```
/*
 * SetStar.c.   AmigaMail '*' wildcard example. Pure code if pragmas are used.
 * Thursday 12-Jul-91 09:33:01, Ewout
 *
 * Compiled with SAS/C 5.10a: lc -cfis -v -d0 -b0 SetStar.c blink from SetStar.o
 * to SetStar \ lib lib:amiga.lib       ; if you don't have pragmas
 *
 */
#include <exec/types.h>
#include <dos/dosextens.h>
#include <dos/rdargs.h>

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>

/*
 * undef PRAGMAS if you don't have them #define PRAGMAS
 */
#ifdef PRAGMAS
#include <pragmas/exec_pragmas.h>
#include <pragmas/dos_pragmas.h>
#else
struct ExecBase *SysBase;
struct DosLibrary *DOSBase;

#endif

static UBYTE   *VersTag = "\0$VER: SetStar 37.1 (12.07.91)";


VOID            main(VOID);
UWORD           StrLen(UBYTE *);

VOID
main(VOID)
{
#ifdef PRAGMAS
    struct DosLibrary *DOSBase;

#endif

    struct RDArgs  *readargs;
    LONG            rargs[2];
    UWORD           on, off;

#ifndef PRAGMAS
    /* set up SysBase */
    SysBase = (*((struct Library **) 4));
#endif

    /* Fail silently if < 37 */
    if (DOSBase = (struct DosLibrary *) OpenLibrary("dos.library", 37))
    {
        rargs[0] = 0;
        rargs[1] = 0;

        /* See the DOS Autodocs for more information about ReadArgs() */
        if (readargs = ReadArgs("ON/S,OFF/S", rargs, NULL))
        {
            on = (UWORD) (rargs[0]);
            off = (UWORD) (rargs[1]);

            /*
             * The RNF_WILDSTAR bit in the rn_Flags field indicates whether the
             * '*' should be treated as wildcard or not.
             *
             * Show current setting if both ON & OFF or specified or neither.
             */
            if (on == off)
            {
                if (DOSBase->dl_Root->rn_Flags & RNF_WILDSTAR)
                    rargs[0] = (LONG) "ON";
                else
                    rargs[0] = (LONG) "OFF";
                VFPrintf(Output(), "Wildstar is %s\n", rargs);
            }
            else
            {
                if (on)
                    DOSBase->dl_Root->rn_Flags |= RNF_WILDSTAR;
                else
                    DOSBase->dl_Root->rn_Flags &= ~RNF_WILDSTAR;
            }
            FreeArgs(readargs);
        }
        else
            PrintFault(IoErr(), NULL);
        CloseLibrary((struct Library *) DOSBase);
    }
}
```

```
;/* Pattern.c.   AmigaMail pattern matching example.  Compiled with SAS/C 5.10a:
lc -cfis -v -d0 -b0 -j73 Pattern.c
blink from Pattern.o to Pattern lib lib:amiga.lib
quit
*/

#include <exec/types.h>
#include <exec/memory.h>
#include <dos/dos.h>
#include <dos/dosasl.h>
#include <dos/rdargs.h>

#include <clib/exec_protos.h>
#include <clib/dos_protos.h>
#include <clib/utility_protos.h>

/* define pragmas if you have them
#define PRAGMAS */
#ifdef PRAGMAS
#include <pragmas/exec_pragmas.h>
#include <pragmas/dos_pragmas.h>
#include <pragmas/utility_pragmas.h>
#else
struct ExecBase *SysBase;
struct DosLibrary *DOSBase;
struct Library *UtilityBase;
#endif

VOID            main(VOID);
UWORD           StrLen(UBYTE *);

VOID main(VOID)
{
#ifdef PRAGMAS
    struct DosLibrary *DOSBase;
    struct Library *UtilityBase;

#endif

    struct RDArgs  *readargs;
    LONG            rargs[3];
    LONG            vargs[4];
    UBYTE         **strings;
    UBYTE          *pattern, *parsebuffer;
    UWORD           case_sensitive, buffersize;
    LONG            iswild, success;
    COUNT           i;

#ifndef PRAGMAS
    /* set up SysBase */
    SysBase = (*((struct Library **) 4));
#endif

    /* Fail silently if < 37 */
    if (DOSBase = (struct DosLibrary *) OpenLibrary("dos.library", 37))
    {
        UtilityBase = DOSBase->dl_UtilityBase;

        /* See the DOS Autodocs for more information about ReadArgs() */
        if (readargs = ReadArgs("PATTERN/A,CASE/S,STRINGS/A/M", rargs, NULL))
        {

            /* The pattern. */
            pattern = (UBYTE *) (rargs[0]);

            /*
             * Case sensitive or not? (default not. Note filename matching
             * should ALWAYS be case insensitive).
             */
            case_sensitive = (UWORD) (rargs[1]);

            /* Pointer to array of strings to match */
            strings = (UBYTE **) (rargs[2]);

            /* Get a buffer big enough to hold all the tokens */
            buffersize = StrLen(pattern) * 3;

            if (parsebuffer = AllocMem(buffersize, MEMF_CLEAR))
            {

                /* Parse the pattern, according to case sensitivity flag */
                if (case_sensitive)
                    iswild = ParsePattern(pattern, parsebuffer, buffersize);
                else
                {
                    /* make pattern uppercase in case of character classes */
                    i = 0;
                    while (pattern[i])
                        pattern[i] = ToUpper(pattern[i++]);
                    iswild = ParsePatternNoCase(pattern, parsebuffer, buffersize);
                }

                /*
                 * -1 if ParsePattern() failed, 0 for no wildcards, 1 for
                 * wildcards. For this I don't care if the supplied pattern had
                 * wildcards or not.
                 */
                if (iswild != -1)
                {
                    /* The array of strings is terminated with a NULL */
                    while (*strings)
                    {

                        /*
                         * MatchPattern() returns 1 for a successful match, 0
                         * for no match
                         */
                        if (case_sensitive)
                            success = MatchPattern(parsebuffer, *strings);
                        else
                            success = MatchPatternNoCase(parsebuffer, *strings);
                        if (success)
                        {
                            vargs[0] = (LONG) * strings;
                            VFPrintf(Output(), "Match: %s\n", vargs);
                        }
                        else
                        {
                            if (IoErr() != 0)
                            {
                                VFPrintf(Output(), "Overflow\n", NULL);
                                break;
                            }
                        }
                        strings++;
                    }
                }
                else
                    PrintFault(ERROR_BAD_TEMPLATE, pattern);
                FreeMem(parsebuffer, buffersize);
            }
            else
                PrintFault(ERROR_NO_FREE_STORE, NULL);
            FreeArgs(readargs);
        }
        else
            PrintFault(IoErr(), NULL);
        CloseLibrary((struct Library *) DOSBase);
    }
}

UWORD StrLen(UBYTE * string)
{
    UBYTE          *length = string + 1;

    while (*string++ != '\0');
    return ((UWORD) (string - length));
}
```