**Code Generation**

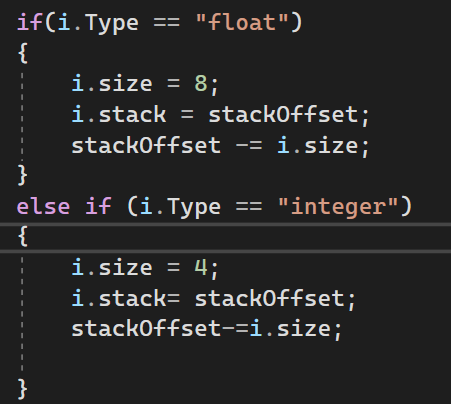
This is a code generation project that converts a high-level programming language into executable Moon assembly code.

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 | # |  |
| Object of array |  |  |
| Object access basic | # |  |
| Object access array |  |  |
| **Expression** |  |  |
| Complex expression | # | # |
| array | # |  |
| Object facter |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

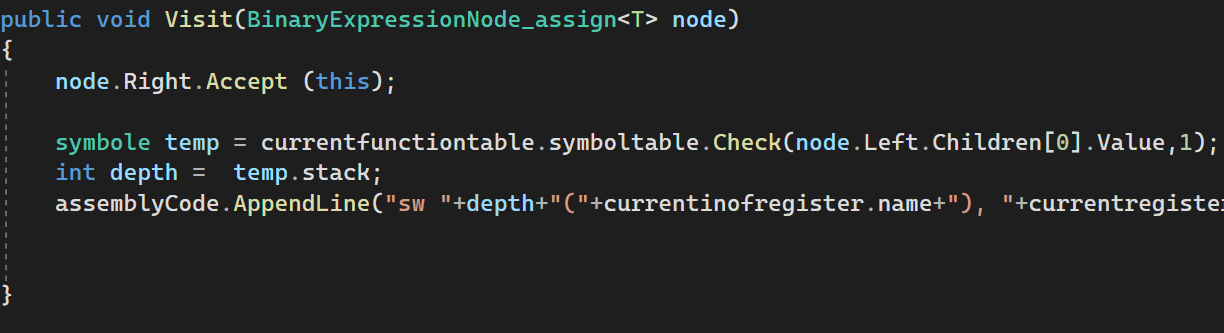
**Desgin**

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

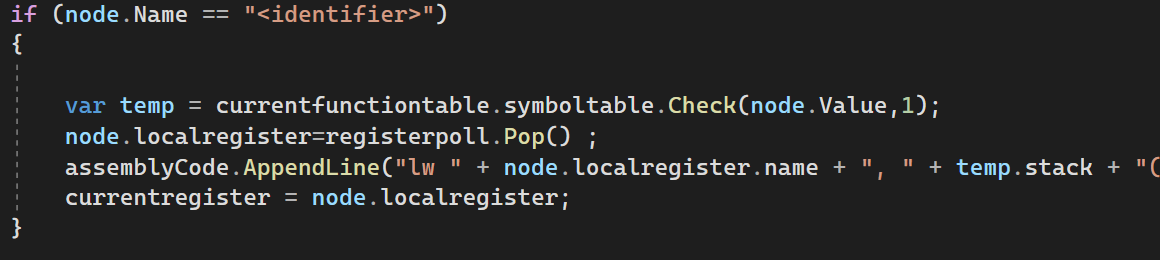


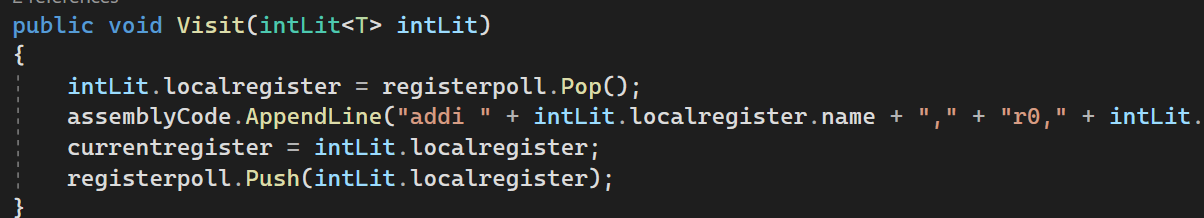
Functions: handling of functions, including parameter passing, return value management, and member function access.

