Department of Mathematics
Dr. Ulf Lorenz
Christian Brandenburg

TECHNISCHE
UNIVERSITÄT
DARMSTADT

winter term 2008/2009
09/02/2009

Introduction to Mathematical Software

8th Exercise Sheet

Course Attendence Certificate

The Rules

This is the assignemet you have to solve in order to obtain your course attendance certificate, i.e. for receiving the credits for this course. Please read the following instructions carefully so that you fully understand what is asked of you!

- The assignement is due on Monday, February 9, at 8:00am in the exercise session.
- You are allowed to work in groups of up to three students.
- There will be only the marks "passed" and "not passed".
- In order to pass, it is necessary to solve both exercises!
- In the exercise session on Monday, February 9, the results will be discussed group by group. Only those members of a group can "pass" who actively participate in this short discussion. Every group member should be able to explain any part of the exercise.
- visit the course homepage regularly in case there are corrections or additional hints for the exercises.
- In exercise 1, the exam procedure is as follows: for each part, we will give you a random value of ϵ, for which your program has to return the correct result.

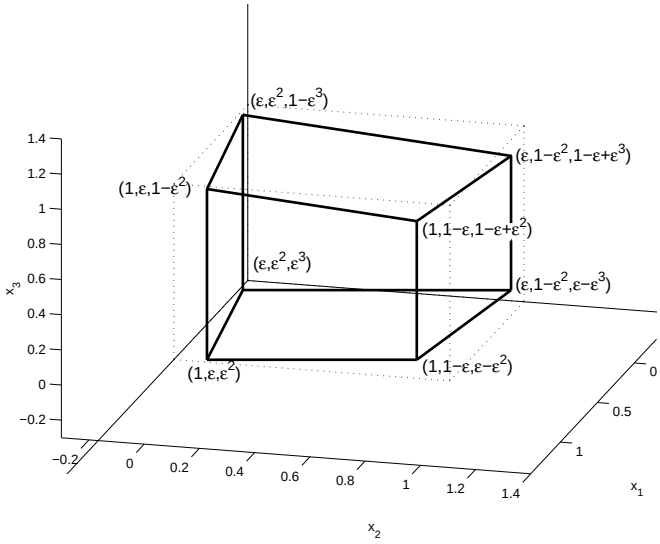

Figure 1: Klee-Minty-Cube

Exercise 1: Maple

The Klee-Minty-Cube

In this exercise, we consider the Klee-Minty-Cube, see Figure 1, which results from the unit cube by displacing the vertices in such a way that the faces are still planar (i.e. the four vertices defining a face are still on a plane). The faces, of course, are no squares anymore but arbitrary quadrangles in the general case.

To create a Klee-Minty-Cube, the vertices of the unit cube are displaced in the following way:

$$\begin{array}{cc} (0,0,0) & (\epsilon, \epsilon^2, \epsilon^3) \\ (1,0,0) & (1, \epsilon, \epsilon^2) \\ (1,1,0) & (1, 1-\epsilon, \epsilon-\epsilon^2) \\ (0,1,0) \mapsto & (\epsilon, 1-\epsilon^2, \epsilon-\epsilon^3) \\ (0,0,1) & (\epsilon, \epsilon^2, 1-\epsilon^3) \\ (1,0,1) & (1, \epsilon, 1-\epsilon^2) \\ (1,1,1) & (1, 1-\epsilon, 1-\epsilon+\epsilon^2) \\ (0,1,1) & (\epsilon, 1-\epsilon^2, 1-\epsilon+\epsilon^3) \end{array}$$

Here, $0 \leq \epsilon < 0.5$, where for $\epsilon = 0$ we regain the unit cube.

We recommend to use the `geom3d` package for this exercise. If you prefer, you can also use the `LinearAlgebra` package, however, this will make the exercise more challenging.

1. Write a Maple routine that, for a given value of ϵ, draws the corresponding Klee-Minty-Cube. *Hint:* use the `segment` command.

2. Use a Maple procedure to check for a given ϵ that the Klee-Minty-Cube really is a cube, i.e. all faces are planar.

   To this end, first write a procedure that returns for 4 points as input arguments the distance of the fourth point from the plane spanned by the first three points. Your procedure should have the following form:

```
DistanceFromPlane := proc(p1, p2, p3, p4)

  return dist;
end proc
```

   Use this procedure to check if the faces of the Klee-Minty-Cube are indeed planar.

