Some New Libraries for Power C

Glenn Holmer (“ShadowM”)
C=4 Expo 2007

The Power C Compiler

- published by Pro–Line, Spinnaker
- K & R (pre–ANSI) C, old–style function declarations
- good standard libraries, library utility
- excellent editor with multiple buffer support
- not difficult to find

K & R function declarations

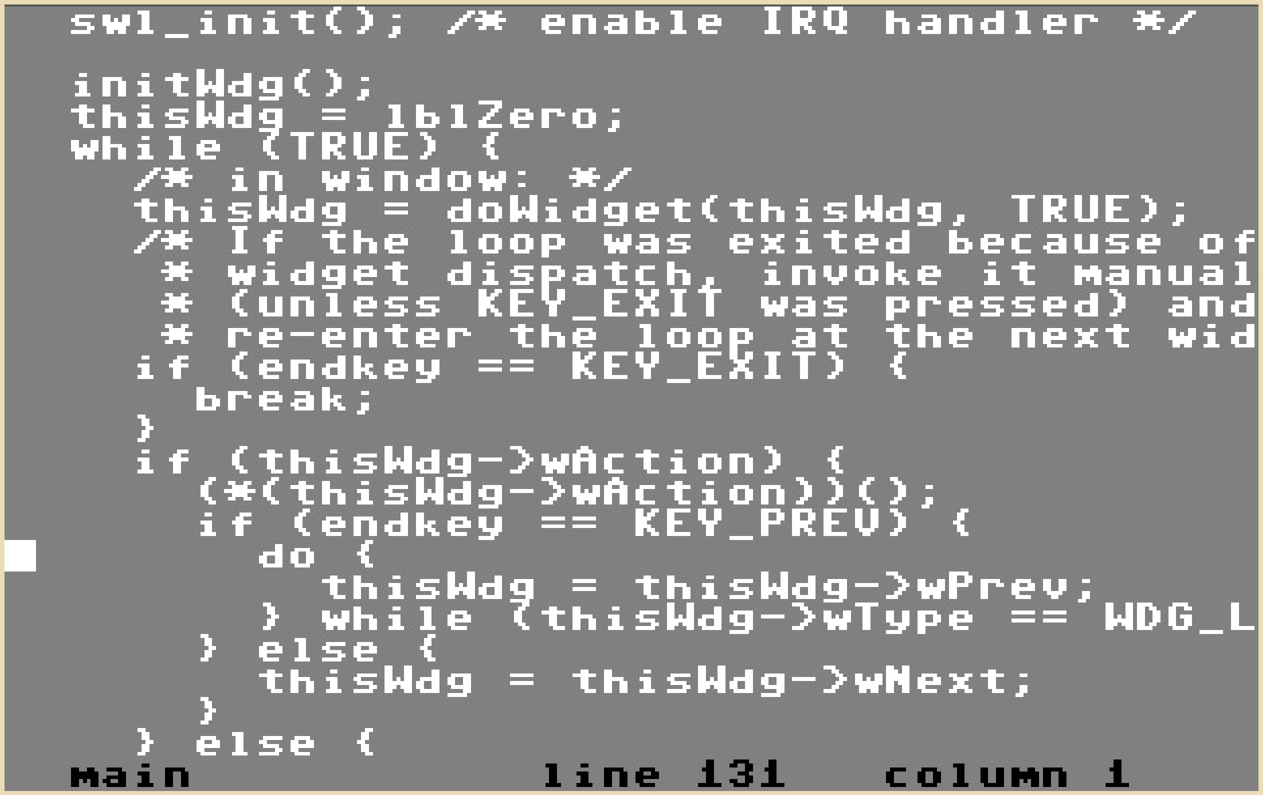
modern function declaration:

```
int foo(int bar) {
  // clever code here
}
```

K & R (old-style) function declaration:

```
int foo(bar)
int bar; {
  // clever code here
}
```

The Power C Editor



```
  swl_init(); /* enable IRQ handler */

  initWdg();
  thisWdg = lblZero;
  while (TRUE) {
    /* in window: */
    thisWdg = doWidget(thisWdg, TRUE);
    /* If the loop was exited because of
     * widget dispatch, invoke it manual
     * (unless KEY_EXIT was pressed) and
     * re-enter the loop at the next wid
    if (endkey == KEY_EXIT) {
      break;
    }
    if (thisWdg->wAction) {
      (*(thisWdg->wAction))();
      if (endkey == KEY_PREV) {
        do {
          thisWdg = thisWdg->wPrev;
        } while (thisWdg->wType == WDG_L
      } else {
        thisWdg = thisWdg->wNext;
      }
    } else {
  main              line 131   column 1
```

