

Data Structures and Algorithms

Exercise 1

Markus Innerebner, Romans Kasperovics
dsa(a)inf.unibz.it

Free University of Bozen - Bolzano, Italy

Thursday, February 3rd, 2009

Exercises

Language / Time

- Exercises are taught every Thursday (except holidays)
 - 08:30-10:30 – in Italian by Markus Innerebner (room **E531**)
 - 08:30-10:30 – in English by Romans Kasperovics (room **E431**)
 - 14:00-16:00 – in German by Markus Innerebner (room **E531**)
- There are also office hours for each instructor when you can come with additional questions
 - R. Kasperovics's office hours are on Mondays, 14:00-15:00
 - M. Innerebner's office hours are on Mondays, 14:00-15:00
- **Please write an e-mail before you come**

Exercises

Rough Plan

- There are 12 exercises in total, 120 minutes each
- Ex.01 (this one)
 - introduction to exercises, Linux command shell and C language
 - you get an assignment (homework) + some hints
 - you are solving the assignment
- Ex.02-Ex.10
 - you are explained the solutions of the previous assignment
 - you get a new assignment + some hints
 - you are solving the assignment
- Ex.11
 - you are explained the solutions of the previous assignment
 - you solve different tasks preparing for exam
- Ex.12
 - you solve different tasks preparing for exam

Assignments

Solutions

- An assignment may require algorithms, program code, explanations, mathematical proofs or derivations as well as drawings
- The solutions are accepted in electronic form sent to **dsa[at]inf.unibz.it** or in paper form (only non-programming tasks)
- Code tasks must be sent as single compilable file (*.c)
- Non-code tasks can be packed in a single pdf document
- **Check twice that you attached all necessary files to your message**
- Your solution is expected **within 1 week till next Thursday's 08:30**
- **The solutions received after next Thursday's 08:30 are not accepted**

Program Code

- If an assignment requires a program written in C, make sure to compile and to test it.
 - compilation doesn't take a lot of time
 - it can show you the mistakes
 - if your program is not working correctly and it even do not compile - it is not acceptable
- **Normally, a single task do not require more than 50 lines of code!**
- **Comment your programs!**
- Use a C compiler (cc, gcc), do not use a C++ compiler

Copyright / Copyleft / Credit points

- Read the assignments carefully!
 - **The assignments are slightly different from previous years**
 - If you don't understand some of the tasks, ask questions during the exercises or office hours
- **You are supposed to work individually**
 - An exercise instructor might ask you to explain your solutions during the exercise and if you are unable to do that you get 0.
- **Try to do all assignments**
 - For mostly correct solved assignments you get 0.5 points, else 0. You can get in total at most 5 points. Each owned point is accumulated to the grade in the final exam.
E.g. grade exam: $16 + 2$ (solved 4 assignments) $\Rightarrow 18$.

Naming Conventions

- For each assignment create a folder *nn_yourlogin*
 - *nn* – is a number of the assignment
 - *yourlogin* – is your login to the university network
- Name your files according to the tasks in the assignment (see below)
- Put your name and student ID in all files

```
/ 01_minnerebner
  task12.pdf
  task3.c
  task4.c
```

Figure: Example directory structure of a solution

- Send the solution folder via e-mail archived as ***.zip** (standard windows) or ***.tar.gz** (standard linux) file. Do not use rar.

Environment

- The exercises are taught using C programming language
- GNU C compiler is used to compile the examples on Linux platforms
- Linux is installed on all lab computers
 - you have to select Linux at boot time
 - use your university account to log in
- We will use Linux command shell to write, compile and run programs
- To access Linux command shell use programs Konsole (in KDE), Gnome Terminal (in Gnome) or xterm
- You can also connect to a Linux server **russel.inf.unibz.it** from a Windows PC using ssh client (such as PuTTY). Note: X is not working

Linux Command Shell

- The command shell allows you do with Linux machine all the things you can do through visual interfaces and more
 - travel through directory structure
 - view / create / copy / move / delete files and directories
 - run programs
- Essential shell commands:
 - `pwd` – prints the current directory you are in
 - `cd` – changes the current directory to a given, for example:
 - `cd /var/log` changes the current directory to `/var/log`
 - `cd ~/dsa/01_minnerebner` changes the current directory to your home directory's subdirectory `dsa/01_minnerebner`
 - `ls` – lists the content of the current directory
 - `man` – gives the help on a given command, for example:
 - `man ls` gives all options for the command `ls`
 - `man sh` gives the general help for linux command shell

Text Editor vim

- There are many different editors you can use (`vi`, `vim`, `pico`, `nano`, `emacs`, `kate`, `gedit`)
- Further on we explain vim
- To start vim, type in the command shell:
 - `vim` – starts vim with a new empty file with no name
 - `vim myoldfile.c` – starts vim and opens existant file `myoldfile.c`
 - `vim mynewfile.c` – starts vim with a new file with name `mynewfile.c`
- Inside vim:
 - `i` – switches to INSERT mode
 - `esc` – exits INSERT mode
 - `:q!` – exits vim discarding any changes
 - `:w` – writes to a file (if a file already has a name)
 - `:w myfile.c` – writes to `myfile.c`

