```
/*-
  worms.cpp -- A primitive game example of wiggly worms moving about

  (c) 2013 pmateti@wright.edu

  Build it: 'g++ -o worms worms.cpp -lncurses'
  Run   it: './worms 3'
*/

/* TBD
   signals ^C
*/

#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <time.h>

#define random(x)       (rand() % x)
#define isDigit(x)      ('0' <= x && x <= '9')
#define min(a, b)       ((a) < (b)? (a) : (b))
#define ESC             '\033'    // the ESCape char
#define CARROT          '.'       // a carrot is shown as a dot

const char * say[] = {          // worms carry these strings
  "do-not-optimize-too-soon*",
    "If it does not have to be correct, making the program efficient is easy.",
    "90% of code executes only 10% of the time!",
    "Is there a program, longer than say 1000 lines, that is correct?",
    "OS is not bug-free, the compiler is not bug-free, so is my program!",
    "ABCDEFGHIJKLMNOPQRSTUVWXYZ",
    "98765432109876543210*",
    "Edsger-Dijkstra*",             // legendary programmers ...
    "C-A-R-Hoare*",
    "Donald-Knuth*",
    "Richard-Stallman*",
    "Linux-Torvalds*",
};
#define Nsay sizeof(say)/sizeof(char *) // #sayings we have

typedef struct {
  int onec;                     // one character
  int attr;                     // its color
} ASOG;                         // A Square On the Ground

typedef enum {
  NORTH, NORTHEAST, EAST, SE, S, SW, W, NW
} DIRECTION;                    // as numbers: 0 .. 7

typedef enum {
  VEGETARIAN, CANNIBAL, SCISSORHEAD
} WORM_KIND;                    // 0, 1, 2

typedef struct {
  int color;
  int capacity;                 // stomach size
  int foodValue;
} WORM_INFO;

typedef enum {
  EATEN, DEAD, ALIVE
} LIFE;

typedef struct {
  int x, y;                     // coordinates of the segment
  char c;                       // the letter it carries
} SEGMENT;
```

```
68
69  #define MAXsegs  100            // max # segments per worm
70  typedef struct {
71    WORM_KIND type;               // once set, type does not change
72    int direction;                // its (head's) direction
73    int nsegs;                    // # of segs in this worm
74    int stomach;                  // food value
75    LIFE status;                  // EATEN, DEAD, or ALIVE
76    SEGMENT body[MAXsegs];        // body parts
77  } WORM;
78  #define headSeg(wp)     (wp->body + wp->nsegs)
79  #define xOf(wp)         (headSeg(wp)->x)
80  #define yOf(wp)         (headSeg(wp)->y)
81
82  #define MAXworms 100            // max # worms in our program
83  WORM worm[MAXworms];            // vars for the worms
84  int hxWorms;                    // high water mark of the worm[]
85  int xworms[3];                  // counts of different worms
86  #define nVegetarians    xworms[0]
87  #define nCannibals      xworms[1]
88  #define nScissors       xworms[2]
89  #define nAllWorms       (nVegetarians + nCannibals + nScissors)
90
91  #define MAXrow 100
92  #define MAXcol 100
93  ASOG passive[MAXrow * MAXcol];  // for carrots and dead worms
94  ASOG screen[MAXrow * MAXcol];   // what gets displayed
95  int asogRows, asogCols;         // the actual size of 'worm area'
96  #define asog(ptr, row, col) (ptr + row*asogCols + col)
97
98  #include <ncurses.h>
99  #define mvCursor(y, x) move(y, x)
100 #define putChar(c) addch(c)
101 #define putString(s) addstr(s)
102 #define getChar() getch()
103 #define screenFlush() refresh()
104 #define screenClear() clear()
105 #define EOLN "\n"               // End Of LiNe string
106
107 void endCurses()
108 {
109   if (!isendwin())
110     endwin();
111 }
112
113 void startCurses()
114 {
115   initscr();                    // ncurses init
116   cbreak();                     // unbuffered getChar
117   noecho();                     // no echoing of keys pressed
