```
;/* ListDir.c - Amiga Mail simple ExAll() example.
lc -cfis -v -d0 -b0 -j73 ListDir.c
blink from ListDir.o to ListDir lib lib:amiga.lib ;if you don't have pragmas
quit
 *
 * Pure code if pragmas are used.
 * Tuesday, 16-Jul-91 16:21:14, Ewout
 *
 * Compiled with SAS/C 5.10a
 */

#include <exec/memory.h>
#include <dos/dosextens.h>
#include <dos/rdargs.h>
#include <dos/exall.h>

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

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

#endif

/* Buffersize to receive filenames in */
#define BUFFERSIZE 512

VOID            main(VOID);

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

#endif
    struct RDArgs  *readargs;
    LONG            rargs[2];
    struct ExAllControl *excontrol;
    struct ExAllData *ead, *buffer;
    UBYTE          *source;
    BPTR            sourcelock;
    BOOL            exmore;
    LONG            error;

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

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

        if (readargs = ReadArgs("DIRECTORY/A", rargs, NULL))
        {

            source = (UBYTE *) rargs[0];

            if (buffer = AllocMem(BUFFERSIZE, MEMF_CLEAR))
            {

                if (sourcelock = Lock(source, SHARED_LOCK))
                {
                    if (excontrol = AllocDosObject(DOS_EXALLCONTROL, NULL))
                    {
                        excontrol->eac_LastKey = 0;
                        do
                        {

                            exmore = ExAll(sourcelock,
                                           buffer,
                                           BUFFERSIZE,
                                           ED_NAME,
                                           excontrol);
                            error = IoErr();
                            if ((exmore == NULL &&
                                (error != ERROR_NO_MORE_ENTRIES)))
                                break;

                            if (excontrol->eac_Entries == 0)
                                continue;

                            ead = buffer;
                            do
                            {

                                /* Check for CTRL-C */
                                if (SetSignal(0L, SIGBREAKF_CTRL_C) &
                                        SIGBREAKF_CTRL_C)
                                {
                                    error = ERROR_BREAK;
                                    exmore = FALSE;
                                    break;
                                }

                                rargs[0] = (LONG) ead->ed_Name;
                                VFPrintf(Output(), "%s\n", rargs);

                                ead = ead->ed_Next;
                            } while (ead);
                        } while (exmore);

                        if (error != ERROR_NO_MORE_ENTRIES)
                            PrintFault(error, NULL);

                        FreeDosObject(DOS_EXALLCONTROL, excontrol);
                    }
                    else
                        PrintFault(ERROR_NO_FREE_STORE, NULL);

                    UnLock(sourcelock);
                }
                else
                    PrintFault(IoErr(), source);

                FreeMem(buffer, BUFFERSIZE);
            }
            else
                PrintFault(ERROR_NO_FREE_STORE, NULL);
            FreeArgs(readargs);

        }
        else
            PrintFault(IoErr(), NULL);
        CloseLibrary((struct Library *) DOSBase);
    }
}
```

```
;/* ListDir2.c   AmigaMail seconds ExAll() example.
lc -cfis -v -d0 -b0 -j73 ListDir2.c
blink from ListDir2.o to ListDir2 lib lib:amiga.lib ;if you don't have pragmas
quit
 *
 * Pure code if pragmas are used.
 * Tuesday, 16-Jul-91 16:21:14, Ewout
 *
 * Compiled with SAS/C 5.10a
 */
#include <exec/memory.h>
#include <dos/dosextens.h>
#include <dos/rdargs.h>
#include <dos/exall.h>
#include <utility/hooks.h>

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

/* undef PRAGMAS if you don't have them */
#define PRAGMAS
#undef 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

/* Buffersize to receive filenames in */
#define BUFFERSIZE 512


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

/* SAS/C specific, use asm stub otherwise */
#define ASM __asm
#define REG(x) register __## x
BOOL ASM  ExAllHook(REG(a0) struct Hook * hook,
                    REG(a1) struct ExAllData * data,
                    REG(a2) LONG * datatype);

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

#endif
    struct RDArgs  *readargs;
    LONG            rargs[4];
    struct ExAllControl *excontrol;
    struct ExAllData *ead, *buffer;
    struct Hook     exallhook;
    UBYTE          *pattern, *parsebuffer;