Power C Idiosyncrasies

- programs not reloaded when run again
- on rare occasions, compiler locks up
- **`trim`** utility is buggy, don't use
- works well with SuperCPU
- works well with CMD drives, except that include files can't be pulled in from other directories

C–ASSM assembler

- public–domain assembler, written in C to be compiled with Power C
- a bit slow, but worth it!
- produces Power C object files that can be linked with your programs
- supports include files
- source available, also a reverse assembler for Power C object files

Existing Power C Libraries

- enhanced shell
- scripting utilities
- graphics libraries
- bitmapped windowing library
- these were available on Q-Link and the Pro-Line BBS

Some New Libraries

- enhanced text input with error bell
- character-based windowing
- menus and submenus
- widgets (label, text, checkbox, listbox)
- Q-Link style function key definitions
- disk, partition, and file enumeration
- relative file support

Using the Libraries

- “SWL” libraries (menus, widgets):
  - **`#include <swl.h>`**
  - have **`swl*.o`** (six files) on your work disk
  - link with **`swl.l`**
- disk libraries (drive lists, relative files):
  - **`#include <disk.h>`**
  - link with **`drv-query.o`**, **`relfiles.o`**

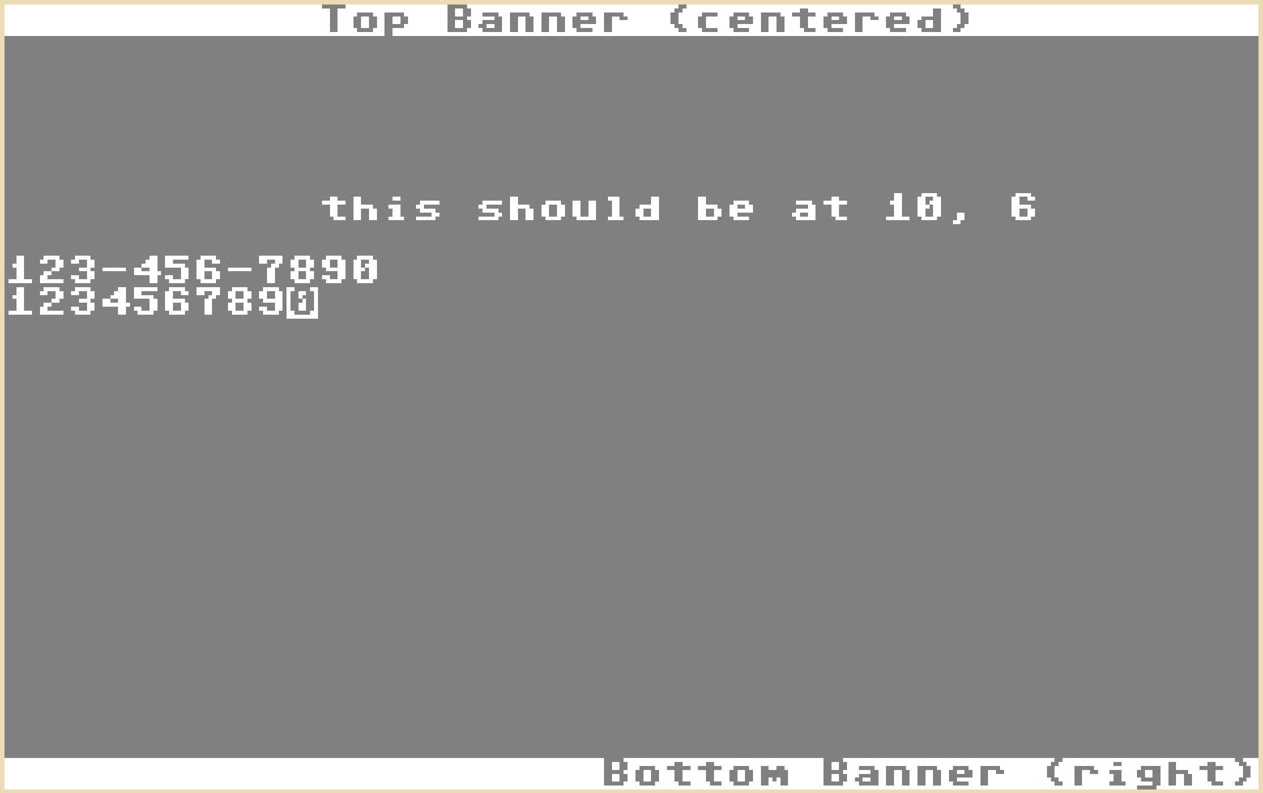
Enhanced Text Input

- cursor location
- single-line text input
- basic cursor controls
- restricted length and content
- customizable “end keys”
- minimal support for control characters

