Documentație Proiect ITBI

Echipa: Nothing but Thieves

Studenți: Găbreanu Răzvan-George, Mocanu Ruxandra-Ioana, Săpunaru Maia

Introducere:

Problema aleasă:

6. XML (2-3 studenți) - Scrieți un program (script) shell care să parseze fișiere XML. Acesta trebuie să poată citi și scrie date din, respectiv, în format XML.

Se presupune că primim fișiere în format XML sau TXT. Trebuie să transformăm dintr-un format în celălalt, păstrând indentările corespunzătoare și funcționalitatea fișierului.

Programul de bază:

Odată ce rulăm programul, în terminal apare un meniu cu funcțiile dorite.

```
meniu_principal() {
        echo "Meniu"
        echo "1. XML --> TXT"
        echo "2. TXT --> XML"
        echo "3. Iesire"
        echo "Alege o optiune:"
}
```

```
while true; do
        meniu_principal
        read choice
        case $choice in
        1)
                xml_to_txt
                ;;
        2)
                txt_to_xml
                ;;
        3)
                echo "Ieșire"
                break
                ;;
        *)
                echo "Optiune gresita"
                ;;
        esac
done
```

În funcție de opțiunea aleasă, programul sare la subprogramul dorit. Este tratat cazul în care se alege opțiunea greșită.

```
xml_to_txt_validare() {
    printf "Introduceti numele fișierului XML (de intrare):\n"
    read input_file

    # Verificam daca fisierul de intrare exista
    if ! existenta_fisier "$input_file"; then
        return 1
    fi

    # Verificam extensia fisierului de intrare (trebuie sa fie XML)
    if ! validare_format "$input_file" "xml"; then
        return 1
    fi
```

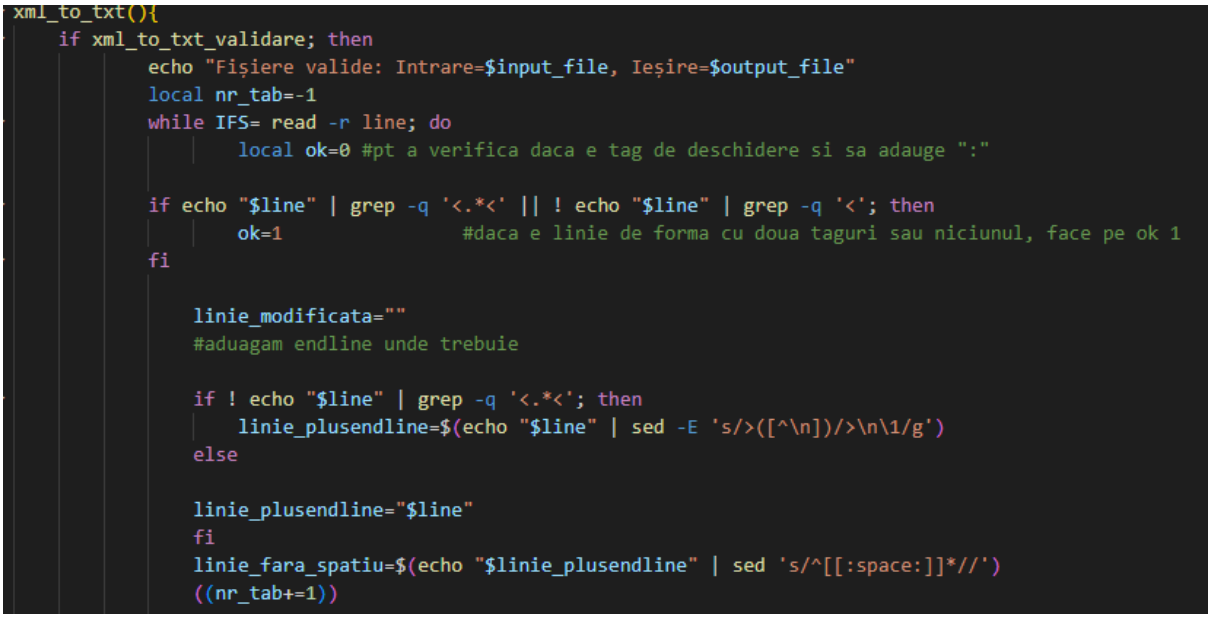
Odată ce o opțiune de parsare este aleasă, trebuie sa verificam existența și forma fișierului.

```
existenta_fisier() {
    local file=$1
    if [ ! -f "$file" ]; then
        echo "Eroare: Fisierul $file nu exista"
        return 1
    fi
    return 0
}

#Aici validam formatul fisierului

validare_format() {
    local file=$1
    local extensie=$2

    if [[ $file =~ \.$extensie$ ]]; then
        return 0
    else
        echo "Eroare: Fisierul trebuie sa aiba extensia .$extensie"
        return 1
    fi
}
```

Transformarea unui fișier din format XML în format TXT

Pentru transformarea unui fișier de tip XML într-un fișier de tip TXT, codul a fost scris într-o structură repetitivă de tip “while”, citind, pe rând, fiecare linie din fișier. Numărul de tab-uri care trebuie adăugate la final se va face utilizând variabila “nr_tab”, pe tot parcursul transformării.

