

# AST 구조 분석기 제작

WHS 3기 29반 장인영 (5455)

## 목차

[개요](#)

[ast.json 구조 분석하기](#)

[함수 개수 추출하기](#)

[함수들의 리턴 타입 추출하기](#)

[함수들의 파라미터 타입, 변수명 추출하기](#)

[함수들의 if 조건 개수 추출하기](#)

[main 함수](#)

[전체 코드\(Github 링크\)와 실행 결과](#)

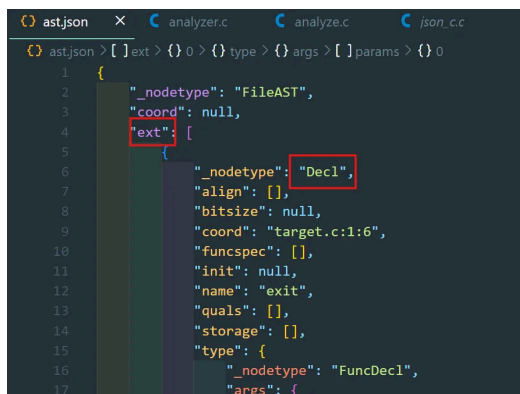
[C언어 함수로 변환하기](#)

## 개요

C 소스 코드를 파싱하여 생성된 추상 구문 트리(AST) JSON 파일을 분석하여, 함수 개수, 이름, 리턴 타입, 파라미터 타입, 변수명, 조건 개수를 자동으로 추출하는 도구를 개발한다.

## ast.json 구조 분석하기

먼저 `ast.json` 파일의 구조를 분석하고, 어떠한 방식으로 함수가 표현되어 있는 지 살펴본다. `ast.json` 은 추상 구문 트리(Abstract Syntax Tree, AST)를 JSON 형식으로 표현한 파일로, `ext` 필드는 전체 프로그램의 선언 또는 정의를 배열 형태로 포함하고 있다.



```
ast.json > [ ] ext > { } 0 > { } type > { } args > [ ] params > { } 0
1 {
2   "_nodetype": "FileAST",
3   "coord": null,
4   "ext": [
5     {
6       "_nodetype": "Decl",
7       "align": [],
8       "bitsize": null,
9       "coord": "target.c:1:6",
10      "funspec": [],
11      "init": null,
12      "name": "exit",
13      "quals": [],
14      "storage": [],
15      "type": {
16        "_nodetype": "FuncDecl",
17        "args": {
```

배열의 각 요소는 다음과 같은 구조를 가지며, 주요 항목에 대해 정리하면 다음과 같다.

```
{
  "_nodetype": "Decl",
  "align": [],
  "bitsize": null,
  "coord": "target.c:1:6",
  "funccspec": [],
  "init": null,
  "name": "exit",
  "quals": [],
  "storage": [],
  "type": {
    "_nodetype": "FuncDecl",
    "args": {
      "_nodetype": "ParamList",
      "coord": "target.c:0:1",
      "params": [

```

- `_nodetype`  
노드의 종류 (Decl, FuncDef, IF 등)
- `name` : 함수 또는 변수의 이름
- `type` : 해당 선언의 타입 구조
- `coord` : 소스 코드 상에서 위치 정보

특히 함수는 두 가지 방식으로 표현된다.

### 1. 함수 선언 ( `FuncDecl` )

```
"type": {
  "_nodetype": "FuncDecl",
  "args": {
    "_nodetype": "ParamList",
    "coord": "target.c:0:1",
    "params": [
      {
        "_nodetype": "Typename",
        "align": null,
        "coord": "target.c:0:1",
        "name": null,
        "quals": [],
        "type": {
          "_nodetype": "TypeDecl",
          "align": null,
          "coord": null,
          "declname": null,
          "quals": [],
          "type": {
            "_nodetype": "IdentifierType",
            "coord": "target.c:1:11",
            "names": [
              "int"
            ]
          }
        }
      ]
    ]
  }
}
```

`_nodetype` 이 `Decl` 이고 `type` 내부에 `FuncDecl` 이 포함되어 있는 경우, 이는 함수의 선언을 의미한다. 이 때 `args` 필드는 파라미터 목록을 포함하며, `type` 내부의 `IdentifierType` 은 리턴 타입을 정의한다.

