Code Generation

This is a code generation project that converts a high-level programming language into executable Moon assembly code.

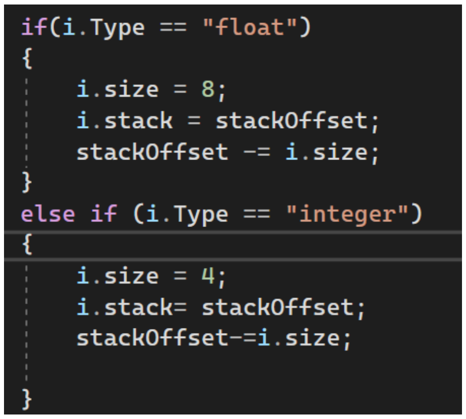
Not all generation implemented.

	Implemented	tested
Memory allocation		
int	#	#
float	#	#
array	#	
object	#	
Array of object	#	
functions		
branch	#	#
Pass parameter	#	#
return	#	#
Data member		
statement		
assignment	#	#
condition	#	#
loop		
input		
output	#	#
Aggregate data		
array	c	
Object of array		
Object access basic		
Object access array		
Expression		
Complex expression	#	#
array		

Object facter		

Desgin

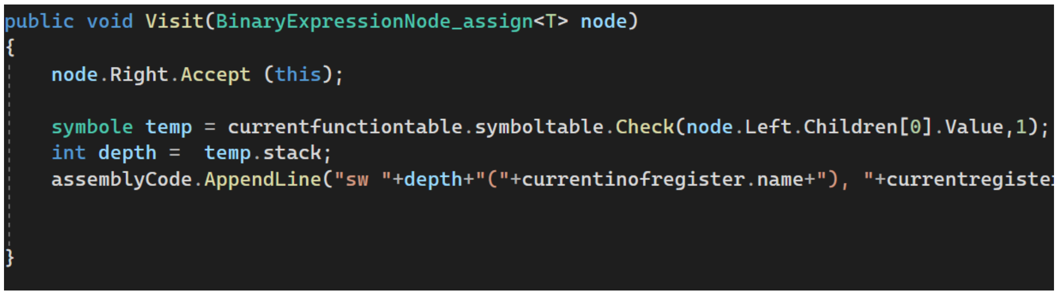
Memory Allocation: memory allocation for different types of variables, ensuring proper memory space allocation and management.



```
if(i.Type == "float")
{
    i.size = 8;
    i.stack = stackOffset;
    stackOffset -= i.size;
}
else if (i.Type == "integer")
{
    i.size = 4;
    i.stack= stackOffset;
    stackOffset-=i.size;

}
```

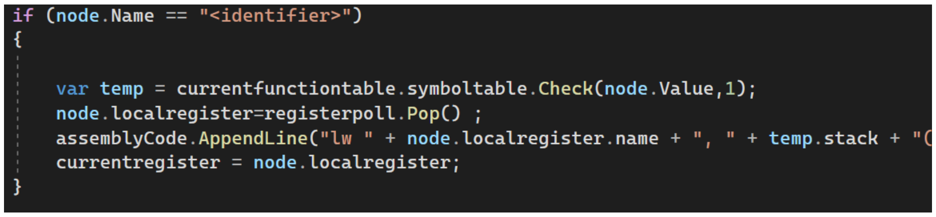
Functions: handling of functions, including parameter passing, return value management, and member function access.

Statements: Translation of various statements into executable Moon code, such as assignment, conditional, loop, input/output, and return statements.

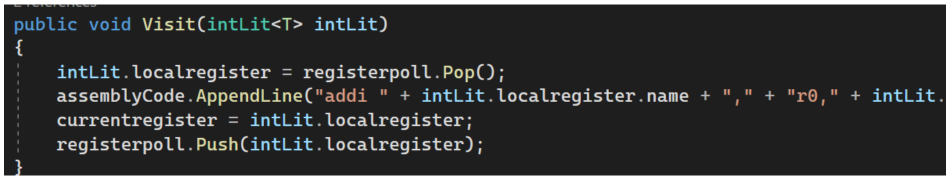


```
public void Visit(BinaryExpressionNode_assign<T> node)
{
    node.Right.Accept (this);

    symbole temp = currentfunctiontable.symboltable.Check(node.Left.Children[0].Value,1);
    int depth =  temp.stack;
    assemblyCode.AppendLine("sw "+depth+"("+currentinofregister.name+"), "+currentregiste


}
```

Aggregate Data Elements Access: Implementation of access to aggregate data types, including arrays and objects, using offset calculations.

