1. Intro. This program finds all solutions to Don Woods's Twenty Questions, having a given pattern of true/false grades, assuming that at least 18 answers will be correct. The command line lists two answers that must be incorrect, if nonzero.

I've tried to write this in a straightforward way that will avoid errors in my logic, yet leave it flexible enough to experiment with slight tweaks to the problem.

```
#define A 0
#define B 1
#define C 2
#define D 3
#define E 4
#define AA (1 ≪ A)
#define BB (1 ≪ B)
#define CC (1 ≪ C)
#define DD (1 ≪ D)
#define EE (1 ≪ E)
#define o mems++
#define oo mems += 2
#define stacksize 1000000 /* I think 20 × 20 × 6 is actually an upper bound */
#define delta 10000000000 /* print status every this many mems */
#include <stdio.h>
#include <stdlib.h>
  typedef unsigned long long ull;
  ull mems; /* memory references */
  ull nodes; /* size of search tree */
  ull count; /* solutions found */
  ull thresh = delta; /* time to print next report */
  ull profile[22];
  int false1, false2; /* command-line parameters */
  int score; /* the score of each solution that is found */
  char falsity[21]; /* is this answer to be false or not? */
  char believe3; /* is falsity[3] zero? */
  char mem[41]; /* status subject to backtracking */
  char vbose; /* print lots of debugging info? */
  ⟨ Other global variables 6 ⟩;
  ⟨ Subroutines 7 ⟩;
  main(int argc, char *argv[])
  {
    register int i, j, k, l, p, q, t, u, x, y, really_bad;
    ⟨ Process the command line 3 ⟩;
    ⟨ Set the initial constraints 5 ⟩;
    ⟨ Backtrack through all possibilities 12 ⟩;
  done:
    fprintf(stderr, "Altogether␣%llu␣solutions␣(%llu␣mems,␣%llu␣nodes).\n", count, mems, nodes);
    if (vbose) ⟨ Print the profile 2 ⟩;
  }
```

2. ⟨ Print the profile 2 ⟩ ≡

```
{
  fprintf(stderr, "Profile:␣␣␣␣␣␣␣␣␣␣1\n");
  for (k = 2; k ≤ 21; k++) fprintf(stderr, "%19lld\n", profile[k]);
}
```

This code is used in section 1.

3. ⟨ Process the command line 3 ⟩ ≡

```
if (argc < 3 ∨ sscanf(argv[1], "%d", &false1) ≠ 1 ∨ sscanf(argv[2], "%d", &false2) ≠ 1) {
  fprintf(stderr, "Usage:␣%s␣false1␣false2␣[verbose]\n", argv[0]);
  exit(−1);
}
score = 20;
if (false1 > 0 ∧ false1 ≤ 20) falsity[false1] = 1, score−−;
if (false2 > 0 ∧ false2 ≤ 20 ∧ false2 ≠ false1) falsity[false2] = 1, score−−;
believe3 = ¬falsity[3];
vbose = argc − 3;      /* extra arguments are ignored but they increase verbosity */
if (falsity[6]) {
  if (vbose) fprintf(stderr, "Question␣6␣can't␣be␣wrong.\n");
  goto done;
}
```

This code is used in section 1.

4. The strategy. All the critical data about partial information is kept in a small array called *mem*, consisting of 8-bit quantities. Locations *mem*[1] thru *mem*[20] are bitmaps that show the nonexcluded possibilities for each of the 20 answers. (For example, the bitmap `AA + DD` means that A and D are still possible.) The next 20 locations hold tag information, which is nonzero if the answer for that question should be doublechecked at level 21. (We catch easy inconsistencies early if it's convenient, but defer the complicated tests.)

When the entry in *mem*[*p*] changes from *a* to *b*, we put $(p \ll 8) + a$ on the stack so that the old value can be restored later. The number of items on the stack upon entry to level *l* is stored in *frame*[*l*].

I thought about maintaining upper and lower bounds for the number of A's, B's, C's, D's, E's. But that seemed prone to error, and experiments by hand indicated that reasonable cutoffs were possible without such fancy footwork.

```
#define tag 20 /* offset for tag data */
```

5. Initially all of the answers are unrestricted.

```
⟨ Set the initial constraints 5 ⟩ ≡
  for (q = 1; q ≤ 20; q++) o, mem[q] = AA + BB + CC + DD + EE;
```

This code is used in section 1.

6. Some of the questions are easy to deal with near the root of the search tree, so I've chosen a heuristic order in which to build the partial solutions. This ordering remains fixed; thus, for example, I'll know that when I'm working on question 1, I've already given answers to several others including questions 2 and 3. It's nice to put question 3 first, because it usually rules out many possibilities immediately: If that answer is supposed to be correct, no two consecutive answers can match, with the exception of *loguy* and *higuy*, which are defined at root level.

```
⟨ Other global variables 6 ⟩ ≡
  int order[21] = {0, 3, 15, 20, 19, 2, 1, 17, 10, 5, 4, 16, 11, 13, 14, 7, 18, 6, 8, 12, 9};
  int loguy, higuy;
  int rho[32] = {−1, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0,
      4, 0, 1, 0, 2, 0, 1, 0, 3, 0, 1, 0, 2, 0, 1, 0}; /* the ruler function for 5-bit numbers */
  int stack[stacksize], frame[21]; /* for undoing changes to mem */
  int stackptr; /* the current number of elements in stack */
  int pstack[21]; /* for propagating forced changes */
  int pstackptr; /* the current number of elements in pstack */
```

See also sections 31 and 52.

This code is used in section 1.

7. Here's a subroutine that illustrates the conventions. If option x is presently available for question q, it is removed and the residual bitmap is returned.

Furthermore, if question 3 is not falsified, the question number is placed on *pstack* whenever the bitmap has been reduced to exactly one bit.

```
⟨ Subroutines 7 ⟩ ≡
  int remov(int q, int x)
  {
    register int t, b, bb;
    t = 1 ≪ x;
    o, b = mem[q];
    if (b & t) {
      if (vbose > 2) fprintf(stderr, "(%d: not %d%c)\n", stackptr, q, x + (falsity[q] ? 'a' : 'A'));
      o, mem[q] = bb = b − t;
      o, stack[stackptr++] = (q ≪ 8) + b;
      if (believe3 ∧ bb ∧ ¬(bb & (bb − 1))) o, pstack[pstackptr++] = q;
      return bb;
    }
    return b;
  }
```

See also sections 8, 9, 10, and 11.

This code is used in section 1.

