PL65#

Background#

PL65, like the better-known Action, is a high-level ALGOL-derived language intended to produce high-performance code in a more readable format than assembler. The two languages share much in common in overall terms and even syntax and structure. The main difference is that PL65 includes a STRING type, something Action! lacked, but like Action!, PL65 also lacks a FLOAT type.

One unique feature of PL65 is that it includes two full-screen editors. The main editor is KED, and includes modern features like indentation support, search and replace, and block copy/paste. A second editor, the "micro editor", is included in the compiler program, allowing you to make minor changes without having to switch programs.

PL65 allows libraries to be referenced in a program using the INCLUDE directive, in a fashion largely identical to C's #include. However, it also includes the LINK command, normally placed at the end of a file, which allows several files to be stitched together into one larger file. The difference is that INCLUDE begins compiling the referenced file when it is seen in the code, whereas LINK occurs only when the current file is completely compiled, even if the LINK is placed higher in the code.

Here's a working disk image of PL65 for anyone who wants to try what must be the rarest of Atari 8-bit languages. To use it, boot with SpartaDOS (an X cart image in one of the emulators is a good development environment) then run PL65.COM and choose from the various menu options. SAMPLE.PRG is a useful example program to figure out some of the finer points of the language (there's no manual on the disk) and S.OBJ is the compiled source. Little or no third-party documentation exists and I don't have a manual. I have one in-depth magazine article (in 8:16, BaPAUG's magazine, written by Simon Trew). The package consists of a compiler, editor, library and a sample game.

ATR Image#

- pl65.atr ; Thank you Fujix from AtariAge for giving us this very rare language! :-)))
- pl65-fixed.atr ; Thank you ddez from AtariAge for fixing the existing image! :-)
- PL65_Compiler_SpartaDOS_X_33a_360.atr ; Thank you 576XE from AtariAge for providing this version! :-)

Source Code#

- pl65src.zip ; Thank you DanBoris from AtariAge for giving us this partial commented disassembly of the PL65 sample program along with some plain ascii versions of the source file and libraries. We really appreciate your help! :-)))

Manual#

- PL65-Manual.pdf ; size: 6.8 MB ; Thank you so much spookt from AtariAge for your help in this case and MrFish for the post processing. :-)

Reference#

- PL65 at AtariAge
- PL65 'cracked' at AtariAge
- Complete manual of PL65 at AtariAge

- PL65 Problem - WRTSTR/WRTLN at AtariAge
- Software ad at AtariAge

Picture#

PL65 manual and disk

PL65 disk sticker

PL65 startscreen

PL65 manual cover

PL65 manual example

Movies#

- Hello World in Noahsoft's PL65 for the Atari 400/800
- Sample Toolchain for Atari PL65

