Computation I 5EIA0
C Exercises for Week 1
(v0.2, September 6, 2021)
Solutions

1 Introduction

The solutions are surrounded by the following standard code.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <math.h>
int main (void) {
  // code is inserted here
}
```

2 Print statements

1. Write a program that prints
   ```
   Hello, there!
   ```
   using a single printf statement.
   ```
   printf("Hello, there!\n");
   ```

2. Write a program that prints
   ```
   Hello, there!
   How are you?
   ```
   using a single printf statement.
   ```
   printf("Hello, there!\nHow are you?\n");
   ```

3. Write a program that prints
   ```
   Hello!
   How are you?
   ```
   using as few printf statements possible. Each statement prints at most five characters on the screen.
   ```
     printf("Hello");
     printf("\nHow ");
     printf("are y");
     printf("ou?\n");
     // note that the newline character '\n' is a single character
   ```

4. Write a program that declares integer i1 with initial value 10 and integer i2 with initial value 200. Print the integers as follows
   ```
   i1=10 i2=200
   ```
   using a single printf statement.
   ```
   int i1 = 10, i2 = 200;
   printf("i1=%d i2=%d\n", i1, i2);
   ```

5. Write a program that declares integers i1, i2, i3, i4 with initial values 10, 200, 3000, 40000, respectively. Generate the following output:

```
10
200
3000
40000
```

using four *identical* `printf` statements, i.e. the format string as well as the variable that's printed must be identical. (Hint: use an intermediate variable `i`.)

```
int i, i1 = 10, i2 = 200, i3 = 3000, i4 = 40000;
i = i1; printf("%d\n", i);
i = i2; printf("%d\n", i);
i = i3; printf("%d\n", i);
i = i4; printf("%d\n", i);
```

6. Write a program that declares integers i1, i2, i3, i4 with initial values 10, 200, 3000, 40000, respectively. Generate the following output:

```
   10
  200
 3000
40000
```

using four *identical* `printf` statements, i.e. the format string as well as the variable that's printed must be identical. (Hint: use an intermediate variable `i`, and see the `printf` format string on page 11 of K&R.)

```
int i, i1 = 10, i2 = 200, i3 = 3000, i4 = 40000;
i = i1; printf("%5d\n", i);
i = i2; printf("%5d\n", i);
i = i3; printf("%5d\n", i);
i = i4; printf("%5d\n", i);
```

3 Loops

1. Write a program that prints the integers 0 to 9 using a single `printf` statement in a `for` loop. Use the increment operator (++) to update the loop variable.

```
int i; for (i = 0; i < 9; i++) { printf("%d\n", i); }
```

2. Write a program that prints the integers 9 to 0 using a single `printf` statement in a `for` loop. Use the increment operator (++) to update the loop variable.

```
int i; for (i = 0; i < 9; i++) { printf("%d\n", 9-i); }
```

3. Write a program that prints the integers 9 to 0 using a single `printf` statement in a `for` loop. Use => in the termination condition.

```
int i; for (i = 9; i >= 0; i--) { printf("%d\n", i); }
```

4. Write a program that prints the integers 9 to 0 using a single `printf` statement in a `for` loop. Use the decrement operator (−−) to update the loop variable.

```
int i; for (i = 9; i > -1; i--) { printf("%d\n", i); }
```

5. Write a program that prints the integers 9 to 0 using a single `printf` statement in a `for` loop. Use the increment operator (++) to update the loop variable. Use => in the termination condition.

```
int i; for (i = 0; 9 >= i; i++) { printf("%d\n", 9-i); }
```

6. Write a program that prints the integers 0 to 9 using a single `printf` statement in a `while` loop. Use the increment operator (++) to update the loop variable.

```
int i = 0; while (i < 9) { printf("%d\n", i); i++; }
```

7. Write a program that prints the integers 0 to 9 using a single `printf` statement in a `do/while` loop. Use the increment operator (++) to update the loop variable.

