Example.Doc

COLLABORATORS

	TITLE : Example.Doc	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 7, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Example.Doc

1.1 Menu

```
                  +-------------------+
| ACE v2.3 Examples |
+-------------------+
```

Introduction

FILES

MESSAGE WRITE

SGN

ABS

FIX

MID$

SHARED

ADDRESS

FONT

MOD

SHL

ALLOC

FOR..NEXT

MOUSE

SHR

AND

```
FORWARD
   MOUSE ON
SHORTINT

ARG$

FRE

MSGBOX

SINGLE

ARGCOUNT
   GADGET
NAME

SIZEOF

AREA
   GADGET CLOSE
NOT

SIN

AREAFILL

GADGET MOD

OCT$

SLEEP

ASC
   GADGET ON
ON..GOTO/GOSUB

SLEEP FOR

ASSEM
   GADGET WAIT
OPEN
   SOUND

ATN

GOSUB..RETURN

OPTION

SPACE$

BACK

GOTO

OR
```

SPC

BEEP

HANDLE

PAINT

SQR

BEVELBOX

HEADING

PALETTE

STICK

BIN$

HEX$
   PATTERN
STOP

BREAK

HOME

PEEKx

STR$

CALL

IF

PENDOWN

STRIG

CASE
   IFF
PENUP

STRING

CHDIR
   IFF CLOSE
POINT

STRING$

CHR$
   IFF OPEN
POKEx

```
STRUCT

CINT
   IFF READ
POS

STYLE

CIRCLE

IMP

POTX

SUB..END SUB

CLEAR ALLOC

INKEY$

POTY

SWAP

CLNG

INPUTBOX

PRINT

SYSTEM

CLOSE

INPUTBOX$

PRINT #

TAB

CLS

INPUT$

PRINTS

TAN

COLOR

INPUT

PSET

TIME$

CONST
```

```
   INPUT #
PTAB

TIMER

COS

INSTR

RANDOMIZE
   TIMER ON

CSNG

INT

READ

TRANSLATE$

CSRLIN

KILL

REM

TURN

CSTR

LEFT$

REPEAT..UNTIL

TURNLEFT

DATA

LEN

RESTORE

TURNRIGHT

DATE$

LET

RIGHT$

UCASE$

DAY
   LIBRARY
RND

VAL
```

```
DECLARE

LINE

SADD

VARPTR
  DEF FN
LINE INPUT

SAY
   WAVE

DEFxxx

LOCATE

SCREEN

WHILE..WEND

DIM

LOF

SCREEN BACK

WINDOW

END

LOG

SCREEN CLOSE

WINDOW CLOSE

EOF

LONGINT

SCREEN FORWARD

WINDOW ON

ERR

MENU

SCROLL

WINDOW OUTPUT

ERROR

MENU CLEAR
```

```
   SERIAL
WRITE

EQV

MENU ON
   SERIAL CLOSE
XCOR

EXIT FOR

MENU WAIT
   SERIAL OPEN
YCOR

EXIT SUB

MESSAGE CLOSE
   SERIAL READ
XOR

EXP

MESSAGE OPEN
   SERIAL WRITE

EXTERNAL

MESSAGE READ

SETHEADING

FILEBOX$

MESSAGE WAIT

SETXY
```

1.2 Introduction

Sometimes it is helpful to see an actual example of a command or function in use, as opposed to simply reading about it in a manual or wading through large amounts of code.

This AmigaGuide document was created with the intention of providing a supplement to the existing ACE documentation.

Peter Zielinski suggested that something like this might be a good idea. I too had been toying with doing this kind of thing for some time. I approached John Stiwinter about it (who has converted all other ACE docs to AmigaGuide format) and he eagerly took the project on.

The creation and content of this guide was a joint effort by

John and I, with John doing much of the early work.

Most programs and code fragments in this guide are short,
with only a few exceptions.

You may notice that some code is reused for a number of
commands and also that the guide will sometimes link to
a particular example program in the ACE distribution.

This first version of the example guide is just that: a first
approximation to the intended product. Please report any bugs
you find and give us constructive criticism as to how this
guide can be improved.

David Benn

The examples contained in this document have been designed
and written with the novice in mind, although some of the
code snippets do touch on more advanced topics we hope that
we have maintained an air of clarity in all of the examples.

Every effort has been made to ensure that all examples are
bug free and work on a variety of platforms. All of the
code snippets found in this document are considered to be
in the public domain, so feel free to use anything you wish
in your own programs.

Good luck and happy ACEing.

John Stiwinter

1.3 ABS

```
a = 20
b = -500
PRINT ABS(A);
PRINT ABS(B);
END
```

The following is displayed on the screen:

```
20  500
```

1.4 ADDRESS

```
ADDRESS myAddress, yourAddress
```

```
Declares two variables of type address. This is the same
as declaring long integer variables but more descriptive
when the data to be stored is memory location values.
```

1.5 ALLOC

```
MemLoc& = ALLOC(5000,5)
PRINT MemLoc&
END


MemLoc& holds the starting address of 5000 bytes of cleared
public memory.
```

1.6 AND

```
a = 98
b = 243
c = a AND b
PRINT c
END

The number 98 is given as a result.

ie..
         98 = 01100010
        243 = 11110011


98 AND 243 = FTTFFFTF

           = 01100010
           = 98

Here's another example:

         Truth = 1 < 2 AND 3 > 0

The variable "Truth" now holds the value -1 (ie. TRUE).
```

1.7 ARGCOUNT

```
FOR N=0 to ARGCOUNT
  PRINT ARG$(N)
NEXT

This code will display the command line arguments for the
current program. Since ARG$(0) is the name of the program,
this will be displayed first. If there are no command line
arguments, only the program name will be displayed.
```

1.8 AREAFILL

```
'Start at a location of 50 pixels across and 50 pixels down.

'move across 250 pixels from our last position.
'move down 100 pixels.
'now move back 250 pixels (to the left side of the screen).
'and finally, move up 100 pixels to complete the shape.
'fill the area with the current output color.

AREA (50,50)
AREA STEP (250,0)
AREA STEP (0,100)
AREA STEP (-250,0)
AREA STEP (0,-100)
AREAFILL
END

This will draw a filled square.
```

1.9 ASC

```
letter$="Hello"
a=ASC("A")
PRINT a, ASC(letter$)
END

65      72
```

will be printed to the screen.

