Department of Computer Science
Technical University of Cluj-Napoca

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 fii 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 fii conditia "if" sau variabile, iar a doua este alcatuita din codul propriu zis care va fii 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

```
<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>

CODE_START

declare value = 0

Increment {
    value = value + 1
}

Reset {
    value = 0
}

CODE_END
```

Figure 1: Codul de input

out.html > html > head

```
<!--A parsat ok html-ul-->
<!--A parsat ok codul-->
<!DOCTYPE html>
<html lang="en">
<head>    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Framework example</title>
</head>
<body>

 <div class="box">
    <h1>
        Counter:
        <span class="value-binding"></span>
    </h1>
    <button class="box cond-with-id0" onclick="Increment()">
        Increment
    </button>
    <button onclick="Reset()">
        Reset
    </button>
 </div>
</body>
```

Figure 2.1: HTML

out.html > html > script

```
<script>
const conditions = [
{
   condition: () => getVar('value') < 10,
   element: document.querySelector(".cond-with-id0")
},
]
const variables = {
'value' : {
    value: 0,
    elements: document.querySelectorAll('.value-binding')
},
}

const updateUI = (varName, newValue) => {
    variables[varName].elements.forEach(element=>element.innerHTML = newValue);
    conditions.forEach((cond, i)=>{
        if(cond.condition()){
            cond.element.style.display = '';
        }
        else{
            cond.element.style.display = 'none';
        }
    });
}
const updateVar = (varName, value) => {
    variables[varName].value = value;
    updateUI(varName, value);
}
const getVar = (varName) => {
    return variables[varName].value;
}

 const Increment = () => {
    updateVar('value',getVar('value') + 1)
 }
 const Reset = () => {
    updateVar('value',0)
 }
</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.

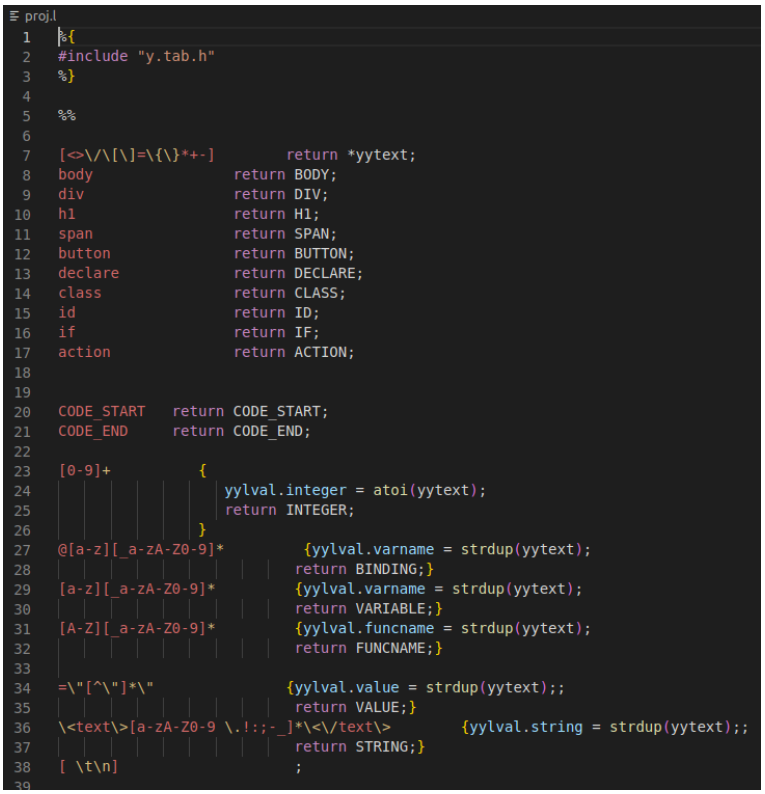

Figure 3: LEX

YACC

In partea de Yacc ne-am definit structuri, arborii de parsare pentru cele doua tipuri de cod, functii de afisare si alte funtii 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 fii 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 fii conditiile, cum ar fii "if". Structura este urmatoarea: if = [expresie]. Asemanator 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 traducerea 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.

Penru a declara o variabila este necesara utilizarea cuvantului 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 succesor celui actual. Statement-ul este reprezentat de un union alcatuit din urmatoarele 3 structuri:

```
typedef union _statement{
    stmtEnum type;
    struct _assignation assignation;
    struct _varDeclarationStruct varDeclaration;
    struct _funcDeclarationStruct funcDeclaration;
}statement;
```

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

```
typedef struct _assignation{
    stmtEnum type;
    char* varname;
    struct exp *expression;
}assignation;
```

- 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;
```

Link pentru GitHub

Click This