CUGS the MONITOR

October, 1992 Commodore Users Group of Saskatchewan Vol. 7, No. 8

Obligatory Stuff

EDITORIAL

by
Jarrett Currie

October, already! Not only is it getting closer to Christmas, but it's also getting closer to that annual event when the members of CUGS elect a new Executive. And that, of course, means that everyone in the membership has the opportunity to volunteer for the most prestigious position of Editor. I hope that anyone who is interested in the position will seriously consider letting their name stand for this rewarding position.

Some time ago I read an article mentioning the "WOW" factor in regards to the field of computers. I have spent some time pondering this devised factor, and I have realized that now, more than ever, this factor has become communicable.

Let me explain: when a salesperson shows off the new features of a computer, the victim of the sales pitch (ie. customer) invariable remarks "WOW", hence the name of this factor. Typical examples of such awesome features include: millions of colors, multi-megahertz of speed, multi-megabytes of storage, huge video resolutions, and so on. Each of these features, while impressive in their own right, often are more of a selling feature than of significant use to the end user. The "WOW" factor soon wears off, and the buyer wonders why they spend their hard-earned money on a machine that gives them so little satisfaction. The excitement felt in the high-tech store quickly fades, and is replaced by an overwhelming sense of confusion.

Home computer users are purchasing these powerful computers after succumbing to this expensive affliction. As software becomes more powerful, thereby causing an increased "WOW" factor, it requires more powerful machinery to run it. But, paradoxically, it is not necessarily the functionality of the software that is used by the home user that is eating up these extra resources. Consider a popular word processor available

The Deadline for the Next Issue of the Monitor is October 23, 1992 for the Windows environment for the IBM computer. The operating system takes of a couple of megabytes of disk space, and requires at least 2 megabytes of internal memory to run satisfactorily. And that's only the operating system! The word processor takes many more megabytes of storage, and costs as much as a C128! Of course it does everything anyone could every hope to do with a word processor, but much more than any home user would expect to use. Naturally only those afflicted by the "WOW" factor would consider purchasing a Ferrari only to drive to the 7-11!

I am composing this editorial with a word processor I purchased for $8 at our club's software sale. It does most of everything I would want to do with a word processor, and for those very rare times when I need to do something extraordinary sophisticated, I can always go to an IBMer's house and use their very expensive models!

The President's Message

Hello again. Welcome to the CUGS Monitor for October and to the month of Halloween. A time of year we can play harmless pranks and dress up as strange things. He he he... ha ha ha... snicker! What can we do to scare someone this month? (noise of door creaking and squealing open) Scare them out of their closets? He he he, let's be sneaky about this one and plan it well. Ssshhhhht be quiet, AAAAHHHHH, I know, let's scare some people by pulling our 64/128s out of the closet and scare them with what an 8-bit computer can do compared to the high priced mega-memory monsters out there. Ha ha ha ha ha (eerie laughter echoing in the distance, gradually fading into the background noise of the line noise beast [see last issue by Tristan Miller]).

At the last Executive meeting, it was decided to plan a mall demo to show off the 64/128 line of computers running various programs. Our hope is to give the club some exposure to the general public. We hope to stir up peoples' interest in getting their 62/128, or even the Vic 20, out of the proverbial closet and dust them off and get them working. We would like to bring up the subject of the mall demo and let you help us with ideas, suggestions and comments on what we can do to make this a success.

In other news, I want to inform you of the BBS activity statistics over the last year. In August, the board saw over 400 file transfers take place. The average number of calls is approximately 15 or more callers a day with the average time on of 15 minutes. The board has seen over 7900 calls since it first went up in February, 1991. Since then, over 21 megabytes of files were up-loaded and over 62 megabytes of files were sent out over the phone line. 105,000 messages have been read, 5500 messages posted. That translates to each message being read an average of 19 times. The BBS has a 700 message capacity set up right now and can be expanded if need be. The board has seen about 187,000 minutes of on-line use (or approximately 130 days of solid 24 hour a day use). 60% of the callers call using an 80 column terminal while the other 40% use a 40 column terminal.

Adventure Construction Set
by
Tristan Miller

Ever wished you could write your own computer game? On the other hand, ever wished the computer could WRITE A COMPLEX GAME ITSELF for you in about an hour? Well now all this is made possible, with Adventure Construction Set from Electronic Arts.

This program, designed and written by Stuart Smith, is specially designed so that even the most inexperienced of programmers will be able to create their own adventure games with ease through the use of simple menu-driven commands and graphics editors. The program enables you to create Ultima-type adventures using the three basic construction sets: Fantasy, Spy/Mystery and Science Fiction. Each set has custom-designed graphics and objects for its theme, which can be altered, added to, or erased.

