# §A6 Answers to all the exercises

World is crazier and more of it than we think,
Incorrigibly plural. I peel and portion
A tangerine and spit the pips and feel
The drunkenness of things being various.
— Louis MacNeice (1907–1963), *Snow*

## *Exercises in Chapter II*

• 1 (p. 80) Here's one way: attach a property to the mushroom which records whether or not it has been picked.

```
    mushroom_picked,
    after [;
        Take: if (self.mushroom_picked)
                  "You pick up the slowly-disintegrating mushroom.";
              self.mushroom_picked = true;
              "You pick the mushroom, neatly cleaving its thin stalk.";
        Drop: "The mushroom drops to the ground, battered slightly.";
    ],
```

Note that the mushroom is allowed to call itself `self` instead of `mushroom`, though it doesn't have to.

• 2 (p. 81) Here is one possibility. Don't forget the `light`, or the player will find only darkness at the foot of the steps and won't ever be able to read the description.

```
Object Square_Chamber "Square Chamber"
  with description
           "A sunken, gloomy stone chamber, ten yards across. A shaft
           of sunlight cuts in from the steps above, giving the chamber
           a diffuse light, but in the shadows low lintelled doorways
           lead east and south into the deeper darkness of the
           Temple.",
  has  light;
```

The map connections east, south and back up will be added in §9.

• 3 (p. 90)  The neatest way is to make the door react to movement through it:

```
Class ObligingDoor
 with name 'door',
      react_before [;
          Go: if (noun.door_dir == self.door_dir)
                  return self.open_by_myself();
          Enter: if (noun == self) return self.open_by_myself();
      ],
      open_by_myself [ ks;
          if (self has open) rfalse;
          print "(first opening ", (the) self, ")^";
          ks = keep_silent; keep_silent = true;
          <Open self>; keep_silent = ks;
          if (self hasnt open) rtrue;
      ],
  has door openable lockable static;
```

Here react_before picks up any action in the same location as the door, whether or not it directly involves the door. So if the door is the east connection out of this location, it reacts to the action Go e_obj action (because e_obj.door_dir is e_to, which is the door's door_dir as well). Another point to notice is that it reacts to Enter as well. Normally, if the player types "go through oaken door", then the action Enter oaken_door is generated, and the library checks that the door is open before generating Go e_obj. Here, of course, the whole point is that the door is closed, so we have to intercept the Enter action as early as possible. (A caveat: if door_dir values are to be routines, the above needs to be complicated slightly to call these routines and compare the return values.)

• 4 (p. 90)  The following assumes that all the keys in the game belong to a class called Key, and that there are no more than 16 of them. This allows the set of keys tried so far to be stored (as a bit vector) in a 16-bit Inform integer, and as games with many keys and many doors are exceptionally irritating, the limit is a blessing in disguise. Add the following to the ObligingDoor class:

```
has_been_unlocked,
which_keys_tried,
before [ key_to_try j bit ks;
    Open:
        if (self has locked) {
            if (self.has_been_unlocked) {
                key_to_try = self.with_key;
                if (key_to_try notin player)
                   "You have mislaid ", (the) key_to_try,
                " and so cannot unlock ", (the) self, ".";
                print "(first unlocking ";
            }
            else {
                bit=1;
                objectloop (j ofclass Key)
                {   if (self.which_keys_tried & bit == 0
                        && j in player) key_to_try = j;
                    bit = bit*2;
                }
                if (key_to_try == nothing) rfalse;
                print "(first trying to unlock ";
            }
            print (the) self, " with ", (the) key_to_try, ")^";
            ks = keep_silent; keep_silent = true;
            <Unlock self key_to_try>; keep_silent = ks;
            if (self has locked) rtrue;
        }
    Lock: if (self has open) {
            print "(first closing ", (the) self, ")^";
            ks = keep_silent; keep_silent = true;
            <Close self>; keep_silent = ks;
            if (self has open) rtrue;
        }
    Unlock:
        bit=1;
        objectloop (j ofclass Key) {
            if (second == j)
                self.which_keys_tried = self.which_keys_tried + bit;
            bit = bit*2;
        }
],
after [;
    Unlock: self.has_been_unlocked = true;
],
```

• 5 (p. 91) Briefly: provide a `GamePreRoutine` which tests to see if `second` is an object, rather than `nothing` or a number. If it is, check whether the object has a `second_before` rule (i.e. test the condition `(object provides second_before)`). If it has, send the `second_before` message to it, and return the reply as the return value from `GamePreRoutine`. If `second` isn't an object, be sure to explicitly return `false` from `GamePreRoutine`, or you might inadvertently run into the `]` at the end, have `true` returned and find lots of actions mysteriously stopped.

## *Exercises in Chapter III*

• 6 (p. 109) At the key moment, move orange_cloud to location, where the orange cloud is defined as follows:

```
Object orange_cloud "orange cloud"
  with name 'orange' 'cloud',
       react_before [;
           Look: "You can't see for the orange cloud surrounding you.";
           Go, Exit: "You wander round in circles, choking.";
           Smell: if (noun == nothing or self) "Cinnamon? No, nutmeg.";
       ],
  has  scenery;
```

• 7 (p. 110) It's essential here to return true, to prevent the library from saying "Dropped." as it otherwise would. Fortunately print_ret causes a return value of true:

```
after [;
    Drop: move noun to Square_Chamber;
        print_ret (The) noun, " slips through one of the burrows
            and is quickly lost from sight.";
],
```

(The rest of the Wormcast's definition is left until §9 below.) This is fine for 'Ruins', but in a more complicated game it's possible that other events besides a Drop might move items to the floor. If so, the Wormcast could be given an each_turn (see §20) to watch for items on the floor and slip them into the burrows.

• 8 (p. 113) The Wormcast has one orthodox exit (west) and three confused ones:

```
Object Wormcast "Wormcast"
  with description
           "A disturbed place of hollows carved like a spider's web,
           strands of empty space hanging in stone. The only burrows
           wide enough to crawl through begin by running northeast,
           south and upwards.",
       w_to Square_Chamber,
       ne_to [; return self.s_to(); ],
       u_to [; return self.s_to(); ],
       s_to [;
           print "The wormcast becomes slippery around you, as though
               your body-heat is melting long hardened resins, and
               you shut your eyes tightly as you burrow through
               darkness.^";
```

```
            if (eggsac in player) return Square_Chamber;
            return random(Square_Chamber, Corridor, Forest);
        ],
        cant_go "Though you feel certain that something lies behind
            the wormcast, this way is impossible.",
  has  light;
```

This is a tease, as it stands, and is in need of a solution to the puzzle and a reward for solving it.

• 9 (p. 113) Define four objects along the lines of:

```
CompassDirection white_obj "white wall" compass
  with name 'white' 'sac' 'wall', door_dir n_to;
```

This means there are now sixteen direction objects, some of which refer to the same actual direction: the player can type "white" or "north" with the same effect. If you would like to take away the player's ability to use the ordinary English nouns, add the following line to `Initialise`:

```
remove n_obj; remove e_obj; remove w_obj; remove s_obj;
```

('Curses' does a similar trick when the player boards a ship, taking away the normal directions in favour of "port", "starboard", "fore" and "aft".) As a fine point of style, turquoise (*yax*) is the world colour for 'here', so add a grammar line to make this cause a "look":

```
Verb 'turquoise' 'yax' * -> Look;
```

• 10 (p. 114) This time it's not enough to define a new way of saying an old direction: "xyzzy" is an entirely new direction. It needs a direction property, say `xyzzy_to`, and this property needs to be declared as a "common property", partly for efficiency reasons, partly because that's just how the library works. (Common properties are those declared in advance of use with `Property`. See §3 for further details.) So:

```
Property xyzzy_to;
CompassDirection xyzzy_obj "magic word" compass
  with name 'xyzzy', door_dir xyzzy_to;
```

• 11 (p. 114) We exchange the east and west direction references:

```
[ SwapDirs o1 o2 x;
  x = o1.door_dir; o1.door_dir = o2.door_dir; o2.door_dir = x;
];
[ ReflectWorld;
  SwapDirs(e_obj, w_obj);
  SwapDirs(ne_obj, nw_obj);
  SwapDirs(se_obj, sw_obj);
];
```

• 12 (p. 114) This is a prime candidate for using variable strings (see §1.11):

```
[ NormalWorld; string 0 "east"; string 1 "west"; ];
[ ReversedWorld; string 0 "west"; string 1 "east"; ];
```

where `NormalWorld` is called in `Initialise` and `ReversedWorld` when the reflection happens. Write `@00` in place of "east" in any double-quoted printable string, and similarly `@01` for "west". It will be printed as whichever is currently set.

• 13 (p. 114) Make all the location objects members of a class called:

```
Class Room
 with n_to, e_to, w_to, s_to, ne_to, nw_to, se_to, sw_to,
      in_to, out_to, u_to, d_to;
```

These properties are needed to make sure we can always write any map connection value to any room. Now define a routine which works on two opposite direction properties at a time:

```
[ TwoWay x dp1 dp2 y;
  y = x.dp1; if (metaclass(y) == Object && y ofclass Room) y.dp2 = x;
  y = x.dp2; if (metaclass(y) == Object && y ofclass Room) y.dp1 = x;
];
```

Note that some map connections run to doors (see §13) and not locations, and any such map connections need special handling which this solution can't provide: so we check that `y ofclass Room` before setting any direction properties for it. The actual code to go into `Initialise` is now simple:

```
objectloop (x ofclass Room) {
    TwoWay(x, n_to, s_to);   TwoWay(x, e_to, w_to);
    TwoWay(x, ne_to, sw_to); TwoWay(x, nw_to, se_to);
    TwoWay(x, u_to, d_to);   TwoWay(x, in_to, out_to);
}
```

• 14 (p. 117) The following code only works because `react_before` happens in advance of `before`. It uses `react_before` (rather than `each_turn`, say, or a `daemon`) to see if the gloves can be joined, in order to get in ahead of any `Look`, `Search` or `Inv` action which might want to talk about the gloves; whereas the `before` routine applies only if the player specifically refers to one of the gloves by name:

```
Class Glove
```

```
 with name 'white' 'glove' 'silk',
      article "the",
      react_before [;
          if (self in gloves) rfalse;
          if (parent(right_glove) ~= parent(left_glove)) rfalse;
          if (left_glove has worn && right_glove hasnt worn) rfalse;
          if (left_glove hasnt worn && right_glove has worn) rfalse;
          if (left_glove has worn) give gloves worn;
              else give gloves ~worn;
          move gloves to parent(right_glove);
          move right_glove to gloves; move left_glove to gloves;
          give right_glove ~worn; give left_glove ~worn;
      ],
      before [;
          if (self notin gloves) rfalse;
          move left_glove to parent(gloves);
          move right_glove to parent(gloves);
          if (gloves has worn) {
              give left_glove worn; give right_glove worn;
          }
          remove gloves;
      ],
  has clothing;
Object -> gloves "white gloves"
  with name 'white' 'gloves' 'pair' 'of',
       article "a pair of",
  has  clothing transparent;
Glove -> -> left_glove "left glove"
  with description "White silk, monogrammed with a scarlet R.",
       name 'left';
Glove -> -> right_glove "right glove"
  with description "White silk, monogrammed with a scarlet T.",
       name 'right';
```

• 15 (p. 119) "That was an unaccompanied bass pedal solo by Michael Rutherford. . . this is *The Musical Box*." (Announcement following tuning-up in a Genesis concert, Peter Gabriel, February 1973.) The easy thing to forget here is to make the container openable, because all the keys in the world won't help the player if the box hasn't got that.

```
Object -> "musical box",
  with name 'musical' 'box',
       with_key silver_key,
       capacity 1,
  has  lockable locked openable container;
Object -> -> "score of a song" with name 'score' 'music' 'song',
       article "the",
       description "~The Return of Giant Hogweed~.";
Object -> silver_key "silver key"
  with name 'silver' 'key';
```

• 16 (p. 120)

```
Object -> bag "toothed bag"
  with name 'toothed' 'bag',
       description "A capacious bag with a toothed mouth.",
       before [;
           LetGo: "The bag defiantly bites itself
                  shut on your hand until you desist.";
           Close: "The bag resists all attempts to close it.";
       ],
       after [;
           Receive:
               "The bag wriggles hideously as it swallows ",
               (the) noun, ".";
       ],
  has  container open openable;
```

• 17 (p. 121)

```
Object -> glass_box "glass box with a lid"
  with name 'glass' 'box' 'with' 'lid'
  has  container transparent openable open;
Object -> steel_box "steel box with a lid"
  with name 'steel' 'box' 'with' 'lid'
  has  container openable open;
```

• 18 (p. 121) See §14 for `switchable`, used below to make the power button actually do something.

```
Object television "portable television set"
  with name 'tv' 'television' 'set' 'portable',
       before [;
           SwitchOn: <<SwitchOn power_button>>;
           SwitchOff: <<SwitchOff power_button>>;
           Examine: <<Examine screen>>;
       ],
  has  transparent;
Object -> power_button "power button"
  with name 'power' 'button' 'switch',
       after [;
           SwitchOn, SwitchOff: <<Examine screen>>;
       ],
  has  switchable;
Object -> screen "television screen"
  with name 'screen',
       before [;
           Examine: if (power_button hasnt on) "The screen is black.";
               "The screen writhes with a strange Japanese cartoon.";
       ];
```

• 19 (p. 121) The describe part of this answer is only decoration. Note the careful use of inp1 and inp2 rather than noun or second, just in case an action involves a number or some text instead of an object. (See the note at the end of §6.)

```
Object -> macrame_bag "macrame bag"
  with name 'macrame' 'bag' 'string' 'net' 'sack',
       react_before [;
           Examine, Search, Listen, Smell: ;
           default:
               if (inp1 > 1 && noun in self)
                   print_ret (The) noun, " is inside the bag.";
               if (inp2 > 1 && second in self)
                   print_ret (The) second, " is inside the bag.";
       ],
       before [;
           Open: "The woollen twine is knotted hopelessly tight.";
       ],
       describe [;
           print "^A macrame bag hangs from the ceiling, shut tight";
           if (child(self)) {
               print ". Inside you can make out ";
               WriteListFrom(child(self), ENGLISH_BIT);
           }
           ".";
       ],
  has  container transparent openable;
Object -> -> "gold watch"
  with name 'gold' 'watch',
       description "The watch has no hands, oddly.",
       react_before [;
           Listen:
               if (noun == nothing or self) "The watch ticks loudly.";
       ];
```

For WriteListFrom, see §27.

• 20 (p. 124) The "plank breaking" rule is implemented here in its door_to routine. Note that this returns true after killing the player.

```
Object -> PlankBridge "plank bridge"
  with description "Extremely fragile and precarious.",
       name 'precarious' 'fragile' 'wooden' 'plank' 'bridge',
       when_open "A precarious plank bridge spans the chasm.",
       door_to [;
           if (children(player) > 0) {
               deadflag = true;
               "You step gingerly across the plank, which bows under
               your weight. But your meagre possessions are the straw
               which breaks the camel's back!";
           }
           print "You step gingerly across the plank, grateful that
               you're not burdened.^";
           if (self in NearSide) return FarSide; return NearSide;
       ],
       door_dir [;
           if (self in NearSide) return s_to; return n_to;
       ],
       found_in NearSide FarSide,
  has  static door open;
```

There might be a problem with this solution if your game also contained a character who wandered about, and whose code was clever enough to run door_to routines for any doors it ran into. If so, door_to could perhaps be modified to check that the actor is the player.

• 21 (p. 124)

```
Object -> illusory_door "ironbound door",
  with name 'iron' 'ironbound' 'door',
       door_dir e_to, door_to Armoury,
       before [;
           Enter: return self.vanish();
       ],
       react_before [;
```

```
            Go: if (noun == e_obj) return self.vanish();
        ],
        vanish [;
            location.(self.door_dir) = self.door_to; remove self;
            print "The door vanishes, shown for the illusion it is!^";
            <<Go e_obj>>;
        ],
   has  locked lockable door openable;
```

We need to trap both the `Go e_obj` *and* `Enter illusory_door` actions and can't just wait for the latter to be converted into the former, because the door's `locked`, so it never gets as far as that.

