Sourcecode: Example3.c

COLLABORATORS			
	TITLE : Sourcecode: Example3.c		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Sourcecode: Example3.c

1.1 Example3.c

```
/*********************************************************/
/*                                                       */
/* Amiga C Encyclopedia (ACE)           Amiga C Club (ACC) */
/* --------------------------           ------------------ */
/*                                                       */
/* Manual:  AmigaDOS                    Amiga C Club       */
/* Chapter: Files                       Tulevagen 22       */
/* File:    Example3.c                  181 41  LIDINGO    */
/* Author:  Anders Bjerin               SWEDEN             */
/* Date:    93-03-15                                     */
/* Version: 1.0                                          */
/*                                                       */
/*   Copyright 1993, Anders Bjerin - Amiga C Club (ACC)  */
/*                                                       */
/* Registered members may use this program freely in their */
/*     own commercial/noncommercial programs/articles.   */
/*                                                       */
/*********************************************************/

/* This program simply writes two strings to a file, moves the */
/* file cursor back some characters and then collects some     */
/* characters in the middle of the file. This example does    */
/* exactly what is explained in picture  ReadWrite.pic .      */



/* Include the dos library definitions: */
#include <dos/dos.h>

/* Now we include the necessary function prototype files:         */
#include <clib/dos_protos.h>       /* General dos functions...    */
#include <stdio.h>                 /* Std functions [printf()...] */
#include <stdlib.h>                /* Std functions [exit()...]   */
#include <string.h>                /* Std functions [strlen()...] */



/* The size of our buffer: (Remember to never */
```

```
/* read more data than can fit your buffer!)  */
#define MAX_LENGTH  50

/* Number of characters that we will read: */
#define READ_LENGTH  7



/* Set name and version number: */
UBYTE *version = "$VER: AmigaDOS/InputOutput/Example3 1.0";



/* Declared our own function(s): */

/* Our main function: */
int main( int argc, char *argv[] );



/* Main function: */

int main( int argc, char *argv[] )
{
  /* A "BCPL" pointer to our file: */
  BPTR my_file;

  /* The strings we want to save: */
  UBYTE *string1 = "HELLO";
  UBYTE *string2 = " WORLD";

  /* Some memory where the data we read can be saved: */
  UBYTE my_buffer[ MAX_LENGTH ];

  /* Store here the number of characters (bytes) actually read/written: */
  long actual;

  /* Old file cursor position: */
  int old_pos;



  /* 1. Try to open file "RAM:Introduction.doc" as a new file: */
  /* (If the file does not exist, it will be created. If it,   */
  /* on the the other hand, exist, it will be overwritten.)    */
  my_file = Open( "RAM:Introduction.doc", MODE_NEWFILE );

  /* Have we opened the file successfully? */
  if( !my_file )
  {
    /* Inform the user: */
    printf( "Error! Could not open the file!\n" );

    /* Exit with an error code: */
    exit( 20 );
  }
```

```
/* The file has now been opened: */
printf( "1. File open!\n" );



/* 2. We have now opened a file and the file cursor is pointing    */
/* to the first character (byte) in our new file. We can now write */
/* the first string to the file:                                   */
actual = Write( my_file, string1, strlen( string1 ) );

/* Were all characters successfully saved? */
if( actual != strlen( string1 ) )
{
  /* NO! Problems while writing! */
  printf( "Writing error while saving the first string!\n" );
}
else
  printf( "2. String 1 written!\n" );



/* 3. Add the second string to the file: (Since we have not */
/* moved the file cursor since we wrote the first string    */
/* this string will be added directly after the first one.) */
actual = Write( my_file, string2, strlen( string2 ) );

/* Were all characters successfully saved? */
if( actual != strlen( string2 ) )
{
  /* NO! Problems while writing! */
  printf( "Writing error while saving the second string!\n" );
}
else
  printf( "3. String 2 written!\n" );



/* 4. Move the file cursor three characters from the   */
/* beginning of the file: (We could equally well move  */
/* the file cursor eight characters backwards from the */
/* current position or the end of the file.)           */
old_pos = Seek( my_file, 3, OFFSET_BEGINNING );

/* Tell the uer where the file cursor was and is now: */
printf( "4. File cursor moved!\n" );
printf( "     Old position: %d\n", old_pos );
printf( "     New position: 3\n" );



/* 5. Collect "READ_LENGTH" (7) number of characters: */
actual = Read( my_file, my_buffer, READ_LENGTH );

/* Did we get all characters? */
if( actual != READ_LENGTH )
{
  /* Problems! Could not read all data as expected. */
```

```
    /* We have either reached an unexpected EOF or    */
    /* there was an error while we tried to read:     */
    if( actual == -1 )
      printf( "Error while reading!\n" );
    else
      printf( "Unexpected EOF!\n" );
  }
  else
  {
    /* Note that we only collected some characters in the middle */
    /* of the file. There will therefore not be any NULL sign at */
    /* the end of the collected string, so we have to put one    */
    /* one there ourself. (If we do not put a NULL sign at the   */
    /* end of the string and then later tries to print it there  */
    /* will probably come a lot of junk after the collected      */
    /* characters. The prinf() function will continue to print   */
    /* haracters until a NULL sign is reached.)                  */
    my_buffer[ READ_LENGTH ] = NULL;

    /* Some extra information: A "collection of characters" is */
    /* like a string but with no NULL sign at the end, while a */
    /* "string" is a collection of characters with a NULL sign */
    /* at the end. Whenever you are using string functions in  */
    /* C you must make sure that you realy have a string, and  */
    /* not just a collection of characters. Otherwise a lot of */
    /* unexpected things might happen!                         */

    /* Print the string: */
    printf( "5. Read \"%s\"!\n", my_buffer );
  }



  /* 6. Close the file: (Since we can not do very much if the */
  /* function will fail to close the file we simply ignore    */
  /* any returned errors.)                                    */
  Close( my_file );
  printf( "6. File closed!\n" );

  /* The End! */
  exit( 0 );
}
```