118   //  intrflush(stdscr, 0); // TBD
119   nodelay(stdscr, TRUE);        // get a char *if* available
120   atexit(endCurses);
121   start_color();
122   use_default_colors();
123   init_pair(1, COLOR_RED, -1);
124   init_pair(2, COLOR_GREEN, -1);
125   init_pair(3, COLOR_BLUE, -1);
126   getmaxyx(stdscr, asogRows, asogCols);
127   if (asogCols > MAXcol) asogCols = MAXcol;
128   if (asogRows > MAXrow) asogRows = MAXrow;
129   asogRows -= 6;          // 6 lines for msgs, asogRows needs to be > 0
130 }
131
132 int getOneChar()
133 {
134   nodelay(stdscr, FALSE);       // wait until a char is typed
```

```
  int c =  getChar();
  nodelay(stdscr, TRUE);
  return c;
}

// ncurses code ends here

void showScreen()
{
  ASOG * sp = screen;
  for (int y = 0; y < asogRows; y++) {
    mvCursor(y, 0);
    for (int x = 0; x < asogCols; x++, sp++)
      putChar(sp->onec | sp->attr);
  }
  screenFlush();
}

char msg[1024];

void showMsg(int y)
{
  msg[1023] = '\0';
  mvCursor(y, 0);
  putString(msg);
  screenFlush();
}

void sprinkleCarrots()
{
  for(ASOG *p = asog(passive, asogRows, 1); p-- > passive;) {
    p->attr = WA_DIM;
    p->onec = CARROT;
  }
}

WORM_INFO worm_info[] = {       // indexed by WORM_KIND
  {COLOR_PAIR(1), 3, 3},        // from <ncurses.h>
  {COLOR_PAIR(2), 4, 5},
  {COLOR_PAIR(3), 5, 4}
};

/* Show a single worm.  pre: wp != 0.  post: It is drawn on passive[]
   if it is dead.  If alive, it is drawn on screen[].  If eaten, not
   drawn. ; */

void showOneWorm(WORM * wp)
{
  if (wp->status != EATEN) {
    ASOG *f = (wp->status == ALIVE ? screen : passive);
    SEGMENT *bp, *hp = headSeg(wp);
    for (bp = wp->body + 1; bp <= hp; bp++) {
      ASOG *g = asog(f, bp->y, bp->x);
      g->attr = worm_info[wp->type].color;
      g->onec = bp->c;
    }
  }
}

/* Display the carrots, the dead worms, and the current positions of
   one (pre: wp != 0) or all (pre: wp == 0) worms.  globals: passive[]
   read-only.  pre: none; post: ... ; */

void showWormsAndCarrots(WORM * wp)
{
  ASOG *sp = screen, *se = asog(screen, asogRows, 1);
  for (ASOG *pp = passive; sp < se; )
```

```
202       *sp++ = *pp++;              // copy the carrots + dead worms
203     WORM * headWorm = worm + (wp? 1 : hxWorms);
204     for (WORM * wp = worm; wp < headWorm; )
205       showOneWorm(wp++);
206     showScreen();
207   }
208
209   /* It is hungry if its tummy is at least 25% empty.  pre: wp != 0;
210      post: Return 1 if worm wp is hungry; otherwise return 0.  */
211
212   int isHungry(WORM * wp)
213   {
214     int m = wp->stomach;
215     int n = wp->nsegs * worm_info[wp->type].capacity;
216     return (wp->status == ALIVE && 4*m < 3*n);
217   }
218
219   /* pre: wp != 0 */
220
221   void gotEaten(WORM * wp)
222   {
223     if (wp->status == ALIVE) {
224       wp->status = EATEN;
225       xworms[wp->type] --;
226     }
227   }
228
229   /* pre: wp->status == ALIVE; post: if stomach is 0 or less, mark it
230      dead, and send its body to the cemetary. */
231
232   void checkStomach(WORM * wp)
233   {
234     if (wp->stomach <= 0) {
235       wp->status = DEAD;
236       xworms[wp->type] --;
237       showOneWorm(wp);            // mv the body to passive
238     }
239   }
240
241   /* Eat a carrot, if available.  pre: wp->status == ALIVE; post:
242      ...; */
243
244   void eatACarrot(WORM * wp)
245   {
246     ASOG *sg = asog(passive, yOf(wp), xOf(wp));
247     if (CARROT == sg->onec) {
248       wp->stomach += 2;           // food value of one carrot
249       sg->onec = ' ';
250     }
251   }
252
253   /* Check if any segment of worm wp is at (col, row).  Return this
254      segment's number.  Recall that body[0] is just an eraser-blank, not
