HTMLPrettyPrinter

*Realizat de echipa: **Wham!** (**Nedelcu Alexandru-Constantin** și **Săndulescu Ioan**)*

*Grupa: **141***

Link repository: https://github.com/alex-nedelcu04/141-ITBI-Wham-HTMLPrettyPrinter

ENUNȚ

Scrieți un program (script) shell care primește ca input un fișier HTML si îl formatează a.î. să reflecte vizual structura ierarhică și nivelul de imbricare al resurselor folosite prin indentarea corespunzatoare a tag-urilor si strigurilor.

SOLUȚIE

1. Furnizarea fișierului HTML ca input

Secvență de cod relevantă:

```
if [ -z "$1" ]; then # daca fisierul nu a fost oferit ca argument
    # loop pana la primirea unui fisier valid ca argument
    while true; do
        echo -n "Please enter the path to an HTML file: "
        read file
        if [ -f "$file" ]; then # daca este un fisier valid
            break
        else
            echo "Not a valid file. Try again"
        fi
    done
else # daca fisierul a fost oferit ca argument
    file="$1"
    if ! [ -f "$file" ]; then
        echo "Not a valid file"
        exit 1
    fi
fi
```

Pentru ca scriptului să i se furnizeze cu succes un fișier HTML, există două variante acceptate:

a. După executarea comenzii de rulare a scriptului:

Imediat ce scriptul începe să ruleze, acesta intră într-un loop până la primirea unui fișier valid. În caz contrar, afișează un mesaj de eroare și solicită un alt nume de fișier.

Exemplu:

```
alexn@AlexG14:~/FMI/ProiectITBI$ ./HTMLPrettyPrinter.sh
Please enter the path to an HTML file: invalid.html
Not a valid file. Try again
Please enter the path to an HTML file: abc
Not a valid file. Try again
Please enter the path to an HTML file: main00.html
Pretty-Printing succesful!
```

*(Mesajul 'Pretty-Printing successful!' nu este afișat de această secvență, ci doar după rularea completă și corectă a programului.)

b. După executarea comenzii de rulare a scriptului: Ca parte a comenzii de rulare a scriptului (dacă a fost oferit ca argument)

Dacă fișierul primit ca argument nu este valid, va fi afișat un mesaj de eroare, iar rularea programului va fi oprită.

Exemplu:

```
alexn@AlexG14:~/FMI/ProiectITBI$ ./HTMLPrettyPrinter.sh invalid.html
Not a valid file
alexn@AlexG14:~/FMI/ProiectITBI$ ./HTMLPrettyPrinter.sh abc
Not a valid file
alexn@AlexG14:~/FMI/ProiectITBI$ ./HTMLPrettyPrinter.sh test123
Not a valid file
alexn@AlexG14:~/FMI/ProiectITBI$ ./HTMLPrettyPrinter.sh main00.html
Pretty-Printing succesful!
```

*(Mesajul 'Pretty-Printing successful!' nu este afișat de această secvență, ci doar după rularea completă și corectă a programului.)

2. Prelucrarea fișierului pentru a ușura modificările ulterioare

a. Adăugarea unui newline la sfârșitul fișierului, în cazul în care nu există:

Secvență de cod relevantă:

```
# adaugare newline la sfarsitul fisierului in cazul in care lipseste
if [ "$(cat "$file" | tail -1 | wc -l)" -eq 0 ]; then
    echo >> "$file"
fi
```

Explicații:

- **tail -1** => selectează doar ultima linie a fișierului
- **wc -l** => returnează numărul de newline-uri de pe linia respective
- **daca valorea comenzii cat "$file" | tail -1 | wc -l este 0** => niciun newline => trebuie adăugat

b. Punerea fiecărui tag și a fiecarei secvențe text pe câte o linie:

Plasarea fiecărui tag și a fiecărei secvențe de text pe câte o linie.