1.10 ASSEM

```
ASSEM
  addq #4,sp
END ASSEM

This has the effect of directly inserting the assembly
code "addq #4,sp" into the assembly target file produced
by ACE.
```

1.11 ATN

```
INPUT "ENTER a number: ",a
b = ATN(a)
PRINT "The Arc tangent of"a"is"b
END

The output will look like this:
```

```
Enter a number: 24
The Arc tangent of 24 is 1.5291538
```

1.12 BACK

```
{*
** The following program demonstrates typical usage of
** ACE's Turtle Graphics commands.
*}

SUB shape_a
    COLOR 2
    PENDOWN
    FORWARD 60
    TURNLEFT 30
    FORWARD 60
    TURNLEFT 150
    FORWARD 60
    TURNLEFT 30
    FORWARD 60
    TURNLEFT 150
END SUB

SUB shape_b
    COLOR 3
    PENDOWN
    BACK 30
    TURNRIGHT 30
    BACK 30
    TURNRIGHT 150
    BACK 30
    TURNRIGHT 30
    BACK 30
    TURNRIGHT 150
END SUB

SCREEN 1,640,200,3,2
WINDOW 1,,(0,0)-(640,200),32,1
x=320 : y=100
FOR t = 1 TO 100
  CIRCLE (320,100),1,3
  LINE (x-int(t*3.2),y-t)-(x+int(t*3.2),y+t),1,BF
NEXT t
PENUP
SETXY x,y
  FOR t = 0 to 360 STEP 15
    TURN t
    shape_a
  NEXT t
  PENUP
  FOR t = 0 to 360 STEP 5
    TURN t
    shape_b
  NEXT t
```

```
WHILE INKEY$="":WEND
WINDOW CLOSE 1
SCREEN CLOSE 1
END
```

1.13 BEEP

```
FOR t = 1 to 3
        BEEP
        SLEEP FOR .5
NEXT t
PRINT "Do I have your attention?"
END

This program will beep 3 times and print a message
on the screen.
```

1.14 BEVELBOX

```
DEFLNG a-z
WINDOW 1,"ACE Clock v1",(220,75)-(324,111),2
MENU 1,0,1,"Project"
MENU 1,1,1,"About..."
MENU 1,2,1,"Quit","Q"
ON MENU gosub handle_menu : MENU ON
BEVELBOX (5,3)-(90,19),2
finished = 0
WHILE not finished
  LOCATE 2,3:PRINT TIME$;
  SLEEP FOR .1
WEND
WINDOW CLOSE 1
STOP
handle_menu:
  x = MENU(0):y=MENU(1)
  IF x = 1 AND y = 1 THEN result = MsgBox("by David Benn","Continue")
  IF x = 1 AND y = 2 THEN finished = -1
RETURN
END

This program creates a digital clock which can be moved around
with the mouse and exited with Amiga-Q or via the Project menu.

The main display area is a recessed bevel-box.
```

1.15 BIN$

```
INPUT "Enter a number: "a
PRINT a"equals "BIN$(a)" binary."
END
```

```
This will produce the following:

Enter a number: 65
65 equals 1000001 binary.
```

1.16 CALL

```
SUB doub(n)
  PRINT n*2
END SUB

CALL doub(3)
END

This code displays:

 6
```

1.17 CASE

```
CONST default = -1
x=3
CASE
  x=1     : PRINT "one"
  x=2     : PRINT "two"
  default : PRINT "default"
END CASE
END

Since x is equal to neither 1 nor 2 here, and the final
statement evaluates to true, "default" is printed.
```

1.18 CHDIR

```
CHDIR "RAM:"
FILES
END

This will display all the files in the RAM disk.
```

1.19 CHR$

```
INPUT "Enter a number between 64 and 123: ",a
b$ = CHR$(a)
PRINT "The character of ASCII value"a"is "b$
END
```

```
The output will look like this:

Enter a number between 64 and 123: 65
The character of ASCII value 65 is A
```

1.20 CINT

```
a = 28.75
b = 101.50
c = -23.875
PRINT CINT (a)
PRINT CINT (b)
PRINT CINT (c)
END

This program will print:

29
102
-24
```

1.21 CIRCLE

```
CIRCLE (320,100),75
END

This program will draw a circle at the center of the screen
(assuming a high-resolution screen) with a radius of 75 pixels.
```

1.22 CLEAR ALLOC

```
PRINT "Free:";FRE(-1)

REPEAT
  addr& = ALLOC(100000);
  IF addr& <> 0 THEN PRINT " ** OK **"; ELSE PRINT " ** NOT OK **";
  PRINT TAB(15);"Free:";FRE(-1);
  PRINT TAB(30);"Lgst CHIP:";FRE(2);
  PRINT TAB(50);"Lgst FAST:";FRE(3)
UNTIL addr& = 0

PRINT "Deallocating memory..."
CLEAR ALLOC
PRINT "Free:";FRE(-1)
END

This program allocates memory in 100,000 byte chunks until memory
is either exhausted or too fragmented for ALLOC to return a valid
address (non-zero value). CLEAR ALLOC is then invoked to free all
```

memory allocated via ALLOC.

The FRE(n) function is used to indicate how much memory is free
at each stage of the program run.