    BPTR            sourcelock;
    BOOL            exmore;
    COUNT           i;
    LONG            parselength, type, error;

#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;

        rargs[1] = 0L;
        rargs[2] = 0L;
        if (readargs = ReadArgs("PATTERN/A,DIRS/S,FILES/S", rargs, NULL))
        {

            pattern = (UBYTE *) rargs[0];

            /*
             * If DIRS or files not specified or both, accept
             * both files and directories
             */
            if (rargs[1] == rargs[2])
                type = 0;
            else
            {
                /* Accept only directories */
                if (rargs[1])
                    type = 1;
                /* Accept only files */
                else
                    type = -1;
            }

            parselength = StrLen(pattern) * 3;
            if (parsebuffer = AllocMem(parselength, MEMF_CLEAR))
            {

                /* Make pattern uppercase for possible character classes */
                i = 0;
                while (pattern[i])
                    pattern[i] = ToUpper(pattern[i++]);

                if ((ParsePatternNoCase(pattern, parsebuffer, parselength)) != -1)
                {

                    if (buffer = AllocMem(BUFFERSIZE, MEMF_CLEAR))
                    {

                        sourcelock =
                                ((struct Process *) FindTask(NULL))->pr_CurrentDir;

                        if (excontrol = AllocDosObject(DOS_EXALLCONTROL, NULL))
                        {

                            exallhook.h_Entry = ExAllHook;
                            exallhook.h_Data = (VOID *) type;

                            excontrol->eac_MatchString = parsebuffer;
                            excontrol->eac_MatchFunc = &exallhook;

                            do
                            {

                                exmore = ExAll(sourcelock,
                                               buffer,
                                               BUFFERSIZE,
                                               ED_TYPE,
                                               excontrol);
                                error = IoErr();
                                if ((exmore == NULL &&
                                    (error != ERROR_NO_MORE_ENTRIES)))
                                    break;

                                if (excontrol->eac_Entries == 0)
                                    continue;

                                ead = buffer;
                                do
                                {

                                    /* Check for CTRL-C */
                                    if (SetSignal(0L, SIGBREAKF_CTRL_C) &
                                        SIGBREAKF_CTRL_C)
```

```
                                    {
                                        error = ERROR_BREAK;
                                        exmore = FALSE;
                                        break;
                                    }

                                    rargs[0] = (LONG) ead->ed_Name;
                                    VFPrintf(Output(), "%s\n", rargs);

                                    ead = ead->ed_Next;
                                } while (ead);
                            } while (exmore);

                            if (error != ERROR_NO_MORE_ENTRIES)
                                PrintFault(error, NULL);

                            FreeDosObject(DOS_EXALLCONTROL, excontrol);
                        }
                        else
                            PrintFault(ERROR_NO_FREE_STORE, NULL);

                        FreeMem(buffer, BUFFERSIZE);
                    }
                    else
                        PrintFault(ERROR_NO_FREE_STORE, NULL);
                }
                else
                    PrintFault(ERROR_BAD_TEMPLATE, NULL);
                FreeMem(parsebuffer, parselength);
            }
            else
                PrintFault(ERROR_NO_FREE_STORE, NULL);
            FreeArgs(readargs);

        }
        else
            PrintFault(IoErr(), NULL);
        CloseLibrary((struct Library *) DOSBase);
    }
}

BOOL ASM
ExAllHook(REG(a0) struct Hook * hook,
          REG(a1) struct ExAllData * data,
          REG(a2) LONG * datatype)
{
    LONG            neededfiletype = (LONG) hook->h_Data;
    BOOL            success = TRUE;

    if (neededfiletype != 0)
    {
        if (data->ed_Type > 0 && neededfiletype < 0)
            success = FALSE;
        if (data->ed_Type < 0 && neededfiletype > 0)
            success = FALSE;
    }
    return (success);
}


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

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