• 22 (p. 128)

```
Object -> cage "iron cage"
  with name 'iron' 'cage' 'bars' 'barred' 'iron-barred',
       when_open
           "An iron-barred cage, large enough to stoop over inside,
           looms ominously here.",
       when_closed "The iron cage is closed.",
       inside_description "You stare out through the bars.",
  has  enterable container openable open transparent static;
```

• 23 (p. 129)   First define a class `Road`:

```
Class Road has light;
```

Every road-like location should belong to this class, so for instance:

```
Road Trafalgar_Square "Trafalgar Square"
with n_to National_Gallery, e_to Strand,
     w_to Pall_Mall, s_to Whitehall,
     description "The Square is overlooked by a pillared statue
         of Admiral Lord Horatio Nelson (no relation), naval hero
         and convenience to pigeons since 1812.";
```

Now change the car's `before` as follows:

```
     before [ way;
         Go: way = location.(noun.door_dir)();
             if (~~(way ofclass Road)) {
                 print "You can't drive the car off-road.^";
                 return 2;
             }
             if (car has on) "Brmm! Brmm!";
```

```
            print "(The ignition is off at the moment.)^";
    ],
```

The first line of the Go clause works out what the game's map places in the given direction. For instance, noun is e_obj, so that its direction property is held in noun.door_dir, whose value is e_to. Sending Trafalgar_Square.e_to(), we finally set way to Strand. This turns out to be of class Road, so the movement is permitted. As complicated as this seems, getting a car across the real Trafalgar Square is substantially more convoluted.

• 24 (p. 130) We can't push up or down, but the ball lives on a north to south slope, so we'll just convert pushing up into pushing north, and down into south. Inserting these lines into the before rule for PushDir does the trick:

```
if (second == u_obj) <<PushDir self n_obj>>;
if (second == d_obj) <<PushDir self s_obj>>;
```

• 25 (p. 132) The following involves more parsing expertise than is strictly needed, partly because it's a good opportunity to show off a routine called ParseToken. First we define:

```
Class Biblical with name 'book' 'of';
Class Gospel class Biblical with name 'gospel' 'saint' 'st';
Gospel "Gospel of St Matthew" with name 'matthew', chapters 28;
Gospel "Gospel of St Mark" with name 'mark', chapters 16;
Gospel "Gospel of St Luke" with name 'luke', chapters 24;
Gospel "Gospel of St John" with name 'john', chapters 21;
...
Biblical "Book of Revelation" with name 'revelation', chapters 22;
```

And here is the Bible itself:

```
Object -> "black Tyndale Bible"
  with name 'bible' 'black' 'book',
       initial "A black Bible rests on a spread-eagle lectern.",
       description "A splendid foot-high Bible, which must have
           survived the burnings of 1520.",
       before [ bk ch_num;
           Consult:
               do {
                   wn = consult_from;
                   bk = ParseToken(SCOPE_TT, BiblicalScope);
               } until (bk ~= GPR_REPARSE);
               if (bk == GPR_FAIL) "That's not a book in this Bible.";
               if (NextWord() ~= 'chapter') wn--;
               ch_num = TryNumber(wn);
```

```
            if (ch_num == -1000)
                "You read ", (the) bk, " right through.";
            if (ch_num > bk.chapters) "There are only ", †
                (number) bk.chapters," chapters in ",(the) bk,".";
            "Chapter ", (number) ch_num, " of ", (the) bk,
                " is too sacred for you to understand now.";
    ];
```

The first six lines under `Consult` look at what the player typed from word `consult_from` onwards, and set `bk` to be the Book which best matches. The call to `ParseToken` makes the parser behave as if matching $\boxed{\texttt{scope=BiblicalScope}}$, which means that we need to define a scope routine:

```
[ BiblicalScope bk;
  switch (scope_stage) {
      1: rfalse;
      2: objectloop (bk ofclass Biblical) PlaceInScope(bk); rtrue;
  }
];
```

See §32 for more, but this tells the parser to accept only the name of a single, specific object of class `Biblical`. The effect of all of this fuss is to allow the following dialogue:

```
>look up gospel in bible
Which do you mean, the Gospel of St Matthew, the Gospel of St Mark, the
Gospel of St Luke or the Gospel of St John?
>mark
You read the Gospel of St Mark right through.
>look up St John chapter 17 in bible
Chapter seventeen of the Gospel of St John is too sacred for you to understand
now.
```

For a simpler solution, `Consult` could instead begin like this:

```
wn = consult_from;
switch (NextWord()) {
    'matthew': bk = St_Matthew;
    'mark': bk = St_Mark;
...
    default: "That's not a book in this Bible.";
}
```

† Actually, Philemon, II. John, III. John and Jude have only one chapter apiece, so we ought to take more care over grammar here.

• 26 (p. 137) Note that whether reacting before or after, the psychiatrist does not cut any actions short, because his `react_before` and `react_after` both return `false`.

```
Object -> psychiatrist "bearded psychiatrist"
  with name 'bearded' 'doctor' 'psychiatrist' 'psychologist' 'shrink',
       initial "A bearded psychiatrist has you under observation.",
       life [;
           "He is fascinated by your behaviour, but makes no attempt
           to interfere with it.";
       ],
       react_after [;
           Insert: print "~Subject associates ", (name) noun, " with ",
               (name) second, ". Interesting.~^^";
           PutOn: print "~Subject puts ", (name) noun, " on ",
               (name) second, ". Interesting.~^^";
           Look: print "^~Pretend I'm not here.~^^";
       ],
       react_before [;
           Take, Remove: print "~Subject feels lack of ", (the) noun,
               ". Suppressed Oedipal complex? Hmm.~^^";
       ],
  has  animate;
```

• 27 (p. 139) There are several ways to do this. The easiest is to add more grammar to the parser and let it do the hard work:

```
Object -> computer "computer"
  with name 'computer',
       theta_setting,
       orders [;
           Theta: if (noun < 0 || noun >= 360)
                      "~That value of theta is out of range.~";
                  self.theta_setting = noun;
                  "~Theta set. Waiting for additional values.~";
           default: "~Please rephrase.~";
       ],
  has  talkable;
...
[ ThetaSub; "You must tell your computer so."; ];
Verb 'theta' * 'is' number -> Theta;
```

• 28 (p. 139) Add the following lines, after the inclusion of `Grammar`:

```
[ SayInsteadSub; "[To talk to someone, please type ~someone, something~
```

```
   or else ~ask someone about something~.]"; ];
Extend 'answer' replace * topic -> SayInstead;
Extend 'tell'   replace * topic -> SayInstead;
```

A slight snag is that this will throw out "nigel, tell me about the grunfeld defence" (which the library will normally convert to an Ask action, but can't if the grammar for "tell" is missing). To avoid this, you could instead of making the above directives Replace the TellSub routine (see §25) by the SayInsteadSub one.

• 29 (p. 140) Obviously, a slightly wider repertoire of actions might be a good idea, but:

```
Object -> Charlotte "Charlotte"
  with name 'charlotte' 'charlie' 'chas',
       simon_said,
       grammar [;
           self.simon_said = false;
           wn = verb_wordnum;
           if (NextWord() == 'simon' && NextWord() == 'says') {
               if (wn > num_words) print "Simon says nothing, so ";
               else {
                   self.simon_said = true;
                   verb_wordnum = wn;
               }
           }
       ],
       orders [ i;
           if (self.simon_said == false)
               "Charlotte sticks her tongue out.";
           WaveHands: "Charlotte waves energetically.";
           default: "~Don't know how,~ says Charlotte.";
       ],
       initial "Charlotte wants to play Simon Says.",
  has  animate female proper;
```

(The variable i isn't needed yet, but will be used by the code added in the answer to the next exercise.) The test to see if wn has become larger than num_words prevents Charlotte from setting verb_wordnum to a non-existent word, which could only happen if the player typed just "charlotte, simon says". If so, the game will reply "Simon says nothing, so Charlotte sticks her tongue out."

• 30 (p. 140) First add a Clap verb (this is easy). Then give Charlotte a number property (initially 0, say) and add these three lines to the end of Charlotte's grammar routine:

```
    self.number = TryNumber(verb_wordnum);
    if (self.number ~= -1000) {
        action = ##Clap; noun = 0; second = 0; rtrue;
    }
```

Her orders routine now needs the new clause:

```
    Clap: if (self.number == 0) "Charlotte folds her arms.";
          for (i=0: i<self.number: i++) {
              print "Clap! ";
              if (i == 100)
                  print "(You must be regretting this by now.) ";
              if (i == 200)
                  print "(What a determined person she is.) ";
          }
          if (self.number > 100)
             "^^Charlotte is a bit out of breath now.";
         "^^~Easy!~ says Charlotte.";
```

• 31 (p. 140) The interesting point here is that when the grammar property finds the word "take", it accepts it and has to move verb_wordnum on by one to signal that a word has been parsed succesfully.

```
Object -> Dan "Dyslexic Dan"
  with name 'dan' 'dyslexic',
       grammar [;
           if (verb_word == 'take') { verb_wordnum++; return 'drop'; }
           if (verb_word == 'drop') { verb_wordnum++; return 'take'; }
       ],
       orders [;
           Take: "~What,~ says Dan, ~you want me to take ",
                     (the) noun, "?~";
           Drop: "~What,~ says Dan, ~you want me to drop ",
                     (the) noun, "?~";
           Inv: "~That I can do,~ says Dan. ~I'm empty-handed.~";
           No: "~Right you be then.~";
           Yes: "~I'll be having to think about that.~";
           default: "~Don't know how,~ says Dan.";
       ],
       initial "Dyslexic Dan is here.",
  has  animate proper;
```

Since the words have been exchanged before any parsing takes place, Dan even responds to "drop inventory" and "take coin into slot".

• 32 (p. 141) Suppose Dan's grammar (but nobody else's) for the "examine" verb is to be extended. His grammar routine should also contain:

```
    if (verb_word == 'examine' or 'x//') {
        verb_wordnum++; return -'danx,';
    }
```

(Note the crudity of this: it looks at the actual verb word, so you have to check any synonyms yourself.) The verb "danx," must be declared later:

```
Verb 'danx,' * 'conscience' -> Inv;
```

and now "Dan, examine conscience" will send him an `Inv` order: but "Dan, examine cow pie" will still send `Examine cow_pie` as usual.

• 33 (p. 141) See §20 for `the_time` and other chronological matters. In particular, note that the game will need to call the library's `SetTime` routine to decide on a time of day.

```
[ AlTime x; print (x/60), ":", (x%60)/10, x%10; ];
Object -> alarm_clock "alarm clock"
  with name 'alarm' 'clock',
       alarm_time 480, ! 08:00
       description [;
           print "The alarm is ";
           if (self has on) print "on, "; else print "off, but ";
           "the clock reads ", (AlTime) the_time,
           " and the alarm is set for ", (AlTime) self.alarm_time, ".";
       ],
       react_after [;
           Inv: if (self in player) { new_line; <<Examine self>>; }
           Look: if (self in location) { new_line; <<Examine self>>; }
       ],
       daemon [ td;
           td = (1440 + the_time - self.alarm_time) % 1440;
           if (td >= 0 && td <= 3 && self has on)
               "^Beep! Beep! The alarm goes off.";
       ],
       grammar [; return 'alarm,'; ],
       orders [;
           SwitchOn: give self on; StartDaemon(self);
               "~Alarm set.~";
           SwitchOff: give self ~on; StopDaemon(self);
               "~Alarm off.~";
           SetTo: self.alarm_time=noun; <<Examine self>>;
```

```
            default: "^On, off or a time of day, pliz.^";
        ],
        life [;
            "[Try ~clock, something~ to address the clock.]";
        ],
  has  talkable;
```

(So the alarm sounds for three minutes after its setting, then gives in.) Next, add a new verb to the grammar:

```
Verb 'alarm,'
    * 'on' -> SwitchOn
    * 'off' -> SwitchOff
    * TimeOfDay -> SetTo;
```

using a token for parsing times of day called `TimeOfDay`: as this is one of the exercises in §31, it won't be given here. Note that since the word "alarm," can't be matched by anything the player types, this verb is concealed from ordinary grammar. The orders we produce here are not used in the ordinary way (for instance, the action `SwitchOn` with no `noun` or `second` would never ordinarily be produced by the parser) but this doesn't matter: it only matters that the grammar and the `orders` property agree with each other.

•34 (p. 141)

```
Object -> tricorder "tricorder"
  with name 'tricorder',
       grammar [; return 'tc,'; ],
       orders [;
           Examine: if (noun == player) "^You radiate life signs.^";
               print "^", (The) noun, " radiates ";
               if (noun hasnt animate) print "no ";
               "life signs.^";
           default: "The tricorder bleeps.";
       ],
       life [;
           "The tricorder is too simple.";
       ],
  has  talkable;
...
Verb 'tc,' * noun -> Examine;
```

•35 (p. 141)

```
Object replicator "replicator"
  with name 'replicator',
       grammar [;  return 'rc,'; ],
       orders [;
           Give:
               if (noun in self)
                   "The replicator serves up a cup of ",
                   (name) noun, " which you drink eagerly.";
               "~That is not something I can replicate.~";
           default: "The replicator is unable to oblige.";
       ],
       life [;
           "The replicator has no conversational skill.";
       ],
  has  talkable;
Object -> "Earl Grey tea" with name 'earl' 'grey' 'tea';
Object -> "Aldebaran brandy" with name 'aldebaran' 'brandy';
Object -> "distilled water" with name 'distilled' 'water';
...
Verb 'rc,' * held -> Give;
```

The point to note here is that the `held` token means 'held by the replicator' here, as the actor is the replicator, so this is a neat way of getting a 'one of the following phrases' token into the grammar.

• 36 (p. 141) This is similar to the previous exercises. Suppose that the crew are all members of the class `StarFleetOfficer`. The `orders` property for the badge is then:

```
orders [;
    Examine:
        if (parent(noun))
            "~", (name) noun, " is in ", (name) parent(noun), ".~";
        "~", (name) noun, " is no longer aboard this demonstration
        game.~";
    default: "The computer's only really good for locating the crew.";
],
```

and the grammar simply returns `'stc,'` which is defined as:

```
[ Crew i;
  switch(scope_stage)
  {  1: rfalse;
     2: objectloop (i ofclass StarFleetOfficer) PlaceInScope(i); rtrue;
  }
];
Verb 'stc,' * 'where' 'is' scope=Crew -> Examine;
```

An interesting point is that the scope routine `scope=Crew` doesn't need to do anything at scope stage 3 (usually used for printing out errors) because the normal error-message printing system is never reached. Something like "computer, where is Comminder Doto" causes a `##NotUnderstood` order.

• 37 (p. 142)

```
Object -> Zen "Zen"
  with name 'zen' 'flight' 'computer',
       initial "Square lights flicker unpredictably across a hexagonal
           fascia on one wall, indicating that Zen is on-line.",
       grammar [; return 'zen,'; ],
       orders [;
           ZenScan: "The main screen shows a starfield,
               turning through ", noun, " degrees.";
           Go:  "~Confirmed.~ The ship turns to a new bearing.";
           SetTo: if (noun > 12) "~Standard by ", (number) noun,
                   " exceeds design tolerances.~";
               if (noun == 0) "~Confirmed.~ The ship's engines stop.";
               "~Confirmed.~ The ship's engines step to
               standard by ", (number) noun, ".";
           Take: if (noun ~= force_wall) "~Please clarify.~";
               "~Force wall raised.~";
           Drop: if (noun ~= blasters)   "~Please clarify.~";
               "~Battle-computers on line.
               Neutron blasters cleared for firing.~";
           NotUnderstood: "~Language banks unable to decode.~";
           default: "~Information. That function is unavailable.~";
       ],
  has  talkable proper static;
Object -> -> force_wall "force wall"
  with name 'force' 'wall' 'shields';
Object -> -> blasters "neutron blasters"
  with name 'neutron' 'blasters';
...
[ ZenScanSub; "This is never called but makes the action exist."; ];
Verb 'zen,'
    * 'scan' number 'orbital' -> ZenScan
    * 'set' 'course' 'for' scope=Planet -> Go
    * 'speed' 'standard' 'by' number -> SetTo
    * 'raise' held -> Take
    * 'clear' held 'for' 'firing' -> Drop;
```

Dealing with `Ask`, `Answer` and `Tell` are left to the reader. As for planetary parsing:

```
[ Planet;
  switch (scope_stage) {
      1: rfalse; ! Disallow multiple planets
      2: ScopeWithin(galaxy); rtrue; ! Scope set to contents of galaxy
  }
];
Object galaxy;
Object -> "Earth" with name 'earth' 'terra';
Object -> "Centauro" with name 'centauro';
Object -> "Destiny" with name 'destiny';
```

and so on for numerous other worlds of the oppressive if somewhat cheaply decorated Federation.

• 38 (p. 142) This is a typical bit of "scope hacking", in this case done with the `InScope` entry point, though providing the viewscreen with a similar `add_to_scope` routine would have done equally well. See §32.

```
    [ InScope;
       if (action_to_be == ##Examine or ##Show && location == Bridge)
           PlaceInScope(noslen_maharg);
       if (scope_reason == TALKING_REASON)
           PlaceInScope(noslen_maharg);
       rfalse;
    ];
```

The variable `scope_reason` is always set to the constant value `TALKING_REASON` when the game is trying to work out who you wish to talk to: so it's quite easy to make the scope different for conversational purposes.

