SAMPLE EXAM ♯1

The purpose of the exam is to analyse this program:

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int highest (Tab lazied, Tab piecing) {
10   int cooky;
11   int bifocal;
12   int sibyls;
13   int vast = 0;
14   for (sibyls=0; sibyls<piecing.len; sibyls++) {
15     bifocal = 0;
16     for (cooky=0; cooky<lazied.len; cooky++) {
17       if (piecing.val[sibyls] == lazied.val[cooky]) {
18         bifocal = 1;
19         break;
20       }
21     }
22     if (!bifocal) {
23       vast += piecing.val[sibyls];
24     }
25   }
26   return vast;
27 }
28
29 int main () {
30   Tab cupful = {.len=8, .val=malloc(8 * sizeof(int))};
31   Tab getup = {.len=10, .val=malloc(10 * sizeof(int))};
32   getup.val[0] = 7;
33   getup.val[1] = 3;
34   getup.val[2] = 1;
35   getup.val[3] = 7;
36   getup.val[4] = 8;
37   getup.val[5] = 6;
38   getup.val[6] = 10;
39   getup.val[7] = 2;
40   getup.val[8] = 4;
41   getup.val[9] = 3;
42   cupful.val[0] = 3;
43   cupful.val[1] = 3;
44   cupful.val[2] = 4;
45   cupful.val[3] = 4;
46   cupful.val[4] = 8;
47   cupful.val[5] = 9;
48   cupful.val[6] = 2;
49   cupful.val[7] = 3;
50   printf("%i\n", highest(getup, cupful));
51   free(cupful.val);
52   free(getup.val);
53 }
54
```

Question 1 ♣   What does function highest computes?

- [ ] the number of values that are both in `lazied` and in `piecing`
- [ ] the number of values that are both in `piecing` and in `lazied`
- [ ] the largest value that is in `piecing` but not in `lazied`
- [x] the sum of the values that are in `lazied` but not in `piecing`
- [ ] the number of values that are in `lazied` but not in `piecing`
- [ ] the largest value that is both in `piecing` and in `lazied`
- [ ] the number of values that are in `piecing` but not in `lazied`
- [ ] the smallest value that is in `lazied` but not in `piecing`
- [ ] the largest value that is both in `lazied` and in `piecing`
- [ ] the smallest value that is in `piecing` but not in `lazied`
- [ ] the sum of the values that are both in `lazied` and in `piecing`
- [ ] the sum of the values that are in `piecing` but not in `lazied`
- [ ] the smallest value that is both in `lazied` and in `piecing`
- [ ] the sum of the values that are both in `piecing` and in `lazied`
- [ ] the smallest value that is both in `piecing` and in `lazied`
- [ ] the largest value that is in `lazied` but not in `piecing`

Question 2 ♣   What does the program prints?

- [ ] 11
- [ ] 2
- [ ] 14
- [ ] 7
- [ ] 6
- [ ] 27
- [ ] 1
- [ ] 8
- [x] 9
- [ ] 15
- [ ] 20
- [ ] 21
- [ ] 31
- [ ] 5
- [ ] 10

Question 3 ♣   What does the program prints?

■ 0	■ 0	☐ 0
☐ 1	☐ 1	☐ 1
☐ 2	☐ 2	☐ 2
☐ 3	☐ 3	☐ 3
☐ 4	☐ 4	☐ 4
☐ 5	☐ 5	☐ 5
☐ 6	☐ 6	☐ 6
☐ 7	☐ 7	☐ 7
☐ 8	☐ 8	☐ 8
☐ 9	☐ 9	■ 9

SAMPLE EXAM ♯2

The purpose of the exam is to analyse this program:

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int junkie (Tab dislike, Tab news) {
10   int severer;
11   int undo = 0;
12   int dizzy;
13   int abode;
14   for (abode=0; abode<news.len; abode++) {
15     dizzy = 0;
16     for (severer=0; severer<dislike.len; severer++) {
17       if (news.val[abode] == dislike.val[severer]) {
18         dizzy = 1;
19         break;
20       }
21     }
22     if ((!dizzy) && (news.val[abode] > undo)) {
23       undo = news.val[abode];
24     }
25   }
26   return undo;
27 }
28
29 int main () {
30   Tab nickels = {.len=8, .val=malloc(8 * sizeof(int))};
31   Tab skimpy = {.len=5, .val=malloc(5 * sizeof(int))};
32   skimpy.val[0] = 7;
33   skimpy.val[1] = 7;
34   skimpy.val[2] = 7;
35   skimpy.val[3] = 2;
36   skimpy.val[4] = 1;
37   nickels.val[0] = 9;
38   nickels.val[1] = 1;
39   nickels.val[2] = 3;
40   nickels.val[3] = 8;
41   nickels.val[4] = 5;
42   nickels.val[5] = 1;
43   nickels.val[6] = 7;
44   nickels.val[7] = 8;
45   printf("%i\n", junkie(skimpy, nickels));
46   free(skimpy.val);
47   free(nickels.val);
48 }
```

