Dokumentace úlohy SYN: Zvýraznění syntaxe v PHP do IPP 2016/2017

Jméno a příjmení: Dominik Skála

Login: xskala11

Tato dokumentace slouží jako popis implementace projektu SYN/06. Pojednává o jednotlivých problémech při implementaci, v několika místech také o jednoduchostech implementace. Hlavním popisem dokumentace je hlavní popis celkové práce a celkové funkčnosti skriptu.

Celý skript pracuje na základě jednoduché kostry, která se skládá z několika částí: zpracování vstupu, kontrola přítomnosti a zpracování formátování, načtení vstupu, formátování vstupu, tisk vstupu.

Každá z těchto částí je zapsána jednou či více funkcemi, které jsou volány uvnitř jednotlivých souborů, každý tento soubor je zpracováván jako třída, případně jsou volány přímo kostrou skriptu, níže bude každá z nich blíže popsána.

Jak již byl zmíněno výše, skript má určitou kostru, ta je nastíněna v hlavním souboru skriptu s názvem: „`syn.php`“. Prvním krokem je zahrnutí tří pomocných souborů obsahujících externí funkce, potřebné pro práci skriptu. Následně je provedena kontrola vstupních argumentů skriptu, pokud vše proběhne v pořádku, dojde k načtení formátovacího souboru, následně dojde také k jeho rozboru. Provede se kontrola validity jednotlivých regulárních výrazů a formátovacích značek HTML. Po rozboru se načte všechen potřebný vstup, ať už ze standardního vstupu terminálu, či z vstupního souboru. Provede se formátování vstupu podle rozboru, následně se tento text uloží do zadaného výstupního souboru, či se vytiskne na standardní výstup terminálu.

Prvním bodem je kontrola argumentů, to obstarává funkce `checkArguments`() ze třídy `Common`. Tato funkce obdrží na vstupu pole argumentů, které jazyk PHP samo vytváří při spuštění každého skriptu. Následně se ověří počet těchto argumentů, zda vyhovuje zadání, za využití funkce `preg_match` se regulárním výrazem zajistí kontrola vstupních argumentů – které byly předány a které ne. Pokud některý chybí či byl zadán vícekrát, skript se zachová adekvátně vzhledem k zadání. Následně se provede za účelem zrychlení skriptu kontrola na přítomnost vstupního a výstupního souboru. Pokud je skriptu předán formátovací soubor, který neexistuje, skript toto nezajímá. Možnými argumenty jsou `–input`, `--output`, `--format`, `--br` a `–help`. Příkaz `–help` však nemůže být kombinován s ostatními, slouží pouze a jen k tisku nápovědy, ostatní argumenty mohou být kombinovány dle libosti bez duplicitních výskytů.

Dalším krokem je následné načtení formátovacího souboru a jeho rozbor. To řeší funkce parseFormatFile() obsažena ve třídě `Parser`. Jejími vstupními parametry je samotná třída `Common` a zadaný formátovací soubor. Pokud k zadání formátovacího souboru nedojde, formátování se přeskočí a funkce vrací prázdné pole formátovacích prvků.

Pokud je však formátovací soubor zadán, provede se za pomocí operace `foreach` průchod souboru po řádcích, kdy dochází podle regulárního výrazu: „/([\S\s"]+[\t]+[\w\d,: \t]+)$/“ ke kontrole validity jednotlivých řádků, zda obsahují znaky, které obsahovat mají. Následně se provede rozdělení do pole polí `$formats[$i]` podle všech mezer (tabulátory, mezery). Jakmile je toto pole naplněno, dojde k průchodu tohoto pole a každý řádek je následně zpracován příslušnými funkcemi. Jednotlivé regulární výrazy jsou zpracovány funkcí `parseRegex`(), HTML tagy `parseTag`().

Samotný rozbor HTML tagů je triviální, pouze dojde ke kontrole každého prvku pomocí funkce `preg_match`(), zda obsahuje tag, který je specifikací dovolen. Pokud obsahuje jiný, či stejný tag, ale v jiném formátu než dovoluje specifikace, dojde k chybě a ukončí se skript.