Enhanced Text Example

```
#include <swl.h>
extern char allowed;

char str[13];
strcpy(str, "123045607890");
addEnder(KEY_SEL);    /* make F1 work like CR */
strcpy(&allowed, “-”);  /* allow entry of '-' */

swl_init();   /* initialize interrupt handler */
/* numeric only, but allow certain keys,
   and check the “endkeys” table to exit */
wchrin(str, MSK_NUM | MSK_ALW | MSK_END);
```

Enhanced Text Demo



```
                    Top Banner (centered)




                    this should be at 10, 6

123-456-7890
1234567890
                                        Bottom Banner (right)
```

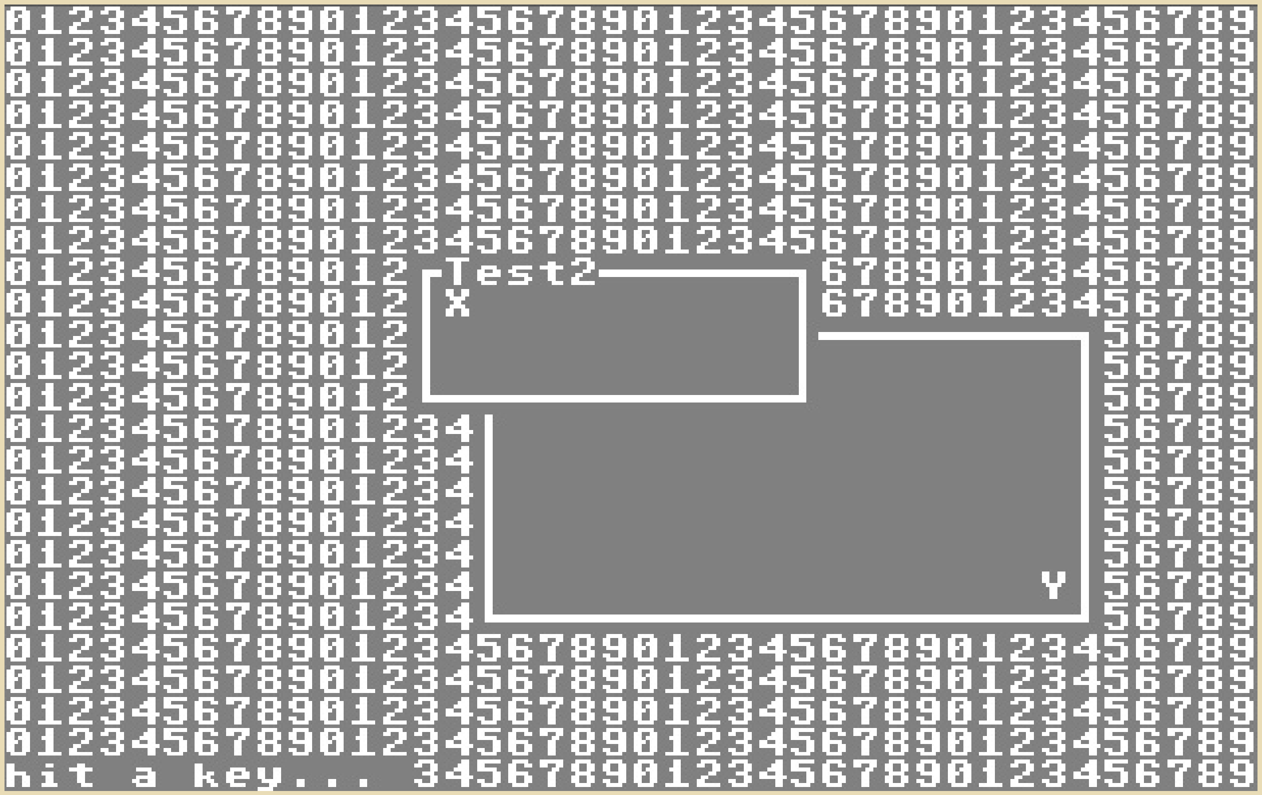
Character-Based Windowing

- windows are drawn directly to screen memory, which is buffered and restored
- support for multiple overlapping windows (8 maximum)
- special routines for cursor location within a window (only the last opened window can be printed into)