8. The *innerforce* routine throws out all options *but* x.

```
⟨ Subroutines 7 ⟩ +≡
  int innerforce(int q, int x)
  {
    register int t, b;
    t = 1 ≪ x;
    o, b = mem[q];
    if ((b & t) ≡ 0) return 0;
    if (b ≠ t) {
      if (vbose > 2) fprintf(stderr, "(%d: %d%c)\n", stackptr, q, x + (falsity[q] ? 'a' : 'A'));
      o, mem[q] = t;
      o, stack[stackptr++] = (q ≪ 8) + b;
      if (believe3) o, pstack[pstackptr++] = q;
    }
    return t;
  }
```

9. I'll need some help when debugging.

```
⟨ Subroutines 7 ⟩ +≡
  void print_state(void)
  {
    register int b, c, q, z;
    for (q = 1; q ≤ 20; q++)
      if (q < 10) fprintf(stderr, "%d␣␣␣␣", q);
      else fprintf(stderr, "%d␣␣␣", q);
    printf("\n");
    for (q = 1; q ≤ 20; q++) {
      for (b = mem[q], c = z = 0; c < 5; c++) {
        if (b & (1 ≪ c)) fprintf(stderr, "%c", c + (falsity[q] ? 'a' : 'A'));
        else z++;
      }
      if (mem[q + tag]) z−−, fprintf(stderr, "*");
      for (c = 0; c < z; c++) fprintf(stderr, "␣");
    }
    printf("\n");
  }
```

10.

```
⟨ Subroutines 7 ⟩ +≡
  void print_cols(void)
  {
    register int c, q;
    for (c = 0; c < 5; c++) {
      fprintf(stderr, "%c:", 'A' + c);
      for (q = 1; q ≤ 20; q++)
        if (mem[q] & (1 ≪ c)) fprintf(stderr, "␣%d", q);
      fprintf(stderr, "\n");
    }
  }
```

11.

```
⟨ Subroutines 7 ⟩ +≡
  void print_stack(int p)
  {
    register int i, l, b, ii;
    for (i = 0, l = 1; i < p; i++) {
      ii = stack[i] ≫ 8;
      fprintf(stderr, "mem[%d%s]:", ii > 20 ? ii − tag : ii, ii > 20 ? "*" : "");
      for (b = 0; b < 5; b++)
        if (stack[i] & (1 ≪ b)) fprintf(stderr, "%c", b + 'A');
      while (i ≡ frame[l]) {
        fprintf(stderr, "␣l%dq%d%s", l, order[l], falsity[l] ? "-" : "");
        l++;
      }
      fprintf(stderr, "\n");
    }
  }
```

12. Backtracking. Here's the environment/context in which I'll embed the logic for individual questions.

```
#define pack(u,q,x) (((u) ≪ 8) + ((q) ≪ 3) + (x))
⟨ Backtrack through all possibilities 12 ⟩ ≡
b1: l = 1, stackptr = 0;
b2: nodes++, profile[l]++;
  if (mems ≥ thresh) ⟨ Print a status report and reset thresh 14 ⟩;
  if (l > 20) ⟨ Check for solution and goto b5 16 ⟩;
  oo, q = order[l], u = falsity[q];
  o, y = mem[q];
b3: if (y ≡ 0) {
    fprintf(stderr, "I'm␣confused!\n"); exit(−69);
  }
  p = stackptr;
  o, x = rho[y];
  pstackptr = 0;
  if (vbose > 1) fprintf(stderr, "Level␣%d(%d),␣trying␣%d%c\n", l, p, q, x + (u ? 'a' : 'A'));
  switch (pack(u, q, x)) {
    ⟨ Cases for the big switch 19 ⟩
  default: fprintf(stderr, "Impossible␣case␣%d%c!\n", q, x + (u ? 'a' : 'A'));
    exit(−30);
  }
defer: oo, mem[tag + q] = 1, stack[stackptr++] = (q + tag) ≪ 8;    /* must check condition later */
  if (vbose > 1) fprintf(stderr, "(%d:␣deferring␣%d%c)\n", stackptr − 1, q, x + (u ? 'a' : 'A'));
okay: force(q, x);
  ⟨ Propagate forced consequences, possibly going to bad 15 ⟩;
  o, frame[l] = p;
  l++; goto b2;
bad: if (really_bad) {
    really_bad = 0;    /* we can't choose x nor can we not choose it */
    goto b5;
  }
  while (stackptr > p) ⟨ Restore saved state 13 ⟩;
b4: really_bad = 1;    /* we've found that x isn't a good answer for question q */
  pstackptr = 0;
  deny(q, x);    /* this will backtrack if x was our last choice for q */
  ⟨ Propagate forced consequences, possibly going to bad 15 ⟩;
  really_bad = 0;
  y −= 1 ≪ x;    /* now (it says here) y should equal mem[q] */
  if (y ≡ mem[q]) goto b3;
  fprintf(stderr, "I␣screwed␣up!\n"); exit(−666);
b5: if (−−l) {
    oo, q = order[l], u = falsity[q];
    o, p = frame[l];
    while (stackptr > p) ⟨ Restore saved state 13 ⟩;
    oo, y = mem[q], x = rho[y];
    goto b4;
  }
```

This code is used in section 1.

13. ⟨ Restore saved state 13 ⟩ ≡

```
{
  o, t = stack[--stackptr];
  o, mem[t ≫ 8] = t & #1f;
}
```

This code is used in section 12.

14. ⟨ Print a status report and reset *thresh* 14 ⟩ ≡

```
{
  fprintf(stderr, "After␣%llu␣mems,␣l=%d,␣stackptr=%d\n", mems, l, stackptr);
  print_state( );
  thresh += delta;
}
```

This code is used in section 12.

15. All changes to *mem*[1] thru *mem*[20] are made by the *remov* and *innerforce* routines. Macros *deny* and *force* are used to ensure that those routines don't remove an answer's final option.

If question 3 hasn't been falsified on the command line, we can't have consecutive equal answers except in certain cases. This means that one change to *mem* can propagate to its neighbors. For example, suppose the remaining choices for answers 5, 6, 7, 8, 9 are respectively CD, AC, BD, ABE, BCD. Then if answer 8 is forced to be B, answer 7 can only be D; hence answer 6 is also forced to be A. Also answer 9 can no longer be B. Such propagations are handled by the simple *pstack* mechanism implemented here.

```
#define deny(q, x)
        { if (¬remov(q, x)) goto bad;
          if (q ≡ loguy ∧ ¬remov(higuy, x)) goto bad;
          if (q ≡ higuy ∧ ¬remov(loguy, x)) goto bad; }
#define force(q, x)
        { if (¬innerforce(q, x)) goto bad; }
```

⟨ Propagate forced consequences, possibly going to *bad* 15 ⟩ ≡

```
while (pstackptr) {
  o, t = pstack[--pstackptr];
  oo, j = rho[mem[t]];
  if (vbose > 3) fprintf(stderr, "(propagating␣from␣%d%c)\n", t, j + 'A');
  if (t ≡ loguy) force(t + 1, j)
  else if (t < 20) deny(t + 1, j);
  if (t ≡ higuy) force(t − 1, j)
  else if (t > 1) deny(t − 1, j);
}
```

This code is used in section 12.

