codeanatomy – Draw Code Anatomy*

Usage

Hồng-Phúc Bùi †

Released 2023/01/24

Contents

1 Introduction

The idea of this Package is to typeset illustrations of pieces of code with annotations on each single parts of code (Code Anatomy). The origin of this idear is code illustrations in the texbook [1]. This package just provides tool to draw those figures.

2 Tutorial

In this tutorial we will draw an anatomy of a function like the figure 1 step by step.

2.1 Package Usage

To use this package, just insert `\usepackage{codeanatomy}` in your LaTeX file.

*This file describes v0.4-Beta, last revised 2023/01/24.

†E-mail: hong-phuc.bui (at) htwsaar dot de

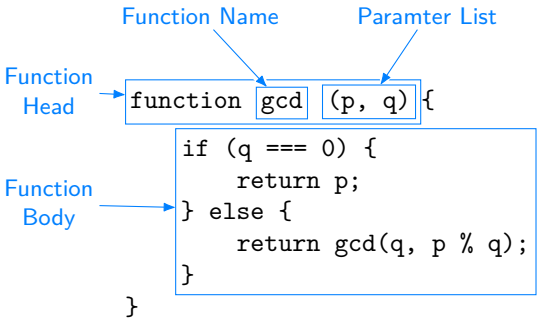


Figure 1: Anatomy of a function

2.2 Create an anatomy environment

Next step is to create a `tikzpicture` environment. All commands in this package must be placed in a `tikzpicture` environment with option `remember picture`.

necsessary for later mark

```
\begin{tikzpicture}[remember picture]
{[on background layer]\draw[code grid debug] (-0.5,-0.5) grid (6.5,4.5);}
% ...
\end{tikzpicture}
```

plot a thin grey grid on background
usefull to determinate coordinate of annotation

2.3 Typeset Code Listing in a TikZ-Node

As next step we need to put the piece of code in the `tikzpicture` environment using the command `\codeBlock`.

```
\begin{tikzpicture}[remember picture]
{[on background layer]\draw[code grid debug]
        (-0.5,-0.5) grid (6.5,4.5);}
\codeBlock{%
function gcd(p, q) {
    if (q === 0) {
        return p;
    } else {
        return gcd(q, p%q);
    }
}
}
\end{tikzpicture}
```

Anatomy of Code

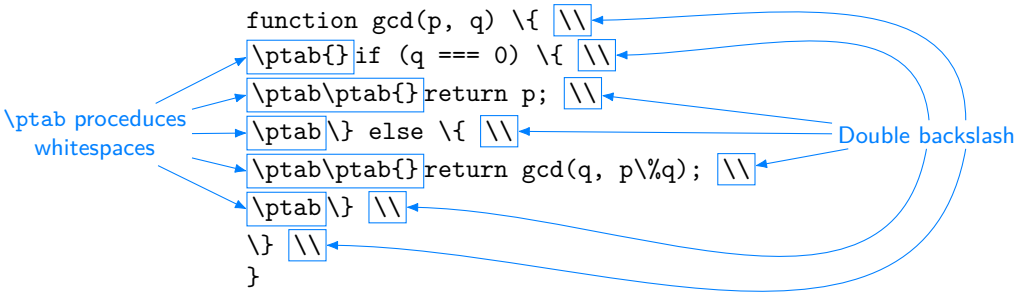
The result of the above code is shown in the figure 2, which is not what we really want. All extra whitespaces and newlines in the listing are removed, further more `{` and `}` are interpreted as LaTeX tokens and are not displayed.