Windowing Example

```
openWnd(15, 10, 20, 10, "Test");
wlocate(0, 0);
putchar('X');
wlocate(17, 7);
putchar('Y');

openWnd(13, 8, 13, 5, "Test2");
wlocate(0, 0);
putchar('X');
if (wlocate(11, 3)) { /* should fail */
  putchar('Y');
}
clsWnd(); /* second window */
clsWnd(); /* first window  */
```

Windowing Demo



```
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012 Test2       67890123456789
0123456789012 X           67890123456789
0123456789012                      56789
0123456789012                      56789
0123456789012                      56789
012345678901234                    56789
012345678901234                    56789
012345678901234                    56789
012345678901234                    56789
012345678901234                    56789
012345678901234                  Y 56789
012345678901234                    56789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
0123456789012345678901234567890123456789
hit a key... 345678901234567890123456789
```

Menus and Submenus

- menus are a singly-linked list of structures; submenus called recursively
- menu mode opens a window, list mode doesn't buffer and ignores submenus (meant for widget)
- menu items are scrollable
- “constructor” and item add functions
- can attach a dispatch function to an item

Menu Structure

```
struct menu {
  int mLeft;    /* left edge (in columns)  */
  int mTop;     /* top edge (in rows)      */
  int mVisible; /* maximum visible items   */
  int mCuritem; /* current selected item   */
  int mTopitem; /* (internal use)          */
  struct menu *mParent;
  char *mTitle;
  int *mExtend; /* for "subclassing"            */
  int *mFirst;  /* pointer to first menu item   */
  int mChosen;  /* true if item has been chosen */
  int mEndkey;  /* key used to bypass menus     */
};
```

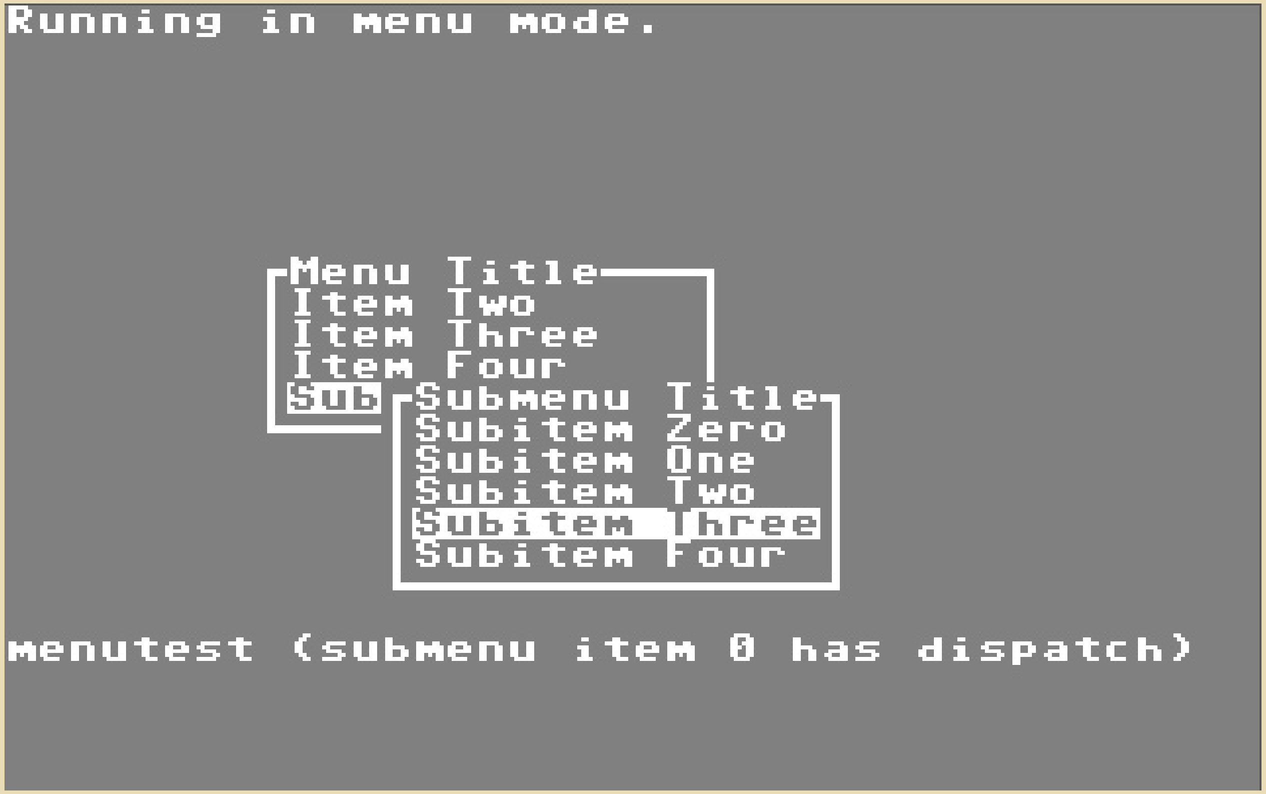
Menu Item Structure

```
struct menuitem {
  char *iText;
  int (*iAction)();         /* dispatch function  */
  int *iExtend;             /* for "subclassing"  */
  struct menu *iSubmenu;   /* pointer to submenu */
  struct menuitem *iNext; /* next menu item     */
};
```