When you first boot up the program, you are presented with a menu with various items. By choosing "CONSTRUCT AN ADVENTURE", you are taken to a menu which is totally joystick/mouse operational. You are asked for the name of your game, your name, its theme (Fantasy/Mystery/Sci Fi), and a few other things, such as theme music and opening text message.

By selecting "DO MORE DETAILED WORK", you a presented with more menus, including a map menu and an object/creature menu In the map menu you can design the main map and several sub-maps. If you've ever played an Ultima, you'll get the hang of this style of cartography immediately. However, to "non-Ultima" people, the concept is still easy to grasp.

On the main map you can choose where to place different terrain (mountains, forests, rivers, grassland, etc.) which is predefined (but can be altered). You can also define terrain to perform different functions (e.g. you can make rivers navigable only if one owns a canoe). Also on the map you can place up to 15 "doors" leading to "regions". For example, you could place a picture of a city on the map and when that picture is "stepped on" the player will be taken to a detailed map of the city.

In the detailed maps, you can place objects, creatures, and text messages. Each region can have up to 16 rooms and 300 objects. You can also place areas that play music or send text messages, or both.

The third menu features an object, creature, and graphic editor. These are wonderfully designed; being as complicated or as simple as you like. In the creature editor, you can edit or create people, animals, or monsters that move around, talk, cast spells, and attack people. You can also choose their personality and their alignment (friend/enemy/neutral) towards the player. You can equip them with the weapons and protections you create in the objects menu

You can create hundreds of places and objects that talk, play music, make sound, display messages, move you around, help you in the adventure, cast spells, protect you, or a combination of these. You can create stores that sell your objects and thieves that can steal them. The whole editor is totally menu-driven and comprehensible to even the most novice programmer.

Finally, you have the graphic editor which allows you to edit (in high-resolution multi-color) the graphics for your creatures, objects, places, terrains, walls, buildings, doors, costumes, weapons, armours, and magical items.

The really big surprise, however, comes when you tell the computer to write its own adventure (or even finish yours!). Simply type in your parameters and at the press of a key, ACS will start making your adventure. It may take an hour or more (depends on the size you selected), but if you have JiffyDOS or some other fastloader then the process should speed up greatly.

I have played several ACS and have found them to be quite good. Whenever I'm bored on a Saturday afternoon I just load up ACS, pop in a disk and let it go, and within a few minutes I have a great adventure ready to play.

This program used to have a club devoted to it, but since ACS has been out of production for a number of years now, it has been dispersed. However I hope that those of you reading this article would like to start a club up again so we can swap adventures. If anyone has ACS and would like to trade games please give me a call at 586-2036.

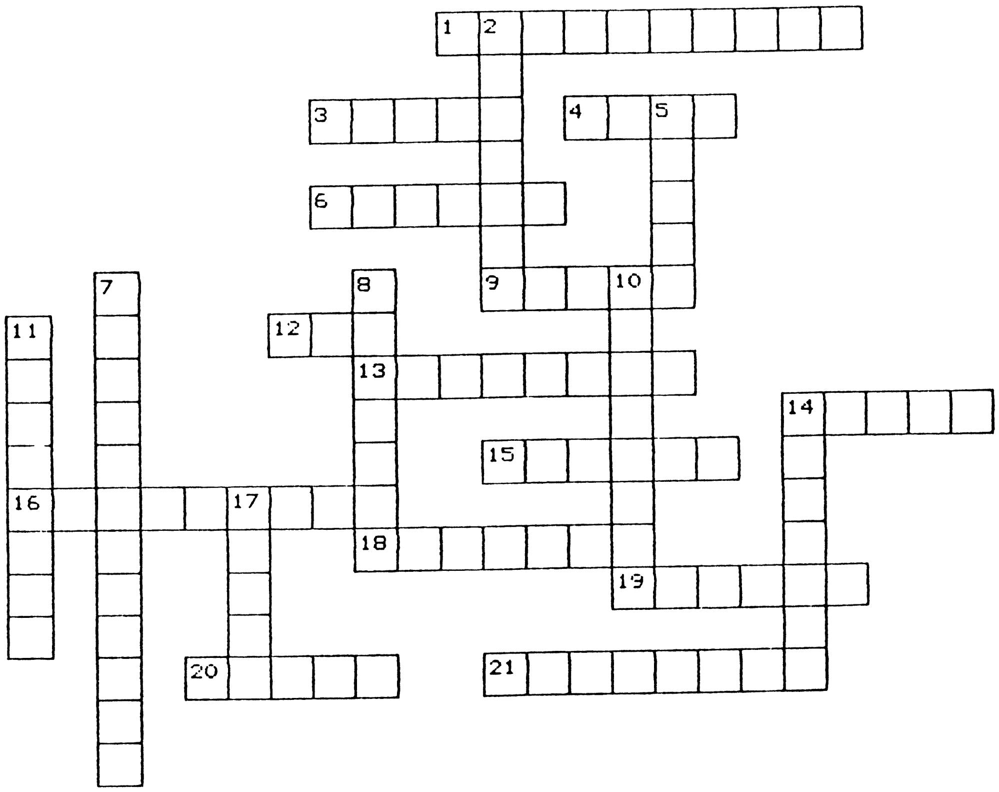
OK, now for how to find this program. This should be relatively hard to do, seeing as it has been out of production for many years. However alot of used-to-be Commodore dealers still have ACS on their shelves, collecting dust. It's really a shame though because this is the best program/game I've ever played. At the present, I know of only one store who has a large supply of ACS: Zellers in the Golden Mile. If anyone wants it they'll have to go to the audio/video department and ask that the software case be unlocked so you can get your hands on it.

