Narrative Programming Style

Ryszard Kubiak

BachoTEX 2024

Goal: Publish GUST Foundry Sources

- Promise from BachoTEX 2023: Simplify to share
- Name changed from Algotype to Fontplant
- Major changes in Python part
- Minor changes in Metapost part
- Fonts 99.31415% the same as old versions
- Documentation: fontplant-story.pdf

TEX: The Program – tex.web

```
@ Here is a similar routine, but it compares two strings in the string pool,
and it does not assume that they have the same length.

@p function str_eq_str(@!s,@!t:str_number):boolean;
  {test equality of strings}
label not_found; {loop exit}
var j,@!k: pool_pointer; {running indices}
@!result: boolean; {result of comparison}
begin result:=false;
if length(s)<>length(t) then goto not_found;
j:=str_start[s]; k:=str_start[t];
while j<str_start[s+1] do
  begin if str_pool[j]<>str_pool[k] then goto not_found;
  incr(j); incr(k);
  end;
result:=true;
not_found: str_eq_str:=result;
end;
```

tangle(tex.web) → tex.tex

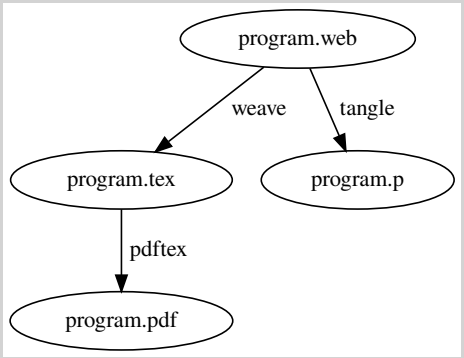
```
\M46. Here is a similar routine, but it compares two strings in the string
pool,
and it does not assume that they have the same length.

\Y\P\4\&{function}\1\  \37$\\{str\_eq\_str}(\|s,\39\|t:\\{str\_number})$: \37%
\\{boolean};\C{test equality of strings}\6
\4\&{label} \37\\{not\_found};\C{loop exit}\6
\4\&{var} \37$\|j,\39\|k$: \37\\{pool\_pointer};\C{running indices}\6
\\{result}: \37\\{boolean};\C{result of comparison}\2\6
\&{begin} \37$\\{result}\K\\{false}$;\6
\&{if} $\\{length}(\|s)\I\\{length}(\|t)$ \1\&{then}\5
\&{goto} \37\\{not\_found};\2\6
$\|j\K\\{str\_start}[\|s]$;\5
$\|k\K\\{str\_start}[\|t]$;\6
\&{while} $\|j<\\{str\_start}[\|s+1]$ \1\&{do}\6
\&{begin} \37\&{if} $\\{str\_pool}[\|j]\I\\{str\_pool}[\|k]$ \1\&{then}\5
\&{goto} \37\\{not\_found};\2\6
$\\{incr}(\|j)$;\5
$\\{incr}(\|k)$;\6
\&{end};\2\6
$\\{result}\K\\{true}$;\6
\4\\{not\_found}: \37$\\{str\_eq\_str}\K\\{result}$;\6
\&{end};\par
\fi
```

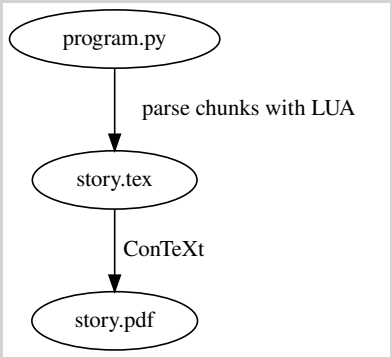
weave(tex.web) → tex.pdf

46. Here is a similar routine, but it compares two strings in the string pool, they have the same length.

```
function str_eq_str(s,t : str_number): boolean;   { test equality of strings }
  label not_found;   { loop exit }
  var j,k: pool_pointer;   { running indices }
    result: boolean;   { result of comparison }
  begin result ← false;
  if length(s) ≠ length(t) then goto not_found;
  j ← str_start[s]; k ← str_start[t];
  while j < str_start[s+1] do
    begin if str_pool[j] ≠ str_pool[k] then goto not_found;
    incr(j); incr(k);
    end;
  result ← true;
not_found: str_eq_str ← result;
  end;
```

