Exercises

Ex 1.

Write a program that:

1. Asks user to enter the price of a bus ticket
2. Asks user to enter the price of a taxi trip
3. Asks user to enter how much money he/she has
4. If user has not enough money for either type tells user to walk and then the program stops
5. Asks user to choose taxi or bus (use numbers for selection)
6. Checks if user has enough money for the selected trip type
    - If User has enough money reduce the money by the selected trip's price and print how much money is left, and go back to step 4
    - If user has not enough money for the selected trip type then tell that to user and go back to step 4

For example:

```
Enter price of bus ticket: 3.50
Enter price of taxi: 25.70
How much money you have: 30
You have 30.00 euros left.
Do you want to take
 1) bus (3.50 euros)
 2) taxi (25.70 euros)
Enter your selection: 2
You chose taxi.
You have 4.30 euros left.
Do you want to take
 1) bus (3.50 euros)
 2) taxi (25.70 euros)
Enter your selection: 2
You chose taxi.
You don't have enough money for taxi.
You have 4.30 euros left.
Do you want to take
 1) bus (3.50 euros)
 2) taxi (25.70 euros)
Enter your selection: 1
You chose bus.
You have 0.80 euros left.
You need to walk. Bye
```

Ex 2.

Write a program that defines two floating-point arrays of 12 elements each. The program then asks user to enter tax percentage and yearly income limit up to which the percentage applies to and the (greater) tax percentage that will be applied to income that is over the limit. Then program asks user to enter salary of each month and stores the values in the array. After user has entered all monthly salaries, the program calculates tax for each month.

The program prints all salaries with two decimal precision and the amount of tax for each month.

Example: (three dots indicates rows not shown in this example):

```
Enter tax rate: 18.0
Enter income limit: 10000
Enter tax rate for income over the limit: 36.0
Enter income for month 1: 998
...
Enter income for month 6: 1489.51
...
Enter income for month 12: 998
month     income      tax
    1     998.00   179.64
    2     998.00   179.64
...
    6    1489.51   268.11
...
   12     998.00   359.28
```

You need to get familiar with printf field width specifiers to complete this!

Note that decimal points must be aligned as shown above!

The tricky part is the month where (if) the total income goes over the limit. That month part of salary is taxed with lower rate and part of the salary with higher rate.

Ex 3.

That asks number of students and creates an array of that many integers.

Program then initializes the array elements to -1.

Then the program asks user to enter student number in range 1 – nr of students or zero to stop. If valid non zero number is given the program asks user to enter grade in the range 0-5 or -1 to not enter a grade for the student.

When user stops by entering 0 then program prints student numbers followed by grade or N/A if grade is -1.

For example:

```
How many students: 4

Enter student number (1 – 4) or 0 to stop: 1
Enter grade (0 – 5) for student 1 or -1 to cancel: 2

Enter student number (1 – 4) or 0 to stop: 6
Invalid student number!

Enter student number (1 – 4) or 0 to stop: 3
Enter grade (0 – 5) for student 3 or -1 to cancel: 7
Invalid grade!
Enter grade (0 – 5) for student 3 or -1 to cancel: 5


Enter student number (1 – 4) or 0 to stop: 2
Enter grade (0 – 5) for student 2 or -1 to cancel: -1

Enter student number (1 – 4) or 0 to stop: 0

Student   Grade
1         2
2         N/A
3         5
4         N/A
```

Notes about input validation

When reading numbers with scanf it is possible to accidentally create an infinite loop. For example:

```
int selection = 0;
do {
    printf("Enter selection: ");
    scanf("%d", &selection);
} while(selection == 0);
```

In the code above if you accidentally enter any other character than number or white space then code will get stuck in an infinite loop.

The code below shows a workaround that will ignore all characters in the input buffer until the end of line is met. This will clear all incorrect input from the input buffer and prevents infinite loops.

```
int selection = 0;
do {
    printf("Enter selection: ");
    if (scanf("%d", &selection) != 1) {
        while(getchar() != '\n');
        printf("invalid input\n");
    }
} while(selection == 0);
```

Ex4

Write a function called **read_integer** that takes no parameters and returns an integer. The function reads a single integer. If the reading fails the function prints “invalid input” and tries to read an integer again. When an integer is successfully read the function returns the integer.