O problemă pe care am întampinat-o ulterior a fost nevoia de a păstra într-o alta variabilă linia curentă nemodificată, pentru a verifica anumite condiții.



```
xml_to_txt(){
    if xml_to_txt_validare; then
            echo "Fișiere valide: Intrare=$input_file, Ieșire=$output_file"
            local nr_tab=-1
            while IFS= read -r line; do
                    local ok=0 #pt a verifica daca e tag de deschidere si sa adauge ":"

            if echo "$line" | grep -q '<.*<' || ! echo "$line" | grep -q '<'; then
                    ok=1                #daca e linie de forma cu doua taguri sau niciunul, face pe ok 1
            fi

                linie_modificata=""
                #aduagam endline unde trebuie

                if ! echo "$line" | grep -q '<.*<'; then
                    linie_plusendline=$(echo "$line" | sed -E 's/>([^\n])/>\n\1/g')
                else

                linie_plusendline="$line"
                fi
                linie_fara_spatiu=$(echo "$linie_plusendline" | sed 's/^[[:space:]]*//')
                ((nr_tab+=1))
```

Astfel, după cum se poate observa, variabila “line” ține minte șirul de caractere inițial, variabila “linie_fara_spatiu” va fi folosită pentru a modifica șirul pe parcurs. Sunt folosite, de exemplu, comenzile “echo” si “grep”, legate printr-un operator de redirecționare “|” (astfel rezultatul comenzii “echo $line” va fi transmisa către grep, iar comanda “ grep -q ‘<.*<’ “ caută șirul aflat intre caracterele ‘<’, ‘>’).

```
# elimina spatiile inutile
    line=$(echo "$line" | sed 's/^[ \t]*//;s/[ \t]*$//')

    #verifica daca linia contine formatul <tag>content</tag>
    if echo "$line" | grep -qE '^<[^>]+>.*<\/[^>]+>$'; then
        #extrage tag-ul dintre <>
        tag=$(echo "$line" | sed -E 's/^<([^>]+)>.*<\/[^>]+>$/\1/')

        #extrage conținutul dintre tag-uri
        content=$(echo "$line" | sed -E 's/^<[^>]+>(.*)<\/[^>]+>$/\1/')

        #adauga ':' dupa tag
        tag="${tag}:"

        #construieste linia formatată
        linie_fara_spatiu=$(printf '%s\n%s\n' \
            "$(printf '\t%.0s' $(seq 1 $nr_tab))$tag" \
            "$(printf '\t%.0s' $(seq 1 $((nr_tab + 1))))$content")
            ((ok = 2))
fi
```

Cazul pentru care atât tag-ul de deschidere, cât și tag-ul de închidere se află pe aceeași linie, împreună cu textul aflat între cele două tag-uri, este tratat separat, mai sus. Comanda “sed” (stream editor) este folosită pentru a modifica informații transmise prin pipe. (de exemplu, pe aceeași linie se află textul <name>John</name>).

Secvența de cod modifică linia curentă, adăugând atât endline (“/n”), dupa tag-ul de deschidere, cât și caracterul “:”. Codul extrage, de asemenea, textul dintre caracterele “<... >”

```
if [ "$ok" -ne 2 ]; then
        linie_modificata+="$(printf '%s%s\n' "$(printf '\t%.0s' $(seq 1 $nr_tab))" "$linie_fara_spatiu")"
    else
        linie_modificata="$linie_fara_spatiu"
    fi
    linie_modificata=$(echo "$linie_modificata" | sed 's/<//g')        # daca este tag de deschidere elimina < si pe urm linie de cod elimina >
    linie_modificata=$(echo "$linie_modificata" | sed 's/>//g')

    if [[ "$ok" -eq 0 ]]; then
        linie_modificata="${linie_modificata}:"      #adauga: daca e tag
 fi

if [[ "$line" == *\</* ]]; then
            ((nr_tab-=1))
    fi

    if [[ "$line" != "<"* ]]; then     #daca nu exista < la inceputul cuvantului
        ((nr_tab-=1))
fi

    echo "$linie_modificata" >> $output_file
```

