EDMONTON 99'er COMPUTER USERS' SOCIETY

The Edmonton 99'er Computer Users' Society
p.o. box 11983
Edmonton, Alberta
CANADA T5J 3L1

99'er ON LINE... is the news letter of the Edmonton 99'er Computer User's Society published ten times a year. Unless otherwise stated, all articles may be republished in other news letters provided that source and author are identified. We will credit authors quoted in 99'er ON LINE.

CORRESPONDENCE: Newsletter editor: Bob Pass, 59 Labelle Cr, St. Albert, Alberta, Canada T8N-2G6. (403)458-7658. All other correspondence should be sent to the address at left.

OFFICERS: president Tom Hall, vice-pres Ken Godbeer, treasurer Jim Mulligan, secretary Roxanne Appelt

REGULAR MEETINGS... of the Edmonton 99'er Computer User's Society are held on the second Tuesday of each month in room 849 of the General Services building of the University of Alberta from 7:00 till 10:00 PM and are open to all members in good standing. Non-members may attend their first meeting free of charge.

ADVERTIZING... Commercial space is available in this news letter at the following rates: full page $20.00, half page $15.00, 1/4 page $10.00. Discuss your needs with Jim Mulligan at 467-6021, at the next meeting, or send "photo ready" copy to the P/O Box above. Members may advertise their personal computer related items for free but are asked to limit their ads to about 50 words. Mail your ads to the editor's address or hand it to him at the general meeting; newsletter deadline 3'rd Monday of the month.

MEMBERSHIP FEES: Family 12 months, $20.00, 6 months, $15.00. Students 12 months, $15.00, 6 months, $10.00. New member initiation. $20.00.

MARCH MEETING

CLUBLINE/99 -- The February issue was distributed at the meeting. About 5 extra copies are available at each meeting for general sales at $2.00 each on a first come, first serve basis. Also, back issues of the monthly diskettes are available; place your order with Bob Pass.

ATTENDANCE -- at the meeting once again exceded 30 members. It is good to see a good turn out for the meetings.

GROWLIES -- Ken Godbeer was at the meeting, so we got donuts this month! (Thanks for ruining my diet Ken.)

YVES CHEVALIER -- lugged in about 80 pounds of teletype rolls and gave them away to those who could make use of them. Thanks Yves.

DEMONSTRATION -- Paul Helwig presented a second tutorial on c99 covering the general points required to get a program written. The program discussed used the circle graphics utility included in the c99 package. Paul handed out Xerox copies of the program to the attentive listeners. Paul also announced that a c99 SIG will be set up shortly.

NEXT MEETING

The next meeting will be Tuesday, April 14'th at 7:15 PM. Same place as usual; General Services Building, U of A campus in room 849.

The next meeting will be "Pot luck" again as I have no idea what's on the agenda!

NEWSLETTER DISKETTE AWARDS

This month's awards are a repeat of last month. A disk and a hearty "Thank-you" to Michal Jaegermann for contributing yet another artical in his c99 series and a ready to run memory browser program written in, of course, c99.

Repeating his award of last month, Jim Beck has submitted another of his famous games programs. But this one is different, it is written in c99 which is a major departure from Jim's usual XBasic submissions. Thanks Jim for sharing the c99 world with us older types who are too chicken yet to tackle this great language.

Don't forget folks that by contributing to this newsletter you become eligible for a free diskette. So let's see some more input from you closet hackers!

FIRST STEPS INTO c99 - A PROGRAM AT LAST

by Michal Jaegermann

After I *needed* a whole, long article to explain how to print a number on a screen it is a time to present a real, albeit short, program. I have chosen a "memory browser" - a program which will let you to walk into different places in memory, of our computer, not yours, and examine its contents. After all, what we have to do is to print a lot of hexadecimal numbers. A source for the program accompanies this article. It will be also available from our bulletin board, with some extras included. It was prepared for the latest (2.1) version of c99 compiler. If you are still using the old one, then some minor modifications will be in order.

First question - why to use hexadecimal numbers. (Old assembler and Forth hackers may skip that paragraph). As you know, our computer, and nearly all others, stores internally everything in a form of fixed length strings of binary digits, a.k.a. bits. A natural unit for a 9900 processor will be a sixteen bit word. This is great for a computer, but hardly readable for humans. On the other hand, every group of four bits can be considered as a representation of a number between 0 and 15 (decimal) and represented as a ONE hexadecimal (base 16) digit, cutting the length of a typographical representation by four and making it much more heterogeneous visually, therefore easier to read. That is a reason for TI-BASIC to use a hexadecimal notation for character patterns - you never knew that you were talking hex? - or to display a computer memory that way.

To show all this hexadecimal galore we may use a developed previously function, or use something from available libraries - like printf. This is great, but one may encounter some problems, like the fact that we would probably prefer to keep leading zeros. Also a sign problem may cause some headaches. It is true that a full printf, which satisfies all requirements of ANSI standard for C libraries, will let you specify all fancy formats and more on a top of it. On the other hand you may find out that a size of its object file far exceeds a size of your small program. On the other hand, a correspondence described in a previous paragraph lets us to use shortcuts and to achieve our goal in a simple way, without divisions, taking remainders and similar costly operations.

A plan is as follows: take a sixteen bit number 'num' and perform the following operations:

- repeat four times procedures below
- cut off the last four bits of 'num'
- convert the result to an ASCII code of a corresponding hexadecimal digit
- adjust a pointer to the rightmost free position and store there the converted digit
- shift right 'num' by four bits

Once you are done you may add or necessary and unnecessary embelishments and show a result of your efforts with puts.

This is implemented, nearly as described, as a function tohexs (a short for tohexstring), which as a second argument takes a pointer to a location where we want to have a leftmost digit of our number. It would be possible to pass a pointer pointing right past the rightmost digit. This would let you save a statement "pos = pos + 4", making your program a little bit faster and a source much harder to comprehend. Don't do it. If you are really in a situation in which speed is more important then a style, then write a whole function in assembler, heavily commenting your code. Gains will be much higher.