16. Finally all the deferred tests must be made.

```
⟨ Check for solution and goto b5 16 ⟩ ≡
  {
    ⟨ Compute the distribution of answers 30 ⟩;
    for (q = 1; q ≤ 20; q++)
      if (o, mem[tag + q]) {
        oo, x = rho[mem[q]];
        o, u = falsity[q];
        if (vbose > 1) fprintf(stderr, "Checking␣%d%c\n", q, x + (u ? 'a' : 'A'));
        switch (pack(u, q, x)) {
          ⟨ Cases for the deferred switch 20 ⟩
        default: fprintf(stderr, "Impossible␣deferred␣case␣%d%c!\n", q, x + (u ? 'a' : 'A'));
          exit(−31);
        }
      }
    ⟨ Print a solution 17 ⟩;
    goto b5;
  }
```

This code is used in section 12.

17. ⟨ Print a solution 17 ⟩ ≡

```
  count++;
  printf("%lld:␣", count);
  for (q = 1; q ≤ 20; q++) printf("%c", (falsity[q] ? 'a' : 'A') + rho[mem[q]]);
  printf("\n");
```

This code is used in section 16.

18. The twenty questions. I'll quote each of the questions here, verbatim, because they're a bit of a moving target. (Don Woods learned in 2001 that his original questions, proposed in 2000, could be "cooked" by dozens of unintended answer lists. So he changed them at that time—only to discover, alas, that the new set could also be "cooked"! Evidently problems such as this are by no means easy to solve correctly. Further tweaking by the author in 2015, with the help of this `CWEB` program and in consultation with Don himself, has led to the present version of the questionnaire, which retains the original flavor and—it says here—is uncookable.)

Logical reasoning can be tricky. Hopefully the code below is transparent enough to reveal bugs? In each case I'll try to state what ordering assumptions I'm making, when they are relevant.

Each case in the big switch exits either to *okay* or *bad* or *defer*.

19. "1. The first question whose answer is A is: (A) 1 (B) 2 (C) 3 (D) 4 (E) 5"

We can use the fact that questions 3 and 2 have preceded question 1 in the ordering, hence $mem[2]$ and $mem[3]$ already contain definite values.

```
⟨ Cases for the big switch 19 ⟩ ≡
case pack(0,1,A): goto okay;
case pack(0,1,B): deny(1,A); force(2,A); goto okay;
case pack(0,1,C): deny(1,A); deny(2,A); force(3,A); goto okay;
case pack(0,1,D): deny(1,A); deny(2,A); deny(3,A); force(4,A); goto okay;
case pack(0,1,E): deny(1,A); deny(2,A); deny(3,A); deny(4,A); force(5,A);
  goto okay;
case pack(1,1,A): goto bad;
case pack(1,1,B): deny(2,A); goto okay;      /* 2 < 1 in the ordering */
case pack(1,1,C): if (o,mem[2] ≠ AA) deny(3,A); goto okay;      /* 3 < 1 */
case pack(1,1,D): if ((o,mem[2] ≠ AA) ∧ (o,mem[3] ≠ AA)) deny(4,A); goto okay;
case pack(1,1,E): if ((o,mem[2] ≡ AA) ∨ (o,mem[3] ≡ AA)) goto okay;
  goto defer;
```

See also sections 21, 23, 25, 27, 28, 29, 33, 35, 37, 38, 40, 42, 44, 46, 48, 50, 51, 53, and 55.

This code is used in section 12.

20. Deferred cases exit to *b5* if the postponed test fails.

```
⟨ Cases for the deferred switch 20 ⟩ ≡
case pack(1,1,E): if ((o,mem[4] ≠ AA) ∧ (o,mem[5] ≡ AA)) goto b5; break;
```

See also sections 22, 24, 26, 32, 34, 36, 39, 41, 43, 45, 47, 49, and 54.

This code is used in section 16.

21. "2. The next question with the same answer as this one is: (A) 4 (B) 6 (C) 8 (D) 10 (E) 12"

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 2, *A*): *deny*(3, *A*); *force*(4, *A*); **goto** *okay*;
case *pack*(0, 2, *B*): *deny*(3, *B*); *deny*(4, *B*); *deny*(5, *B*); *force*(6, *B*); **goto** *okay*;
case *pack*(0, 2, *C*): *deny*(3, *C*); *deny*(4, *C*); *deny*(5, *C*); *deny*(6, *C*); *deny*(7, *C*);
force(8, *C*); **goto** *okay*;
case *pack*(0, 2, *D*): *deny*(3, *D*); *deny*(4, *D*); *deny*(5, *D*); *deny*(6, *D*); *deny*(7, *D*);
deny(8, *D*); *deny*(9, *D*); *force*(10, *D*);
goto *okay*;
case *pack*(0, 2, *E*): *deny*(3, *E*); *deny*(4, *E*); *deny*(5, *E*); *deny*(6, *E*); *deny*(7, *E*);
deny(8, *E*); *deny*(9, *E*); *deny*(10, *E*); *deny*(11, *E*); *force*(12, *E*);
goto *okay*;
case *pack*(1, 2, *A*): **if** (*o*, *mem*[3] ≠ AA) *deny*(4, *A*); **goto** *okay*; /* 3 < 2 */
case *pack*(1, 2, *B*): **if** (*o*, *mem*[3] ≡ BB) **goto** *okay*; **goto** *defer*;
case *pack*(1, 2, *C*): **if** (*o*, *mem*[3] ≡ CC) **goto** *okay*; **goto** *defer*;
case *pack*(1, 2, *D*): **if** (*o*, *mem*[3] ≡ DD) **goto** *okay*; **goto** *defer*;
case *pack*(1, 2, *E*): **if** (*o*, *mem*[3] ≡ EE) **goto** *okay*; **goto** *defer*;