Well I see I've overstepped my space limits by a bit so that's all for now.

Crossword Challenge

by Barry Bircher

HALLOWEEN



ACROSS

1. VICTIMS REACTION TO A CURSE
3. WITCHES VEHICLE
4. FLYING CREATURES
6. ONE OF 2 THINGS
9. WHEN THEY ALL COME OUT
12. SCARY WORD
13. ANIMAL ASSOCIATED TO HALLOEEN
14. HANDED OUT
15. FRUIT
16. TIME OF YEAR
18. CREEPY CRAWLERS
19. SCARY NOISE
20. VISIABLE SPIRITS
21. NOTHING BUT BONES

DOWN

2. VEGTABLE
5. GOOD THINGS
7. SCARY LIGHTED PUMPKIN
8. LITTLE DEVILS
10. THE SPECIAL HORSEMAN
11. YELLS OF FUN
14. STICKY THREADS
17. FEMALE WARLOCK

QLINK SPECIAL

LoRes to HiRes

by Steve Emsley (SYSHELP SE)

Converting lores to hires is not that hard, because in a way, lores is a form of hires mode. Each keyboard character is constructed of an 8 by 8 matrix of bits, either on or off.

This will explain how a lores screen is converted. Since it will become a Doodle! file, the color will be saved as well. The procedure will also take care of redefined character sets if they are used; the VIC chip video bank, whether it is the default bank 0 or not; and screen memory relocation.

The first step is to open a program file with the name DDfilename. I'm not going to get into this, since most know how it's done. It will not be a SAVE kernal routine. Instead, each byte will be written one at a time. All that's necessary for now is to open a PRG file for writing, and write the bytes #$00 and #$5C to it so that it will have a load address of $5C00.

The first task is to find the screen and character set locations. It is done in a series of steps. I'll use locations instead of labels for the benefit of people without memory maps.

A temporary location called BANK will be used to hold the video bank that the VIC chip is using...

```
LDA $DD00:AND #$03:STA BANK
```

The VIC chip sees memory in 16k byte chunks called video banks. There are four video banks in the 64k byte address space of the computer. The most common is bank 0. We now will find the page in memory where the character set starts. It always starts on a page boundary, so we do not need a 16 bit address, just a page number. We will set aside a location called CHARSET to hold the page number. First we put the video bank base page number in it. This can be done with a data table set up with four values of C0, 80, 40 and 00. We'll call the table WHICHBANK. The bytes are in this order because the BANK value is a twos complement of the actual video bank relative to the address space.

```
LDX BANK
LDA WHICHBANK
STA CHARSET
```

Now for the character set.

```
LDA $D018:AND #$0E
```

These bits tell us where it is in memory, but we have to multiply it by 1024 to get the actual address. 1024 is 256 * 4, so we can shift the byte left twice and combine it with the page pointer we are setting up.

```
ASL:ASL
ORA CHARSET:STA CHARSET
```

This now tells us where in memory the character set is. We could run into a problem, though. If you are using the normal ROM set, the location will point to page $10, which is the ROM image at location $1000 rather than the actual location in RAM. The actual data is at $D000 even though the VIC chip thinks it is at $1000. Simply check for a $10 and change it to $D0 if it is. Then save the final result in the location CHARSET.

```
  CMP #$10:BNE +
  LDA #$D0
+ STA CHARSET
```

We're done with this for a moment until we are ready to get the bitmap data for the characters. Now we can start writing the data to the file. The file contains 1024 bytes for the color which will be loaded at $5C00 through $5CFF. Lets go...

