Inform_Designers_Manual_Answers

COLLABORATORS

	TITLE : Inform_Designers_Manual_Answers	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		February 12, 2023	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Inform_Designers_Manual_Answers

1.1 Answers to all the Exercises

```
                (Note: Please read this notice.)
                                    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

Problem 1
~~~~~~~~~
Change the mushroom's after rule to:

    after
    [;  Take: if (self hasnt general)
        {   give self general;
            "You pick the mushroom, neatly cleaving its thin stalk.";
        }
        "You pick up the slowly-disintegrating mushroom.";
        Drop: "The mushroom drops to the ground, battered slightly.";
    ],

Note that general is a general-purpose attribute, always free for the
designer to use. The 'neatly cleaving' message can only happen once,
because after that the mushroom object (which calls itself self) must have
general.

Problem 2
~~~~~~~~~
    if (obj.&door_to == 0) { ... }

Problem 3
~~~~~~~~~
Put any validation rules desired into the GamePreRoutine. For example, the
following will filter out any stray Drop actions for unheld objects:
```

```
    [ GamePreRoutine;
        if (action==Drop && noun notin player)
            print_ret "You aren't holding ", (the) noun, ".";
        rfalse;
    ];
```

Problem 4
~~~~~~~~~
Declare a fake action called, say, OpenUp. Then:

```
    Object medicine "guaranteed child-proof medicine bottle" cupboard
    with name "medicine" "bottle",
        description "~Antidote only: no preventative effect.~",
        before
        [;  Open, Unlock: "It's adult-proof too.";
            OpenUp: give self open ~locked; "The bottle cracks open!";
        ],
    has  container openable locked;
```

Any other code in the game can execute <OpenUp medicine> to crack open the bottle. For brevity, this solution assumes that the bottle is always visible to the player when it is opened.

Problem 5
~~~~~~~~~
Briefly: provide a GamePreRoutine which tests to see if second is a non-zero object. If it is, then call a second_before property for the object (having already created this new property), and so on.

Problem 6
~~~~~~~~~

```
    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==0) "Cinnamon?  No, nutmeg.";
        ],
    has  scenery;
```

Directions (such as "north") are objects called n_obj, s_obj and so on: in this case, in_obj. (They are not to be confused with the property names n_to and so on.) Moreover, you can change these directions: as far as Inform is concerned, a direction is any object in the special object compass.

Problem 7
~~~~~~~~~
Define four objects along the lines of:

```
    Object white_obj "white wall" compass
    with name "white" "sac" "wall", article "the", door_dir n_to
    has  scenery;
```

and add the following line to Initialise:

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

(We could even alias a new property white_to to be n_to, and then enter map directions in the source code using Mayan property names.) 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;
```

Problem 8
~~~~~~~~~

```
    [ 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);
    ];
```

Problem 9
~~~~~~~~~

This is a prime candidate for using variable strings @nn. Briefly: at the head of the source, define

```
    Lowstring east_str "east"; Lowstring west_str "west";
```

and then add two more routines to the game,

```
    [ NormalWorld; String 0 east_str; String 1 west_str; ];
    [ ReversedWorld; String 0 west_str; String 1 east_str; ];
```

where NormalWorld is called in Initialise or to go back to normal, 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. (Inform provides up to 32 such variable strings.)

Problem 10
~~~~~~~~~~

```
    Nearby 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:
                  print_ret "The bag wriggles hideously as it swallows ",
                            (the) noun, ".";
        ],
      has  container open;
```

Problem 11
~~~~~~~~~~

```
 Object television "portable television set" lounge
```

```
   with name "tv" "television" "set" "portable",
        before
        [;  SwitchOn: <<SwitchOn power_button>>;
            SwitchOff: <<SwitchOff power_button>>;
            Examine: <<Examine screen>>;
        ],
   has  transparent;
 Nearby power_button "power button"
   with name "power" "button" "switch",
        after
        [;  SwitchOn, SwitchOff: <<Examine screen>>;
        ],
   has  switchable;
 Nearby 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.";
        ];

Problem 12
~~~~~~~~~~
 Nearby glass_box "glass box with a lid"
   with name "glass" "box" "with" "lid"
   has  container transparent openable open;
 Nearby steel_box "steel box with a lid"
   with name "steel" "box" "with" "lid"
   has  container openable open;

Problem 13
~~~~~~~~~~
(The description part of this answer uses a routine from x19, but is  only
decoration.)  Note  the  careful  use of inp1 and inp2 rather than noun or
second: see the note at the end of x4.

 Nearby macrame_bag "macrame bag"
   with name "macrame" "bag" "string" "net" "sack",
        react_before
        [;  Examine, Search, Listen, Smell: ;
            default:
                if (inp1>1 && inp1 in self)
                    print_ret (The) inp1, " is tucked away in the bag.";
                if (inp2>1 && inp2 in self)
                    print_ret (The) inp2, " is tucked away in the bag.";
        ],
        describe
        [;  print "^A macrame bag hangs from the ceiling, shut tight";
            if (child(self)==0) ".";
            print ".  Inside you can make out ";
            WriteListFrom(child(self), ENGLISH_BIT); ".";
        ],
   has  container transparent;
 Object watch "gold watch" macrame_bag
   with name "gold" "watch",
        description "The watch has no hands, oddly.",
        react_before
        [;  Listen: if (noun==0 or self) "The watch ticks loudly."; ];
```

```
Problem 14
~~~~~~~~~~
The "plank breaking" rule is implemented here in its door_to routine. Note
that this returns 'true' after killing the player.

 Nearby 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=1;
               "You step gingerly across the plank, which bows under \
                your weight. But your meagre possessions are the straw \
                which breaks the camel's back! There is a horrid crack...";
            }
            print "You step gingerly across the plank, grateful that \
                   you're not burdened.^";
            if (location==NearSide) return FarSide; return NearSide;
        ],
        door_dir
        [;  if (location==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.

Problem 15
~~~~~~~~~~
 Nearby 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.",
   has  enterable container openable open static;

Problem 16
~~~~~~~~~~
Change the car's before to

     before
     [; Go: if (noun==e_obj)
            {   print "The car will never fit through your front door.^";
                return 2;
            }
            if (car has on) "Brmm!  Brmm!";
            print "(The ignition is off at the moment.)^";
     ],

Problem 17
```

```
~~~~~~~~~~
Insert these lines into the before rule for PushDir:

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

Problem 18
~~~~~~~~~~
 Nearby bible "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
        [ w x; Consult:
                wn = consult_from; w = NextWord();
                switch(w)
                {   'matthew': x="Gospel of St Matthew";
                    'mark': x="Gospel of St Mark";
                    'luke': x="Gospel of St Luke";
                    'john': x="Gospel of St John";
                    default: "There are only the four Gospels.";
                }
                if (consult_words==1)
                    print_ret "You read the ", (string) x, " right through.";
                w = TryNumber(wn);
                if (w==-1000)
                    print_ret "I was expecting a chapter number in the ",
                              (string) x, ".";
                print_ret "Chapter ", (number) w, " of the ", (string) x,
                      " is too sacred for you to understand now.";
        ];