22. ⟨ Cases for the deferred switch 20 ⟩ +≡
case *pack*(1, 2, *B*): **if** ((*o*, *mem*[4] ≠ BB) ∧ (*o*, *mem*[5] ≠ BB) ∧ (*o*, *mem*[6] ≡ BB)) **goto** *b5*;
break;
case *pack*(1, 2, *C*): **if** ((*o*, *mem*[4] ≠ CC) ∧ (*o*, *mem*[5] ≠ CC) ∧ (*o*, *mem*[6] ≠ CC) ∧
(*o*, *mem*[7] ≠ CC) ∧ (*o*, *mem*[8] ≡ CC)) **goto** *b5*;
break;
case *pack*(1, 2, *D*): **if** ((*o*, *mem*[4] ≠ DD) ∧ (*o*, *mem*[5] ≠ DD) ∧ (*o*, *mem*[6] ≠ DD) ∧
(*o*, *mem*[7] ≠ DD) ∧ (*o*, *mem*[8] ≠ DD) ∧ (*o*, *mem*[9] ≠ DD) ∧ (*o*, *mem*[10] ≡ DD)) **goto** *b5*;
break;
case *pack*(1, 2, *E*): **if** ((*o*, *mem*[4] ≠ EE) ∧ (*o*, *mem*[5] ≠ EE) ∧ (*o*, *mem*[6] ≠ EE) ∧
(*o*, *mem*[7] ≠ EE) ∧ (*o*, *mem*[8] ≠ EE) ∧ (*o*, *mem*[9] ≠ EE) ∧ (*o*, *mem*[10] ≠ EE) ∧
(*o*, *mem*[11] ≠ EE) ∧ (*o*, *mem*[12] ≡ EE)) **goto** *b5*;
break;

23. "3. The only two consecutive questions with identical answers are questions: (A) 15 and 16 (B) 16 and 17 (C) 17 and 18 (D) 18 and 19 (E) 19 and 20"

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 3, *A*): *loguy* = 15, *higuy* = 16; **goto** *okay*;
case *pack*(0, 3, *B*): *loguy* = 16, *higuy* = 17; **goto** *okay*;
case *pack*(0, 3, *C*): *loguy* = 17, *higuy* = 18; **goto** *okay*;
case *pack*(0, 3, *D*): *loguy* = 18, *higuy* = 19; **goto** *okay*;
case *pack*(0, 3, *E*): *loguy* = 19, *higuy* = 20; **goto** *okay*;
case *pack*(1, 3, *A*): **case** *pack*(1, 3, *B*): **case** *pack*(1, 3, *C*): **case** *pack*(1, 3, *D*): **case** *pack*(1, 3, *E*):
goto *defer*;

24. ⟨ Cases for the deferred switch 20 ⟩ +≡
case *pack*(1, 3, *A*): **if** ((*oo*, *mem*[15] ≡ *mem*[16]) ∧ (*o*, *mem*[16] ≠ *mem*[17]) ∧
(*o*, *mem*[17] ≠ *mem*[18]) ∧ (*o*, *mem*[18] ≠ *mem*[19]) ∧ (*o*, *mem*[19] ≠ *mem*[20])) **goto** *test3*;
break;
case *pack*(1, 3, *B*): **if** ((*oo*, *mem*[15] ≠ *mem*[16]) ∧ (*o*, *mem*[16] ≡ *mem*[17]) ∧
(*o*, *mem*[17] ≠ *mem*[18]) ∧ (*o*, *mem*[18] ≠ *mem*[19]) ∧ (*o*, *mem*[19] ≠ *mem*[20])) **goto** *test3*;
break;
case *pack*(1, 3, *C*): **if** ((*oo*, *mem*[15] ≠ *mem*[16]) ∧ (*o*, *mem*[16] ≠ *mem*[17]) ∧
(*o*, *mem*[17] ≡ *mem*[18]) ∧ (*o*, *mem*[18] ≠ *mem*[19]) ∧ (*o*, *mem*[19] ≠ *mem*[20])) **goto** *test3*;
break;
case *pack*(1, 3, *D*): **if** ((*oo*, *mem*[15] ≠ *mem*[16]) ∧ (*o*, *mem*[16] ≠ *mem*[17]) ∧
(*o*, *mem*[17] ≠ *mem*[18]) ∧ (*o*, *mem*[18] ≡ *mem*[19]) ∧ (*o*, *mem*[19] ≠ *mem*[20])) **goto** *test3*;
break;
case *pack*(1, 3, *E*): **if** ((*oo*, *mem*[15] ≠ *mem*[16]) ∧ (*o*, *mem*[16] ≠ *mem*[17]) ∧
(*o*, *mem*[17] ≠ *mem*[18]) ∧ (*o*, *mem*[18] ≠ *mem*[19]) ∧ (*o*, *mem*[19] ≡ *mem*[20])) **goto** *test3*;
break;
test3: **if** ((*oo*, *mem*[1] ≠ *mem*[2]) ∧ (*o*, *mem*[2] ≠ *mem*[3]) ∧ (*o*, *mem*[3] ≠ *mem*[4]) ∧
(*o*, *mem*[4] ≠ *mem*[5]) ∧ (*o*, *mem*[5] ≠ *mem*[6]) ∧ (*o*, *mem*[6] ≠ *mem*[7]) ∧
(*o*, *mem*[7] ≠ *mem*[8]) ∧ (*o*, *mem*[8] ≠ *mem*[9]) ∧ (*o*, *mem*[9] ≠ *mem*[10]) ∧
(*o*, *mem*[10] ≠ *mem*[11]) ∧ (*o*, *mem*[11] ≠ *mem*[12]) ∧ (*o*, *mem*[12] ≠ *mem*[13]) ∧
(*o*, *mem*[13] ≠ *mem*[14]) ∧ (*o*, *mem*[14] ≠ *mem*[15])) **goto** *b5*; **break**;

25. "4. The answer to this question is the same as the answers to questions: (A) 10 and 13 (B) 14 and 16 (C) 7 and 20 (D) 1 and 15 (E) 8 and 12"
Questions 1, 10, 15, and 20 precede this one in the ordering.