Se va folosi fișierul auxiliar “formatted_tags” (creat în cadrul scriptului si șters înainte de se finaliza)

Secvență de cod relevantă:

```
# fiecare tag si fiecare secventa text va fi pusa pe cate o linie pentru a usura prelucrarea

# citire linii fisier
while IFS= read -r line; do
    # primul sed => adauga \n dupa <
    # al doilea sed => adauga \n dupa >
    # grep "\S" => selecteaza doar liniile care au elemente nenule (nu sunt goale)

    echo "$line" | sed 's/</\n</g'  | sed 's/>/>\n/g' | grep "\S"  >> formatted_tags
done < "$file"
```

Explicații:

- **sed 's/</\n</g'** => adaugă “\n” după ”<”
- **sed 's/>/>\n/g'** => adaugă “\n” după “>”
- **grep "\S"** => selectează doar liniile care au elemente nenule (nu sunt goale)

Exemplu:

Înainte de efectuarea secventei:

```
<!DOctype html> <html>

<hEaD>
    <title>Title</title>
</HEAD>
<bodY>
<h1>      My First Heading    </h1>   <p>   My first paragraph.</p>
</body>              </html>
```

După efectuarea secvenței:

```
<!DOctype html>
<html>
<hEaD>
<title>
Title
</title>
</HEAD>
<bodY>
<h1>
      My First Heading
</h1>
<p>
   My first paragraph.
</p>
</body>
</html>
```

c. Modificarea tag-urilor pentru a respecta convențiile de utilizare a majusculelor și a minusculelor.

Tag-urile în HTML nu sunt case-sensitive, astfel că nu va afecta corectitudinea dacă acestea sunt prelucrate pentru a fi complet cu litere mari sau complet cu litere mici, conform convențiilor. Se utlizează fișierul auxiliar “temp”.

Secvență de cod relevantă:

```
while IFS= read -r line; do
    tag_name=$(echo "$line" | grep -oP '(?<=<)[^<> ]+')
    upper_case_tag_name=$(echo "$tag_name" | tr '[:lower:]' '[:upper:]')
    if [ "$upper_case_tag_name" != "!DOCTYPE" ]; then
        tag_name=$(echo "$tag_name" | tr '[:upper:]' '[:lower:]')
        echo "$line" | sed -E "s|<[^ >]+|<$tag_name|g" >> temp
    else # daca primul cuvant al tag-ului este !DOCTYPE
        tag_name=$(echo "$upper_case_tag_name")
        second_word=$(echo "$line" | grep -oP '(?<=<)[^<>]+' | awk '{print $2}' | tr '[:upper:]' '[:lower:]')
        tag_name="$tag_name $second_word"
        echo "$line" | sed -E "s|<[^>]+|<$tag_name|g" >> temp
    fi

done < formatted_tags
cat temp > formatted_tags
rm temp
```

Explicații:

- **grep -oP '(?<=<)[^<>]+'** => extrage primul cuvant aflat între “<” și “>”
  ex: <<!DOctype html>> =>!DOctype
- **tr '[:lower:]' '[:upper:]''** => tag-ul devine scris cu majuscule
  ex: !DOctype => !DOCTYPE
- Daca tag-ul incepe cu !DOCTYPE => se prelucrează al doilea cuvânt pentru a îl face să fie format doar din minuscule
  - **grep -oP '(?<=<)[^<>]+'** – extrage toate cuvintele aflate între “<” și “>”
  - **awk '{print $2}'** – se extrage doar al doilea cuvânt
- **sed -E "s|<[^ >]+|<$tag_name|g"** - înlocuiește primul cuvânt cu varianta nouă doar cu majuscule / doar cu minuscule (în funcție de caz)
- **sed -E "s|<[^>]+|<$tag_name|g"** - înlocuiește intreaga secvență dintre “<” și “>”cu varianta nouă (!DOCTYPE cu majuscule, celălalt cuvânt cu minuscule)

Exemplu:

Înainte de efectuarea secventei:

```
<!DOctype HtMl>
<htML>
<hEaD>
<title>
Title
</title>
</HEAD>
<bodY>
<h1>
      My First Heading
</h1>
<p>
   My first paragraph.
</p>
</body>
</html>
```

După efectuarea secvenței:

```
<!DOCTYPE html>
<html>
<head>
<title>
Title
</title>
</head>
<body>
<h1>
      My First Heading
</h1>
<p>
   My first paragraph.
</p>
</body>
</html>
```

3. Verificare fișier HTML valid

Pentru a verifica validitatea fișierului HTML, se va simula modul de funcționare a unei stive. Conceptul utilizat este similar cu cel al verificării închiderii parantezelor.