Menu/Submenu Example

```
struct menu *menu, *submenu, *thisMenu;
menu = newMenu(8, 8, 0, "Menu Title");
menu->mVisible = 4; /* no. visible items */
menu->mCuritem = 2; /* selected item */
submenu = newMenu(12, 12, menu, "Submenu Title");
submenu->mVisible = 5;
submenu->mCuritem = 3;
addItem(menu, newItem("Item Zero"));
...
addMenu(menu, submenu);
item = newItem("Subitem Zero");
item->iAction = &action;
addItem(submenu, item);
...
thisMenu = doMenu(menu); /* or doList() */
```

Menu/Submenu Demo



```
Running in menu mode.








         ┌Menu Title───────┐
         │Item Two         │
         │Item Three       │
         │Item Four        │
         │Sub ┌Submenu Title┐
         └────│Subitem Zero │
              │Subitem One  │
              │Subitem Two  │
              │Subitem Three│
              │Subitem Four │
              └─────────────┘

menutest (submenu item 0 has dispatch)
```

Widgets

- widgets are a circular, doubly-linked list of structures
- label, text input, checkbox, listbox
- navigation with F7, F8, select with F1, dismiss with F5 (à la Q-Link)
- each widget can have its own allowed keys and end keys
- dispatch functions cause loop to exit

Widget Structure

```
struct widget {
  int wType, wLeft, wTop, wMask, wState;
  char wAllowed[16]; /* wchrin() allowed chars. */
  char wEndkeys[16]; /* wchrin() end keys */
  char *wText;
  int *wExtend;      /* for "subclassing" */
  int (*wAction)();  /* dispatch function */
  struct widget *wPrev;
  struct widget *wNext;
};
```

