2. EXERCISE SHEET, RETURN DATE *MAY 7/8TH 2015*, INDIVIDUALLY

(CORRECTED APRIL 24TH)

General notes

Individually means, that this homework sheet needs to be handed in by everyone seperately. There will be a new doodle-link on the website.

If you want to use loops you need to add

```
-std=c99
```

to the compiler options.

I also suggest you to use the options

```
-Wall -pedantic
```

to show all warnings.

Resolve all warnings if possible.

Create test functions for everything you program.

The compiler option

```
-lm
```

links the math library corresponding to

```
#include <math.h>
```

Exercise 1: open a window with sdl

You will need to install the library SDL2 from www.libsdl.org into your home directory. Try doing this by the instructions from their website. Otherwise here comes the solution:

In your home folder download the source code of SDL2

```
wget https://www.libsdl.org/release/SDL2-2.0.3.tar.gz
```

and extract the archive with

```
tar -xvf SDL2-2.0.3.tar.gz
```

Then change into the directory SDL2-2.0.3 (cd SDL2-2.0.3).

Then we can use the standard way to install linux programs from source code. Except we have to specify the installation directory manually by adding the option –prefix to configure.

The following line will have to be different in the obvious way if you don't use the pm2-accounts. If you use the pm2-account, then replace the stars with the number of your account.

```
./configure --prefix=/homes/pm2/pm2-***/SDL2
```

will check the properties of your computer and decide how to compile.

```
make
```

will compile the library and

```
make install
```

will copy everything neccessary to the folder SDL2 that we have specified using the –prefix option. The SDL2 folder contains a directory called include containing all C header files containing the functions that are provided by the library. The directory lib contains the compiled library.

We can now remove the archive file and the SDL2-2.0.3 folder containing the source files.

To compile programs using the SDL-library we need to add a couple of compiler options. Namely

```
-I/homes/pm2/pm2-***/SDL2/include
```

to specify where to find the header files of our library,

```
-L/homes/pm2/pm2-***/SDL2/lib
```

to tell the compiler (more precisely the linker) about the library,

```
-Wl,-rpath,/homes/pm2/pm2-***/SDL2/lib
```

to tell where the compiled program needs to look for the library and

```
-lSDL2
```

to tell the linker which library we want to use. The complete compilation command looks like this (all in one line)

```
gcc -Wall -pedantic -std=c99 -I/homes/pm2/pm2-***/SDL2/include
-L/homes/pm2/pm2-***/SDL2/lib -Wl,-rpath,/homes/pm2/pm2-***/SDL2/lib
-o executableName sourceFileName.c -lSDL2 -lm
```

Note that now the order of the arguments may not be arbitrarily changed for reasons I do not know.

The following source will open a Window, wait 2 seconds and then close the window.

```
#include <stdio.h>
#include <stdlib.h>
#include <SDL2/SDL.h>
int main(int argc, char ** argv){
        //initializes the SDL and checks for success
        if (SDL_Init(SDL_INIT_VIDEO) != 0){
                return 1;
        }
        SDL_Window *win
        = SDL_CreateWindow("Hello World!", 100, 100, 800, 600, SDL_WINDOW_SHOWN);
        SDL_Delay(2000);
        SDL_DestroyWindow(win);
        SDL_Quit(); //cleans up the pointers of SDL
        return 0;
}
```

Extra: make the window stay open until closed by user.

EXERCISE 2: PAINT EVERYTHING RED USING OPENGL

OpenGL is already installed on most computers. Even on your smart phone. So we only need to add

–lGL

to the compiler options and include

```
#include <SDL2/SDL_opengl.h>
```

Furthermore you will need to change

```
SDL_WINDOW_SHOWN
```

to

```
SDL_WINDOW_OPENGL
```

in your code.

The following SDL call creates an openGL context in our window. This means that everything we do with openGL will be drawn into this window.

```
SDL_GLContext context = SDL_GL_CreateContext(win);
```

OpenGL will first draw onto some graphics memory (the *framebuffer*) and not the screen directly. This has the advantage, that we don't see the drawing process, but only the final image. This area of memory needs to have the same number of pixels as our window. To specify these dimensions call:

```
glViewport( 0, 0, ( GLsizei )800, ( GLsizei )600 );
```

Next we set the color for clearing the image (here for example red)

```
glClearColor( 1.0f, 0.0f, 0.0f, 1.0f );
```

The following function actually applies the clearing of the image

```
glClear( GL_COLOR_BUFFER_BIT );
```

Then we can use openGL commands to draw onto this image. But we will do this in exercise 3.

