Tema 3
Generare de cod

CPL

Cuprins

1 Obiective

Obiectivele temei sunt următoarele:

- **Generarea de cod** MIPS pornind de la programe Cool.
- Utilizarea opțională a unui **instrument** dedicat pentru generarea de cod, *StringTemplate*.

2 Descriere

Tema abordează ultima etapă a construcției compilatorului pentru limbajul Cool, generarea de cod. Aceasta pornește de la reprezentarea intermediară elaborată în etapa anterioară, de analiză semantică, în forma unui arbore de derivare sau a unui arbore de sintaxă abstractă, adnotat cu simboluri și tipuri.

Rezultatul acestei etape îl constituie un **program MIPS** echivalent cu programului Cool inițial. Acesta va putea fi rulat în simulatoarele *QtSpim* sau *spim*.

Tema de față va primi ca parametri în linia de comandă numele unuia sau mai multor fișiere conținând programe Cool, și va tipări la *standard output* **programul MIPS** dorit. Programele de test vor fi **corecte lexical, sintactic și semantic**!

Punctul de plecare îl constituie **sistemul de execuție** Cool, sub forma unui fișier *exception handler*, `trap.handler.nogc`, încărcat în simulator. Printre altele, acesta conține mecanisme de **alocare dinamică** a memoriei, precum și implementarea **metodelor predefinite**, din clasele `Object`, `IO` și `String`, pe care le veți invoca din codul generat de voi.

Mai jos, funcționalitatea este exemplificată pornind de la cel mai simplu program Cool:

```
1  class Main {
2      main() : Object { 0 };
3  };
```

Un posibil program MIPS echivalent este ilustrat începând cu pagina următoare. În continuare, este prezentată **structura de ansamblu** a acestuia, urmând ca detaliile să fie furnizate în secțiunea 3.