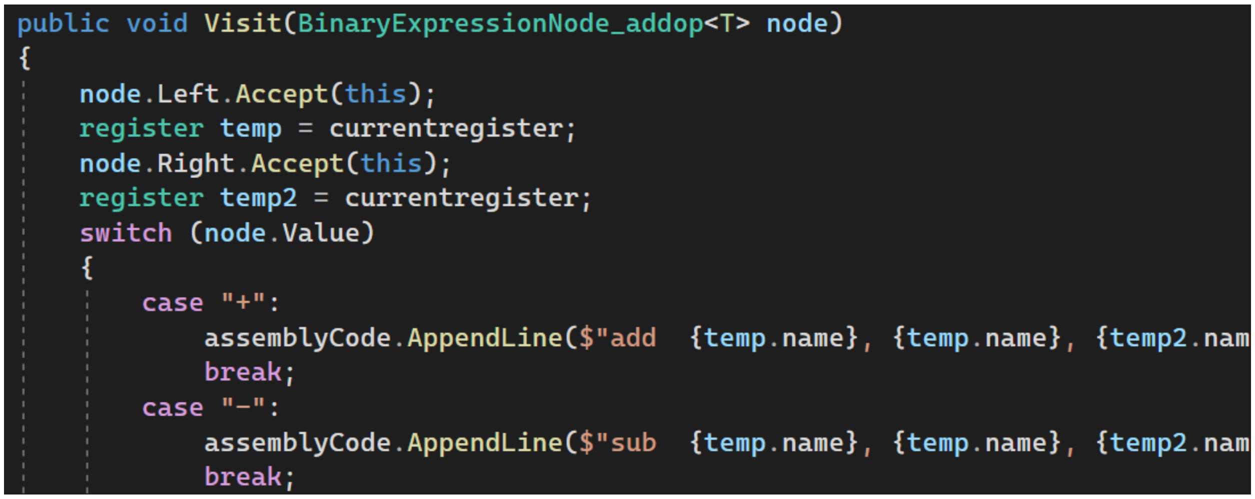


```
if (node.Name == "<identifier>")
{

    var temp = currentfunctiontable.symboltable.Check(node.Value,1);
    node.localregister=registerpoll.Pop() ;
    assemblyCode.AppendLine("lw " + node.localregister.name + ", " + temp.stack + "(
    currentregister = node.localregister;
}
```



```
public void Visit(intLit<T> intLit)
{
    intLit.localregister = registerpoll.Pop();
    assemblyCode.AppendLine("addi " + intLit.localregister.name + "," + "r0," + intLit.
    currentregister = intLit.localregister;
    registerpoll.Push(intLit.localregister);
}
```

Expressions: Evaluation of complex expressions, including arithmetic, logical, and relational operations, array indexing, and nested object member access.

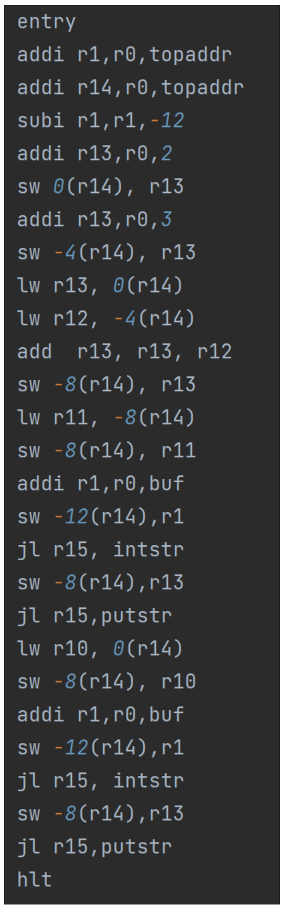


```
public void Visit(BinaryExpressionNode_addop<T> node)
{
    node.Left.Accept(this);
    register temp = currentregister;
    node.Right.Accept(this);
    register temp2 = currentregister;
    switch (node.Value)
    {
        case "+":
            assemblyCode.AppendLine($"add  {temp.name}, {temp.name}, {temp2.nam
            break;
        case "-":
            assemblyCode.AppendLine($"sub  {temp.name}, {temp.name}, {temp2.nam
            break;
```