We have to get the background and foreground for all 1000 locations. The background is easy enough to find.

```
LDA $D021:AND #$0F:STA BACKGROUND
```

Since we're done with locations $03 and $04, we'll use these to get the foreground colors, which start at $D800.

```
  LDY #$00:LDA #$D8
  STY $03:STA $04:LDX #$04
- LDA ($03),Y:ASL:ASL:ASL:ASL
  (this puts the foreground into the high nibble)
  ORA BACKGROUND:JSR $FFD2
  (now the background is in the low nibble,
  write it to the file)
  INY:BNE -
  (until one page is done)
  INC $04:DEX:BNE -
  (until 4 pages are done)
```

We're writing 1024 bytes (24 extra), but remember that Doodle! files start the bitmap at $6000, so they save the extra 24 bytes in the file.

Now comes the fun part: getting a character, finding its 8 byte description, and writing it to the file. Set up locations $03 and $04 with the beginning of screen memory.

```
LDY #$00:LDA $0288:STY $03:STA $04
```

Even if the screen is relocated, location $0288 (648) will have the correct page for the beginning. Set up 2 locations which will be used to find the character data. We'll use $05-$06 for this. Another spot called PICSIZE will be a counter of pages for the size of the picture. Put a 4 in this location, since the screen is 4 pages in size. Remember that we have the page number where the character set is at in a location called CHARSET. We don't have to worry about the low byte since it will always be zero. The following is then done for each character on the lores screen:

```
BEGIN LDA ($03),Y:STA $05
LDA #$00:STA $06
```

(a quick example: if the character was the letter A, locations $05-$06 would look like this: $01 $00)

This is the character NUMBER. We multiply it by 8 to find the offset to its data bytes. Multiplying by 8 is done by shifting the 16 bits left three times...

```
          LDX #$03
        - ASL $05
          ROL $06
          DEX:BNE -
```

Add the base page address of the character set in RAM...

```
          CLC
          LDA CHARSET:ADC $06:STA $06
```

Locations $05 and $06 now contain the pointers for the data that makes up the letter A. At this point, we need to save the Y register, since we will use it for something else.

```
          TYA:PHA:LDY #$00
```

```
        - SEI:LDA $01:AND #$FB:STA $01
```

This is necessary to get to the ROM charset. A quick check could be put in here to see if its redefined, but its really not necessary.

```
          LDA ($05),Y:PHA
          LDA $01:ORA #$04:STA $01:CLI
          PLA:JSR $FFD2
          (after all that, we write to the disk)
          JSR $FFB7:BNE DONE
          INY:CPY #$08:BNE -
          (at this point, all 8 bytes have been written
          to the disk. Time to get the Y register that we
saved)
          PLA:TAY:INY:BNE BEGIN
          INC $04
          (point to the next page of scrn RAM)
          DEC PICSIZE:BNE BEGIN
```

That's basically it. At this point, you would have your DONE routine to close the file and proper channels.

The Write Disk by Barry Bircher

I have always wanted to write about using BASIC to create a disk file, store information to it and read it back again. When I first started using my trusty 64 back in '85, I wrote lots of programs for my own use. For several months, there was no need to learn about how to open up disk files and store information in them. There was lots of room for storing any information right in the program. I was content to just the information in print statements or data statements and use the regular SAVE"FILENAME",8 to save it or LOAD"filename",8 to load it back in. However, as BASIC got easier to understand and the programs got more elaborate, there came a time when learning to write information to the disk was required. Hopefully this instalment will help those of you who resisted the programming aspects relating to disk files.

Files on disks are great ways to store information about various things (just about anything) besides just programs that are loaded and saved. But how can a person, who is writing a program, save information about various things the program is manipulating? Having written dozens of programs in the last several years, I cannot program anything, it seems, with out creating a file to save information about this and that. It may be numbers, peoples' names, addresses, 6/49 winning numbers, data base storage, computed results, etc.

Creating a file on disk is as simple as opening up a channel to the disk drive, writing to that channel and closing it. Retrieving information from that file is just as easy: open up a channel to a filename, getting the information, and closing the file. The following is the bare essentials; disk error checking routines are not included. I will be giving more details into improvements to this little starting program in the next issue of the Monitor.