```
int i = 0; do { printf("%d\n", i); i++; } while (i <= 9);
```

8. Write a program that prints the integers 9 to 0 using a single `printf` statement in a `do/while` loop. Use the decrement operator (`--`) to update the loop variable.

   ```
   int i = 9; do { printf("%d\n", i); i--; } while (i >= 0);
   ```

9. Write a program that prints the integers 9 to 0 using a single `printf` statement in a `do/while` loop. Use the decrement operator (`--`) to update the loop variable.

   ```
   int i = 9; do { i--; printf("%d\n", i); } while (i > 0);
   ```

10. Write a program that does nothing but loop forever, using a `for` loop. Use no loop variables. You can use `^C` (control-C) to stop the program.

    ```
    for (;;);
    ```

11. Write a program that prints `Hello!` forever, using a `for` loop. You can use `^C` (control-C) to stop the program.

    ```
      for (;;) printf("Hello!\n");
    ```

12. Write a program that prints `Hello!` forever, using a `while` loop. You can use `^C` (control-C) to stop the program.

    ```
      while(1) printf("Hello!\n");
    ```

13. Write a program that prints `Hello!` forever, using a `do/while` loop. You can use `^C` (control-C) to stop the program.

    ```
      do { printf("Hello!\n"); } while (1);
    ```

14. Write a program that generates the following output using `for` loops and printing one character at a time.

    ```
    *
    **
    ***
    ****
    *****
    ******
    ```

    ```
    for (int i = 0; i < 6; i++) {
      for (int j = 0; j < i; j++) {
        printf("*");
      }
      printf("\n");
    }
    ```

15. Write a program that generates the following output using three `for` loops and printing one character at a time.

    ```
         *
        **
       ***
      ****
     *****
    ******
    ```

    ```
    for (int i = 0; i < 6; i++) {
      for (int j = 1; j < 6-i; j++) { printf(" "); }
      for (int j = 0; j <= i; j++) { printf("*"); }
      printf("\n");
    }
    ```

16. Write a program that generates the following output using three `for` loops and printing one character at a time.

    ```
    ******
     *****
      ****
       ***
        **
         *
    ```

```
for (int i = 5; i >= 0; i--) {
  for (int j = 1; j < 6-i; j++) { printf(" "); }
  for (int j = 0; j <= i; j++) { printf("*"); }
  printf("\n");
}
```

17. Write a program that generates the following output using two `for` loops and printing one character at a time.

```
******
*****
****
***
**
*
```

```
for (int i = 6; i > 0; i--) {
  for (int j = 0; j < i; j++) {
    printf("*");
  }
  printf("\n");
}
```

18. Write a program that generates the following output using three `for` loops and printing one character at a time.

```
     *
    ***
   *****
  *******
 *********
***********
```

```
for (int i = 0; i < 6; i++) {
  for (int j = 1; j < 6-i; j++) { printf(" "); }
  for (int j = 0; j <= i*2; j++) { printf("*"); }
  printf("\n");
}
```

19. Write a program that generates the following output using three `for` loops.

```
     O
    O*O
   O*O*O
  O*O*O*O
 O*O*O*O*O
O*O*O*O*O*O
```

```
for (int i = 0; i < 6; i++) {
  for (int j = 1; j < 6-i; j++) { printf(" "); }
  for (int j = 0; j < i; j++) { printf("O*"); }
  printf("O\n");
}
```

4 Operators

1. Write a program that generates the following output using a single `printf` statement in a loop. (Hint: use the modulo operator.)

```
0 0 0
1 1 1
2 2 0
3 0 1
4 1 0
5 2 1
6 0 0
7 1 1
8 2 0
9 0 1
```

```
int i; for (i = 0; i < 10; i++) { printf("%d %d %d\n", i, i%3, i%2); }
```

2. Write a program that generates the following output using a single `printf` statement in a loop. (Hint: use the modulo operator.)

```
0=3*0+0
1=3*0+1
2=3*0+2
3=3*1+0
4=3*1+1
5=3*1+2
6=3*2+0
7=3*2+1
8=3*2+2
9=3*3+0
```

```
int i; for (i = 0; i < 10; i++) { printf("%d=3*%d+%d\n", i, i/3, i%3); }
```

5 Input

1. Write a program that asks for an integer and prints it out.

```
Integer? 10
You entered: 10
```

```
int i;
printf("Integer? ");     // no newline '\n'
scanf("%d", &i);         // note the &
printf("You entered: %d\n", i);
```

2. What happens if you type something that is not an integer, such as Hello? (Hint: you get garbage because no value is read into i.

```
Integer? Hello
You entered: <uninitialised value that could be anything>
```

What happens if you initialise the integer with 999 and then enter an incorrect (non-integer) input? (Hint: it keeps its value.)

```
Integer? Hello
You entered: 999
```

You may accidentally run into this problem when programming. For this reason it is always a good idea to initialise your variables. Otherwise you may get different random values every time your program runs, making it hard to debug.)

3. Write a program that asks for two integers using two `scanf` statements. Print the two integers.