• 39 (p. 142) Martha and the sealed room are defined as follows:

```
Object sealed_room "Sealed Room"
  with description
          "I'm in a sealed room, like a squash court without a door,
           maybe six or seven yards across",
  has  light;
Object -> ball "red ball" with name 'red' 'ball';
Object -> martha "Martha"
  with name 'martha',
       orders [ r;
           r = parent(self);
           Give:
               if (noun notin r) "~That's beyond my telekinesis.~";
               if (noun == self) "~Teleportation's too hard for me.~";
               move noun to player;
               "~Here goes...~ and Martha's telekinetic talents
                   magically bring ", (the) noun, " to your hands.";
           Look:
               print "~", (string) r.description;
               if (children(r) == 1) ". There's nothing here but me.~";
               print ". I can see ";
               WriteListFrom(child(r), CONCEAL_BIT + ENGLISH_BIT);
              ".~";
           default: "~Afraid I can't help you there.~";
       ],
       life [;
           Ask: "~You're on your own this time.~";
           Tell: "Martha clucks sympathetically.";
           Answer: "~I'll be darned,~ Martha replies.";
       ],
   has animate female concealed proper;
```

but the really interesting part is the `InScope` routine to fix things up:

```
[ InScope actor;
   if (actor == martha) PlaceInScope(player);
   if (actor == player && scope_reason == TALKING_REASON)
       PlaceInScope(martha);
   rfalse;
];
```

Note that since we want two-way communication, the player has to be in scope to Martha too: otherwise Martha won't be able to follow the command "martha, give me the fish", because "me" will refer to something beyond her scope.

• 40 (p. 145) In an outdoor game on a summer's day, one way would be to make a scenery object for the Sun which is `found_in` every location and has `light`. A sneakier method is to put the line

```
    give player light;
```

in `Initialise`. Now there's never darkness near the player. Unless there are wrangles involving giving instructions to people in different locations (where it's still dark), or the player having an out-of-body experience (see §21), this will work perfectly well. The player is never told "You are giving off light", so nothing seems incongruous in play.

• 41 (p. 146) Just test if `HasLightSource(gift) == true`.

• 42 (p. 148) This is a crude implementation, for brevity (the real 'Zork' thief has an enormous stock of attached messages). It does no more than choose a random exit and move through it on roughly one turn in three. A `life` routine is omitted, and of course this particular thief steals nothing. See 'The Thief' for a much fuller, annotated implementation.

```
Object -> thief "thief"
  with name 'thief' 'gentleman' 'mahu' 'modo',
       each_turn "^The thief growls menacingly.",
       daemon [ direction thief_at way exit_count exit_chosen;
           if (random(3) ~= 1) rfalse;
           thief_at = parent(thief);
           objectloop (direction in compass) {
               way = thief_at.(direction.door_dir);
               if (way ofclass Object && way hasnt door) exit_count++;
           }
           if (exit_count == 0) rfalse;
           exit_chosen = random(exit_count); exit_count = 0;
           objectloop (direction in compass) {
               way = thief_at.(direction.door_dir);
               if (way ofclass Object && way hasnt door) exit_count++;
               if (exit_count == exit_chosen) {
                   move self to way;
                   if (thief_at == location) "^The thief stalks away!";
                   if (way == location) "^The thief stalks in!";
                   rfalse;
               }
           }
       ],
  has  animate;
```

(Not forgetting to `StartDaemon(thief)` at some point, for instance in the game's `Initialise` routine.) So the thief walks at random but never via doors, bridges and the like, because these may be locked or have rules attached. This is only a first try, and in a good game one would occasionally see the thief do something surprising, such as open a secret door. As for the `name`, "The Prince of Darkness is a gentleman. Modo he's called, and Mahu" (William Shakespeare, *King Lear* III iv).

• 43 (p. 148) We shall use a new property called `weight` and decide that any object which doesn't provide any particular weight will weigh 10 units. Clearly, an object which contains other objects will carry their weight too, so:

```
[ WeightOf obj t i;
   if (obj provides weight) t = obj.weight; else t = 10;
   objectloop (i in obj) t = t + WeightOf(i);
   return t;
];
```

Once every turn we shall check how much the player is carrying and adjust a measure of the player's fatigue accordingly. There are many ways we could choose to calculate this: for the sake of example we'll define two constants:

```
Constant FULL_STRENGTH = 500;
Constant HEAVINESS_THRESHOLD = 100;
```

Initially the player's strength will be the maximum possible, which we'll set to 500. Each turn the amount of weight being carried is subtracted from this, but 100 is also added on (without exceeding the maximum value). So if the player carries more than 100 units then strength declines, but if the weight carried falls below 100 then strength recovers. If the player drops absolutely everything, the entire original strength will recuperate in at most 5 turns. Exhaustion sets in if strength reaches 0, and at this point the player is forced to drop something, giving strength a slight boost. Anyway, here's an implementation of all this:

```
Object WeightMonitor
  with players_strength,
       warning_level 5,
       activate [;
           self.players_strength = FULL_STRENGTH;
           StartDaemon(self);
       ],
       daemon [ warn strength item heaviest_weight heaviest_item;
           strength = self.players_strength
               - WeightOf(player) + HEAVINESS_THRESHOLD;
           if (strength < 0) strength = 0;
           if (strength > FULL_STRENGTH) strength = FULL_STRENGTH;
           self.players_strength = strength;
           if (strength == 0) {
               heaviest_weight = -1;
               objectloop(item in player)
                   if (WeightOf(item) > heaviest_weight) {
                       heaviest_weight = WeightOf(item);
                       heaviest_item = item;
                   }
               if (heaviest_item == nothing) return;
               print "^Exhausted with carrying so much, you decide
                   to discard ", (the) heaviest_item, ": ";
               <Drop heaviest_item>;
               if (heaviest_item in player) {
                   deadflag = true;
                   "^Unprepared for this, you collapse.";
               }
               self.players_strength =
                   self.players_strength + heaviest_weight;
           }
           warn = strength/100; if (warn == self.warning_level) return;
           self.warning_level = warn;
           switch (warn) {
               3: "^You are feeling a little tired.";
               2: "^Your possessions are weighing you down.";
               1: "^Carrying so much weight is wearing you out.";
               0: "^You're nearly exhausted enough to drop everything
                  at an inconvenient moment.";
           }
       ];
```

When exhaustion sets in, this daemon tries to drop the heaviest item. (The actual dropping is done with Drop actions: in case the item is, say, a wild boar, which would bolt away into the forest when released. Also, after the attempt to Drop, we check to see if the drop has succeeded, because the heaviest item might be a cannonball superglued to one's hands, or a boomerang, or some such.) Finally, of course, at some point – probably in Initialise – the game needs to send the message WeightMonitor.activate() to get things going.

• 44 (p. 148) Note that we use StopTimer before StartTimer in case the egg timer is already running. (StopTimer does nothing if the timer isn't running, while StartTimer does nothing if it is.)

```
Object -> "egg timer in the shape of a chicken"
  with name 'egg' 'timer' 'egg-timer' 'eggtimer' 'chicken' 'dial',
       description
           "Turn the dial on the side of the chicken to set this
           egg timer.",
       before [;
           Turn: StopTimer(self); StartTimer(self, 3);
               "You turn the dial to its three-minute mark, and the
               chicken begins a sort of clockwork clucking.";
       ],
       time_left,
       time_out [;
           "^~Cock-a-doodle-doo!~ says the egg-timer, in defiance of
           its supposedly chicken nature.";
       ];
```

• 45 (p. 149) See the next answer.

• 46 (p. 149)

```
Object tiny_claws "sound of tiny claws" thedark
  with article "the",
       name 'tiny' 'claws' 'sound' 'of' 'scuttling' 'scuttle'
            'things' 'creatures' 'monsters' 'insects',
       initial "Somewhere, tiny claws are scuttling.",
       before [;
           Listen: "How intelligent they sound, for mere insects.";
           Touch, Taste: "You wouldn't want to. Really.";
           Smell: "You can only smell your own fear.";
           Attack: "They easily evade your flailing about.";
           default: "The creatures evade you, chittering.";
       ],
       each_turn [; StartDaemon(self); ],
       turns_active,
       daemon [;
           if (location ~= thedark) {
               self.turns_active = 0; StopDaemon(self); rtrue;
           }
           switch (++(self.turns_active)) {
               1: "^The scuttling draws a little nearer, and your
                  breathing grows loud and hoarse.";
               2: "^The perspiration of terror runs off your brow.
                  The creatures are almost here!";
               3: "^You feel a tickling at your extremities and kick
                  outward, shaking something chitinous off. Their
                  sound alone is a menacing rasp.";
               4: deadflag = true;
                  "^Suddenly there is a tiny pain, of a
                  hypodermic-sharp fang at your calf. Almost at once
                  your limbs go into spasm, your shoulders and
                  knee-joints lock, your tongue swells...";
           }
       ];
```

• 47 (p. 150) Either set a daemon to watch for `the_time` suddenly dropping, or put such a watch in the game's `TimePasses` routine.

• 48 (p. 150) A minimal solution is as follows. Firstly, we'll define a class of outdoor locations, and make two constant definitions:

```
Constant SUNRISE = 360;  ! i.e., 6 am
Constant SUNSET = 1140;  ! i.e., 7 pm
Class OutdoorLocation;
```

We handle night and day by having a light-giving scenery object present in outdoor locations during the day: we shall call this object "the sun":

```
Object Sun "sun"
  with name 'sun',
       found_in [;
           if (real_location ofclass OutdoorLocation) rtrue;
       ],
       before [;
           Examine: ;
           default: "The sun is too far away.";
       ],
       daemon [;
           if (the_time >= SUNRISE && the_time < SUNSET) {
               if (self has absent) {
                   give self ~absent;
                   if (real_location ofclass OutdoorLocation) {
                       move self to place;
                       "^The sun rises, illuminating the landscape!";
                   }
               }
           } else {
               if (self hasnt absent) {
                   give self absent; remove self;
                   if (real_location ofclass OutdoorLocation)
                       "^As the sun sets, the landscape is plunged
                       into darkness.";
               }
           }
       ],
   has light scenery;
```

In the Initialise routine, you need to call StartDaemon(Sun);. If the game starts in the hours of darkness, you should also give the Sun absent. Daybreak and nightfall will be automatic from there on.

• 49 (p. 150) Because you don't know what order daemons will run in. A 'fatigue' daemon which makes the player drop something might come after the 'mid-air' daemon has run for this turn. Whereas each_turn happens after daemons and timers have run their course, and can fairly assume no further movements will take place this turn.

• 50 (p. 150) It would have to provide its own code to keep track of time, and it can do this by providing a TimePasses() routine. Providing "time" or even "date" verbs to tell the player would also be a good idea.

• 51 (p. 156) Two reasons. Firstly, sometimes you need to trap orders to other people, which react_before doesn't. Secondly, the player's react_before rule is not necessarily the first to react. Suppose in the deafness example that a cuckoo also has a react_before rule covering Listen, printing a message about birdsong. If this happens before the player object is reached, the deafness rule never applies.

• 52 (p. 156)

```
orders [;
    if (gasmask hasnt worn) rfalse;
    if (actor == self && action ~= ##Answer or ##Tell or ##Ask) rfalse;
    "Your speech is muffled into silence by the gas mask.";
],
```

• 53 (p. 157) The common man's *wayhel* was a lowly mouse. Since we think much more highly of the player:

```
Object warthog "Warthog"
  with name 'wart' 'hog' 'warthog', description "Muddy and grunting.",
       initial "A warthog snuffles and grunts about in the ashes.",
       orders [;
           Go, Look, Examine, Smell, Taste, Touch, Search,
               Jump, Enter: rfalse;
           Eat: "You haven't the knack of snuffling up to food yet.";
           default: "Warthogs can't do anything so involved. If it
               weren't for the nocturnal eyesight and the lost weight,
               they'd be worse off all round than people.";
       ],
  has  animate proper;
```

Using ChangePlayer(warthog); will then bring about the transformation, though we must also move the warthog to some suitable location. The promised nocturnal eyesight will be brought about by giving the warthog light for the period when the player is changed to it.

• 54 (p. 157) Change the cant_go message of the wormcast, dropping an even larger hint on the way:

```
    cant_go [;
        if (player ~= warthog)
            "Though you begin to feel certain that something lies
            behind and through the wormcast, this way must be an
            animal-run at best: it's far too narrow for your
            armchair-archaeologist's paunch.";
        print "The wormcast becomes slippery around your warthog
            body, and you squeal involuntarily as you burrow
            through the darkness, falling finally southwards to...^";
        PlayerTo(Burial_Shaft); rtrue;
    ],
```

• 55 (p. 157) This is more straightforward than it sounds. The most interesting point is that the map connection isn't between two rooms: it's between an enterable object, the cage, and a room, the burial chamber.

```
Object -> cage "iron cage"
  with name 'iron' 'cage' 'bars' 'barred' 'frame' 'glyphs',
       description "The glyphs read: Bird Arrow Warthog.",
       when_open
           "An iron-barred cage, large enough to stoop over inside,
           looms ominously here, its door open.  There are some glyphs
           on the frame.",
       when_closed "The iron cage is closed.",
       after [;
           Enter:
               print "The skeletons inhabiting the cage come alive,
                   locking bony hands about you, crushing and
                   pummelling. You lose consciousness, and when you
                   recover something grotesque and impossible has
                   occurred...^";
               move warthog to Antechamber; remove skeletons;
               give self ~open; give warthog light;
               self.after = 0;
               ChangePlayer(warthog, 1); <<Look>>;
       ],
       floor_open false,
       inside_description [;
           if (self.floor_open)
               "From the floor of the cage, an open earthen pit cuts
               down into the burial chamber.";
```

```
            "The bars of the cage surround you.";
        ],
        react_before [;
            Go: if (noun == d_obj && self.floor_open) {
                    PlayerTo(Burial_Shaft); rtrue;
                }
        ],
  has   enterable transparent container openable open static;
Object -> -> skeletons "skeletons"
  with name 'skeletons' 'skeleton' 'bone' 'skull' 'bones' 'skulls',
       article "deranged",
   has pluralname static;
Object Burial_Shaft "Burial Shaft"
  with description
           "In your eventual field notes, this will read:
           ~A corbel-vaulted crypt with an impacted earthen plug
           as seal above, and painted figures conjecturally
           representing the Nine Lords of the Night. Dispersed
           bones appear to be those of one elderly man and
           several child sacrifices, while other funerary remains
           include jaguar paws.~ (In field notes, it is essential
           not to give any sense of when you are scared witless.)",
       cant_go
           "The architects of this chamber were less than generous in
           providing exits. Some warthog seems to have burrowed in
           from the north, though.",
       n_to Wormcast,
       u_to [;
           cage.floor_open = true;
           self.u_to = self.opened_u_to;
           move selfobj to self;
           print "Making a mighty warthog-leap, you butt at the
               earthen-plug seal above the chamber, collapsing your
               world in ashes and earth. Something lifeless and
               terribly heavy falls on top of you: you lose
               consciousness, and when you recover, something
               impossible and grotesque has happened...^";
           ChangePlayer(selfobj); give warthog ~light; <<Look>>;
       ],
       before [;
           Jump: <<Go u_obj>>;
       ],
       opened_u_to [;
```

```
          PlayerTo(cage); rtrue;
      ],
  has  light;
```

• 56 (p. 158)

```
      orders [;
          if (player == self) {
              if (actor ~= self)
                 "You only become tongue-tied and gabble.";
              rfalse;
          }
          Attack: "The Giant looks at you with doleful eyes.
              ~Me not be so bad!~";
          default: "The Giant cannot comprehend your instructions.";
      ],
```

• 57 (p. 164) Give the "chessboard" room a short_name routine (it probably already has one, to print names like "Chessboard d6") and make it change the short name to "the gigantic Chessboard" if and only if action is currently set to ##Places.

• 58 (p. 165)

```
Class Quotation;
Object PendingQuote; Object SeenQuote;
[ QuoteFrom q;
  if (~~(q ofclass Quotation)) "*** Oops! Not a quotation. ***";
  if (q notin PendingQuote or SeenQuote) move q to PendingQuote;
];
[ AfterPrompt q;
  q = child(PendingQuote);
  if (q) {
      move q to SeenQuote;
      q.show_quote();
  }
];
Quotation AhPeru
 with show_quote [;
          box "Brother of Ingots -- Ah, Peru --"
              "Empty the Hearts that purchased you --"
              ""
              "-- Emily Dickinson";
      ];
```

`QuoteFrom(AhPeru)` will now do as it is supposed to. The children of the object `PendingQuote` act as a last in, first out queue, so if several quotations are pending in the same turn, this system will show them in successive turns, most recently requested first.