În finalul secvenței de cod care modifică fiecare linie din fișierul XML, ne folosim de variabila “ok” (inițializată la începutul programului cu 0) ca să vedem în care dintre cazuri se află string-ul curent:

- dacă ok a rămas 0, este vorba de un tag de deschidere de forma <tag>, care va fi transformat în tag:
- dacă ok este 1, atunci este vorba de un text fără niciun tag.
- dacă ok este 2, este vorba de cazul discutat mai devreme, de o variabilă ce conține un string de forma <tag>name</tag>

Cum am modificat mai sus pentru cazul particular in care ok este 2, mai rămâne să indentăm restul cazurilor cu numărul necesar de taburi, iar, la final, să prelucreze iar valoarea variabilei “nr_tab”, folosindu-ne de variabila “line”, despre care am spus la început că păstrează o copie a liniei inițiale nemodificate, folosită pentru verificări.

Partea a 2-a

Transformarea unui fișier din format TXT în format XML

Pentru a transforma un fișier din format TXT în format XML, am ales să scriem un cod sursă care se folosește de stivă pentru a ține minte valoarea viitoarelor tag-uri (<tag></tag>) din fișierul XML și pentru a calcula indentările necesare scrierii unui fișier XML. Astfel, vom putea ține minte și accesa fiecare tag pentru a putea închide corespunzător toate structurile, lucru care va fi de folos în special în cazul structurilor “nested”, care trebuie inchise in ordine LIFO (Last In First Out), asemenea stivei.

```
txt_to_xml(){
     if txt_to_xml_validare; then
            echo "Fișiere valide: Intrare=$input_file, Ieșire=$output_file"

        #initializam aici stiva pentru a tine minte ordinea tag-urilor curente
        declare -a stack=()

        #ierarhia (prin stiva) incepe cu root
        stack+=("root")

        #aici facem citirea fisierului pe linii
        while IFS= read -r line; do
              #eliminam spatiile albe
              line_0=$(echo "$line" | sed 's/^[[:space:]]*//;s/[[:space:]]*$//')

              #daca linia este goala in fisier, sarim peste ea si mergem la urmatoarea linie
              if [[ -z "$line_0" ]]; then
                    continue
              fi

              #numaram tab-urile de la inceputul liniei pentru a determina indentarea
              indent_level=$(echo "$line" | sed 's/^\(\t*\).*/\1/' | wc -c)

              #calculam adancimea locului in care suntem acum cu ajutorul stivei si al indentarii
              depth=$((indent_level - 1))  # Fiecare tab reprezintă un nivel
```

Mai sus se poate observa atât declararea stivei, cât și începutul structurii repetitive prin care vom face citirea (linie cu linie) din fișierul TXT. Motivul pentru care am luat decizia să folosim o stivă este reprezentat de faptul că ne-a fost foarte dificil să ținem minte corespunzător unde începeau anumite structuri și unde era necesar să fie închise, cât și numele structurilor (tag-urilor) care mai aveau nevoie să fie închise, iar cazul care ne-a adus cele mai multe probleme a fost cel cu structuri imbricate (nested).

Este o situație diferită decât cea precedentă, când transformam un fișier XML într-un fișier TXT, pentru că în cazul fișierului TXT nu mai era nevoie să închidem fiecare tag.

Exemplu:
<tag> tag:
exemplu s-ar transforma doar în: exemplu
</tag>

În acest caz, ne-a fost imposibil să găsim o variantă mai simplă decât stiva pentru a ține minte și tag-urile precedente, întrucat prima noastră încercare de rezolvare conținea doar variabile care țineau minte numele tag-ului/tag-urilor precedente, dar devenea problematică în momentul în care adăugăm mai multe tag-uri în interiorul altui tag.

```
#numaram tab-urile de la inceputul liniei pentru a determina indentarea
indent_level=$(echo "$line" | sed 's/^\(\t*\).*/\1/' | wc -c)

#calculam adancimea locului in care suntem acum cu ajutorul stivei si al indentarii
depth=$((indent_level - 1))  # Fiecare tab reprezintă un nivel

#ajustam stiva in functie de adancime
while (( ${#stack[@]} > depth + 1 )); do
      # Închidem ultimul tag
      last_tag=${stack[-1]}
      echo "$(printf '%*s' $((2 * (depth))))</$last_tag>" >> "$output_file"
      stack=("${stack[@]::${#stack[@]}-1}")  # Scoatem ultimul element din stivă
done

#verificam daca linia se termina cu ":" (adica este un tag)
if [[ "$line_0" == *:* ]]; then
      #este un tag, eliminam ":"
      tag_name=$(echo "$line_0" | cut -d':' -f1 | tr '[:upper:]' '[:lower:]')
  #aici verificam daca nu cumva e root, pe care il prelucram separat
      if [[ "${stack[-1]}" == "root" && ${#stack[@]} -eq 1 ]]; then
            echo "<$tag_name>" >> "$output_file"
      else
             echo "$(printf '\t%.0s' $(seq 1 $depth))<$tag_name>" >> "$output_file"
      fi
  #aici adaugam in stiva
      stack+=("$tag_name")
```

