udstasksheet – Documentation

Robert Pietsch (robert.pietsch@uni-saarland.de)

Task 1 – Getting Started

The easiest way to use this template is to copy the fitting example code from the `examples/` folder. In there, you will find three different variants for different ways of submitting tasks:

- `all-exercises-in-one-file` The whole source code for each task sheet is packed into a single LaTeX source file. Recommended if you submit your sheets alone and all tasks should be submitted in one PDF file.
- `exercises-in-split-files` The source code is split into a seperate file for each task but all tasks are compiled into a single PDF file. Recommended if you solve sheets in a group (especially when using Git) and all tasks should be submitted in one PDF file.
- `pdf-per-exercise` Both the source code and the PDF output is split by task into seperate files. Recommended if task sheets should be submitted with a PDF per task instead per sheet.

Once you picked the right example code, you have to change a few lines to it to be ready for use.
First of all, you should set the language for your submission. To do so, enter the correct argument (`ngerman` or `english`) for the `\documentclass` command.

```
\documentclass[10pt,a4paper,ngerman]{article} % For submissions in German
\documentclass[10pt,a4paper,english]{article} % For submissions in English
```

Having done this, you have to set the correct path to the headers directory. You can do so by changing the following line. Remember that `../` means “go up to the parent folder”.

```
\def\headersdir{../../headers/}
```

You should also enter some information for the title and page headers. These include the name of the lecture (`\lecture`), the task sheet number (`\setsheetno`, remove to hide the sheet number), the due date (`\date`, remove to hide due date), and the authors (`\author`). Make sure to put these in the preamble (the code before `\begin{document}`).

```
\lecture{Foundation of \LaTeX}
\setsheetno{1}
\date{March 13}
\author{Dr. Herrmann Einstein (4201337)}
```

With all of this set, you are ready to write down your solutions. Have fun!

Task 2 – Special Task Sheet Macros

To mark the beginning of a new task, subtask, or sub-subtask use the following macros. You can specify a title for the given ((sub-)sub-)task by putting it in square brackets after the respective macro.

```
\ex
\subex
\subsubex

\ex[An awesome title]
\subex[An even more awesome title]
\subsubex[The most awesome title]
```

To change the style of numbering for the sheet or for tasks, you can use the following macros. These macros can also be used for ((sub-)sub-)tasks by replacing the `\sheet` part inside the macros by `\ex`, `\subex`, or `\subsubex`, respectivly.

```
\sheetstylearabic   % Arabic numerals (1, 2, 3, ...)
\sheetstyleroman    % Lowercase roman numerals (i, ii, iii, ...)
\sheetstyleRoman    % Uppercase roman numerals (I, II, III, ...)
\sheetstylealph     % Lowercase latin alphabet (a, b, c, d)
\sheetstyleAlph     % Uppercase latin alphabet (A, B, C, D)
```

To change the number of a sheet or task, you can use the following macros. Independent of the numbering style, the new number has to be provided in arabic numerals.

```
\setsheetno{1}
\setexno{3}
\setsubexno{3}
\setsubsubexno{7}
```

Task 3 – Typesetting Math

To make typesetting math more convenient, this template provides a plethora of macros that can be used in math mode.

Source Code	Result	Description
`\dif x`	$\mathrm{d}x$	d for integrals and differentials
`\im x`	$\operatorname{Im} x$	Imaginary part
`\re x`	$\operatorname{Re} x$	Real part
`45 \dgr`	45°	Degree sign
`x \inv`	x^{-1}	Inverse
`x \tx {3}`	$x \cdot 10^3$	Power of ten (scientific notation)
`\abs {x}`	$\lvert x \rvert$	Absolute value
`\brk {x}`	(x)	Round brackets
`\sbr {x}`	$[x]$	Square brackets
`\ceil {x}`	$\lceil x \rceil$	Euclidian round-up brackets
`\floor {x}`	$\lfloor x \rfloor$	Euclidian round-down brackets
`\set {x}`	$\{x\}$	Set
`\set {x \where y}`	$\{x \mid y\}$	Set with condition
`x \bdiv y`	$x \operatorname{div} y$	Integer division
`x \bmod y`	$x \bmod y$	Integer division remainder
`\vcol`	$:$	Vertically centered colon
`\defeq`	$:=$	Definition operator
`\eqdef`	$=:$	Mirrored definition operator
`\NN`	$\mathbb{N}$	Set of Natural Numbers
`\ZZ`	$\mathbb{Z}$	Set of Integers
`\QQ`	$\mathbb{Q}$	Set of Rational Numbers
`\RR`	$\mathbb{R}$	Set of Real Numbers
`\CC`	$\mathbb{C}$	Set of Complex Numbers
`\PP`	$\mathbb{P}$	Set of Prime Numbers
`\HH`	$\mathbb{H}$	Set of Quarternions
`\BB`	$\mathbb{B}$	Set of Booleans
`\EE`	$\mathbb{E}$	Expected value
`\Ave`	Ave	Average
`\Var`	Var	Variance
`\Cov`	Cov	Covariance
`\Pr`	Pr	Probability
`\frac {x}{y}`	$\frac{x}{y}$	Fraction
`\sfrac {x}{y}`	x/y	Diagonal fraction
`\bsfrac {x}{y}`	$y\backslash x$	Backwards diagonal fraction

Continued on the following page...