Linux Command Shell

Manipulation of Files and Directories

- `mkdir` – makes a new directory, for example
`mkdir ~/dsa`
- `rmdir` – removes an empty directory, for example
`rmdir ~/dsa`
- `rm` – removes a file, for example
`rm ~/dsa/01_minnerebner/task1.c`
- `cp` – copies a file, for example
`cp ~/task1.c ~/dsa/01_minnerebner`
- `mv` – renames or moves a file, for example
`mv ~/task01.c ~/dsa/01_minnerebner/task1.c`
- `tar -czf` – creates an archive from the given directory, for example
`tar -czf 01_minnerebner.tar.gz 01_minnerebner`
- `tar -xzf` – extracts files from the given archive, for example
`tar -xzf 01_minnerebner.tar.gz`

C Programming Language

Example 1: Using Input and Output

- Create a new file: `cd ~/dsa/01_<yourlogin> ; vim example1.c`
- type in a program (press 'i' to enter INSERT mode):

```
#include <stdio.h>
#define N 10 // this is how we can emulate the input
int main()
{
    printf(" * %s\n", "DSA"); // displays a string "DSA"
    printf(" * %c\n", '='); // displays a character '='
    printf(" * %d\n", 3 * N); // displays an integer number 30
    printf(" * %f\n", 2.321); // displays a real number 2.321
    return 0;
}
```
- save the file (press 'esc' to exit INSERT mode and type ':wq')
- compile the program: `cc -o example1 example1.c`
- run the program: `./example1`

C Programming Language

Example 2: Using Arrays and Random Numbers

Write a C program that prints n unique random numbers which are greater or equal 0 and less than k .

```
#include <stdio.h>
#include <stdlib.h>
#define N 10 // input 1 - the number of random numbers
#define K 100 // input 2 - the range of the random numbers
int main() {
    int i,j,A[N];
    for (i=0; i<N; i++) {
        A[i] = rand() % K; // generating numbers in [0..K-1]
        for (j=0; j<i; j++) { // checking if the number already in
            if (A[i]==A[j]) {
                i--; break; // if number already exists, reset generation
            }
        }
    }
    for (i=0; i<N; i++) printf("%i\n",A[i]);
}
```

C Programming Language

Example 3: Recursion

Write a C program that prints n -th Fibonacci number. The number of recursion is the input parameter.

```
#include <stdio.h>

int fib(int n) {
    if (n <= 2) {
        return 1;
    } else {
        return fib(n-1) + fib(n-2);
    }
}

int main(int argc, char *argv[]) {
    int n = atoi(argv[1]);
    printf("%d\n", fib(n));
    return 0;
}
```

C Programming Language

Example 4: Using glut

Write a C program that prints a triangle using *glut* library.

```
#include <GL/glut.h>

void DrawGLScene() {
    glClearColor(1,1,1,0); glColor3f(0,0,0);
    glClear(GL_COLOR_BUFFER_BIT); glBegin(GL_LINE_STRIP);
    glVertex2f (-0.5, 0.0); glVertex2f (0.5, 0.0);
    glVertex2f (0.0, 0.7); glVertex2f (-0.5, 0.0);
    glEnd(); glFlush();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutCreateWindow("Simple OpenGL Example");
    glutDisplayFunc(DrawGLScene);
    glutMainLoop();
}

// gcc -lglut -o example4 example4.c; ./example4
```

Useful Links

- C for Java programmers
<http://www.cs.vu.nl/~jason/college/course.html>
- C/C++ reference
<http://www.cppreference.com>

Assignment 01

Task 1

Given an array $A[n]$ of n integers, write a C program that outputs a two-dimensional array $B[n][n]$ in which the entry $B[i][j]$ for $i < j$ contains the sum of the array entries $A[i]$ through $A[j]$. That is:

$$B[i][j] = \begin{cases} A[i] + A[i + 1] + \dots + A[j - 1] + A[j] & , i \leq j \\ 0 & , i > j \end{cases}$$

Assignment 01

Task 2

Implement the insertion sort algorithm described in the lecture notes to sort an array of n integers. Use an array of random integers as an input. Measure running times for different n values and fill in the table:

n	time
100	
1 000	
10 000	
100 000	

Use `rand()` function from `<stdlib.h>` library to generate pseudo-random numbers. To measure a program execution time in Linux use command
`/usr/bin/time -f "%e" <your_program>`

Assignment 01

Task 3

A fractal is generally a fragmented geometric shape that can be split into parts, each of which is a reduced-size copy of the whole. The *Sierpinski Triangle* in this figure is a fragment and the black triangles can be drawn recursively.

- Implement a recursive algorithm that prints the drawing instructions for i -th iteration of Sierpinski Triangle.
- The input parameter is the number of iterations.

Hint: Use `glut` as rendering library. The code template is available on our web page <http://www.inf.unibz.it/dis/teaching/DSA/>

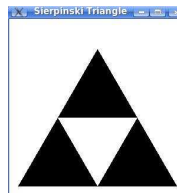


Figure: Sierpinski Triangle