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

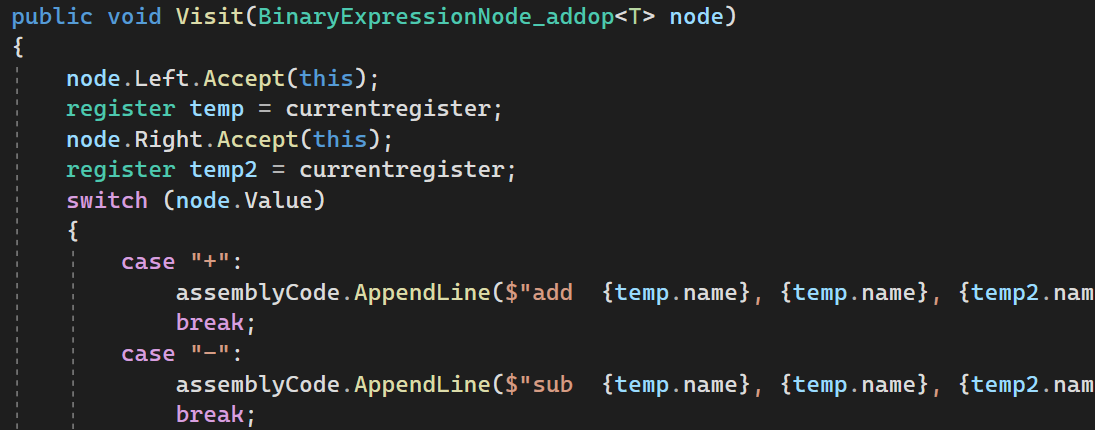


Aggregate Data Elements Access: Implementation of access to aggregate data types, including arrays and objects, using offset calculations.

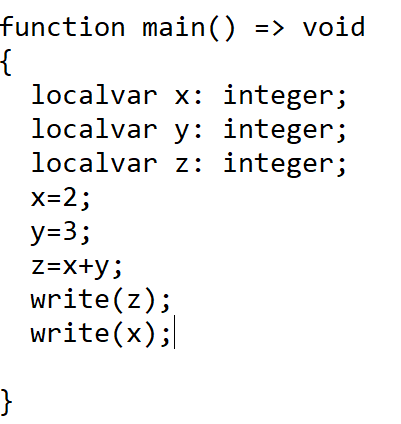




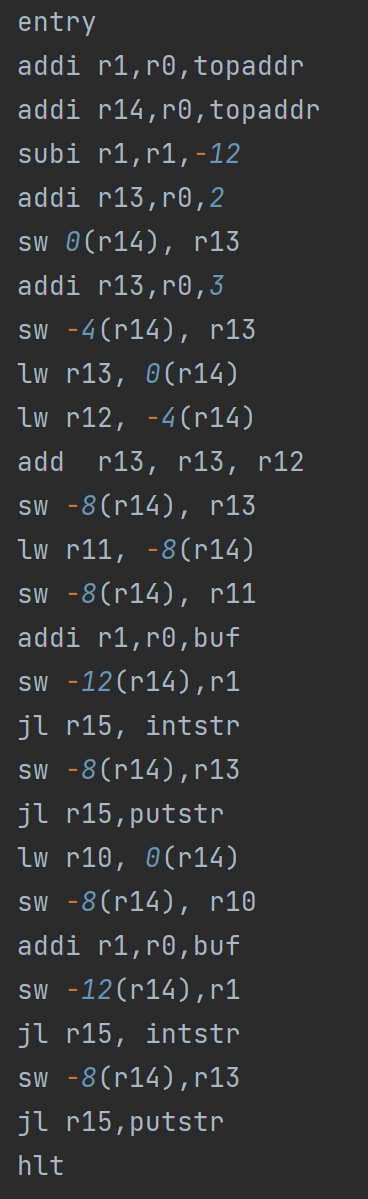
Expressions: Evaluation of complex expressions, including arithmetic, logical, and relational operations, array indexing, and nested object member access.