Problem 19
~~~~~~~~~~
Note that whether reacting before or after, the psychiatrist does not  cut
any actions short, because react_before and react_after both return false.

 Nearby 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 puts ", (name) noun, " in ",
                        (name) second, ". Interesting.~^^";
            Look: print "~Pretend I'm not here,~ says the psychiatrist.^";
        ],
        react_before
        [;  Take, Remove: print "~Subject feels lack of ", (the) noun,
                ". Suppressed Oedipal complex? Mmm.~^";
        ],
   has  animate;

Problem 20
```

~~~~~~~~~~
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 * ConTopic -> SayInstead;
 Extend "tell"   replace * ConTopic -> 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 Replace the TellSub routine (see x17) by the SayInsteadSub one.

Problem 21
~~~~~~~~~~
There are several ways to do this. The easiest is to add more grammar to the parser and let it do the hard work:

```
       Nearby computer "computer"
         with name "computer",
              orders
              [;  Theta: print_ret "~Theta now set to ", noun, ".~";
                 default: print_ret "~Please rephrase.~";
              ],
         has  talkable;
       ...
       [ ThetaSub; "You must tell your computer so."; ];
       Verb "theta" * "is" number -> Theta;
```

Problem 22
~~~~~~~~~~
Obviously, a slightly wider repertoire of actions might be a good idea, but:

```
 Nearby Charlotte "Charlotte"
   with name "charlotte" "charlie" "chas",
        grammar
        [;  give self ~general;
            wn=verb_wordnum;
            if (NextWord()=='simon' && NextWord()=='says')
            {   give self general;
                verb_wordnum=verb_wordnum+2;
            }
        ],
        orders
        [ i;  if (self hasnt general) "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;
```

Problem 23
~~~~~~~~~~
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 a local variable called i, and 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 girl she is.) ";
                  }
                  if (self.number>100)
                     "^^Charlotte is a bit out of breath now.";
                 "^^~Easy!~ says Charlotte.";
```

Problem 24
~~~~~~~~~~
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.

```
 Nearby 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
        [ i;
            Take: print_ret "~What,~ says Dan, ~ you want me to take ",
                      (the) noun, "?~";
            Drop: print_ret "~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;
```

Problem 25
~~~~~~~~~~
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.

Problem 26
~~~~~~~~~~

```
 [ PrintTime x; print (x/60), ":", (x%60)/10, (x%60)%10; ];
 Nearby alarm_clock "alarm clock"
   with name "alarm" "clock",
        number 480,
        description
        [;  print "The alarm is ";
            if (self has general) print "on, "; else print "off, but ";
            print_ret "the clock reads ", (PrintTime) the_time,
                  " and the alarm is set for ", (PrintTime) self.number, ".";
        ],
        react_after
        [;  Inv:  if (self in player)   { new_line; <<Examine self>>; }
            Look: if (self in location) { new_line; <<Examine self>>; }
        ],
        daemon
        [;  if (the_time >= self.number && the_time <= self.number+3
                && self has general) "^Beep! Beep! The alarm goes off.";
        ],
        grammar [; return 'alarm,'; ],
        orders
        [;  SwitchOn:  give self general; StartDaemon(self); "~Alarm set.~";
            SwitchOff: give self ~general; StopDaemon(self); "~Alarm off.~";
            SetTo:     self.number=noun; <<Examine self>>;
            default: "~Commands are on, off or a time of day only, pliz.~";
        ],
        life
        [;  Ask, Answer, Tell:
               "[Try ~clock, something~ to address the clock.]";
        ],
   has  talkable;
```

and add a new verb to the grammar:

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

(using the TimeOfDay token from the exercises of x23). 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.

Problem 27
~~~~~~~~~~

```
 Nearby 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
        [;  Ask, Answer, Tell: "The tricorder is too simple.";
        ],
   has  talkable;
 ...
 Verb "tc,"    * noun      -> Examine;
```

Problem 28
~~~~~~~~~~

```
 Object replicator "replicator"
   with name "replicator",
        grammar [;  return 'rc,'; ],
        orders
        [;  Give:
                print_ret "The replicator serves up a cup of ",
                    (name) noun, " which you drink eagerly.";
            default: "The replicator is unable to oblige.";
        ],
        life
        [;  Ask, Answer, Tell: "The replicator has no conversation skill.";
        ],
   has  talkable;
 Nearby earl_grey "Earl Grey tea"    with name "earl" "grey" "tea";
 Nearby brandy    "Aldebaran brandy" with name "aldebaran" "brandy";
 Nearby water     "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.

Problem 29
~~~~~~~~~~

This is similar to the previous exercises. One creates an attribute called crewmember and gives it to the crew objects: the orders property is

```
    orders
    [;  Examine:
        if (parent(noun)==0)
            print_ret "~", (name) noun,
                " is no longer aboard this demonstration game.~";
            print_ret "~", (name) noun, " is in ", (name) parent(noun), ".~";
        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: for (i=selfobj+1:i<=top_object:i++)
            if (i has crewmember) PlaceInScope(i); rtrue;
   }
 ];
 Verb "stc,"   * "where" "is" scope=Crew -> Examine;
```

An interesting point is that the scope routine doesn't need to do anything at 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.

Problem 30
~~~~~~~~~~

```
 Object Zen "Zen" Flight_Deck
   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
        [;  Show: print_ret "The main screen shows a starfield, \
                    turning through ", noun, " degrees.";
            Go:  "~Confirmed.~  The ship turns to a new bearing.";
            SetTo: if (noun==0) "~Confirmed.~  The ship comes to a stop.";
                if (noun>12) print_ret "~Standard by ", (number) noun,
                                " exceeds design tolerances.~";
                print_ret "~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.~";
            default: "~Language banks unable to decode.~";
        ],
   has  talkable proper static;
 Nearby force_wall "force wall"     with name "force" "wall" "shields";
 Nearby blasters "neutron blasters" with name "neutron" "blasters";
 ...
 Verb "zen,"   * "scan" number "orbital"        -> Show
               * "set" "course" "for" 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.

Problem 31
~~~~~~~~~~

```
       [ InScope;
          if (action_to_be == ##Examine or ##Show or ##ShowR)
              PlaceInScope(noslen_maharg);
          if (scope_reason == TALKING_REASON)
              PlaceInScope(noslen_maharg);
       ];
```

Note that ShowR is a variant form of Show in which the parameters are 'the other way round': thus "show maharg the phaser" generates ShowR maharg phaser internally, which is then converted to the more usual Show phaser maharg.

Problem 32
~~~~~~~~~~
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;
 Nearby ball "red ball" with name "red" "ball";
 Nearby 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;
                print_ret "~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.