```
Integer one? 10
Integer two? 20
You entered 10 and 20
```

What happens when you enter both integers on the first line? Why?

```
Integer one? 10 20
...
```

```
int i1, i2;
printf("Integer one? ");     // no newline '\n'
scanf("%d", &i1);            // note the &
printf("Integer two? ");     // no newline '\n'
scanf("%d", &i2);            // note the &
printf("You entered %d and %d\n", i1, i2);
```

4. Write a program that asks for two integers using *one* `scanf` statements. Print the two integers.

```
Two integers? 10 20
You entered 10 and 20
```

What happens when you add some spaces, tabs and empty lines in between? Why does it still work?

```
Two integers?
 10


       20
You entered 10 and 20
```

These are all "white space" that are skipped by `scanf`.

```
int i1, i2;
printf("Integer one? ");
scanf("%d %d", &i1, &i2);
printf("You entered %d and %d\n", i1, i2);
```

6 If statement

1. Write a program that asks for an integer and prints out if it is negative, zero, or positive.

```
Integer? 10
10 is a positive number
```

```
int i = 0;
scanf("%d", i);
if (i < 0) {
  printf("%d is a negative number\n");
} else {
  if (i == 0) {
    printf("%d is zero\n");
  }
  else {
    printf("%d is a negative number\n");
  }
}
```

Or:

```
if (i < 0) printf("%d is a negative number\n");
else if (i == 0) { printf("%d is zero\n");
lse printf("%d is a negative number\n");
```

2. Write a program that asks for an integer and prints out if it is negative, zero, or positive. Now use the negation (`!`) operator in every condition. Make sure that you test your program for all cases (e.g. with the values -1, 0, 1 as input).

```
Integer? -10
-10 is a negative number
```

```
int i = 0;
scanf("%d", i);
if (!(i > 0 || i == 0)) {
  // alternatives:
  // if (!(i > 0) && !(i == 0)) {
  // if (!(i > 0) && i != 0) {
  // if (!(i >= 0)) {
  printf("%d is a negative number\n");
} else {
  if (!(i < 0 || i == 0)) {
    // alternatives:
    // if (!(i < 0) && !(i == 0)) {
    // if (!(i < 0) && i != 0) {
    // if (!(i <= 0)) {
    printf("%d is a negative number\n");
  }
  else {
    printf("%d is zero\n");
  }
}
```

3. Write a program that uses a `for` loop from 0 to 6 and prints the following. Use `if` statements. You can split up the printing of a line in multiple print statements, but use only three print statements with a newline. (Hint: remember the modulo operator.)

```
0 is even and less than 3
1 is odd and less than 3
2 is even and less than 3
3 is odd and equal to 3
4 is even and greater than 3
5 is odd and greater than 3
6 is even and greater than 3
```

```
int i;
for (i = 0; i < 6; i++) {
  if (i % 2 == 0) printf("%d is even", i); else printf("%d is odd", i);
  if (i < 3) {
    printf(" and less than 3\n");
  } else {
    if (i > 3)
      printf(" and greater than 3\n");
    else
      printf(" and equal to 3\n");
  }
}
```

4. Write a program that uses a `for` loop from 0 to 6 and prints the following. Use `if` statements. Now every `printf` statement must print a complete line. (Hint: use the `&&` operator in the `if` statements. You will need five `if` statements.)

```
0 is even and less than 3
1 is odd and less than 3
2 is even and less than 3
3 is odd and equal to 3
4 is even and greater than 3
5 is odd and greater than 3
6 is even and greater than 3
```

```
for (i = 0; i < 6; i++) {
  if (i % 2 == 0 && i < 3) printf("%d is even and less than 3\n", i);
  if (i % 2 == 0 && i > 3) printf("%d is even and greater than 3\n", i);
  if (i % 2 == 1 && i < 3) printf("%d is odd and less than 3\n", i);
  if (i % 2 == 1 && i > 3) printf("%d is odd and greater than 3\n", i);
  if (i == 3) printf("3 is odd and equal to 3\n");
  // note that the "else" are optional, since all conditions are mutually exclusive
}
```

5. Replace all && operators by || operators in the previous program. (Hint: De Morgan's law states that A and B is the same as not((not A) or (not B)).)