• 59 (p. 191) Place the following definition in between the inclusion of `"Parser.h"` and of `"Verblib.h"`:

```
Object LibraryMessages
  with before [;
           Prompt: switch (turns) {
               1: print "^What should you, the detective, do now?^>";
               2 to 9: print "^What next?^>";
               10: print "^(Aren't you getting tired of seeing ~What
                   next?~ From here on, the prompt will be much
                   shorter.)^^>";
               default: print "^>";
           }
           rtrue;
       ];
```

• 60 (p. 191) Looking in §A4, we find that the offending message is `Go`, number 1, but that this message appears in two cases: when the player is on a `supporter`, which we want to deal with, and when the player is in a `container`, which we want to leave alone.

```
Object LibraryMessages
  with before [ previous_parent;
           Go: if (lm_n == 1 && lm_o has supporter) {
                   print "(first getting off ", (the) lm_o, ")^";
                   previous_parent = parent(player);
                   keep_silent = true; <Exit>; keep_silent = false;
                   if (player in parent(previous_parent)) <<Go noun>>;
                   rtrue;
               }
       ];
```

Note that after we've tried to perform an `Exit` action, either we've made some progress (in that the player is no longer in the same parent) and can try the `Go` action again, or else we've failed, in which case something has been printed to that effect. Either way, we return `true`.

• 61 (p. 191)

```
Object LibraryMessages
```

```
  with before [;
           Push: if (lm_n == 3 && noun has switchable) {
                    if (noun has on) <<SwitchOff noun>>;
                    <<SwitchOn noun>>;
                }
       ];
```

*Exercises in Chapter IV*

• 62 (p. 194) Simply define the following, for accusative, nominative and capitalised nominative pronouns, respectively.

```
[ PronounAcc i;
    if (i hasnt animate || i has neuter) print "it";
    else { if (i has female) print "her"; else print "him"; } ];
[ PronounNom i;
    if (i hasnt animate || i has neuter) print "it";
    else { if (i has female) print "she"; else print "he"; } ];
[ CPronounNom i;
    if (i hasnt animate || i has neuter) print "It";
    else { if (i has female) print "She"; else print "He"; } ];
```

• 63 (p. 196) We first set up the pieces. Bobby Fischer (Black, lower case, vs D. Byrne, Rosenwald Tournament, New York, 1956), aged thirteen and with queen *en prise*, is about to play Be6! and will win.

```
Array Position ->
    "r...r.k.\
     pp...pbp\
     .qp...p.\
     ..B.....\
     ..BP..b.\
     Q.n..N..\
     P....PPP\
     ...R.K.R";
```

(The backslashes \ remove spacing, so that this array contains just 64 entries.) Now for the objects. It will only be an illusion that there are sixty-four different locations, so we make sure the player drops nothing onto the board's surface.

```
Object Chessboard
  with description [;
           print "A square expanse of finest ";
```

```
        if ((self.rank + self.file - 'a') % 2 == 1)
            print "mahogany"; else print "cedarwood"; ".";
    ],
    short_name [;
        if (action==##Places) { print "the Chessboard"; rtrue; }
        print "Square ", (char) self.file, self.rank; rtrue;
    ],
    rank 1, file 'a',
    n_to  [; return self.try_move_to(self.rank+1,self.file);   ],
    ne_to [; return self.try_move_to(self.rank+1,self.file+1); ],
    e_to  [; return self.try_move_to(self.rank,  self.file+1); ],
    se_to [; return self.try_move_to(self.rank-1,self.file+1); ],
    s_to  [; return self.try_move_to(self.rank-1,self.file);   ],
    sw_to [; return self.try_move_to(self.rank-1,self.file-1); ],
    w_to  [; return self.try_move_to(self.rank,  self.file-1); ],
    nw_to [; return self.try_move_to(self.rank+1,self.file-1); ],
    try_move_to [ r f na p;
         if (~~(r >= 1 && r <= 8 && f >= 'a' && f <= 'h'))
             "That would be to step off the chessboard.";
         move Piece to self; na = Piece.&name; na-->1 = 'white';
         give Piece ~proper ~female;
         p = Position->((8-r)*8 + f - 'a');
         switch (p) {
             '.': remove Piece;
             'p', 'r', 'n', 'b', 'q', 'k': na-->1 = 'black';
         }
         switch (p) {
             'p', 'P': na-->0 = 'pawn';
             'r', 'R': na-->0 = 'rook';
             'n', 'N': na-->0 = 'knight';
             'b', 'B': na-->0 = 'bishop';
             'q', 'Q': na-->0 = 'queen'; give Piece female;
             'k', 'K': na-->0 = 'king'; give Piece proper;
         }
         switch (p) {
             'p': Piece.short_name = "Black Pawn";
             ...
             'K': Piece.short_name = "The White King";
         }
         self.rank = r; self.file = f; give self ~visited;
         return self;
    ],
    after [;
```

```
            Drop: move noun to player;
                "From high above, a ghostly voice whispers ~J'adoube~,
                and ", (the) noun, " springs back into your hands.";
        ],
  has  light;
Object Piece
  with name '(kind)' '(colour)' 'piece' 'chess',
       short_name "(A short name filled in by Chessboard)",
       initial [; "This square is occupied by ", (a) self, "."; ],
  has  static;
```

And to get the player onto the board in the first place,

```
            PlayerTo(Chessboard.try_move_to(1,'a'));
```

• 64 (p. 197) Use the invent routine to signal to short_name and article routines to change their usual habits:

```
Object "ornate box"
  with name 'ornate' 'box' 'troublesome',
       altering_short_name,
       invent [;
           self.altering_short_name = (inventory_stage == 1);
       ],
       short_name [;
           if (self.altering_short_name) { print "box"; rtrue; }
       ],
       article [;
           if (self.altering_short_name) print "that troublesome";
           else print "an";
       ],
  has  container open openable;
```

Thus the usual short name and article "an ornate box" becomes "that troublesome box" in inventory listings, but nowhere else.

• 65 (p. 200) This answer is cheating, as it needs to know about the library's lookmode variable (set to 1 for normal, 2 for verbose or 3 for superbrief, according to the player's most recent choice). Simply include:

```
[ TimePasses;
  if (action ~= ##Look && lookmode == 2) <Look>;
];
```

• 66 (p. 203)

```
[ DoubleInvSub item number_carried number_worn;
  print "You are carrying ";
  objectloop (item in player) {
      if (item hasnt worn) { give item workflag; number_carried++; }
      else { give item ~workflag; number_worn++; }
  }
  if (number_carried == 0) print "nothing";
  else WriteListFrom(child(player),
          FULLINV_BIT + ENGLISH_BIT + RECURSE_BIT + WORKFLAG_BIT);
  if (number_worn == 0) ".";
  if (number_carried == 0) print ", but"; else print ". In addition,";
  print " you are wearing ";
  objectloop (item in player)
      if (item hasnt worn) give item ~workflag;
      else give item workflag;
  WriteListFrom(child(player),
      ENGLISH_BIT + RECURSE_BIT + WORKFLAG_BIT);
  ".";
];
```

• 67 (p. 204)

```
Class Letter
 with name 'letter' 'scrabble' 'piece' 'letters//p' 'pieces//p',
      list_together [;
          if (inventory_stage == 1) {
              print "the letters ";
              c_style = c_style | (ENGLISH_BIT + NOARTICLE_BIT);
              c_style = c_style &~ (NEWLINE_BIT + INDENT_BIT);
          }
          else print " from a Scrabble set";
      ],
      short_name [;
          if (listing_together ofclass Letter) rfalse;
          print "letter ", (object) self, " from a Scrabble set";
          rtrue;
      ],
      article "the";
```

The bitwise operation `c = c | ENGLISH_BIT` sets the given bit (i.e., adds it on to c) if it wasn't already set (i.e., if it hadn't already been added to c). The operation `&~`, which is actually two operators `&` (and) and `~` (not) put together, takes away something if it's present and otherwise does nothing. As many letters as desired can now be created, along the lines of

```
Letter -> "X" with name 'x//';
```

• 68 (p. 204)

```
Class Coin
 with name 'coin' 'coins//p',
      description "A round unstamped disc, presumably local currency.",
      list_together "coins",
      plural [;
          print (address) self.&name-->0;
          if (~~(listing_together ofclass Coin)) print " coins";
      ],
      short_name [;
          print (address) self.&name-->0;
          if (~~(listing_together ofclass Coin)) print " coin";
          rtrue;
      ],
      article [;
          if (listing_together ofclass Coin) print "one";
          else print "a";
      ];
Class GoldCoin   class Coin with name 'gold';
Class SilverCoin class Coin with name 'silver';
Class BronzeCoin class Coin with name 'bronze';
SilverCoin ->;
```

The trickiest lines here are the `print (address)` ones. This is the least commonly used printing-rule built into Inform, and is only really used to print out the text of a dictionary word: whereas

```
    print (string) 'gold';
```

is likely to print garbled and nonsensical text, because `'gold'` is a dictionary word not a string. Anyway, these lines print out the first entry in the `name` list for the coin, so they rely on the fact that `name` words accumulate in the front of the list. The class `Coin` starts the list as ("coin", "coins"), whereupon `SilverCoin` augments it to ("silver", "coin", "coins"). Finally, because a dictionary word only stores up to nine letters, a different solution would be needed to cope with molybdenum coins.

• 69 (p. 205)   Firstly, a printing rule to print the state of coins. Coin-objects will have a property called `heads_up` which is always either `true` or `false`:

```
[ Face x; if (x.heads_up) print "Heads"; else print "Tails"; ];
```

There are two kinds of coin but we'll implement them with three classes: `Coin` and two sub-categories, `GoldCoin` and `SilverCoin`. Since the coins only join up into trigrams when present in groups of three, we need a routine to detect this:

```
[ CoinsTogether cla member p common_parent;
  objectloop (member ofclass cla) {
      p = parent(member);
      if (common_parent == nothing) common_parent = p;
      else if (common_parent ~= p) return nothing;
  }
  return common_parent;
];
```

Thus `CoinsTogether(GoldCoin)` decides whether all objects of class `GoldCoin` have the same parent, returning either that parent or else `nothing`, and likewise for `SilverCoin`. Now the class definitions:

```
Class Coin
 with name 'coin' 'coins//p',
      heads_up true, article "the", metal "steel",
      after [;
          Drop, PutOn:
              self.heads_up = (random(2) == 1); print (Face) self;
              if (CoinsTogether(self.which_class)) {
                  print ". The ", (string) self.metal,
                      " trigram is now ", (Trigram) self;
              }
              ".";
      ];
[ CoinLT common_parent member count;
  if (inventory_stage == 1) {
      print "the ", (string) self.metal, " coins ";
      common_parent = CoinsTogether(self.which_class);
      if (common_parent &&
          (common_parent == location || common_parent has supporter)) {
          objectloop (member ofclass self.which_class) {
              print (name) member;
              switch (++count) {
                  1: print ", "; 2: print " and ";
                  3: print " (showing the trigram ",
                     (Trigram) self, ")";
              }
          }
          rtrue;
      }
      c_style = c_style | (ENGLISH_BIT + NOARTICLE_BIT);
      c_style = c_style &~ (NEWLINE_BIT + INDENT_BIT);
  }
  rfalse;
];
Class GoldCoin class Coin
 with name 'gold', metal "gold", interpretation gold_trigrams,
      which_class GoldCoin,
      list_together [; return CoinLT(); ];
Class SilverCoin class Coin
 with name 'silver', metal "silver", interpretation silver_trigrams,
      which_class SilverCoin,
      list_together [; return CoinLT(); ];
Array gold_trigrams -->   "fortune" "change" "river flowing" "chance"
                          "immutability" "six stones in a circle"
                          "grace" "divine assistance";
Array silver_trigrams --> "happiness" "sadness" "ambition" "grief"
                          "glory" "charm" "sweetness of nature"
                          "the countenance of the Hooded Man";
```

(There are two unusual points here. Firstly, the CoinsLT routine is not simply given as the common list_together value in the coin class since, if it were, all six coins would be grouped together: we want two groups of three, so the gold and silver coins have to have different list_together values. Secondly, if a trigram is together and on the floor, it is not good enough to simply append text like "showing Tails, Heads, Heads (change)" at inventory_stage 2 since the coins may be listed in a funny order. In that event, the order the coins are listed in doesn't correspond to the order their values are listed in, which is misleading. So instead CoinsLT takes over entirely at inventory_stage 1 and prints out the list of three itself, returning true to stop the list from being printed out by the library as well.) To resume: whenever coins are listed together, they are grouped into gold and silver. Whenever trigrams are visible they are to be described by either Trigram(GoldClass) or Trigram(SilverClass):

```
[ Trigram acoin cla member count state;
  cla = acoin.which_class;
  objectloop (member ofclass cla) {
      print (Face) member;
      if (count++ < 2) print ","; print " ";
      state = state*2 + member.heads_up;
  }
  print "(", (string) (acoin.interpretation)-->state, ")";
];
```

Note that the class definitions refer to their arrays, but do not actually include the arrays themselves – this saves each coin carrying a copy of the whole array around with it, which would be wasteful. It's a marginal point, though, as there are only six actual coins:

```
GoldCoin -> "goat" with name 'goat';
GoldCoin -> "deer" with name 'deer';
GoldCoin -> "chicken" with name 'chicken';
SilverCoin -> "robin" with name 'robin';
SilverCoin -> "snake" with name 'snake';
SilverCoin -> "bison" with name 'bison';
```

If these were found in (say) a barn, we might have to take more care not to let them be called "goat" and so on, but let us assume not.

• 70 (p. 205) There are innumerable methods for sorting, and an extensive research literature can advise on which method is likely to perform fastest in which circumstances. Here matters are complicated by the fact that it is not easy to exchange two objects in the tree without shuffling many other objects backwards and forwards to do so. The solution below is concise rather than efficient. Briefly, if there are $N$ members of the `AlphaSorted` class then the game pauses before play begins to make $N^2$ comparisons of strings (this could easily be made faster but only happens once anyway): it then numbers off the members 1, 2, 3, ... in their correct alphabetical order, and subsequently uses this list to check only those objects which have moved since they were last checked.

```
Object heap1; Object heap2; Object heap3; Object heap4;
Class AlphaSorted
  with last_parent, last_sibling, sort_ordering,
       react_before [ t u v;
           Look, Search, Open, Inv:
               if (parent(self) == self.last_parent
                   && sibling(self) == self.last_sibling) rfalse;
               if (self.sort_ordering == 0)
                   objectloop (t ofclass AlphaSorted)
                       t.order_yourself();
               t = parent(self);
               while ((u = child(t)) ~= 0) {
                   if (u ofclass AlphaSorted) {
                       v = self.sort_ordering - u.sort_ordering;
                       if (v < 0) move u to heap1;
                       if (v == 0) move u to heap2;
                       if (v > 0) move u to heap3;
                   }
                   else move u to heap4;
               }
```

```
            while ((u = child(heap1)) ~= 0) move u to t;
            while ((u = child(heap2)) ~= 0) move u to t;
            while ((u = child(heap3)) ~= 0) move u to t;
            while ((u = child(heap4)) ~= 0) move u to t;
            self.last_parent = parent(self);
            self.last_sibling = sibling(self);
    ],
    order_yourself [ y val;
        objectloop (y ofclass AlphaSorted
                    && CompareObjects(self, y) > 0) val++;
        self.sort_ordering = val + 1;
    ];
```

The above code assumes that a routine called `CompareObjects(a,b)` exists, and returns a positive number, zero or a negative number according to whether a should come after, with or before b in lists. You could substitute any sorting rule here, but here as promised is the rule "in alphabetical order of the object's short name":

```
Array sortname1 -> 128;
Array sortname2 -> 128;
[ CompareObjects obj1 obj2 i d l1 l2;
  sortname1 --> 0 = 125; sortname2 --> 0 = 125;
  for (i = 2: i < 128: i++) { sortname1->i = 0; sortname2->i = 0; }
  @output_stream 3 sortname1; print (name) obj1; @output_stream -3;
  @output_stream 3 sortname2; print (name) obj2; @output_stream -3;
  for (i = 2: : i++) {
      l1 = sortname1->i; l2 = sortname2->i;
      d = l1 - l2; if (d) return d;
      if (l1 == 0) return 0;
  }
];
```

• 71 (p. 210)

```
parse_name [ n w colour;
    if (self.ripe) colour = 'red'; else colour = 'green';
    do { w = NextWord(); n++; } until (w ~= colour or 'fried');
    if (w == 'tomato') return n;
    return 0;
],
```

• 72 (p. 210)

```
Object -> "/?%?/ (the artiste formally known as Princess)"
  with name 'princess' 'artiste' 'formally' 'known' 'as',
       kissed false,
       short_name [;
           if (~~(self.kissed)) { print "Princess"; rtrue; }
       ],
       react_before [;
           Listen: print_ret (name) self, " sings a soft siren song.";
       ],
       initial [;
           print_ret (name) self, " is singing softly.";
       ],
       parse_name [ x n;
            if (~~(self.kissed)) {
                if (NextWord() == 'princess') return 1;
                return 0;
            }
            x = WordAddress(wn);
            if (x->0 == '/' && x->1 == '?' && x->2 == '%'
                && x->3 == '?' && x->4 == '/') {
                ! See notes below for what this next line is for:
                while (wn <= parse->1 && WordAddress(wn++) < x+5) n++;
                return n;
            }
            return -1;
       ],
       life [;
            Kiss: self.kissed = true; self.life = NULL;
                "In a fairy-tale transformation, the Princess
                steps back and astonishes the world by announcing
                that she will henceforth be known as ~/?%?/~.";
       ],
  has  animate proper female;
```

The line commented on above needs some explanation. What it does is to count up the number of "words" making up the five characters in `x->0` to `x->4`. This isn't really needed in the example above, but it would be if (say) the text to be matched was `a.,.a` because the full stops and comma would make the parser consider this text as five separate words, so that `n` should be set to 5.