- Ca în orice program MIPS, se observă cele două **zone de memorie**, `data` (linia 1), respectiv `text` (linia 186).
- Zona `data` debutează cu o serie de definiții **globale** (liniile 3–11), utilizând directiva `.globl`, care sunt vizibile și în alte programe MIPS încărcate concomitent. În particular, este vorba despre **sistemul de execuție** menționat mai sus, care utilizează anumite etichete generate de voi.
- Cele trei etichete, `_int_tag`, `_string_tag` și `_bool_tag` (liniile 12–17) precizează **etichetele** celor trei clase predefinite, fiind utilizate de sistemul de execuție pentru implementarea testului de **egalitate** între valori predefinite.
- În liniile 18–66, sunt definiți toți **literalii șir de caractere** din program, incluzând **numele claselor**, și eventual ale **fișierelor**, în anumite cazuri. Reprezentarea acestor literali este discutată ulterior.
- În liniile 67–91, sunt definiți toți **literalii întregi** din program. Reprezentarea acestora este discutată ulterior.
- În liniile 92–101, sunt definiți cei doi **literalii booleeni**. Reprezentarea acestora este discutată ulterior.
- Liniile 102–108 ilustrează **tabelul numelor de clase** (`class_nameTab`), necesar pentru afișarea anumitor mesaje de eroare de către sistemul de execuție. **Atenție!** Tabelul este indexat de **eticheta** clasei, reprezentată printr-un număr întreg.

```
	.data
	.align	2
	.globl	class_nameTab
	.globl	Int_protObj
	.globl	String_protObj
	.globl	bool_const0
	.globl	bool_const1
	.globl	Main_protObj
	.globl	_int_tag
	.globl	_string_tag
	.globl	_bool_tag
_int_tag:
	.word	2
_string_tag:
	.word	3
_bool_tag:
	.word	4
str_const0:
	.word	3
	.word	5
	.word	String_dispTab
	.word	int_const0
	.asciiz	""
	.align	2
str_const1:
	.word	3
	.word	6
	.word	String_dispTab
	.word	int_const1
	.asciiz	"Object"
	.align	2
str_const2:
	.word	3
	.word	5
	.word	String_dispTab
	.word	int_const2
	.asciiz	"IO"
	.align	2
str_const3:
	.word	3
	.word	5
	.word	String_dispTab
	.word	int_const3
	.asciiz	"Int"
	.align	2
str_const4:
	.word	3
	.word	6
	.word	String_dispTab
	.word	int_const1
	.asciiz	"String"
	.align	2
str_const5:
	.word	3
	.word	6
	.word	String_dispTab
	.word	int_const4
	.asciiz	"Bool"
	.align	2
str_const6:
	.word	3
	.word	6
	.word	String_dispTab
	.word	int_const4
	.asciiz	"Main"
	.align	2
int_const0:
	.word	2
	.word	4
	.word	Int_dispTab
	.word	0
int_const1:
	.word	2
	.word	4
	.word	Int_dispTab
	.word	6
int_const2:
	.word	2
	.word	4
	.word	Int_dispTab
	.word	2
int_const3:
	.word	2
	.word	4
	.word	Int_dispTab
	.word	3
int_const4:
	.word	2
	.word	4
	.word	Int_dispTab
	.word	4
bool_const0:
	.word	4
	.word	4
	.word	Bool_dispTab
	.word	0
bool_const1:
	.word	4
	.word	4
	.word	Bool_dispTab
	.word	1
class_nameTab:
	.word	str_const1
	.word	str_const2
	.word	str_const3
	.word	str_const4
	.word	str_const5
	.word	str_const6
```

```
class_objTab:
    .word   Object_protObj
    .word   Object_init
    .word   IO_protObj
    .word   IO_init
    .word   Int_protObj
    .word   Int_init
    .word   String_protObj
    .word   String_init
    .word   Bool_protObj
    .word   Bool_init
    .word   Main_protObj
    .word   Main_init
Object_protObj:
    .word   0
    .word   3
    .word   Object_dispTab
IO_protObj:
    .word   1
    .word   3
    .word   IO_dispTab
Int_protObj:
    .word   2
    .word   4
    .word   Int_dispTab
    .word   0
String_protObj:
    .word   3
    .word   5
    .word   String_dispTab
    .word   int_const0
    .asciiz ""
    .align  2
Bool_protObj:
    .word   4
    .word   4
    .word   Bool_dispTab
    .word   0
Main_protObj:
    .word   5
    .word   3
    .word   Main_dispTab
Object_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
IO_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   IO.out_string
    .word   IO.out_int
    .word   IO.in_string
    .word   IO.in_int
Int_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
String_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   String.length
    .word   String.concat
    .word   String.substr
Bool_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
Main_dispTab:
    .word   Object.abort
    .word   Object.type_name
    .word   Object.copy
    .word   Main.main
    .globl  heap_start
heap_start:
    .word   0
    .text
    .globl  Int_init
    .globl  String_init
    .globl  Bool_init
    .globl  Main_init
    .globl  Main.main
Object_init:
    addiu   $sp $sp -12
    sw      $fp 12($sp)
    sw      $s0 8($sp)
    sw      $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    move    $a0 $s0
    lw      $fp 12($sp)
    lw      $s0 8($sp)
    lw      $ra 4($sp)
    addiu   $sp $sp 12
    jr      $ra
IO_init:
    addiu   $sp $sp -12
    sw      $fp 12($sp)
    sw      $s0 8($sp)
    sw      $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    jal     Object_init
    move    $a0 $s0
    lw      $fp 12($sp)
    lw      $s0 8($sp)
    lw      $ra 4($sp)
```

```
    addiu   $sp $sp 12
    jr      $ra
Int_init:
    addiu   $sp $sp -12
    sw      $fp 12($sp)
    sw      $s0 8($sp)
    sw      $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    jal     Object_init
    move    $a0 $s0
    lw      $fp 12($sp)
    lw      $s0 8($sp)
    lw      $ra 4($sp)
    addiu   $sp $sp 12
    jr      $ra
String_init:
    addiu   $sp $sp -12
    sw      $fp 12($sp)
    sw      $s0 8($sp)
    sw      $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    jal     Object_init
    move    $a0 $s0
    lw      $fp 12($sp)
    lw      $s0 8($sp)
    lw      $ra 4($sp)
    addiu   $sp $sp 12
    jr      $ra
Bool_init:
    addiu   $sp $sp -12
    sw      $fp 12($sp)
    sw      $s0 8($sp)
    sw      $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    jal     Object_init
    move    $a0 $s0
    lw      $fp 12($sp)
    lw      $s0 8($sp)
    lw      $ra 4($sp)
    addiu   $sp $sp 12
    jr      $ra
Main_init:
    addiu   $sp $sp -12
    sw      $fp 12($sp)
    sw      $s0 8($sp)
    sw      $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    jal     Object_init
    move    $a0 $s0
    lw      $fp 12($sp)
    lw      $s0 8($sp)
    lw      $ra 4($sp)
    addiu   $sp $sp 12
    jr      $ra
Main.main:
    addiu   $sp $sp -12
    sw      $fp 12($sp)
    sw      $s0 8($sp)
    sw      $ra 4($sp)
    addiu   $fp $sp 4
    move    $s0 $a0
    la      $a0 int_const0
    lw      $fp 12($sp)
    lw      $s0 8($sp)
    lw      $ra 4($sp)
    addiu   $sp $sp 12
    jr      $ra
```