1.23 CLNG

```
a = 13.75
b = 12.50
PRINT CLNG(a*b)
END
```

The program responds with 172 as opposed to 171.875

1.24 CLS

```
FOR t = 1 to 10
        PRINT "Hello!"
NEXT t

CLS

END
```

CLS simply clears the screen after 10 "hello world"
strings are displayed.

1.25 COLOR

```
PRINT "Hello World!"
COLOR 3,0
PRINT "I've changed my color!"
COLOR 1,0
PRINT "Good-bye..."
END
```

Changes foreground color which affects PRINT output
in this case.

1.26 CONST

```
CONST true = -1&
```

This creates a named constant which can then be used
instead of a numeric literal, making for more easily
readable and maintainable programs.

```
Note that such named constants take on the data type
of the value on the right-hand-side of the "=" symbol.
```

1.27 COS

```
a = 25
PRINT "The cosine of 200 is" COS(200)
PRINT "The cosine of" a "is" COS(a)
END

The following is displayed:

The cosine of 200 is 0.48718513
The cosine of 25 is 0.99120286
```

1.28 CSNG

```
a% = 12
PRINT "It still looks the same: ";CSNG(a%)

This converts the short integer variable's contents (12)
into single-precision floating point format internally,
and displays the following:

It still looks the same: 12
```

1.29 CSRLIN

```
WINDOW 1,,(0,0)-(640,200)
PRINT
row = CSRLIN
LOCATE 12,1
PRINT "Hello World!"
LOCATE row,1
PRINT "Back again."
WHILE INKEY$="":SLEEP:WEND
WINDOW CLOSE 1
END

This program creates a new window, moves the print
position down one line, stores the current print row
changes the print position to row 12, column 1, displays
a string, then finally moves the print position back to
the location (row,1), displays another string and awaits
a keypress before closing the window.
```

1.30 DATA

```
dt$ = DATE$
      a = DAY
      FOR t = 1 to 7
              READ n,d$
              IF n = a then day$ = d$
      NEXT t
      PRINT "Today is ";day$;" (";dt$;")"
      END

      DATA 0,Sunday,1,Monday,2,Tuesday,3,Wednesday,4,Thursday
      DATA 5,Friday,6,Saturday
```

The weekday and date are displayed by this code.

1.31 DATE$

```
      PRINT "The date is " DATE$
      END

      This will display the date.
```

1.32 DAY

```
dt$ = DATE$
      a = DAY
      FOR t = 1 to 7
              READ n,d$
              IF n = a then day$ = d$
      NEXT t
      PRINT "Today is ";day$;" (";dt$;")"
      END

      DATA 0,Sunday,1,Monday,2,Tuesday,3,Wednesday,4,Thursday
      DATA 5,Friday,6,Saturday
```

The weekday and date are displayed by this code.

1.33 DECLARE

```
      DECLARE FUNCTION Delay(ticks&) LIBRARY
      DECLARE STRUCT DateStamp ds
      DECLARE SUB mySub(n)
      DECLARE FUNCTION Forbid() EXTERNAL

      These lines declare a shared library function, create
```

```
        a static structure variable, declare a forward
        reference to a subprogram and declare an external function.
```

1.34 DEFxxx

```
        DEFLNG a-z

        This directive tells the compiler to treat all
        variable/subprogram/function identifiers which start
  with a letter, as long integers until otherwise notified
  or unless overridden by an explicit variable declaration,
  a trailing qualifier character or another DEFxxx command.
```

1.35 DIM

```
        DIM myNumbers%(100)
        DIM LONGINT myOtherNumbers(50,10)
        DIM myNames$(50) SIZE 40

        These lines of code declare an array of 100 short
        integers, a 2D array (50 x 10 table) of long integers
  and an array of 50 strings, each 40 characters in length.
```

1.36 END

```
        PRINT "Hello World!"
        END

  In this case, END simply appears to mark the end of an ACE
  program, but whenever this command is encountered, program
  execution ceases.
```

1.37 ERR

```
        OPEN "I",#1,"myNonExistentFile"
        PRINT ERR
        END

        Running this code displays:

         205

        which is the AmigaDOS error code for "Object Not Found".
```

1.38 ERROR

```
ON ERROR goto abort_program
ERROR ON
OPEN "I",#1,"myNonExistentFile"
WHILE NOT EOF(1)
  PRINT INPUT$(1,1);
WEND
CLOSE #1
PRINT
STOP

abort_program:
  PRINT "*** Quitting with error code";ERR
END
```

Assuming that the file "myNonExistentFile" doesn't exist, this program will trap an error after the OPEN command causing the program to exit with a message indicating the error code. If the file did exist, it would be displayed a character at a time.

1.39 EQV

```
CONST true = -1&, false = 0&
PRINT true EQV true
PRINT true EQV false
PRINT false EQV true
PRINT false EQV false
END
```

The output of this program is:

```
-1
 0
 0
-1
```

1.40 EXIT FOR

```
FOR i=1 to 1000
  PRINT i,i^2
  IF MOUSE(0) THEN EXIT FOR
NEXT
END
```

This program prints values from 1 to 1000 and their squares. If the left mouse button is pressed, the FOR loop will be exited.

1.41 EXIT SUB

```
SUB choice(n)
  IF n<10 then choice=1:EXIT SUB
  IF n<20 then choice=2:EXIT SUB
  choice=3
END SUB
```

This subprogram tests for values of n of less than 10 or less than 20 and sets choice to 1 or 2 accordingly, then immediately exits from the subprogram, passing control back to the caller. If n is not less than 10 or 20, choice is set to 30 before the subprogram ends.