We need to put `\ptab` and `\\` into code to keep whitespaces and newlines. The characters `{` and `}` also need to be escaped by prefixing a `\` before them.

```
function gcd(p, q)  if (q === 0)  return p;  else  return gcd(q, p%q);
```

Figure 2: Unformated Code

```
\begin{tikzpicture}[remember picture]
{[on background layer]\draw[code grid debug]
        (-0.5,-0.5) grid (6.5,4.5);}
\codeBlock{%
function gcd(p, q) \{ \\
\ptab{}if (q === 0) \{ \\
\ptab\ptab{}return p; \\
\ptab\} else \{ \\
\ptab\ptab{}return gcd(q, p\%q); \\
\ptab\} \\
\} \\
}
\end{tikzpicture}
```



The result (figure 3) is much more like what we expect than the version before (figure 2).

```
function gcd(p, q) {
    if (q === 0) {
        return p;
    } else {
        return gcd(q, p%q);
    }
}
```

Figure 3: Formated Function

2.4 Mark Parts of Code

Now we can mark interesting parts of code with a blue boxes created by `\cPart`. At some positions we can use `\\[<length>]` to add a little amount of vertical space, sothat the boxes do not touch each others.

```
\begin{tikzpicture}[remember picture]
{[on background layer]\draw[code grid debug]
        (-0.5,-0.5) grid (6.5,4.5);}
\codeBlock{%
\cPart{functionHead}{function\cPart{functionName}{gcd} \cPart{paramList}{(p, q)}} \{
\\[2.5pt]
\ptab{} \mtPoint{mostLeft} if (q === 0) \{ \\
\ptab \ptab{}return p; \\
\ptab \} else \{ \\
\ptab \ptab{}return gcd(q, p\%q); \extremPoint{mostRight} \\
\ptab \mbPoint{mostBottom} \} \\
\} \\
}

\fitExtrem{functionBody}{(mostLeft) (mostRight) (mostBottom)}
\end{tikzpicture}
```

cPart can be nested

extremPoints are used to mark outer most points of a multiline code part

fitExtrem draws a rectangle which covers all passed extrem points

```
function gcd (p, q) {
    if (q === 0) {
        return p;
    } else {
        return gcd(q, p%q);
    }
}
```

Figure 4: Function with marked parts

2.5 Create Annotation Labels

We can use `\codeAnnotation` to create annotation labels for each parts of code. To draw an arrow from label to a code part we can use the Ti*k*Zcommand `\draw[->,annotation]` (⟨*annotation label*⟩) `--` (⟨*code part*⟩) `;`.
Whereas (⟨*annotation label*⟩)s are the first argument of `\codeAnnotations` and (⟨*code part*⟩)s are the first argument of `\cParts`.

```
\begin{tikzpicture}[remember picture]
{[on background layer]\draw[code grid debug]
        (-0.5,-0.5) grid (6.5,4.5);}
\codeBlock{%
\cPart{functionHead} {function \cPart{functionName}{gcd} \cPart{paramList}{(p, q)}} \{
\\[2.5pt]
\ptab{}\mtPoint{mostLeft} if (q === 0) \{ \\
\ptab\ptab{} return p; \\
\ptab\} else \{ \\
\ptab\ptab{} return gcd(q, p\%q); \extremPoint{mostRight} \\
\ptab\mbPoint{mostBottom}\} \\
\}
}

\fitExtrem{functionBody}{(mostLeft) (mostRight) (mostBottom)}

% Annotations
\codeAnnotation{functionHeadText}(-1,3){Function\\head}
\codeAnnotation{functionBodyText}(-1,1){Function\\body}
\codeAnnotation{functionNameText}( 1,4){Function\\name}
\codeAnnotation{paramListText} ( 3,4){Parameter\\list}

% Annotation labels to code parts
\draw[->,annotation] (functionHeadText) -- (functionHead);
\draw[->,annotation] (functionBodyText) -- (functionBody);
\draw[->,annotation] (functionNameText) -- (functionName);
\draw[->,annotation] (paramListText) -- (paramList);
\end{tikzpicture}
```

Instead of operator `--` we can use operator `to [`⟨*TikZ options*⟩`]` to draw a path from (⟨*annotation label*⟩) to (⟨*code part*⟩). Finally we can remove the command `\draw[code grid debug]...` at the begin of the `tikzpicture`. The final result is shown in the figure 5, which is almost the same as figure 1.