Problem 33
~~~~~~~~~~
Just test if HasLightSource(gift)==1.

Problem 34
~~~~~~~~~~
We could solve this using a daemon, but for the sake of demonstrating a feature of thedark we won't. In Initialise, write thedark.initial = #r$GoMothGo; and add the routine:

```
       [ GoMothGo;
          if (moth in player)
          {   remove moth;
              "As your eyes try to adjust, you feel a ticklish sensation \
               and hear a tiny fluttering sound.";
          }
       ];
```

Problem 35
~~~~~~~~~~
This is a crude implementation, for brevity (the real Zork thief has an enormous stock of attached messages). A life routine is omitted, and of course this particular thief steals nothing. See 'The Thief' for a much fuller, annotated implementation.

```
 Nearby thief "thief"
   with name "thief" "gentleman" "mahu" "modo",
        each_turn "^The thief growls menacingly.",
        daemon
        [ i p j n k;
            if (random(3)~=1) rfalse;
            p=parent(thief);
            objectloop (i in compass)
            {   j=p.(i.door_dir);
                if (ZRegion(j)==1 && j hasnt door) n++;
            }
            if (n==0) rfalse;
            k=random(n); n=0;
            objectloop (i in compass)
            {   j=p.(i.door_dir);
                if (ZRegion(j)==1 && j hasnt door) n++;
                if (n==k)
                {   move self to j;
                    if (p==location) "^The thief stalks away!";
                    if (j==location) "^The thief stalks in!";
                    rfalse;
                }
            }
        ],
   has  animate;
```

ZRegion(j) works out what kind of value j is: 1 means 'is a valid object number'. So the thief walks at random but never via doors, bridges and the like (because these may be locked or have rules attached); it's only a first approximation, and in a good game one should occasionally see the thief do something surprising, such as open a secret door. As for the name, note that 'The Prince of darkness is a gentleman. Modo he's called, and Mahu' (William Shakespeare, King Lear III iv).

Problem 36

~~~~~~~~~~
First define a new property for object weight:

```
 Property weight 10;
```

(10 being an average sort of weight). Containers weigh more when they hold things, so we will need:

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

Now for the daemon which monitors the player's fatigue:

```
 Object weigher "weigher"
   with number 500,
        time_left 5,
        daemon
        [ w s b bw;
             w=WeightOf(player)-100-player.weight;
             s=self.number; s=s-w; if (s<0) s=0; if (s>500) s=500;
             self.number = s;
             if (s==0)
             {   bw=-1;
                 objectloop(b in player)
                     if (WeightOf(b)>bw) { bw=WeightOf(b); w=b; }
                 print "^Exhausted with carrying so much, you decide \
                     to discard ", (the) w, ": "; <<Drop w>>;
             }
             w=s/100; if (w==self.time_left) rfalse;
             if (w==3) print "^You are feeling a little tired.^";
             if (w==2) print "^You possessions are weighing you down.^";
             if (w==1) print "^Carrying so much weight is wearing you out.^";
             self.time_left = w;
        ];
```

Notice that items are actually dropped with Drop actions: one of them might be, say, a wild boar, which would bolt away into the forest when released. The daemon tries to drop the heaviest item. (Obviously a little improvement would be needed if the game contained, say, an un-droppable but very heavy ball and chain.) Now the daemon is going to run every turn forever, but needs to be started: so put StartDaemon(weigher); into the game's Initialise routine.

Problem 37
~~~~~~~~~~
See the next answer.

Problem 38
~~~~~~~~~~
```
 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 in the dark.";
            default: "The creatures evade you, chittering.";
        ],
        each_turn [; StartDaemon(self); ],
        number 0,
        daemon
        [ i; if (location~=thedark) { self.number=0; StopDaemon(self); rtrue; }
             i=self.number+1; self.number=i;
             switch(i)
             {   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=1;
                    "^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...";
             }
        ];
```

Problem 39
~~~~~~~~~~
Either set a daemon to watch for the_time suddenly dropping, or put such a watch in the game's TimePasses routine.

Problem 40
~~~~~~~~~~
A minimal solution is as follows:

```
 Constant SUNRISE  360;  ! i.e., 6 am
 Constant SUNSET  1140;  ! i.e., 7 pm
 Attribute outdoors;     ! Give this to external locations
 Attribute lit;          ! And this to artificially lit ones
 Global day_state = 2;
 [ TimePasses f obj;
   if (the_time >= SUNRISE && the_time < SUNSET) f=1;
   if (day_state == f) rfalse;
   for (obj=1: obj<=top_object: obj++)
   {   if (obj has lit) give obj light;
       if (obj has outdoors && obj hasnt lit)
       {   if (f==0) give obj ~light; else give obj light;
       }
   }
   if (day_state==2) { day_state = f; return; }
   day_state = f; if (location hasnt outdoors) return;
   if (f==1) "^The sun rises, illuminating the landscape!";
  "^As the sun sets, the landscape is plunged into darkness.";
 ];
```

In the Initialise routine, set the time (using SetTime) and then call

TimePasses to set all the light attributes accordingly. Note that with this system, there's no need to set light at all: that's automatic.

Problem 41
~~~~~~~~~~
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.

Problem 42
~~~~~~~~~~
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.

Problem 43
~~~~~~~~~~
Two reasons. Firstly, there are times when we want to be able to trap orders to other people, which react_before does not. Secondly, the player's react_before rule is not necessarily the first to react. In the case of the player's deafness, a cuckoo may have already used react_before to sing. But it would have been safe to use GamePreRoutine, if a little untidy (because a rule about the player would not be part of the player's definition, which makes for confusing source code). See x4 for the exact sequence of events when actions are processed.

Problem 44
~~~~~~~~~~
```
    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.";
    ],
```

Problem 45
~~~~~~~~~~
The common man's wayhel was a lowly mouse. Since we think much more highly of the player:

```
 Object hog "Warthog" Caldera
   with name "wart" "hog" "warthog", description "Muddy and grunting.",
        number 0,
        initial "A warthog snuffles and grunts about in the ash.",
        orders
        [; if (action~=##Go or ##Look or ##Examine)
               "Warthogs can't do anything as tricky as that!";
        ],
   has  animate proper;
```

and we just ChangePlayer(warthog);. Note that the same orders routine applies to the playeras-human typing "warthog, listen" as to the player-as-warthog typing just "listen".

Problem 46
~~~~~~~~~~

```
       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 is unable to comprehend your instructions.";
       ],
```

Problem 47
~~~~~~~~~~
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.

Problem 48
~~~~~~~~~~
Put the following definition between inclusion of "Parser" and "Verblib":

```
    Object LibraryMessages "lm"
        with before
            [;  Prompt: if (turns==1)
                    print "What should you, the detective, do now?^>";
                else
                    print "What next?^>";
                rtrue;
            ];
```

Problem 49
~~~~~~~~~~
The details are left to the reader. One must provide a new grammar file (generating the same actions but from different syntax) and a very large LibraryMessages object.

Problem 50
~~~~~~~~~~
Simply define the following (for accusative, nominative and capitalised nominative pronouns, respectively):

