

# Compilers Assignment 6

106119029 Dipesh Kafle, 106119073 Marmik Upadhyay, 106119117 Sobhagya Singh Dewal

## Features Implemented

- While Loop
- Conditional Statements
- Functions
- Five data types : string, int, double, bool, char
- Arithmetic Operations
- String Concat operation
- Logical Comparisons
- Type Checking
- Basic Type inference
- Undefined symbols static analysis

## Code

[Link](#)

## Error Discovery Capabilities

### Parsing Errors

---

Break Outside of Loop is Error

```
Enma/build on ⚡ main [?] via △ v3.23.0 took 9s
λ cat .../examples/break_error.enma

break;
Enma/build on ⚡ main [?] via △ v3.23.0
λ ./bin/enma-frontend .../examples/break_error.enma
[line 2] Error at ';' : Break statement is not inside any form of loop. This is not valid
Enma/build on ⚡ main [?] via △ v3.23.0
λ
```

---

---

Continue Outside of Loop is Error

```
Enma/build on ⚡ main [?] via △ v3.23.0
λ cat .../examples/continue_error.enma

continue;

Enma/build on ⚡ main [?] via △ v3.23.0
λ ./bin/enma-frontend .../examples/continue_error.enma
[line 2] Error  at ';' : Continue statement is not inside any form of loop. This is not valid
Enma/build on ⚡ main [?] via △ v3.23.0
λ
```

---

Missing condition in while loop

```
Enma/build on ⚡ main [?] via △ v3.23.0
λ cat .../examples/parse_error.enma
while(){ // missing condition inside parens, will be detected
    let x = 0;
    println x;
}
Enma/build on ⚡ main [?] via △ v3.23.0
λ ./bin/enma-frontend .../examples/parse_error.enma
[line 1] Error  at ')' : Unexpected token
[line 4] Error  at ')' : Unexpected token
Enma/build on ⚡ main [?] via △ v3.23.0
λ
```

---

Missing right hand side in let statement

```
Enma/build on ⚡ main [?] via △ v3.23.0 took 12s
λ cat .../examples/missing_initialization.enma
let x : int ; // language requires initialization during declaration
Enma/build on ⚡ main [?] via △ v3.23.0
λ ./bin/enma-frontend .../examples/missing_initialization.enma
[line 1] Error  at ';' : Cannot have empty declartion. It must be of form let <id> = <expr>;
Enma/build on ⚡ main [?] via △ v3.23.0
λ
```

---

Missing Type after : in let statement

```
Enma/build on ↵ main [?] via △ v3.23.0
λ cat ../examples/type_miss_error.enma
let x = 10;
let y : = 20;
Enma/build on ↵ main [?] via △ v3.23.0
λ ./bin/enma-frontend ../examples/type_miss_error.enma
[line 2] Error at '=' : Expected Type in let declaration
Enma/build on ↵ main [?] via △ v3.23.0
λ
```

### Semantic Errors with static analysis

Semantic Analysis done in this file

Type Mismatch

```
Enma/build on ↵ main [!?] via △ v3.23.0
λ cat ../examples/type_mismatch.enma
let x : string =10;
let y : int = 10;

print x + y;

Enma/build on ↵ main [!?] via △ v3.23.0
λ ./bin/enma-frontend ../examples/type_mismatch.enma
=====
S-Expression
=====
(Let x:string 10)
(Let y:int 10)
(Print (+ x y))

=====
Semantic Errors
=====
Type mismatch at line 1 in let stmt
Undeclared symbol x in line 4
Enma/build on ↵ main [!?] via △ v3.23.0
λ
```

## Invalid Operands

```
Enma/build on ⚡ main [$!?] via △ v3.23.0
└ cat ..../examples/bad_operands.enma
let x : string = "abc";
let y : int = 10;

print x + y;

Enma/build on ⚡ main [$!?] via △ v3.23.0
└ ./bin/enma-frontend ..../examples/bad_operands.enma
=====
S-Expression
=====
(Let x:string abc)
(Let y:int 10)
(Print (+ x y))

=====
Semantic Errors
=====
Type of lhs ≠ Type of rhs in BinaryExpr in line 4
Enma/build on ⚡ main [$!?] via △ v3.23.0
└
```

## Undeclared Symbol

```
Enma/build on ⚡ main [$!?] via △ v3.23.0
└ cat ..../examples/undeclared_symbols_input.enma
let z = "abc";

let x : int = 10;

fn sum(n: int ): int {
    let i : int = 0;
    let sm : int = 0;
    while( i ≤ n){
        sm = sm + i;
        i = i + 1;
    }
    return all_sum; // is not declared anywhere
}

Enma/build on ⚡ main [$!?] via △ v3.23.0
└ ./bin/enma-frontend ..../examples/undeclared_symbols_input.enma
=====
S-Expression
=====
(Let z abc)
(Let x:int 10)
(Fn sum [n: int] returns int [ (Let i:int 0); (Let sm:int 0); (While cond (≤ i n) do (Block [ (= sm (+ sm i)); (= i (+ i 1)) ]); (Return all_sum) )]

=====
Semantic Errors
=====
Undeclared symbol all_sum in line 12
Enma/build on ⚡ main [$!?] via △ v3.23.0
└
```