Example:

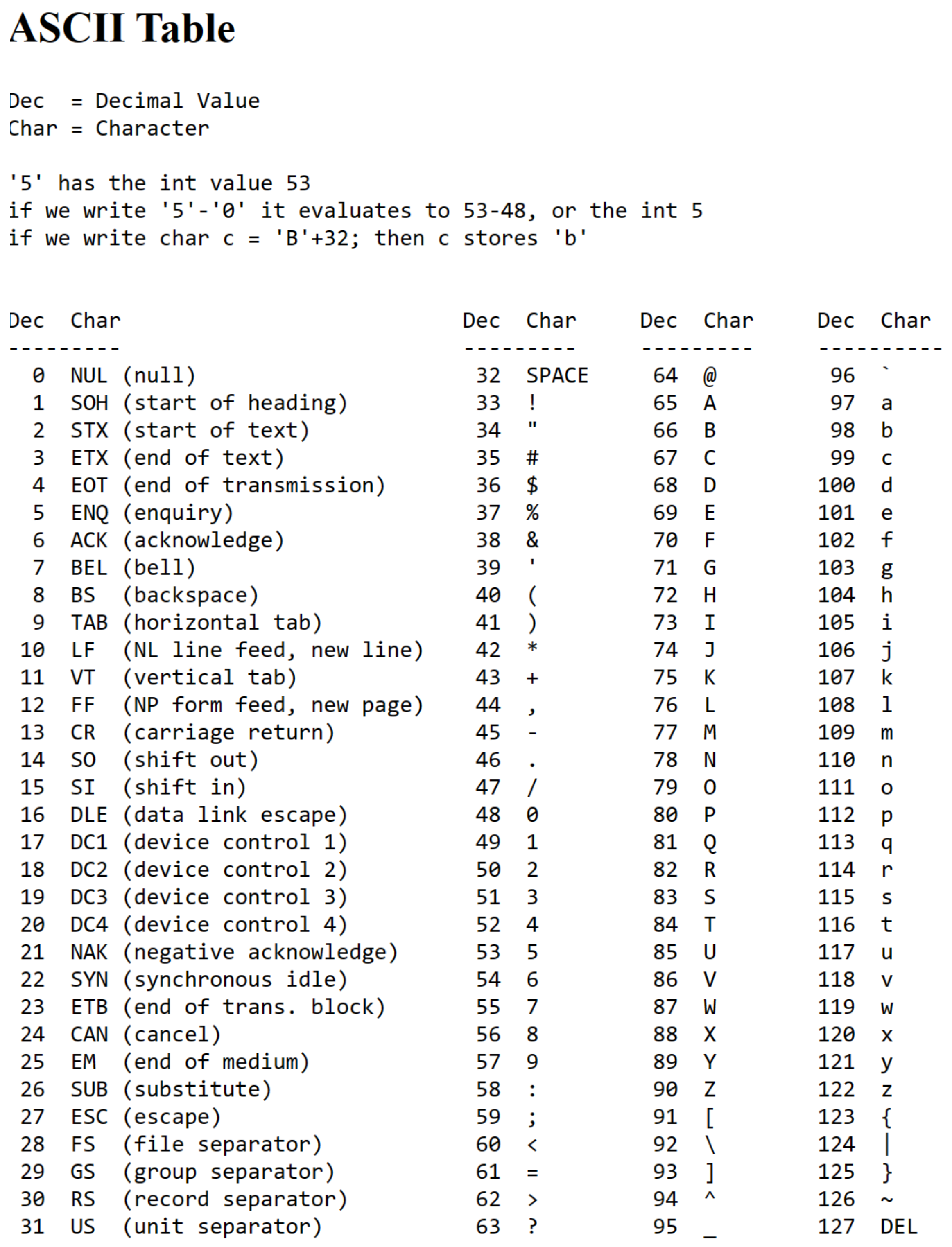
```
function main() => void
{
  localvar x: integer;
  localvar y: integer;
  localvar z: integer;
  x=2;
  y=3;
  z=x+y;
  write(z);
  write(x);

}
```