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

Problem 51
~~~~~~~~~~
Use the invent routine to signal to short_name and article routines to change their usual habits:
~~~~~~~~~~

```
        invent
        [;  if (inventory_stage==1) give self general;
            else give self ~general;
        ],
        short_name
        [;  if (self has general) { print "box"; rtrue; } ],
        article
        [;  if (self has general) { print "that hateful"; rtrue; }
            else print "a"; ],
```

Problem 52
~~~~~~~~~~
This answer is cheating, as it needs to know about the lookmode variable (set to 1 for normal, 2 for verbose or 3 for superbrief). Simply include:

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

Problem 53
~~~~~~~~~~

```
 [ DoubleInvSub i count1 count2;
   print "You are carrying ";
   objectloop (i in player)
   {   if (i hasnt worn) { give i workflag; count1++; }
       else { give i ~workflag; count2++; }
   }
   if (count1==0) print "nothing.";
   else
   WriteListFrom(child(player),
       FULLINV_BIT + ENGLISH_BIT + RECURSE_BIT + WORKFLAG_BIT);
   if (count2==0) ".";
   print ".  In addition, you are wearing ";
   objectloop (i in player)
   {   if (i hasnt worn) give i ~workflag; else give i workflag;
   }
   WriteListFrom(child(player),
       ENGLISH_BIT + RECURSE_BIT + WORKFLAG_BIT);
   ".";
 ];
```

Problem 54
~~~~~~~~~~

```
 Attribute is_letter;
 Class letter
   with list_together
        [;  if (inventory_stage==1)
            {   print "the letters ";
                if (c_style & ENGLISH_BIT == 0) c_style = c_style + ENGLISH_BIT;
                if (c_style & NOARTICLE_BIT == 0) c_style = c_style +  ↩
                   NOARTICLE_BIT;
                if (c_style & NEWLINE_BIT ~= 0) c_style = c_style - NEWLINE_BIT;
                if (c_style & INDENT_BIT ~= 0)  c_style = c_style - INDENT_BIT;
            }
            else print " from a Scrabble set";
        ],
        short_name
```
~~~~~~~~~~

```
        [;  if (listing_together has is_letter) rfalse;
            print "letter ", object self, " from a Scrabble set"; rtrue;
        ],
        article "the",
   has  is_letter;
```

and then as many letters as desired, along the lines of

```
 Nearby s1 "X" class letter with name "x";
```

Problem 55
~~~~~~~~~~

```
 Attribute is_coin;
 Class  coin_class
   with name "coin",
        description "A round unstamped disc, presumably local currency.",
        parse_name
        [ i j w;
          if (parser_action==##TheSame)
          {   if ((parser_one.&name)-->0 == (parser_two.&name)-->0) return -1;
              return -2;
          }
          w=(self.&name)-->0;
          for (::i++)
          {   j=NextWord();
              if (j=='coins') parser_action=##PluralFound;
              else if (j~='coin' or w) return i;
          }
        ],
        list_together "coins",
        plural
        [;  print (string) (self.&name)-->0;
            if (listing_together hasnt is_coin) print " coins";
        ],
        short_name
        [;  if (listing_together has is_coin)
            {   print (string) (self.&name)-->0; rtrue; }
        ],
        article
        [;  if (listing_together has is_coin) print "one"; else print "a";
        ],
   has  is_coin;
 Class  gold_coin_class   class coin_class with name "gold";
 Class  silver_coin_class class coin_class with name "silver";
 Class  bronze_coin_class class coin_class with name "bronze";
 Nearby coin1 "silver coin" class silver_coin_class;
```

... and so on

Problem 56
~~~~~~~~~~

For brevity, the following answer omits the routines: CoinsTogether(attr) which finds if the three coins with this attr (is_gold or is_silver) are together, returning 0 if they aren't and otherwise the object of which they are children; and Trigram(attr) which prints out the trigram currently showing on the coins of that attr, e.g., "Tails, Tails, Heads (Chen)".

```
Attribute is_gold; Attribute is_silver;
[ Face x; if (x.number==1) print "Heads"; else print "Tails"; ];
[ CoinsLT attr k i c;
  if (inventory_stage==1)
  {   if (attr==is_gold) print "the gold"; else print "the silver";
      print " coins ";
      k=CoinsTogether(attr);
      if (k==location)
      {   for (i=selfobj+1:i<=top_object:i++)
          {   if (i has attr)
              {   print (name) i;
                  switch(++c)
                  {  1: print ", "; 2: print " and ";
                     3: print " (showing the trigram ", (Trigram) attr, ")";
                  }
              }
          }
          rtrue;
      }
      if (c_style & ENGLISH_BIT == 0) c_style = c_style + ENGLISH_BIT;
      if (c_style & NOARTICLE_BIT == 0) c_style = c_style + NOARTICLE_BIT;
      if (c_style & NEWLINE_BIT ~= 0) c_style = c_style - NEWLINE_BIT;
      if (c_style & INDENT_BIT ~= 0)  c_style = c_style - INDENT_BIT;
  }
  rfalse;
];
Class  coin
  with number 1, article "the",
       parse_name
       [ i j w;
         if (parser_action==##TheSame) return -2;
         w='gold'; if (self has is_silver) w='silver';
         for (::i++)
         {   j=NextWord();
             if (j=='coins') parser_action=##PluralFound;
             else if (j~='coin' or w or self.name) return i;
         }
       ],
       after
       [ j;  Drop, PutOn:
                 self.number=random(2); print (Face) self, ". ";
                 if (self has is_gold) j=is_gold; else j=is_silver;
                 if (CoinsTogether(j)~=0)
                 {   print "The ";
                     if (j==is_gold) print "gold"; else print "silver";
                     print_ret " trigram is now ", (Trigram) j, ".";
                 }
                 new_line; rtrue;
       ];
Class  gold_coin class coin
 has   is_gold with list_together [; return CoinsLT(is_gold); ];
Class  silver_coin class coin
 has   is_silver with list_together [; return CoinsLT(is_silver); ];
...
Nearby goat "goat" class gold_coin with name "goat";
Nearby deer "deer" class gold_coin with name "deer";
```

```
 Nearby chicken "chicken" class gold_coin with name "chicken";
 Nearby robin "robin" class silver_coin with name "robin";
 Nearby snake "snake" class silver_coin with name "snake";
 Nearby bison "bison" class silver_coin with name "bison";
```

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 (Tui)" at inventory_stage 2 since the coins may be listed in a funny order: for example, in the order snake, robin, bison. 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.

Problem 57
~~~~~~~~~~

```
 parse_name
 [ i j w; if (self has general) j='red'; else j='green';
          w=NextWord();
          while (w==j or 'fried')
          {   w=NextWord(); i++;
          }
          if (w=='tomato') return i+1;
          return 0;
 ],
```

Problem 58
~~~~~~~~~~

```
 Nearby princess "/?%?/ (the artiste formerly known as Princess)"
   with name "princess" "artiste" "formerly" "known" "as",
        short_name
        [;   if (self hasnt general) { print "Princess"; rtrue; }
        ],
        parse_name
        [ x; if (self hasnt general)
             {   if (NextWord()=='princess') return 1;
                 return 0;
             }
             x=WordAddress(wn);
             if (   x->0 == '/' && x->1 == '?' && x->2 == '%'
                 && x->3 == '?' && x->4 == '/') return 1;
             return -1;
        ],
        react_before
        [;  Listen: if (noun==0)
                print_ret (name) self, " sings a soft siren song.";
        ],
        life
        [;  Kiss: give self general; 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;

Problem 59
~~~~~~~~~~
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
way itobj is set to the delivered drink, so that "drink it" works nicely.