Two more words on "cutting". In order to extract four lower bits I perform a bitwise AND of 'num' with a binary pattern "0000000000001111". c99 supports only decimal constants, so I have to use a decimal equivalent of that pattern and this happens to be 15 = 8 + 4 + 2 + 1.

A proposed way to do the job is, of course, not a unique. You may start, for example, from left, cutting the first four bits with a pattern "1111000000000000" = -4096 and continuing from that on. Some limitations of c99 will turn our to be troublesome, and conversion to ASCII symbols will be more complicated. But maybe somebody else have better ideas?

If you will look at the program you will find out that "tohexs" stores results of its efforts in two display buffers called 'lbuffer' and 'dbuff' declared at the very beginning of the program - outside of any function. These two objects are global, i.e. accessible by any function in the program. For example, an array of characters 'lbuffer' is initialized by a function "main" and later "showline" writes something to it. This of course can be done not only with an array, but with any other variable or pointer. And then you may use them exactly in the same manner like in BASIC. Actually, in c99 at least, an access to global variables is somehow faster then to locals. So why not to declare all needed variables at the top of a program and to get rid off all function arguments with theirs declarations and so on. Exactly because they are global. This means that one function can influence what the other one is doing, without telling nobody, in an explicit manner, what is going on. In no time at all you have such mess, then even an author of a BASIC program after two weeks has a hard time to tell how his program works. Also a library function could not use globals, because it is a black box, which does not know, or could not care less, what are particular variable names used by a calling program. When you are passing an argument to a function you are passing its VALUE, a global variable tells everybody where it is LOCATED. Another trick, impossible with global variables, you will find in a function "percnt". (What is that - look carefully!!). So why to use globals at all. Well, sometimes you have a variable which will be accessed by nearly everybody. Like a timer counter in a game, which can be incremented by nearly everyone. And everything is fine if you are dealing with a small number of well defined, and

clearly described in comments, globals. use of which is clear to everybody. Frankly speaking, in our program 'dbuff' does not have to be a global at all, and a need for 'lbuffer; to be global can be eliminated for a price of passing one extra pointer to "showline". But I wanted to have something - purely for a demonstration purposes.

Note also a way in which 'dbuff' was initialized. It would be possible to do exactly the same with 'lbuffer', but I decided against writing a string of 39 characters - mostly blanks. A question - why 39 characters and not 40? An array 'lbuffer' has 40 locations numbered from 0 to 39.

A hexadecimal input to the program is read, for a change, with a library function "scanf". A declaration "extern scanf();" tells c99 compiler that a reference to scanf will be resolved by a linking a program with another file. A string "%x" in an argument list is a format string and tells scanf that it should consider an input string as a representation of a hexadecimal number. An expression 'scanf ("%x %d %s", &a, &b, buffer)' tells scanf - read three strings separated by blanks: convert the first one as hexadecimal number and store in a variable 'a', the next one should give a decimal integer to be stored in 'b' and the third one is just a string to be copied into a 'buffer'. Note, that scanf has to change variables a and b, so it is not interested in their values, but it has to know where they are located. So I passed as arguments ADDRESSES of a and b. 'buffer' is presumably a name of an array of characters, therefore it is a required pointer. Taking an address of a location where this pointer is is stored will not help you at all. Scanf will clobber it and probably some other locations as well. Watch for that bug. Remember that "scanf" in c99 is only some approximation of a similar function in C. It follows that some more subtle examples, described in C tutorials, may not work here.

A next new construct, which I did not used before is a switch. I think that it is rather clear what this thing is doing. Watch out for "break;" statements which are terminating actions for different cases. If you will miss one, then an action will "fall through" to the action of the next case. This is exactly what I wanted for 'b' and 'f', but in other situations a missing "break;" would cause some chaos. The last "break;" is, strictly speaking, not required. But this is, so called, defensive programming. If I would want to change, for any reasons, a sequence of cases, or to add some more, the fact that this "break;" is already in place may make my life much easier. A switch may have one more optional label, which usually follows all "case" labels, and is called "default :" , without any constant (do not forget your colons). This will performed when all other cases will be missed. This more general form may look something like that:

```
.....
case (key)
{
    case 'A' : puts ("key A was pressed"); break;
    case 'B' : puts ("key B was pressed"); break;
    case 'C' : puts ("key C was pressed"); break;
    default  : puts ("some other key was pressed");
}
.....
```

Since, in the program, no special action in other cases is taken then "default :" was skipped. If you missed what this breaks are for, you may include the fragment above into a small test program, remove some breaks, and see for yourself what will happen.

One more thing which require explanation is a way in which I mix different types. For example, in a function "showline", which assembles and shows one line of a display, 'from' was declared as a pointer to integer and 'cptr' as a pointer to character. Later I have a statement:

```
cptr = from;
```

Well, C is not enforcing very strongly a distinction between different types, assuming that a programmer knows what is s/he doing. but a decent C compiler should complain at this moment, at least with a warning. C program needs in this situation so called cast. c99 does not have any casts at all, but lets you get away with a strangest possible assignments - which can be dangerous for your mental health while debugging. Moral - if you are doing stunts without a safety net, then be double careful. Why I needed a change of type? I am displaying a memory in words and in printable characters if possible. Now, where a pointer to integer points when it is incremented by one, and is the same answer valid for a pointer to a character? When in doubt consult the previous installment.

A similar trick is performed in "percnt" function, where I would like to perform calculation on addresses in an "integer fashion". A knowledge that pointers are 16 bit quantities is dangerous for portability. This may not be true on the other machine. But c99 programs are not portable anyway, so little is lost here. A question may arise why I will not declare all arguments to "percnt" as integers. The point is that address comparison is unsigned one, which is exactly what I need. That also accounts for a seemingly unusual way in which I bring a 'dump' pointer back into range. (A code which follows switch). Ponder a little bit over that fragment of a program.

