



Limbaje Formale si Translatoare

Proiect Laborator

Name: Barbu Bogdan, Maracine Stefania

Group:30233

Email: bogdan.barbu.dev@gmail.com

maracine_stefania@yahoo.com



Tema aleasă

Pentru acest proiect am ales sa cream un mini framework web care faciliteaza scrierea codului HTML si JavaScript. Ca si exemplu am creat o simpla pagina web cu un counter care poate fi incrementat, resetat, iar la atingerea valorii 10 butonul de increment dispare.

In imaginile ce urmeaza se poate vedea diferenta de content in fisierul de input si cel de output. In prima imagine, codul de input este alcatuit din doua parti, prima este un fel de cod HTML, dar imbunatatit, care contine elemente speciale cum ar fi conditia "if" sau variabile, iar a doua este alcatuita din codul propriu zis care va fi transpus in cod JavaScript, codul ce va contine funtiile de incrementare sau de resetare a counter-ului. In cea de a doua imagine este output-ul programului, mult mai mare ca si volum de cod fata de input.

```
input.html > body
1 <body>
2   <div class="box">
3     <h1><text>Counter:</text>@value</h1>
4     <button class="box" if=[value < 10] action="Increment"><text>Increment</text></button>
5     <button action="Reset" ><text>Reset</text></button>
6   </div>
7 </body>
8
9 CODE_START
10
11 declare value = 0
12
13 Increment {
14   value = value + 1
15 }
16
17 Reset {
18   value = 0
19 }
20
21 CODE_END
```

Figure 1: Codul de input

```
out.html > html > head
1 <!--A parsat ok html-->
2 <!--A parsat ok codul-->
3 <!DOCTYPE html>
4 <html lang="en">
5 <head> <meta charset="UTF-8">
6   <meta http-equiv="X-UA-Compatible" content="IE=edge">
7   <meta name="viewport" content="width=device-width, initial-scale=1.0">
8   <title>Framework example</title>
9 </head>
10 <body>
11
12   <div class="box">
13     <h1>
14       Counter:
15       <span class="value-binding"></span>
16     </h1>
17     <button class="box cond-with-id0" onclick="Increment()">
18       Increment
19     </button>
20     <button onclick="Reset()">
21       Reset
22     </button>
23   </div>
24 </body>
```

Figure 2.1: HTML

```
out.html > html > script
26 <script>
27 const conditions = [
28   {
29     condition: () => getVar('value') < 10,
30     element: document.querySelector(".cond-with-id0")
31   },
32 ]
33 const variables = {
34   'value' : {
35     value: 0,
36     elements: document.querySelectorAll('.value-binding')
37   },
38 }
39
40 const updateUI = (varName, newValue) => {
41   variables[varName].elements.forEach(element=>element.innerHTML = newValue);
42   conditions.forEach((cond, i)=>{
43     if(cond.condition()){
44       cond.element.style.display = '';
45     }
46     else{
47       cond.element.style.display = 'none';
48     }
49   });
50 }
51 const updateVar = (varName, value) => {
52   variables[varName].value = value;
53   updateUI(varName, value);
54 }
55 const getVar = (varName) => {
56   return variables[varName].value;
57 }
58
59 const Increment = () => {
60   updateVar('value', getVar('value') + 1)
61 }
62 const Reset = () => {
63   updateVar('value', 0)
64 }
65 </script>
```

Figure 2.2: JavaScript

Figure 2: Codul de output

Explicații logică și cod

LEX

Pentru partea de Lex ne-am definit cateva regex-uri specifice HTML, dar si cateva specifice framework-ului creat de noi. Avem regexuri pentru variabile, numere intregi si operatii.

```
1  %{
2  #include "y.tab.h"
3  %}
4
5  %%
6
7  [<\/\[\]=\{\}]+ return *yytext;
8  body return BODY;
9  div return DIV;
10 h1 return H1;
11 span return SPAN;
12 button return BUTTON;
13 declare return DECLARE;
14 class return CLASS;
15 id return ID;
16 if return IF;
17 action return ACTION;
18
19
20 CODE_START return CODE_START;
21 CODE_END return CODE_END;
22
23 [0-9]+ {
24     yyval.integer = atoi(yytext);
25     return INTEGER;
26 }
27 @[a-z][_a-zA-Z0-9]* {
28     yyval.varname = strdup(yytext);
29     return BINDING;
30 }
31 [a-z][_a-zA-Z0-9]* {
32     yyval.varname = strdup(yytext);
33     return VARIABLE;
34 }
35 [A-Z][_a-zA-Z0-9]* {
36     yyval.funcname = strdup(yytext);
37     return FUNCNAME;
38 }
39
40 ="[^"]*" {
41     yyval.value = strdup(yytext);
42     return VALUE;
43 }
44 \<text\>[a-zA-Z0-9 \.!:;_-]*\</text\> {
45     yyval.string = strdup(yytext);
46     return STRING;
47 }
48 [ \t\n] ;
```