```
;/* Find.c - Amiga Mail ExAll() example.
lc -cfis -v -d0 -b0 -j73 Find.c
blink from Find.o to Find lib lib:amiga.lib; if you don't have pragmas
quit
 *
 * Pure code if pragmas are used.
 * Tuesday, 16-Jul-91 16:21:14, Ewout
 *
 * Compiled with SAS/C 5.10a:
 */
#include <exec/memory.h>
#include <dos/dosextens.h>
#include <dos/rdargs.h>
#include <dos/exall.h>

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

/* undef PRAGMAS if you don't have them */
/* #define PRAGMAS */
#undef 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

/* ExAll buffersize to receive data in */
#define BUFFERSIZE 512
/* Default buffersize for hold fully qualified filenames */
#define NAMEBUFFERSIZE 512

/* Used to pass data around to functions */
struct FindControl
{
  struct DosLibrary *fc_DOSBase;
  UBYTE          *fc_Parsebuffer;       /* Buffer which contains the parsed pattern */
  ULONG           fc_Parselength;       /* The length of this buffer */
  UBYTE          *fc_Namebuffer;        /* Buffer to hold the filename */
  ULONG           fc_Namelength;        /* The length of that buffer */
  BOOL            fc_Files;  /* BOOLEAN which tells if we should only look for files */
  BOOL            fc_Dirs;   /* BOOLEAN which tells if we should only look for dirs */
  BOOL            fc_All;    /* ALL keyword? */
};

static UBYTE   *VersTag = "\0$VER: Find 37.1 (16.07.91)";

LONG            main(VOID);
LONG            ScanDirectory(struct FindControl *, UBYTE *);
BOOL            IsAssign(struct FindControl *, UBYTE *);
LONG            MultiScanDirectory(struct FindControl *, UBYTE *);
UWORD           StrLen(UBYTE *);

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

#endif
  struct RDArgs  *readargs;
  LONG            rargs[6];
  struct FindControl *fc;
  UBYTE          *pattern, **directories;
  struct Process *process;
  APTR            windowptr;
  COUNT           i;
  LONG            rc = 0, error = 0, fatalerror = 0;
```

```
#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;

    rargs[0] = 0L;
    rargs[1] = 0L;
    rargs[2] = 0L;
    rargs[3] = 0L;
    rargs[4] = 0L;
    rargs[5] = 0L;

    if (readargs =
        ReadArgs("PATTERN/A,DIRECTORY/A/M,FILES/S,DIRS/S,ALL/S,BUFFER/K/N",
                 rargs,
                 NULL))
    {

      if (fc = AllocMem(sizeof(struct FindControl), MEMF_CLEAR))
      {

#ifdef PRAGMAS
        fc->fc_DOSBase = DOSBase;
#endif

        pattern = (UBYTE *) rargs[0];

        fc->fc_Parselength = StrLen(pattern) * 3;
        if (fc->fc_Parsebuffer = AllocMem(fc->fc_Parselength, MEMF_CLEAR))
        {

          /* Make pattern uppercase for possible character classes */
          i = 0;
          while (pattern[i])
            pattern[i] = ToUpper(pattern[i++]);

          if ((ParsePatternNoCase(pattern,
                                  fc->fc_Parsebuffer,
                                  fc->fc_Parselength)) != -1)
          {

            directories = (UBYTE **) rargs[1];

            fc->fc_Files = (BOOL) rargs[2];
            fc->fc_Dirs = (BOOL) rargs[3];
            fc->fc_All = (BOOL) rargs[4];

            /*
             * Both set or cleared, clear both anyway. Easier checking later on.
             */
            if (fc->fc_Files == fc->fc_Dirs)
              fc->fc_Files = fc->fc_Dirs = FALSE;

            if (rargs[5])
              fc->fc_Namelength = *((LONG *) rargs[5]);

            if (fc->fc_Namelength < NAMEBUFFERSIZE || fc->fc_Namelength > 4096)
              fc->fc_Namelength = NAMEBUFFERSIZE;

            if (fc->fc_Namebuffer = AllocMem(fc->fc_Namelength, MEMF_CLEAR))
            {
              process = (struct Process *) FindTask(NULL);
              windowptr = process->pr_WindowPtr;
              process->pr_WindowPtr = (APTR) - 1L;

              while (*directories)
              {

                /*
                 * Check if this is a standalone assign which appears in the assign
                 * list?
                 */
                if (IsAssign(fc, *directories))
                  error = MultiScanDirectory(fc, *directories++);
                else
                  error = ScanDirectory(fc, *directories++);

                if (error != 0)
                  break;
              }

              process->pr_WindowPtr = windowptr;

