

OBJECT ORIENTED PROGRAMMING

LABORATORY 1

OBJECTIVES

The main objective of this laboratory is to get the students acquainted with the basic elements of C programs: the structure of a program, input/output operations, data types, and conditional and repetitive statements. In this laboratory, it is recommended that you use C functions to read/display data to the user.

THEORETICAL NOTIONS

In the first laboratories, we'll be using the C programming language.

Printing values

To print formatted data to *stdout* (the screen), you should use the *printf* function. The *printf* function is included in the header file `<stdio.h>`.

The first argument of the *printf()* function is a format string that specifies the format of the output. The format string may contain plain text characters and *format specifiers*, which are special character sequences that start with a % character and specify the type and format of the output value.

A format specifier follows this prototype: [see compatibility note below]

%[flags][width][.precision][length]specifier

Here is an example of displaying an decimal integer:

```
printf("The value of x is %d\n", x);
```

In the example above, %d is a format specifier that indicates that the value of x should be printed as a decimal integer. The \n character sequence at the end of the format string is an escape sequence that represents a newline character, causing the output to be printed on a new line.

The printf() function can also be used with multiple format specifiers to print multiple values in a single call:

```
printf("The value of the decimal number x is %d and the value of the real  
number y is %f\n", x, y);
```

In this case, the format string contains two format specifiers: %d and %f. The first specifier is replaced by the value of x as a decimal integer, and the second specifier is replaced by the value of y as a floating-point number.

The documentation of the printf function can be found here:

<https://cplusplus.com/reference/cstdio/printf/>

Reading data

scanf() is a commonly used function in the C programming language that is used to read input from the standard input stream (usually the keyboard). Similar to *printf* it is part of the standard input/output library (<stdio.h>).

The first argument to the *scanf()* function is a format string that specifies the format of the input.

Here is an example of how to read an integer with *scanf*:

```
int x;
scanf("%d", &x);
```

In this example, %d is a format specifier that indicates that a decimal integer should be read from the input. & is the *address of* operator, and it is used to pass the memory address of the variable x to scanf(), so that the *scanf* function can modify this value (we'll cover memory addresses in details in the next labs).

To read a string of characters with *scanf* you don't need to use the address of operator, because in C the name of an array points to the first element of the array (again, will be covered in a future lab). So to read a string of characters you should use:

```
char str[80];
printf ("Enter your family name: ");
scanf ("%79s", str);
```

In the example above, %s means that you will be reading a string; keep in mind that *scanf* stops at the first whitespace character found. **So you cannot read strings with spaces using scanf.** %79s means that you will be reading at most 79 characters. In C strings, always the last character is '\0' or the null character that marks the end of the string.

To read strings that contain spaces, you can use:

```
char str[100];
printf("Type something: \n");
fgets(str, 100, stdin);
printf("You typed: %s\n", str);
```

In the example above *fgets* reads characters from *stdin* (standard input) and stores them as a C string into *str* until (100 - 1) characters have been read or a newline has been read. So you can use *fgets* to read strings with spaces.

The return value of *scanf()* is the number of input elements that were successfully matched and assigned. If an error occurs during input, a negative value is returned. So you should check the output of *scanf* when solving the assignments.

The documentation of the *scanf* function can be found here:

<https://cplusplus.com/reference/cstdio/scanf/>

PROPOSED PROBLEMS

1. Write a simple *Hello world* program that prompts the user for his/her name and birth year, and you display a greeting message as well as the age of the person.
2. Alice forgot her card's PIN code. She remembers that her PIN code had 4 digits, all the digits were distinct and in decreasing order, and that the sum of these digits was 24. Write a C program that prints all the PIN codes which fulfill these constraints.
You should print the pin codes in **decreasing** order.
Think of all the possibilities you can improve the performance of your program!