 Nearby drinksmat "drinks machine",
   with name "drinks" "machine",
        initial
           "A drinks machine here has buttons for Cola, Coffee and Tea.",
   has  static;
 Nearby thebutton "drinks machine button"
   has  scenery
  with  parse_name
       [ i flag type;
            for (: flag == 0: i++)
            {   flag = 1;
                switch(NextWord())
                {   'button', 'for': flag = 0;
                    'coffee': if (type == 0) { flag = 0; type = 1; }
                    'tea':    if (type == 0) { flag = 0; type = 2; }
                    'cola':   if (type == 0) { flag = 0; type = 3; }
                }
            }
            if (type==drink.number && i==2 && type~=0 && drink in player)
                return 0;
            self.number=type; return i-1;
        ],
        number 0,
        before
        [; Push, SwitchOn:
             if (self.number == 0)
                "You'll have to say which button to press.";
             if (parent(drink) ~= 0) "The machine's broken down.";
             drink.number = self.number; move drink to player; itobj = drink;
             print_ret "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 "drink"
  with  parse_name
        [ i flag type;
            for (: flag == 0: i++)
            {   flag = 1;
                switch(NextWord())
                {   'drink', 'cup', 'of': flag = 0;
                    'coffee': if (type == 0) { flag = 0; type = 1; }
                    'tea':    if (type == 0) { flag = 0; type = 2; }
                    'cola':   if (type == 0) { flag = 0; type = 3; }
                }
            }
            if (type ~= 0 && type ~= self.number) return 0;
            return i-1;
```

```
        ],
        short_name
        [;  print "cup of ";
            switch (self.number)
            { 1: print "coffee"; 2: print "tea"; 3: print "cola"; }
            rtrue;
        ],
        number 0,
         before
         [; Drink: remove self;
             "Ugh, that was awful.  You crumple the cup and responsibly \
              dispose of it.";
         ];
```

Problem 60
~~~~~~~~~~
Create a new property adjective, and move names which are adjectives to it: for instance,

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

(Recall that dictionary words can only be written in " quotes for the name property.) Then (using the same IsAWordIn routine),

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

Problem 61
~~~~~~~~~~

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

Problem 62
~~~~~~~~~~

```
       [ ParseNoun;
         if (WordLength(wn)==1 && WordAddress(wn)-->0 == '#') return 1;
         return -1;
       ];
```

Problem 63
~~~~~~~~~~

```
       [ ParseNoun;
         if (WordLength(wn)==1 && WordAddress(wn)-->0 == '#') return 1;
         if (WordLength(wn)==1 && WordAddress(wn)-->0 == '*')
         {   parser_action = ##PluralFound; return 1; }
         return -1;
       ];
```

Problem 64
~~~~~~~~~~
The trick is to convert "fly in amber" into "fly fly amber" (a harmless name) before the parser gets under way.
~~~~~~~~~~

```
 [ BeforeParsing i j;
   for (i=parse->1,j=2:j<i:j++)
   {   wn=j-1;
       if (NextWord()=='fly' && NextWord()=='in' && NextWord()=='amber')
           parse-->(j*2-1) = 'fly';
   }
 ];
```

Problem 65
~~~~~~~~~~

```
 Global c_warned = 0;
 Class  cherub_clas
   with parse_name
        [ i j flag;
          for (flag=1:flag==1:flag=0)
          {   j=NextWord();
              if (j=='cherub' or j==self.name) flag=1;
              if (j=='cherubs' && c_warned==0)
              {   c_warned=1;
                  parser_action=##PluralFound; flag=1;
  print "(I'll let this go once, but the plural of cherub is cherubim.)^";
              }
              if (j=='cherubim')
              {   parser_action=##PluralFound; flag=1; }
              i++;
          }
          return i-1;
        ];
```

Then again, Shakespeare even writes "cherubins" in 'Twelfth Night', so who are we to censure?

Problem 66
~~~~~~~~~~

Because the parser might go on to reject the line it's working on: for instance, if the player typed "inventory splurge" then the message "Shazam!" followed by a parser complaint will be somewhat unedifying.

Problem 67
~~~~~~~~~~

Define two properties:

```
 Property place_name;
 Property to_places;
```

The scheme will work like this: a named room should have the place_name property set to a single dictionary word; say, the Bedquilt cave could be called 'bedquilt'. Then in any room, a list of those other rooms which can be moved to in this way should appear in the to_places entry. For instance,

```
 to_places Bedquilt Slab_Room Twopit_Room;
```

Now the code: see if a not-understood verb is a place name of a nearby room, and if so store that room's object number in goto_room, converting the verb to a dummy.
~~~~~~~~~~

```
Global goto_room;
[ UnknownVerb word p i;
    p = location.&to_places; if (p==0) rfalse;
    for (i=0:(2*i)<location.#to_places:i++)
        if (word==(p-->i).place_name)
        {   goto_room = p-->i; return 'go#room';
        }
    rfalse;
];
[ PrintVerb word;
    if (word=='go#room')
    { print "go to "; PrintShortName(goto_room); rtrue; }
    rfalse;
];
```

(The supplied PrintVerb is icing on the cake: so the parser can say something like "I only understood you as far as wanting to go to Bedquilt." in reply to, say, "bedquilt the nugget".) It remains only to create the dummy verb:

```
[ GoRoomSub;
    if (goto_room hasnt visited) "But you have never been there.";
    PlayerTo(goto_room);
];
Verb "go#room"  *                                   -> GoRoom;

Note that if you don't know the way, you can't go there! A purist might prefer  ↩
   instead to not
recognise the name of an unvisited room, back at the UnknownVerb stage, to avoid  ↩
   the player being
able to deduce names of nearby rooms from this 'error message'.
```

Problem 68

```
Nearby genies_lamp "brass lamp"
  with name "brass" "lamp",
       before
       [; Rub: if (self hasnt general) give self general;
               else give self ~general;
               print_ret "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 parting wink.";
       ];
Nearby white_stone "white stone" with name "white" "stone";
Nearby black_stone "black stone" with name "black" "stone";
...
[ BeforeParsing;
   if (genies_lamp hasnt general) return;
   for (wn=1::)
   {   switch(NextWordStopped())
       {   'white': parse->(wn*2-3) = 'black';
           'black': parse->(wn*2-3) = 'white';
           -1: return;
       }
   }
```

```
 ];