- Liniile 109–121 ilustrează **tabelul prototipurilor și al rutinelor de inițializare** ale instanțelor claselor (`class_objTab`), discutate mai jos. Tabelul este necesar pentru construcția `new SELF_TYPE`, care alocă un nou obiect cu tipul dinamic al lui `self`. **Atenție!** Tabelul este indexat în funcție de **eticheta** clasei, reprezentată printr-un număr întreg.
- Liniile 122–150 conțin **prototipurile instanțelor** fiecărei clase din program (`<class>_protObj`), atât predefinite, cât și definite de programator. Prototipurile respectă structura discutată la curs.
- Liniile 151–182 surprind **tabelele de metode** aferente fiecărei clase din program (*dispatch table*, `<class>_dispTab`), ca secvență de etichete din zona `text`, unde se găsesc implementările acestor metode.
- Liniile 184–185 marchează finalul zonei statice și începutul zonei de ***heap*** (`heap_start`), în conformitate cu cerințele sistemului de execuție.
- Zona `text` debutează în linia 186 cu câteva definiții **globale**, solicitate de sistemul de execuție.
- Începând cu linia 192, sunt definite **rutinele de inițializare** a instanțelor fiecărei clase (`<class>_init`), și **metodele** din fiecare clasă, cu numele `<class>.<method>`.

Sistemul de execuție realizează următorii pași la **lansarea** unui program:

- Este instanțiată **clasa** `Main`. Acest lucru se realizează copiind pe *heap* obiectul prototip `Main_protObj`.
- Este apelată **rutina de inițializare** `Main_init` pentru această nouă instanță.
- Este apelată **metoda** `Main.main`, adresa instanței (`self`) fiind în registrul `$a0`, iar adresa de revenire, în registrul `$ra`.

3 Cerințe

În continuare, sunt prezentate particularitățile de reprezentare și implementare pentru anumite construcții de limbaj. Utilizați **semantica operațională** a limbajului Cool (secțiunea 13 a manualului) pentru a înțelege ce presupune evaluarea fiecărei expresii. Pentru cazuri concrete, consultați testele (vezi secțiunea 4).

Atenție! NU este necesar să generați cod identic cu cel din teste. Acesta va fi verificat în raport cu funcționalitatea îndeplinită.

3.1 Definiții de clase

Pentru fiecare definiție declasă, predefinită sau definită de programator, este necesară definirea unui **obiect prototip** (`<class>_protObj`) și a unui **tabel de metode** (`<class>_dispTab`) în zona `data`, precum și a unei **rutine de inițializare** (`<class>_init`) în zona `text`.

Câmp	*Offset*
Etichetă clasă	0
Dimensiune în cuvinte	4
Dispatch pointer	8
Atribut 1	12
Atribut 2	16
⋮	⋮

Figura 1: Reprezentarea obiectelor în memorie

Reprezentarea în memorie a **obiectelor prototip** respectă convențiile discutate în curs, pe slide-urile 303–306, și este reluată în fig. 1. **Atenție!** Atributele din reprezentarea unui obiect le includ atât pe cele definite în clasa curentă, cât și pe toate cele de pe întregul lanț de **moștenire**! Atributele vor fi inițializate la **valorile implicite** în funcție de tip (vezi slide-ul 332).

Tabelele de metode respectă convențiile de pe slide-urile 307–308, și cuprind toate metodele accesibile unui obiect, de pe întregul lanț de **moștenire**. Spre exemplu, `IO_dispTab` include atât metodele specifice clasei `IO`, cât și pe cele ale clasei `Object`.

