Intro to LaTeX

March 4, 2025

Contents

1 Why Use LaTeX?

Similar to how you write C code then tell the compiler (ex. gcc) to convert it into an executable, you can write LaTeX instructions to a .tex file, then compile it into a PDF document. So, it is fundamentally different from editors you've used before like MS Word and Google Docs. Instead of formatting things on-the-fly, you write formatting commands that get compiled into documents robustly.

1.1 Benefits

- **Professional Quality**: Used for things like academic papers, research, and technical documents. You've probably noticed some profs prepare their lecture slideshows using LaTeX.
- **Templates**: Sometimes you can start from a given template (.tex file) and fill in the rest. The benefit is that formatting commands are already there, so you just have to fill in the content, and you get a nice looking document. For example, I'd recommend using a LaTeX template for your resume. It will probably look better and also be picked up by resume screening tools better than Google Docs/Word.
- **Styles**: Similar to templates, you can create custom styles and copy-paste them across .tex files for easy re-use. For example, the style for a code block in C.
- **Math Support**: Lots of commands for formatting math equations, both inline and in separate blocks.
- **Automation**: Handles table of contents, references, numbering, etc. automatically

2 Installation and Setup

Note, we'll dicuss how you can setup/use LaTeX on your local devices. An alternative could be to use online software like Overleaf. It's up to you which one you want to use, but I've found that VS Code on my local device with the LaTeX Workshop extension is pretty efficient. On a software team, you would likely use LaTeX with Git, so you get the benefits of collaboration and version control. However, Overleaf can be useful if you want to work on a shared professional document (ex. research paper) with collaborators who are not familiar with Git.

2.1 Linux/WSL

I don't recommend instaling all packages (texlive-full), as it's about 6 GB and you won't need most of it. Instead, you can run the following in the terminal to get most of the widely used packages and fonts (this is 670 MB):

```
# In terminal
sudo apt install texlive-latex-recommended texlive-latex-extra
   texlive-fonts-recommended
```

```
# If that's too large, you can try only getting the base packages
sudo apt install texlive-latex-base

# Verify the install in terminal
latex --version

# Create a .tex file using an editor (or do this through VS Code
   interface)
nano file.tex

# when you want to compile to PDF run:
pdflatex file.tex

# You should see a bunch of files appear like file.aux, file.log,
   ..., you can ignore those, and just focus on file.pdf
```

2.2 On MacOS:

Mac users can use MacTex (https://tug.org/mactex/). Or you can use a smaller version (basicTex) and add the packages you actually need:

```
# Using Homebrew
brew install --cask basictex

# Add packages using TeX Live Manager (tlmgr)
# Update the package manager first
sudo tlmgr update --self

# Install recommended packages similar to texlive-latex-recommended
sudo tlmgr install collection-latexrecommended

# Install extra packages if needed
sudo tlmgr install collection-latexextra

# Install specific packages
sudo tlmgr install <package-name>
```

If you have the space and want the full MacTex:

```
# Using Homebrew
brew install --cask mactex
```

2.3 VS Code Integration

I highly recommend getting the VS Code **LaTeX Workshop** extension:

1. Search for the LaTeX Workshop extension in VS Code Extensions and install it

2. You should see 'Build LaTeX Project' and 'View LaTeX PDF file' options appear in the top right of VS Code (next to Run Code)

3. You can use shortcuts for these: Build (Ctrl+Alt+B) and Preview (Ctrl+Alt+V)

4. Every time you edit and save your .tex file, changes are automatically reflected in the preview PDF

3 LaTeX Overview

3.1 Document Structure

Every LaTeX document follows this basic structure:

```
\documentclass[12pt]{article}
\usepackage{amsmath}
\usepackage{graphicx}

\title{Document Title}
\author{Your Name}
\date{\today}

\begin{document}
\maketitle          % Creates the title
\tableofcontents    % Optional

% Your content here with sections
\section{Introduction}
Some text...

% Other sections etc.
...

\end{document}
```

3.2 Common Document Elements

3.2.1 Sections and Paragraphs

```
\section{Main Section}
\subsection{Sub Section}
\subsubsection{Sub Sub Section}

This is a paragraph.
```

```
This is another paragraph.
```

3.2.2 Text Formatting

Bold *Italic* Underlined `Monospace`

```
\textbf{Bold}
\textit{Italic}
\underline{Underlined}
\texttt{Monospace}
```

3.2.3 Math Content

Inline math: $E = mc^2$
Display math:

$$E = mc^2$$

Numbered equation:

$$E = mc^2 \tag{1}$$

```
Inline math: $E = mc^2$

Display math:
$$E = mc^2$$

Numbered equation:
\begin{equation}
E = mc^2
\end{equation}
```