Problem 69
~~~~~~~~~~
 Constant MAX_FOOTNOTES 10;
 Array footnotes_seen -> MAX_FOOTNOTES;
 Global footnote_count;
 [ Note n i pn;
     for (i=0:i<footnote_count:i++)
         if (n==footnotes_seen->i) pn=i;
     if (footnote_count==MAX_FOOTNOTES) "** MAX_FOOTNOTES exceeded! **";
     if (pn==0) { pn=footnote_count++; footnotes_seen->pn=n; }
     print " [",pn+1,"]";
 ];
 [ FootnoteSub n;
     if (noun>footnote_count)
     {   print "No footnote [",noun,"] has been mentioned.^"; rtrue; }
     if (noun==0) "Footnotes count upward from 1.";
     n=footnotes_seen->(noun-1);
     print "[",noun,"]  ";
     switch(n)
     {   0: "This is a footnote.";
         1: "D.G.REG.F.D is inscribed around English coins.";
         2: "~Jackdaws love my big sphinx of quartz~, for example.";
     }
 ];
 Verb "footnote" "note" * number              -> Footnote;

 And then you can code, for instance,

     print "Her claim to the throne is in every pocket ", (Note) 1,
        ", her portrait in every wallet.";

Problem 70
~~~~~~~~~~
The general parsing routine needed is:

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

Problem 71
~~~~~~~~~~
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.

 [ DigitNumber n type x;
   x = NextWordStopped(); if (x==-1) return -1; wn--;
```

```
   if (type==0)
   {   x = WordAddress(wn);
       if (x->n>='0' && x->n<='9') return (x->n) - '0';
       return -1;
   }
   if (x=='nought' or 'oh') { wn++; return 0; }
   x = TryNumber(wn++); if (x==-1000  x>=10) x=-1; return x;
 ];
 [ FloatingPoint a x b w d1 d2 d3 type;
   a = TryNumber(wn++);
   if (a==-1000) return -1;
   w = NextWordStopped(wn); if (w==-1) return a*100;
   x = NextWordStopped(wn); if (x==-1) return -1; wn--;
   if (w=='point') type=1;
   else
   {   if (WordAddress(wn-1)->0~='.'  WordLength(wn-1)~=1)
           return -1;
   }
   d1 = DigitNumber(0,type);
   if (d1==-1) return -1;
   d2 = DigitNumber(1,type); d3 = DigitNumber(2,type);
   b=d1*10; if (d2>=0) b=b+d2; else d3=0;
   if (type==1)
   {   x=1; while (DigitNumber(x,type)>=0) x++; wn--;
   }
   else wn++;
   parsed_number = a*100 + b;
   if (d3>=5) parsed_number++;
   return 1;
 ];
```

Problem 72
~~~~~~~~~~

Again, the first question is how to store the number dialled: in this case, into a string array. The token is:

```
 Constant MAX_PHONE_LENGTH 30;
 Array dialled_number string MAX_PHONE_LENGTH;
 [ PhoneNumber f a l ch pp i;
   pp=1; if (NextWordStopped()==-1) return 0;
   do
   {   a=WordAddress(wn-1); l=WordLength(wn-1);
       for (i=0:i<l:i++)
       {   ch=a->i;
           if (ch>='0' && ch<='9')
           {   if (pp<MAX_PHONE_LENGTH) dialled_number->(pp++)=ch-'0';
           }
           else
           {   if (ch~='-') f=1; if (i~=0) return -1;   }
       }
       if (f==1)
       {   if (pp==1) return -1; dialled_number->0 = pp-1; return 0; }
   } until (NextWordStopped()==-1);
   if (pp==1) return -1;
   dialled_number->0 = pp-1;
   return 0;
 ];
```
~~~~~~~~~~

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;
```

Problem 73
~~~~~~~~~~
The time of day will be returned as a number in the usual Inform time format: as hours times 60 plus minutes (on the 24-hour clock, so that the 'hour' part is between 0 and 23).

```
 Constant TWELVE_HOURS 720;
 [ NumericTime hr mn word x;
   if (hr>=24) return -1;
   if (mn>=60) return -1;
   x=hr*60+mn; if (hr>=13) return x;
   x=x%TWELVE_HOURS; if (word=='pm') x=x+TWELVE_HOURS;
   if (word~='am' or 'pm' && hr==12) x=x+TWELVE_HOURS;
   return x;
 ];
 [ MyTryNumber wordnum i j;
   i=wn; wn=wordnum; j=NextWordStopped(); wn=i;
   switch(j)
   {   'twenty-five': return 25;
       'thirty': return 30;
       default: return TryNumber(wordnum);
   }
 ];
 [ TimeOfDay i j k flag loop ch hr mn;
i=NextWord();
switch(i)
{  'midnight': parsed_number=0; return 1;
   'midday', 'noon': parsed_number=TWELVE_HOURS; return 1;
}
!   Next try the format 12:02
j=WordAddress(wn-1); k=WordLength(wn-1);
flag=0;
for (loop=0:loop<k:loop++)
{   ch=j->loop;
    if (ch==':' && flag==0 && loop~=0 && loop~=k-1) flag=1;
    else { if (ch<'0') flag=-1; if (ch>'9') flag=-1; }
}
if (k<3) flag=0; if (k>5) flag=0;
if (flag==1)
{   for (loop=0:j->loop~=':':loop++, hr=hr*10)
        hr=hr+j->loop-'0';
    hr=hr/10;
    for (loop++:loop<k:loop++, mn=mn*10)
        mn=mn+j->loop-'0';
    mn=mn/10;
    j=NextWordStopped();
```
~~~~~~~~~~

```
    parsed_number=NumericTime(hr, mn, j);
    if (parsed_number<0) return -1;
    if (j~='pm' or 'am') wn--;
    return 1;
}
!   Next the format "half past 12"
j=-1; if (i=='half') j=30; if (i=='quarter') j=15;
if (j<0) j=MyTryNumber(wn-1); if (j<0) return -1;
if (j>=60) return -1;
k=NextWordStopped();
if (k==-1)
{   hr=j; if (hr>12) return -1; jump TimeFound; }
if (k=='o^clock' or 'am' or 'pm')
{   hr=j; if (hr>12) return -1; jump TimeFound; }
if (k=='to' or 'past')
{   mn=j; hr=MyTryNumber(wn);
    if (hr<=0)
    {   switch(NextWordStopped())
        {   'noon', 'midday': hr=12;
            'midnight': hr=0;
            default: return -1;
        }
    }
    if (hr>=13) return -1;
    if (k=='to') { mn=60-mn; hr=hr-1; if (hr==-1) hr=23; }
    wn++; k=NextWordStopped();
       jump TimeFound;
   }
   hr=j; mn=MyTryNumber(--wn);
   if (mn<0) return -1; if (mn>=60) return -1;
   wn++; k=NextWordStopped();
  .TimeFound;
   parsed_number = NumericTime(hr, mn, k);
   if (parsed_number<0) return -1;
   if (k~='pm' or 'am' or 'o^clock') wn--;
   return 1;
 ];
```

Problem 74
~~~~~~~~~~

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 -1;
    w=NextWord(); if (w=='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 -1;                !  Failure!
    }
    w=NextWord(); if (w~='slide') wn--;    !  (Leaving word counter at the
```
~~~~~~~~~~

```
                                          !  first misunderstood word)
    parsed_number=n;
    return 1;                             !  Success!
 ];
 Global slide_settings --> 5;             !  A five-word array
 [ SetSlideSub;
    slide_settings-->(noun-1) = second;
    print_ret "You set slide ", (number) noun,
              " to the value ", second, ".";
 ];
 [ XSlideSub;
    print_ret "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;
```

Problem 75
~~~~~~~~~~
(See the Parser file.) NextWord roughly returns parse-->(w*2-1) (but it worries a bit about commas and full stops).

