SAMPLE EXAM ♯1

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 highest (Tab lazied, Tab piecing) {
  int cooky;
  int bifocal;
  int sibyls;
  int vast = 0;
  for (sibyls=0; sibyls<piecing.len; sibyls++) {
    bifocal = 0;
    for (cooky=0; cooky<lazied.len; cooky++) {
      if (piecing.val[sibyls] == lazied.val[cooky]) {
        bifocal = 1;
        break;
      }
    }
    if (!bifocal) {
      vast += piecing.val[sibyls];
    }
  }
  return vast;
}

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

```

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`
- ☐ 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
☐ 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
#include <stdio.h>
#include <stdlib.h>

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

int junkie (Tab dislike, Tab news) {
  int severer;
  int undo = 0;
  int dizzy;
  int abode;
  for (abode=0; abode<news.len; abode++) {
    dizzy = 0;
    for (severer=0; severer<dislike.len; severer++) {
      if (news.val[abode] == dislike.val[severer]) {
        dizzy = 1;
        break;
      }
    }
    if ((!dizzy) && (news.val[abode] > undo)) {
      undo = news.val[abode];
    }
  }
  return undo;
}

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

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	☐ 2	☐ 1	☐ 7
☐ 4	☐ 14	☐ 16	☐ 11
☐ 5	☐ 22	☐ 9	☐ 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:

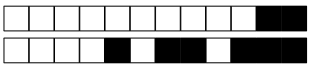
```c
#include <stdio.h>
#include <stdlib.h>

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

int belie (Tab endue, Tab moodily) {
  int jerkin = 0;
  int nouns;
  int accord;
  int inn;
  for (accord=0; accord<endue.len; accord++) {
    nouns = 0;
    for (inn=0; inn<moodily.len; inn++) {
      if (endue.val[accord] == moodily.val[inn]) {
        nouns = 1;
        break;
      }
    }
    if (!nouns) {
      jerkin += endue.val[accord];
    }
  }
  return jerkin;
}

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

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`
- ☐ 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	☐ 24	☐ 5	☐ 11
☐ 9	☐ 23	☐ 21	☐ 7
☐ 4	☐ 10	☐ 26	☐ 3
☐ 2	☐ 16	☐ 1	



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

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

int sired (Tab phlox, Tab same) {
  int clawed;
  int hauls = 0;
  int wattle;
  int meaner;
  for (meaner=0; meaner<same.len; meaner++) {
    wattle = 1;
    for (clawed=0; clawed<phlox.len; clawed++) {
      if (same.val[meaner] == phlox.val[clawed]) {
        wattle = 0;
        break;
      }
    }
    if (!wattle) {
      hauls += same.val[meaner];
    }
  }
  return hauls;
}

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

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`
- ☐ 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
- ☐ 26
- ☐ 4
- ☐ 9
- ☐ 21
- ☐ 10
- ☐ 1
- ☐ 14
- ☐ 20
- ☐ 5
- ☐ 36
- ☐ 19
- ☐ 28

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

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

int cot (Tab rough, Tab returns) {
  int chaff;
  int bleeder = 999;
  int hues;
  int endive;
  for (hues=0; hues<rough.len; hues++) {
    endive = 0;
    for (chaff=0; chaff<returns.len; chaff++) {
      if (rough.val[hues] == returns.val[chaff]) {
        endive = 1;
        break;
      }
    }
    if ((!endive) && (rough.val[hues] < bleeder)) {
      bleeder = rough.val[hues];
    }
  }
  return bleeder;
}

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

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`
- ☐ 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	☐ 2	☐ 6	☐ 0
☐ 11	☐ 25	☐ 24	☐ 1
☐ 3	☐ 4	☐ 7	☐ 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 ♯6

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 states (Tab insult, Tab atom) {
  int linked = 999;
  int itchier;
  int chassis;
  int bathe;
  for (itchier=0; itchier<atom.len; itchier++) {
    bathe = 0;
    for (chassis=0; chassis<insult.len; chassis++) {
      if (atom.val[itchier] == insult.val[chassis]) {
        bathe = 1;
        break;
      }
    }
    if (bathe && (linked > atom.val[itchier])) {
      linked = atom.val[itchier];
    }
  }
  return linked;
}

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

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`
- ☐ the smallest value that is both in `insult` and in `atom`

Question 2 ♣ What does the program prints?

- ☐ 34
- ☐ 13
- ☐ 2
- ☐ 16
- ☐ 29
- ☐ 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:

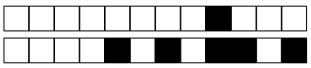
```c
#include <stdio.h>
#include <stdlib.h>

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

int magic (Tab slaw, Tab creeks) {
  int gable;
  int shack;
  int lighten = 999;
  int agonize;
  for (agonize=0; agonize<slaw.len; agonize++) {
    shack = 1;
    for (gable=0; gable<creeks.len; gable++) {
      if (slaw.val[agonize] == creeks.val[gable]) {
        shack = 0;
        break;
      }
    }
    if ((!shack) && (lighten > slaw.val[agonize])) {
      lighten = slaw.val[agonize];
    }
  }
  return lighten;
}

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

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`
- ☐ 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	☐ 1	☐ 5	☐ 6
☐ 10	☐ 13	☐ 19	☐ 4
☐ 25	☐ 7	☐ 20	☐ 17
☐ 8	☐ 16	☐ 3	☐ 0



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 ♯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`
- ☐ the number of values that are both in `buggier` and in `posh`

Question 2 ♣ What does the program prints?

☐ 6	☐ 5	☐ 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:

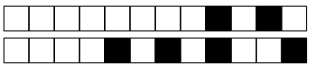
```c
#include <stdio.h>
#include <stdlib.h>

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

int gyrated (Tab depute, Tab corneas) {
  int cellars;
  int discuss = 0;
  int plush;
  int alien;
  for (cellars=0; cellars<depute.len; cellars++) {
    plush = 0;
    for (alien=0; alien<corneas.len; alien++) {
      if (depute.val[cellars] == corneas.val[alien]) {
        plush = 1;
        break;
      }
    }
    if (!plush) {
      discuss += depute.val[cellars];
    }
  }
  return discuss;
}

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

```

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`
- ☐ 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
- ☐ 15



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