```
0 is even and less than 3
1 is odd and less than 3
2 is even and less than 3
3 is odd and equal to 3
4 is even and greater than 3
5 is odd and greater than 3
6 is even and greater than 3
```

```
for (i = 0; i < 6; i++) {
  if (i == 3) printf("3 is odd and equal to 3\n");
  else if (!(i % 2 != 0 || i > 3)) printf("%d is even and less than 3\n", i);
  else if (!(i % 2 != 0 || i < 3)) printf("%d is even and greater than 3\n", i);
  else if (!(i % 2 != 1 || i > 3)) printf("%d is odd and less than 3\n", i);
  else if (!(i % 2 != 1 || i < 3)) printf("%d is odd and greater than 3\n", i);
  // note that the "else" are NOT optional, since all conditions are NOT mutually exclusive
  // try removing them!
}
```

7 More complex programs

1. Write a program that prints the Fibonacci series: 1 1 2 3 5 8 13 21 34 55 89, etc. up to 100. Each number is the sum of the two preceeding numbers. (Hint: use three variables: `first`, `second`, `sum`, and a `while` loop. Initialise `first` and `second` to 1. In the loop, print the sum and re-assign the variables.)

```
int first = 1, second = 1, sum = 2;
printf("1\n1\n");
while (sum < 100) {
  printf("%d\n", sum);
  first = second;
  second = sum;
  sum = first + second;
}
```

Alternative solution with a `for` loop:

```
printf("1\n1\n");
for (int first=1, second=1; first+second < 100;) {
  int sum = first + second;
  printf("%d\n", sum);
  first = second;
  second = sum;
}
```

2. Write a program that checks if a number is a prime. A prime number is divisible only by 1 and itself. In other words, it has two divisors. (Hint: write a `for` loop that counts the number of divisors by using the modulo operator and an `if` statement.) Run your program on several numbers, e.g. 11, 71, 63091, 63097, 99901.

```
int i = 11;
int divisors = 0;
for (int d=1; d <= i; d++) {
  if (i % d == 0) divisors++;
}
if (divisors == 2)
  printf("%d is a prime\n", i);
else
  printf("%d is not a prime\n", i);
```

3. Optimise the previous program to minimise the number of modulo operations. (Hint: you know that 1 and the number are always divisors, so there is no need to check this. Also, you can stop as soon as the number of divisors is larger than two. Use the `&&` operator in the loop condition.)

```
int i = 11;
int divisors = 2;
for (int d=2; d < i && divisors == 2; d++) {
  if (i % d == 0) divisors++;
}
if (divisors == 2)
  printf("%d is a prime\n", i);
else
  printf("%d is not a prime\n", i);
```

4. Now print all primes up to one million by adding a loop. (Hint: there are 78498 of them; it took 2 minutes on my computer.)

5. Ask for a positive integer and print it as a Roman number. Exit the program when you enter 0. Use a `do/while` loop. You can make the simplifying assumption that a larger digit is never printed after a smaller one.

```
Number? 1
I
Number? 4
IIII
Number? 5
V
Number? 9
VIIII
Number? 10
X
Number? 49
XXXXVIIII
Number? 50
L
Number? 100
C
Number? 500
D
Number? 1000
M
Number? 1966
MDCCCCLXVI
Number? 2021
MMXXI
Number? 0
Bye!
```

Hint: to print out the right number of e.g. `"M"` you can use the following:

```
while (i >= 1000) { printf("M"); i = i - 1000; }
```

```
int stop = 0;
do {
  int i;
  printf("Number? ");
  scanf("%d", &i);
  if (i == 0) stop = 1;
  while (i >= 1000) { printf("M"); i -= 1000; }
  while (i >= 500) { printf("D"); i -= 500; }
  while (i >= 100) { printf("C"); i -= 100; }
  while (i >= 50) { printf("L"); i -= 50; }
  while (i >= 10) { printf("X"); i -= 10; }
  while (i >= 5) { printf("V"); i -= 5; }
  while (i >= 1) { printf("I"); i -= 1; }
  if (!stop) printf("\n");
} while (!stop);
printf("Bye!\n");
```

6. `scanf` returns the number of items that it successfully read. Use the return value of `scanf` to check that the user entered a valid value.

```
Integer? Hello
You did not enter an integer!
```

```
int i, valid_values;
printf("Integer? ");
valid_values = scanf("%d", &i);
if (valid_values == 1)
  printf("You entered: %d\n", i);
else
  printf("You did not enter an integer!\n");
```

- **19/1 v0.1** First version.
- **6/9 v0.2** Removed some superfluous conditions.