```
 [ WordAddress w; return buffer + parse->(w*4+1); ];
 [ WordLength w; return parse->(w*4); ];
```

Problem 76
~~~~~~~~~~
(Cf. the blackboard code in _toyshop'.)

```
 Global from_char; Global to_char;
 [ QuotedText i j f;
    i = parse->((++wn)*4-3);
    if (buffer->i=='"')
    {   for (j=i+1:j<=(buffer->1)+1:j++)
            if (buffer->j=='"') f=j;
        if (f==0) return -1;
        from_char = i+1; to_char=f-1;
        if (from_char>to_char) return -1;
        while (f> (parse->(wn*4-3))) wn++; wn++;
        return 0;
    }
    return -1;
 ];
```

Note that in the case of success, the word marker wn is moved beyond the last word accepted (since the Z-machine automatically tokenises a double-quote as a single word). The text is treated as though it were a preposition, and the positions where the quoted text starts and finishes in the raw text buffer are recorded, so that an action routine can easily extract the text and use it later. (Note that "" with no text inside is not matched by this routine but only because the last if statement throws out that one case.)

Problem 77
~~~~~~~~~~
```
 [ NeverMatch; return -1; ];
```

Problem 78
~~~~~~~~~~
Perhaps to arrange better error messages when the text has failed all the ‘real’ grammar lines of a verb (see ‘Encyclopaedia Frobozzica’ for an example).

Problem 79
~~~~~~~~~~
(See the NounDomain specification in x36.) This routine passes on any REPARSE_CODE, 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=NounDomain(player,location,0);
   if (x==REPARSE_CODE) return x; if (x==0) return -1; third = x;
   return 0;
 ];
```

Problem 80
~~~~~~~~~~
```
 Global scope_count;
 [ PrintIt obj; print_ret ++scope_count, ": ", (a) obj, " (", obj, ")"; ];
 [ ScopeSub; LoopOverScope(#r$PrintIt);
   if (scope_count==0) "Nothing is in scope.";
 ];
 Verb meta "scope" *                              -> Scope;
```

Problem 81
~~~~~~~~~~
```
 [ MegaExam obj; print "^", (a) obj, ": "; <Examine obj>; ];
 [ MegaLookSub; <Look>; LoopOverScope(#r$MegaExam); ];
 Verb meta "megalook" *                           -> MegaLook;
```

Problem 82
~~~~~~~~~~
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)
   {   for (i=1:i<=top_object:i++) 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 scope=Anything will match anything at all, even things like the abstract concept of ‘east’.

Problem 83
~~~~~~~~~~
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.

```
 Property far_side;
 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 (ggw has general) rfalse;
              print "^The room is divided by a great glass window";
              if (location.far_side hasnt light) " onto darkness.";
              print ", stretching from floor to ceiling.^";
              if (Locale(location.far_side,
                     "Beyond the glass you can see",
                     "Beyond the glass you can also see")~=0) ".";
        ],
   has  light;
 Object window_w "West of Window" class window_room
   with far_side window_e;
 Object window_e "East of Window" class window_room
   with far_side window_w;
 Object ggw "great glass window"
   with name "great" "glass" "window",
        before
        [ place; Examine, Search: place=location;
                if (place.far_side hasnt light)
                    "The other side is dark.";
                give self general;
                PlayerTo(place.far_side,1); <Look>; PlayerTo(place,1);
                give self ~general;
                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 (actor in window_w && window_e has light) ScopeWithin(window_e);
    if (actor in window_e && window_w has light) ScopeWithin(window_w);
    rfalse;
 ];
```

Problem 84
~~~~~~~~~~
For good measure, we'll combine this with the previous rule about moved objects being in scope in the dark. The following can be inserted into the 'Shell' game:

```
 Object coal "dull coal" Blank_Room
   with name "dull" "coal";
 Object Dark_Room "Dark Room"
   with description "An empty room with a west exit.",
        each_turn
        [; if (self has general) self.each_turn=0;
           else "^You hear the breathing of a dwarf.";
        ],
        w_to Blank_Room;
 Nearby light_switch "light switch"
   with name "light" "switch",
        initial "On one wall is the light switch.",
        after
        [; SwitchOn: give Dark_Room light;
           SwitchOff: give Dark_Room ~light;
        ],
   has  switchable static;
 Nearby diamond "shiny diamond"
   with name "shiny" "diamond"
   has  scored;
 Nearby dwarf "dwarf"
   with name "voice" "dwarf",
        life
        [; Order: if (action==##SwitchOn && noun==light_switch)
                  {   give Dark_Room light general;
                      give light_switch on; "~Right you are, squire.~";
                  }
        ],
   has  animate;
 [ InScope person i;
   if (parent(person)==Dark_Room)
   {   if (person==dwarf  Dark_Room has general)
           PlaceInScope(light_switch);
   }
   if (person==player && location==thedark)
       objectloop (i near player)
           if (i has moved  i==dwarf)
               PlaceInScope(i);
   rfalse;
 ];
```

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.

Problem 85
~~~~~~~~~~

```
~~~~~~~~~~
In the Initialise routine, move newplay somewhere and ChangePlayer to  it,
where:

 Object newplay "yourself"
   with description "As good-looking as ever.", number 0,
        add_to_scope nose,
        capacity 5,
        before
        [;  Inv: if (nose has general) print "You're holding your nose.  ";
            Smell: if (nose has general)
                      "You can't smell a thing with your nose held.";
        ],
   has  concealed animate proper transparent;
 Object nose "nose"
   with name "nose", article "your",
        before
        [; Take: if (self has general)
                    "You're already holding your nose.";
                 if (children(player) > 1) "You haven't a free hand.";
                 give self general; player.capacity=1;
                "You hold your nose with your spare hand.";
           Drop: if (self hasnt general) "But you weren't holding it!";
                 give self ~general; player.capacity=5;
                 print "You release your nose and inhale again.  ";
                 <<Smell>>;
        ],
   has  scenery;

Problem 86
~~~~~~~~~~
 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 ".  ";
            }
```

```
        ],
   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;

Problem 87
~~~~~~~~~~
The label object itself is not too bad:

 Nearby label "red sticky label"
   with name "red" "sticky" "label",
        number 0,
        before
        [;  PutOn, Insert:
                if (self.number~=0)
                {   print "(first removing the label from ",
                    (the) self.number, ")^"; self.number=0; move self to player;
                }
                if (second==self) "That would only make a red mess.";
                self.number=second; remove self;
                print_ret "You affix the label to ", (the) second, ".";
        ],
        react_after
        [ x; x=self.number; if (x==0) rfalse;
            Look: if (x in location)
                    print "^The red sticky label is stuck to ", (the) x, ".^";
            Inv:  if (x in player)
                    print "^The red sticky label is stuck to ", (the) x, ".^";
        ],
        each_turn
        [;  if (parent(self)~=0) self.number=0; ];
```

Note that label.number holds the object the label is stuck to, or 0 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;
 [ InScope actor i1 i2;
   if (label.number==0) rfalse; if (disable_self==1) rfalse;
   disable_self=1;
   i1 = TestScope(label, actor);
   i2 = TestScope(label.number, actor);
   disable_self=0;
   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).

Problem 88
~~~~~~~~~~
Firstly, create an attribute is_key and give it to all the keys in the game. 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)
       if (i has is_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.