A compiler bug, mentioned in comment in "percnt" consists of the fact, that although an address comparison is unsigned, then shift and other arithmetic are signed, forcing some program acrobatics. One may argue that this is not a "bug" but "feature", since c99 does not have unsigned types, and I am just cheating. One way, or another, I have to cheat to achieve program goals. This is just small-C.

To sum up. The program, as is, is for sure not "kludge attack" proof. You may have fun trying to find interesting ways to crash. But is undoubtily usable and you may also try to improve it, by making it more robust and adding features. The most obvious missing one - output redirection to file or printer. But then you may have problems fitting such improved version in our newsletter. Though who knows.

Have a good time with c99.

Appendix for Basic Programmers Learning C

Abacus Software, Inc. (P.O. Box 7219, Grand Rapids, MI 49510) published a book "ST Basic to C". The book is meant for those people who feel comfortable in BASIC and would like to learn C. It presents codes, which implement the same ideas, written both in BASIC and C and goes into detailed comparisons. It also discusses common mistakes of BASIC programmers when learning C. As far as a "Computer Shopper" reviewer, from where I glimpsed that information, can tell, a presentation of C is correct (this is not so obvious as you may think).

There are two problems, though. On a BASIC side the book uses ST BASIC, for Atari ST series of computers. This is close enough to Microsoft‡ BASIC, that it should not really cause a trouble. Differences with TI-BASIC obviously exist, but if you are a BASIC programmer... On the other hand, a C side presents a full C compiler (Alcyon C), which is not exactly c99. Still not further from it then anything else in this department. Any volunteers to write a similar tutorial for an old orphan?

As you know me, I would not touch anything with BASIC in a title with a ten foot pole (just kiddin'). But if you are interested - check around with your friendly bookstore, or Atari dealer, or mail order. Suggested retail price is $19.95 (US - of course).

‡1 Microsoft is a trademark of Microsoft and who knows whomever else.

```
‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡
‡                       ‡
‡     Robot Rampage     ‡
‡                       ‡
‡      by Jim Beck      ‡
‡                       ‡
‡     Written in C99    ‡
‡                       ‡
‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡‡
```

All the robots in the city have gone on a rampage! They have stolen crystals used by the power plant to make electricity. Without the crystals, no power! Your job is to recover the power crystals from the robots and escape without getting splatted.

There are four sectors of the city to cover. Each one has a different and faster type of robot in it. There are five power crystals in each sector to pick up. When all five have been gathered, a door appears at the top left of the screen. Use it to escape to the next sector.

The robots are fast, but not exceptionly bright. To destroy them, you can fool them into crashing into each other. After two robots collide, a small pile of wreckage is left. This is full of expensive electronic equipment and can be picked up for points.

Because you would not stand a chance on foot, you must use your car to move around. You steer with the joystick. If you manage to clear all four sectors, you go back to the first and the game speeds up. Don't get stepped on by a robot!

GOOD LUCK!

PROGRAMMING TIPS

by Jim Beck

These 15 basic steps will help you to become a better programer.

1) Never comment your code: it only helps people to find redundancys in it.

2) Remember: If it aint a game, it aint worth programming. (Unless someone pays you.)

3) Never work on a program for more than four hours. If it takes that long, save it on a disk, place it in your disk box (preferably at the back) and forget about it. Plan on finishing it between now and the death of your great grandchildren.

4) Never use structured programming techniques. All programs should look sloppy, confusing, and completely

unreadable. When programming in BASIC, every third statement should be a GOTO. This, when others look at your listings, will make you look like a genius. (Which, in most cases, is a sharp contrast to reality.)

5) Never organize your disks. Remember: A tidy disk box is the sign of an exceptionally weak mind.

6) Debugging is to programs as decaffeinating is to coffee! Never keep your source code after you've compiled it. Only an idiot puts out later "bug-free" versions.

7) Always use global variables. Programmers have enough problems without passing parameters.

8) Never make flowcharts or pseudocode. A program should come from the top (or in some cases the bottom) of your head. It should never do what was planned for if and should seldom, if ever, do what it does completely correctly.

9) Try not to use descriptive variable names. In languages that require you to predefine your variables, predefine all posible combinations of letters and save it as a template for all your programs. This allows you to use whatever variables you want without having to go back and insert.

10) Never spill Coke on your keyboard more than once a week.

11) Only use disks as coasters under cold drinks.

12) Always have an excuse if things don't work properly. Some good examples are: "It's not supposed to do that!", "The instructions were unclear/unreadable/completely wrong.", and, of course, "This computer is broke!"

XX) Never use the number 13 in your programs: it's unlucky. Define it as a constant, such as "XX".

14) Do not hit your computer with objects like wooden baseball bats. Use aluminum ones: you'll have less problems with static.

15) Never, but NEVER read the instructions. If it does not work, plug it in, fix it, or bang it against a wall.

P.S. By the way, if your computing teacher says anything contrary to these basic rules, calmly point it out to them. If they still disagree, then ask them why they make their living teaching instead of programming, if they're so smart!

USER GROUP NEWS

by: Bob Pass

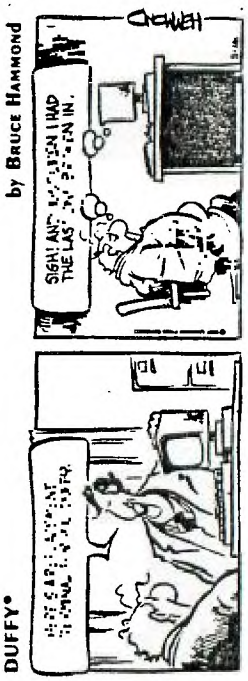
As most of you are aware by now, I will be turning over the editor's desk in June to fresher scribes. John Harbour has graciously volunteered his services and I think that he will make a fine editor and a better speller than I. However, this newsletter is no longer a one man job; John will need one or two assistants or co-editors to get the newsletter out each month. If you are interested, see either John or myself at the next meeting or give one of us a call on the phone.