Rutinele de inițializare vizează **atributele** instanței curente. De asemenea, este necesară inițializarea atributelor moștenite, printr-un apel la rutina de inițializare a **superclasei**. Spre exemplu, `IO_init` invocă `Object_init`. Acestea se comportă ca niște metode, care sunt discutate pe larg în secțiunea 3.4. Ele primesc adresa obiectului curent în registrul `$a0`, unde trebuie să o și întoarcă.

3.2 Valori predefinite

Instanțele clasei **`Int`** conțin drept unic atribut valoarea întreagă MIPS. Spre exemplu, `int_const2` are valoarea întreagă `2`.

Instanțele clasei **`String`** conțin două atribute: adresa unui obiect `Int` cu lungimea șirului, urmat de șirul propriu-zis. **Atenție! Dimensiunea** în cuvinte a unui obiect `String` va depinde de **lungimea** șirului! Spre exemplu `str_const2`, redând șirul `"IO"`, are lungimea descrisă de `int_const2`, iar câmpul de dimensiune este `5`: 3 cuvinte pentru antet, 1 cuvânt pentru lungime, și 1 cuvânt pentru șirul `"IO"` plus caracterul terminator de șir ($2 + 1 = 3$ octeți, deci 1 cuvânt). Având în vedere că un șir se poate încheia în mijlocul unui cuvânt, definițiile următoare trebuie **aliniate**, utilizând instrucțiunea `.align 2`. De asemenea, este important să procesați șirurile înainte să le emiteți, deoarece pot conține **caractere speciale**, ca `'\n'`.

Instanțele clasei **`Bool`**, sunt similare celor ale clasei `Int`, unicul atribut fiind `0` sau `1`, pentru `false`, respectiv `true`.

`void` este reprezentat prin întregul MIPS `0`.

3.3 Literali

Pentru fiecare literal întreg, șir de caractere și boolean din program, există o definiție aferentă în zona `data`, respectând convențiile din secțiunea 3.2. În acest

Conținut	Adresă
Parametru n	
⋮	⋮
Parametru 2	`$fp + 16`
Parametru 1	`$fp + 12`
`$fp`	
`$s0`	
`$ra`	`$fp`
	`$sp`

Figura 2: Înregistrarea de activare

scop, este util să vă definiți **tabele de literali** indexate prin numele literalilor, pentru a le putea **refolosi** pe cele deja definite.

3.4 Definiții și apeluri de metode

Protocolul de apel este similar celui din curs, slide-urile 287–294, cu câteva modificări, datorate (1) prezenței suplimentare a obiectului pentru care realizăm apelul, în afara parametrilor, și (2) faptului că sistemul de execuție Cool consideră că registrul `$fp` (*frame pointer*) este gestionat exclusiv de către **apelat**, nu și de apelant! Prin urmare, salvarea registrului `$fp` se face tot de către apelat, la fel ca și reîncărcarea lui la final. Această strategie este **mai eficientă**, întrucât salvarea se face acum o singură dată, la intrarea în metodă, și nu de fiecare dată când se apelează o altă metodă din metoda curentă, ca în curs.

În ceea ce privește obiectul **`self`**, adresa acestuia este depusă în registrul `$a0`. Având în vedere că acesta va fi **modificat** în cursul execuției metodei, fie pentru a invoca alte metode, fie pentru a stoca rezultatul evaluării unei expresii, se recomandă **salvarea** acestui registru în `$s0`, pentru acces facil. În consecință, vechea valoare a acestui registru trebuie ea însăși **salvată** pe stivă, alături de `$fp` și `$ra`. Noua **înregistrare de activare** este ilustrată în fig. 2. Exemplul de cod din secțiunea 2 reflectă acest protocol.

În vederea implementării **apelurilor dinamice** de metodă (*dynamic dispatch*), este necesară accesarea intrării corecte din tabelul de metode aferent tipului dinamic al obiectului. Odată ce încărcați într-un registru adresa metodei, o puteți invoca utilizând instrucțiunea `jalr` (*jump and link register*), care, similar instrucțiunii `jal`, depune în registrul `$ra` adresa de revenire.

În cazul în care apelul se realizează pentru un obiect **`void`**, este necesar să invocați rutina predefinită `_dispatch_abort` (vezi secțiunea 3.9).

3.5 Construcția `let`

Variabilele de `let` sunt alocate pe **stivă**. Puteți utiliza orice schemă de stocare a acestora. Modelele din teste utilizează convenția din fig. 3.