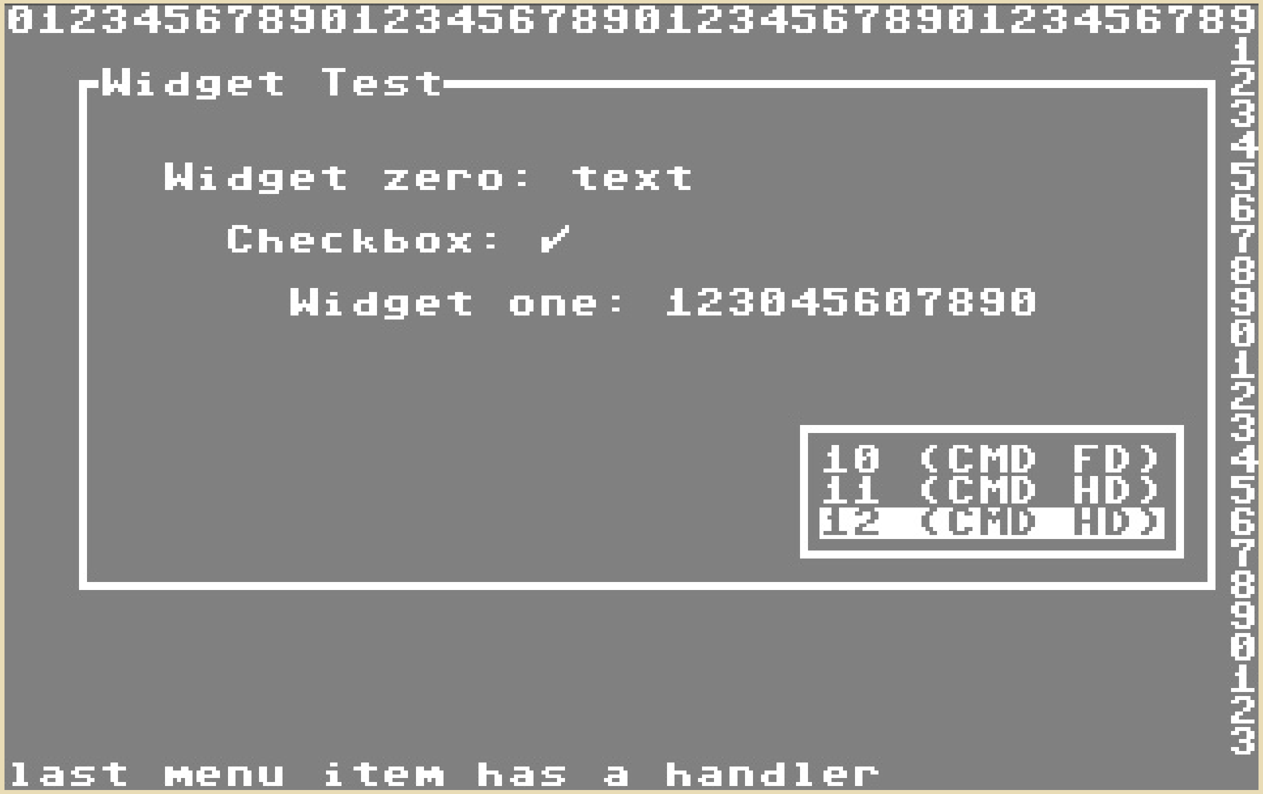
Widget Example

```
...
chkOne = newChk(14, 4, TRUE, " ");
strcpy(chkOne->wEndkeys, " ");
chkOne->wAction = &chkAct;
addWdg(chkOne, lblZero);
...
txtTwo = newTxt(18, 6, "123045607890");
txtTwo->wMask = MSK_NUM | MSK_ALW;
setAllow("-", txtTwo);
addWdg(txtTwo, lblZero);
...
mnuList = newMenu(22, 10, 0, "");
mnuList->mVisible = 5;
mnuList->mCuritem = 1;
addItem(mnuList, newItem("08 (CMD FD)"));
...
lst = newList(22, 10, mnuList);
lst->wAction = &itemAct;
addWdg(lst, lblZero);
...
```

Widget Example (cont'd.)

```
swl_init(); /* enable IRQ handler */
initWdg();  /* enable widget end keys (F1, F5, F7, F8) */
thisWdg = lblZero; /* first widget */
while (TRUE) {
  thisWdg = doWidget(thisWdg, FALSE);
  if (endkey == KEY_EXIT) {
    break;
  }
  if (thisWdg->wAction) {
    (*(thisWdg->wAction))();
    if (endkey == KEY_PREV) {
      do {
        thisWdg = thisWdg->wPrev;
      } while (thisWdg->wType == WDG_LBL);
    } else {
      thisWdg = thisWdg->wNext;
    }
  } else {
    break;
  }
}
killWdg();
```

Widget Demo



```
0123456789012345678901234567890123456789
                                       1
  ┌Widget Test────────────────────────┐2
  │                                   │3
  │                                   │4
  │  Widget zero: text                │5
  │                                   │6
  │    Checkbox: ✓                    │7
  │                                   │8
  │      Widget one: 123045607890     │9
  │                                   │0
  │                                   │1
  │                                   │2
  │                      ┌───────────┐│3
  │                      │10 (CMD FD)││4
  │                      │11 (CMD HD)││5
  │                      │12 (CMD HD)││6
  │                      └───────────┘│7
  └───────────────────────────────────┘8
                                       9
                                       0
                                       1
                                       2
                                       3
last menu item has a handler
```

Disk Enumeration

- list attached drives and their types (based on code by Todd Elliott)
- disk structure provided in header, but it's up the the programmer to populate it
- list partitions on CMD drives
- list files on current disk / partition (pass first structure, the rest are allocated)

Disk Enumeration Structures

```
/* file, directory, or partition */
struct dirent {
  int siznum; /* file size or partition no. */
  char fileName[17];
  char fileType[4];
  struct dirent *pNext;
};

struct drive {
  int device;
  int drvType;
  struct dirent *dPartn;
  struct drive *dNext;
};
```