Literate vs. Narrative





A narrative chunk

```
#+
#: Here is the main function for parsing an OTI file. It reads
#: the entire contents of the file into a string and delegates
#: the task of parsing it to the `parse_oti_text` function.

def parse_oti_file(oti_path: str, with_epses: bool) -> OTI:
    with open(oti_path, 'r') as oti_file:
        txt = oti_file.read()
        return parse_oti_text(txt, with_epses)

#: You may have noticed that the function obtains a Boolean parameter
#: named `with_epses`. This parameter is relevant because Metapost can
#: execute scripts in two modes: one involves evaluating EPS files that
#: represent glyph outlines, while the other entails generating encoding
#: files, without the necessity to process glyph outlines. FontPlant sets
#: the value `with_epses=True` for the function when processing |PFB:| or
#: |OTF:| lines from a bonds file. On the other hand, the value
#: `with_epses=False` is used for generating |ENC| files.
#-
```

Chunk in PDF

oti_lib 877

5.5 The parsing process as such – do one thing at a time

Here is the main function for parsing an OTI file. It reads the entire contents of the file into a string and delegates the task of parsing it to the `parse_oti_text` function.

```
def parse_oti_file(oti_path: str, with_epses: bool) -> OTI:
    with open(oti_path, 'r') as oti_file:
        txt = oti_file.read()
    return parse_oti_text(txt, with_epses)
```

You may have noticed that the function obtains a Boolean parameter named `with_epses`. This parameter is relevant because Metapost can execute scripts in two modes: one involves evaluating EPS files that represent glyph outlines, while the other entails generating encoding files, without the necessity to process glyph outlines. FontPlant sets the value `with_epses=True` for the function when processing `PFB:` or `OTF:` lines from a bonds file. On the other hand, the value `with_epses=False` is used for generating `ENC` files.

lib 848

The following function undertakes a more intricate task compared to its caller, `parse_oti_file`. It delves into the intricacies of the OTI file internals, transforming them into the resulting object of type `OTI`.

Characteristics of chunks

- Starts with `#+`, ends with `#-`
- Doc part: lines starting with `#:`
- Code part: all other lines within the chunk
- A sequence of Doc and Code segments
- Each chunk has a label

Chunks go to ConTEXt

- `chunks_reader.read_chunks_from_file(file_name)`

  Chunk labels evaluated where not given

- `chunk_splitter.split_chunk_into_segments(chunk)`

  `Doc` and `Code` segments recognized

- `for_tex.prepare_doc_segment(segment)`

  `` `abc` `` → `\py{abc}`

  `|xyz|` → `\type{xyz}`

- `for_tex.prepare_code_segment(segment)`

  `@dataclass` annotations removed

  Code blocks indentation reduced to zero

ConTEXt welcomes chunks

- In the beginning of `fontplant-story.tex`:

```
\startluacode
  chunks["runner"] =
      read_chunks_from_module("runner.py")
  chunks["bonds_lib"] =
      read_chunks_from_module("bonds_lib.py")
  ...
\stopluacode
```

- Later on in `fontplant-story.tex`:

```
\chunk{runner}{main routine}
\chunk{runner}{run}

\chunk{bonds_lib}{parse_bonds_file}
\chunk{bonds_lib}{parse_bonds_file_text}
```

Programmer's experience

- Notation for chunks invented and used only recently
- The story is being written only after the program is ready
- Good: documentation close to code
- Documentation influences code
- Need for chunk formatting tools (`vscode`, `Emacs`)
- Python support in `vscode` of great help

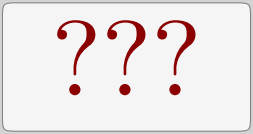
Author's experience

- Spirit of comments in chunks similar to D.E.K's
- Dangerous words: two, three, below, above
- Also the names of: functions, classes, variables
- Overfulls possible as result of long names
- Avoid links to the external story
- Support on English style from AI's ChatGPT

Reader's comfort

- Need for Python syntax colouring
- Python's `self` problem
- Cross references?
- Index?

Reader's experience



Fontplant story 60% ready