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 4, *A*): *force*(10, *A*); *force*(13, *A*); **goto** *okay*;
case *pack*(0, 4, *B*): *force*(14, *B*); *force*(16, *B*); **goto** *okay*;
case *pack*(0, 4, *C*): *force*(7, *C*); *force*(20, *C*); **goto** *okay*;
case *pack*(0, 4, *D*): *force*(1, *D*); *force*(15, *D*); **goto** *okay*;
case *pack*(0, 4, *E*): *force*(8, *E*); *force*(12, *E*); **goto** *okay*;
case *pack*(1, 4, *A*): **if** (*o*, *mem*[10] ≠ `AA`) **goto** *okay*; **goto** *defer*;
case *pack*(1, 4, *B*): **case** *pack*(1, 4, *E*): **goto** *defer*;
case *pack*(1, 4, *C*): **if** (*o*, *mem*[20] ≠ `CC`) **goto** *okay*; **goto** *defer*;
case *pack*(1, 4, *D*): **if** ((*o*, *mem*[1] ≠ `DD`) ∨ (*o*, *mem*[15] ≠ `DD`)) **goto** *okay*; **goto** *bad*;

26. ⟨ Cases for the deferred switch 20 ⟩ +≡
case *pack*(1, 4, *A*): **if** ((*o*, *mem*[13] ≡ `AA`)) **goto** *b5*; **break**;
case *pack*(1, 4, *B*): **if** ((*o*, *mem*[14] ≡ `BB`) ∧ (*o*, *mem*[16] ≡ `BB`)) **goto** *b5*; **break**;
case *pack*(1, 4, *C*): **if** (*o*, *mem*[7] ≡ `CC`) **goto** *b5*; **break**;
case *pack*(1, 4, *E*): **if** ((*o*, *mem*[8] ≡ `EE`) ∧ (*o*, *mem*[12] ≡ `EE`)) **goto** *b5*; **break**;

27. "5. The answer to question 14 is: (A) B (B) E (C) C (D) A (E) D"

⟨ Cases for the big switch 19 ⟩ +≡
```
case pack(0,5,A): force(14,B); goto okay;
case pack(0,5,B): force(14,E); goto okay;
case pack(0,5,C): force(14,C); goto okay;
case pack(0,5,D): force(14,A); goto okay;
case pack(0,5,E): force(14,D); goto okay;
case pack(1,5,A): deny(14,B); goto okay;
case pack(1,5,B): deny(14,E); goto okay;
case pack(1,5,C): deny(14,C); goto okay;
case pack(1,5,D): deny(14,A); goto okay;
case pack(1,5,E): deny(14,D); goto okay;
```

28. "6. The answer to this question is: (A) A (B) B (C) C (D) D (E) none of those"

⟨ Cases for the big switch 19 ⟩ +≡
```
case pack(0,6,A): case pack(0,6,B): case pack(0,6,C): case pack(0,6,D): case pack(0,6,E):
  goto okay;
case pack(1,6,A): case pack(1,6,B): case pack(1,6,C): case pack(1,6,D): case pack(1,6,E):
  goto bad;
```

29. "7. An answer that appears most often is: (A) A (B) B (C) C (D) D (E) E"

⟨ Cases for the big switch 19 ⟩ +≡
```
case pack(0,7,A): case pack(0,7,B): case pack(0,7,C): case pack(0,7,D): case pack(0,7,E):
  case pack(1,7,A): case pack(1,7,B): case pack(1,7,C): case pack(1,7,D): case pack(1,7,E):
  goto defer;
```

30. ⟨ Compute the distribution of answers 30 ⟩ ≡
```
{
  register int dA = 0, dB = 0, dC = 0, dD = 0, dE = 0;
  for (q = 1; q ≤ 20; q++) {
    o, y = mem[q];
    if (y ≡ AA) dA++;
    else if (y ≡ BB) dB++;
    else if (y ≡ CC) dC++;
    else if (y ≡ DD) dD++;
    else dE++;
  }
  o, dist[A] = dA;
  o, dist[B] = dB;
  o, dist[C] = dC;
  o, dist[D] = dD;
  o, dist[E] = dE;
  trick = (1 ≪ dA) + (1 ≪ dB) + (1 ≪ dC) + (1 ≪ dD) + (1 ≪ dE);      /* see 14 below */
}
```
This code is used in section 16.

31. ⟨ Other global variables 6 ⟩ +≡
```
int dist[5];      /* the number of A's, B's, C's, D's, E's */
int tie[5];       /* cases that are tied with at least one other */
int trick;
```

32. ⟨ Cases for the deferred switch 20 ⟩ +≡
```
case pack(0,7,A): case pack(0,7,B): case pack(0,7,C): case pack(0,7,D): case pack(0,7,E):
    case pack(1,7,A): case pack(1,7,B): case pack(1,7,C): case pack(1,7,D): case pack(1,7,E):
    for (o, i = 0, j = dist[x]; i < 5; i++)
    if (o, dist[i] > j) break;
  if (¬u ∧ i < 5) goto b5;
  if (u ∧ i ≡ 5) goto b5; break;
```

33. "8. Ignoring answers that appear equally often, the least common answer is: (A) A (B) B (C) C (D) D (E) E"

⟨ Cases for the big switch 19 ⟩ +≡
```
case pack(0,8,A): case pack(0,8,B): case pack(0,8,C): case pack(0,8,D): case pack(0,8,E):
  case pack(1,8,A): case pack(1,8,B): case pack(1,8,C): case pack(1,8,D): case pack(1,8,E):
  goto defer;
```

34. ⟨ Cases for the deferred switch 20 ⟩ +≡
```
case pack(0,8,A): case pack(0,8,B): case pack(0,8,C): case pack(0,8,D): case pack(0,8,E):
    case pack(1,8,A): case pack(1,8,B): case pack(1,8,C): case pack(1,8,D): case pack(1,8,E):
    for (i = 0; i < 5; i++) o, tie[i] = 0;
  for (i = 0; i < 5; i++)
    for (j = i + 1; j < 5; j++)
      if (oo, dist[i] ≡ dist[j]) oo, tie[i] = tie[j] = 1;
  for (i = 0, j = 100; i < 5; i++)
    if ((o, tie[i] ≡ 0) ∧ (o, dist[i] < j)) j = dist[i];      /* possibly j ≡ 100 */
  if (¬u ∧ j ≠ dist[x]) goto b5;
  if (u ∧ j ≡ dist[x]) goto b5; break;
```

35. "9. The sum of all question numbers whose answers are correct and the same as this one is: (A) ∈ [59 . . 62] (B) ∈ [52 . . 55] (C) ∈ [44 . . 49] (D) ∈ [59 . . 67] (E) ∈ [44 . . 53]"

⟨ Cases for the big switch 19 ⟩ +≡
```
case pack(0,9,A): case pack(0,9,B): case pack(0,9,C): case pack(0,9,D): case pack(0,9,E):
  case pack(1,9,A): case pack(1,9,B): case pack(1,9,C): case pack(1,9,D): case pack(1,9,E):
  goto defer;
```