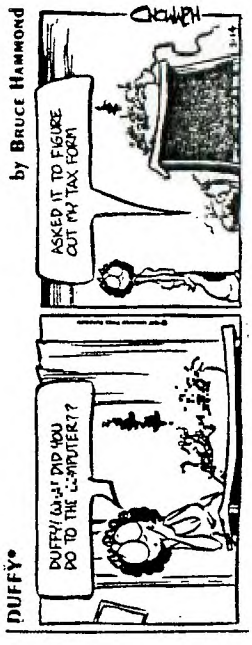
Speaking of things changing in June, Tom hall announced at the March meeting that he will not seek re-election to the presidential suite due to prior commitments. Be reminded that our annual election will take place during the June general meeting. Enclosed with this newsletter is a nomination form you can use to nominate members to stand for any of the four executive offices. Please take the time to fill out the form and bring it to the next meeting or mail it in ASAP. We need your input NOW so we know who to sweet talk!

FOR SALE

A complete TI-99/4A system for sale. Beige console, silver console for parts, PE Box, TI disk controller, SSSD Shugart disk drive, 32 K memory expansion, RS 232, Zenith monochrome monitor (green), Editor Assembler, Extended Basic, TI-Writer, Adventure, Mini-Memory, assorted software, all manuals. All in excellent condition. Asking $650.00 O.B.O. Call Paul Helwig; (403) 432-0613.

```
/* **** CPU memory browser ****

   Michal Jaegermann. Edmonton, Alberta, CANADA
   28 Feb 1987
*/

/* Copyright 1987, Michal Jaegermann.
   This code is released for a personal, non-commercial use
   You are free to give away this code, but not to sell it, as long as this
   and a previous statement will remain intact.  If you make substantial
   revisions, explain them and sign it with YOUR name.
*/

/* Usage: will be obvious, once you will run the program.
   Any key, but '<', on the first two promts will cause an acceptance
   of defaults.  keys active - G (end of range), g (beginning of range),
   e, x, u, b, f, all others (nearly) - try them.
   Apart of g and G they are case insensitive.
   After you type > the program expects a hexadecimal address.
   Two decimal digits 'nn' will send you, roughly, nn percent
   into the range.
   This program does not have any safeguards against walking into
   "forbidden" areas of memory.  Crash on your own risk!
*/


#define VOLUME   92    /* 23 lines * 4 locations per line */
#define VOLUM2   184   /* 2*VOLUME */

extern scanf ();

char lbuffer [40];
char dbuff [] = "    "; /* 4 blanks */
/* these buffers are global, so they will accessible
   to formatting functions */

main()
{
    int  *dmpstart, *dmpend, *dump;
    int  key, line, up, temp;
    char *cp;

    /* initialize line buffer and default values */
    lbuffer [0]  = ' ';
    lbuffer [1]  = '>';
    lbuffer [6]  = ':';
    for (cp=&lbuffer[7]; cp<&lbuffer[38]; cp++)
        *cp = ' ';
    *cp++ = '\n';
    *cp = 0;
    dmpend = dmpstart = 0;

    while (poll (0)) ;  /* wait for key up */
    puts ("\n start address ");
    if ('>' == (key = getchar()))
        scanf ("%x", &dmpstart);
    puts ("\n   end address ");
    if ('>' == (key = getchar()))
        scanf ("%x", &dmpend);
    dmpstart = dmpstart & (-2); /* ensure even address */
    dmpend   = dmpend   & (-2);
    dump = dmpstart;
    dmpend = dmpend - VOLUME; /* get the last screen in bounds */

    cp = dbuff + 1;  /* pointer to dbuff [1] */
    for (;;)         /* forever */
    {
       locate (1,1);

       for (line = 0; line < 23; line++)
       {
```





```
            showline (dump);
            dump = dump + 4;
        }
        puts (":    ");
        locate (24, 13);
        tohexs (dmpstart, cp);
        puts (dbuff);
        puts ("     ");
        tohexs (dmpend + VOLUME - 1, cp); /* the last location to show     */
        puts (dbuff);
        locate (24,3);
        while (0 == (key = poll (0)))     /* wait for key to be pressed    */
            ;                             /* do not want an automatic echo */
        if ((key > '@') & (key != 'G')) key = key | 32; /* to lower case   */

        if (key == 'q') exit (0);                       /* done!!          */

        if ('0'<= key) & (key <= '9'))                  /* if digit...     */
        {
            temp = 10 * (key - '0');
            putchar (key);
            key  = 0;                         /* set flag for case */
        }
        up = 0;
        switch (key)
        {
            case 'g' : dump = dmpstart;         break;
            case 'G' : dump = dmpend;           break;
            case 'u' : { dump = dump - VOLUME;
                         up++; }                break; /* set flag for check */
            case 'b' :
            case 'e' : { dump = dump - 140;
                         up++; }                break; /* back 12 lines */
            case 'f' :
            case 'x' : dump = dump - 44;        break; /* 12 lines forward */
            case '>' : { putchar (key); scanf ("%x", &dump);}    break;
            case 0   : { key = getchar();
                         if ('0'< key) & (key <= '9'))
                             temp = temp + key - '0';
                         dump = percnt (temp, dmpstart, dmpend);
                       } break;
        /* all other keys do not change anything */
        }
        if ((dump - dmpstart) > (dmpend - dmpstart))  /* out of range? */
            if (up)
                dump = dmpstart;
            else
                dump = dmpend;
    }
}

showline (from)
int *from ; /* pointer to a memory location to be shown */
{
    int  i;
    char ch, *cptr;

    tohexs (from, &lbuffer [2]);
    cptr = from;

    /* prepare display in hex words */
    for (i=8; i<28; i=i+5)
        tohexs (*from++, &lbuffer [i]);

    /* now try the same in ASCII */
    for (i=30; i<38; i++)
    {
        ch = *cptr++;
        if (ch < ' ' | ch > '~') ch = '~';   /* non-printable? */
        lbuffer [i] = ch;
    }
```

```
    /* get the whole bundle to screen */
    puts (lbuffer);
}

tohexs (num, pos)
int num;
char *pos;
{
    char ch;
    int i;

    pos = pos + 4;
    for (i=3; i>=0; i--)
    {
        ch = num & 15;          /* extract one byte    */
        if (ch > 9) ch = ch + 7; /* 7 = 'A' - ('9' + 1) */
        *(--pos) = ch + '0';
        num = num >> 4;         /* shift by one byte   */
    }
}

percnt (t, a, b) /* returns number t% in range between a and b */
int t;           /* !! Roughly !! */
char *a, *b;     /* this is a kludge to force an unsigned arithmetic */
{
    if (b > a)
    {
        /* program around compiler bugs */
        b = ((b - a) >> 1) & 32767;     /* clear the highest bit */
        b = ((b + 25) / 50) * t;
        return (a + b & (-2));
    }
    else
        return (a);
}
```