1.42 EXP

```
a = 2
PRINT EXP(a)
PRINT EXP(25)
END
```

The following will be displayed on the screen:

```
7.3890571
7.2004902E+10
```

1.43 EXTERNAL

```
  EXTERNAL FUNCTION Delay
  Delay(50&)
  END
```

or

```
  DECLARE FUNCTION Delay(ticks&) EXTERNAL
  Delay(50&)
  END
```

This code delays execution by 1 second (50 ticks) by calling the dos.library function Delay(). This has the same effect as declaring it as a shared library function except that the ami.lib C-style function is called and ACEbmaps:dos.bmap is not required.

The following code declares an external reference to a variable in ami.lib:

```
  EXTERNAL LONGINT RangeSeed
```

1.44 FILEBOX$

```
myFileName$ = FILEBOX$("Select a file")
END

This code displays a file requester (ASL or ARP) with
the title string "Select a file" and assigns the fully
qualified path of the selected file to the string variable
myFileName$.
```

1.45 FILES

```
FILES TO "ram:myFiles","df0:"
END

This creates a list of all files and directories
on df0: and stores them in a file called "myFiles"
in the RAM: disk.
```

1.46 FIX

```
PRINT 12.25
PRINT FIX(12.25)
END

This displays:

12.25
12
```

1.47 FONT

```
OPTION w+
WINDOW 1,"Font",(0,0)-(200,100),31
FONT "opal",9
LOCATE 1,1
WHILE -1
  PRINT " Hello World!"
WEND
END

This program opens a window, sets the text font to 9-point
opal and prints "Hello World!" until the close gadget is
clicked at which time the window is closed and the program
exits.
```

1.48 FOR..NEXT

```
FOR t = 0 to 20 STEP 2
        PRINT t;
NEXT t
END
```

This program will display the following on the screen:

```
 0  2  4  6  8  10  12  14  16  18  20
```

1.49 FRE

```
PRINT FRE(-1);
PRINT " Bytes of chip & fast memory are available."
END
```

1.50 GADGET MOD

```
WINDOW 1,"Slider",(0,0)-(100,145),10
GADGET 1,ON,50,(35,15)-(60,115),POTY
FOR n=1 to 50
  GADGET MOD 1,n
  SLEEP FOR .25
NEXT
GADGET 1,OFF
GADGET WAIT 0
GADGET CLOSE 1
WINDOW CLOSE 1
END
```

This program opens a window, renders a vertical slider gadget, moves the knob through all 50 levels, deactivates the slider, waits for any gadget to be pressed (window close gadget is the only one left) and cleans up before exiting.

1.51 GOSUB..RETURN

```
INPUT "Enter a number: ",a
INPUT "Enter another number: ",b
GOSUB Multiply
PRINT c
STOP

Multiply:
        c = a*b
        RETURN
END
```

1.52 GOTO

```
ON BREAK GOTO quit
BREAK ON

DEFLNG a-z
x=0
myLoop:
        PRINT x
        ++x
GOTO myLoop

quit:
END
```

This is an infinite loop which will PRINT positive integers starting from 0.

CTRL-C will break this loop however.

1.53 HANDLE

```
CONST NULL=0&
DECLARE FUNCTION xRead LIBRARY
OPEN "I",#1,"myFile"
myHandle& = HANDLE(1)
IF myHandle& <> NULL THEN
  buffer& = ALLOC(LOF(1)+1,5)  '..cleared public memory!
  IF buffer& <> NULL THEN CALL xRead(myHandle&,buffer&,LOF(1))
  PRINT CSTR(buffer&)
END IF
CLOSE #1
END
```

This program opens "myFile", obtains its AmigaDOS file handle, allocates a buffer for the file's contents, reads the whole file into memory, then displays it as a string. If the file is a not a text file, the display will be fairly weird.

1.54 HEADING

```
SETHEADING 10
TURNRIGHT 20
PRINT HEADING
END
```

This code will display:

```
 30
```

which indicates the current turtle heading.

1.55 HEX$

```
INPUT "Enter a number: ",a
PRINT "The Hex of"a"is " HEX$(a)
END
```

Displays the hexadecimal equivalent of the variable 'a'.

1.56 HOME

```
HOME

This command moves the turtle back to the top left
corner of the current window. If the turtle's pen
is down, a line will be drawn from the current pen
position to the home location.
```

1.57 IF

```
INPUT "Enter a word: "a$
IF a$ = "Hello" THEN
        PRINT "Hello to you!"
ELSE
        PRINT "I don't understand."
END IF
END
```

Demonstrates selection using a block IF..THEN..ELSE statement.

1.58 IMP

```
DEFLNG a-z
CONST true=-1
CONST false=0

SUB show.imp$(p,q)
  pq=p IMP q
  CASE
    pq     : show.imp$="T"
    NOT pq : show.imp$="F"
  END CASE
END SUB

CLS
PRINT "------------------"
PRINT "p    q   |  p -> q"
PRINT "------------------"
PRINT "T    T   |    ";show.imp$(true,true)
```

```
PRINT "T    F   |    ";show.imp$(true,false)
PRINT "F    T   |    ";show.imp$(false,true)
PRINT "F    F   |    ";show.imp$(false,false)
PRINT "------------------"

END
```

This program displays the following:

```
------------------
p    q   |  p -> q
------------------
T    T   |    T
T    F   |    F
F    T   |    T
F    F   |    T
------------------
```

1.59 INKEY$

```
WHILE UCASE$(INKEY$) <> "Q"
  SLEEP
WEND
```

This code loops until the "Q" key is pressed. The SLEEP command puts the program to sleep between system events (such as Intuition timer events, key presses and so on).

1.60 INPUTBOX

```
MyNumber& = INPUTBOX("OK for Default","Enter a Number","100",50,50)
PRINT MyNumber&
END
```

This program opens an input requester and returns a number which is then printed.

1.61 INPUTBOX$

```
MyWord$ = INPUTBOX$("OK for Default","Enter a Word","Hello",50,50)
PRINT MyWord$
END
```

This program opens an input requester and returns a string which is then printed.