3. Given a natural number *n* write a C program that computes the number of 1s in its binary representation.
For example, if *N* = 10, your program should display 2 (because 10 in binary is 1010). If *N* = 32 your program should display 1 (because 32 in binary is 100000).
4. We all know that Easter falls on a Sunday in spring, but which one? Formally:

"Easter is celebrated on the first Sunday following the full Moon that occurs on or just after the spring equinox".

Sounds pretty complicated, but is it? According to an algorithm published in *Nature* journal in 1876¹, the algorithm for computing the catholic Easter date is the following:

A = year *mod* 19

B = year *mod* 4

C = year *mod* 7

D = (19**A* + 24) *mod* 30

E = (2**B* + 4**C* + 6**D* + 5) *mod* 7 where *mod* is the **remainder** of the division of *x* to *y*.

Easter day is then (22 + *E*+*D*) March. Note that this formula can give a date from April if 22 + *E* + *D* > 31; also take this case into account.

Write a program which reads a year (strictly positive integer) and displays the Easter date for that year. Create an appropriate data structure to store a date.

¹ <https://en.wikipedia.org/wiki/Computus>

If the year the user entered is not valid, you should display exactly this message:

Invalid input, the year should be greater or equal to 1876

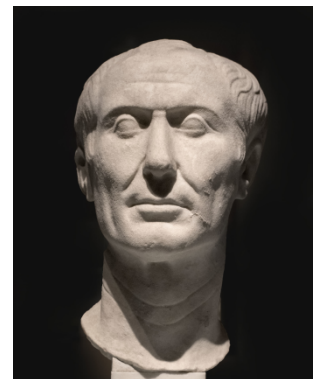
otherwise, display the message:

In year [year] the Easter date is [March|April] [day]

; the day should be displayed on two digits with leading zeros.

| Input | Expected output |
|-------|--|
| 10 | Invalid input, the year should be greater or equal to 1876 |
| a | Invalid input, the year should be greater or equal to 1876 |
| 1875 | Invalid input, the year should be greater or equal to 1876 |
| 1934 | In 1934 the Easter date is April 01 |
| 1935 | In 1935 the Easter date is April 21 |
| 1991 | In 1991 the Easter date is March 31 |
| 1989 | In 1989 the Easter date is March 26 |
| 2070 | In 2070 the Easter date is March 30 |
| 2068 | In 2068 the Easter date is April 22 |

5. It is known Julius Caesar used to send his private and important military messages using the following encoding: each letter from the original text is replaced by a letter situated at a fixed number of positions down the alphabet.



If he had anything confidential to say, he wrote it in cipher, that is, by so changing the order of the letters of the alphabet, that not a word could be made out. If anyone wishes to decipher these, and get at their meaning, he must substitute the fourth letter of the alphabet, namely D, for A, and so with the others. —
Suetonius, Life of Julius Caesar 56

Nowadays, this is one of the simplest methods of encryption and is called a substitution cipher or Caesar's cipher.

Write a program which reads a natural number n and a string s . The string s is encoded using Caesar's cipher with a displacement of n (either positive or negative) and can contain spaces (but no newlines).

Decode the message and display it on the screen. Punctuation marks and digits are left as they are.

| Input | Expected output |
|--|--|
| 7 clup, cpkp, cpjp! | <i>veni, vidi, vici!</i> |
| 10 aey noybew ycdoxdk od sxswsmybew sxsaesdkc eymkd. Skmdk kvok ocd! | <i>quo deorum ostenta et inimicorum iniquitas uocat. Iacta alea est!</i> |

What does the program print if: $n = 5$ and the encoded string is: N qtaj tgojhy twnjisyji uwtlwfrnsl! ?