3.6 Construcția `new`

Instrucțiunea `new` presupune alocarea dinamică a unui nou obiect pe *heap*, pornind de la obiectele prototip din zona statică. În cazul în care `new` este utili-

Conținut	Adresă
Parametru n	
⋮	⋮
Parametru 2	`$fp + 16`
Parametru 1	`$fp + 12`
`$fp`	
`$s0`	
`$ra`	`$fp`
Variabilă `let` 1	`$fp - 4`
Variabilă `let` 2	`$fp - 8`
⋮	⋮
Variabilă `let` m	
	`$sp`

Figura 3: Posibilă înregistrare de activare cu variabile de `let`

zat cu un **nume de clasă**, e.g. `new C`, se aplică mai întâi metoda predefinită `Object.copy` asupra obiectului prototip `C_protObj` (vezi secțiunea 3.9), iar apoi, asupra noului obiect, rutina de inițializare `C_init`.

În cazul expresiei **`new SELF_TYPE`**, este utilizat tabelul `class_objTab`, indexat pornind de la eticheta clasei reprezentând tipul dinamic al lui `self`. Astfel, dacă eticheta este *tag*, obiectul prototip se găsește în tabel la *offset*-ul $8 \cdot tag$, iar rutina de inițializare, la $8 \cdot tag + 4$.

3.7 Verificarea egalității obiectelor

Pentru verificarea egalității obiectelor, este necesară mai întâi testarea egalității **referințelor**. Dacă acestea sunt diferite, trebuie invocată rutina predefinită `equality_test` (vezi secțiunea 3.9), care verifică egalitatea de **conținut** în cazul valorilor predefinite, de tip `Int`, `String` și `Bool`.

3.8 Construcția `case`

Construcția `case` solicită prezența la execuție a anumitor informații despre ierarhia de clase. O observație utilă este că, asociind etichete claselor în ordinea dată de **parcurgerea în adâncime** a arborelui de moștenire, fiecare clasă și toate subclasele ei vor ocupa un interval de etichete consecutive, e.g. `5`, `6` și `7`, astfel încât extremele acestui interval pot fi utilizate pentru a surprinde orice tip dinamic aferent unei anumite ramuri a construcției `case`.

În cazul în care nicio ramură **nu se potrivește**, este necesară invocarea rutinei predefinite `_case_abort`. În cazul în care obiectul analizat este **`void`**, este necesară invocarea rutinei predefinite `_case_abort2` (vezi secțiunea 3.9).

3.9 Metode și rutine predefinite

Secțiunea prezintă metodele și rutinele predefinite, și modalitatea lor de invocare:

`Object.copy` Copiază pe *heap* obiectul cu adresa în registrul `$a0` și depune adresa noului obiect tot în `$a0`.

`Object.abort` Afișează numele tipului dinamic al obiectului cu adresa în `$a0` și încheie execuția programului. Tabelul `class_nameTab` este accesat în acest scop.

`Object.type_name` Întoarce în `$a0` numele tipului dinamic al obiectului cu adresa în `$a0`.

`IO.out_string` Afișează obiectul șir de caractere primit ca parametru.

`IO.out_int` Afișează obiectul număr întreg primit ca parametru.

`IO.in_string` Întoarce în `$a0` adresa obiectului șir de caractere citit de la tastatură.

`IO.in_int` Întoarce în `$a0` adresa obiectului număr întreg citit de la tastatură.

`String.length` Întoarce în `$a0` adresa obiectului număr întreg care reprezintă lungimea obiectului șir de caractere cu adresa în `$a0`.

`String.concat` Întoarce în `$a0` adresa unui **nou** obiect șir de caractere, rezultat din concatenarea obiectului cu adresa în `$a0` cu obiectul primit ca parametru.

`String.substr` Întoarce în `$a0` adresa unui nou obiect șir de caractere, reprezentând subșirul obiectului cu adresa în `$a0`, determinat de index-ul și lungimea date ca parametri.

`equality_test` Verifică dacă obiectele cu adresele în `$t1` și `$t2` au același tip dinamic predefinit și aceeași valoare. Dacă da, este întoarsă valoarea din `$a0`; altfel, din `$a1`.