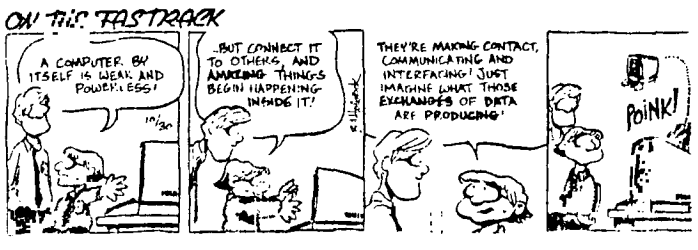
Robot Rampage by Jim Beck

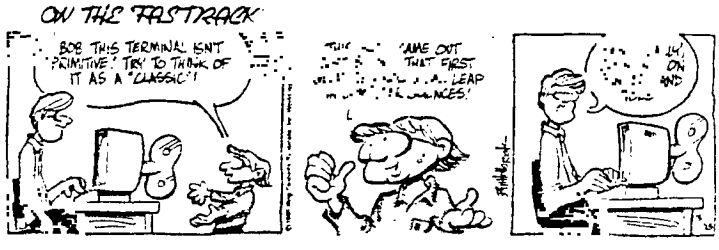
```
#include "dsk1.grflrf";
#include "dsk1.soundlib";
int rp,cp,bdd,r,or,oc,c,x,y,fr,a,b,dir,num;
int t1[17],t2[5],i,t,rb[14],cb[14],scr;
int sr,sc,hscr,del,lev;
main()
{
grfl();
screen(11);
clear();
chrdef(128,"FF878E9CBBF7ECD8D8CCC6C6C6C6CCD8D8ECF7BB9C8E87FF");
chrdef(131,"FFE171391D8FC77F0000000000000007FC78F1D3971E1FF");
chrdef(136,"FFFFFFBFDFAFD7ABD5AAD4A8D0A0C0FFFFFFFFFDFBF5EBD5");
chrdef(139,"AB552B150B0503FF");
chrdef(140,"D5ABD5ABD5ABD5ABFFFFFFFFFFFFFFFFFFFF000000000000FF");
chrdef(144,"8040A050A854AA55AA55AA55AA55AA55AA55");
chrdef(152,"0000241[illegible]40000C42A59999A5423C");
chrdef(160,"ADB5A[illegible]ADB5");
chrdef(170,"03300[illegible]1A9084");
chrdef(176,"3C7E7E3C24667E240066FFF2F2FF6600247E66243C7E7E3C");
chrdef(179,"0066FF4F4FFF6600");
chrdef(184,"00183C66663C1800060F9FFFFF9F0F06");
color(16,13,15);
color(17,2,10);
color(18,5,11);
color(19,5,15);
color(20,14,8);
color(21,6,15);
color(22,2,15);
color(23,16,15);
hscr=0;
t2[1]=1109; t2[2]=932; t2[3]=831;
t2[4]=784;
t1[1]=175; t1[2]=175; t1[3]=175;
t1[4]=175; t1[5]=208; t1[6]=208;
t1[7]=208; t1[8]=175; t1[9]=30000;
t1[10]=175; t1[11]=233; t1[12]=233;
t1[13]=175; t1[14]=175; t1[15]=156;
t1[16]=156;
title();
while(del!=2000)
{
clear();
lev=1;
setup();
scl();
num=0;
del=1000;
bdd=168;
or=0;
oc=0;
dir=179;
scr=0;
x=0;
sr=2;
sc=20;
wait();
while(x!=10)
 {
 game();
 t=0;
 while(t<del) t=t+1;
 }
hchar(r,c,170,1);
soundn(40,-7,0);
locate(11,3);
puts("          GAME OVER          ");
if (scr>hscr) hscr=scr;
locate(14,3);
puts("  Your Score:                ");
sr=14;
sc=17;
```

```
score();
scr=hscr;
locate(17,3);
puts("  High Score:              ");
sr=17;
score();
locate(20,3);
puts("Press FIRE button to restart");
wait();
}
}
setup()
{
 hchar(1,1,145,8);
 hchar(2,1,145,9);
 hchar(3,1,145,10);
 hchar(4,1,145,30);
 vchar(5,1,145,20);
 vchar(5,2,145,20);
 hchar(24,1,145,32);
 vchar(5,31,145,20);
 vchar(6,32,145,19);
 hchar(1,9,144,1);
 hchar(2,10,144,1);
 hchar(3,11,144,1);
 hchar(4,31,144,1);
 hchar(5,32,144,1);
 hchar(1,3,128,1);
 hchar(2,3,129,1);
 hchar(3,3,130,1);
 hchar(1,4,131,1);
 hchar(2,4,132,3);
 hchar(3,4,133,1);
 vchar(3,6,132,2);
 hchar(5,3,132,28);
 vchar(5,30,132,19);
 r=6;
 c=3;
 while(r<22)
  {
   while(c<28)
    {
     vchar(r,c,132,2);
     hchar(r+2,c,132,3);
     hchar(r,c+1,136,1);
     hchar(r,c+2,136,1);
     hchar(r+1,c+1,137,1);
     hchar(r+1,c+2,139,1);
     c=c+3;
    }
   c=3;
   r=r+3;
  }
 locate(2,14);
 puts("SCORE:00000");
 locate(1,14);
 puts("LEVEL:");
 hchar(1,20,1..+64,1);
}
sc1()
{
 chrdef(168,"00183C4E4E3C0000000000183C72723C00");
 r=2;
 c=4;
 color(21,6,15);
 hchar(r,c,179,1);
 or=0; oc=0;
 rb[6]=-1;
 cb[6]=-1;
 rb[1]=17; cb[1]=4;
 rb[2]=11; cb[2]=18;
 rb[3]=17; cb[3]=18;
 rb[4]=23; cb[4]=19;
 rb[5]=8; cb[5]=30;
 t=1;
 while(t<6)
  {
   hchar(rb[t],cb[t],168,1);
   t=t+1;
  }
 hchar(5,17,152,1);
 hchar(11,5,152,1);
 hchar(17,8,152,1);
 hchar(17,25,152,1);
 hchar(20,28,152,1);
 vchar(12,3,160,2);
 vchar(9,6,160,2);
 hchar(15,5,141,3);
 hchar(16,5,142,3);
 hchar(20,7,160,2);
 vchar(15,9,160,2);
 hchar(14,10,160,2);
 hchar(17,16,160,2);
 hchar(20,16,160,2);
 vchar(15,18,160,2);
 vchar(7,19,140,3);
 vchar(7,20,140,3);
 hchar(9,20,141,3);
 hchar(10,20,142,3);
 vchar(19,25,140,3);
 vchar(19,26,140,3);
 hchar(17,28,160,2);
 vchar(18,27,160,2);
}
sc2()
{
 r=2;
 c=4;
 or=0; oc=0;
 hchar(r,c,179,1);
 rb[9]=-1; cb[9]=-1;
 chrdef(168,"7EDBFFC3FF4247E07EDBFFC3FF42E207");
 color(21,7,15);
 rb[1]=5;  cb[1]=21;
 rb[2]=8;  cb[2]=27;
 rb[3]=11; cb[3]=30;
 rb[4]=14; cb[4]=21;
 rb[5]=16; cb[5]=9;
 rb[6]=20; cb[6]=4;
 rb[7]=23; cb[7]=15;
 rb[8]=23; cb[8]=24;
 t=1;
 while(t<9)
  {
   hchar(rb[t],cb[t],168,1);
   t=t+1;
  }
 hchar(8,29,152,1);
 hchar(14,5,152,1);
 hchar(14,16,152,1);