Source Code	Result	Description
`\bigO`	O	Landau symbols (Big-O notation)
`\R`	R	Recursive languages
`\RE`	RE	Recursively enumerable languages
`\coRE`	co−RE	Complement-recursively enumerable languages
`\REC`	REC	Recursively decidable languages
`\P`	P	Polynomial-time problems
`\NP`	NP	Nondeterministic poly-time problems
`\coNP`	co−NP	Complement-nondeterministic poly-time problems
`\DSpace`	DSpace	Deterministic space class
`\NSpace`	NSpace	Nondeterministic space class
`\DTime`	DTime	Deterministic time class
`\NTime`	NTime	Nondeterministic time class

Apart from these new macros, the template loads the `amsmath` and `amsthm` packages to provide the most relevant math macros and environments.

Task 4 – Displaying Code

Task 4.a – Real code

To render code into your submissions, you can use the features provided by `minted`. Take the following python snippet.

```python
def fib(n):
    if n == 0 or n == 1:
        return 1
    return fib(n - 1) + fib(n - 2)
```

You can render it using this code:

```
\begin{minted}{python}
    def fib(n):
        if n == 0 or n == 1:
            return 1
        return fib(n - 1) + fib(n - 2)
\end{minted}
```

Task 4.b – Pseudocode

To produce pseudocode, you can use the `pseudo` environment.

Take this sample program from the TCS lecture:

Program 1: Construction P

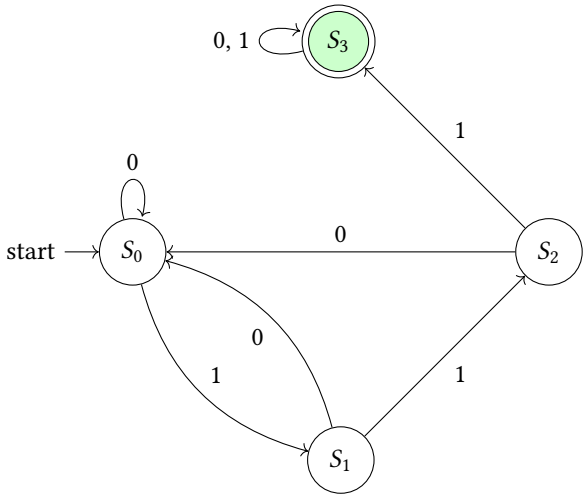
```
1  P';
2  if x_1 ≠ 0 then
3  |  x_0 := x_0
4  else
5  |  for x_0 do
6  |  |  Do nothing...
7  |  od
8  fi
9  ;
10 if x_0 = 0 then
11 |  x_0 := 1;
12 |  while x_0 ≠ 0 do
13 |  |  x_1 := 1
14 |  od
15 fi
```

You can render it using this code:

```
\begin{pseudo}
    $P'$\;
    \eIf{$x_1 \neq 0$}{
        $x_0 \defeq x_0$
    }{
        \For{$x_0$}{
            Do nothing...
        }
    }\;
    \If{$x_0 = 0$}{
        $x_0 \defeq 1$\;
        \While{$x_0 \neq 0$}{
            $x_1 \defeq 1$
        }
    }
    \caption{Construction $P$}
\end{pseudo}
```

Task 5 – Drawing Automata

You can use TikZ to draw automata. See the following example:



You can produce it using this code:

```latex
\begin{tikzpicture}[node distance=3cm, auto]
    % Define states
    \node[state, initial] (s0)  {$S_0$};
    \node[state, below right = of s0] (s1)  {$S_1$};
    \node[state, above right = of s1] (s2)  {$S_2$};
    \node[state, accepting, above left = of s2] (s3)  {$S_3$};

    % Define transitions
    \path[->]
        (s0) edge [loop above] node {0} ()
        (s3) edge [loop left] node {0, 1} ()

        (s0) edge [bend right] node {1} (s1)
        (s1) edge [bend right] node {0} (s0)
        (s1) edge [swap] node {1} (s2)
        (s2) edge [swap] node {1} (s3)
        (s2) edge [swap] node {0} (s0)
    ;
\end{tikzpicture}
```