Operating Systems

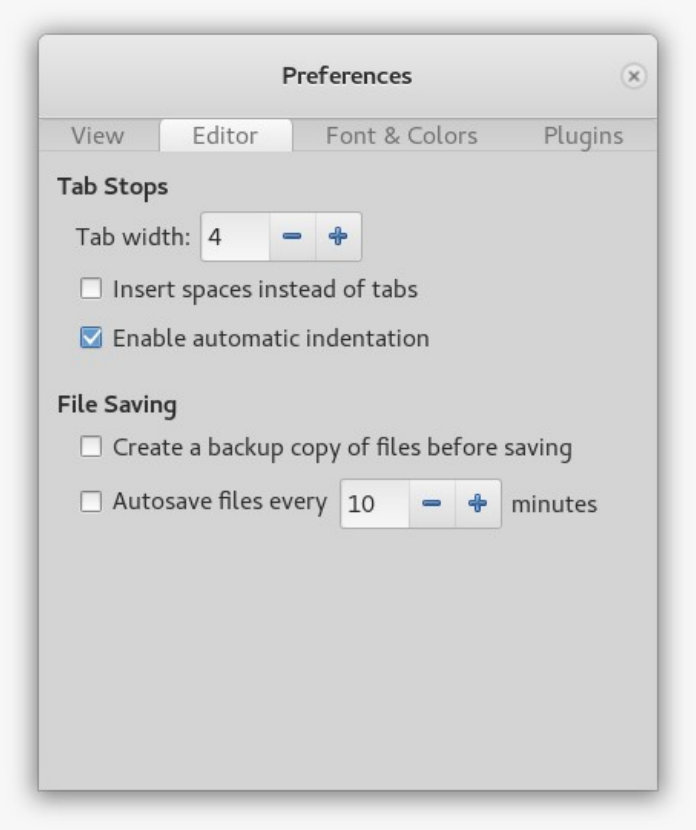
HOCHSCHULE DER MEDIEN

Exercise A: C introduction

1. Configuring your IDE

First of all we need to configure the text editor (gedit) in order to work properly.

- Go to the Edit > Preferences > Editor tab
  - set the "Tab width" to 4
  - uncheck the option "Insert spaces instead of tabs"
  - uncheck the option "Create a backup copy of files before saving"



Operating Systems

2. Hello World

With every language you learn, the very first thing to code is always the “hello world” message. Of course this is also the case in this lecture.

- Type or copy & paste the “hello world” source code from the lecture into the text editor.
- Save it in a file called “hello.c” in a folder “first_steps” on your desktop. It will contain the source code for our little program.
- Then open a terminal and go to the directory using the following commands:

```
cd Desktop
cd first_steps
```

- Compile the program with the following command. Simply copy it to the command line.

```
gcc -std=c99 -Wall -o hello hello.c
```

After the execution of the command has finished successfully you should find a file “hello” in your current folder. This is your binary executable.

- Execute the program using the following command:

```
./hello
```

Now you should see the message printed to your command line.

Congratulations. You have mastered the first step to programming C :)

Operating Systems

3. Rolling dice

In order to learn the very basics of C we are going to write a very simple game. In this game three dice are rolled. If all of them show a winning number (in our case 6) we won the game. Because this can take a while we let the computer do the work of rolling. We just wait and see how long it took.

- First we start off by creating a file “dice.c”. Add a main-skeleton and print a test message. Use the commands from the first exercise to compile and run the program.

The first thing our program has to do is start of the random number generator. Therefore we initialize the generator with a seed. The random number generator is later needed to simulate the dice.

- Add the following line at the beginning of your main-function.

```
srand( (unsigned) time( NULL ) );
```

Now we can start rolling our dice.

- Implement a mechanism to simulate rounds of the games.
- In each round, first roll the dice, then print a message to the console displaying the numbers shown by the dice. Use the following function to produce random numbers in a certain interval:

```
int getRandomNumber(int minRange, int maxRange) {
    // return random number between minRange and maxRange
    return rand() % maxRange + minRange;
}
```

- Check if the dice are a winning combination.
- To see how many rounds it took to get the winning combination, build in a counter. Display it when the game is won.

Hints

Includes

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

4. Guessing Game (Yet another game about math ;))

This game picks a random number between 1 and 50 which the player then has to guess. He doesn't have to do it all by his own, the game will help him a little. Every time the player guesses a number, the game will tell him whether the number was smaller or bigger than the number the game picked.

- To tell the player what to to, first print a tutorial on the screen. It should contain all the necessary information that are needed to play the game.

For starters the player has 10 attempts to guess the number. This value should be variable so we can change it later on.

- Use the function from the second exercise to generate a random number in the given range. Don't forget to initialize the random number generator.
- Implement a game loop as well as the logic to end the game, if all attempts have been used.

Every time the player is allowed to guess, he should be able to enter a number and then see if he was correct or not. If the player won, a nice message should be printed. And of course, if he looses, as well.

- Ask the player for input on the command line. He should type in a number.
- Add a condition to the game loop that checks whether the player has guessed correctly. It should also include a helping statement, if the player was wrong. The message should tell if he guessed to high or too low.
- Display a winning/loosing message on the screen.

Hints

Includes

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