255      really a part of the worm.  pre: wp != 0; post: ...; */
256
257   int isAt(WORM * wp, int row, int col)
258   {
259     SEGMENT *bp = headSeg(wp);
260     for (int sn = wp->nsegs; sn > 0; sn--, bp--)
261       if ((bp->y == row) && (bp->x == col))
262         return sn;
263   }
264
265   /* The following two vars should be 'out' parameters (var params), but
266      for ease of understanding, let us declare these as globals. */
267
268   WORM *victimWormp;
```

```
int victimSegNum;

/* Find a worm that is on the same square as "my" head.  pre: me != 0;
   post: victimSegNum > 0 => victimWormp != 0 and victimWormp's
   victimSegNum segment is at the same square as the head of me.  */

void setVictimWorm(WORM * me)
{
  WORM *headWorm = worm + hxWorms, *wp;
  victimWormp = 0;
  victimSegNum = 0;
  for (wp = worm; wp < headWorm; wp++) {
    if (wp != me && wp->status == ALIVE) {
      victimSegNum = isAt(wp, yOf(me), xOf(me));
      if (victimSegNum > 0) {
        victimWormp = wp;
        return;                 // found the victim
      }
    }
  }
}

/* We have a new worm about to join.  Find a slot for it in our worm[]
   array.  pre: 0 <= hxWorms <= MAXworms; post: Return a ptr to a slot
   for the new work, or 0, if we do not succeed in finding a slot.  */

WORM * findSlot()
{
  WORM * zp = worm + hxWorms;
  if (hxWorms < MAXworms) {
    hxWorms++;
    return zp;
  }
  /* see if there are any dead/eaten worms */
  for (WORM * wp = worm; wp < zp; wp++)
    if (wp->status != ALIVE)
      return wp;
  return 0;
}

/* Copy the body segments.  pre: vp != 0 && 0 < vsn <= vp->nsegs;
   post: old body[vsn+1..end] should become new body[1..new-length];
   body[0] remains as the eraser-blank */

void shiftBodyDown(WORM * vp, int vsn)
{
  SEGMENT *dp = vp->body + 1;       // dp destination, sp source
  SEGMENT *hp = headSeg(vp);
  for (SEGMENT *sp = vp->body + vsn + 1; sp <= hp; )
    *dp++ = *sp++;
  int tsegs = vp->nsegs;
  vp->nsegs -= vsn;
  vp->stomach = vp->stomach / tsegs * vp->nsegs;
  checkStomach(vp);
}

/* A scissor-head or cannibal hit the victim at segment number vsn.
   Slice the victim into two.  pre: vp != 0 && 0 < vsn <= vp->nsegs;
   post: The first one with vp->body[1..vsn-1] and the second one with
   vp->body[vsn+1..vp->nsegs] as its new body.  */

WORM * sliceTheVictim(WORM * vp, int vsn)
{
  int tsegs = vp->nsegs;
  WORM *wp = findSlot();
  if (wp == 0)                  // could not find a slot
    return vp;
```

```
336
337   xworms[vp->type]++;
338
339   *wp = *vp;                    // bottom-half
340   wp->nsegs = vsn - 1;
341   wp->stomach = vp->stomach / tsegs * wp->nsegs;
342   checkStomach(wp);
343
344   shiftBodyDown(vp, vsn);       // the top-half
345   return wp;
346 }
347
348 /* Eat whatever you find at the wp's head position.  If wp is a
349    cannibal or scissor-head it will also eat a carrot, if available.
350    Update the stomach contents.  pre: isHungry(wp) && wp->status ==
351    ALIVE; post: ...; */
352
353 void eat(WORM * wp)
354 {
355   if (wp->type != VEGETARIAN) {
356     setVictimWorm(wp);
357     if (victimWormp != 0) {
358       wp->stomach += worm_info[victimWormp->type].foodValue;
359       WORM * zp = sliceTheVictim(victimWormp, victimSegNum);
360       if (wp->type == CANNIBAL && zp->status == ALIVE) {
361         wp->stomach += zp->nsegs * worm_info[zp->type].foodValue;
362         gotEaten(zp);
363       }
364     }
365   }
366   eatACarrot(wp);
367 }
368
369 // dx/dy changes, indexed by DIRECTION; (+0 just to line up)
370 const int dxa[] = { +0, +1, +1, +1, +0, -1, -1, -1 };
371 const int dya[] = { -1, -1, +0, +1, +1, +1, +0, -1 };
372
373 const int nextTurn[16] = {      // experiment with other values
374   0, 0, 0, 0,  0, 0, 0, 0,      // think: why 16?