## Codegen for continue and break

```
Enma/build on ⚡ main [$!?] via ▲ v3.23.0
└ cat ./examples/continue_and_break.enma

let i: int =0;
while(i<10){
    if (i<5){
        continue;
    }
    break;
}
Enma/build on ⚡ main [$!?] via ▲ v3.23.0
└ ./bin/enma-frontend ./examples/continue_and_break.enma
=====
S-Expression
=====
(Let i:int 0)
(While cond (< i 10) do (Block [ (If (< i 5) then (Block [ (continue) ]); (break) )])
=====

Intermediate Code
=====
PushI 0 //Push intermediate
Load i
.WHILE0:
Push i //Push variable
PushI 10 //Push intermediate
Op < // Pops top 2, performs op and pushes result
Jnz <IP+2> // if true go to start of block
Jmp <IP+11> // if false go to end of block
.BLOCK0:
Push i //Push variable
PushI 5 //Push intermediate
Op < // Pops top 2, performs op and pushes result
.IFO:
Jz <IP+3> // if false go to after then block
.BLOCK1:
Jmp WHILE0
Jmp <IP+2> // break by going to end of loop
Jmp WHILE0 // jump back to loop
Enma/build on ⚡ main [$!?] via ▲ v3.23.0
└ █
```

Valid Program [Here in first and third line , type is inferred automatically with type inference]

```
Enma/build on ⚡ main [$!?] via △ v3.23.0
λ cat .../examples/input.emma
let z = "abc";

let x = 10;

fn is_origin(x: int, y: int){
    let ans = true;
    if (x == 0 and y == 0 ){
        ans= true;
    }else{
        ans= !true;
    }
    return ans;
}

/* THis is sum function
It sums from 0 to n inclusive
*/
fn sum(n: int ): int {
    let i : int = 0;
    let sm : int = 0;
    while( i ≤ n){
        sm = sm + i;
        i = i + 1;
    }
    /*
    I want it to be able to return even if we just have
    ^
    sm
    ^
    without the return in front of it. Like in rust. Not supported
    as of now though.
    */
    return sm;
}
Enma/build on ⚡ main [$!?] via △ v3.23.0
λ
```

CodeGen for above program

```

Enma/build on ⚡ main [$!?] via ▲ v3.23.0 [125/516]
└─ ./bin/enma-frontend ..../examples/input.enma

S-Expression
(Let z abc)
(Let x 10)
(Fn is_origin [x: int, y: int] returns void [ (Let ans true); (If (and (= x 0) (= y 0)) then (Block [ (= ans true) ]) else (Block [ (= ans (! true)) ])); (Return ans) ] )
(Fn sum [n: int] returns int [ (Let i:int 0); (Let sm:int 0); (While cond (<= i n) do (Block [ (= sm (+ sm i)); (= i (+ i 1)) ]); (Return sm) ] )

Intermediate Code
PushI "abc" //Push intermediate
Load z
PushI 10 //Push intermediate
Load x
is_origin:
PushI true //Push intermediate
Load ans
Push x //Push variable
PushI 0 //Push intermediate
Op = // Pops top 2, performs op and pushes result
Push y //Push variable
PushI 0 //Push intermediate
Op = // Pops top 2, performs op and pushes result
Op and // Pops top 2, performs op and pushes result
.IFO:
Jz <IP+5> // if false go to after then block
.BLOCK0:
Push ans //Push variable
PushI true //Push intermediate
Op = // Pops top 2, performs op and pushes result
Jmp <IP+6>
.BLOCK1:
Push ans //Push variable
PushI true //Push intermediate
PrefixOp ! //Pops one from stack, does op and pushes it back
Op = // Pops top 2, performs op and pushes result
Ret
.sum:
PushI 0 //Push intermediate
Load i
PushI 0 //Push intermediate
Load sm
.WHILE0:
Push i //Push variable
Push n //Push variable
Op ≤ // Pops top 2, performs op and pushes result
Jnz <IP+2> // if true go to start of block
Jmp <IP+13> // if false go to end of block
.BLOCK2:
Push sm //Push variable
Push sm //Push variable
Push i //Push variable
Op + // Pops top 2, performs op and pushes result
Op = // Pops top 2, performs op and pushes result
Push i //Push variable
Push i //Push variable
PushI 1 //Push intermediate
Op + // Pops top 2, performs op and pushes result
Op = // Pops top 2, performs op and pushes result
Jmp WHILE0 // jump back to loop
Ret
Enma/build on ⚡ main [$!?] via ▲ v3.23.0
└─ [0:2.0] 1:nvim 2:[tmux]* 3:nvim- 4:evince
02 May 2022 - 02:56 PM - dipesh

```