Question 1 ♣   What does function junkie computes?

- ☐ the number of values that are both in `news` and in `dislike`
- ☐ the sum of the values that are in `dislike` but not in `news`
- ☐ the largest value that is both in `dislike` and in `news`
- ☐ the smallest value that is in `news` but not in `dislike`
- ☐ the number of values that are both in `dislike` and in `news`
- ☐ the smallest value that is both in `news` and in `dislike`
- ☐ the sum of the values that are in `news` but not in `dislike`
- ☐ the largest value that is both in `news` and in `dislike`
- ☐ the number of values that are in `news` but not in `dislike`
- ☐ the sum of the values that are both in `news` and in `dislike`
- ☐ the smallest value that is in `dislike` but not in `news`
- ☐ the sum of the values that are both in `dislike` and in `news`
- ☐ the smallest value that is both in `dislike` and in `news`
- ☐ the number of values that are in `dislike` but not in `news`
- ■ the largest value that is in `dislike` but not in `news`
- ☐ the largest value that is in `news` but not in `dislike`

Question 2 ♣   What does the program prints?

- ☐ 33
- ☐ 4
- ☐ 5
- ☐ 2
- ☐ 14
- ☐ 22
- ☐ 1
- ☐ 16
- ■ 9
- ☐ 7
- ☐ 11
- ☐ 3

Question 3 ♣ What does the program prints?

■ 0	■ 0	☐ 0
☐ 1	☐ 1	☐ 1
☐ 2	☐ 2	☐ 2
☐ 3	☐ 3	☐ 3
☐ 4	☐ 4	☐ 4
☐ 5	☐ 5	☐ 5
☐ 6	☐ 6	☐ 6
☐ 7	☐ 7	☐ 7
☐ 8	☐ 8	☐ 8
☐ 9	☐ 9	■ 9

SAMPLE EXAM ♯3

The purpose of the exam is to analyse this program:
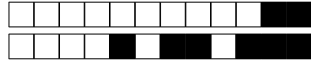
```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int belie (Tab endue, Tab moodily) {
10   int jerkin = 0;
11   int nouns;
12   int accord;
13   int inn;
14   for (accord=0; accord<endue.len; accord++) {
15     nouns = 0;
16     for (inn=0; inn<moodily.len; inn++) {
17       if (endue.val[accord] == moodily.val[inn]) {
18         nouns = 1;
19         break;
20       }
21     }
22     if (!nouns) {
23       jerkin += endue.val[accord];
24     }
25   }
26   return jerkin;
27 }
28
29 int main () {
30   Tab blamer = {.len=9, .val=malloc(9 * sizeof(int))};
31   Tab clamps = {.len=8, .val=malloc(8 * sizeof(int))};
32   blamer.val[0] = 1;
33   blamer.val[1] = 3;
34   blamer.val[2] = 7;
35   blamer.val[3] = 9;
36   blamer.val[4] = 9;
37   blamer.val[5] = 4;
38   blamer.val[6] = 6;
39   blamer.val[7] = 3;
40   blamer.val[8] = 5;
41   clamps.val[0] = 1;
42   clamps.val[1] = 2;
43   clamps.val[2] = 5;
44   clamps.val[3] = 10;
45   clamps.val[4] = 2;
46   clamps.val[5] = 9;
47   clamps.val[6] = 1;
48   clamps.val[7] = 2;
49   printf("%i\n", belie(blamer, clamps));
50   free(blamer.val);
51   free(clamps.val);
52 }
```

Question 1 ♣   What does function belie computes?