1.62 INPUT

```
INPUT "What is your name";a$
PRINT "Hello "a$
END
```

1.63 INSTR

```
a$ = "The quick brown fox jumped over the lazy dog"
b = INSTR(a$,"z")
PRINT b
END

This will display 39 on the screen, since the letter 'z'
is the 39th character in the sentence.
```

1.64 INT

```
a = 25.5
PRINT a
PRINT INT(a)
END

The following is displayed on the screen:

25.5
25
```

1.65 LEFT$

```
a$ = "Hello"
PRINT LEFT$(a$,2)
END

The following is displayed on the screen:

He
```

1.66 LEN

```
a$ = "The quick brown fox jumped over the lazy dog"
PRINT LEN(a$)
END

This program will return a value of 44 since there are
44 characters in the line.
```

1.67 LET

```
LET a = 10
PRINT a
END
```

Assigns a value of 10 to the variable 'a' and then prints the contents of that variable.

1.68 LINE

```
LINE (100,25)-(250,75),3,b
LINE (100,25)-(250,75),1,b
LINE (50,50)-(200,100),3,bf
LINE (50,50)-(200,100),1,b
LINE (100,25)-(50,50),1,
LINE (250,25)-(200,50),1,
LINE (250,75)-(200,100),1,
END
```

This will produce a 3-D cube with a solid front, in the current output window.

1.69 LINE INPUT

```
OPEN "I",#1,"a_text_file"
WHILE NOT EOF(1)
  LINE INPUT #1,x$
  PRINT x$
WEND
CLOSE #1
END
```

This program displays a text file, one line at a time.

1.70 LOG

```
FOR i=1 to 5
  PRINT LOG(i)
NEXT
END
```

This program displays the natural logarithms of the numbers 1 to 5, ie:

```
 0
 0.69314721
 1.0986123
 1.3862943
 1.6094378
```

1.71 LONGINT

```
LONGINT n

SUB LONGINT max(LONGINT n, LONGINT m)
  IF n > m THEN max=n else max=m
END SUB

PRINT max(1,2)
END

This declares a long integer variable called n
as well as a subprogram with a long integer return
type and two long integer parameters. The program
displays:

 2
```

1.72 MESSAGE CLOSE

See the programs and ReadMe files in ACE:prgs/ACEports.

1.73 MESSAGE OPEN

See the programs and ReadMe files in ACE:prgs/ACEports.

1.74 MESSAGE READ

See the programs and ReadMe files in ACE:prgs/ACEports.

1.75 MESSAGE WAIT

See the programs and ReadMe files in ACE:prgs/ACEports.

1.76 MESSAGE WRITE

See the programs and ReadMe files in ACE:prgs/ACEports.

1.77 MID$

```
a$ = "The quick brown fox jumped over the lazy dog"
FOR t = 1 to LEN(a$)
        PRINT MID$(a$,t,1)
NEXT t
END
```

This will print out each letter in the sentence on a new line.

1.78 MOD

```
a = 22
b = 7
PRINT a/b
PRINT a MOD b
END
```

Displays the single-precision quotient and the integer remainder of the division of 22 by 7.

1.79 MOUSE

```
WINDOW 1,,(0,0)-(640,200)
WHILE NOT MOUSE(0):SLEEP:WEND
PENUP:SETXY MOUSE(1),MOUSE(2)
WHILE MOUSE(0)
  LINE STEP (MOUSE(1),MOUSE(2))
WEND
WHILE INKEY$="":SLEEP:WEND
WINDOW CLOSE 1
END
```

This program opens a window, waits for the left mouse button to be pressed and while pressed draws a line from the last to the current X,Y mouse coordinates.

After this, the program awaits a keypress. When it receives one, the window is closed and the program ends.

1.80 MSGBOX

```
f$ = FILEBOX$("Delete which file?")
okay = MSGBOX("Really Delete "+f$+"?","Ok","Cancel")
IF okay THEN KILL f$
END
```

This rather powerful three lines of code generates a file

```
requester, asks whether the selected file should be deleted
via a message requester and then deletes it if the user clicks
the "Ok" button.
```

1.81 NAME

```
NAME "a_file" AS "the_file"

This command renames a file called "a_file" to "the_file".
```

1.82 NOT

```
truth = 1 > 3
PRINT NOT truth
END

This code results in the following output:

-1

Because 1 is not greater than 3, truth holds the value
0 and NOT 0 = -1 (ie. true).
```

1.83 OCT$

```
a&=123456
PRINT OCT$(a&)
END

Displays the octal equivalent of the value of the
variable a&.
```

1.84 ON..GOTO/GOSUB

```
INPUT "Enter 1, 2 or 3 ",choice
ON choice GOTO one, two, three
PRINT "Out of range!"
STOP

one:
  PRINT 1 : STOP
two:
  PRINT 2 : STOP
three:
  PRINT 3
END
```

```
This code asks for the entry of a value (1, 2 or 3)
and jumps to a label according the number entered
or prints an error message.
```

1.85 OPEN

```
a$ = "Hello There!"
b = 256
c$ = "Good-bye."
OPEN "O",#1,"My_File"
PRINT #1,a$,b,c$
WRITE #1,a$,b,c$
CLOSE 1
END

The following will be written to My_File:

Hello There!             256            Good-bye.
"Hello There!",256,"Good-bye."

Notice the different formats for PRINT # and WRITE #.
```

1.86 OPTION

```
OPTION l+
a$ = "Hello There!"
b = 256
c$ = "Goodbye."
OPEN "O",#1,"My_File"
PRINT #1,a$,b,c$
OPTION l-
WRITE #1,a$,b,c$
CLOSE 1
END

This causes source code to be listed during
compilation between the two OPTION directives.
```