Rozbor regulárních výrazů je již zábavnější ovšem při použití funkce `preg_match`() a `preg_replace`() jde opět a pouze jen o triviální automat. Zároveň však stojí za zmínku, že funkce `preg_match`() a `preg_preplace`() vcelku jednoduše umožňují implementaci rozšíření NQS, které má zajistit zkrácení zápisu regulárních výrazů. V tomto případě se opravdu jednalo pouze o 6 řádků podmínek s nahrazením textu. Je jasné, že se nejedná o plnohodnotný automat, můj účel však splnil, jsem si vědom některých chyb, kterých jsem se při implementaci tímto stylem dopustil, co je však podstatné je, že automat je schopen pracovat se základními regulárními výrazy.

Kde všude šla implementace hladce se mi vrátilo ve funkci `editInputFile`(). Tato funkce slouží pro editaci vstupního souboru na základě regulárních výrazů a HTML tagů. Každý regulární výraz je zpracován a jeho funkčnost je korektně zjištěna. Pokud není regulární výraz validní, program se ukončí, pokud však je, pokračuje se několikanásobným průchodem nejprve jednotlivých polí se shodnými texty, tyto texty i s vzdáleností od počátku slova jsou poté podle klíče (klíčem je vzdálenost od začátku slova, tedy od nultého znaku) seřazeny, mezi začátek slova a regulární výraz je konkatenován HTML tag. Prakticky to samé se děje i s pravou stranou, jen v opačném pořadí, kdy je vše seřazeno od největšího po nejmenší, inkrementuje se velikost klíče o délku levé strany HTML tagu, poté se pole znovu seřadí, tentokrát vzestupně a provede se stejná implementace, jako pro konkatenaci levých stran HTML. Výsledek se vrátí ve formě modifikovaného řetězce. Nad touto samotnou funkcí jsem strávil asi téměř dva dny čistého času, než jsem ji dovedl k dokonalosti, je zde spousta matematiky a zanořování, které v rámci polích v poli je hůře řešitelné. PHP však značně zjednodušuje práci s poli a tak mi alespoň v tomto bylo částečně uleveno. Bohužel kvůli delšímu ladění jsem byl nucen zpracovat hůře automat regulárních výrazů. Naštěstí má implementace této funkce je, troufám si po třech dnech úmorného kódování a testování říci, téměř dokonalá a bezchybná.

Ještě před tiskem formátovaného vstupu je třeba ověřit, zda nebyl zadán na vstup i argument “—br”. To ověří a zaručí funkce `insertNewLine`(). Pokud byl argument předán, na konec každého řádku se vloží řetězec: “`<br />`”.
Toto je řešeno jednoduchým rozdělením formátovaného vstupu za pomocí funkce `explode`() po nových řádcích, tedy „`\n`“, a následným „slepením“ všech řádků za pomocí funkce `implode`(), užitím řetězce: „`<br />\n`“.

Tato funkce je opravdu kratičká a tak si dovolím ji vyzdvihnout v této dokumentaci:

```
public function insertNewLine($flags, $file) {
      if ($flags["br"]) {
            return $br = implode("<br />\n", $br = explode("\n", $file));
      }
      return $file;
}
```

Jak si můžete povšimnout, funkce je opravdu jednoduchá na implementaci a byla sepsána za méně než pět minut.

Tisk do souboru je řešen funkcí `writeToFile`(). Její alternativou je `writeToStdout`(). Obě funkce dělají prakticky to samé, funkce `writeToFile`() pouze navíc otevírá pointer na soubor zadaný. Krátkou kontrolou by obě funkce šly spojit do jedné, o něco větší. Rozhodl jsem se však obě funkce oddělit, aby každá sloužila svému účelu. Jak z pohledu zkušeného programátora, tak z pohledu lajka je okamžitě jasné, k čemu která funkce slouží. Je však třeba připomenout, že funkce writeToFile neočekává přítomnost výstupního souboru, a tak výstupní soubor VŽDY přepsán bez možné náhrady ztracených dat!