout, "%s", buf);
        fflush(stdout);
    }

    fclose(in);
    return 0;
}
```

```
/*
 * PURPOSE:
 *   Illustrate the sscanf function (and an innocent looking
 *   printf bug).
 */
#include <stdio.h>

int main()
{
    char *s = "ABC 123                    890";
    char c[10];
    int i, j;

    sscanf(s, "%s %d %d", c, &i, &j);
    printf("c is %s\n", c);
    printf("i is %s\n", i);
    printf("j is %s\n", j);
    return 0;
}
```

```
/*
 * PURPOSE
 *  Illustrate the use of fread and fwrite for reading
 *  and writing binary files, and fseek to move the
 *  file pointer to a specific location.
 */

#include <stdio.h>

int main(int argc, char *argv[])
{
    if (argc < 2)
    {
        fprintf(stderr, "usage: %s <mp3 file>\n", argv[1]);
        return 1;
    }
    else
    {
        FILE *in = fopen(argv[1], "rb");
        if (!in)
        {
            fprintf(stderr, "error: cannot open %s for reading\n", argv[1]);
            return 2;
        }
        char c[33];
        fseek(in, −128, SEEK_END);
        fread(c, 3, 1, in);
        c[3] = 0;
        if (c[0] != ’T’ && c[1] != ’A’ && c[2] != ’G’)
        {
            fprintf(stderr,
                "%s does not have a valid ID3v1 tag (tag is %s)\n",
                argv[1], c);
            exit(1);
        }
        fseek(in, 30, SEEK_CUR); /* skip over song title */
        fread(c, 30, 1, in);
        c[30] = 0;
        fprintf(stdout, "%s\n", c);
        fclose(in);
    }
}
```