- [ ] the smallest value that is both in endue and in moodily
- [ ] the number of values that are in endue but not in moodily
- [ ] the largest value that is in endue but not in moodily
- [ ] the sum of the values that are in moodily but not in endue
- [ ] the sum of the values that are both in moodily and in endue
- [ ] the number of values that are both in endue and in moodily
- [ ] the largest value that is in moodily but not in endue
- [ ] the largest value that is both in moodily and in endue
- [ ] the smallest value that is in moodily but not in endue
- [ ] the largest value that is both in endue and in moodily
- [ ] the sum of the values that are both in endue and in moodily
- [ ] the number of values that are in moodily but not in endue
- [ ] the smallest value that is both in moodily and in endue
- [x] the sum of the values that are in endue but not in moodily
- [ ] the smallest value that is in endue but not in moodily
- [ ] the number of values that are both in moodily and in endue

Question 2 ♣   What does the program prints?

- [ ] 29
- [ ] 9
- [ ] 4
- [ ] 2
- [ ] 24
- [x] 23
- [ ] 10
- [ ] 16
- [ ] 5
- [ ] 21
- [ ] 26
- [ ] 1
- [ ] 11
- [ ] 7
- [ ] 3

Question 3 ♣ What does the program prints?

■0	□0	□0
□1	□1	□1
□2	■2	□2
□3	□3	■3
□4	□4	□4
□5	□5	□5
□6	□6	□6
□7	□7	□7
□8	□8	□8
□9	□9	□9

SAMPLE EXAM ♯4

The purpose of the exam is to analyse this program:
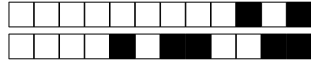
```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int sired (Tab phlox, Tab same) {
10   int clawed;
11   int hauls = 0;
12   int wattle;
13   int meaner;
14   for (meaner=0; meaner<same.len; meaner++) {
15     wattle = 1;
16     for (clawed=0; clawed<phlox.len; clawed++) {
17       if (same.val[meaner] == phlox.val[clawed]) {
18         wattle = 0;
19         break;
20       }
21     }
22     if (!wattle) {
23       hauls += same.val[meaner];
24     }
25   }
26   return hauls;
27 }
28
29 int main () {
30   Tab fiddly = {.len=10, .val=malloc(10 * sizeof(int))};
31   Tab scruffy = {.len=10, .val=malloc(10 * sizeof(int))};
32   scruffy.val[0] = 8;
33   scruffy.val[1] = 7;
34   scruffy.val[2] = 5;
35   scruffy.val[3] = 9;
36   scruffy.val[4] = 6;
37   scruffy.val[5] = 5;
38   scruffy.val[6] = 5;
39   scruffy.val[7] = 3;
40   scruffy.val[8] = 8;
41   scruffy.val[9] = 1;
42   fiddly.val[0] = 4;
43   fiddly.val[1] = 10;
44   fiddly.val[2] = 1;
45   fiddly.val[3] = 3;
46   fiddly.val[4] = 8;
47   fiddly.val[5] = 9;
48   fiddly.val[6] = 8;
49   fiddly.val[7] = 7;
50   fiddly.val[8] = 4;
51   fiddly.val[9] = 10;
52   printf("%i\n", sired(scruffy, fiddly));
53   free(scruffy.val);
54   free(fiddly.val);
55 }
56
```

Question 1 ♣   What does function sired computes?