              FreeMem(fc->fc_Namebuffer, fc->fc_Namelength);
            }
            else
              fatalerror = ERROR_NO_FREE_STORE;
          }
          else
            fatalerror = ERROR_BAD_TEMPLATE;

          FreeMem(fc->fc_Parsebuffer, fc->fc_Parselength);
        }
        else
          fatalerror = ERROR_NO_FREE_STORE;
        FreeMem(fc, sizeof(struct FindControl));
      }
      else
        fatalerror = ERROR_NO_FREE_STORE;

      FreeArgs(readargs);

    }
    else
      fatalerror = IoErr();

    /*
     * Error handling: To be informative, errors are shown while scanning, so the
     * file name which caused the error can be displayed. Other errors are shown
     * here. Errors which occured in the main loop are considered fatal, others
     * (except BREAK) just error.
     */

    if (fatalerror)
    {
      error = fatalerror;
      PrintFault(fatalerror, NULL);
    }

    SetIoErr(error);
    if (error != 0)
    {
      if (fatalerror)
        rc = RETURN_FAIL;
      else if (error == ERROR_BREAK)
        rc = RETURN_WARN;
      else
        rc = RETURN_ERROR;
    }

    CloseLibrary((struct Library *) DOSBase);
  }
  else
    rc = RETURN_FAIL;
  return (rc);
}


LONG
ScanDirectory(struct FindControl * fc, UBYTE * source)
{
#ifdef PRAGMAS
  struct DosLibrary *DOSBase = fc->fc_DOSBase;
```

```
#endif
  LONG            vargs[1];
  struct ExAllControl *excontrol;
  struct ExAllData *ead, *buffer;
  BPTR            sourcelock, namelock, olddirlock;
  BOOL            exmore;
  LONG            error;

  /*
   * Because this function may be recursively, get a fresh buffer per function call.
   */
  if (buffer = AllocMem(BUFFERSIZE, MEMF_CLEAR))
  {

    /* Get a lock on the start directory and make it the current directory */
    if (sourcelock = Lock(source, SHARED_LOCK))
    {
      olddirlock = CurrentDir(sourcelock);

      if (excontrol = AllocDosObject(DOS_EXALLCONTROL, NULL))
      {

        do
        {
          /* Get both file name and type to support FILES/DIRS kewords */
          exmore = ExAll(sourcelock, buffer, BUFFERSIZE, ED_TYPE, excontrol);
          error = IoErr();
          if ((exmore == NULL && (error != ERROR_NO_MORE_ENTRIES)))
          {
            PrintFault(error, source);
            break;
          }
          if (excontrol->eac_Entries == 0)
            continue;

          ead = buffer;
          do
          {

            /* Check for CTRL-C */
            if (SetSignal(0L, SIGBREAKF_CTRL_C) & SIGBREAKF_CTRL_C)
            {
              error = ERROR_BREAK;
              PrintFault(error, NULL);
              exmore = FALSE;
              break;
            }

            /*
             * Check if this one matches. If it does see if it is of the right type.
             */
            if (MatchPatternNoCase(fc->fc_Parsebuffer, ead->ed_Name))
            {
              if ((ead->ed_Type < 0 && fc->fc_Dirs == FALSE)
                  || (ead->ed_Type > 0 && fc->fc_Files == FALSE))
              {
                /* It is. Lock it and get the fully qualified file name */
                if (namelock = Lock(ead->ed_Name, SHARED_LOCK))
                {
                  if ((NameFromLock(namelock,
                                    fc->fc_Namebuffer,
                                    fc->fc_Namelength)) == DOSTRUE)
                  {
                    vargs[0] = (LONG) fc->fc_Namebuffer;
                    VFPrintf(Output(), "%s\n", vargs);
                  }
                  else
                  {
                    error = IoErr();
                    PrintFault(error, ead->ed_Name);
                  }
                  UnLock(namelock);
                }
                else
                {
                  error = IoErr();
                  PrintFault(error, ead->ed_Name);
                }
              }
            }

            /*
             * If the ALL keyword is used and this is a directory, step in it by
             * calling this function recursively.
             */
            if (ead->ed_Type > 0 && fc->fc_All)
            {
              error = ScanDirectory(fc, ead->ed_Name);
              if (error != 0)
              {
                exmore = FALSE;
                break;
              }
            }
            ead = ead->ed_Next;
          } while (ead);
        } while (exmore);