Sample Code#

Thank you 576XE from AtariAge!

```
!====================================!
! KEYS.LIB
! PL65 Keys operations.
!------------------------------------!
CONST UPA=142,DNA=143,LTA=134,RTA=135
CONST ESC=28,SPC=33,TAB=44,RET=12
CONST none=255
BYTE CH=764
!------------------------------------!
PROC AnyKey()
BEGIN
  WHILE CH=none DO ENDWHILE CH=none
END
!------------------------------------!
FUNC InKey()
  BYTE k
BEGIN
  WHILE CH=none DO ENDWHILE k=CH CH=none
END k
!------------------------------------!
! Only Declarations for clear compiling!
! Definitions must be in MAIN() as BODY.
PROC EscF() FORWARD
PROC UpaF() FORWARD
PROC DnaF() FORWARD
PROC LtaF() FORWARD
PROC RtaF() FORWARD
PROC SpcF() FORWARD
PROC TabF() FORWARD
PROC RetF() FORWARD
PROC DefF() FORWARD
!------------------------------------!
```

```
PROC ParsKey(BYTE Key)
BEGIN
  CASE Key
    OF ESC DO EscF() ENDOF
    OF UPA DO UpaF() ENDOF
    OF DNA DO DnaF() ENDOF
    OF LTA DO LtaF() ENDOF
    OF RTA DO RtaF() ENDOF
    OF SPC DO SpcF() ENDOF
    OF TAB DO TabF() ENDOF
    OF RET DO RetF() ENDOF
  ELSE
    DefF()
  ENDCASE
END
!-----------------------------------!
ENDFILE

! Keyboard Parser
INCLUDE TERMINAL.LIB
INCLUDE KEYS.LIB

PROC POS(INT COL=$55 BYTE ROW=$54)
BEGIN END ! Gets data from stack
!-----------------------------------!
PROC CLR() BEGIN PUT(0,125) END
!-----------------------------------!
BODY EscF BEGIN POS(18,10) WRTSTR("ESC") END
BODY UpaF BEGIN POS(18,10) WRTSTR("UPA") END
BODY DnaF BEGIN POS(18,10) WRTSTR("DNA") END
BODY LtaF BEGIN POS(18,10) WRTSTR("LTA") END
BODY RtaF BEGIN POS(18,10) WRTSTR("RTA") END
BODY SpcF BEGIN POS(18,10) WRTSTR("SPC") END
BODY TabF BEGIN POS(18,10) WRTSTR("TAB") END
BODY RetF BEGIN POS(18,10) WRTSTR("RET") END
BODY DefF BEGIN POS(18,10) WRTSTR("BAD") END
!-----------------------------------!
MAIN()
  BYTE k
BEGIN
  CLR()
  WRTLN("Keys Parsing Sample")
  WRTLN("Waiting a KEY...") CR()

  REPEAT
    k=InKey() ParsKey(k)
  UNTIL k=ESC
END

!===================================!
! ALLOC.LIB Memory allocation library
!       for PL65 compiler           !
! First-fit algorythm for malloc/free
!-----------------------------------!
! by Evgeny Zolotarev (aka 576XE),2017
!===================================!
CONST NULL=0,FRED=1,USED=0
CONST MEMSTART=$4000,MEMEND=$BC00
CONST MEMSIZ=MEMEND-MEMSTART
CONST METALEN=6,BYTSZ=1,INTSZ=2
```

```
!-----------------------------------!
POINTER gpP INT gpV BASED gpP
POINTER freList
!-----------------------------------!
FUNC getFlag(INT adr)
BEGIN gpP=adr END gpV
!-----------------------------------!
FUNC getSize(INT adr)
BEGIN gpP=adr+2 END gpV
!-----------------------------------!
FUNC getNext(INT adr)
BEGIN gpP=adr+4 END gpV
!-----------------------------------!
PROC setFlag(INT adr,flag)
BEGIN gpP=adr gpV=flag END
!-----------------------------------!
PROC setSize(INT adr,size)
BEGIN gpP=adr+2 gpV=size END
!-----------------------------------!
PROC setNext(INT adr,next)
BEGIN gpP=adr+4 gpV=next END
!-----------------------------------!
PROC memInit()
BEGIN
  setFlag(freList,FRED)
  setSize(freList,MEMSIZ-METALEN)
  setNext(freList,NULL)
END
!-----------------------------------!
PROC split(POINTER fits INT size)
  POINTER new INT blkSiz
BEGIN
  blkSiz=METALEN+size
  new=fits+blkSiz
  setFlag(new,FRED)
  setSize(new,getSize(fits)-blkSiz)
  setNext(new,getNext(fits))
  setFlag(fits,USED)
  setSize(fits,size)
  setNext(fits,new)
END
!-----------------------------------!
FUNC alloc(INT nBytes)
  POINTER curr,result
BEGIN
  IF (getSize(freList)=0) THEN
    memInit()
    WRTLN("Memory initialized") CR()
  ENDIF

  curr=freList
  WHILE ((getSize(curr)<nBytes OR getFlag(curr)=USED) AND getNext(curr)<>NULL) DO
    curr=getNext(curr)
    WRTLN("- Search for fitting block...")
  ENDWHILE

  IF getSize(curr)=nBytes THEN
    setFlag(curr,USED)
    result=curr+METALEN
```

```
    WRTLN("+ Exact fitting block allocated")
    RETURN result
  ENDIF

  IF getSize(curr)>nBytes+METALEN THEN
    split(curr,nBytes)
    result=curr+METALEN
    BASE=16
    WRTSTR("Block $") WRITE(result) WRTSTR("-$") WRITE(result+nBytes)
    WRTSTR(" size=") BASE=10 WRITE(getSize(curr)) WRTLN(" allocated")
    RETURN result
  ELSE
    result=NULL
    WRTLN("- No sufficient memory to allocate")
  ENDIF
END result
!-------------------------------------!
PROC merge()
  POINTER curr
  INT size,next
BEGIN
  curr=freList
  WHILE getNext(curr)<>NULL DO
    IF (getFlag(curr) AND getFlag(getNext(curr))) THEN
      size=getSize(curr)+getSize(getNext(curr))+METALEN
      setSize(curr,size)
      next=getNext(getNext(curr))
      setNext(curr,next)
    ENDIF
    curr=getNext(curr)
  ENDWHILE
END
!-------------------------------------!
PROC free(POINTER ptr)
  POINTER curr
BEGIN
  IF ptr>=MEMSTART AND ptr<=MEMEND THEN
    curr=ptr
    curr=curr-METALEN
    setFlag(curr,FRED)
    merge()
    WRTLN("+ Block freed, space merged")
  ELSE
    WRTLN("Please provide a valid allocated pointer")
  ENDIF
END
!=====================================!
! End of Library
ENDFILE

INCLUDE TERMINAL.LIB
INCLUDE ALLOC.LIB

MAIN()
  POINTER p,r,k
  POINTER q,w
BEGIN
  freList=MEMSTART

  p=alloc(100*INTSZ)
```

```
  q=alloc(250*BYTSZ)
  r=alloc(1000*INTSZ)

  free(p)
  w=alloc(700)

  free(r)
  k=alloc(500*INTSZ)

  CR() WRTLN("Allocation and deallocation")
  WRTLN("are done successfully!")
END
```