- [ ] the smallest value that is in `phlox` but not in `same`
- [ ] the largest value that is in `same` but not in `phlox`
- [ ] the sum of the values that are in `same` but not in `phlox`
- [ ] the sum of the values that are in `phlox` but not in `same`
- [ ] the number of values that are in `phlox` but not in `same`
- [ ] the sum of the values that are both in `same` and in `phlox`
- [ ] the number of values that are in `same` but not in `phlox`
- [ ] the smallest value that is in `same` but not in `phlox`
- [ ] the number of values that are both in `same` and in `phlox`
- [ ] the smallest value that is both in `phlox` and in `same`
- [ ] the number of values that are both in `phlox` and in `same`
- [ ] the smallest value that is both in `same` and in `phlox`
- [x] the sum of the values that are both in `phlox` and in `same`
- [ ] the largest value that is both in `phlox` and in `same`
- [ ] the largest value that is in `phlox` but not in `same`
- [ ] the largest value that is both in `same` and in `phlox`

Question 2 ♣   What does the program prints?

- [ ] 6
- [ ] 21
- [ ] 20
- [ ] 28
- [ ] 26
- [ ] 10
- [ ] 5
- [ ] 4
- [ ] 1
- [x] 36
- [ ] 9
- [ ] 14
- [ ] 19

Question 3 ♣ What does the program prints?

■ 0	□ 0	□ 0
□ 1	□ 1	□ 1
□ 2	□ 2	□ 2
□ 3	■ 3	□ 3
□ 4	□ 4	□ 4
□ 5	□ 5	□ 5
□ 6	□ 6	■ 6
□ 7	□ 7	□ 7
□ 8	□ 8	□ 8
□ 9	□ 9	□ 9

SAMPLE EXAM ♯5

The purpose of the exam is to analyse this program:

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int cot (Tab rough, Tab returns) {
10   int chaff;
11   int bleeder = 999;
12   int hues;
13   int endive;
14   for (hues=0; hues<rough.len; hues++) {
15     endive = 0;
16     for (chaff=0; chaff<returns.len; chaff++) {
17       if (rough.val[hues] == returns.val[chaff]) {
18         endive = 1;
19         break;
20       }
21     }
22     if ((!endive) && (rough.val[hues] < bleeder)) {
23       bleeder = rough.val[hues];
24     }
25   }
26   return bleeder;
27 }
28
29 int main () {
30   Tab wimpled = {.len=5, .val=malloc(5 * sizeof(int))};
31   Tab buyers = {.len=6, .val=malloc(6 * sizeof(int))};
32   buyers.val[0] = 6;
33   buyers.val[1] = 3;
34   buyers.val[2] = 1;
35   buyers.val[3] = 9;
36   buyers.val[4] = 9;
37   buyers.val[5] = 4;
38   wimpled.val[0] = 6;
39   wimpled.val[1] = 2;
40   wimpled.val[2] = 2;
41   wimpled.val[3] = 1;
42   wimpled.val[4] = 7;
43   printf("%i\n", cot(buyers, wimpled));
44   free(buyers.val);
45   free(wimpled.val);
46 }
```

Question 1 ♣   What does function cot computes?

- [ ] the largest value that is in **rough** but not in **returns**
- [ ] the number of values that are in **returns** but not in **rough**
- [ ] the sum of the values that are in **returns** but not in **rough**
- [ ] the smallest value that is in **returns** but not in **rough**
- [ ] the number of values that are in **rough** but not in **returns**
- [ ] the largest value that is both in **rough** and in **returns**
- [ ] the largest value that is in **returns** but not in **rough**
- [ ] the sum of the values that are both in **rough** and in **returns**
- [ ] the sum of the values that are both in **returns** and in **rough**
- [x] the smallest value that is in **rough** but not in **returns**
- [ ] the number of values that are both in **rough** and in **returns**
- [ ] the smallest value that is both in **rough** and in **returns**
- [ ] the sum of the values that are in **rough** but not in **returns**
- [ ] the number of values that are both in **returns** and in **rough**
- [ ] the smallest value that is both in **returns** and in **rough**
- [ ] the largest value that is both in **returns** and in **rough**

Question 2 ♣   What does the program prints?

- [ ] 21
- [ ] 11
- [x] 3
- [ ] 2
- [ ] 25
- [ ] 4
- [ ] 6
- [ ] 24
- [ ] 7
- [ ] 0
- [ ] 1
- [ ] 9

Question 3 ♣ What does the program prints?

SAMPLE EXAM ♯6

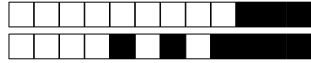The purpose of the exam is to analyse this program:

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int states (Tab insult, Tab atom) {
10   int linked = 999;
11   int itchier;
12   int chassis;
13   int bathe;
14   for (itchier=0; itchier<atom.len; itchier++) {
15     bathe = 0;
16     for (chassis=0; chassis<insult.len; chassis++) {
17       if (atom.val[itchier] == insult.val[chassis]) {
18         bathe = 1;
19         break;
20       }
21     }
22     if (bathe && (linked > atom.val[itchier])) {
23       linked = atom.val[itchier];
24     }
25   }
26   return linked;
27 }

28
29 int main () {
30   Tab callow = {.len=10, .val=malloc(10 * sizeof(int))};
31   Tab quires = {.len=9, .val=malloc(9 * sizeof(int))};
32   callow.val[0] = 2;
33   callow.val[1] = 10;
34   callow.val[2] = 9;
35   callow.val[3] = 8;
36   callow.val[4] = 7;
37   callow.val[5] = 1;
38   callow.val[6] = 4;
39   callow.val[7] = 7;
40   callow.val[8] = 1;
41   callow.val[9] = 1;
42   quires.val[0] = 7;
43   quires.val[1] = 1;
44   quires.val[2] = 8;
45   quires.val[3] = 1;
46   quires.val[4] = 5;
47   quires.val[5] = 5;
48   quires.val[6] = 7;
49   quires.val[7] = 9;
50   quires.val[8] = 6;
51   printf("%i\n", states(callow, quires));
52   free(quires.val);
53   free(callow.val);
54 }
```