36. ⟨ Cases for the deferred switch 20 ⟩ +≡
```
case pack(0,9,A): case pack(0,9,B): case pack(0,9,C): case pack(0,9,D): case pack(0,9,E):
  case pack(1,9,A): case pack(1,9,B): case pack(1,9,C): case pack(1,9,D): case pack(1,9,E):
  for (j = 0, i = 1; i ≤ 20; i++)
    if ((o, falsity[i] ≡ 0) ∧ (o, mem[i] ≡ (1 ≪ x))) j += i;
  switch (x) {
  case A: i = (j ≥ 59 ∧ j ≤ 62); break;
  case B: i = (j ≥ 52 ∧ j ≤ 55); break;
  case C: i = (j ≥ 44 ∧ j ≤ 49); break;
  case D: i = (j ≥ 59 ∧ j ≤ 67); break;
  case E: i = (j ≥ 44 ∧ j ≤ 53); break;
  }
  if (¬u ∧ ¬i) goto b5;
  if (u ∧ i) goto b5; break;
```

37. "10. The answer to question 17 is: (A) D (B) B (C) A (D) E (E) wrong"

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 10, *A*): *force*(17, *D*); **goto** *okay*;
case *pack*(0, 10, *B*): *force*(17, *B*); **goto** *okay*;
case *pack*(0, 10, *C*): *force*(17, *A*); **goto** *okay*;
case *pack*(0, 10, *D*): *force*(17, *E*); **goto** *okay*;
case *pack*(0, 10, *E*): **if** (*o*, *falsity*[17] ≡ 1) **goto** *okay*; **goto** *bad*;
case *pack*(1, 10, *A*): *deny*(17, *D*); **goto** *okay*;
case *pack*(1, 10, *B*): *deny*(17, *B*); **goto** *okay*;
case *pack*(1, 10, *C*): *deny*(17, *A*); **goto** *okay*;
case *pack*(1, 10, *D*): *deny*(17, *E*); **goto** *okay*;
case *pack*(1, 10, *E*): **if** (*o*, *falsity*[17] ≡ 1) **goto** *bad*; **goto** *okay*;

38. "11. The number of questions whose answer is D is: (A) 2 (B) 3 (C) 4 (D) 5 (E) 6"

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 11, *A*): **case** *pack*(0, 11, *B*): **case** *pack*(0, 11, *C*): **case** *pack*(0, 11, *D*): **case** *pack*(0, 11, *E*):
case *pack*(1, 11, *A*): **case** *pack*(1, 11, *B*): **case** *pack*(1, 11, *C*): **case** *pack*(1, 11, *D*): **case** *pack*(1, 11, *E*):
goto *defer*;

39. ⟨ Cases for the deferred switch 20 ⟩ +≡
case *pack*(0, 11, *A*): **case** *pack*(0, 11, *B*): **case** *pack*(0, 11, *C*): **case** *pack*(0, 11, *D*): **case** *pack*(0, 11, *E*):
case *pack*(1, 11, *A*): **case** *pack*(1, 11, *B*): **case** *pack*(1, 11, *C*): **case** *pack*(1, 11, *D*): **case** *pack*(1, 11, *E*):
if (¬*u* ∧ (*o*, *dist*[*D*] ≠ *x* + 2)) **goto** *b5*;
if (*u* ∧ (*o*, *dist*[*D*] ≡ *x* + 2)) **goto** *b5*; **break**;

40. "12. The number of *other* questions with the same answer as this one is the same as the number of questions with answer: (A) B (B) C (C) D (D) E (E) none of those"

Here we see that (E) is quite different from 'A'. "None of those" means that options A, B, C, and D are all false.

And there's more subtlety too, because the *dist* table would be different if we had really chosen A instead of E. Suppose 12E has been selected. Then 12A does *not* mean "the number of other questions with answer A is the same as the number questions with answer B"; it means "the number of other questions with answer E is the same as the number questions with answer B." Thus 12E is true if and only if *dist*[*B*], *dist*[*C*], *dist*[*D*], and *dist*[*E*] all differ from *dist*[*E*] − 1. But 12A is true if and only if *dist*[*A*] − 1 = *dist*[*B*]. Got that?

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 12, *A*): **case** *pack*(0, 12, *B*): **case** *pack*(0, 12, *C*): **case** *pack*(0, 12, *D*): **case** *pack*(0, 12, *E*):
case *pack*(1, 12, *A*): **case** *pack*(1, 12, *B*): **case** *pack*(1, 12, *C*): **case** *pack*(1, 12, *D*): **case** *pack*(1, 12, *E*):
goto *defer*;

41. ⟨ Cases for the deferred switch 20 ⟩ +≡
case *pack*(0, 12, *A*): **case** *pack*(0, 12, *B*): **case** *pack*(0, 12, *C*): **case** *pack*(0, 12, *D*):
if (*oo*, *dist*[*x*] − 1 ≠ *dist*[*x* + 1]) **goto** *b5*; **break**;
case *pack*(0, 12, *E*): **if** ((*oo*, *dist*[*E*] − 1 ≡ *dist*[*B*]) ∨ (*o*, *dist*[*E*] − 1 ≡ *dist*[*C*]) ∨
(*o*, *dist*[*E*] − 1 ≡ *dist*[*D*])) **goto** *b5*; **break**;
case *pack*(1, 12, *A*): **case** *pack*(1, 12, *B*): **case** *pack*(1, 12, *C*): **case** *pack*(1, 12, *D*):
if (*oo*, *dist*[*x*] − 1 ≡ *dist*[*x* + 1]) **goto** *b5*; **break**;
case *pack*(1, 12, *E*): **if** ((*oo*, *dist*[*E*] − 1 ≠ *dist*[*B*]) ∧ (*o*, *dist*[*E*] − 1 ≠ *dist*[*C*]) ∧
(*o*, *dist*[*E*] − 1 ≠ *dist*[*D*])) **goto** *b5*; **break**;

42. "13. The number of questions whose answer is E is: (A) 5 (B) 4 (C) 3 (D) 2 (E) 1"

⟨ Cases for the big switch 19 ⟩ +≡
case $pack(0,13,A)$: **case** $pack(0,13,B)$: **case** $pack(0,13,C)$: **case** $pack(0,13,D)$: **case** $pack(0,13,E)$:
case $pack(1,13,A)$: **case** $pack(1,13,B)$: **case** $pack(1,13,C)$: **case** $pack(1,13,D)$: **case** $pack(1,13,E)$:
goto *defer*;

43. ⟨ Cases for the deferred switch 20 ⟩ +≡
case $pack(0,13,A)$: **case** $pack(0,13,B)$: **case** $pack(0,13,C)$: **case** $pack(0,13,D)$: **case** $pack(0,13,E)$:
case $pack(1,13,A)$: **case** $pack(1,13,B)$: **case** $pack(1,13,C)$: **case** $pack(1,13,D)$: **case** $pack(1,13,E)$:
if $(\neg u \wedge (o, dist[E] \neq 5 - x))$ **goto** *b5*;
if $(u \wedge (o, dist[E] \equiv 5 - x))$ **goto** *b5*; **break**;

44. "14. No answer appears exactly this many times: (A) 2 (B) 3 (C) 4 (D) 5 (E) none of those"

Here I can't help using a dirty trick. Case (E) means that the digits 2, 3, 4, 5 all appear in the *dist* table. But the *dist* table sums to 20, so it must also contain 6! This condition happens if and only if the *trick* computed above is $(1 \ll 2) + (1 \ll 3) + (1 \ll 4) + (1 \ll 5) + (1 \ll 6)$, which is $^{\#}$`7c`.