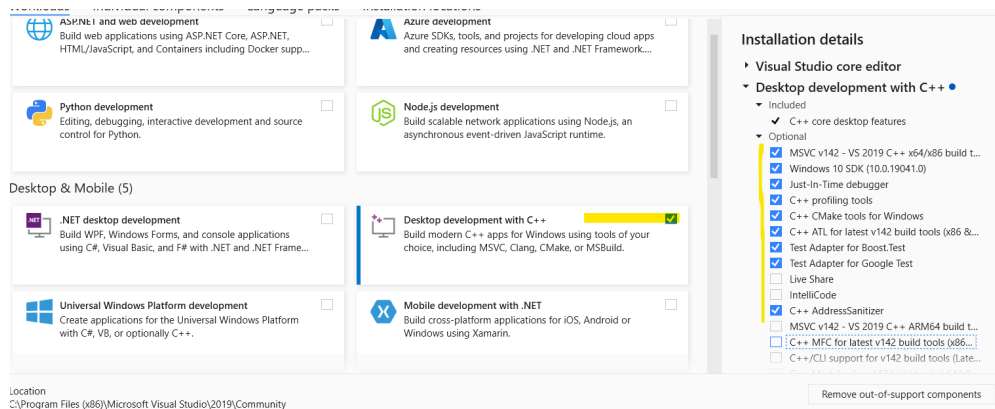
If you don't have any IDE (integrated development environment) installed on your machine, you can work on an online environment (https://www.onlinegdb.com/online_c_compiler) while your IDE is installing.

ENVIRONMENT SETUP

Integrated Development Environment

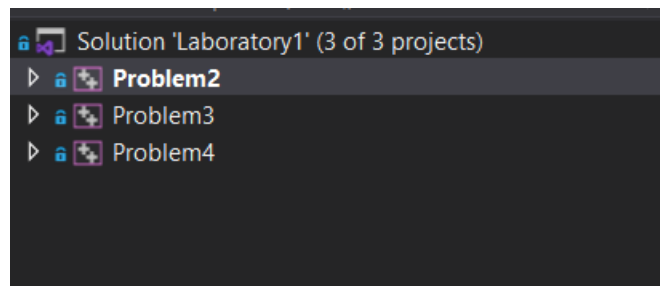
We will be using Visual Studio as IDE in the first part of the semester. Download the Visual Studio Community edition from here: <https://visualstudio.microsoft.com/vs/community/>

When installing, select just ***Desktop Development with C++*** (highlighted in yellow):



In Visual Studio, a **solution** is a container that can hold multiple projects, while a **project** is a single unit of work that can be compiled into an executable or library.

For example, in the first lab you have the solution *Laboratory 1*, with 3 projects: Problem2, Problem3 and Problem4.



To switch between the active projects, right click on the project name and select *Set as startup project*.

To compile and run a project just press F5.

Version control (git)

For most of the assignments you will be given a starter code on github classroom.

1. Installation

First create an account on github classroom: <https://github.com/> and download git on your computer. For Windows users: <https://git-scm.com/downloads>

2. Setup git

You will be using the GitBash terminal on Windows or the terminal on other operating systems. You first need to configure Git with your name and email address before you start using it. Use the following commands to do this (but change you name and your email address):

```
git config --global user.name "Your Name"
git config --global user.email "youremail@example.com"
```

3. Getting the assignment URL

You will be given a link to the github assignment. Follow that link and accept the assignment:


Wait a few seconds and then refresh the page:

oop-mie-2023


Accept the assignment — OOPLaboratory1

Once you accept this assignment, you will be granted access to the `ooplaboratory1-diana1auraborza` repository in the `ubbfmi` organization on GitHub.


Accept this assignment



You accepted the assignment, **OOPLaboratory1**. We're configuring your repository now. This may take a few minutes to complete. Refresh this page to see updates.

 Your assignment is due by **Mar 10, 2023, 10:00 EEST**

Note: You may receive an email invitation to join `ubbfmi` on your behalf. No further action is necessary.



Join the GitHub Student Developer Pack

Verified students receive free GitHub Pro plus thousands of dollars worth of the best real-world tools and training from GitHub Education partners — for free. [Learn more](#)

Apply

Then, you will have a url with your repository (highlighted in light blue):




You're ready to go!