        FreeDosObject(DOS_EXALLCONTROL, excontrol);

      }
      else
        error = ERROR_NO_FREE_STORE;

      CurrentDir(olddirlock);
      UnLock(sourcelock);
    }
    else
    {
      error = IoErr();
      PrintFault(error, source);
    }
    FreeMem(buffer, BUFFERSIZE);
  }
  else
    error = ERROR_NO_FREE_STORE;

  if (error == ERROR_NO_MORE_ENTRIES)
    error = 0;
  else if (error == ERROR_NO_FREE_STORE)
    PrintFault(error, NULL);

  return (error);
}

BOOL
IsAssign(struct FindControl * fc, UBYTE * name)
{
#ifdef PRAGMAS
  struct DosLibrary *DOSBase = fc->fc_DOSBase;
  struct Library *UtilityBase = DOSBase->dl_UtilityBase;

#endif
  struct DosList *doslist;
  UBYTE          *assignname;
  UCOUNT          assignlength;
  LONG            position;
  BOOL            result = FALSE;

  /* First lets check if this resembles a devicename. */
  position = SplitName(name, ':', fc->fc_Namebuffer, 0, fc->fc_Namelength);

  if (position != -1)
  {
    /* Hmmm. */
    if (name[position] == '\0')
    {

      /*
       * I guess it does. Lets see if we can find it in the assign list. Keep the
```

```
       * DosList locked as briefly as possible. This shouldn't take long.
       */
      if (doslist = AttemptLockDosList(LDF_ASSIGNS | LDF_READ))
      {
        while (doslist = NextDosEntry(doslist, LDF_ASSIGNS))
        {

          /* It's a BPTR */
          assignname = (UBYTE *) BADDR(doslist->dol_Name);
          assignlength = assignname[0];

          if ((Strnicmp(assignname + 1, fc->fc_Namebuffer, assignlength)) == 0)
          {
            /* Yup, it is. */
            result = TRUE;
            break;
          }
        }
        UnLockDosList(LDF_ASSIGNS | LDF_READ);
      }    /* Can't lock DosList, don't bother */
    }
  }
  return (result);
}

LONG
MultiScanDirectory(struct FindControl * fc, UBYTE * source)
{
#ifdef PRAGMAS
  struct DosLibrary *DOSBase = fc->fc_DOSBase;

#endif
  struct DevProc *cproc = NULL;
  struct MsgPort *filesystemtask;
  LONG            error;

  filesystemtask = GetFileSysTask();

  do
  {
    /* Find handler */
    if (cproc = GetDeviceProc(source, cproc))
    {
      SetFileSysTask(cproc->dvp_Port);
      if ((NameFromLock(cproc->dvp_Lock,
           fc->fc_Namebuffer,
           fc->fc_Namelength)) == DOSTRUE)
      {
        error = ScanDirectory(fc, fc->fc_Namebuffer);
      }
      else
      {
        error = IoErr();
        PrintFault(error, source);
      }

      if (error != 0)
        break;
    }
    else
    {
      error = IoErr();
      PrintFault(error, source);
    }

    /* Handle multi-assign */
  } while (cproc && (cproc->dvp_Flags & DVPF_ASSIGN));

  SetFileSysTask(filesystemtask);
  if (cproc)
    FreeDeviceProc(cproc);

  return (error);
}
```

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

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

```
;/* ListPattern.c - AmigaMail MatchFirst()/MatchNext() example.
lc -cfis -v -d0 -b0 -j73 ListPattern.c
blink from ListPattern.o to ListPattern lib lib:amiga.lib ;if you don't have pragmas
quit
 */

#include <exec/memory.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
#undef PRAGMAS
#ifdef PRAGMAS
#include <pragmas/exec_pragmas.h>
#include <pragmas/dos_pragmas.h>
#else

struct ExecBase *SysBase;
struct DosLibrary *DOSBase;
#endif

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

VOID
main(VOID)
{

#ifdef PRAGMAS
    struct DosLibrary *DOSBase;
#endif

    struct RDArgs  *readargs;
    LONG            rargs[1];
    LONG            vargs[4];
    UBYTE          *pattern;
    struct AnchorPath *anchorpath;
    LONG            error;