Figure 3: LEX

YACC

In partea de Yacc ne-am definit structuri, arborii de parsare pentru cele doua tipuri de cod, functii de afisare si alte functii ajutatoare. In acest capitol vom prezenta pe scurt ideile importante si modul de functionare a framework-ului.

Pentru partea de cod de input de HTML, acesta suporta si traduce tag-uri specifice HTML cum ar fi h1, div, text, button, class, span, dar avem si imbunatatiri. Elementul "onclick" din HTML nu exista in framework-ul nostru, este reprezentat de elementul "action". Fiecare element declarat poate sa aiba clasa, id, actiune si conditie.

Prima imbunatatire ar fii variabilele. Putem declara variabile prin utilizarea adnotarii @ in fata unui nume de variabila care sa inceapa cu litera mica. Putem

avea numeroare variabile, astfel incat in codul de output de JavaScript vom avea o lista cu variabilele create, continand numele si valoarea acestora.

Alta imbunatatire ar fi conditiile, cum ar fi "if". Structura este urmatoarea: if = [expresie]. Asemănător cu variabilele, in codul de Javascript tradus o sa avem o lista de conditii.

In codul de Yacc avem un contor pentru numarul de variabile create si un contor pentru numarul de conditii create astfel incat cand se realizeaza tradurecea sa putem crea listele si sa putem numi conditiile. Pentru a accesa elementul specific unei conditii ii vom adauga o clasa cu numele "cond-with-id" concatenat cu valoarea counter-ului, pe cand la variabile vom adauga clasa numele_variabilei concatenat cu "-binding".

HTML Code

```
<body>
  <div class="box">
    <h1><text>Counter:</text>@value</h1>
    <button class="box" if=[value < 10]
      action="Increment"><text>Increment</text></button>
    <button action="Reset" ><text>Reset</text></button>
  </div>
</body>
```

Pentru partea de cod din input inceputul este marcat de "CODE_START", iar sfarsitul de "CODE_END". In interior se pot declara variabile si functii si se pot executa expresii.

Pentru a declara o variabila este necesara utilizarea cuvintului cheie "declare" urmat de numele variabilei si optional "=" plus valoarea dorita.

Pentru a declara o functie este necesar sa denumim functia respectiva, numele acesteia va incepe cu litera mare pentru a o diferentia de variabile. Se deschid acoladele, iar in interiorul acestora putem avea declarare de variabile, asignari a unor variabile sau chiar alte functii.

Principala structura de date a codului este:

```
typedef struct _stmt_node{
    union _statement stmt;
    struct _stmt_node *next;
}_stmt_node;
```

Figure 4: Stmt_node Structure

Elementul next pointeaza la statement-ul succesori celui actual. Statement-ul este reprezentat de un union alcatuit din urmatoarele 3 structuri:

```
typedef union _statement{  
    stmtEnum type;  
    struct _assignment assignment;  
    struct _varDeclarationStruct varDeclaration;  
    struct _funcDeclarationStruct funcDeclaration;  
}statement;
```

- o asignare, de exemplu: **value = value + 1**

```
typedef struct _assignment{  
    stmtEnum type;  
    char* varname;  
    struct exp *expression;  
}assignment;
```

- o declarare de variabila, de exemplu: **declare value = 0**

```
typedef struct _varDeclarationStruct{  
    stmtEnum type;  
    char* varname;  
    struct exp *expression;  
}varDeclarationStruct;
```

- o declarare de functie, de exemplu: **Reset {value = 0}**

```
typedef struct _funcDeclarationStruct{  
    stmtEnum type;  
    char* funcname;  
    struct _stmt_node *statements;  
}funcDeclarationStruct;
```