You accepted the assignment, **OOPLaboratory1**.

Your assignment repository has been created:

 <https://github.com/ubbfmi/ooplaboratory1-dianalauraborza>

We've configured the repository associated with this assignment ([update](#)).

 Your assignment is due by **Mar 10, 2023, 10:00 EEST**

Note: You may receive an email invitation to join [ubbfmi](#) on your behalf. No further action is necessary.

4. Cloning the repository

git clone is a Git command that creates a copy of an existing Git repository, including all of its files, folders, and version history, onto your machine.

When you use the **git clone** command, you provide it with the URL of the existing Git repository (the one highlighted in blue above), and Git will automatically download and copy all of the files and folders from that repository onto your local machine.

So, from the terminal or *GitBash*, change the working directory to the directory where you want to clone the repo:

```
cd D:/school/OOP
git clone url
```

`cd` (change directory) is a command used to change the working directory.

Replace `D:/school/OOP` with a path from your computer, and `url` with the url generated at step 3.

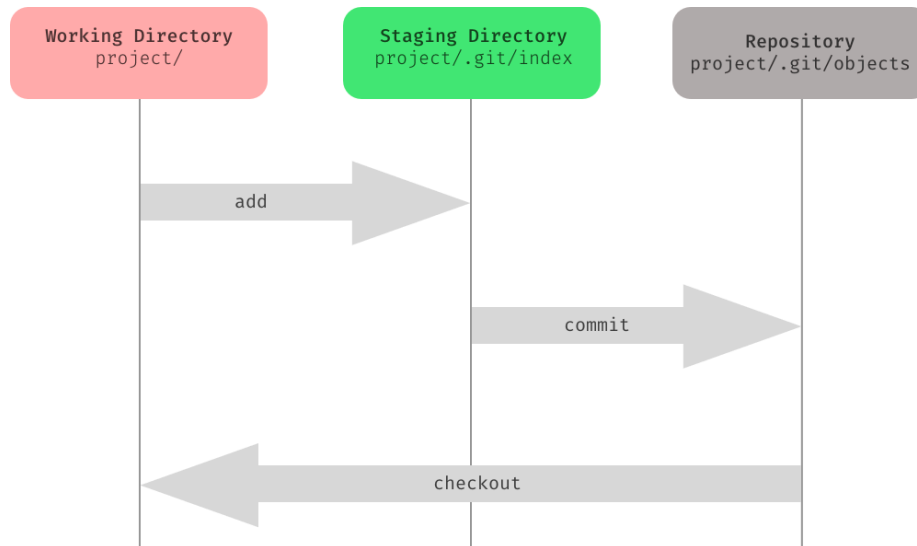
Now you should have the code on your machine. Right-click on the `.sln` file inside the newly created directory to open the Visual Studio solution.

Now you can start working on the assignment. Each time you have something working, you can “save” it to git.

5. Saving changes to git

git add and *git commit* are two fundamental Git commands used to save changes to a Git repository.

git add is used to stage changes for commit. When you make changes to files in a Git repository, those changes are initially "unstaged" and must be explicitly added to the staging area in order to be included in the next commit. The staging area is a buffer between your working directory and your Git repository, where you can review and modify your changes before committing them.



To stage changes using *git add*, you simply specify the files you want to stage (of course, replace filename with the path of the file you want to add):

```
git add filename
```

or, if you want to add all the files tracked by git:

```
git add -u
```

Once you've staged your changes using *git add*, you're ready to create a new commit using *git commit*.

git commit is used to save changes to the Git repository. When you run *git commit*, Git creates a new commit containing the changes you've staged using *git add*, along with a message describing the changes you've made.

To create a new commit using *git commit*, you simply run the command:

```
git commit -m "commit message"
```

, where "commit message" is a brief description of the changes you've made.

6. Pushing the changes to the remote repository

Now you can push changes to the remote git repository using the following command:

```
git push
```