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

    /* Fail silently if < 37 */
    if (DOSBase = (struct DosLibrary *) OpenLibrary("dos.library", 37))
    {
        /* See the DOS Autodocs for more information about ReadArgs() */
        if (readargs = ReadArgs("PATTERN/A", rargs, NULL))
        {
            pattern = (UBYTE *) rargs[0];
            if (anchorpath = AllocMem(sizeof(struct AnchorPath) + 512, MEMF_CLEAR))
            {
                anchorpath->ap_Strlen = 512;
                anchorpath->ap_BreakBits = SIGBREAKF_CTRL_C;

                if ((error = MatchFirst(pattern, anchorpath)) == 0)
                {
                    do
                    {
                        vargs[0] = (LONG) anchorpath->ap_Buf;
                        VFPrintf(Output(), "%s\n", vargs);
                    } while ((error = MatchNext(anchorpath)) == 0);
                }

                MatchEnd(anchorpath);

                if (error != ERROR_NO_MORE_ENTRIES)
                    PrintFault(error, NULL);

                FreeMem(anchorpath, sizeof(struct AnchorPath) + 512);
            }
            FreeArgs(readargs);
        }
        else
            PrintFault(IoErr(), NULL);
        CloseLibrary((struct Library *) DOSBase);
    }
}
```

```
;/* DirComp.c - AmigaMail Directory Compare example using MatchFirst()/Next().
lc -cfis -v -d0 -b0 -j73 DirComp.c
blink from DirComp.o to DirComp lib lib:amiga.lib ; if you don't have pragmas
quit
 *
 * Pure code if pragmas are used.
 * Monday, 15-Jul-91 16:07:31, Ewout
 *
 * Compiled with SAS/C 5.10a
 */
#include <exec/memory.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 */
#undef PRAGMAS
#ifdef PRAGMAS
#include <pragmas/exec_pragmas.h>
#include <pragmas/dos_pragmas.h>
#else
struct ExecBase *SysBase;
struct DosLibrary *DOSBase;

#endif

/* Default size of buffer to build full targetpaths in */
#define BUFFERSIZE 256

static UBYTE   *VersTag = "\0$VER: DirComp 37.1 (15.07.91)";

LONG            main(VOID);
LONG            GetPath(UBYTE * path, UBYTE * buffer, LONG buffersize);
UBYTE          *ItsWild(UBYTE * string);
UWORD           StrLen(UBYTE *);

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

#endif
  struct RDArgs  *readargs;
  LONG            rargs[5], vargs[5];
  UBYTE          *source, *target;
  ULONG           buffersize = 0;
  UBYTE          *sourcedir, *targetdir;
  UBYTE          *textbuffer, *tmp, *tmp1, *tmp2;
  struct AnchorPath *anchorpath;
  struct FileInfoBlock *fib, *targetfib;
  struct Process *process;
  APTR            wptr;
  BPTR            dirlock, filelock;
  BOOL            checkdatestamp, all;
  LONG            date, error, rc = 0;

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

  /* Fail silently if < 37 */
  if (DOSBase = (struct DosLibrary *) OpenLibrary("dos.library", 37))
  {
    rargs[0] = 0L;
    rargs[1] = 0L;
    rargs[2] = 0L;
    rargs[3] = 0L;
    rargs[4] = 0L;

    if (readargs = ReadArgs("SOURCE/A,TARGET/A,DATE/S,ALL/S,BUFFER/K/N", rargs, NULL))
    {

      source = (UBYTE *) rargs[0];
      target = (UBYTE *) rargs[1];
      checkdatestamp = (BOOL) rargs[2];
      all = (BOOL) rargs[3];

      if (!(sourcedir = AllocMem(StrLen(source) + 129, MEMF_CLEAR)))
        error = ERROR_NO_FREE_STORE;
      else
      {
        /* 128 bytes to print informative text */
        textbuffer = sourcedir + StrLen(source) + 1;

        /* use user specified buffersize if indicated */
        if (rargs[4])
          buffersize = *((LONG *) rargs[4]);
        if (buffersize < BUFFERSIZE || buffersize > 4096)
          buffersize = BUFFERSIZE;

        if (!(targetdir = AllocMem(buffersize, MEMF_CLEAR)))
          error = ERROR_NO_FREE_STORE;
        else
        {
          if (!(targetfib = AllocDosObject(DOS_FIB, NULL)))
            error = ERROR_NO_FREE_STORE;
          else
          {