Question 1 ♣   What does function states computes?

- [ ] the number of values that are in `insult` but not in `atom`
- [ ] the sum of the values that are in `insult` but not in `atom`
- [ ] the largest value that is both in `insult` and in `atom`
- [ ] the smallest value that is in `atom` but not in `insult`
- [ ] the largest value that is in `insult` but not in `atom`
- [ ] the number of values that are in `atom` but not in `insult`
- [ ] the smallest value that is both in `atom` and in `insult`
- [ ] the sum of the values that are both in `insult` and in `atom`
- [ ] the sum of the values that are in `atom` but not in `insult`
- [ ] the number of values that are both in `insult` and in `atom`
- [ ] the number of values that are both in `atom` and in `insult`
- [ ] the largest value that is in `atom` but not in `insult`
- [ ] the smallest value that is in `insult` but not in `atom`
- [ ] the sum of the values that are both in `atom` and in `insult`
- [ ] the largest value that is both in `atom` and in `insult`
- [x] the smallest value that is both in `insult` and in `atom`

Question 2 ♣   What does the program prints?

- [ ] 34
- [ ] 13
- [ ] 2
- [ ] 16
- [ ] 29
- [x] 1
- [ ] 5
- [ ] 33
- [ ] 7
- [ ] 10
- [ ] 6
- [ ] 21
- [ ] 3
- [ ] 9

Question 3 ♣ What does the program prints?