3.2.4 Lists

- First item
- Second item

1. First item
2. Second item
   (a) sub item 1
   (b) sub item 2
   (c) ...

```latex
\begin{itemize}  % Bulleted list
  \item First item
  \item Second item
\end{itemize}

\begin{enumerate}  % Numbered list
  \item First item
  \item Second item
  \begin{enumerate}
      \item sub item 1
      \item sub item 2
      \item ...
  \end{enumerate}
\end{enumerate}
```

3.2.5 Tables

Table 1: Test Table

A	B	C
1	2	3
4	5	6

```latex
% See how the table is constructed row by row
% Also note the [H] and \vspace for positioning

\begin{table}[H]
  \centering
  \caption{Test Table}
  \vspace{2mm}
  \begin{tabular}{|l|c|r|}  % Left, center, right aligned
     columns
    \hline
    A & B & C \\
    \hline
    1 & 2 & 3 \\
    4 & 5 & 6 \\
    \hline
  \end{tabular}
\end{table}
```

3.2.6 Figures

```
\begin{figure}
  \centering
  \includegraphics[width=0.7\textwidth]{image.png}
  \caption{An example image.}
  \label{fig:example}
\end{figure}
```

Reference the figure with `\ref{fig:example}`. You can put the image file in the same folder as your .tex file, or put it in a subfolder and provide the relative path. Also, you can use figures for plots, so you don't need to bother learning plot commands in LaTeX. Basically, you can just export the plot png/jpeg from other software like Python and load it as a figure in LaTeX.

3.2.7 Code Blocks

You can provide the language for the code block so that relevant strings can be highlighted. You can also create custom styles for different languages, and display them differently. For example, compare the LaTeX code blocks in this doc to the following C code block.

```
int main() {
    printf("Hello, World!");
    return 0;
}
```

```
\usepackage{listings}

\begin{lstlisting}[language=C]
int main() {
    printf("Hello, World!");
    return 0;
}
end{lstlisting}
```

3.2.8 Bibliography

Create a .bib file with your references:

```
@article{kumar2025,
  author  = {Kumar, Samarth},
  title   = {Some Paper},
  journal = {Some Journal},
  year    = {2025}
}
```

In your main document:

```
\cite{kumar2025} showed that...

\bibliographystyle{plain}
\bibliography{references}
```

Compile with: `pdflatex file.tex`, `bibtex file`, `pdflatex file.tex` (twice)

4 Exercise: Code Documentation With LaTeX

In this exercise, you will create a LaTeX document that includes a C program along with documentation of your approach and test cases.

4.1 Task in C

Write a C program that checks if a given string is a palindrome. A palindrome reads the same forward and backward (ex. "racecar"). You can also consider an empty string as a palindrome. You can assume strings are empty or composed of lowercase letters only. Your program should:

- create an `isPalindrome()` function which takes a string as input. The return type should be `bool`.
- Include an array of strings called `testStrings` in `main()`.

  Recall: a string is really an array of characters. So an array of strings can be thought of as a 2D char array. In C, we can use the following. The first approach requires knowing the max length of strings within the array (but entries are modifiable). The second approach makes the strings **read-only**. You can also use `malloc` for dynamic allocation (not shown here).

  - `char testStrings[][MAX_STR_LEN] = {...};`
  - `const char* testStrings[] = {...};`
- Loop over testStrings, passing each test string into the isPalindrome() function, and print each input/output result on a new line.
  - Use the ternary operator to print "true" or "false" inside `printf` based on your isPalindrome() result

Bonus: try implementing `bool isPalindromeNum(int n)`, which takes an integer input instead of string, and returns whether it is a palindrome using only mathematical operations (Hint: think of modulo (%) and how you can 'reverse' a number). You can assume inputs will be ≥ 0 (unsigned), and all inputs will fit inside 32 bits and will not cause overflow. You can include a `testNums` array in `main()`.

4.2 Task in LaTeX

Document your program using:

- Separate code blocks for:
  1. the top of your C code (libraries used)
  2. your isPalindrome() function
  3. your main() function
- An itemized list below your isPalindrome() code block explaining your approach.
- An enumerated list below the main() code block which explains your choice of test strings. Have you covered the necessary edge/main cases?
- A table showing all of your test cases. There should be columns for input, expected output, actual output, and explanations (if necessary).
- A figure which contains a screenshot of your code output (test case results) in the terminal

Make sure you compile your .tex file with `pdflatex yourFile.tex` (or with the VS Code extension). You can show me your final PDF.