With

```
SDL_GL_SwapWindow(win);
```

we swap the window content with the framebuffer and the picture we have drawn becomes visible in the window.

With

```
SDL_GL_DeleteContext(context);
```

all pointers created by SDL_GL_CreateContext are freed.

Exercise 3: Draw a triangle

openGL is a machine that takes 3-dimensional vertices as input and outputs an image. These vertices are usually the corners of triangles that are then drawn by the GL. Before giving the vertices, we need to tell the machine what to do with these. openGL stores two matrices that the vertices are multiplied by when they enter the machine. The functions

```
glMatrixMode( GL_PROJECTION );
glLoadIdentity( );
glMatrixMode( GL_MODELVIEW );
glLoadIdentity( );
```

set these two matrices to the identity.

After the vertices have been transformed by these two matrices, the x- and y-coordinates determine the position of the vertex on the final image. The coordinate (1,1,z) corresponds to the upper right corner of the window and (-1,-1,z) to the lower left. If we have specified to draw triangles for example, then the triangle spanned by 3 consecutive vertices will be drawn by the GL.

We can tell the machine the color in which it should draw by calling

```
glColor4f(1.0f,1.0f,1.0f,1.0f);
```

supplying a color in red-green-blue-alpha format. Alpha we don't need for now.

There's one more thing. The z-coordinate of the transformed vertices is used to determine, if what needs to be drawn is in front of what has already be drawn or can be discarded. We won't need this functionality for now and turn it off by

```
glDisable( GL_DEPTH_TEST );
```

Thus everything will be drawn, independent of its z-coordinate.

Now we can specify the vertices in the following fashion

```
glBegin(GL_TRIANGLES);
        glVertex3f(x,y,z);
        ...
        glVertex3f(x,y,z);
glEnd();
```

Now write a program that draws a green triangle with the following three corners: middle of the window, middle of upper border, middle of right border. Draw a blue triangle that is a bit smaller onto the first.

Exercise 4: Function pointers

In C we can pass a function as an argument like in matlab. In fact we will pass a pointer to the memory location of the function code. The syntax is show below.

Write a function that plots a function

```
void plot(double (*func)(double), double xmin, double xmax, double ymin, double ymax){
        ...
        double y = func(x);
        ...
}
```

For doing this you should use openGLs functionality to draw lines. Pairs of vertices are connected by a line if you call the following

```
glBegin(GL_LINES);
        glVertex3f(x,y,0);
        ...
        glVertex3f(x,y,0);
glEnd();
```

Plot the sin-function from the math-library by calling:

```
plot(sin, 0.0, 4.0, -1.0, 1.0);
```

Exercise 5: Plot numerical solutions

Implement the following algorithms: Euler, Modified Euler, Heun

Use the syntax

```
void euler(double (*f)(double, double), double x0, double y0, double h, double* x, doub
int numSteps)
```

where h is the step size, and the pointers x and y are arrays of size $(numSteps + 1)$ for storing the solution.

For plotting you will need a function

```
void plotArray(double* x, double* y, int size, double xmin, double xmax, double ymin,
double ymax)
```

Now solve the following ODEs with all your three methods:

- $y'(x) = cos(x)$, $y(0) = 0$
- $y'(x) = x$, $y(0) = 0$
- $y'(x) = y(x)$, $y(0) = 1$
- $y'(x) = -y(x)$, $y(0) = 1$
- $y'(x) = -x \cdot y(x)$, $y(0) = 1$
- $y'(x) = \frac{1}{1-x}$, $y(0) = 0$

For plotting use a white background. Draw the analytical solution in black, the Euler solution in red, the modified Euler solution in blue and the Heun solution in green. Choose the size of the domain reasonably. Take screenshots which show the difference in behaviour of the three methods for all these ODE's. Choose one ODE for comparing the convergence behaviour for different step sizes. Assemble the screenshots into a pdf-file. Mention the scale on the axes in the pdf-document. If you like, use SDL methods to draw the diagram axes.

Are the numerical solutions consistent with the analytical solutions of the ODE's? How large can you choose the step size to obtain a reasonable approximation?

Exercise 6: geometrical interpretation

Draw pictures illustrating the algorithms you used. For example for the Euler method you would draw a tangent line to your function and the point on that line for time $t + h$. (only one step, not the whole numerical solution)