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. If user does not select quit the program prints the menu again.

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 two 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. Note that we don't pass the size of the array as a parameter but we use zero as an end marker to stop at the end of the array.

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 otherwise program asks again user to enter a number to find.

Ex10

Write a program that asks user to enter a string. Program must use fgets to read user input and remove the linefeed at the end of the string. Then program prints the length of the string and checks if the string is "stop". If it is the program stops else program asks user to enter a new string (and prints the length etc.).

Ex11

Write a function called **replace_char** that takes two strings as parameters and returns an integer. The first string is the string to modify and the second is a string containing two characters. The function finds every occurrence of the first character of the second string and replaces them with the second character. For example:

```
char text[80] = "I am so tired of Python. C is much better language";
count = replace_char(test, "e3");
```

The call above will find every occurrence of 'e' and replace it with '3'.

Prototype of the function:

```
int replace_char(char *str, const char *repl);
```

The function returns the number of characters replaced. The return value can be zero if no characters were found or if the second string does not contain two characters.

Write a program that asks user to enter both strings and then calls replace_char. The program prints both return value and the modified string if the return value is greater than zero. If the return value is zero program prints "String was not modified".

Ex12

Write a function count_words that takes two strings as a parameter and returns an integer. The function counts the number of times the second string occurs in the first string and returns the count. Use strstr() from standard library to find the strings. Note that if the string is found you need to advance past the current occurrence before searching again. Hint: use a pointer to the string and use pointer arithmetic to advance it past the current match.

Prototype of the function:

int count_words(const char* str, const char *word);

Write a program that asks user to enter a string and a word. Then program calls count_words and prints the return value. If the word was "stop" the program stops otherwise it asks user to enter a string and a word again.

Ex13

Write a program that asks user to enter a filename. Then program tries to open the file in textmode for reading. If opening the file fails the program prints an error message with the filename to **stderr** and then exits. If the file is opened the program starts reading integers from the file until reading fails or the file ends. When reading stops the program prints the count of numbers and the lowest and highest number that was read from the file. Then program closes the file.

For testing make a few text files with integers using your favourite text editor.

Ex14

Write a program that reads lines of text in to an array of strings. The maximum length of the line is 80 characters and the maximum number of lines is 100.

The program asks user to enter a filename. Then program tries to open the file in textmode for reading. If opening the file fails the program prints an error message with the filename to **stderr** and then exits. The program reads lines from the file into the array until the file ends or the array is full (100 lines read). Then program closes the file.

Then program converts all letters to upper case (see toupper() – function).

Then program opens the file for writing in text mode and writes all read lines to file and closes the file.

Ex15

Write a program that reads data from a text file into an array of structures. The size of the array is 40 elements.

The structure is defined as:

```
typedef struct menu_item_ {
    char name[50];
    double price;
} menu_item;
```

The program asks user to enter a filename. Then program tries to open the file in textmode for reading. If opening the file fails the program prints an error message with the filename to **stderr** and then exits.

Program reads data from the text file into the array until end of file or the array is full. Each row of the file contains data of one structure. The members are separated by semicolons. For example:

```
Mega double burger with bacon and cheese; 23.50
Belgian style fries with vegan mayo; 5.60
```

When reading ends the program closes the file and prints the content of the array in formatted columns. Price is printed into a field of eight characters with two decimals precision followed by the name of the item. Unused elements are not printed.

Ex16

Write a program that asks user to enter numbers or to enter **end** to stop entering numbers. If user enters any other text except end the program prints an error message and discards the input. The program must store the numbers in a linked list where each node is dynamically allocated.

```
typedef struct node {
    int number;
    struct node *next;
} nnode;
```

When user enters **end** the program prints all entered numbers, frees the allocated memory and then ends.

Ex17

Write a password generator **function** that takes three parameters: character pointer, integer (size of the array), a const char pointer (a word) and returns a bool. The function generates password and stores it in the array and returns true. The length of the generated password is the $length\ of\ word\ \times 2 + 1$ characters. If the password does not fit in the string function returns false and does not modify the string.