În această stivă se vor adăuga tag-urile de început atunci când sunt întâlnite. La întâlnirea unui tag de sfârșit, se verifică dacă în vârful stivei se află tag-ul de început echivalent. Dacă acesta este în vârf, va fi eliminat. Altfel, se consideră că fișierul nu este valid și se va afișa un mesaj de eroare, iar rularea scriptului se va opri.

- Dacă stiva este goală la final => Fișier valid

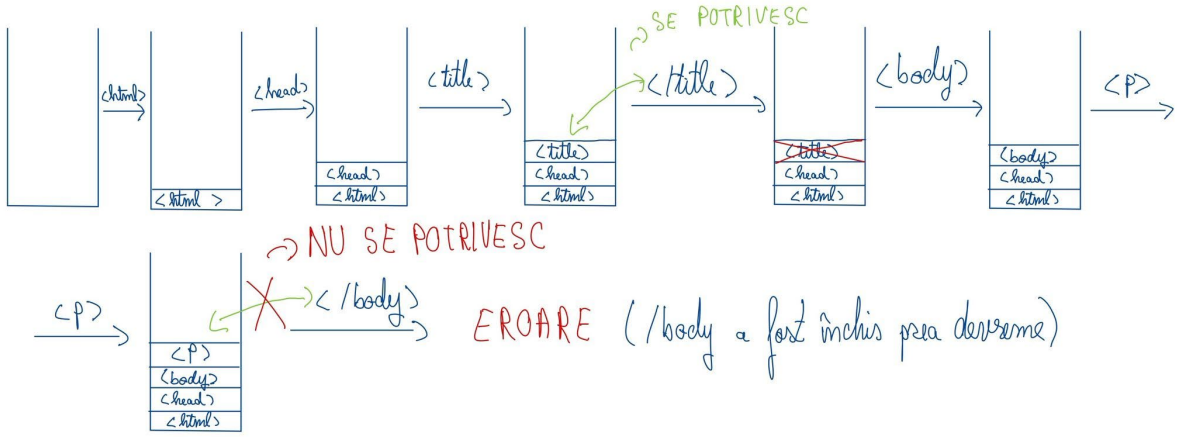
- Dacă stiva NU este goală => Fișier invalid

- Elementele care nu au si tag de start si tag de sfârșit sunt ignorate

Exemplu grafic:

-pentru input-ul

<!DOCTYPE html> <html> <head> <title>Title</title> </head> <body> <p> Paragraf </body> </html>



Output:

```
alexn@AlexG14:~/FMI/ProiectITBI$ ./HTMLPrettyPrinter.sh wrong.html
ERROR - Unexpected closure of tag type <body>;
```

Secvență de cod relevantă:

```
# se va folosi o stiva in care vor fi memorate tag-urile de start
# acestea vor fi eliminate in momentul in care se intalneste un tag de sfarist egal
# daca stack-ul nu este gol la final sau daca se intalneste alt tag de sfarsit fata
# de tag-ul de start din varful stive => fisier invalid


stack=()
```

```
while IFS= read -r line; do
    tag_text_only=$(echo "$line" | grep -oP '(?<=<)[^/<> ]+')

    if [ -n "$tag_text_only" ]; then # daca tag-ul este de start
        line="$tag_text_only"
       if [[ "$line" = "!DOCTYPE" || "$line" = "area" || "$line" = "base" || "$line" = "br" ||
        "$line" = "col" || "$line" = "embed" || "$line" = "hr" || "$line" = "img" ||
        "$line" = "input" || "$line" = "link" || "$line" = "meta" || "$line" = "source" ||
        "$line" = "track" || "$line" = "wbr" ]]; then
            continue
        fi
        stack+=("$line")
    else
        tag_text_only=$(echo "$line" | grep -oP '(?<=</)[^/<> ]+')
        if [ -n "$tag_text_only" ]; then  # daca tag-ul este de inchidere (si nu secveta de text)
            if [ "$tag_text_only" == "${stack[-1]}" ]; then
               unset 'stack[-1]'
            else
                echo "ERROR - Unexpected closure of tag type <$tag_text_only>;"
                rm formatted_tags
                exit 1
            fi
        fi
    fi
done < formatted_tags
```

```
if ! [ "${#stack[@]}" -eq 0 ]; then # daca stiva nu este goala => fisier invalid
    echo "ERROR - Number of tag opens and tag closes doesn't match - left to close: <${stack[@]}>;"
    rm formatted_tags
    exit 1
fi

# daca programul ajunge aici => fisierul este valid
```