■ 0	■ 0	□ 0
□ 1	□ 1	■ 1
□ 2	□ 2	□ 2
□ 3	□ 3	□ 3
□ 4	□ 4	□ 4
□ 5	□ 5	□ 5
□ 6	□ 6	□ 6
□ 7	□ 7	□ 7
□ 8	□ 8	□ 8
□ 9	□ 9	□ 9

SAMPLE EXAM ♯7

The purpose of the exam is to analyse this program:

```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  unsigned int len;
  int *val;
} Tab;

int roar (Tab filmed, Tab gerbil) {
  int side;
  int bathing;
  int franc = 0;
  int colors;
  for (colors=0; colors<gerbil.len; colors++) {
    side = 0;
    for (bathing=0; bathing<filmed.len; bathing++) {
      if (gerbil.val[colors] == filmed.val[bathing]) {
        side = 1;
        break;
      }
    }
    if ((gerbil.val[colors] > franc) && side) {
      franc = gerbil.val[colors];
    }
  }
  return franc;
}

int main () {
  Tab maces = {.len=6, .val=malloc(6 * sizeof(int))};
  Tab wood = {.len=6, .val=malloc(6 * sizeof(int))};
  wood.val[0] = 6;
  wood.val[1] = 7;
  wood.val[2] = 7;
  wood.val[3] = 10;
  wood.val[4] = 1;
  wood.val[5] = 5;
  maces.val[0] = 5;
  maces.val[1] = 1;
  maces.val[2] = 2;
  maces.val[3] = 1;
  maces.val[4] = 7;
  maces.val[5] = 1;
  printf("%i\n", roar(wood, maces));
  free(maces.val);
  free(wood.val);
}
```

Question 1 ♣   What does function roar computes?

- ■ the largest value that is both in `filmed` and in `gerbil`
- ☐ the largest value that is in `gerbil` but not in `filmed`
- ☐ the smallest value that is both in `gerbil` and in `filmed`
- ☐ the sum of the values that are both in `gerbil` and in `filmed`
- ☐ the largest value that is both in `gerbil` and in `filmed`
- ☐ the smallest value that is both in `filmed` and in `gerbil`
- ☐ the number of values that are both in `gerbil` and in `filmed`
- ☐ the largest value that is in `filmed` but not in `gerbil`
- ☐ the sum of the values that are in `filmed` but not in `gerbil`
- ☐ the number of values that are both in `filmed` and in `gerbil`
- ☐ the number of values that are in `gerbil` but not in `filmed`
- ☐ the smallest value that is in `gerbil` but not in `filmed`
- ☐ the sum of the values that are both in `filmed` and in `gerbil`
- ☐ the sum of the values that are in `gerbil` but not in `filmed`
- ☐ the smallest value that is in `filmed` but not in `gerbil`
- ☐ the number of values that are in `filmed` but not in `gerbil`

Question 2 ♣   What does the program prints?

☐ 5	☐ 6	☐ 15	☐ 20
☐ 21	■ 7	☐ 10	☐ 1
☐ 2	☐ 4	☐ 29	☐ 16

Question 3 ♣   What does the program prints?

■0	■0	□0
□1	□1	□1
□2	□2	□2
□3	□3	□3
□4	□4	□4
□5	□5	□5
□6	□6	□6
□7	□7	■7
□8	□8	□8
□9	□9	□9

SAMPLE EXAM ♯8

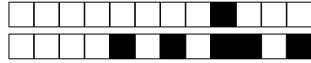The purpose of the exam is to analyse this program:

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int magic (Tab slaw, Tab creeks) {
10   int gable;
11   int shack;
12   int lighten = 999;
13   int agonize;
14   for (agonize=0; agonize<slaw.len; agonize++) {
15     shack = 1;
16     for (gable=0; gable<creeks.len; gable++) {
17       if (slaw.val[agonize] == creeks.val[gable]) {
18         shack = 0;
19         break;
20       }
21     }
22     if ((!shack) && (lighten > slaw.val[agonize])) {
23       lighten = slaw.val[agonize];
24     }
25   }
26   return lighten;
27 }
28
29 int main () {
30   Tab renders = {.len=7, .val=malloc(7 * sizeof(int))};
31   Tab jesting = {.len=10, .val=malloc(10 * sizeof(int))};
32   jesting.val[0] = 4;
33   jesting.val[1] = 5;
34   jesting.val[2] = 2;
35   jesting.val[3] = 2;
36   jesting.val[4] = 8;
37   jesting.val[5] = 3;
38   jesting.val[6] = 5;
39   jesting.val[7] = 10;
40   jesting.val[8] = 2;
41   jesting.val[9] = 1;
42   renders.val[0] = 6;
43   renders.val[1] = 8;
44   renders.val[2] = 4;
45   renders.val[3] = 3;
46   renders.val[4] = 7;
47   renders.val[5] = 7;
48   renders.val[6] = 5;
49   printf("%i\n", magic(jesting, renders));
50   free(jesting.val);
51   free(renders.val);
52 }
```

Question 1 ♣   What does function magic computes?

- [ ] the smallest value that is both in `creeks` and in `slaw`
- [ ] the smallest value that is in `creeks` but not in `slaw`
- [ ] the largest value that is both in `creeks` and in `slaw`
- [ ] the largest value that is both in `slaw` and in `creeks`
- [ ] the number of values that are in `creeks` but not in `slaw`
- [ ] the number of values that are in `slaw` but not in `creeks`
- [ ] the sum of the values that are both in `creeks` and in `slaw`
- [ ] the sum of the values that are in `creeks` but not in `slaw`
- [ ] the largest value that is in `creeks` but not in `slaw`
- [x] the smallest value that is both in `slaw` and in `creeks`
- [ ] the number of values that are both in `slaw` and in `creeks`
- [ ] the sum of the values that are in `slaw` but not in `creeks`
- [ ] the largest value that is in `slaw` but not in `creeks`
- [ ] the number of values that are both in `creeks` and in `slaw`
- [ ] the sum of the values that are both in `slaw` and in `creeks`
- [ ] the smallest value that is in `slaw` but not in `creeks`

Question 2 ♣   What does the program prints?

- [ ] 15
- [ ] 10
- [ ] 25
- [ ] 8
- [ ] 1
- [ ] 13
- [ ] 7
- [ ] 16
- [ ] 5
- [ ] 19
- [ ] 20
- [x] 3
- [ ] 6
- [ ] 4
- [ ] 17
- [ ] 0

Question 3 ♣ What does the program prints?

SAMPLE EXAM ♯9

The purpose of the exam is to analyse this program:
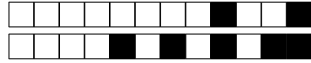
```c
#include <stdio.h>
#include <stdlib.h>

typedef struct {
  unsigned int len;
  int *val;
} Tab;

int licked (Tab buggier, Tab posh) {
  int malts = 0;
  int outpost;
  int alibied;
  int legal;
  for (legal=0; legal<posh.len; legal++) {
    outpost = 0;
    for (alibied=0; alibied<buggier.len; alibied++) {
      if (posh.val[legal] == buggier.val[alibied]) {
        outpost = 1;
        break;
      }
    }
    if (outpost) {
      malts++;
    }
  }
  return malts;
}