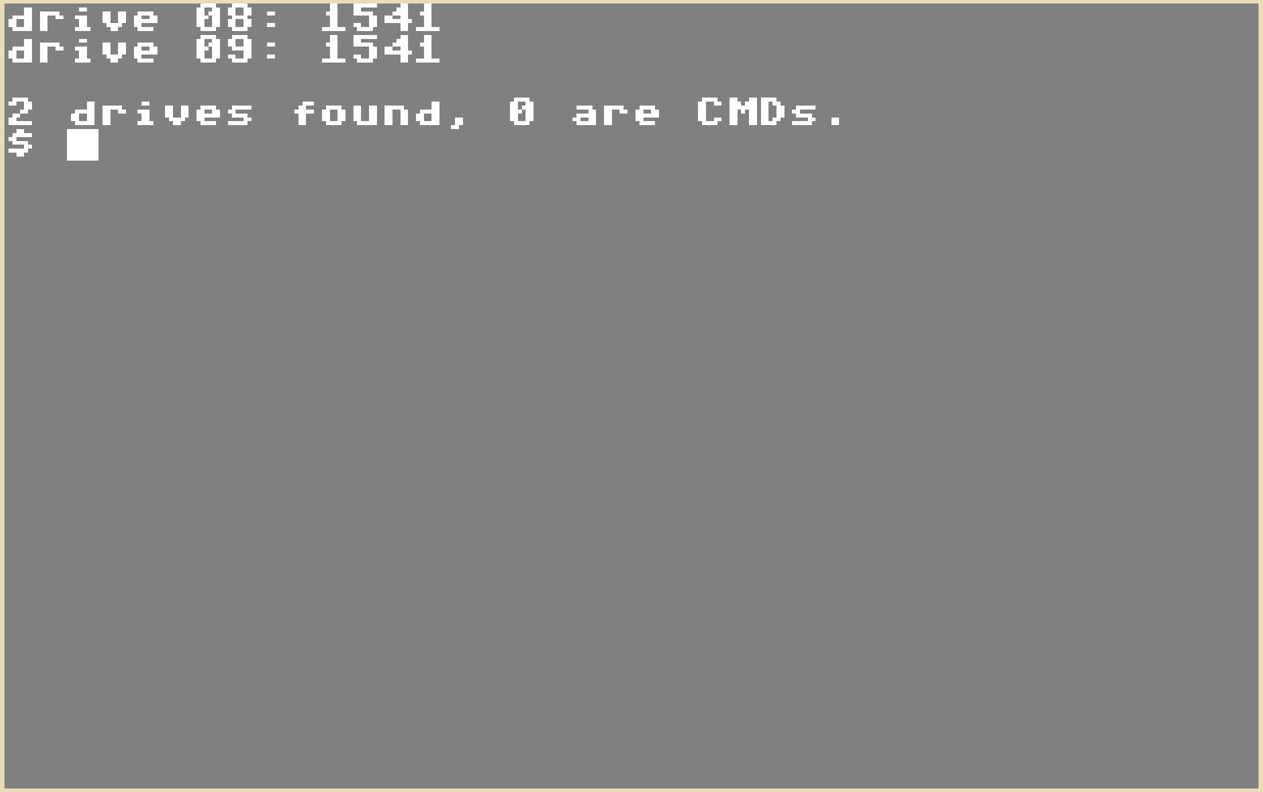
Disk Enumeration Example

```
extern char drives[];
struct drive *firstDrv, *nextDrv;
int i;

firstDrv = calloc(1, sizeof(struct drive));
drvQuery(); /* call assembler module */
nextDrv = 0;
for (i = 0; i < 23; i++) {
  if (drives[i]) {
    if (nextDrv == 0) {
      nextDrv = firstDrv;
    } else {
      nextDrv->dNext = calloc(1, sizeof(struct drive));
      nextDrv = nextDrv->dNext;
    }
    nextDrv->device = i + 8;
    nextDrv->drvType = drives[i];
    nextDrv->dPartn = 0;
    nextDrv->dNext = 0;
  }
  ...
```

Disk Enumeration cont'd.

```
  if (drives[i] & 0x80) { /* CMD drive */
    nextDrv->dPartn = calloc(1, sizeof(struct dirent));
    result = getDir(i + 8, nextDrv->dPartn, TRUE);
    if (result) {
      nextDrv->dPartn = 0;
    }
  }
}

/* At this point, dPartn is the head of a linked list of dirent
   structures (which could also be files and directories). */
```

Enumeration Demo



```
drive 08: 1541
drive 09: 1541

2 drives found, 0 are CMDs.
$ █
```