The password must contain the word given as a parameter so that password starts with a random printable character and every other letter is a letter from the word and every other is a random printable character. The password ends with a random printable character. Note that printable characters are not limited to alphanumerical characters.

Write a program that asks user to enter a word to place in the password or to enter "stop" to stop the program. The size of the string to read the word into must be 32. If user does not enter stop the program generates a password using the generator function. If a password is generated successfully the program prints it otherwise an error message is printed. Then the program asks for another word.

For example:

User enters: metropolia

Program prints: #m%eGtqrHo&p2o+lBimaY

You don't need to use colours. They are just a visual aid. The random characters will naturally be different on your program.

Ex18

Write a program that asks user to enter a number in the range from 0 to 15 or a negative number to stop. If user enters a number between 0 and 15 the program generates a random number and prints it in hexadecimal. Then program shifts the number to the right by the amount user entered and uses bitwise and to clear all other bits except bits 0-5 and prints the result in hexadecimal with two digits and leading zeros.

Note: bit numbering starts from 0

Ex19

Extend program of Ex15 so that after reading the file content the program asks user to choose sorting order of the menu. User can choose to sort by price or by name.

Write comparison functions for sorting by name and by price and use qsort() from standard library to sort the array before printing.

Ex20

Implement a program that consists of three files: main.c, debug.c and debug.h.

Debug.c contains two functions:

- set_debug_level
- dprintf

Set_debug_level takes one integer parameter (debug level) and stores the value in a static variable that is accessible to all functions in the file but not globally.

```
void set_debug_level(int debug_level);
```

Dprintf works like printf but there is an extra integer parameter (debug level) that comes before the usual printf parameters:

```
int dprintf(int debug_level, const char *fmt, …);
```

If dprintf debug level is lower or equal to the stored debug level then function prints the output to stderr prefixed with [DBGx] , where x is the debug level given as parameter. Otherwise, function prints nothing and returns zero.

Write a program that asks user to enter debug level in range 0 – 4 and calls set_debug_level with the number. Then program prints five messages with a random debug level in range 0 – 4 using dprintf. Get a new random debug level for each message. The messages must have at least one numerical argument that is a running index that starts from one. First message has index 1,second message 2, etc

Ex21

Write a program that asks user to enter a file name. Program opens the file in text mode. The file is supposed to contain NMEA (GPS) data. Program looks for lines that start with '$' and have an asterisk '*' later on the line. If a line meeting the condition is found the program verifies the checksum of the line. If checksum is ok line is printed prefixed with **[OK]** otherwise line is prefixed with **[FAIL]**.

Checksum is calculated by XORing all characters after '$' and before '*'. The result is checked against the two-digit hexadecimal value that follows '*'. If they are equal line is ok.

For example:

```
$GPGGA,092750.000,5321.6802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76
$GPGSA,A,3,10,07,05,02,29,04,08,13,,NOT,OK,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,19,13,28,070,17,26,23,252,,04,14,186,14*79
$GPGSV,3,3,11,29,09,301,24,16,09,020,,36,,,*76
$GPRMC,092750.000,A,5321.6802,N,00630.3372,W,0.02,31.66,280511,,,A*43
$GPGGA,092751.000,5321.6802,N,00630.3371,W,1,8,1.03,61.7,M,55.3,M,,*75
$GPGSA,A,3,10,07,05,02,29,04,08,13,BAD,,,,1.72,1.03,1.38*0A
$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70
$GPGSV,3,2,11,02,39,223,16,13,28,070,17,26,23,252,,04,14,186,15*77
$GPGSV,3,3,11,29,09,301,24,16,09,020,BROKEN,36,,,*76
$GPRMC,092751.000,A,5321.6802,N,00630.3371,W,0.06,31.66,280511,,,A*45
```

Above the lines that read "NOT,OK", "BAD" and "BROKEN" have a failing checksum.