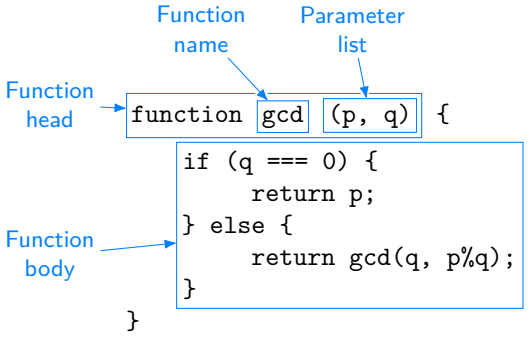


Figure 5: Function with Annotation Labels

3 Usage in conjuntion with listings

As we see in the previous section, the command `\codeBlock` cannot typeset whitespaces correctly as we expect. A way to typeset code listing is using the package listings. See `codeanatomy.lstlisting.pdf` for more information.

4 Customize style

Maybe we want to highlight the function name `gcd` with some background color like figure 6. The best way to do this task is to assign this highlight format to a Ti*k*Z style –e.g. `jsid`– and apply the style to the highlighted nodes. So they have the same format.

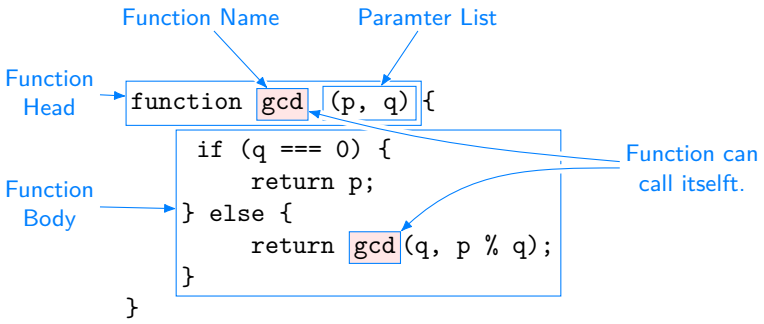


Figure 6: Highlighted name of the function

```
\begin{tikzpicture}[remember picture]
\tikzstyle{jsid} = [code part, fill=red!10]
\codeBlock{
\cPart{functionHead} {function \cPart[jsid]{functionName}{gcd} \cPart{parameterList}{(p, q)}} \{ \\[2.5pt]
\ptab{}\mtPoint{mostLeft} if (q === 0) \{ \\
\ptab\ptab{} return p; \\
\ptab{}\} else \{ \\
\ptab\ptab{} return \cPart[jsid]{recursive}{gcd}(q, p\%q); \extremPoint{mostRight}\\
\ptab{}\mbPoint{mostBottom}\} \\
\}
}

\fitExtrem{functionBody}{(mostLeft) (mostRight) (mostBottom)}

% Annotations
\codeAnnotation{functionHeadText} (-1,3) {Function\\Head}
\codeAnnotation{functionBodyText} (-1,1.5) {Function\\Body}
\codeAnnotation{functionNameText} ( 1,4) {Function Name}
\codeAnnotation{parameterListText}( 4,4) {Parameter List}

% Annotation labels to code parts
\draw[->,annotation] (functionBodyText) -- (functionBody);
\draw[->,annotation] (functionHeadText) -- (functionHead);
\draw[->,annotation] (functionNameText) -- (functionName);
\draw[->,annotation] (parameterListText) -- (paramList);

\draw[->, annotation] (recursiveText) to[out=175,in=-15] (functionName);
\draw[->, annotation] (recursiveText) to[out=180,in=45] (recursive.north east);
\end{tikzpicture}
```

Change History

v0.4-Beta

General: Add Option [⟨*style*⟩] to `cPart` and `iPart` 6

References

[1] Robert Sedgewick and Kevin Wayne. *Computer Science. An Interdisciplinary Approach*. Addison-Wesley, 2016.