3. Write a Maple procedure that computes the volume of the Klee-Minty-Cube for $0 \leq \epsilon < 0.5$.
   Hint: Subdivide the Klee-Minty-Cube into geometrical objects for which it is easy to calculate the volume. You may reuse part 2. The volume of the Klee-Minty-Cube with $\epsilon = 0.2$ is 0.4864 (depending on your implementation, your result may vary up to $\pm 1 * 10^{-6}$).

4. We now consider truncated Klee-Minty-Cubes. The truncated Klee-Minty-Cube is construncted as follows: Each edge of the Klee-Minty-Cube is divided into three segments of equal length, yielding two points on each edge which are $edgelength/3$ apart. For each vertex of the original Klee-Minty-Cube, the points on the edges which are closest to this vertex are connected. Then, the tetrahedron consisting of these four points is truncated, yielding a new triangular face. For $\epsilon = 0$, a sketch of the truncated Klee-Minty-Cube is shown in Figure 2.

   Write a Maple procedure to compute the volume of the truncated Klee-Minty-Cube.
   Hint: Starting from the volume of the Klee-Minty-Cube, it might be easiest to substract the volumes of the parts that have been cut off. The volume of the truncated Klee-Minty-Cube with $\epsilon = 0.2$ is 0.4623802469 (again, your result may slightly vary).

Exercise 2: C++

Sorting Algorithms

In the C++ lecture, you learned three sorting algorithms: BubbleSort, MergeSort and BucketSort. In this exercise, we will combine these algorithms with the graphical output class you got to know in exercise class 6.

1. To start with, obtain the files `window.cc`, `window.h`, `MergeAndBucketsort.cc`, and `Bubblesort.cc` from the course webpage and save them in a subdirectory.

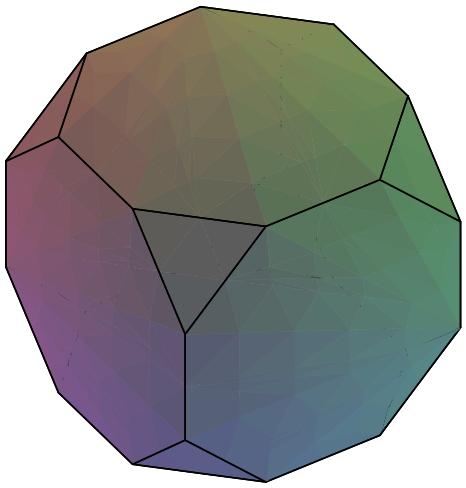

Figure 2: Truncated Klee-Minty-Cube

2. Compile the program `MergeAndBucketsort` by typing
   `g++ -g window.cc MergeAndBucketsort.cc -lX11 -o MergeAndBucketsort .`
   What this line does is the following: `g++` calls the GNU-Compiler Collection's C++ compiler, `-g` adds debugging symbols to the executable file, `window.cc MergeAndBucketsort.cc` are the source files to be compiled, `-lX11` tells `g++` to link the X-window libraries which are needed for the output window, and `-o MergeAndBucketsort` tells `g++` to name the executable `MergeAndBucketsort`

   Execute the resulting binary (type `./MergeAndBucketsort` on the command line).

   Try to understand what happens in the subfunctions
   `int OpticSort::mergesort(point *myarray,int l,int r)` and
   `int OpticSort::linearsort(point *myarray,int from,int to)`.

3. Your next task is to implement the same graphical output for the Bubblesort algorithm. To this end, open the file `Bubblesort.cc` with a text-editor of your choice. The file contains a lot of predefined functions, but you will not need to understand all of them in detail!

   Before changing anything, check that everything works as expected by compiling the program with
   `g++ -g window.cc Bubblesort.cc -lX11 -o Bubblesort` . Run the program by typing `./Bubblesort` . The program should open a new blank window. Pressing `enter` two times on the command line will stop the program.

   After you have checked that the program works as expected, you can start modifying it so that it actually does something useful.

The only function you have to edit is `int OpticSort::bubsort(point *myarray,int from,int to)`, which at the moment has an empty body. Write the Bubblesort algorithm you have learned in the lecture into this function. Similarly to the functions `int OpticSort::mergesort(point *myarray,int l,int r)` and `int OpticSort::linearsort(point *myarray,int from,int to)` in the file `MergeAndBucketsort.cc`, modify the Bubblesort algorithm such that the graphical window shows how the Bubblesort algorithm works.

Hints:

- If your program doesn't terminate after a longer time and seems to be stuck, you can stop it by pressing `Ctrl-c` (`Strg-c` on German keyboards) in the terminal window.
- For debugging your program, it might be useful to write a function that iterates over the array and prints the element stored in each field. This way, you can check whether the array is sorted correctly.