1.87 OR

```
CONST true = -1&, false = 0&
PRINT true OR true
PRINT true OR false
PRINT false OR true
PRINT false OR false
END

The output of this program is:

-1
```

```
-1
-1
 0
```

1.88 PAINT

```
CIRCLE (320,100),30
PAINT (320,100)

This code paints a circle using the current
foreground color.
```

1.89 PALETTE

```
SCREEN 1,640,200,3,2
WINDOW 1,,(0,0)-(640,200),32,1
PALETTE 0,0,0,0 '..black (bgnd)
PALETTE 1,1,1,1 '..white (fgnd)
PALETTE 2,1,0,0 '..red
PALETTE 3,0,1,0 '..green
PALETTE 4,0,0,1 '..blue
FOR i=1 TO 4
  COLOR i
  PRINT "Hello World"
NEXT
WHILE INKEY$="":SLEEP:WEND
WINDOW CLOSE 1
SCREEN CLOSE 1
END

This program opens a hi-resolution screen, then opens
a window, redefines the first 5 colors in the palette
for the screen, displays "Hello World" in 4 colors
before awaiting a key press and closing the window
and screen.
```

1.90 PENDOWN

```
DEFINT a-z
SCREEN 1,640,200,3,2
WINDOW 1,,(0,0)-(640,200),32,1
RANDOMIZE TIMER
LOCATE 23,31
PRINT "Press 'Q' to quit.";
PENUP
SETXY 320,100
WHILE UCASE$(INKEY$)<>"Q"
    c = INT(RND(0)*3)+1
    COLOR c
    PENDOWN
```

```
    FOR t = 0 TO 359 STEP 15
        SETHEADING t
        FOR r = 1 to 4
           FORWARD 30
           TURNRIGHT 90
        NEXT r
    NEXT t
WEND
WINDOW CLOSE 1
SCREEN CLOSE 1
END
```

This will display a design with changing colors on the screen.

1.91 POINT

```
DEFINT a-z
CLS

IF ARGCOUNT<>1 THEN
  PRINT "usage: ";arg$(0);" string"
  STOP
END IF

PRINT ARG$(1)

FOR j=0 TO 7
  LOCATE 2+j,1
  FOR k=0 TO (LEN(ARG$(1))*8)-1
    IF POINT(k+5,j+11) = 0 THEN
      PRINT " ";
    ELSE
      PRINT "@";
    END IF
  NEXT
  PRINT
NEXT
```

This program prints a string in large format
by reading each pixel of each letter.

1.92 POKEx

```
SUB usage
        PRINT "usage: ";ARG$(0);" on | off"
        STOP
END SUB
IF ARGCOUNT<>1 THEN CALL usage
CMD$ = UCASE$(ARG$(1))
CONST reg = &Hbfe001
IF CMD$ = "ON" THEN
        POKE reg,peek(reg) AND 253
```

```
ELSE
        IF CMD$ = "OFF" THEN
                POKE reg,PEEK(reg) OR 2
        ELSE
                usage
        END IF
END IF
```

This code will turn the power LED on or off.

1.93 POS

```
WINDOW 1,,(0,0)-(640,200)
CLS
PRINT "                 ";
row = CSRLIN
column = POS
LOCATE 12,1
PRINT "Hello World!"
LOCATE row,column
PRINT "Back again."
WHILE INKEY$="":SLEEP:WEND
WINDOW CLOSE 1
END
```

This program clears the current window, prints some spaces, stores the current print position, changes the print position to row 12, column 1, displays a string, then finally moves the print position back to the location (row,column) and displays another string.

1.94 POTX

```
PRINT POTX(2)
END
```

Returns the raw resistance value of whatever is currently connected to the POTX line of the game port.

1.95 POTY

```
PRINT POTY(2)
END
```

Returns the raw resistance value of whatever is currently connected to the POTY line of the game port.

1.96 PRINT

```
PRINT "This is an example of the PRINT command."
PRINT "This", "is", "with", "commas."
PRINT "This"; "is"; "with"; "semicolons."
PRINT
PRINT "^^^ This is a null line ^^^"
PRINT "This is the end."
END
```

Shows typical usage of the PRINT statement.

1.97 PRINTS

Now redundant since version 2.0 of ACE but it is a faster way of printing to a user-defined screen or window. The usage is the same as for PRINT.

See
PRINT

1.98 PSET

```
WINDOW 1,,(0,0)-(640,200)
FOR i=1 to 500
  PSET (RND*640,RND*200)
NEXT
WHILE INKEY$="":SLEEP:WEND
WINDOW CLOSE 1
END
```

This program plots 500 randomly positioned pixels in the current output window.

1.99 REM

```
REM This is a single-line comment
' So is this
END
```

1.100 REPEAT..UNTIL

```
REPEAT
  SLEEP
UNTIL INKEY$=CHR$(27)
END
```

This code loops until the Escape key is pressed.

1.101 RESTORE

```
REPEAT
  RESTORE
  FOR i=1 to 3
    READ colr$
    PRINT colr$
  NEXT
UNTIL INKEY$=CHR$(27)

DATA red,green,blue
END
```

This repeatedly displays the 3 strings "red","green","blue"
by restoring the data pointer to the start of the DATA for
the program.

1.102 RIGHT$

```
a$ = "Hello"
PRINT RIGHT$(a$,2)
END
```

The following is displayed on the screen:

```
lo
```

1.103 RND

```
RANDOMIZE TIMER

FOR t = 1 to 10
        dice1 = INT(RND*6)+1
        dice2 = INT(RND*6)+1
        PRINT dice1, dice2
NEXT t
END
```

This program simulates 2 dice.

It first seeds ACE's random number generator with the
number of seconds past midnight. Doing so ensures
that the random sequence generated by RND differs
each time a program is run.