            /*
             * Check if source and target are valid.
             *
             * Separate source path from pattern (if any). Use the source path figure
             * out what to append to the target.
             */

            /* No requesters */
            process = (struct Process *) FindTask(NULL);
            wptr = process->pr_WindowPtr;
            process->pr_WindowPtr = (APTR) - 1L;

            if ((error = GetPath(source, sourcedir, StrLen(source) + 1) == 0))
            {
              if (!(dirlock = Lock(sourcedir, SHARED_LOCK)))
                error = IoErr();
              else
              {
                UnLock(dirlock);
                if (!(dirlock = Lock(target, SHARED_LOCK)))
                  error = IoErr();
                else
                {
                  UnLock(dirlock);

                  if (anchorpath = AllocMem(sizeof(struct AnchorPath) + buffersize,
                                            MEMF_CLEAR))
                  {
                    anchorpath->ap_Strlen = buffersize;

                    /* Allow to break on CTRL-C */
                    anchorpath->ap_BreakBits = SIGBREAKF_CTRL_C;

                    if ((error = MatchFirst(source, anchorpath)) == 0)
                    {

                      do
                      {
                        fib = &(anchorpath->ap_Info);

                        /*
                         * APF_DIDDIR indicates that we used returned from a
                         * directory. In that case we clear both APF_DIDDIR and
                         * APF_DODIR, so we can start afresh with the next one.
                         */
```

```
                        if (anchorpath->ap_Flags & APF_DIDDIR)
                          anchorpath->ap_Flags &= ~(APF_DODIR | APF_DIDDIR);
                        else
                        {

                          /*
                           * Make a filename for the target directory. First copy
                           * targetname into buffer.
                           */
                          targetdir[0] = '\0';
                          tmp = targetdir;
                          tmp1 = target;
                          while (*tmp++ = *tmp1++);

                          /* Skip sourcename in ap_Buf */
                          tmp1 = sourcedir;
                          tmp2 = anchorpath->ap_Buf;
                          while (*tmp1++ == *tmp2++);
                          /* Skip back 1 if not after a separator */
                          if (*(tmp2 - 1) != '/')
                            tmp2--;

                          /*
                           * We hit the source itself, don't compare it, but enter
                           * it.
                           */
                          if (*tmp2 == 0)
                          {
                            anchorpath->ap_Flags |= APF_DODIR;
                            continue;
                          }

                          /* Build it */
                          if (AddPart(targetdir, tmp2, buffersize - 1))
                            vargs[0] = (LONG) targetdir;
                          else
                          {
                            PrintFault(ERROR_NO_FREE_STORE, NULL);
                            break;
                          }

                          /* Lock it and check it out */
                          if (filelock = Lock(targetdir, SHARED_LOCK))
                          {
                            if ((Examine(filelock, targetfib)) == DOSTRUE)
                            {
                              textbuffer[0] = '\0';

                              /*
                               * To get nice output without work I use AddPart() to
                               * add differences to the textbuffer.
                               */
                              if (targetfib->fib_DirEntryType
                                  != fib->fib_DirEntryType)
                                AddPart(textbuffer, "of different type", 128);
                              else
                              {
                                if (targetfib->fib_Size < fib->fib_Size)
                                  AddPart(textbuffer, "smaller", 128);
                                else if (targetfib->fib_Size > fib->fib_Size)
                                  AddPart(textbuffer, "larger", 128);

                                if (checkdatestamp)
                                {
                                  date = CompareDates((struct DateStamp *)
                                        & (fib->fib_Date),
                                      (struct DateStamp *) & (targetfib->fib_Date));
                                  if (date < 0)
                                    AddPart(textbuffer, "older", 128);
                                  else if (date > 0)
                                    AddPart(textbuffer, "newer", 128);
                                }
                              }
                              if (*textbuffer != NULL)
                              {
                                vargs[1] = (LONG) textbuffer;
                                VFPrintf(Output(), "%s: object %s\n", vargs);
                              }
                            }
                            else
                              PrintFault(IoErr(), targetdir);
                            UnLock(filelock);
                          }
                          else
                          {
                            PrintFault(IoErr(), targetdir);