375   1, 1, 1, 7,  7, 7, 2, 6
376 };
377
378 /* One moment in the life of a worm: crawl by one step using one unit
379    of food, may change direction, eat if hungry, mark as dead if
380    stomach is now empty.  */
381
382 void live(WORM * wp)
383 {
384   SEGMENT *hp = headSeg(wp);
385   for (SEGMENT * bp = wp->body; bp < hp; bp++) // < not <=
386     bp->x = (bp + 1)->x, bp->y = (bp + 1)->y;  // crawl
387
388   int dir = wp->direction = (wp->direction + nextTurn[random(16)]) % 8;
389   hp->y += dya[dir];
390   hp->x += dxa[dir];
391   if (hp->y < 0) hp->y = asogRows-1;
392   if (hp->x < 0) hp->x = asogCols-1;
393   if (hp->y >= asogRows) hp->y = 0;
394   if (hp->x >= asogCols) hp->x = 0;
395
396   wp->stomach--;
397   if (isHungry(wp))
398     eat(wp);
399   checkStomach(wp);
400 }
401
402 /* Create a worm.  It should be just long enough to carry the phrase
```

```
403     sy[].  It is going to crawl out of a random hole.  pre: s != 0;
404     post: ...; */
405
406   void createWorm(WORM_KIND type, const char * sy)
407   {
408     WORM *wp = findSlot();
409     if (wp == 0)
410       return;                       // no room for a new worm
411
412     int n = wp->nsegs = min(strlen(sy), MAXsegs - 1);
413     wp->stomach = n * worm_info[type].capacity;
414     wp->direction = NORTH;
415     wp->status = ALIVE;
416     wp->type = type;
417
418     SEGMENT * bp = wp->body;
419     int yy = bp->y = random(asogRows);    // birth place of this worm
420     int xx = bp->x = random(asogCols);
421     bp->c = ' ';                          // works as an "eraser"
422
423     SEGMENT * hp = headSeg(wp);
424     for (bp++; bp <= hp; bp++) {
425       bp->c = *sy++;                // store the phrase s[]
426       bp->x = xx;                   // worm hangs down in the z-axis
427       bp->y = yy;
428     }
429     xworms[type] ++;
430   }
431
432   /* parameters for the 'graphical' (such as it is) display */
433   int slowness;                     // slowness number
434   int paused = 0;                   // == 1 iff paused
435   int idelay = 10;                  // delay increment
436   int tdelay;                       // total delay yet to do
437
438   int userKeyPress(int c)
439   {
440     switch (c) {
441     case '+': slowness -= min(slowness, 100); break;
442     case '-': slowness += 100; break;
443     case 'f': slowness = 0; break;
444     case 's': /* for you todo: highlight a worm */ break;
445     case 'k': /* for you todo: kill the highlighted worm */ break;
446     case 'w': createWorm((WORM_KIND)random(3), say[random(Nsay)]); break;
447     case ' ': paused ^= 1; break;
448     }
449     return c;
450   }
451
452   /* User can control the speed, etc.  The total delay required, tdelay,
453      is doled out in increments of idelay so that keyboard interaction
454      is more responsive.  pre: none; post: returns ESC or '\0';; */
455
456   int userControl()
457   {
458     sprintf
459       (msg,
460        "SPC %s, ESC terminates, k kills-, w creates-, s shows-a-worm" EOLN
461        "%2d Vegetarians,%2d Cannibals,%2d Scissor-heads,%2d hi-water-mark" EOLN
462        "%04d slowness, - increases, + reduces, f full-speed" EOLN,
463        (paused? "resumes" : "pauses "),
464        nVegetarians, nCannibals, nScissors, hxWorms, slowness);
465     showMsg(asogRows + 1);
466     for (tdelay = slowness+1; tdelay > 0; tdelay -= idelay) {
467       char c = userKeyPress(getChar()); // no-delay
468       if (c == ESC) return ESC;
469       if (paused) tdelay += idelay;
```

```
    if (idelay > 0)
      usleep(1000 * idelay);
  }
  return '\0';
}

int main(int argc, char * argv[])
{
  slowness = 10*(argc > 1? argv[1][0] - '0' : 1);
  if (slowness < 0) slowness = 0;
  srand(time(0));                  // pseudo-random number generator seed
  for (int ch = 0; ch != ESC; ) {
    startCurses();
    sprinkleCarrots();
    hxWorms = nVegetarians = nCannibals = nScissors = 0;
    for (int i = random(6) + 3; i > 0; i--)
      createWorm((WORM_KIND) random(3), say[random(Nsay)]);

    while (ch != ESC && nAllWorms > 0) {
      for (WORM * wp = worm; wp < worm + hxWorms; wp++) // hxWorms may +-
        if (wp->status == ALIVE)
          live(wp);
      showWormsAndCarrots(0);
      ch = userControl();
    }

    sprintf(msg, "press ESC to terminate, or any other key to re-run"
            EOLN  EOLN  EOLN  EOLN);
    showMsg(asogRows + 1);
    ch = getOneChar();
    endCurses();
  }
  return 0;
}

/* -eof- */
```