`_dispatch_abort` Este utilizată în cazul unui apel de metodă pe un obiect **`void`**. Afișează un mesaj de eroare, ce include numele fișierului, ca obiect șir de caractere, cu adresa în `$a0`, și numărul liniei din fișier, ca întreg MIPS (utilizați instrucțiunea `li`) în `$t1`. Încheie execuția programului.

`_case_abort` Este utilizată când **nu** se realizează potrivire cu nicio ramură a instrucțiunii `case`. Afișează un mesaj de eroare ce include numele tipului dinamic al obiectului cu adrsa în `$a0` și încheie execuția programului.

`_case_abort2` Este utilizată când instrucțiunea `case` analizează un obiect **`void`**. Similar cu `_dispatch_abort`, afișează un mesaj de eroare cu numele fișierului (`$a0`) și linia (`$t1`) aferente obiectului problematic, și încheie execuția programului.

4 Testare

Testele pot fi **rulate** pe Linux, executând scriptul `tester.sh`, în cadrul căruia puteți actualiza la nevoie variabila `CLASSPATH` pentru a indica `jar`-ul de ANTLR. Pentru rularea manuală a unui program MIPS în simulatorul ***spim***, în linia de comandă, utilizați:

```
spim -exception_file trap.handler.nogc -file <file.s>
```

Pentru rularea în ***QtSpim***, selectați `Simulator -> Settings -> MIPS -> Load exception handler` și alegeți fișierul `trap.handler.nogc`. Apoi, încărcați programul dorit utilizând butonul `Reinitialize and Load File`.

Testele se află în directorul `tests/tema3` din rădăcina arhivei de pornire. Fișierele `.cl` conțin programe Cool de compilat. Modele de programe MIPS echivalente sunt furnizate în fișierele `.s-model`. Fișierele `.ref` conțin ieșirile de referință ale simulatorului. Pentru fiecare test, sistemul de testare redirectează ieșirea standard a compilatorului către un fișier `.s`, pe care îl execută în simulator, și stochează ieșirea acestuia într-un fișier `.out`, pe care îl compară apoi cu cel de referință.

Având în vedere că testele verifică **incremental** funcționalitatea compilatorului, le puteți folosi pentru a vă ghida **dezvoltarea** temei!

Primele patru teste nu afișează nimic. Scopul lor este de a vă ajuta să organizați codul generat pentru ierarhia de clase. Ultimul test, 32, este singurul care citește intrarea de la consolă, folosind metoda `IO.in_int`. Sistemul de testare fixează intrarea acestuia la `5`, astfel încât să nu fie necesară introducerea manuală a numărului de fiecare dată când rulați testele.

5 Precizări

- Clasele, variabilele și metodele trebuie **indexate**, pentru a le putea referi corect în codul generat. Spre exemplu, clasele necesită etichete, atributele trebuie referite în funcție de *offset*-ul în cadrul reprezentării obiectului din care fac parte, metodele sunt caracterizate de *offset*-ul în cadrul tabelelor de metode, parametrii formali, de *offset*-ul pe stivă etc. Puteți reține aceste informații în **simbolurile** asociate.

- ***StringTemplate*** posedă anumite mecanisme mai avansate, pe care le puteți exploata în rezolvarea temei. Spre exemplu, apelarea metodei `add` pe un *template*, utilizând un obiect `List` ca valoare, este echivalentă cu secvența de apeluri `add` pentru fiecare element al listei. *Template*-urile pot fi aplicate asupra fiecărei valori a unui atribut (*mapping*), utilizând sintaxa `<attr:transform()>`, unde `attr` este un atribut cu mai multe valori, iar `transform` este un *template* cu un singur parametru. De asemenea, există și *template*-ul special `<if>`, prin care puteți cosmetiza formatarea. Găsiți mai multe detalii în documentația de la adresa `https://github.com/antlr/stringtemplate4/blob/master/doc/index.md`.

6 Referințe

1. Manualul limbajului Cool.
   `https://acs.curs.pub.ro/2018/mod/resource/view.php?id=2754`

2. Documentația sistemului de execuție al limbajului Cool.
   `http://web.stanford.edu/class/cs143/materials/cool-runtime.pdf`

3. Documentația SPIM.
   `https://acs.curs.pub.ro/2018/mod/resource/view.php?id=7816`

4. Documentația *StringTemplate*.
https://github.com/antlr/stringtemplate4/blob/master/doc/index.md