 hchar(17,22,152,1);
 hchar(23,16,152,1);
 hchar(18,14,141,3);
 hchar(19,14,142,3);
 hchar(21,14,141,3);
 hchar(22,14,142,3);
 vchar(7,7,140,3);
 vchar(7,8,140,3);
 vchar(16,19,140,3);
 vchar(16,20,140,3);
 hchar(12,11,141,3);
 hchar(13,12,141,1);
 hchar(14,12,153,1);
 hchar(15,12,142,1);
 hchar(16,11,142,3);
 vchar(13,10,140,3);
 hchar(14,11,140,1);
 vchar(13,14,140,3);
 hchar(14,13,140,1);
 hchar(13,11,136,1);
 hchar(13,13,138,1);
 hchar(15,11,137,1);
 hchar(15,13,139,1);
 hchar(5,16,160,2);
 hchar(8,4,160,2);
 hchar(8,10,160,2);
 hchar(11,7,160,2);
 hchar(11,22,160,2);
 hchar(11,28,160,2);
 hchar(14,22,160,2);
 hchar(17,4,160,5);
 hchar(20,7,160,2);
 vchar(6,15,160,2);
 vchar(9,21,160,2);
 vchar(15,21,160,2);
 vchar(21,18,160,2);
 vchar(21,9,160,2);
}
sc3()
{
 r=2;
 c=4;
 hchar(r,c,179,1);
 or=0; oc=0;
 rb[12]=-1; cb[12]=-1;
 rb[1]=5;   cb[1]=15;
 rb[2]=5;   cb[2]=27;
 rb[3]=12;  cb[3]=15;
 rb[4]=12;  cb[4]=21;
 rb[5]=12;  cb[5]=27;
 rb[6]=14;  cb[6]=6;
 rb[7]=17;  cb[7]=6;
 rb[8]=17;  cb[8]=18;
 rb[9]=20;  cb[9]=27;
 rb[10]=23; cb[10]=12;
 rb[11]=23; cb[11]=18;
 t=1;
 chrdef(168,"10387CD6FE28448210387CD6FE281028");
 color(21,14,15);
 while(t<12)
  {
   hchar(rb[t],cb[t],168,1);
   t=t+1;
  }
 hchar(6,6,160,3);
 vchar(7,8,160,3);
 hchar(10,6,160,3);
 vchar(7,4,140,3);
 vchar(7,5,140,3);
 vchar(7,6,145,3);
 vchar(7,7,145,3);
 vchar(19,4,140,3);
 vchar(19,5,140,3);
 vchar(16,16,140,3);
 vchar(16,17,140,3);
 vchar(19,25,140,3);
 vchar(19,26,140,3);
 hchar(15,11,141,3);
 hchar(16,11,142,3);
 hchar(18,17,141,3);
 hchar(19,17,142,3);
 hchar(9,20,141,3);
 hchar(10,20,142,3);
 hchar(11,6,152,1);
 hchar(23,11,152,1);
 hchar(23,30,152,1);
```

```
 hchar(11,21,152,1);
 hchar(9,18,152,1);
 hchar(5,10,160,2);
 hchar(5,25,160,2);
 hchar(11,13,160,2);
 hchar(11,19,160,2);
 hchar(11,22,160,2);
 hchar(17,7,160,2);
 hchar(23,16,160,2);
 vchar(9,15,160,2);
 vchar(12,16,160,2);
 vchar(12,30,160,2);
 vchar(15,6,160,2);
 vchar(15,3,160,2);
 vchar(15,18,160,2);
 vchar(18,24,160,2);
 vchar(18,27,160,2);
 vchar(21,12,160,2);
}
sc4()
{
 r=2;
 c=4;
 or=0; oc=0;
 hchar(r,c,179,1);
 rb[9]=-1; cb[9]=-1;
 chrdef(168,"3C5A7ED5B1B1AB7E3C5A7ED5AB7E0000");
 color(21,3,15);
 rb[1]=5;    cb[1]=18;
 rb[2]=8;    cb[2]=30;
 rb[3]=11;   cb[3]=4;
 rb[4]=14;   cb[4]=12;
 rb[5]=14;   cb[5]=23;
 rb[6]=17;   cb[6]=28;
 rb[7]=20;   cb[7]=27;
 rb[8]=23;   cb[8]=27;
 t=1;
 while(t<9)
  { hchar(rb[t],cb[t],168,1);
    t=t+1;
  }
 hchar(14,30,184,1);
 hchar(14,22,185,1);
 hchar(14,24,152,1);
 hchar(11,5,152,2);
 hchar(9,18,152,1);
 hchar(23,21,152,1);
 t=19;
 while(t<31)
  {
   vchar(5,t,145,3);
   t=t+1;
  }
 hchar(6,16,132,2);
 hchar(7,16,132,2);
 vchar(13,19,140,3);
 vchar(13,20,140,3);
 vchar(19,25,140,3);
 vchar(19,26,140,3);
 hchar(9,14,141,3);
 hchar(10,14,142,3);
 hchar(18,8,141,6);
 hchar(19,8,142,6);
 hchar(15,26,141,3);
 hchar(16,26,142,3);
 hchar(5,4,160,2);
 hchar(8,4,160,2);
 hchar(14,4,160,2);
 hchar(17,4,160,2);
 hchar(20,4,160,2);
 hchar(14,7,160,2);
 hchar(5,10,160,2);
 hchar(23,10,160,2);
 hchar(17,16,160,2);
 hchar(11,22,160,2);
 hchar(14,25,160,2);
 vchar(6,9,160,2);
 vchar(12,15,160,2);
 vchar(12,12,160,2);
 vchar(12,24,160,2);
 vchar(12,30,160,2);
 vchar(15,24,160,2);
 vchar(18,18,160,2);
}
game()
{
 a=joyst(1,&x,&y);
 if (y!=0) x=0;
 if (a)
   if ((gchar(r-(y/4),c+(x/4))==132) |
       (gchar(r-(y/4),c+(x/4))==152) |
       (gchar(r-(y/4),c+(x/4))==170))
    {
     or=-y/4;
     oc=x/4;
     if (x==-4) dir=177;
     if (x==4)  dir=179;
     if (y==-4) dir=178;
     if (y==4)  dir=176;
    }
 if (gchar(r+or,c+oc)!=132)
  if (gchar(r+or,c+oc)==152)
   {
    num=num+1;
    if (num==5)
      {
       num=0;
       hchar(2,4,153,1);
      }
    scr=scr+200;
    sound3(10,262,0,260,0,263,0);
    score();
   }
  else
  if (gchar(r+or,c+oc)==184)
   {
    hchar(14,22,132,1);
    sound3(10,330,0,332,0,329,0);
   }
