v0.8.2 Multi-Level Nesting Implementation Plan

Analysis Complete

Current State (v0.8.1):

- Single-level nesting supported
- Block parsing done inline (lots of code duplication)
- Warnings for nested control flow: lines 469-471, 551-553, 653, 761
- Flattening algorithm: reserve slot, parse simple statements, fill in indices

Target State (v0.8.2):

- Multi-level nesting (2-5 levels) supported
- Recursive block flattening
- Proper index calculation at all depths
- Clean, maintainable code structure

Implementation Strategy

Key Changes Needed:

1. **Create Recursive Flatten_Block Procedure**:
   - Input: Block_JSON (string containing JSON array), Array_Pos (position of '[‘), Depth (current nesting level)
   - Output: Modifies Module.Functions(Func_Idx).Statements array
   - Logic:
   ```ada
   procedure Flatten_Block (Block_JSON : String; Array_Pos : Natural; Depth : Natural := 0) is
   Stmt_Pos : Natural := Array_Pos + 1;
   Stmt_Start, Stmt_End : Natural;
   begin
   if Depth > 5 then
   Put_Line (“[ERROR] Maximum nesting depth (5) exceeded”);
   return;
   end if;

   while Module.Functions (Func_Idx).Stmt_Cnt < Max_Statements loop
   Get_Next_Object (Block_JSON, Stmt_Pos, Stmt_Start, Stmt_End);
   exit when Stmt_Start = 0 or Stmt_End = 0;
   ```

```
 declare
   Stmt_JSON : constant String := Block_JSON (Stmt_Start .. Stmt_End);
   Stmt_Type : constant String := Extract_String_Value (Stmt_JSON, "type");
 begin
   -- Reserve slot
   Module.Functions (Func_Idx).Stmt_Cnt := Module.Functions (Func_Idx).Stmt_Cnt +
1;
   Current_Idx : constant Positive := Module.Functions (Func_Idx).Stmt_Cnt;

   -- Initialize defaults
   [... initialization code ...]

   -- Parse based on type
   if Stmt_Type = "assign" or Stmt_Type = "var_decl" then
     [... assign parsing ...]
   elsif Stmt_Type = "return" then
     [... return parsing ...]
   elsif Stmt_Type = "call" then
     [... call parsing ...]
   elsif Stmt_Type = "if" then
     -- RECURSIVE CASE
     Module.Functions (Func_Idx).Statements (Current_Idx).Kind := Stmt_If;
     Cond_Str : constant String := Extract_String_Value (Stmt_JSON, "condition");
     Then_Array_Pos : constant Natural := Find_Array (Stmt_JSON, "then_block");
     Else_Array_Pos : constant Natural := Find_Array (Stmt_JSON, "else_block");

     -- Store condition
     Module.Functions (Func_Idx).Statements (Current_Idx).Condition := ...;

     -- Recursively flatten then_block
     Then_Start_Idx : Natural := Module.Functions (Func_Idx).Stmt_Cnt + 1;
     Count_Before : constant Natural := Module.Functions (Func_Idx).Stmt_Cnt;

     if Then_Array_Pos > 0 then
       Flatten_Block (Stmt_JSON, Then_Array_Pos, Depth + 1);  -- RECURSIVE CALL
     end if;

     Then_Count : constant Natural := Module.Functions (Func_Idx).Stmt_Cnt - Count_
Before;

     -- Recursively flatten else_block
     Else_Start_Idx : Natural := 0;
     Else_Count : Natural := 0;
     if Else_Array_Pos > 0 then
       Else_Start_Idx := Module.Functions (Func_Idx).Stmt_Cnt + 1;
       Count_Before2 : constant Natural := Module.Functions (Func_Idx).Stmt_Cnt;
       Flatten_Block (Stmt_JSON, Else_Array_Pos, Depth + 1);  -- RECURSIVE CALL
       Else_Count := Module.Functions (Func_Idx).Stmt_Cnt - Count_Before2;
     end if;

     -- Fill in indices
     Module.Functions (Func_Idx).Statements (Current_Idx).Block_Start := Then_Start
_Idx;
     Module.Functions (Func_Idx).Statements (Current_Idx).Block_Count :=
Then_Count;
     Module.Functions (Func_Idx).Statements (Current_Idx).Else_Start := Else_Start_
Idx;
     Module.Functions (Func_Idx).Statements (Current_Idx).Else_Count :=
Else_Count;

   elsif Stmt_Type = "while" then
     -- Similar recursive logic for while
```

```
      [... while with recursive Flatten_Block call for body ...]
    elsif Stmt_Type = "for" then
      -- Similar recursive logic for for
      [... for with recursive Flatten_Block call for body ...]
    end if;
  end;

  Stmt_Pos := Stmt_End + 1;
```

end loop;
end Flatten_Block;
```

2. **Replace Main Body Parsing**:
   - Remove all the old duplicate code (lines 287-797)
   - Replace with single call: `Flatten_Block (Func_JSON, Body_Pos);`

## Testing Plan

1. Build the modified code
2. Test with existing single-level test cases
3. Test with nested_2_levels.json
4. Test with nested_3_levels.json
5. Test with nested_4_levels.json
6. Test with nested_5_levels.json
7. Create custom test specs for mixed nesting
8. Compare output with Python pipeline
9. Verify IR correctness
10. Test end-to-end C code generation

## Risk Mitigation

- Keep git backup before major changes
- Test incrementally
- Compare with Python reference implementation
- Verify indices manually for simple cases
- Use logging to debug index calculations

## Next Steps

1. Create complete new version of body parsing section
2. Test compilation
3. Test with single-level cases (regression)
4. Test with multi-level cases (new functionality)
5. Document and commit
```