În secvența de cod de mai sus se poate observa cum calculăm indentarea necesară fiecărei linii în funcție de numărul de tab-uri și felul în care gestionăm structurile de tip tag-uri nested/imbricate: ne folosim de dimensiunea stivei, pe care o comparăm cu

adâncimea liniei curente (adică o variabilă prin care ne dăm seama cât de “adânc imbricată” este o porțiune din structură, adică cat de departe de marginea din stânga, este linia). De asemenea, ne ocupăm și de închiderea tag-urilor rămase deschise, cât și de ștergerea lor din stivă, până în locul în care ne aflăm la tag-uri de “aceeași adâncime”.

```
        else
                #este doar o valoare (adica continutul tag-ului)
                current_tag=${stack[-1]}
                echo "$(printf '\t%.0s' $(seq 1 $depth))$line_0" >> "$output_file"

                #inchidem tag-ul după valoare
                echo "$(printf '\t%.0s' $(seq 2 $depth))</$current_tag>" >> "$output_file"

                stack=("${stack[@]::${#stack[@]}-1}")  # scoatem ultimul element din stivă
        fi
done < "$input_file"
```

În secvența de mai sus, scriem în fișier doar variabila din interiorul tag-ului, pe care nu mai e nevoie să o adăugăm în stiva, întrucât nu ne vom mai folosi de ea.

```
#inchidem toate tag-urile ramase deschise
while [[ ${#stack[@]} -gt 1 ]]; do
      last_tag=${stack[-1]}
      #aici inchidem orice este deschis, mai putin root, care este prelucrat separat
      if [[ "$last_tag" == "root" ]]; then
            echo "</$last_tag>" >> "$output_file"
      else
            echo "$(printf '\t%.0s' $(seq 1 $(( ${#stack[@]} - 2 ))))</$last_tag>" >> "$output_file"
      fi
      stack=("${stack[@]::${#stack[@]}-1}")  # scoatem ultimul element din stiva
done
```

Finalul secvenței de cod este reprezentat de închiderea tuturor tag-urilor rămase deschise și de ștergerea lor din stiva, pentru a putea afla când am închis toate structurile. Tag-ul root este tratat separat, pentru că am avut o problemă cu indentarea lui, așa ca am ales sa îl tratăm separat, printr-un if, întrucât este o excepție care apare o singură data în tot programul.

Exemple de funcționare

Rezultate Experimentale:

```
<root>
    <person>
        <name>John</name>
        <age>30</age>
    </person>
    <city>
        New York
    </city>
</root>
```

Primul fișier de intrare, cu extensia .xml, în partea stângă.

După parsare apare al doilea fișier, cu extensia .txt, în partea dreaptă.

```
root:
        person:
                name:
                        John
                age:
                        30
        city:
                New York
```

```
<root>
  <person>
    <name>
      John
    </name>
    <age>
      30
    </age>
  </person>
  <city>
    New York
  </city>
```

Al treilea fișier, din partea stângă, este rezultatul parsării fișierului 2 înapoi în format .xml.

Al patrulea fișier, cu extensia .txt, din partea dreaptă este pentru a arăta exemplul invers.

```
Root:
        Person:
                Name:
                        Surname:
                                Smith
                        FirstName:
                                John
                Age:
                        30
        State:
                City:
                        New York
                County:
                        Manhattan
```

```
<root>
  <person>
    <name>
      <surname>
        Smith
      </surname>
      <firstname>
        John
      </firstname>
    </name>
    <age>
      30
    </age>
  </person>
  <state>
    <city>
      New York
    </city>
    <county>
      Manhattan
    </county>
  </state>
</root>
```

Al cincilea fișier, cel din stânga, este rezultatul parsării fișierului 4.

Al șaselea fișier, din dreapta, este rezultatul parsării inverse.

```
root:
        person:
                name:
                        surname:
                                Smith
                        firstname:
                                John
                age:
                        30
        state:
                city:
                        New York
                county:
                        Manhattan
```