

Input: Function *func* at WyIL level

Output: The *list* of optimised C code using copy and deallocation analysers

```
1: Variables
2:   CopyAnalyser: Copy elimination analyser
3:   DeallocAnalyser: De-allocation analyser
4:   list: a list of optimised C code
5: end Variables
6: procedure CODE_OPTIMISE(func, list)
7:   list := []
8:   // Beginning function signature
9:   list.append("return_type")// Function return type
10:  list.append("function_name")// Function name
11:  for each param in func do// Function parameters
12:    type ← type of param from func function declaration
13:    list.append("type param,")// Append param
14:    if param is Array then// Add extra passed parameter
15:      list.append("size.t param_size,")
16:      list.append("bool param_dealloc,")
17:    end if
18:  end for
19:  list.append("{}")// Ending function signature
20:  // Beginning variable declaration
21:  vars = variable tables from func variable declaration
22:  for each var in vars do
23:    type ← type of var
24:    list.append("type var;")
25:    if var is Array then
26:      list.append("size.t var_size = 0;")
27:      list.append("bool var_dealloc = false;")// Add deallocation flag
28:    end if
29:  end for// Ending variable declaration
30:  // Beginning function body
31:  for each code in function body do// Generate Optimised Code
32:    switch code do
33:      case Array Generator
34:        list.append(ARRAYGENERATOROPT(code, func))
35:      case Array Assignment
36:        list.append(ASSIGNMENTOPT(code, func))
37:      case Function Call
38:        list.append(FUNCTIONCALLOPT(code, func))
39:      case Return
40:        list.append(RETURNOPT(code, func))
41:      case Default// Generate Naive Code
42:        list.append(GENERATECODE(code, func))
43:    end for
44:    list.append("}")// Ending function body
45:  return list
46: end procedure
```