• 73 (p. 210) Something to note here is that the button can't be called just "coffee" when the player's holding a cup of coffee: this means the game responds sensibly to the sequence "press coffee" and "drink coffee". Also note the call to `PronounNotice(drink)`, which tells the Inform parser that the pronouns (in English, "it", "him", "her", "them") to reset themselves to the drink where appropriate.

```
Object -> drinksmat "drinks machine",
  with name 'drinks' 'machine',
       initial
           "A drinks machine has buttons for Cola, Coffee and Tea.",
  has  static transparent;
Object -> -> thebutton "drinks machine button"
  with button_pushed,
       parse_name [ w n flag drinkword;
           for (: flag == false: n++) {
               switch (w = NextWord()) {
                   'button', 'for':
                   'coffee', 'tea', 'cola':
                       if (drinkword == 0) drinkword = w;
                   default: flag = true; n--;
               }
           }
           if (drinkword == drink.&name-->0 && n==1 && drink in player)
               return 0;
           self.button_pushed = drinkword; return n;
       ],
       before [;
           Push, SwitchOn:
               if (self.button_pushed == 0)
                   "You'll have to say which button to press.";
               if (parent(drink) ~= 0) "The machine's broken down.";
               drink.&name-->0 = self.button_pushed;
               move drink to player;
               PronounNotice(drink);
               "Whirr! The machine puts ", (a) drink,
                   " into your glad hands.";
           Attack: "The machine shudders and squirts cola at you.";
           Drink: "You can't drink until you've worked the machine.";
       ];
Object drink
  with name 'liquid' 'cup' 'of' 'drink',
       short_name [;
           print "cup of ", (address) self.&name-->0;
           rtrue;
       ],
       before [;
           Drink: remove self;
               "Ugh, that was awful. You crumple the cup and
               responsibly dispose of it.";
       ];
```

• 74 (p. 210) Most of the work is handed over to `WordInProperty(w,obj,prop)`, a routine in the library which checks to see if `w` is one of the entries in the word array `obj.&prop`, returning `true` or `false` as appropriate:

```
parse_name [ n;
    while (WordInProperty(NextWord(), self, name)) n++;
    return n;
],
```

• 75 (p. 210) Create a new property `adjective`, and move names which are adjectives to it: for instance,

```
name 'tomato' 'vegetable', adjective 'fried' 'green' 'cooked',
```

Then, again using `WordInProperty` routine as in the previous answer,

```
[ ParseNoun obj n m;
  while (WordInProperty(NextWord(),obj,adjective) == 1) n++; wn--;
  while (WordInProperty(NextWord(),obj,name) == 1) m++;
  if (m == 0) return 0; return n+m;
];
```

• 76 (p. 210)

```
[ ParseNoun obj;
  if (NextWord() == 'object' && TryNumber(wn) == obj) return 2;
  wn--; return -1;
];
```

• 77 (p. 210) Since this affects the parsing of all nouns, not just a single object, we need to set the entry point routine:

```
[ ParseNoun;
  if (NextWord() == '#//') return 1;
  wn--; return -1;
];
```

• 78 (p. 210)

```
[ ParseNoun;
  switch (NextWord()) {
      '#//': return 1;
      '*//': parser_action = ##PluralFound; return 1;
  }
  wn--; return -1;
];
```

• 79 (p. 213) This solution is a little simplified, as it only allows the player to write single words on cubes. Notice that if two cubes have the same text written on them, then they become indistinguishable from each other again.

```
Class FeaturelessCube
 with description "A perfect white cube, four inches on a side.",
      text_written_on 0 0 0 0 0 0 0 0, ! Room for 16 characters of text
      text_length,
      article "a",
      parse_name [ i j flag;
          if (parser_action == ##TheSame) {
              if (parser_one.text_length ~= parser_two.text_length)
                  return -2;
              for (i = 0: i < parser_one.text_length: i++)
                  if (parser_one.&text_written_on->i
                      ~= parser_two.&text_written_on->i) return -2;
              return -1;
          }
          for (:: i++, flag = false) {
              switch (NextWordStopped()) {
                  'cube', 'white': flag = true;
                  'featureless', 'blank':
                      flag = (self.text_length == 0);
                  'cubes': flag = true; parser_action = ##PluralFound;
                  -1: return i;
                  default:
                      if (self.text_length == WordLength(wn-1))
                          for (j=0, flag=true: j<self.text_length: j++)
                              flag = flag && (self.&text_written_on->j
                                  == WordAddress(wn-1)->j);
              }
              if (flag == false) return i;
          }
      ],
      short_name [ i;
          if (self.text_length == 0) print "featureless white cube";
          else {
              print "~";
              for (i = 0: i<self.text_length: i++)
                  print (char) self.&text_written_on->i;
              print "~ cube";
          }
          rtrue;
      ],
      plural [;
          self.short_name(); print "s";
      ];
Object -> burin "magic burin"
  with name 'magic' 'magical' 'burin' 'pen',
       description
           "This is a magical burin, used for inscribing objects with
           words or runes of magical import.",
       the_naming_word,
       before [ i;
           WriteOn:
               if (~~(second ofclass FeaturelessCube)) rfalse;
               if (second notin player)
                   "Writing on a cube is such a fiddly process that
                   you need to be holding it in your hand first.";
               if (burin notin player)
                   "You would need some powerful implement for that.";
               second.text_length = WordLength(self.the_naming_word);
               if (second.text_length > 16) second.text_length = 16;
               for (i=0: i<second.text_length: i++)
                   second.&text_written_on->i
                       = WordAddress(self.the_naming_word)->i;
               second.article="the";
               "It is now called ", (the) second, ".";
       ];
```

And this needs just a little grammar, to define the "write ... on ..." command:

```
[ AnyWord; burin.the_naming_word=wn++; return burin; ];
[ WriteOnSub; "Casual graffiti is beneath an enchanter's dignity."; ];
Verb 'write' 'scribe'
    * AnyWord 'on' held -> WriteOn
    * AnyWord 'on' noun -> WriteOn;
```

`AnyWord` is a simple example of a general parsing routine (see §31) which accepts any single word, recording its position in what the player typed and telling the parser that it refers to the burin object. Thus, text like "write pdl on cube" is parsed into the action `<WriteOn burin cube>` while `burin.the_naming_word` is set to 2.

• 80 (p. 214) This is a little more subtle than first appears, because while the warning must be printed only once, the rule allowing "cherubs" to be recognised might need to be applied many times during the same turn – for instance, if the player types "get cherubs" where there are twelve plaster cherubs, the rule must allow "cherubs" twelve times over.

```
Global cherubim_warning_turn = -1;
Class Cherub
 with parse_name [ n w this_word_ok;
          for (::) {
              w = NextWord();
              this_word_ok = false;
              if (WordInProperty(w, self, name)) this_word_ok = true;
              switch (w) {
                  'cherub': this_word_ok = true;
                  'cherubim': parser_action = ##PluralFound;
                      this_word_ok = true;
                  'cherubs':
                      if (cherubim_warning_turn == -1) {
                          cherubim_warning_turn = turns;
                          print "(I'll let this go once, but the
                              plural of cherub is cherubim.)^";
                      }
                      if (cherubim_warning_turn == turns) {
                          this_word_ok = true;
                          parser_action = ##PluralFound;
                      }
              }
              if (this_word_ok == false) return n;
              n++;
          }
      ];
```

Then again, Shakespeare wrote "cherubins" (in 'Twelfth Night'), so who are we to censure?

• 81 (p. 215) This makes use of the entry point routine, to meddle directly with the parsing table produced by the text. (See §2.4 for details of the format of this table.) Note that BeforeParsing is called *after* this table has been constructed.

```
Object -> genies_lamp "brass lamp"
  with name 'brass' 'lamp',
       colours_inverted false,
       before [;
           Rub: self.colours_inverted = ~~self.colours_inverted;
```

```
               "A genie appears from the lamp, declaring:^^
               ~Mischief is my sole delight:^
               If white means black, black means white!~^^
               She vanishes away with a vulgar wink.";
       ];
Object -> white_stone "white stone" with name 'white' 'stone';
Object -> black_stone "black stone" with name 'black' 'stone';
...
[ BeforeParsing;
   if (genies_lamp.colours_inverted)
       for (wn = 1 ::)
           switch (NextWordStopped()) {
               'white': parse-->(wn*2-3) = 'black';
               'black': parse-->(wn*2-3) = 'white';
               -1: return;
           }
];
```

• 82 (p. 221) You can fix this using the `PrintVerb` entry point routine:

```
[ PrintVerb word;
    if (word == 'go.verb') {
        if (go_verb_direction ofclass String) print "go somewhere";
        else {
            print "go to ",
                (name) real_location.(go_verb_direction.door_dir);
        }
        rtrue;
    }
    rfalse;
];
```

• 83 (p. 225) The puckish comedy of the footnote was introduced into adventure games by 'The Hitchhiker's Guide To The Galaxy'.†

```
Class Footnote with number 0, text "Text of the note.";
Footnote coinage
  with text "D.G.REG.F.D is inscribed around English coins.";
...
[ Note f fn;
```

† Not even the present author can bear to compare Douglas Adams to Edward Gibbon, so the reader is referred to Anthony Grafton's historiography *The Footnote: A Curious History* (1997).

```
  if (f.number == 0)
      objectloop (fn ofclass Footnote && fn ~= f)
          if (fn.number >= f.number)
              f.number = fn.number + 1;
  print "[", f.number, "]";
];
[ FootnoteSub fn;
  if (noun <= 0) "Footnotes count upward from 1.";
  objectloop (fn ofclass Footnote)
      if (fn.number == noun) {
          print "[", noun, "]. "; fn.text(); return;
      }
  "You haven't seen a footnote with that number.";
];
Verb meta 'footnote' 'note' * number -> Footnote;
```

And then you can code, for instance,

```
    print "Her claim to the throne is in every pocket ",
        (Note) coinage, ", her portrait in every wallet.";
```

• 84 (p. 228)

```
[ FrenchNumber n;
    switch (NextWord()) {
        'un', 'une': n=1;
        'deux': n=2;
        'trois': n=3;
        'quatre': n=4;
        'cinq': n=5;
        default: return GPR_FAIL;
    }
    parsed_number = n; return GPR_NUMBER;
];
```

• 85 (p. 228) The token is demonstrated here with a command “status team”, which lists off the team members and their current locations.

```
[ StatusSub; print "is in ", (name) parent(noun), "^"; ];
[ Team x y;
  if (NextWord() ~= 'team') return GPR_FAIL;
  objectloop (y ofclass Adventurer) multiple_object-->(++x) = y;
  multiple_object-->0 = x;
  return GPR_MULTIPLE;
];
```

```
Verb 'status' * Team -> Status;
```

• 86 (p. 228) First we must decide how to store floating-point numbers internally: in this case we'll simply store $100x$ to represent $x$, so that "5.46" will be parsed as 546. This means that we can't parse numbers larger than 327.6749999 . . . but it would be easy to store numbers differently to make room for larger ones, if we needed to.

```
[ FloatingPoint i start digits integer fraction stop n;
  integer = TryNumber(wn++);
  if (integer == -1000) return GPR_FAIL;
  switch (NextWordStopped()) {
      THEN1__WD: if (NextWordStopped() == -1) return GPR_FAIL;
           start = WordAddress(wn-1); digits = WordLength(wn-1);
           for (i = 0: i < digits: i++) {
               if (start->i < '0' || start->i > '9') return GPR_FAIL;
               if (i<3) fraction = fraction*10 + start->i - '0';
           }
      'point':
           do {
               digits++;
               switch (NextWordStopped()) {
                   -1: stop = true;
                   'nought', 'oh', 'zero': n = 0;
                   default: n = TryNumber(wn-1);
                       if (n < 0 || n > 9) { wn--; stop = true; }
               }
               if ((~~stop) && digits <= 3) fraction = fraction*10 + n;
           } until (stop);
           digits--;
           if (digits == 0) return GPR_FAIL;
      -1: ;
      default: wn--;
  }
  for (: digits < 3: digits++) fraction = fraction*10;
  parsed_number = integer*100 + (fraction+5)/10;
  return GPR_NUMBER;
];
```

Here the local variables `integer` and `fraction` hold the integer and fractional part of the number being parsed, and the last calculation performs the rounding-off to the nearest 0.01. Note that `NextWord` and `NextWordStopped` return a full stop as the constant `THEN1__WD`, since it usually plays the same grammatical role as the word "then": "east then south" and "east. south" are understood as meaning the same thing. Further exercise: with a little more code, make "oh point oh one" also work.

• 87 (p. 228) Again, the first question is how to store the number dialled: in this case, into a `string` array, and we store only the digits, stripping out spaces and hyphens. The token is:

```
Constant MAX_PHONE_LENGTH = 30;
Array dialled_number -> MAX_PHONE_LENGTH + 1;
[ PhoneNumber at length dialled dialled_already i;
  do {
      if (wn > num_words) jump number_ended;
      at = WordAddress(wn); length = WordLength(wn);
      for (i=0: i<length: i++) {
          switch (at->i) {
              '0', '1', '2', '3', '4', '5', '6', '7', '8', '9':
                  if (dialled < MAX_PHONE_LENGTH)
                      dialled_number -> (++dialled) = at->i - '0';
              '-': ;
              default: jump number_ended;
          }
      }
      wn++;
      dialled_already = dialled;
  } until (false);
 .number_ended;
  if (dialled_already == 0) return GPR_FAIL;
  dialled_number->0 = dialled_already;
  return GPR_PREPOSITION;
];
```

To demonstrate this in use,

```
[ DialPhoneSub i;
  print "You dialled <";
  for (i=1: i<=dialled_number->0: i++) print dialled_number->i;
  ">";
];
Verb 'dial' * PhoneNumber -> DialPhone;
```

• 88 (p. 228) The time of day will be returned as a number in the usual Inform time format, as hours times 60 plus minutes, where the ‘hours’ part is between 0 and 23. Which gives a value between 0 and 1439: and here is a routine to convert an hours value, a minutes value and a dictionary word which can be either `'am'` or `'pm'` into an Inform time.

```
Constant TWELVE_HOURS = 720;
```

```
[ HoursMinsWordToTime hour minute word x;
  if (hour >= 24) return -1;
  if (minute >= 60) return -1;
  x = hour*60 + minute; if (hour >= 13) return x;
  x = x%TWELVE_HOURS; if (word == 'pm') x = x + TWELVE_HOURS;
  if (word ~= 'am' or 'pm' && hour == 12) x = x + TWELVE_HOURS;
  return x;
];
```

For instance, `HoursMinsWordToTime(4,20,'pm')` returns 980, the Inform time value for twenty past four in the afternoon. The return value is −1 if the hours and minutes make no sense. Next, because the regular `TryNumber` library routine only recognises textual numbers up to `'twenty'`, we need a modest extension:

```
[ ExtendedTryNumber wordnum i j;
  i = wn; wn = wordnum; j = NextWordStopped(); wn = i;
  switch (j) {
      'twenty-one': return 21;
      ...
      'thirty': return 30;
      default: return TryNumber(wordnum);
  }
];
```