int main () {
  Tab navels = {.len=10, .val=malloc(10 * sizeof(int))};
  Tab profuse = {.len=5, .val=malloc(5 * sizeof(int))};
  profuse.val[0] = 4;
  profuse.val[1] = 6;
  profuse.val[2] = 10;
  profuse.val[3] = 1;
  profuse.val[4] = 10;
  navels.val[0] = 4;
  navels.val[1] = 7;
  navels.val[2] = 1;
  navels.val[3] = 3;
  navels.val[4] = 8;
  navels.val[5] = 5;
  navels.val[6] = 5;
  navels.val[7] = 10;
  navels.val[8] = 1;
  navels.val[9] = 2;
  printf("%i\n", licked(profuse, navels));
  free(profuse.val);
  free(navels.val);
}
```

Question 1 ♣   What does function licked computes?

- [ ] the sum of the values that are both in `buggier` and in `posh`
- [ ] the number of values that are in `posh` but not in `buggier`
- [ ] the smallest value that is in `buggier` but not in `posh`
- [ ] the smallest value that is in `posh` but not in `buggier`
- [ ] the largest value that is in `posh` but not in `buggier`
- [ ] the sum of the values that are in `buggier` but not in `posh`
- [ ] the sum of the values that are in `posh` but not in `buggier`
- [ ] the largest value that is both in `buggier` and in `posh`
- [ ] the number of values that are both in `posh` and in `buggier`
- [ ] the smallest value that is both in `posh` and in `buggier`
- [ ] the largest value that is in `buggier` but not in `posh`
- [ ] the number of values that are in `buggier` but not in `posh`
- [ ] the smallest value that is both in `buggier` and in `posh`
- [ ] the sum of the values that are both in `posh` and in `buggier`
- [ ] the largest value that is both in `posh` and in `buggier`
- [x] the number of values that are both in `buggier` and in `posh`

Question 2 ♣   What does the program prints?

- [ ] 6
- [ ] 5
- [x] 4
- [ ] 30
- [ ] 21
- [ ] 1
- [ ] 10
- [ ] 16
- [ ] 8
- [ ] 25
- [ ] 2
- [ ] 22

Question 3 ♣   What does the program prints?

■0	■0	□0
□1	□1	□1
□2	□2	□2
□3	□3	□3
□4	□4	■4
□5	□5	□5
□6	□6	□6
□7	□7	□7
□8	□8	□8
□9	□9	□9

SAMPLE EXAM ♯10

The purpose of the exam is to analyse this program:

```c
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct {
5    unsigned int len;
6    int *val;
7  } Tab;
8
9  int gyrated (Tab depute, Tab corneas) {
10   int cellars;
11   int discuss = 0;
12   int plush;
13   int alien;
14   for (cellars=0; cellars<depute.len; cellars++) {
15     plush = 0;
16     for (alien=0; alien<corneas.len; alien++) {
17       if (depute.val[cellars] == corneas.val[alien]) {
18         plush = 1;
19         break;
20       }
21     }
22     if (!plush) {
23       discuss += depute.val[cellars];
24     }
25   }
26   return discuss;
27 }
28
29 int main () {
30   Tab charms = {.len=8, .val=malloc(8 * sizeof(int))};
31   Tab impels = {.len=6, .val=malloc(6 * sizeof(int))};
32   charms.val[0] = 4;
33   charms.val[1] = 2;
34   charms.val[2] = 4;
35   charms.val[3] = 6;
36   charms.val[4] = 4;
37   charms.val[5] = 9;
38   charms.val[6] = 4;
39   charms.val[7] = 10;
40   impels.val[0] = 1;
41   impels.val[1] = 2;
42   impels.val[2] = 10;
43   impels.val[3] = 4;
44   impels.val[4] = 7;
45   impels.val[5] = 8;
46   printf("%i\n", gyrated(charms, impels));
47   free(impels.val);
48   free(charms.val);
49 }
50
```

Question 1 ♣ What does function gyrated computes?

- [ ] the largest value that is in depute but not in corneas
- [ ] the number of values that are in depute but not in corneas
- [ ] the sum of the values that are in corneas but not in depute
- [x] the sum of the values that are in depute but not in corneas
- [ ] the largest value that is in corneas but not in depute
- [ ] the number of values that are both in depute and in corneas
- [ ] the number of values that are both in corneas and in depute
- [ ] the number of values that are in corneas but not in depute
- [ ] the smallest value that is in depute but not in corneas
- [ ] the smallest value that is in corneas but not in depute
- [ ] the largest value that is both in depute and in corneas
- [ ] the smallest value that is both in corneas and in depute
- [ ] the sum of the values that are both in depute and in corneas
- [ ] the sum of the values that are both in corneas and in depute
- [ ] the smallest value that is both in depute and in corneas
- [ ] the largest value that is both in corneas and in depute

Question 2 ♣ What does the program prints?

- [ ] 6
- [ ] 18
- [ ] 28
- [ ] 7
- [ ] 8
- [ ] 4
- [ ] 3
- [ ] 9
- [ ] 1
- [ ] 2
- [ ] 10
- [ ] 16
- [ ] 13
- [x] 15

Question 3 ♣   What does the program prints?