4. Calculul indentării necesare pentru fiecare linie

Fiecare linie va fi prefixată cu numarul de tab-uri necesare indentării

Ex: <p> => 4~<p>

În cadrul procesului de determinare a numărului de indentări necesare pentru a alinia corect textul de pe un rând, este realizată diferențierea între cele 5 cazuri posibile pentru o linie de text aflată în fișierul „formatted_tags”:

a) <u>tag de început</u>

- crește valoarea curentă a indentării, asigurând separarea față de posibilele alte tag-uri deschise anterior

b) <u>tag de sfârșit</u>

- scade valoarea curentă a indentării, asigurând revenirea la poziționarea tag-ului de start anterior

c) <u>tag fără sfârșit</u>

- acest caz particular este adresat concomitent cu tag-urile de start, iar indentarea acestuia se realizează fără a modifca valoarea indentării curente

d) <u>text normal</u>

- textul este considerat echivalent ca indentare cu un tag fără sfârșit, astfel că nu este crescută indentarea la inserarea acesteia pe linie

e) <u>text poziționat în cadrul tag-ului “pre”</u>

- tag-ul „pre” impune o restricție suplimentară asupra indentării textului, anume că textul introdus de programator în interiorul zonei delimitate de tag-ul „pre” și indicatorul lui de final „/pre” își va păstra formatarea (la nivel de indentare, font etc.) marcată de programator. De aceea, este necesar să impunem ca, temporar, indentarea să fie setată la 0 până când întâlnim tag-ul de sfârșit „/pre”.

Inainte de efectuare secvenței:

```
<!DOCTYPE html>
<html>
<head>
<title>
Title
</title>
</head>
<body>
<h1>
     My First Heading
</h1>
<p>
   My first paragraph.
</p>
</body>
</html>
```

După efectuarea secvenței:

```
0~<!DOCTYPE html>
0~<html>
1~<head>
2~<title>
3~Title
2~</title>
1~</head>
1~<body>
2~<h1>
2~     My First Heading
2~</h1>
2~<p>
3~My first paragraph.
2~</p>
1~</body>
0~</html>
```

5. Aplicarea indentării necesare pentru fiecare linie + memorare rezultat final într-un nou fișier

```
# transformare prefix in tab-uri (indent = n => n tab-uri <=> 4*space-uri )
while IFS= read -r line; do
    indent=$(echo "$line" | grep -oP "^[0-9]+")
    indent=$((indent * 4))
    spaces=$(printf '%*s' "$indent")
    echo "$line" | sed  -E "s/^[0-9]+~/$spaces/g" >> tmp_file
done < formatted_tags

cat tmp_file > formatted_tags
rm tmp_file


file=$(basename "$file")
cat formatted_tags > "result_$file"
echo "Pretty-Printing succesful!"
rm formatted_tags
```

Explicații:

- **grep -oP "^[0-9]+"** => Selecteaza doar cifrele (fără ~)
- **spaces=$(printf '%*s' "$indent")** => Crează un șir de spații de dimenisune “indent”
- **sed -E "s/^[0-9]+~/$spaces/g"** => Înlocuiește șirul format din cifre și ~ cu șirul de spații create anterior

Inainte de efectuare secvenței:

```
0~<!DOCTYPE html>
0~<html>
1~<head>
2~<title>
3~Title
2~</title>
1~</head>
1~<body>
2~<h1>
2~    My First Heading
2~</h1>
2~<p>
3~My first paragraph.
2~</p>
1~</body>
0~</html>
```

După efectuarea secvenței:

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Title
        </title>
    </head>
    <body>
        <h1>
            My First Heading
        </h1>
        <p>
            My first paragraph.
        </p>
    </body>
</html>
```

DIFICULTĂȚI CONFRUNTATE

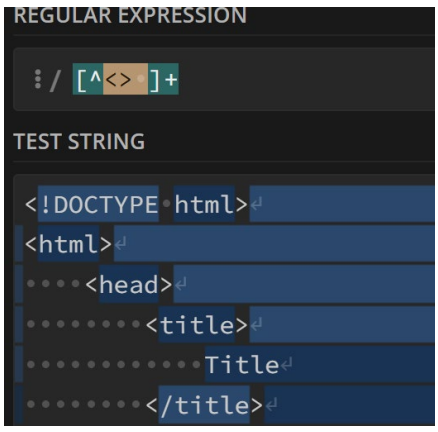
1. Selectarea textului din interiorul unui tag

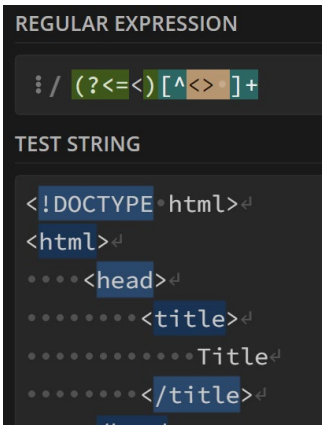
Utilizarea regex-ului “[^<>]+” nu returna rezultatul așteptat. Acesta selecta și spațiile goale de după un tag.

Rezolvare:

Utilizarea conceptului de **positive lookbehind** prin regex-ul “(?<=<)[^<>]+” .

(?<=<) cere să fie îndeplinita condiția de existență a unui “<” exact înaintea șirului selectat





2. Operația de indentare ducea la nerespectarea spațiilor necesare zonelor de text

Această problemă era observabilă în special în cazul tag-ului <pre>, care ține cont exact de numărul de spații aflate înaintea textului propriu-zis.

Spre exemplu, pentru:

```
<!DOctype html> <html> <body>
<PrE>
    aaaa
    aaaa
       aaaa
</pRe>
</body>            </html>
```

Rezultatul era:

```
<!DOCTYPE html>
<html>
    <body>
        <pre>
            aaaa
            aaaa
                aaaa
        </pre>
    </body>
</html>
```

Dar, acesta modifică numărul de spații pentru textul din pre, rezultatul final fiind diferit de așteptările inițiale.

Rezolvare: Tratarea specială a cazurilor legate de secvențele text pentru a asigura păstrarea spațiilor acolo unde este nevoie. (**4. Calculul indentării necesare pentru fiecare linie**). Rezultat corect:

```
<!DOCTYPE html>
<html>
    <body>
        <pre>
    aaaa
    aaaa
       aaaa
        </pre>
    </body>
</html>
```

REZULTATE EXPERIMENTALE

Input:

```
<!DOctype HtMl> <htML>

<hEaD>
    <title>Title</title>
</HEAD>
<bodY>
<h1>   My First Heading    </h1>   <p>   My first paragraph.</p>
</body>              </html>
```

Rezultatul din fișierul nou creat:

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Title
        </title>
    </head>
    <body>
        <h1>
            My First Heading
        </h1>
        <p>
            My first paragraph.
        </p>
    </body>
</html>
```

Input:

```
<!DOctype html> <html>

<hEaD>
    <title>Title</title>
</HEAD>
<bodY>
<h1>   My First Heading    </h1>   <p>   My first paragraph.</p>
<PrE>
    aaaa
    aaaa
       aaaa
</pRe>
<div>
    <input type="button">ahahahhahahaha
    <p>
<addreSS>ahahahahaha</addreSS></P>
</div>
<pre name="hahahaha"></pre>
<input type="button">
<pre name="ahahahah"></pre>
<pre name="fun"></pre>
</body>              </html>
```

Rezultatul din fișierul nou creat:

```
<!DOCTYPE html>
<html>
    <head>
        <title>
            Title
        </title>
    </head>
    <body>
        <h1>
            My First Heading
        </h1>
        <p>
            My first paragraph.
        </p>
        <pre>
    aaaa
    aaaa
        aaaa
        </pre>
        <div>
            <input type="button">
            ahahahhahahaha
            <p>
                <address>
                    ahahahahaha
                </address>
            </p>
        </div>
        <pre name="hahahaha">
        </pre>
        <input type="button">
        <pre name="ahahahah">
        </pre>
        <pre name="fun">
        </pre>
    </body>
</html>
```

Input:

<!DOCTYPE html><html lang="en"><head><meta charset="UTF-8"><meta name="viewport" content="width=device-width,initial-scale=1.0"><title>One-Line HTML</title></head><body><h1>Hello, World!</h1><script>console.log("This is a long one-liner HTML file.")</script></body></html>

Rezultatul din fișierul nou creat:

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <title>
            One-Line HTML
        </title>
    </head>
    <body>
        <h1>
            Hello, World!
        </h1>
        <script>
            console.log("This is a long one-liner HTML file.")
        </script>
    </body>
</html>
```