Finally the time of day token itself, which is really three separate parsing tokens in a row, trying three possible time formats:

```
[ TimeOfDay first_word second_word at length flag illegal_char
      offhour hr mn i;
  first_word = NextWordStopped();
  if (first_word == -1) return GPR_FAIL;
  switch (first_word) {
      'midnight': parsed_number = 0; return GPR_NUMBER;
      'midday', 'noon': parsed_number = TWELVE_HOURS;
          return GPR_NUMBER;
  }
  !   Next try the format 12:02
  at = WordAddress(wn-1); length = WordLength(wn-1);
  for (i=0: i<length: i++) {
      switch (at->i) {
          ':': if (flag == false && i>0 && i<length-1) flag = true;
               else illegal_char = true;
          '0', '1', '2', '3', '4', '5', '6', '7', '8', '9': ;
          default: illegal_char = true;
      }
```

```
}
if (length < 3 || length > 5 || illegal_char) flag = false;
if (flag) {
    for (i=0: at->i~=':': i++, hr=hr*10) hr = hr + at->i - '0';
    hr = hr/10;
    for (i++: i<length: i++, mn=mn*10) mn = mn + at->i - '0';
    mn = mn/10;
    second_word = NextWordStopped();
    parsed_number = HoursMinsWordToTime(hr, mn, second_word);
    if (parsed_number == -1) return GPR_FAIL;
    if (second_word ~= 'pm' or 'am') wn--;
    return GPR_NUMBER;
}
!   Lastly the wordy format
offhour = -1;
if (first_word == 'half') offhour = 30;
if (first_word == 'quarter') offhour = 15;
if (offhour < 0) offhour = ExtendedTryNumber(wn-1);
if (offhour < 0 || offhour >= 60) return GPR_FAIL;
second_word = NextWordStopped();
switch (second_word) {
    ! "six o'clock", "six"
    'o^clock', 'am', 'pm', -1:
        hr = offhour; if (hr > 12) return GPR_FAIL;
    ! "quarter to six", "twenty past midnight"
    'to', 'past':
        mn = offhour; hr = ExtendedTryNumber(wn);
        if (hr <= 0) {
            switch (NextWordStopped()) {
                'noon', 'midday': hr = 12;
                'midnight': hr = 0;
                default: return GPR_FAIL;
            }
        } else wn++;
        if (hr >= 13) return GPR_FAIL;
        if (second_word == 'to') {
            mn = 60-mn; hr--; if (hr<0) hr=23;
        }
        second_word = NextWordStopped();
    ! "six thirty"
    default:
        hr = offhour; mn = ExtendedTryNumber(--wn);
        if (mn < 0 || mn >= 60) return GPR_FAIL;
```

```
          wn++; second_word = NextWordStopped();
  }
  parsed_number = HoursMinsWordToTime(hr, mn, second_word);
  if (parsed_number < 0) return GPR_FAIL;
  if (second_word ~= 'pm' or 'am' or 'o^clock') wn--;
  return GPR_NUMBER;
];
```

True to the spirit of the Inform parser, this will also parse oddities like "quarter thirty o'clock", and we don't care.

• 89 (p. 229) Here goes: we could implement the buttons with five separate objects, essentially duplicates of each other. (And by using a class definition, this wouldn't look too bad.) But if there were 500 slides this would be less reasonable.

```
[ ASlide w n;
   if (location ~= Machine_Room) return GPR_FAIL;
   w = NextWord(); if (w == 'the' or 'slide') w = NextWord();
   switch (w) {
       'first', 'one':   n = 1;
       'second', 'two':  n = 2;
       'third', 'three': n = 3;
       'fourth', 'four': n = 4;
       'fifth', 'five':  n = 5;
       default: return GPR_FAIL;
   }
   if (NextWord() ~= 'slide') wn--;
   parsed_number = n;
   return GPR_NUMBER;
];
Array slide_settings --> 5;
[ SetSlideSub;
   slide_settings-->(noun-1) = second;
  "You set slide ", (number) noun, " to the value ", second, ".";
];
[ XSlideSub;
  "Slide ", (number) noun, " currently stands at ",
       slide_settings-->(noun-1), ".";
];
Extend 'set' first * ASlide 'to' number -> SetSlide;
Extend 'push' first * ASlide 'to' number -> SetSlide;
Extend 'examine' first * ASlide -> XSlide;
Extend 'look' first * 'at' ASlide -> XSlide;
```

• 90 (p. 229)

```
Global from_char; Global to_char;
[ QuotedText start_wn;
  start_wn = wn;
  from_char = WordAddress(start_wn);
  if (from_char->0 ~= '"') return GPR_FAIL;
  from_char++;
  do {
      if (NextWordStopped() == -1) return GPR_FAIL;
      to_char = WordAddress(wn-1) + WordLength(wn-1) - 1;
  } until (to_char >= from_char && to_char->0 == '"');
  to_char--;
  return GPR_PREPOSITION;
];
```

(The code above won't work if the user types `"foo"bar"`, though, because `"` is a word separator to Inform. It would be easy enough to compensate for this if we had to.) The text is treated as though it were a preposition, and the positions where the quoted text starts and finishes are recorded, so that an action routine can easily extract the text and use it later.

```
[ WriteOnSub i;
  print "You write ~";
  for (i = from_char: i <= to_char: i++) print (char) i->0;
  "~ on ", (the) noun, ".";
];
Verb 'write' * QuotedText 'on' noun -> WriteOn;
```

• 91 (p. 229) (See the `NounDomain` specification in §A3.) This routine passes on any `GPR_REPARSE` request, as it must, but keeps a matched object in its own `third` variable, returning the 'skip this text' code to the parser. Thus the parser never sees any third parameter.

```
Global third;
[ ThirdNoun x;
  x = ParseToken(ELEMENTARY_TT, NOUN_TOKEN);
  if (x == GPR_FAIL or GPR_REPARSE) return x;
  third = x; return GPR_PREPOSITION;
];
```

The values `GPR_MULTIPLE` and `GPR_NUMBER` can't be returned, since a $\boxed{\text{noun}}$ token – which is what the call to `ParseToken` asked for – cannot result in them.

• 92 (p. 229)

```
[ InformNumberToken n wa wl sign base digit digit_count;
  wa = WordAddress(wn);
  wl = WordLength(wn); sign = 1; base = 10; digit_count = 0;
  if (wa->0 ~= '-' or '$' or '0' or '1' or '2' or '3' or '4'
                          or '5' or '6' or '7' or '8' or '9')
      return GPR_FAIL;
  if (wa->0 == '-') { sign = -1; wl--; wa++; }
  else {
      if (wa->0 == '$') { base = 16; wl--; wa++; }
      if (wa->0 == '$') { base = 2; wl--; wa++; }
  }
  if (wl == 0) return GPR_FAIL;
  n = 0;
  while (wl > 0) {
      if (wa->0 >= 'a') digit = wa->0 - 'a' + 10;
      else digit = wa->0 - '0';
      digit_count++;
      switch (base) {
          2: if (digit_count == 17) return GPR_FAIL;
         10: if (digit_count == 6) return GPR_FAIL;
             if (digit_count == 5) {
                 if (n > 3276) return GPR_FAIL;
                 if (n == 3276) {
                     if (sign == 1 && digit > 7) return GPR_FAIL;
                     if (sign == -1 && digit > 8) return GPR_FAIL;
                 }
             }
         16: if (digit_count == 5) return GPR_FAIL;
      }
      if (digit >= 0 && digit < base) n = base*n + digit;
      else return GPR_FAIL;
      wl--; wa++;
  }
  parsed_number = n*sign; wn++; return GPR_NUMBER;
];
```

• 93 (p. 229) Add the following, as the opening lines of the `InformNumberToken` routine, which also needs a new local variable `w`:

```
  w = NextWordStopped(); if (w == -1) return GPR_FAIL;
  switch (w) {
      'true':    parsed_number = true; return GPR_NUMBER;
      'false':   parsed_number = false; return GPR_NUMBER;
      'nothing': parsed_number = nothing; return GPR_NUMBER;
      'null':    parsed_number = NULL; return GPR_NUMBER;
  }
  wn--;
```

• 94 (p. 229) Insert the following just after `wa` and `wl` have been set:

```
  if (wl == 3 && wa->0 == ''' && wa->2 == ''') {
      parsed_number = wa->1; wn++; return GPR_NUMBER;
  }
```

• 95 (p. 229) First a tricky little routine which takes as its arguments a printing rule `Rule` and a value `v`, and compares the current word against the text that would result from the statement `print (Rule) v;` (assuming that this does not exceed 126 characters). Not only that, but lower and upper case letters are allowed to match each other. The routine either sets `parsed_number` to `v` and returns `true`, if there's a match, or returns `false` if there isn't.

```
Array tolowercase -> 256;
Array attr_text -> 128;
[ TestPrintedText Rule value j k at length f addto;
  if (tolowercase->255 == 0) {
      for (j=0: j<256: j++) tolowercase->j = j;
      for (j='A',k='a': j<='Z': j++,k++) tolowercase->j = k;
  }
  attr_text-->0 = 62; @output_stream 3 attr_text;
  Rule(value);
  @output_stream -3;
  k = attr_text-->0; addto = 0;
  at = WordAddress(wn);
  length = WordLength(wn);
  if (Rule == DebugAttribute && at->0 == '~') {
      length--; at++; addto = 100;
  }
  if (k == length) {
      f = true;
      for (j=0: j<k: j++)
          if (tolowercase->(attr_text->(j+2)) ~= at->j)
              f = false;
      if (f) { parsed_number = value + addto; rtrue; }
  }
  rfalse;
];
```

Note that the special rule about ~ (which adds 100 to the value in parsed_number) is set up only to apply if attributes are being looked at. Now simply add the lines:

```
for (n=0: n<48: n++)
    if (TestPrintedText(DebugAttribute, n)) {
        wn++; return GPR_NUMBER;
    }
```

as the first lines of InformNumberToken. The routine DebugAttribute is only present if the game has been compiled with Debug mode on, but it seems very unlikely that the above code would be needed except for debugging anyway.

• 96 (p. 229) Properties are numbered from 1 (which is always name) up to some highest one, P, and the trickiest thing in answering this exercise is to find out what P is. One devious way would be to define the following object right at the very end of your source code:

```
Object with zwissler;
```

Being the final new property to be created, the value of zwissler (named after A. M. F. Zwissler, the last person in the Oxford and District telephone directory for 1998) will be P. A tidier approach is to have read, or if possible written, the *Inform Technical Manual*, which reveals that the value #identifiers_table-->0 is exactly P+1. Anyway, the actual parsing is quite like the corresponding case for attributes. Define a suitable printing rule:

```
[ PrintProperty n; print (property) n; ];
```

and then add the next round of comparisons to InformNumberToken:

```
for (n=1: n<#identifiers_table-->0: n++)
    if (TestPrintedText(PrintProperty, n)) { wn++; return GPR_NUMBER; }
```

This and the preceding five exercises, put together, are most of the way to a token which would parse any Inform expression. For a full definition of this, see the InfixRvalueTerm token in the Infix debugger's library file "Infix.h". ("Rvalue" is compiler slang for a value which can appear on the right-hand side of an = assignment.)

• 97 (p. 232) (Note that when a game is compiled with Debug mode, it already has this verb provided.) The following is slightly more useful than what the exercise asked for: it omits to mention the compass directions, making the output much less verbose. (The reason it does so is that compass directions are, for efficiency reasons, only in scope when the current reason for scope checking is PARSING_REASON, whereas because ScopeSub carries out a LoopOverScope the reason here will be LOOPOVERSCOPE_REASON.)

```
Global scope_count;
[ PrintIt obj;
  print_ret ++scope_count, ": ", (a) obj, " (", obj, ")";
];
[ ScopeSub; scope_count = 0; LoopOverScope(PrintIt);
  if (scope_count == 0) "Nothing is in scope.";
];
Verb meta 'scope' * -> Scope;
```

Under normal circumstances, "Nothing is in scope" will never be printed, as – at the very least – an actor is always in scope to himself, but since it's possible for designers to alter the definition of scope, this routine has been written to be cautious.

• 98 (p. 232)   As in the previous answer, the compass directions do not appear in the loop over scope, which neatly excludes walls, floor and ceiling:

```
[ MegaExam obj; print "^", (a) obj, ": "; <Examine obj>; ];
[ MegaLookSub; <Look>; LoopOverScope(MegaExam); ];
Verb meta 'megalook' * -> MegaLook;
```

• 99 (p. 234)   A slight refinement of such a "purloin" verb is already defined in the library (if the constant `DEBUG` is defined), so there's no need. But here's how it could be done:

```
[ Anything i;
  if (scope_stage == 1) rfalse;
  if (scope_stage == 2) {
      objectloop (i ofclass Object) PlaceInScope(i); rtrue;
  }
  "No such in game.";
];
```

(This disallows multiple matches for efficiency reasons – the parser has enough work to do with such a huge scope definition as it is.) Now the token $\boxed{\texttt{scope=Anything}}$ will match anything at all, even things like the abstract concept of 'east'. The restriction to `i ofclass Object` excludes picking up classes.

• 100 (p. 235)   For brevity, the following solution assumes that both sides of the window are lit, and that any objects with descriptions as separate paragraphs (for instance the `initial` descriptions for objects not yet moved) already describe clearly which side of the window they are on. We are only worrying about the piles of items in the "You can also see..." part of the room description. Note the sneaky way looking through the window is implemented, and that the 'on the other side' part of the room description isn't printed in that case.

```
Global just_looking_through;
Class Window_Room
 with description
```

```
            "This is one end of a long east/west room.",
        before [;
            Examine, Search: ;
            default:
              if (inp1 ~= 1 && noun ~= 0 && noun in self.far_side)
                  print_ret (The) noun, " is on the far side of
                      the glass.";
              if (inp2 ~= 1 && second ~= 0 && second in self.far_side)
                  print_ret (The) second, " is on the far side of
                      the glass.";
        ],
        after [;
            Look:
              if (just_looking_through) rfalse;
              print "^The room is divided by a great glass window,
                  stretching from floor to ceiling.^";
              if (Locale(location.far_side,
                         "Beyond the glass you can see",
                         "Beyond the glass you can also see")) ".";
        ],
    has light;
Window_Room window_w "West of Window"
    with far_side window_e;
Window_Room window_e "East of Window"
    with far_side window_w;
Object "great glass window"
    with name 'great' 'glass' 'window',
         before [ place;
             Examine, Search: place = location;
                 just_looking_through = true;
                 PlayerTo(place.far_side,1); <Look>; PlayerTo(place,1);
                 just_looking_through = false;
                 give place.far_side ~visited; rtrue;
         ],
         found_in window_w window_e,
    has  scenery;
```

A few words about `inp1` and `inp2` are in order. `noun` and `second` can hold either objects or numbers, and it's sometimes useful to know which. `inp1` is equal to `noun` if that's an object, or 1 if that's a number; likewise for `inp2` and `second`. (In this case we're just being careful that the action `SetTo eggtimer 35` wouldn't be stopped if object 35 happened to be on the other side of the glass.) We also need:

```
[ InScope actor;
```

```
    if (parent(actor) ofclass Window_Room)
        ScopeWithin(parent(actor).far_side);
    rfalse;
];
```

• 101 (p. 235) For good measure, we'll combine this with the previous rule about moved objects being in scope in the dark:

```
Object Dark_Room "Dark Room"
  with description "A disused broom cupboard.";
Object -> light_switch "light switch"
  with name 'light' 'switch',
       player_knows,
       initial "On one wall is the light switch.",
       after [;
           SwitchOn: give Dark_Room light;
           SwitchOff: give Dark_Room ~light;
       ],
  has  switchable static;
Object -> diamond "shiny diamond"
  with name 'shiny' 'diamond'
  has  scored;
Object -> dwarf "dwarf"
  with name 'voice' 'dwarf' 'breathing',
       life [;
           Order:
               if (action == ##SwitchOn && noun == light_switch) {
                   give Dark_Room light;
                   light_switch.player_knows = true;
                   StopDaemon(self);
                   if (light_switch has on) "^Typical human.^";
                   give light_switch on; "^Right you are, squire.^";
               }
       ],
       daemon [;
           if (location == thedark && real_location == Dark_Room)
               "^You hear the breathing of a dwarf.";
       ],
  has  animate;
[ InScope person i;
  if (person in Dark_Room)
      if (person == dwarf || light_switch.player_knows)
          PlaceInScope(light_switch);
  if (person == player && location == thedark)
      objectloop (i in parent(player))
          if (i has moved || i==dwarf)
              PlaceInScope(i);
  rfalse;
];
```

And in the game's Initialise routine, call StartDaemon(dwarf) to get respiration under way. Note that the routine puts the light switch in scope for the dwarf – if it didn't, the dwarf would not be able to understand "dwarf, turn light on", and that was the whole point. Note also that the dwarf can't hear the player in darkness, no doubt because of all the heavy breathing.