  else
  if (gchar(r+or,c+oc)==170)
   {
    scr=scr+100;
    sound1(1,262,0);
    sound1(1,330,0);
    sound1(1,392,0);
    score();
   }
  else
  if (gchar(r+or,c+oc)==153)
   next();
  else
   {
    or=0;
    oc=0;
   }
 hchar(r,c,132,1);
 r=r+or;
 c=c+oc;
 hchar(r,c,dir,1);
 badguy();
}
 badguy()
{
 t=1;
 while(rb[t]!=-1)
  {
  if (rb[t]>0)
  {
  rp=0;
  cp=0;
  if (c>cb[t]) cp=1;
  if (c<cb[t]) cp=-1;
  a=gchar(rb[t],cb[t]+cp);
  if (cp!=0)
  {
  if ((a==168) | (a==169))
   {
   hchar(rb[t],cb[t],132,1);
   hchar(rb[t],cb[t]+cp,170,1);
   soundn(2,5,0);
   soundn(2,6,2);
   soundn(2,7,4);
   scr=scr+250;
   score();
   }
  if (a>175)
   if (a<180)
    {
    x=10;
    a=132;
    }
  }
  if (a!=132)
   {
    cp=0;
    if (r>rb[t]) rp=1;
    if (r<rb[t]) rp=-1;
    a=gchar(rb[t]+rp,cb[t]);
    if (rp!=0)
    if ((a==168) | (a==169))
     {
     hchar(rb[t],cb[t],132,1);
     hchar(rb[t]+rp,cb[t],170,1);
     soundn(2,5,0);
     soundn(2,6,2);
     soundn(2,7,4);
     scr=scr+250;
     score();
     }
    if (a!=132)
     {
     if (rp!=0)
     if (a>175)
      if (a<180)
       x=10;
     if (x!=10)
      rp=0;
     }
   }
   if ((gchar(rb[t],cb[t])==132) | (gchar(rb[t],cb[t])==170))
    rb[t]=0;
   else
    {
     hchar(rb[t],cb[t],132,1);
     rb[t]=rb[t]+rp;
     cb[t]=cb[t]+cp;
     hchar(rb[t],cb[t],bdd,1);
    }
    }
   t=t+1;
   }
   bdd=bdd+1;
   if (bdd==170) bdd=168;
}
score()
```

```
{
 int o;
 o=scr;
 if (o>30000) { scr=0; o=0; }
 hchar(sr,sc,(o/10000)+48,1);
 o=o-((o/10000)000);
 hchar(sr,sc+1,(o/1000)+48,1);
 o=o-((o/1000)00);
 hchar(sr,sc+2,(o/100)+48,1);
 o=o-((o/100)0);
 hchar(sr,sc+3,(o/10)+48,1);
 o=o-((o/10));
 hchar(sr,sc+4,o+48,1);
}
wait()
{
 locate(3,14);
 if (x!=10) puts("PRESS  FIRE");
 t=0;
 z=1;
 while(key(1,&a)==0)
  {
   t=t+1;
   if (t>16)
    {
     t=1;
     z=z+1;
     if (z>4) z=1;
    }
   sound2(4,t1[t],3,t2[z],3);
   color(22,16,15);
   color(22,2,15);
  }
 hchar(3,14,32,11);
}
next()
{
 sound1(20,262,0);
 sound2(20,262,0,330,0);
 sound3(40,262,0,330,0,392,0);
 scr=scr+500;
 clear();
 del=del-100;
 lev=lev+1;
 setup();
 if (lev==2) sc2();
 if (lev==3) sc4();
 if (lev==4) sc3();
 if (lev==5) { lev=1;
               locate(14,12);
               puts("  WAY TO GO!  ");
               sound3(20,131,0,130,0,132,0);
               sound3(30,175,0,174,0,176,0);
               sound3(10,131,0,130,0,132,0);
               sound3(20,175,0,174,0,176,0);
               sound3(60,208,0,207,0,210,0);
               setup();
               sc1();
             }
 score();
 wait();
}
title()
{
 t=1;
 while(t<11)
 {
  hchar(t,1,132,32);
  t=t+1;
 }
 hchar(4,8,32,16);
 hchar(5,8,32,16);
 hchar(6,8,32,16);
 locate(5,9);
 puts("ROBOT  RAMPAGE");
 locate(11,10);
 puts("By  Jim Beck");
 locate(13,9);
 puts("Written in C99");
 t=15;
 while(t<25)
  {
   hchar(t,1,145,32);
   t=t+1;
  }
 locate(16,11);
 puts("  Fairware  ");
 locate(23,2);
 puts("  Press FIRE button to begin. ");
 x=10;
 hchar(15,32,144,1);
 hchar(14,1,145,1);
 hchar(14,2,144,1);
 hchar(13,1,144,1);
 hchar(2,7,141,18);
 hchar(3,8,141,16);
 hchar(7,8,142,16);
 hchar(8,7,142,18);
 vchar(3,6,140,5);
 vchar(4,7,140,3);
 vchar(4,24,140,3);
 vchar(3,25,140,5);
 hchar(2,6,136,1);
 hchar(3,7,136,1);
 hchar(2,25,138,1);
 hchar(3,24,138,1);
 hchar(7,7,137,1);
 hchar(8,6,137,1);
 hchar(8,25,139,1);
 hchar(7,24,139,1);
 wait();
 clear();
 locate(1,9);
 puts("ROBOT  RAMPAGE");
 puts("\n\n\n");
 puts("\n\n    If you enjoy this program,");
 puts("\n\n  please send $5 (no monopoly ");
 puts("\n\n  money please) to: ");
 puts("\n\n\n                    Jim Beck");
 puts("\n\n               10947 36 Ave. ");
 puts("\n\n             Edmonton,Alberta");
 puts("\n\n              Canada  T6J 0B9");
 puts("\n\n\n            Thank you.");
 locate(24,1);
 puts("   Press FIRE button to begin.");
 wait();
}
```