⟨ Cases for the big switch 19 ⟩ +≡
case $pack(0,14,A)$: **case** $pack(0,14,B)$: **case** $pack(0,14,C)$: **case** $pack(0,14,D)$: **case** $pack(0,14,E)$:
case $pack(1,14,A)$: **case** $pack(1,14,B)$: **case** $pack(1,14,C)$: **case** $pack(1,14,D)$: **case** $pack(1,14,E)$:
goto *defer*;

45. ⟨ Cases for the deferred switch 20 ⟩ +≡
case $pack(0,14,A)$: **case** $pack(0,14,B)$: **case** $pack(0,14,C)$: **case** $pack(0,14,D)$:
for $(i = 0;\ i < 5;\ i{+}{+})$ **if** $(o, dist[i] \equiv x + 2)$ **goto** *b5*;
break;
case $pack(0,14,E)$: **if** $(trick \neq {}^{\#}\texttt{7c})$ **goto** *b5*; **break**;
case $pack(1,14,A)$: **case** $pack(1,14,B)$: **case** $pack(1,14,C)$: **case** $pack(1,14,D)$:
for $(i = 0;\ i < 5;\ i{+}{+})$ **if** $(o, dist[i] \equiv x + 2)$ **break**;
if $(i \equiv 5)$ **goto** *b5*;
break;
case $pack(1,14,E)$: **if** $(trick \equiv {}^{\#}\texttt{7c})$ **goto** *b5*; **break**;

46. "15. The set of odd-numbered questions with answer A is: (A) {7} (B) {9} (C) not {11} (D) {13} (E) {15}"

In the falsifying case, I note that question 3 has been treated earlier in the ordering.

⟨ Cases for the big switch 19 ⟩ +≡
case $pack(0,15,A)$: **case** $pack(0,15,E)$: **goto** *bad*;
case $pack(0,15,B)$: $force(9,A)$; $deny(11,A)$; $deny(13,A)$; **goto** *odd_denials*;
case $pack(0,15,D)$: $deny(9,A)$; $deny(11,A)$; $force(13,A)$; **goto** *odd_denials*;
odd_denials: $deny(1,A)$; $deny(3,A)$; $deny(5,A)$; $deny(7,A)$;
$deny(17,A)$; $deny(19,A)$; **goto** *okay*;
case $pack(1,15,A)$: **case** $pack(1,15,E)$: **goto** *okay*;
case $pack(0,15,C)$: **case** $pack(1,15,B)$: **case** $pack(1,15,D)$:
if $(o, mem[3] \equiv \texttt{AA})$ **goto** *okay*; **goto** *defer*;
case $pack(1,15,C)$: **goto** *defer*;

47. ⟨ Cases for the deferred switch 20 ⟩ +≡
case *pack*(1, 15, *B*): **if** ((*o*, *mem*[9] ≡ AA) ∧ (*o*, *mem*[11] ≠ AA) ∧ (*o*, *mem*[13] ≠ AA)) **goto** *test_odd*;
break;
case *pack*(0, 15, *C*): **case** *pack*(1, 15, *C*): **if** ((*o*, *mem*[1] ≠ AA) ∧ (*o*, *mem*[3] ≠ AA) ∧
(*o*, *mem*[5] ≠ AA) ∧ (*o*, *mem*[7] ≠ AA) ∧ (*o*, *mem*[9] ≠ AA) ∧ (*o*, *mem*[11] ≡ AA) ∧
(*o*, *mem*[13] ≠ AA) ∧ (*o*, *mem*[17] ≠ AA) ∧ (*o*, *mem*[19] ≠ AA)) **goto** *b5*; **break**;
case *pack*(1, 15, *D*): **if** ((*o*, *mem*[9] ≠ AA) ∧ (*o*, *mem*[11] ≠ AA) ∧ (*o*, *mem*[13] ≡ AA)) **goto** *test_odd*;
break;
test_odd: **if** ((*o*, *mem*[1] ≠ AA) ∧ (*o*, *mem*[5] ≠ AA) ∧ (*o*, *mem*[7] ≠ AA) ∧
(*o*, *mem*[17] ≠ AA) ∧ (*o*, *mem*[19] ≠ AA)) **goto** *b5*;
break;

48. "16. The answer to question 8 is the same as the answer to question: (A) 3 (B) 2 (C) 13 (D) 18 (E) 20"

This question is considered later in the ordering than questions 2, 3, and 20.

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 16, *A*): *oo*; *force*(8, *rho*[*mem*[3]]); **goto** *okay*;
case *pack*(0, 16, *B*): *oo*; *force*(8, *rho*[*mem*[2]]); **goto** *okay*;
case *pack*(0, 16, *E*): *oo*; *force*(8, *rho*[*mem*[20]]); **goto** *okay*;
case *pack*(0, 16, *C*): **case** *pack*(0, 16, *D*): **goto** *defer*;
case *pack*(1, 16, *A*): *oo*; *deny*(8, *rho*[*mem*[3]]); **goto** *okay*;
case *pack*(1, 16, *B*): *oo*; *deny*(8, *rho*[*mem*[2]]); **goto** *okay*;
case *pack*(1, 16, *E*): *oo*; *deny*(8, *rho*[*mem*[20]]); **goto** *okay*;
case *pack*(1, 16, *C*): **case** *pack*(1, 16, *D*): **goto** *defer*;

49. ⟨ Cases for the deferred switch 20 ⟩ +≡
case *pack*(0, 16, *C*): **if** (*o*, *mem*[8] ≠ *mem*[13]) **goto** *b5*; **break**;
case *pack*(0, 16, *D*): **if** (*o*, *mem*[8] ≠ *mem*[18]) **goto** *b5*; **break**;
case *pack*(1, 16, *C*): **if** (*o*, *mem*[8] ≡ *mem*[13]) **goto** *b5*; **break**;
case *pack*(1, 16, *D*): **if** (*o*, *mem*[8] ≡ *mem*[18]) **goto** *b5*; **break**;

50. "17. The answer to question 10 is: (A) C (B) D (C) B (D) A (E) correct"

An answer list is invalid if it contains both 10E and 17E, whether or not those answers are supposed to be correct or not. (This is subtle but clarified in my book.) Question 17 precedes 10 in the ordering, so it makes sure question 10 won't mess up.