• 102 (p. 236) In the Initialise routine, move newplayer somewhere and Change-Player to it, where:

```
Object newplayer "yourself"
  with description "As good-looking as ever.",
       add_to_scope nose,
       capacity 5,
       orders [;
           Inv: if (nose.being_held)
                    print "You're holding your nose. ";
           Smell: if (nose.being_held)
                    "You can't smell a thing with your nose held.";
       ],
  has  concealed animate proper transparent;
Object nose "nose"
  with name 'nose', article "your",
       being_held,
       before [ possessed nonclothing;
           Take: if (self.being_held)
                    "You're already holding your nose.";
               objectloop (possessed in player)
                    if (possessed hasnt worn) nonclothing++;
               if (nonclothing > 1) "You haven't a free hand.";
               self.being_held = true; player.capacity = 1;
               "You hold your nose with your spare hand.";
           Drop: if (~~(self.being_held))
                    "But you weren't holding it!";
               self.being_held = false; player.capacity = 5;
               print "You release your nose and inhale again.  ";
               <<Smell>>;
       ],
  has  scenery;
```

• 103 (p. 236)

```
Object steriliser "sterilising machine"
  with name 'washing' 'sterilising' 'machine',
       add_to_scope top_of_wm go_button,
       before [;
           PushDir: AllowPushDir(); rtrue;
           Receive:
               if (receive_action == ##PutOn)
                   <<PutOn noun top_of_wm>>;
           SwitchOn: <<Push go_button>>;
       ],
       after [;
           PushDir: "It's hard work, but the steriliser does roll.";
       ],
       initial [;
           print "There is a sterilising machine on casters here
               (a kind of chemist's washing machine) with a ~go~
               button. ";
           if (children(top_of_wm) > 0) {
               print "On top";
               WriteListFrom(child(top_of_wm),
                   ISARE_BIT + ENGLISH_BIT);
               print ". ";
           }
           if (children(self) > 0) {
               print "Inside";
               WriteListFrom(child(self), ISARE_BIT + ENGLISH_BIT);
               print ". ";
           }
           print "^";
       ],
  has  static container open openable;
Object top_of_wm "top of the sterilising machine",
  with article "the",
  has  static supporter;
Object go_button "~go~ button"
  with name 'go' 'button',
       before [; Push, SwitchOn: "The power is off."; ],
  has  static;
```

• 104 (p. 236) The label object itself is not too bad:

```
Object -> label "red sticky label"
  with name 'red' 'sticky' 'label',
       stuck_onto nothing,
       unstick [;
           print "(first removing the label from ",
               (the) self.stuck_onto, ")^";
           self.stuck_onto = nothing;
           move self to player;
       ],
       saystuck [;
           print "^The red sticky label is stuck to ",
               (the) self.stuck_onto, ".^";
       ],
       before [;
           Take, Remove:
               if (self.stuck_onto) {
                   self.unstick(); "Taken.";
               }
           PutOn, Insert:
               if (second == self.stuck_onto)
                   "It's already stuck there.";
               if (self.stuck_onto) self.unstick();
               if (second == self) "That would only make a red mess.";
               self.stuck_onto = second; remove self;
               "You affix the label to ", (the) second, ".";
       ],
       react_before [;
           Examine: if (self.stuck_onto == noun) self.saystuck();
       ],
       react_after [ x;
           x = self.stuck_onto; if (x == nothing) rfalse;
           Look: if (IndirectlyContains(location, x) &&
                     ~~IndirectlyContains(player, x)) self.saystuck();
           Inv:  if (IndirectlyContains(player, x)) self.saystuck();
       ],
       each_turn [;
           if (parent(self)) self.stuck_onto = nothing;
       ];
```

Note that label.stuck_onto holds the object the label is stuck to, or nothing if it's unstuck: and that when it is stuck, it is removed from the object tree. It therefore has to be moved into scope, so we need the rule: if the labelled object is in scope, then so is the label.

```
Global disable_self = false;
[ InScope actor i1 i2;
  if (disable_self || label.stuck_onto == nothing) rfalse;
  disable_self = true;
  i1 = TestScope(label, actor);
  i2 = TestScope(label.stuck_onto, actor);
  disable_self = false;
  if (i1 ~= 0) rfalse;
  if (i2 ~= 0) PlaceInScope(label);
  rfalse;
];
```

This routine has two interesting points: firstly, it disables itself while testing scope (since otherwise the game would go into an endless recursion), and secondly it only puts the label in scope if it isn't already there. This is just a safety precaution to prevent the label reacting twice to actions, and isn't really necessary since the label can't already be in scope, but is included for the sake of example.

• 105 (p. 239)    Firstly, create a class `Key` of which all the keys in the game are members. Then:

```
Global assumed_key;
[ DefaultLockSub;
  print "(with ", (the) assumed_key, ")^"; <<Lock noun assumed_key>>;
];
[ DefaultLockTest i count;
  if (noun hasnt lockable) rfalse;
  objectloop (i in player && i ofclass Key) {
      count++; assumed_key = i;
  }
  if (count == 1) rtrue; rfalse;
];
Extend 'lock' first * noun=DefaultLockTest -> DefaultLock;
```

(and similar code for "unlock"). Note that "lock strongbox" is matched by this new grammar line only if the player only has one key: the `<DefaultLock strongbox>` action is generated: which is converted to, say, `<Lock strongbox brass_key>`.

• 106 (p. 240)    A neat combination of the two entry points described in this section:

```
[ ChooseObjects obj code;
  obj = obj; ! To stop Inform pointing out that obj was unused
  if (code == 1) {
      ! If the parser wants to include this object in an "all"
      ! in a faintly lit room, force it to be excluded:
      if (action_to_be == ##Take or ##Remove
          && location ofclass FaintlyLitRoom) return 2;
  }
  return 0; ! Carry on, applying normal parser rules
];
```

Since that excludes everything from an “all”, the result will be the error message “Nothing to do!”. As this is not very descriptive:

```
[ ParserError error_code;
  if (error_code == NOTHING_PE)
      ! The error message printed if an "all" turned out empty
      if (action_to_be == ##Take or ##Remove
          && location ofclass FaintlyLitRoom)
         "In this faint light, it's not so easy.";
  rfalse; ! Print standard parser error message
];
```

All this makes a room so dark that even carrying a lamp will not illuminate it fully. If this is undesirable, add the following to the start of `ChooseObjects`:

```
  objectloop (whatever in location)
      if (HasLightSource(whatever)) return 0;
```

## *Exercises in Chapter V*

• 107 (p. 255) After checking for any irregular forms, which take precedence, the `dative` routine looks to see if the last letter is “e”, as in this case it is, and then checks to see if the first six, “cyning”, form a word in the dictionary:

```
Object -> "searo"
  with name 'searo', dativename 'searwe';
Object -> "Cyning"
  with name 'cyning';
[ dative obj word a l;
  ! Irregular dative endings
  if (obj provides dativename)
      return WordInProperty(word, obj, dativename);
  ! Regular dative endings
  a = WordAddress(wn-1);
  l = WordLength(wn-1);
  if (l >= 2 && a->(l-1) == 'e') {
      word = DictionaryLookup(a, l-1);
      if (WordInProperty(word, obj, name)) rtrue;
  }
  rfalse;
];
[ dativenoun;
  if (NextWord() == 'to') return ParseToken(ELEMENTARY_TT, NOUN_TOKEN);
  wn--;
  parser_inflection = dative;
  return ParseToken(ELEMENTARY_TT, NOUN_TOKEN);
];
```

The upshot is that the game designer only has to give names in the `dativename` property if they are irregular.

• 108 (p. 255) The easiest way is to further elaborate `dativenoun`:

```
[ dativenoun it;
    switch (NextWord()) {
        'toit': it = PronounValue('it');
            if (it == NULL) return GPR_FAIL;
            if (TestScope(it, actor)) return it;
            return GPR_FAIL;
        'to': ;
        default: wn--; parser_inflection = dative;
    }
    return ParseToken(ELEMENTARY_TT, NOUN_TOKEN);
];
```

Note that it isn't safe to always allow "it" to be referred to, as "it" might be an object in another room and now out of scope. (We might want to use `ParserError` to give a better error message in this case, since at the moment we'll just get the generic "I didn't understand that sentence" message.) Another possible way for a pronoun to fail is if it remains unset. In the case of English "it", this is unlikely, but a pronoun applying only to "a group of two or more women" might well remain unset for hundreds of turns.

• 109 (p. 255) Actually one simple solution avoids fussing with tokens altogether and just adds all the possible variant forms to the names of the objects:

```
Object ... with name 'brun' 'bruna' 'hund' 'hunden';
Object ... with name 'brunt' 'bruna' 'hus' 'huset';
```

A better way is to switch between two different inflections, one for indefinite and the other for definite noun phrases. The catch is that until you start parsing, you don't know whether it will be definite or not. But you can always take a quick look ahead:

```
[ swedishnoun;
```

```
  parser_inflection = swedish_indefinite_form;
  if (NextWord() == 'den' or 'det')
      parser_inflection = swedish_definite_form;
  else wn--;
  return ParseToken(ELEMENTARY_TT, NOUN_TOKEN);
];
```

Now either write

```
Object ... with swedish_indefinite_form 'brun' 'hund',
    swedish_definite_form 'bruna' 'hunden';
```

or else `swedish_definite_form` and `swedish_indefinite_form` routines to work out the required forms automatically. (Still another way is to rewrite all indefinite forms as definite within `LanguageToInformese`.)

• **110** (p. 268)

```
[ LanguageToInformese x;
  ! Insert a space before each hyphen and after each apostrophe.
  for (x=2: x<2+buffer->1: x++) {
      if (buffer->x == '-') LTI_Insert(x++, ' ');
      if (buffer->x == ''') LTI_Insert(++x, ' ');
  }
  #ifdef DEBUG;
  if (parser_trace >= 1) {
      print "[ After LTI: '";
      for (x=2: x<2+buffer->1: x++) print (char) buffer->x;
      print "']^";
  }
  #endif;
];
```

• **111** (p. 268)    Insert the following code:

```
for (x=1: x<=parse->1: x++) {
    wn = x; word = NextWord();
    at = WordAddress(x);
    if (word == 'dessus') {
      LTI_Insert(at - buffer, ' ');
      buffer->at     = 's'; buffer->(at+1) = 'u'; buffer->(at+2) = 'r';
      buffer->(at+3) = ' '; buffer->(at+4) = 'l'; buffer->(at+5) = 'u';
      buffer->(at+6) = 'i';
      break;
    }
    if (word == 'dedans') {
      LTI_Insert(at - buffer, ' ');
      LTI_Insert(at - buffer, ' ');
      buffer->at     = 'd'; buffer->(at+1) = 'a'; buffer->(at+2) = 'n';
      buffer->(at+3) = 's'; buffer->(at+4) = ' '; buffer->(at+5) = 'l';
      buffer->(at+6) = 'u'; buffer->(at+7) = 'i';
      break;
    }
}
```

Actually, this assumes that only one of the two words will be used, and only once in the command. Which is almost certainly good enough, but if not we could replace both occurrences of break with the code:

```
    @tokenise buffer parse;
    x = 0; continue;
```

so catching even multiple usages.

• 112 (p. 268)    (This solution by Gareth Rees.)

```
[ LTI_Shift from chars i
  start   ! Where in buffer to start copying (inclusive)
  end     ! Where in buffer to stop copying (exclusive)
  dir;    ! Direction to move (1 for left, -1 for right)
  if (chars < 0) {
      start = from; end = buffer->1 + chars + 3; dir = 1;
      if (end <= start) return;
      buffer->1 = buffer->1 + chars;
  } else {
      start = buffer->1 + chars + 2; end = from + chars - 1; dir = -1;
      if (start <= end) return;
      if (start > buffer->0 + 2) start = buffer->0 + 2;
      buffer->1 = start - 2;
  }
  for (i=start: i~=end: i=i+dir) buffer->i = buffer->(i - chars);
];
```

• 113 (p. 268)    The “beware” part is handled by never breaking any word which is already within the dictionary, together with only allowing “da” and “dar” to be broken away from a dictionary word. Thus no break occurs in “davon” if there’s a person in the game called “Davon”, while no break occurs in “dartmouth” because there’s no word “tmouth” in the dictionary.

```
[ LanguageToInformese x c word at len;
```

```
  for (x=0: x<parse->1: x++) {
      word = parse-->(x*2 + 1);
      len = parse->(x*4 + 4);
      at = parse->(x*4 + 5);
      if (word == 0 && buffer->at == 'd' && buffer->(at+1) == 'a') {
          c = 2; if (buffer->(at+2) == 'r') c = 3;
          ! Is the rest of the word, after "da" or "dar", in dict?
          if (DictionaryLookup(buffer+at+c, len-c)) {
              buffer->at = ' '; buffer->(at+1) = ' ';
              if (c == 3) buffer->(at+2) = ' ';
              LTI_Insert(at+len, 's');
              LTI_Insert(at+len, 'e');
              LTI_Insert(at+len, ' ');
              break;
          }
      }
  }
];
```

Note that the text " es" is appended by inserting 's', then 'e', then a space. The routine will only make one amendment on the input line, but then only one such preposition is likely to occur on any input line, so that's all right then.

• 114 (p. 270) Most of the work goes into the printing rule for GNAs:

```
[ GNA g;
  g = GetGNAOfObject(noun);
  switch (g) {
      0,1,2,3,4,5: print "animate ";
      default: print "inanimate ";
  }
  switch (g) {
      0,1,2,6,7,8: print "singular ";
      default: print "plural ";
  }
  switch (g) {
      0,3,6,9: print "masculine";
      1,4,7,10: print "feminine";
      default: print "neuter";
  }
  print " (GNA ", g, ")";
];
[ GNASub;
  print "GNA ", (GNA) noun, "^",
        (The) noun, " / ", (the) noun, " / ", (a) noun, "^";
];
Verb meta 'gna' * multi -> GNA;
```

• 115 (p. 271)

```
Constant LanguageContractionForms = 3;
[ LanguageContraction text;
  if (text->0 == 'a' or 'e' or 'i' or 'o' or 'u'
                or 'A' or 'E' or 'I' or 'O' or 'U') return 2;
  if (text->0 == 'z' or 'Z') return 1;
  if (text->0 ~= 's' or 'S') return 0;
  if (text->1 == 'a' or 'e' or 'i' or 'o' or 'u'
                or 'A' or 'E' or 'I' or 'O' or 'U') return 1;
  return 0;
];
```

• 116 (p. 272)

```
Array LanguageArticles -->
 !   Contraction form 0:     Contraction form 1:
 !   Cdef   Def    Indef     Cdef   Def    Indef
     "Le "  "le "  "un "     "L'"   "l'"   "un "     ! 0: masc sing
     "La "  "la "  "une "    "L'"   "l'"   "une "    ! 1: fem sing
     "Les " "les " "des "    "Les " "les " "des ";   ! 2: plural
                   !             a           i
                   !             s     p     s     p
                   !             m f n m f n m f n m f n
Array LanguageGNAsToArticles --> 0 1 0 2 2 2 0 1 0 2 2 2;
```

• 117 (p. 272)

```
Array LanguageArticles -->
 ! Contraction form 0:   Contraction form 1:   Contraction form 2:
 ! Cdef   Def    Indef   Cdef   Def    Indef   Cdef   Def    Indef
   "Il "  "il "  "un "   "Lo "  "lo "  "uno "  "L'"   "l'"   "un "
   "La "  "la "  "una "  "Lo "  "lo "  "una "  "L'"   "l'"   "un'"
   "I "   "i "   "un "   "Gli " "gli " "uno "  "Gli " "gli " "un "
   "Le "  "le "  "una "  "Gli " "gli " "una "  "Le "  "le "  "un'";
                   !             a           i
                   !             s     p     s     p
                   !             m f n m f n m f n m f n
Array LanguageGNAsToArticles --> 0 1 0 2 3 0 0 1 0 2 3 0;
```

• 118 (p. 272) One contraction form, and one article set (numbered 0), in which all three articles are blank:

```
Constant LanguageContractionForms = 1;
[ LanguageContraction text; return 0; ];
Array LanguageArticles --> "" "" "";
                   !             a           i
                   !             s     p     s     p
                   !             m f n m f n m f n m f n
Array LanguageGNAsToArticles --> 0 0 0 0 0 0 0 0 0 0 0 0;
```