Assembly code:



```
entry
addi r1,r0,topaddr
addi r14,r0,topaddr
subi r1,r1,-12
addi r13,r0,2
sw 0(r14), r13
addi r13,r0,3
sw -4(r14), r13
lw r13, 0(r14)
lw r12, -4(r14)
add  r13, r13, r12
sw -8(r14), r13
lw r11, -8(r14)
sw -8(r14), r11
addi r1,r0,buf
sw -12(r14),r1
jl r15, intstr
sw -8(r14),r13
jl r15,putstr
lw r10, 0(r14)
sw -8(r14), r10
addi r1,r0,buf
sw -12(r14),r1
jl r15, intstr
sw -8(r14),r13
jl r15,putstr
hlt
```

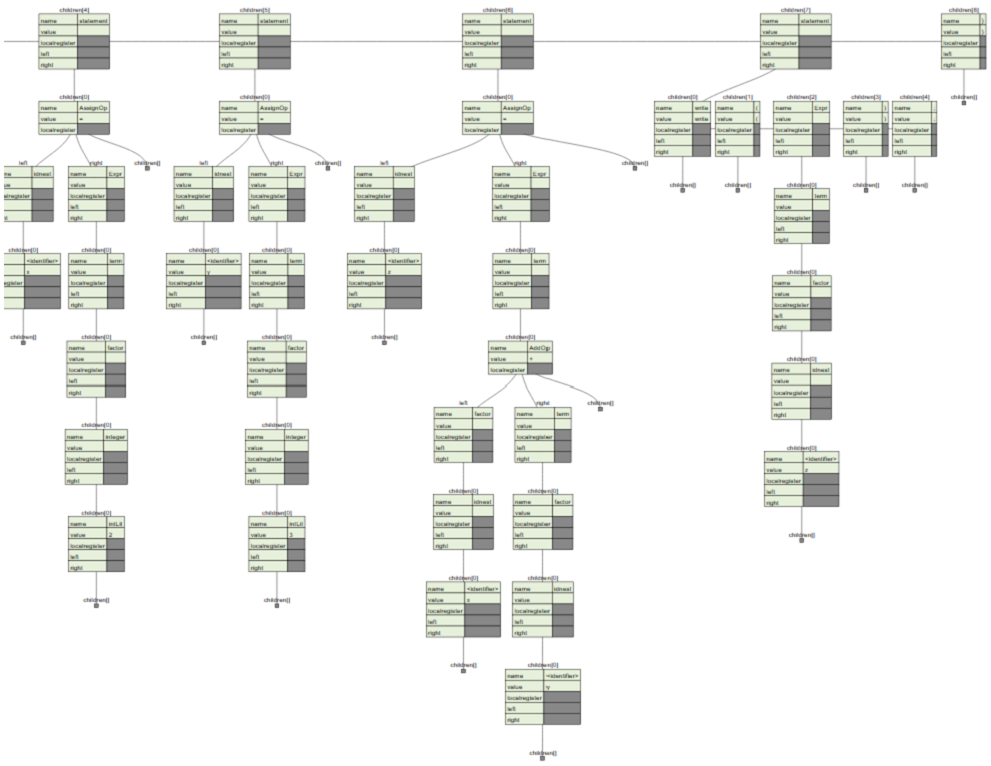
Use of tool:



ASCII Table

```
Dec  = Decimal Value
Char = Character

'5' has the int value 53
if we write '5'-'0' it evaluates to 53-48, or the int 5
if we write char c = 'B'+32; then c stores 'b'
```

Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL



[Online JSON to Tree Diagram Converter (vanya.jp.net)]