⟨ Cases for the big switch 19 ⟩ +≡
case *pack*(0, 17, *A*): *force*(10, *C*); **goto** *okay*;
case *pack*(0, 17, *B*): *force*(10, *D*); **goto** *okay*;
case *pack*(0, 17, *C*): *force*(10, *B*); **goto** *okay*;
case *pack*(0, 17, *D*): *force*(10, *A*); **goto** *okay*;
case *pack*(0, 17, *E*): *deny*(10, *E*); **if** (*o*, *falsity*[10] ≡ 0) **goto** *okay*; **goto** *bad*;
case *pack*(1, 17, *A*): *deny*(10, *C*); **goto** *okay*;
case *pack*(1, 17, *B*): *deny*(10, *D*); **goto** *okay*;
case *pack*(1, 17, *C*): *deny*(10, *B*); **goto** *okay*;
case *pack*(1, 17, *D*): *deny*(10, *A*); **goto** *okay*;
case *pack*(1, 17, *E*): *deny*(10, *E*); **if** (*o*, *falsity*[10] ≡ 0) **goto** *bad*; **goto** *okay*;

51. "18. The number of prime-numbered questions whose answers are vowels is: (A) prime (B) square (C) odd (D) even (E) zero"

All of the prime-numbered questions appear before this one in the ordering.

#define *isvowel*(q) $(o,(mem[q]\ \&\ {}^{\#}\texttt{11})\ ?\ 1:0)$

⟨ Cases for the big switch 19 ⟩ +≡

case *pack*$(0,18,A)$: **case** *pack*$(0,18,B)$: **case** *pack*$(0,18,C)$: **case** *pack*$(0,18,D)$: **case** *pack*$(0,18,E)$:
case *pack*$(1,18,A)$: **case** *pack*$(1,18,B)$: **case** *pack*$(1,18,C)$: **case** *pack*$(1,18,D)$: **case** *pack*$(1,18,E)$:
$j = isvowel(2) + isvowel(3) + isvowel(5) + isvowel(7) +$
$\quad isvowel(11) + isvowel(13) + isvowel(17) + isvowel(19)$;
$j = (1 \ll j)\ \&\ magic[x]$;
if $(\neg u \wedge j)$ **goto** *okay*; **goto** *bad*;
if $(u \wedge \neg j)$ **goto** *okay*; **goto** *bad*;

52. ⟨ Other global variables 6 ⟩ +≡

int $magic[\,] = \{(1 \ll 2) + (1 \ll 3) + (1 \ll 5) + (1 \ll 7),$
$\quad (1 \ll 0) + (1 \ll 1) + (1 \ll 4),$
$\quad (1 \ll 1) + (1 \ll 3) + (1 \ll 5) + (1 \ll 7),$
$\quad (1 \ll 0) + (1 \ll 2) + (1 \ll 4) + (1 \ll 6) + (1 \ll 8),$
$\quad (1 \ll 0)\}$

53. "19. The last question whose answer is B is: (A) 14 (B) 15 (C) 16 (D) 17 (E) 18"

Question 20 has been considered before this one in the ordering.

⟨ Cases for the big switch 19 ⟩ +≡

case *pack*$(0,19,A)$: *force*$(14,B)$; *deny*$(15,B)$; *deny*$(16,B)$; *deny*$(17,B)$;
deny$(18,B)$; *deny*$(20,B)$; **goto** *okay*;
case *pack*$(0,19,B)$: **goto** *bad*;
case *pack*$(0,19,C)$: *force*$(16,B)$; *deny*$(17,B)$; *deny*$(18,B)$; *deny*$(20,B)$; **goto** *okay*;
case *pack*$(0,19,D)$: *force*$(17,B)$; *deny*$(18,B)$; *deny*$(20,B)$; **goto** *okay*;
case *pack*$(0,19,E)$: *force*$(18,B)$; *deny*$(20,B)$; **goto** *okay*;
case *pack*$(1,19,B)$: **goto** *okay*;
case *pack*$(1,19,A)$: **case** *pack*$(1,19,C)$: **case** *pack*$(1,19,D)$: **case** *pack*$(1,19,E)$:
if $(o, mem[20] \equiv \texttt{BB})$ **goto** *okay*; **goto** *defer*;

54. ⟨ Cases for the deferred switch 20 ⟩ +≡

case *pack*$(1,19,A)$: **if** $((o, mem[14] \equiv \texttt{BB}) \wedge (o, mem[15] \neq \texttt{BB}) \wedge (o, mem[16] \neq \texttt{BB}) \wedge$
$\quad (o, mem[17] \neq \texttt{BB}) \wedge (o, mem[18] \neq \texttt{BB}))$ **goto** *b5*; **break**;
case *pack*$(1,19,C)$: **if** $((o, mem[16] \equiv \texttt{BB}) \wedge (o, mem[17] \neq \texttt{BB}) \wedge (o, mem[18] \neq \texttt{BB}))$ **goto** *b5*; **break**;
case *pack*$(1,19,D)$: **if** $((o, mem[17] \equiv \texttt{BB})\ \&\ (o, mem[18] \neq \texttt{BB}))$ **goto** *b5*; **break**;
case *pack*$(1,19,E)$: **if** $(o, mem[18] \equiv \texttt{BB})$ **goto** *b5*; **break**;

55. "20. The maximum score that can be achieved on this test is: (A) 18 (B) 19 (C) 20 (D) indeterminate (E) achievable only by getting this question wrong"

This climactic final question, which Don Woods admits was expressly "designed to create difficulties," obviously needs some special consideration. Discussion in my book shows that option (D) is always false. I assume here that (E) is also false, although that hypothesis must be confirmed by examining outputs of this program. As to (A), (B), (C), I essentially reword the problem to say not "the maximum score" but "the score achieved by all outputs of this run."

```
⟨ Cases for the big switch 19 ⟩ +≡
case pack(0, 20, A): case pack(0, 20, B): case pack(0, 20, C):
  if (score ≡ 18 + x) goto okay; goto bad;
case pack(0, 20, D): case pack(0, 20, E): goto bad;
case pack(1, 20, A): case pack(1, 20, B): case pack(1, 20, C):
  if (score ≠ 18 + x) goto okay; goto bad;
case pack(1, 20, D): case pack(1, 20, E): goto okay;
```

56. Index.

BACK-20Q