• 119 (p. 273)

```
Array SmallNumbersInFrench -->
  "un" "deux" "trois" "quatre" "cinq" "six" "sept" "huit"
  "neuf" "dix" "onze" "douze" "treize" "quatorze" "quinze"
  "seize" "dix-sept" "dix-huit" "dix-neuf";
[ LanguageNumber n f;
  if (n == 0)    { print "z@'ero"; rfalse; }
  if (n < 0)     { print "moins "; n = -n; }
  if (n >= 1000) { if (n/1000 ~= 1) print (LanguageNumber) n/1000, " ";
                   print "mille"; n = n%1000; f = true; }
  if (n >= 100)  { if (f) print " ";
                   if (n/100 ~= 1) print (LanguageNumber) n/100, " ";
                   print "cent"; n = n%100; f = true; }
  if (n == 0) rfalse;
  if (f) { print " "; if (n == 1) print "et "; }
  switch (n) {
      1 to 19: print (string) SmallNumbersInFrench-->(n-1);
      20 to 99:
        switch (n/10) {
           2: print "vingt";
              if (n%10 == 1) { print " et un"; return; }
           3: print "trente";
              if (n%10 == 1) { print " et un"; return; }
           4: print "quarante";
              if (n%10 == 1) { print " et un"; return; }
           5: print "cinquante";
              if (n%10 == 1) { print " et un"; return; }
           6: print "soixante";
              if (n%10 == 1) { print " et un"; return; }
           7: print "soixante";
              if (n%10 == 1) { print " et onze"; return; }
              print "-"; LanguageNumber(10 + n%10); return;
```

```
            8: if (n%10 == 0) { print "quatre vingts"; return; }
               print "quatre-vingt";
            9: print "quatre-vingt-"; LanguageNumber(10 + n%10);
               return;
         }
         if (n%10 ~= 0) { print "-"; LanguageNumber(n%10); }
   }
];
```

• 120 (p. 273)

```
[ LanguageTimeOfDay hours mins i;
  i = hours%12;
  if (i == 0) i = 12;
  if (i < 10) print " ";
  print i, ":", mins/10, mins%10;
  if (hours>= 12) print " pm"; else print " am";
];
```

• 121 (p. 277)

```
[ FrenchNominativePronoun obj;
  switch (GetGNAOfObject(obj)) {
      0, 6: print "il";  1, 7: print "elle";
      3, 9: print "ils"; 4, 10: print "elles";
  }
];
```

## *Exercises in Chapter VII*

• 122 (p. 310) The following implementation is limited to a format string $2 \times 64 = 128$ characters long, and six subsequent arguments. %d becomes a decimal number, %e an English one; %c a character, %% a (single) percentage sign and %s a string.

```
Array printed_text --> 65;
Array printf_vals --> 6;
[ Printf format p1 p2 p3 p4 p5 p6   pc j k;
  printf_vals-->0 = p1; printf_vals-->1 = p2; printf_vals-->2 = p3;
  printf_vals-->3 = p4; printf_vals-->4 = p5; printf_vals-->5 = p6;
  printed_text-->0 = 64; @output_stream 3 printed_text;
  print (string) format; @output_stream -3;
  j = printed_text-->0;
  for (k=2: k<j+2: k++) {
      if (printed_text->k == '%' && k<j+2) {
          switch (printed_text->(++k)) {
              '%': print "%";
              'c': print (char) printf_vals-->pc++;
              'd': print printf_vals-->pc++;
              'e': print (number) printf_vals-->pc++;
              's': print (string) printf_vals-->pc++;
              default: print "<** Unknown printf escape **>";
          }
      }
      else print (char) printed_text->k;
  }
];
```

• 123 (p. 311)  The is from the *Popol Vuh*, the source of Maya mythology.

```
[ TitlePage i;
   @erase_window -1; print "^^^^^^^^^^^^^";
   i = 0->33; if (i > 30) i = (i-30)/2;
   style bold; font off; spaces(i);
   print "            RUINS^";
   style roman; print "^^"; spaces(i);
   print "[Please press SPACE to begin.]^";
   font on;
   box "But Alligator was not digging the bottom of the hole"
       "Which was to be his grave,"
       "But rather he was digging his own hole"
       "As a shelter for himself."
       ""
       "-- from the Popol Vuh";
   @read_char 1 -> i;
   @erase_window -1;
];
```

• 124 (p. 311)  First put the directive `Replace DrawStatusLine;` before including the library; define the global variable `invisible_status` somewhere. Then give the following redefinition:

```
[ DrawStatusLine width posa posb;
  if (invisible_status) { @split_window 0; return; }
  @split_window 1; @set_window 1; @set_cursor 1 1; style reverse;
  width = 0->33; posa = width-26; posb = width-13;
  spaces (width);
  @set_cursor 1 2;  PrintShortName(location);
  if (width > 76) {
      @set_cursor 1 posa; print "Score: ", sline1;
      @set_cursor 1 posb; print "Moves: ", sline2;
  }
  if (width > 63 && width <= 76) {
      @set_cursor 1 posb; print sline1, "/", sline2;
  }
  @set_cursor 1 1; style roman; @set_window 0;
];
```

For simplicity this and the following answers assume that the player's visibility ceiling is always either darkness or the location: imitate the real `DrawStatusLine` in `"parserm.h"` if you need situations when the player is sealed into an opaque container.

• 125 (p. 311) First put the directive `Replace DrawStatusLine;` before including the library. Then add the following routine anywhere after `treasures_found`, an 'Advent' variable, is defined:

```
[ DrawStatusLine;
   @split_window 1; @set_window 1; @set_cursor 1 1; style reverse;
   spaces (0->33);
   @set_cursor 1 2;  PrintShortName(location);
   if (treasures_found > 0) {
       @set_cursor 1 50; print "Treasure: ", treasures_found;
   }
   @set_cursor 1 1; style roman; @set_window 0;
];
```

• 126 (p. 312) `Replace` with the following. (Note the use of `@@92` as a string escape, to include a literal backslash character.) This could be made more sophisticated by looking at the `metaclass` of `location.u_to` and so forth, but let's not complicate things:

```
Constant U_POS 28; Constant W_POS 30; Constant C_POS 31;
Constant E_POS 32; Constant I_POS 34;
[ DrawStatusLine;
  @split_window 3; @set_window 1; style reverse; font off;
  @set_cursor 1 1; spaces (0->33);
  @set_cursor 2 1; spaces (0->33);
  @set_cursor 3 1; spaces (0->33);
  @set_cursor 1 2;  print (name) location;
  @set_cursor 1 51; print "Score: ", sline1;
  @set_cursor 1 64; print "Moves: ", sline2;
  if (location ~= thedark) {
    ! First line
    if (location.u_to)   { @set_cursor 1 U_POS; print "U"; }
    if (location.nw_to)  { @set_cursor 1 W_POS; print "@@92"; }
    if (location.n_to)   { @set_cursor 1 C_POS; print "|"; }
    if (location.ne_to)  { @set_cursor 1 E_POS; print "/"; }
    if (location.in_to)  { @set_cursor 1 I_POS; print "I"; }
    ! Second line
    if (location.w_to)   { @set_cursor 2 W_POS; print "-"; }
                           @set_cursor 2 C_POS; print "o";
    if (location.e_to)   { @set_cursor 2 E_POS; print "-"; }
    ! Third line
    if (location.d_to)   { @set_cursor 3 U_POS; print "D"; }
    if (location.sw_to)  { @set_cursor 3 W_POS; print "/"; }
    if (location.s_to)   { @set_cursor 3 C_POS; print "|"; }
    if (location.se_to)  { @set_cursor 3 E_POS; print "@@92"; }
    if (location.out_to) { @set_cursor 3 I_POS; print "O"; }
  }
  @set_cursor 1 1; style roman; @set_window 0; font on;
];
```

• 127 (p. 312) The tricky part is working out the number of characters in the location name, and this is where `@output_stream` is so useful. This time `Replace` with:

```
Array printed_text --> 64;
[ DrawStatusLine i j;
  i = 0->33;
  font off;
  @split_window 1; @buffer_mode 0; @set_window 1;
  style reverse; @set_cursor 1 1; spaces(i);
  @output_stream 3 printed_text;
  print (name) location;
  @output_stream -3;
  j = (i-(printed_text-->0))/2;
  @set_cursor 1 j; print (name) location; spaces(j-1);
  style roman;
  @buffer_mode 1; @set_window 0; font on;
];
```

Note that the table can hold 128 characters (plenty for this purpose), and that these are stored in `printed_text->2` to `printed_text->129`; the length printed is held in `printed_text-->0`. ('Trinity' actually does this more crudely, storing away the width of each location name.)

• 128 (p. 314)

```
Global indent; Global d_indent = 1;
[ StartWavyMargins;
  @put_wind_prop 0 8 WavyMargins; @put_wind_prop 0 9 1;
];
[ StopWavyMargins;
  @put_wind_prop 0 8 0; @put_wind_prop 0 9 0; @set_margins 0 0 0;
];
[ WavyMargins;
  indent = indent + d_indent*10;
  if (indent == 0 or 80) d_indent = -d_indent;
  @set_margins indent indent 0;
  StartWavyMargins();
];
```

• 129 (p. 317) The following routine returns the ZSCII code of the key pressed, for good measure:

```
[ PressAnyKey k; @read_char 1 -> k; return k; ];
```

• 130 (p. 317) The keyboard is allowed only one tenth of a second to respond before an interrupt takes place, which is set to stop waiting:

```
[ KeyHeldDown k;
  @read_char 1 1 Interrupt -> k; return k;
];
[ Interrupt; rtrue; ];
```

The second 1 in the `@read_char` is the time delay: 1 tenth of a second.

• 131 (p. 317) Place the directive `Replace KeyboardPrimitive;` somewhere before any library files are included. At the end of the file, add these lines:

```
[ KeyboardPrimitive b p k;
  b->1 = 0; p->1 = 0; @aread b p 100 Hurryup -> k;
];
[ Hurryup; print "^Hurry up, please, it's time.^"; rfalse; ];
```

The number 100 represents one hundred tenths of a second, i.e., ten seconds of real time.

• 132 (p. 317) This time, we need to make use of the "terminating character", the value stored by the `@aread` opcode, which is 0 if and only if the reading was halted by an interrupt routine. If so, the command is written by hand and then tokenised using the @tokenise opcode.

```
Global reminders; Global response;
[ KeyboardPrimitive b p k;
  reminders = 0; response = b;
  response->1 = 0; p->1 = 0;
  @aread response p 50 Hurry -> k;
  if (k ~= 0) return;
  response->1 = 4; response->2 = 'l'; response->3 = 'o';
  response->4 = 'o'; response->5 = 'k';
  @tokenise b p;
];
[ Hurry;
  switch (++reminders) {
      1: print "^(Please decide quickly.)^";
      2: print "^(Further delay would be unfortunate.)^";
      3: print "^(I really must insist on a response.)^";
      4: print "^(In ten seconds I'll make your mind up for you.)^";
      6: print "^(~look~ it is, then.)^"; rtrue;
  }
  rfalse;
];
```

Note that Hurry is called every five seconds.

• 133 (p. 317) The timing is the easy part. We shall send the message run_sand to the hourglass every time the game asks the player for a command, and then every second thereafter:

```
[ KeyboardPrimitive b p k;
  hourglass.run_sand(); b->1 = 0; p->1 = 0;
  @aread b p 10 OneSecond -> k;
];
[ OneSecond; hourglass.run_sand(); rfalse; ];
```

The catch is that time spent looking at menus, or waiting to press the space bar after the interpreter prints "[MORE]", isn't registered, and besides that a turn may take more or less than 1 second to parse. So you wouldn't want to set your watch by this hourglass, but on the author's machine it keeps reasonable enough time. Here's the object:

```
Object -> hourglass "hourglass"
  with name 'hourglass',
       sand_in_top,
       when_on [;
```

```
        print "An hourglass is fixed to a pivot on the wall. ";
        switch (self.sand_in_top/10) {
            0: "The sand is almost all run through to a neat pyramid
               in the lower bulb.";
            1: "About a quarter of the sand is left in the upper
               bulb.";
            2: "The upper and lower bulbs contain roughly equal
               amounts of sand.";
            3: "About three-quarters of the sand is still to run.";
            4: "Almost all of the sand is in the upper bulb.";
        }
    ],
    when_off
        "An hourglass fixed to a pivot on the wall is turned
        sideways, so that no sand moves.",
    before [;
        Turn: if (self hasnt on) <<SwitchOn self>>;
            self.sand_in_top = 40 - self.sand_in_top;
            "You turn the hourglass the other way up.";
    ],
    after [;
        SwitchOn:
            if (self.sand_in_top < 20)
                self.sand_in_top = 40 - self.sand_in_top;
            "You turn the hourglass so that the bulb with most sand
            is uppermost, and the grains begin to pour through.";
        SwitchOff:
            "You turn the hourglass sideways.";
    ],
    run_sand [;
        if (self has on && (--(self.sand_in_top) < 0)) {
            self.sand_in_top = 40 - self.sand_in_top;
            if (self in location)
                "^The hourglass elegantly turns itself to begin
                again, all of the sand now in the uppermost bulb.";
        }
    ],
has  static switchable;
```

• 134 (p. 318)

```
Array mouse_array --> 4;
[ Main k;
```

```
  @mouse_window 0;
  for (::) {
      @read_char 1 -> k;
      @read_mouse mouse_array;
      switch(k) {
          253, 254: if (k == 253) print "Double-";
              print "Click at (", mouse_array-->0, ",",
                  mouse_array-->1, ") buttons ",
                  mouse_array-->2, "^";
      }
  }
];
```

• 135 (p. 318)  This needs another replacement of KeyboardPrimitive:

```
Zcharacter terminating 252;
Array mouse_array --> 4;
[ KeyboardPrimitive b p k s i;
  b->1 = 0; p->1 = 0;
  @aread b p -> k;
  if (k ~= 252) return;
  @read_mouse mouse_array;
  s = Do_M-->(1 + mouse_array->7);
  for (i=1: i<=s->0: i++) { b->(i+1) = s->i; print (char) s->i; }
  new_line; b->1=s->0; @tokenise b p;
];
```

And the menu itself must be created:

```
Array D1 string "Do";
Array D2 string "look";
Array D3 string "wait";
Array D4 string "inventory";
Array Do_M table D1 D2 D3 D4;
[ Initialise;
  ...
  @mouse_window 0;
  @make_menu 3 Do_M ?Able;
  "*** Unable to generate menu. ***";
 .Able;
  ...
];
```

• 136 (p. 319) By encoding one or more “characters” into an array and using @save and @restore. The numbers in this array might contain the character’s name, rank and abilities, together with some coding system to show what possessions the character has (a brass lamp, 50 feet of rope, etc.)

• 137 (p. 319) This means using a “bones file” like those generated by the game ‘Hack’. To begin with, two arrays. The array bones_file will hold a number from 0 to 9 (the number of ghosts currently present), followed by the locations of these ghosts.

```
Array bones_filename string "catacomb.bns";
Array bones_file --> 10;
Class Ghost(10)
  with short_name "ghost", plural "ghosts", name 'ghost' 'ghosts//p',
  has  animate;
```

The game’s Initialise routine should do the following:

```
  @restore bones_file 20 bones_filename -> k;
  if (k == 0) bones_file-->0 = 0;
  for (k=1: k<=bones_file-->0: k++) {
      g = Ghost.create(); move g to bones_file-->k;
  }
```

This only leaves updating the bones file in the event of death, using the entry point AfterLife:

```
[ AfterLife k;
  if (bones_file-->0 == 9) {
      for (k=2: k<=9: k++) bones_file-->(k-1) = bones_file-->k;
      bones_file-->9 = real_location;
  }
  else bones_file-->(++(bones_file-->0)) = real_location;
  @save bones_file 20 bones_filename -> k;
];
```

This code doesn’t trouble to check that the save worked properly, because it doesn’t much matter if the feature goes awry: the spirit world is not a reliable place. However, if the restore fails, the game empties the bones_file array just in case some file-handling accident on the host machine had loaded only half of the array, leaving the rest corrupt.

• 138 (p. 320) First, Replace a routine in the library called ActionPrimitive. This little routine calls all action subroutines, such as TakeSub, as and when needed. Its new version pauses to catch a stack frame:

```
[ ActionPrimitive x;
  if ((x = ExceptionHandler()) ~= 0)
```

```
      "^*** Exception: ", (string) x, " ***";
];
Global exception;
[ ExceptionHandler;
  @catch -> exception;
  indirect(#actions_table-->action);
  rfalse;
];
```

Then an action getting in trouble can simply execute a statement like:

```
@throw "Numeric overflow" exception;
```