1.104 SADD

```
a$ = "Hello"
FOR t = 0 to 5
```

```
            PRINT CHR$(PEEK(SADD(a$)+t));
      NEXT t
      PRINT
      END

      Prints the word "Hello" by peeking the values stored
at successive addresses in a string.
```

1.105 SAY

```
      SAY TRANSLATE$("Amiga computers love ACE!")
      a$ = "AHMIY3GAH KUMPYUW3TERZ LAH4V EY4S."
      SAY a$
      END

Demonstrates the SAY command using translated text
and phonemes.
```

1.106 SCREEN FORWARD

```
LIBRARY "intuition.library"

DECLARE FUNCTION ActivateWindow(ADDRESS wdw) LIBRARY

SCREEN 1,640,200,2,2
WINDOW 1,"First",(0,10)-(640,100),6,1
MENU 1,0,1,"Project"
MENU 1,1,1,"Show Second","S"
MENU 1,2,1,"Make Rearmost","M"
MENU 1,3,1,"Quit","Q"

SCREEN 2,640,200,2,2
WINDOW 2,"Second",(0,110)-(640,200),6,2
MENU 1,0,1,"Project"
MENU 1,1,1,"Show First","S"
MENU 1,2,1,"Make Rearmost","M"
MENU 1,3,1,"Quit","Q"

{*
** Use either menu event trapping or menu waiting
** code (ie. comment one block out and use the other).
*}

'** Event trapping code begins

ON MENU GOSUB menu_handler
MENU ON

WHILE -1
    SLEEP
WEND
```

```
'** Event trapping code ends


'** Event waiting code begins

'WHILE -1
  '  MENU WAIT
'  GOSUB menu_handler
'WEND

'** Event waiting code ends

END


menu_handler:
    the_menu = MENU(0)
    the_item = MENU(1)

    IF the_menu = 1 THEN
    IF the_item = 1 THEN
  IF WINDOW(1) = 1 THEN id = 2 ELSE id = 1
  SCREEN FORWARD id
  WINDOW OUTPUT id
  ActivateWindow(WINDOW(7))
        END IF

    IF the_item = 2 THEN
  id = WINDOW(1)
  SCREEN BACK id
  SLEEP FOR 1
  SCREEN FORWARD id
    END IF

        IF the_item = 3 THEN quit
    END IF
RETURN

quit:
  MENU CLEAR
    WINDOW CLOSE 1
    SCREEN CLOSE 1
  MENU CLEAR
    WINDOW CLOSE 2
    SCREEN CLOSE 2
STOP

This program opens two screens with one window each.

Each window has a Project menu with three items. The
first item allows whichever of the two screens is
rearmost to be brought to the front. The second item
is used to momentarily make the current screen rearmost.
The third item closes both screens and windows and ends
the program.

This program also demonstrates menu event trapping vs menu
```

```
waiting.
```

1.107 SCROLL

```
SCREEN 1,320,200,2,1
PALETTE 0,0,0,0         '..black
PALETTE 2,1,.2,.2       '..red

'..draw 4 rows of 8 boxes
FOR row=0 TO 3
  FOR column=0 TO 7
    LINE (column*30,row*20)-(column*30+20,row*20+12),2,BF
  NEXT
NEXT

inc=-1 : column=-1
FOR row=0 TO 11
  column = column-SGN(column)
  inc = -SGN(inc)

  '..move boxes vertically
  SCROLL (column,row*10)-(column+230,row*10+82),0,10
  FOR column=0-80*(inc=-1) TO 80+80*(inc=-1) STEP inc

    '..move them horizontally
    SCROLL (column,row*10)-(column+230,row*10+82),inc,0
  NEXT
  BEEP
NEXT
WHILE INKEY$="":WEND
SCREEN CLOSE 1
```

1.108 SGN

```
IF ARGCOUNT<>1 THEN STOP
n = VAL(ARG$(1))
PRINT SGN(n)
END

This code displays the sign of the first command
line argument, treated as a number.
```

1.109 SHARED

```
SINGLE n

SUB test
SHARED n
  PRINT n
END SUB
```

```
IF ARGCOUNT<>1 THEN STOP
n = VAL(ARG$(1))
test
END
```

A single-precision floating point variable is declared
and then shared with the subprogram "test".

1.110 SHL

```
a = 128
PRINT SHL (a,4)
END
```

This is the equivalent of multiplying 128 by 2 four times, or
multiplying 128 by 16.

1.111 SHR

```
a = 128
PRINT SHR (a,4)
END
```

This is the equivalent of dividing 128 by 2 four times, or
dividing 128 by 16.

1.112 SHORTINT

```
SHORTINT n

SUB SHORTINT max(SHORTINT n, SHORTINT m)
  IF n > m THEN max=n else max=m
END SUB

PRINT max(1,2)
END
```

This declares a short integer variable called n
as well as a subprogram with a short integer return
type and two short integer parameters. The program
displays:

```
 2
```

1.113 SINGLE

```
SINGLE n

SUB SINGLE max(SINGLE n, SINGLE m)
  IF n > m THEN max=n else max=m
END SUB

PRINT max(1,2)
END
```

This declares a single-precision variable called n
as well as a subprogram with a single-precision return
type and two single-precision parameters. The program
displays:

```
 2
```

1.114 SIZEOF

```
n%=5
STRUCT theStruct
  LONGINT a
  STRING b SIZE 100
END STRUCT
DECLARE STRUCT theStruct myStruct
PRINT SIZEOF(LONGINT)
PRINT SIZEOF(n%)
PRINT SIZEOF(myStruct)
END
```

This code produces the following output:

```
 4
 2
 104
```

1.115 SIN

```
FOR i=1 to 5
  PRINT SIN(i)
NEXT
END
```

This program displays the sines of the numbers 1 to 5,
ie:

```
 0.84147111
 0.9092975
 0.14112005
-0.75680259
-0.95892433
```

1.116 SLEEP FOR

```
PRINT "Goodnight, I am going to sleep now."
SLEEP FOR 10
PRINT "Hello, I'm awake now!"
END
```

This program displays a message, sleeps for 10 seconds, then displays another message.