Computers soon in farm tractor cab

REGINA (CP) — Ron Palmer said it was never his goal to take the farmer out of the tractor cab.

But the University of Regina professor is applying the finishing touches to electronic machinery designed to navigate farm equipment around fields.

"How quickly we move depends on how the idea is accepted by farmers," he said. "We have a good groundswell of farmers behind us."

Palmer said he tested the navigation system on his father's tractor last weekend. It successfully steered the tractor which was pulling a cultivator.

A fully automatic system may not be on the market until the 1990s but Palmer's company, Accutrak Systems Ltd., plans to start marketing less sophisticated navigation units within a year.

Six to 12 units will be field tested on Saskatchewan farms next spring.

They will be designed to give farmers substantial savings on pesticides, fertilizers and fuel.

Palmer started work on his tractor navigation system three years ago, but first studied ways of making field work more efficient.

'. . . Either we have to close up shop and say no more agriculture on the Prairies or we have to be more efficient.'

He found one of the biggest wastes involves covering the same ground twice. His studies found an average overlap of 9.5 per cent as farmers circled their fields.

One of the problem activities is the application of pesticides, which leave no visible mark to guide a farmer making a second pass. Palmer said his development will take the guesswork out of such operations and help keep Canadian farmers competitive in a world market.

"If we can't grow wheat for $4.50 a bushel, either we have to close up shop and say no more agriculture on the Prairies or we have to be more efficient," he said.

The system requires two beacons set up on the edge of the field. The tractor uses a receiver and a computer.

After measuring the distance between beacons, the farmer types this information into the computer, along with the width of the equipment being used.

The computer then pinpoints the tractor's location relative to the beacons as the farmer works the field, and tells him which way to steer in order to stay parallel to — and a precise distance from — the first run.

There are possible elaborations of this system, the most expensive one being installation of steering controls operated by the computer.

"I'm not saying we are going to get rid of farmers from tractors," Palmer said.

"All I'm saying is it opens the door to all these operations. Instead of farmers being laborers sitting behind the wheel of a tractor, I see them being more managers."