Tartalomjegyzék

Compiling C

You are free to use any IDE/compiler. If someone is looking for suggestions, I can suggest these:

- gcc
  - This is for those who want to be as minimalistic as possible. You can write your code in any text editor and then compile them in the command line.
  - Automatically installed in most Linux distributions.
  - The "http://wiki.math.bme.hucc"http://wiki.math.bme.hu command on Mac works similarly.
  - Windows-on kicsit bonyolultabb vagy [cygwin https://www.cygwin.com/]-t, vagy a [w64devkit https://github.com/skeeto/w64devkit]-et ajánlom
  - It's a bit more work to get it to work on Windows. I suggest [cygwin https://www.cygwin.com/] or [w64devkit https://github.com/skeeto/w64devkit].
- codeblocks
  - Simple opensource IDE
  - Supports Windows, Linux, Mac
  - If you don't have a C compiler already on your Windows then you should install the **codeblocks-#.#mingw-setup.exe** option. This is an IDE and a compiler in one.
- online c compiler
  - Only in case you need a compiler urgently and you don't have the time to use any of the above.

Once you have your compiler/IDE try to compile this small code:

```
#include <stdio.h>
int main() {
    int x = 2;
    printf("hello world! %d\n", x);
    return 0;
}
```

- In IDEs I recommend starting an "http://wiki.math.bme.huEmpty C project"http://wiki.math.bme.hu.
- If you're walking the way of the command like, then create a new file with the **hello.c** name, then in your command line you can use: **gcc hello.c -o hello**
- In the command line you'll need to issue the **hello** or the **hello.exe** command to execute your new program. In IDEs there's usually a Run command.

- Those who won't shy away from the command line should use the **gcc -W -Wall hello.c -o hello** command to compile from now on. This also gives some warnings that might be useful.

Deliberate errors

Let's make some errors in the "http://wiki.math.bme.huhello.c"http://wiki.math.bme.hu code. After each of these changes save your file, compile the program. Once you've seen the error message fix the code again and try the next one.

- Delete a semicolon from one of the lines
- on the line with the printf write a different letter instead of "http://wiki.math.bme.hux"http://wiki.math.bme.hu
- define a variable y, but don't use it anywhere
- delete the last "http://wiki.math.bme.hu}"http://wiki.math.bme.hu
- delete the "http://wiki.math.bme.hureturn 0;"http://wiki.math.bme.hu line

Exercises

Open a new project for each exercise or a new file if you're in the command line.

1. Conditional print

This is how an if, else if, else statement looks like:

```
if (/* condition */) {
    /* commands */
} else if (/* condition */) {
    /* commands */
} else {
    /* commands */
}
```

Complete the code based on the comments.

```
#include<stdio.h>

int main(void) {
    /* declare and give values to variables x and y */
```

```
    printf("value of X: %d\n", x);
    printf("value of Y: %d\n", y);

    printf("\n");

    /* using conditionals make it so only true statements are printed */
    printf("%d is greater than %d\n", x, y);
    printf("%d is greater than %d\n", y, x);
    printf("%d is equal to %d\n", x, x);

    return 0;
}
```

Once done modify your code so that *x* and *y* doesn't have any values in the beginning. Instead use *scanf()* to read the numbers from the user of the program. An example where the integer *z* is read:

```
int z;
    printf("Provide a value for z: ");
    scanf("%d", &z);
    /* From this line z has the value that the user input. (If it truly was an integer.) */
```

2. For cycle

Here's an example for a *for* cycle that prints the numbers from 0 to 9. (Usually we call the cycle's variable *i* or *j* but this is only a convention.)

```
int i;
for (i = 0; i < 10; i++) {
    printf("Value of the cycle variable: %d\n", i);
}
```

Complete the code below so that it prints the even numbers from 1 to 100.

```
#include<stdio.h>

int main(void) {
    /* variable definitions */

    for (/* inic. */ ; /* condition */; /* after each cycle */) {
        printf("%d\n", i);
    }
    return 0;
}
```

3. Do / while cycle

Write C code that reads integers from the user in a cycle until they input 0. Once that happens the program prints the average of the input numbers.

You can create floating point numbers (for the average) with the *float* type, and print them like so:

```
printf("%f", average);
```

Example for a **while** cycle:

```
while ( /* condition */ ) {
    /* commands that run in each cycle */
}
```

You can also us a *do-while* cycle. This guarantees that the block of the cycle executes at least once. (It doesn't check the condition for the first cycle.)

```
do {
    /* commands that run in each cycle */
} while ( /* condition */ );
```

4. Simple square root

Complete the following code that finds the square root of 5329 (its root is an integer).

(You don't need to write efficient code. It's okay if you check numbers one by one until you find the root.)

```
#include<stdio.h>

int main(void) {
    int num = 5329;
    int root;
    /* use a cycle, you will most likely need an if condition inside the cycle as well */

    printf("The root of %d is %d\n", num, root);
    return 0;
}
```

5. Chess table

Draw an NxN chess table where X represents the black square and the white squares are left empty (space character). No need to give a border to the table. The size of the table (*N*) should be read from the user.

Hint: You can use a cycle inside a cycle, but be sure to use separate cycle variables for them!

Example of a cycle inside a cycle:

```
#include<stdio.h>

int main(void) {
    int i;
    int j;

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

Extra tasks

If you made it all the way here or you simply felt like the above exercises were too boring you can implement the exercises from the first lecture. Or you can play with one of these more complex tasks.

6. Coprimes

Write a program that reads two positive integers (n, m) from the user. It then prints all coprimes of n from 0 to m.

7. Root

Write a program that reads a positive number and prints its root with an accuracy of 0.0001.

8. Twenty questions

Write a program that simulates the twenty questions game. The player thinks of an integer between 0 and 100. The program asks a question, the player responds with 0 (no) or 1 (yes). This goes on until the program knows exactly which number the player thought of, it then prints this number. (What is the fastest way to find the number?)