1.117 SPACE$

```
PRINT "Hello World"
PRINT "Hello ";
PRINT SPACE$(10);
PRINT "World"
END
```

The following is displayed:

```
Hello World
Hello          World
```

1.118 SPC

```
FOR t = 1 TO 10
        PRINT SPC(t) "\"
NEXT t
END
```

Draws a diagonal line using the backslash character.

1.119 SQR

```
FOR i=1 to 5
  PRINT SQR(i)
NEXT
END
```

This program displays the square roots of the numbers 1 to 5, ie:

```
 1
 1.4142135
 1.7320507
 2
 2.236068
```

1.120 STICK

```
WHILE INKEY$=""
   IF STICK(2) OR STICK(3)<>0 THEN
        PRINT "x direction is" STICK(2)
        PRINT "y direction is" STICK(3)
   END IF
   IF STRIG(3) =-1 THEN PRINT "FIRE!!!"
WEND
END
```

1.121 STR$

```
PRINT STR$(65)
END

Prints the number 65 as a string.
```

1.122 STRING

```
STRING myStringVar
STRING myShortStringVar SIZE 10

These two lines of code create a 1K string variable
and a 10 byte string variable.
```

1.123 STRING$

```
PRINT STRING$(5,"A")
PRINT STRING$(5,65)
END

Prints the following:

AAAAA
AAAAA
```

1.124 STRUCT

```
STRUCT date_stamp
  LONGINT ds_Days
  LONGINT ds_Minutes
  LONGINT ds_Ticks
END STRUCT

DECLARE STRUCT date_stamp stamp
DECLARE FUNCTION DateStamp LIBRARY
```

```
      DateStamp(stamp)
      PRINT stamp->ds_Days
      PRINT stamp->ds_Minutes
      PRINT stamp->ds_Ticks
      END

      This code defines a datestamp structure, declares
      a variable of the new data type, calls the dos.library
      DateStamp() function filling the structure's fields,
      which are then all displayed.
```

1.125 STYLE

```
PRINT "Hello ";
STYLE 6
PRINT "World!"
STYLE 0
END

This code will print "Hello World!" with "World!" bold and
italicised. The text style will then be reset to plain.
```

1.126 SWAP

```
      a$="1"
      b$="2"
      FOR t = 1 to 5
              SWAP a$,b$
              PRINT a$,b$
      NEXT t
      END

      The following is displayed:

      1       2
      2       1
      1       2
      2       1
      1       2
```

1.127 SYSTEM

```
      SYSTEM "list > ram:myfiles"
      PRINT SYSTEM
SYSTEM 5
      END

This code invokes the AmigaDOS "list" command, redirecting
the output to a file in the RAM: disk, displays the version
```

```
number of the operating system (eg. 37, 39) and sets the
return value for the program to 5 (WARN).
```

1.128 TAB

```
PRINT "Hello World!"
PRINT TAB(10) "Hello World!"
END

The output is:

Hello World!
         Hello World!
```

1.129 TAN

```
FOR i=1 to 5
  PRINT TAN(i)
NEXT
END

This program displays the tangents of the numbers 1 to 5,
ie:

 1.5574079
-2.18504
-0.14254658
 1.1578214
-3.3805146
```

1.130 TIME$

```
PRINT "The time is " TIME$
END

This will display the system time in hours,
minutes and seconds.
```

1.131 VAL

```
a$="12345"
PRINT VAL(a$)
END

This displays:

 12345
```

1.132 VARPTR

```
a$ = "Hello"
FOR t = 0 to 5
        PRINT CHR$(PEEK(VARPTR(a$)+t));
NEXT t
PRINT
END
```

Prints the word "Hello" by peeking the values stored at successive addresses in a string.

1.133 WINDOW ON

```
CONST havingfun = -1&

WINDOW 1,,(0,0)-(640,200),8

ON WINDOW GOTO quit
WINDOW ON

WHILE havingfun
  FOR i=1 TO 640
    COLOR RND*3+1
    PRINT PTAB(i);"Hello World!"
  NEXT
  CLS
WEND

quit:
WINDOW CLOSE 1
END
```

This program opens a window and displays multi-coloured "Hello World" strings until the window close gadget is clicked.

1.134 WINDOW OUTPUT

```
WINDOW 1,"Window A",(0,0)-(100,50)
WINDOW 2,"Window B",(500,100)-(620,200)
WINDOW OUTPUT 1
PRINT "I am now the current output window."
WHILE INKEY$="":SLEEP:WEND
WINDOW CLOSE 1
WINDOW CLOSE 2
END
```

This program opens two windows, changes the current output window to be the first, prints a message in that window and awaits a keypress before closing the windows and ending.

1.135 XCOR

```
SCREEN 1,640,200,3,2
WINDOW 1,,(0,0)-(640,200),32,1
RANDOMIZE TIMER
PENUP
SETXY 320,100
WHILE INKEY$=""
  c = INT(RND(0)*3)+1
  COLOR c
  PENDOWN
  FOR t = 1 to 4
    FORWARD 10
    TURNRIGHT 90
  NEXT t
  PENUP
  a=XCOR : b=YCOR
  LOCATE 1,1
  COLOR 1
  PRINT "My position is: "a,b
  x = INT(RND(0)*630)+1
  y = INT(RND(0)*190)+1
  SETXY x,y
  SLEEP FOR .4
WEND
WINDOW CLOSE 1
SCREEN CLOSE 1
END
```

1.136 XOR

```
CONST true = -1&, false = 0&
PRINT true XOR true
PRINT true XOR false
PRINT false XOR true
PRINT false XOR false
END

The output of this program is:

 0
-1
-1
 0
```