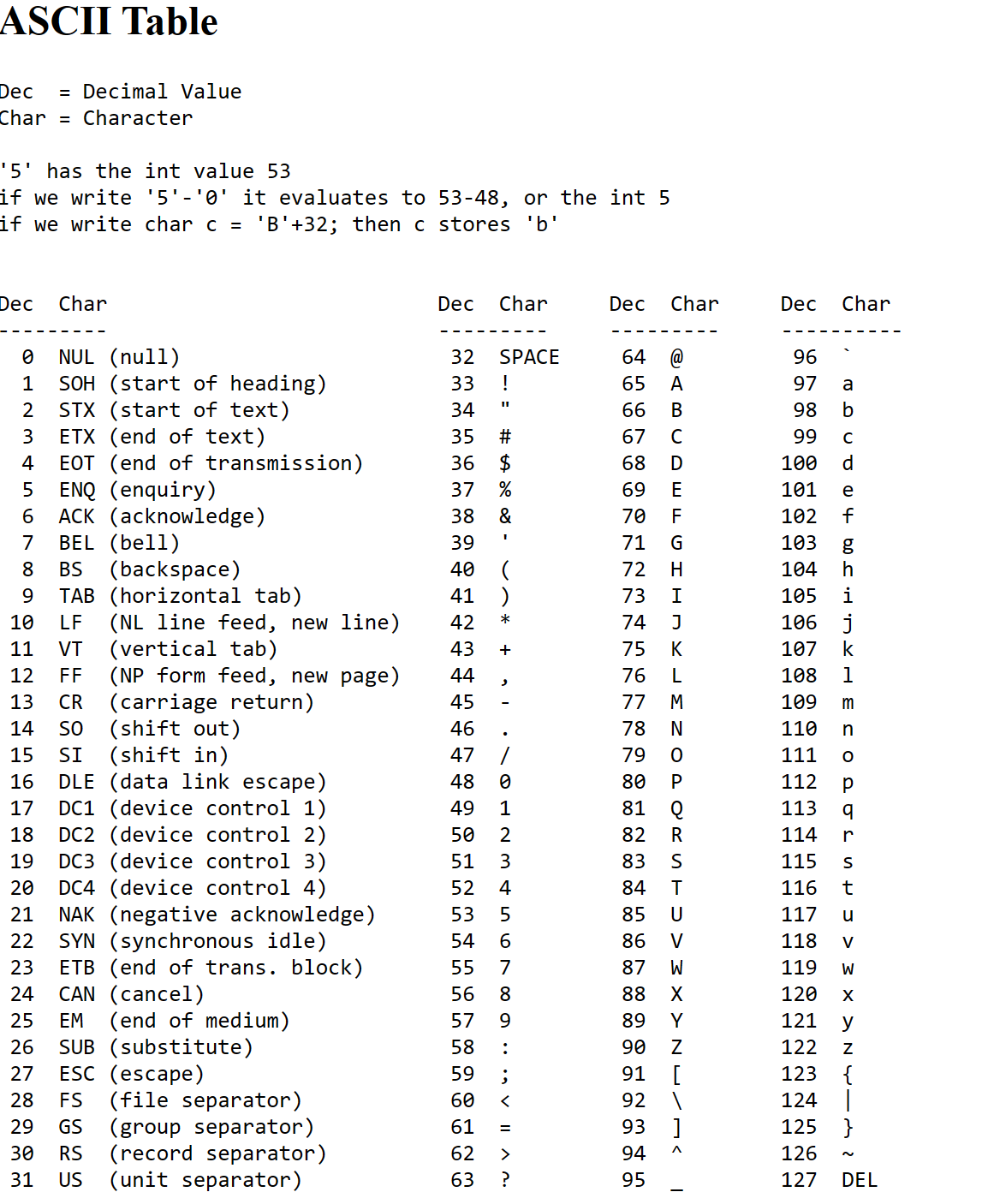
**Example:**

****

**Assembly code:**

****

**Use of tool:**

****



[Online JSON to Tree Diagram Converter (vanya.jp.net)](https://vanya.jp.net/vtree/)

isarray= true;

int tempvar = 4;

assemblyCode.AppendLine($"addi r2, r0, 4");

for (int i = 2; i < node.Children.Count; i++)

{

node.Children[i].Accept(this);

assemblyCode.AppendLine($"lw r3,{currentsymbol}(r0)");

assemblyCode.AppendLine($"mul r2,r2,r3");

tempvar = tempvar \* int.Parse(currentstring);

i++;

}

assemblyCode.AppendLine($"addi r3, r0, 4");

assemblyCode.AppendLine($"add r2,r2,r3");

currentsymbol = node.Children[0].Value;

tempvar += 4;

currentregister=registerpoll.Pop();

assemblyCode.AppendLine($"add {currentregister.name}, r0, r2");

currentarrayheight = tempvar;

registerpoll.Push(currentregister);