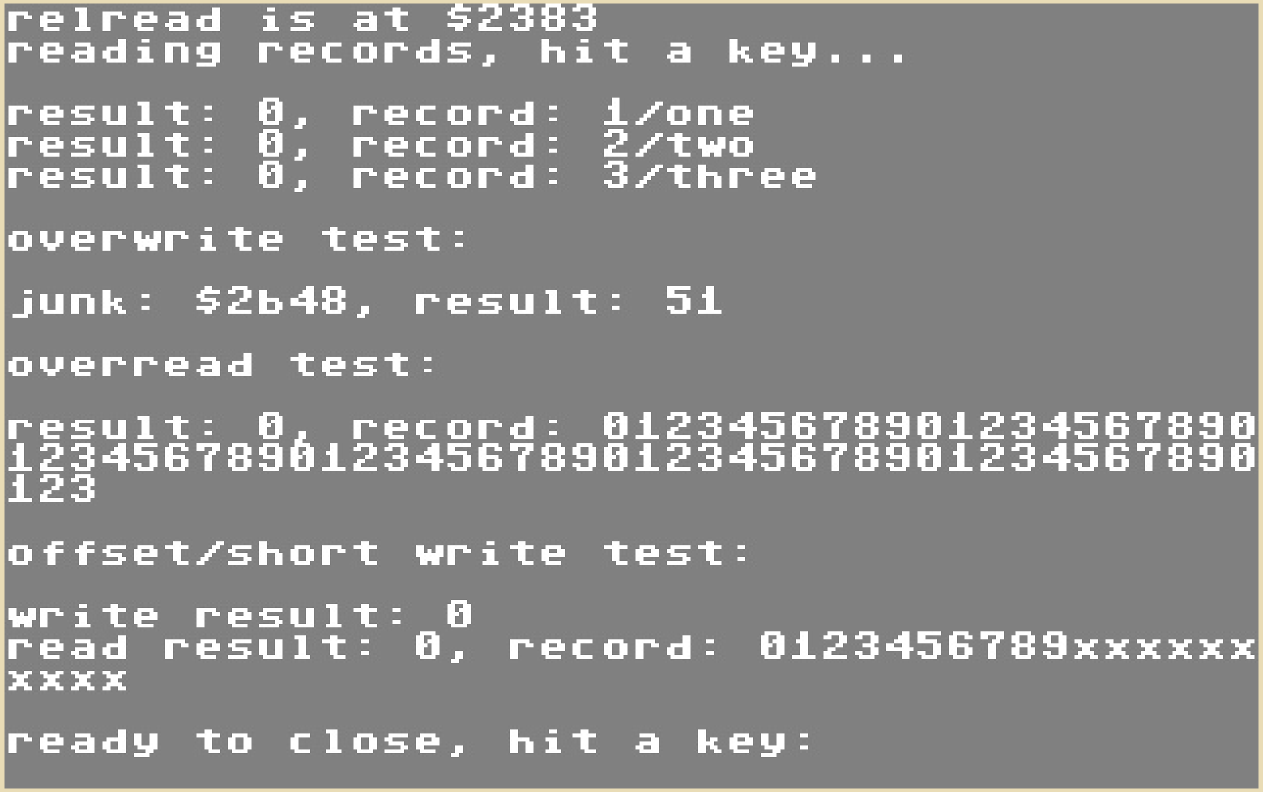
Relative File Support

- provides workaround for the infamous “Shiloh's Raid” bug (based on George Hug's code)
- record offsets supported
- correctly deals with overflows (error 51) and attempts to read past the end of a record
- OK to use **fopen()** and **open()** at the same time the relative file is open

Relative File Example

```
struct datum { ... };
int result, recno, offset;

result = relopen(5, 8, 5, “relfile”, 64);
recno = 123; offset = 1;
result = relwrite(&datum, sizeof(struct datum),
         recno, offset);
result = relread(&datum, sizeof(struct datum),
         recno, offset);

/* Note that there is no position command in the
   API, as it is always used within the context of
   a read or write. */

relclose();
```

Relative File Demo



```
relread is at $2383
reading records, hit a key...

result: 0, record: 1/one
result: 0, record: 2/two
result: 0, record: 3/three

overwrite test:

junk: $2b48, result: 51

overread test:

result: 0, record: 01234567890123456789012345678901234567890123456789012345678901234567890
123

offset/short write test:

write result: 0
read result: 0, record: 0123456789xxxxxx
xxxx

ready to close, hit a key:
```

Getting the Libraries

- http://lyonlabs.org/commodore/powerc.html
- I'll be traveling during May 2007, but will be available for questions after that
- Q-Link: ShadowM
- gholmer@ameritech.net
  (for the Q-Link challenged)