For example, to create a file:

```
10 open 2,8,2,"my first file,s,w"
20 print#2,"this is my first data file!"
30 close2
```

and to read it back in at a later time:

```
40 open 2,8,2,"my first file,s,r"
50 input#2,a$
60 close2
70 rem a$ now contains the string "this is my first data file!"
```

Perhaps a little explanation is in order to better understand what is going on.

__Line 10__
Opens up channel 2, to drive number 8, with a secondary address of 2 (not really important but included to confuse you; I'll explain what that is in a later instalment). With a filename in quotes following the open command, the drive number 8 will open a sequential file named "my first file". The ',s' tells the drive that the filetype is to be a sequential and the ',w' means that you intent to write to it. If you omit the ',w' the drive will think you want to read it only. It's the ',w' that actually tells the disk drive to create the file. (Note: if a file with the same name is already on the disk, the drive light will begin to flash telling you that there is a problem. I will deal with disk errors later). After line 10 is executed, the drive will create a seq file and the drive light will remain lit (this indicates that a file is open to the drive).

__Line 20__
Simply prints to channel number #2. What is printed can be text in quotes, a variable or a string variable.

__Line 30__
Closes the file. The CLOSE command is important because it tells the drive you are finished using that file. In our case, since we have written something to the drive, this tells the drive to actually save the information to the disk if it has not already done so, closes the file and updates the directory. When a file is opened up for writing, the drive actually opens up a "SPLAT" file. You may have seen those "SPLAT" files on some of your disks. A "SPLAT" file gets its name simply because an improperly closed file will have an "*" beside the filetype when the directory is listed eg:

```
   23  "my first file"  *seq
```

The main thing the CLOSE command does is to update the number of chunks of disk space used by the file and remove the "*", signifing that the file was properly closed. (Note: blocks is the common term used to describe 254 byte chucks or sectors on the disk.) As a side note: If by chance you do see a "SPLAT" file on a disk, do not be tempted to scratch (delete) this file (ie: open15,8,15,"s0:filename":close15). It is much safer to use the validate command (open15,8,18,"v0:":close15). The validate command will properly get rid of that "SPLAT" file, wheras the scratch command could inadvertantly get confused and corrupt some other files on the same disk. Why it does this is not really a mystery nor is it a bug, but simply the way the scratch command works. It is a subject I could discuss about next month when I talk about disk errors.

Once the file is closed, your data is now stored on the disk.

__Retrieving the info back__
At a later time, it will be necessary to get the information back out that's contained in the file by using lines 40-60.

__Line 40__
Opens up the channel to your file. The ',s' again is the file type wanted. And the ',r' tells the drive you want to read from it. The ',r' is optional as all files opened default to read mode.

__Line 50__
Inputs one string from the drive and assigns it to the a$ variable.

__Line 60__
Closes the file.

So there you have it. A very simple, no frills and chills program to create files and read them in later on. Any improvements to that program is primarily used to catch drive errors that can and do occur. Next month, I will delve into the mysteries of disk errors and how to communicate to the disk drive and find out what that flashing light means.

EXPERTS LIST

Wordprocessing		
Paperclip (to version E)	Jarrett Currie	757 2391
Paperclip (any version)	Ken Danylczuk	545 0644
Pocket Writer	Barry Bircher	543 8840
Pocket Writer	Real Charron	586 1843
Fontmaster II	Michael Rodgers	728 2595
Spreadsheets		
Pocket Planner	Barry Bircher	543 8840
Better Working SS	Ken Danylczuk	545 0644
Databases		
Pocket Filer	Barry Bircher	543 8840
Oracle (Consultant)	Ken Danylczuk	545 0644
Communication		
Desterm 2.0	Barry Bircher	543 8840
Pro128Term	Jarrett Currie	757 2391
Library files	Barry Bircher	543 8840
Music/Sound		
(Most)	Ken Danylczuk	545 0644
Stereo Sid Editor	Michael Rodgers	728 2595
Enhanced Sid Player	Michael Rodgers	728 2595
Languages		
Forth	Ken Danylczuk	545 0644
Pascal	Ken Danylczuk	545 0644
ML (machine language)	Ken Danylczuk	545 0644
ML (machine language)	Barry Bircher	543 8840
BASIC (2.0-7.0, files)	Ken Danylczuk	545 0644
Graphics		
Print Shop/Master	Ken Danylczuk	545 0644
Koala Painter/Printer	Ken Danylczuk	545 0644
Hardware		
Disk Drive Maintenance	Ken Danylczuk	545 0644
GEOS		
GEOS 64	Jarrett Currie	757 2391
GEOS 128	Barry Bircher	543 8840