The declaration of the function is:

```
int read_integer(void);
```

Write a program that asks user to enter positive integers or a negative number to stop. The program must use read_integer-function to read the number. The program calculates and prints the average of the entered numbers excluding the negative number. For example, if user enters: **1 3 4 2 7 -1**, the program prints 3.4 as the average.

The program must print the average with three decimal precision.

```
Enter positive numbers or negative to stop: 1
Enter positive numbers or negative to stop: 3
Enter positive numbers or negative to stop: 4
Enter positive numbers or negative to stop: 2
Enter positive numbers or negative to stop: 7
Enter positive numbers or negative to stop: -1
You entered 5 positive numbers. The average is: 3.400
```

Ex5

Write a function **read_range** that takes two integer parameters and returns an integer. The function asks user to enter a number in the range given as parameters. The first parameter is the lowest acceptable value and the second is the highest acceptable value. When user enters a number the input is validated in the following ways:

- A number was successfully read
- The number is in the specified range

If the input is invalid then a descriptive error message is printed and user is asked to enter number again. The function returns only if correct input is given. The function returns the entered number. The declaration of the function is:

```
int read_range(int low, int high);
```

Write a program that "plays" a dice game with user. The game consists of three rounds where program asks user to roll a die and enter the result (1 – 6). The program must use read_range to validate input.

The program cheats and always claims to have one higher than what user rolled except when user rolls six – then program says that it is a tie.

```
Let's play!
Roll a die and enter your result.
Enter a number between 1 and 6: 2
I got 3. I win!
Roll a die and enter your result.
Enter a number between 1 and 6: 5
I got 6. I win!
Roll a die and enter your result.
Enter a number between 1 and 6: 6
I got 6. It is a tie!
Better luck next time. Bye!
```

Ex6

Write a program that prints a menu and asks user to select an operation. The operations are:

- Roll D6
- Roll D10
- Quit

Input must be validated and a descriptive error message must be printed if input is invalid. Rolling D6 prints a random number in the range of 1 – 6 and rolling D10 prints a random number in the range of 1 – 10.

Use functions to structure your program.

See: https://en.cppreference.com/w/c/numeric/random/rand for how to produce random numbers in a range.

Ex7

Write a function named read_positive. The function takes a pointer to int as a parameter and returns a bool.

```
bool read_positive(int *value);
```

The function asks (=prints) user to enter a positive number. If user enters a valid number and the number is positive the function stores the number using the pointer and returns true. If reading number fails or number is not positive function returns false without making changes to the value using pointer.

Write a program that "plays" a guessing game with user. The program asks user to guess how much money it has and after the guess claims to twice as much plus 20 euros. Program must use the function above to ask user to enter the number. If reading fails, function returns false, program asks to enter a number again. Program stops when user has entered an incorrect value three times.

```
Guess how much money I have!
Enter a positive number: 50
You didn't get it right. I have 120 euros.
Guess how much money I have!
Enter a positive number: xd
Incorrect input
Guess how much money I have!
Enter a positive number: 70
You didn't get it right. I have 160 euros.
Guess how much money I have!
Enter a positive number: fubar
Incorrect input
Guess how much money I have!
Enter a positive number: not
Incorrect input
I give up! See you later!
```

Ex8

Write a function called print_numbers that takes two parameters pointer to constant integers and number of integers to print.

void print_numbers(const int *array, int count);

The function prints the numbers on separate lines in eight characters wide field.

Write a program that defines an array of 15 integers and fills the array with random numbers and then uses the function to print the array contents.

Ex9

Write a function called find_first that takes three parameters: pointer to constant unsigned integers and an unsigned integer to find and returns an integer.

int find_first(const unsigned int *array, unsigned int what);

The function searches for the given number in the array until it is found or the current number in the array is zero. If the number is not found until a zero is seen the program returns -1. If the number is found function returns the index at which the number was found.

Write a program that defines an array of 20 integers and fills the first 19 elements with random numbers in range 1 - 20 and puts a zero as the last number in the array. The program prints the array one number per line.

Program then asks user to enter number to search for or zero to stop. If the number is not zero program calls find_first to see if the number occurs in the array. If the number is found it prints the index where the number is found or “not found”. If user enters zero the program stops.