                            /*
                             * If and error occured on a directory name, don't enter
                             * it.
                             */
                            if (fib->fib_DirEntryType > 0)
                              continue;

                          }

                          /*
                           * If the ALL keyword has been used and this is a directory
                           * enter it by setting the APF_DODIR flag.
                           */

                          if (fib->fib_DirEntryType > 0 && all != FALSE)
                            anchorpath->ap_Flags |= APF_DODIR;

                        }

                      } while ((error = MatchNext(anchorpath)) == 0);
                    }

                    MatchEnd(anchorpath);

                    if (error == ERROR_NO_MORE_ENTRIES)
                      error = 0;

                    FreeMem(anchorpath, sizeof(struct AnchorPath) + buffersize);
                  }
                }
              }
              /* Reset windowpointer */
              process->pr_WindowPtr = wptr;
            }
            else
              PrintFault(error, NULL);
            FreeDosObject(DOS_FIB, targetfib);
          }
          FreeMem(targetdir, buffersize);
        }
        FreeMem(sourcedir, StrLen(sourcedir) + 129);
      }
      FreeArgs(readargs);
    }
    else
      error = IoErr();

    SetIoErr(error);
    if (error)
    {
      PrintFault(error, NULL);
      if (error = ERROR_BREAK)
        rc = RETURN_WARN;
      else
        error = RETURN_FAIL;
    }
    CloseLibrary((struct Library *) DOSBase);
  }
  return (rc);
}
```

```
LONG
GetPath(UBYTE * path, UBYTE * buffer, LONG buffersize)
{
  UBYTE          *pathpart, *filepart;
  UBYTE          *tmp1, *tmp2;
  BPTR            lock;
  struct FileInfoBlock *fib;
  LONG            error = 0;

  /* Open own copy of dos.library if pragmas are used so it's standalone */
#ifdef PRAGMAS
  struct Library *DOSBase;

  if (!(DOSBase = OpenLibrary("dos.library", 36)))
    return (1);
#endif

  /*
   * If there seems to be no path, the pathpart will point to the filepart too, so we
   * need to check for that.
   */
  filepart = FilePart(path);
  pathpart = PathPart(path);

  /*
   * This also handles cases where there is only a volume/device name, only a
   * directory name or a combo of those.
   */
  if (pathpart == path)
  {

    /*
     * There seems to be only one component. Copy it if it is not wild. Caller will
     * have to check whether if it exists and if it is a file or directory.
     */
    if (!(ItsWild(pathpart)))
      pathpart = NULL;
  }

  if (pathpart != path)
  {

    /*
     * If pathpart equals filepart (pointer wise) then there is only one component
     * (possible preceeded by a volume name).
     */
    if (pathpart == filepart)
    {
      if (!(ItsWild(pathpart)))
        pathpart = NULL;
    }
    else
    {
      /* Try to lock it to determine if the last component is a directory. */
      if (lock = Lock(path, SHARED_LOCK))
      {
        if (fib = AllocMem(sizeof(struct FileInfoBlock), MEMF_CLEAR))
        {
          if ((Examine(lock, fib)) == DOSTRUE)
          {
            /* Hey it's a directory after all */
            if (fib->fib_DirEntryType > 0)
              pathpart = NULL;
          }
          FreeMem(fib, sizeof(struct FileInfoBlock));
        }
        UnLock(lock);
      }                                         /* else treat it as a filename */
    }

    /* Copy the pathpart in the buffer */
    tmp1 = buffer;
    tmp2 = path;
    while ((*tmp1++ = *tmp2++) && (tmp2 != pathpart))
    {
      if (tmp1 == (buffer + buffersize))
      {
        error = ERROR_NO_FREE_STORE;
        break;
      }
    }
    *tmp1 = '\0';                                /* NULL terminate. */
  }

#ifdef PRAGMAS
  CloseLibrary(DOSBase);
#endif
  return (error);
}

UBYTE          *
ItsWild(UBYTE * string)
{
  static UBYTE   *special = "#?*%([|";
  UBYTE          *tmp = string;
  COUNT           i;

  do
  {
    for (i = 0; special[i] != '\0'; i++)
    {
      if (*tmp == special[i])
        return (tmp);
    }
    tmp++;
  } while (*tmp);

  return (NULL);
}

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

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