Problem 89
~~~~~~~~~~
```
 Array quote_done -> 50;
 Global next_quote = -1;
 [ Quote i;
   if (quote_done->i==0) { quote_done->i = 1; next_quote = i; }
 ];
 [ AfterPrompt;
   switch(next_quote)
   {   0: box "His stride is wildernesses of freedom:"
              "The world rolls under the long thrust of his heel."
              "Over the cage floor the horizons come."
              ""
              "-- Ted Hughes, ~The Jaguar~";
       1: ...
   }
   next_quote = -1;
 ];
```

Problem 90
~~~~~~~~~~
Note the magic line of assembly code here, which only works for Advanced games:

```
 [ GiveHint hint keypress;
   print (string) hint; new_line; new_line;
   @read_char 1 0 0 keypress;
   if (keypress == 'H' or 'h') rfalse;
   rtrue;
 ];
```
~~~~~~~~~~

And a typical menu item using it:

```
 if (menu_item==1)
 {   print "(Press ENTER to return to menu, or H for another hint.)^^";
     if (GiveHint("(1/3)  What kind of bird is it, exactly?")==1) return 2;
     if (GiveHint("(2/3)  Magpies are attracted by shiny items.")==1) return 2;
     "(3/3)  Wave at the magpie with the kitchen foil.";
 }
```

Problem 91
~~~~~~~~~~

By encoding the character into a byte 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.)

Problem 92
~~~~~~~~~~

Note that we wait for a space character (32) or either kind of new-line which typical ASCII keyboards produce (10 or 13), just to be on the safe side:

```
[ TitlePage i;
   @erase_window -1; print "^^^^^^^^^^^^^^";
   i = 0->33; if (i==0) i=80; i=(i-50)/2;
   style bold; font off; spaces(i);
   print "                     RUINS^";
   style roman; print "^^"; spaces(i);
   print "         [Please press SPACE to begin.]^";
   font on;
   box "And make your chronicle as rich with praise"
       "As is the ooze and bottom of the sea"
       "With sunken wreck and sumless treasures."
       ""
       "-- William Shakespeare, ~Henry V~ I. ii. 163";
   do { @read_char 1 0 0 i; } until (i==32 or 10 or 13);
   @erase_window -1;
];
```

Problem 93
~~~~~~~~~~

First put the directive Replace DrawStatusLine; before including the library; define the global variable invisible_status somewhere. Then give the following redefinition:

```
[ DrawStatusLine i width posa posb;
   if (invisible_status==1) return;
   @split_window 1; @set_window 1; @set_cursor 1 1; style reverse;
   width = 0->33; posa = width-26; posb = width-13;
   spaces (width-1);
   @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;
 ];
```

Problem 94
~~~~~~~~~~
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)-1;
    @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;
 ];
```

Problem 95
~~~~~~~~~~
Replace with the following. (Note the use of @@92 as a string escape, to include a literal backslash character, and @@124 for a vertical line.)

```
 Constant U_POS 28; Constant W_POS 30; Constant C_POS 31;
 Constant E_POS 32; Constant IN_POS 34;
 [ DrawStatusLine i;
     @split_window 3; @set_window 1; style reverse; font off;
     @set_cursor 1 1; spaces (0->33)-1;
     @set_cursor 2 1; spaces (0->33)-1;
     @set_cursor 3 1; spaces (0->33)-1;
     @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 ~= 0)  { @set_cursor 1 U_POS; print "U"; }
         if (location.nw_to ~= 0) { @set_cursor 1 W_POS; print "@@92"; }
         if (location.n_to ~= 0)  { @set_cursor 1 C_POS; print "@@124"; }
         if (location.ne_to ~= 0) { @set_cursor 1 E_POS; print "/"; }
         if (location.in_to ~= 0) { @set_cursor 1 IN_POS; print "I"; }
         ! Second line
         if (location.w_to ~= 0)  { @set_cursor 2 W_POS; print "-"; }
                                    @set_cursor 2 C_POS; print "o";
         if (location.e_to ~= 0)  { @set_cursor 2 E_POS; print "-"; }
         ! Third line
         if (location.d_to ~= 0)  { @set_cursor 3 U_POS; print "D"; }
         if (location.sw_to ~= 0) { @set_cursor 3 W_POS; print "/"; }
         if (location.s_to ~= 0)  { @set_cursor 3 C_POS; print "@@124"; ]
         if (location.se_to ~= 0) { @set_cursor 3 E_POS; print "@@92"; }
         if (location.out_to ~= 0){ @set_cursor 3 IN_POS; print "O"; }
     }
     @set_cursor 1 1; style roman; @set_window 0; font on;
 ];
```

Problem 96

```
~~~~~~~~~~
```
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 table 64;
 [ DrawStatusLine i j;
   i = 0->33; if (i==0) i=80;
   font off;
   @split_window 1; @buffer_mode 0; @set_window 1;
   style reverse; @set_cursor 1 1; spaces(i);
   printed_text-->0 = 64;
   @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.)

Problem 97

```
~~~~~~~~~~
```
The following implementation is limited to a format string 2 x 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 table 64;
 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 == '%')
       {   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;
   }
 ];
```

Problem 98

```
~~~~~~~~~~
Primes(100), where:

 [ Primes i j k l;
   for (j=2:j<=i:j++)
   {   print j, " : "; l=j;
       while (l > 1)
       for (k=2:k<=l:k++)
           if (l%k == 0) { l=l/k; print k, " "; break; }
       new_line;
   }
 ];
```

(which was the first algorithm ever compiled by Inform).

1.2 Notes

Please note that there may be some slight lexigraphical errors in this file. If there are, these are NOT the fault of the author, Graham Nelson; but, rather, are due to the TeX-to-Text conversion utility I used.

This program converted all {s into -s, and }s into "s, and \s also into "s, and also ~s into "s... so, you see, I had to do a lot of manual hunt-and-pecking in this thing. I hope I've fixed them all, but I more than likely missed a few. So, please remain aware of these possible errors as you're reading these examples.

-- Steven Parradee
   22 November 1995