## 2. 함수 정의 (FuncDef)

```
{
  "_nodetype": "FuncDef",
  "body": {
    "_nodetype": "Compound",
    "block_items": [
      {
        "_nodetype": "Return",
        "coord": "target.c:9:3",
        "expr": {
          "_nodetype": "FuncCall",
          "args": null,
          "coord": "target.c:9:10",
          "name": {
            "_nodetype": "ID",
            "coord": "target.c:9:10",
            "name": "main1"
          }
        }
      }
    ]
  }
}
```

`_nodetype` 이 `FuncDef` 인 노드는 함수 정의 전체를 표현한다.

```
{
  "_nodetype": "FuncDef",
  "body": {
    "_nodetype": "Compound",
    "block_items": [
      {
        "_nodetype": "If",
        "cond": {
          "_nodetype": "BinaryOp",
          "coord": "target.c:36:7",
          "left": {
            "_nodetype": "ID",
            "coord": "target.c:36:7",
            "name": "token_size"
          },
          "op": "<=",
          "right": {
            "_nodetype": "BinaryOp",
            "coord": "target.c:36:21",
            "left": {
              "_nodetype": "ID",
              "coord": "target.c:36:21",
              "name": "i"
            },
            "op": "+",
            "right": {
              "_nodetype": "Constant",
              "coord": "target.c:36:25",
              "type": "int",
              "value": "1"
            }
          }
        },
        "coord": "target.c:36:3",
        "iffalse": null,
        "iftrue": {
          "_nodetype": "Compound",
          "block_items": [
            {
              "_nodetype": "Return",
              "coord": "target.c:36:10",
              "expr": {
                "_nodetype": "ID",
                "coord": "target.c:36:10",
                "name": "main1"
              }
            }
          ]
        }
      }
    ]
  }
}
```

또한 조건문은 `_nodetype` 이 `If` 인 노드로 표현되며, 조건과 참/거짓 분기를 각각 `cond`, `iftrue`, `iffalse` 필드로 표현한다.

이처럼 `ast.json`은 각 요소가 JSON 객체로 표현되어 있어, 코드를 계층적으로 구조화하고 원하는 정보를 정밀하게 추출할 수 있도록 되어 있다.

## 함수 개수 추출하기

AST에서 함수는 함수 정의 ( `FuncDef` ), 함수 선언 ( `FuncDecl` ), 두 가지 방식으로 나타난다. 함수의 개수를 추출하기 위해서는 선언만 존재하는 함수와 실제로 정의된 함수를 모두 포함하여야 한다.

`ext` 배열을 순회하며, `_nodetype` 이 `FuncDef` 인 경우에는 함수 정의, `_nodetype` 이 `Decl` 이고 하위 `type` 의 `_nodetype` 이 `FuncDecl` 인 경우에는 함수 선언으로 분류하여 개수를 파악한다.

```
6  /* 1. 함수 개수 추출 함수 */
7
8  ▼ int count_functions(json_value root) {
9
10     // 함수 개수를 저장할 변수 초기화
11     int count = 0;
12
13     // json 루트 객체에서 "ext"에 해당하는 값 추출
14     json_value ext = json_get(root, "ext");
15
16     // ext 배열의 길이 계산
17     int ext_len = json_len(ext);
18
19     // ext 배열 순회하며 요소 추출
20     ▼ for (int i = 0; i < ext_len; i++) {
21         json_value obj = json_get(ext, i);
22
23         // 현재 객체에서 "_nodetype"에 해당하는 값 추출
24         json_value nodetype = json_get(obj, "_nodetype");
25
26         // case 1 : FuncDef 체크
27         ▼ if (strcmp(json_get_string(nodetype), "FuncDef") == 0) {
28             count++;
29             continue;
30         }
31
32         // case 2 : FuncDecl 체크
33         // 현재 객체에서 "type" 필드 추출
34         json_value type = json_get(obj, "type");
35         // "type" 객체에서 "_nodetype" 필드 추출
36         json_value decl_type = json_get(type, "_nodetype");
37         ▼ if (strcmp(json_get_string(decl_type), "FuncDecl") == 0) {
38             count++;
39         }
40     }
41     return count;
42 }
```

## 함수들의 리턴 타입 추출하기

함수의 리턴 타입은 각 함수 내부 `type` 필드에 중첩 구조로 표현되어 있다. `FuncDecl` 내부 `type` 필드는 다시 `TypeDecl`, `PtrDecl` 등을 포함하며, 최종적으로 `IdentifierType` 에서 리턴 타입 이름을 배열 형태로 제공한다.

`PtrDecl` 은 포인터 타입을 나타내며, `*` 를 결과 문자열 앞에 붙이고 내부 `type` 을 탐색한다.

`ArrayDecl` 은 배열 타입을 나타내며, `[]` 를 뒤에 붙이고 내부 `type` 을 탐색한다.

`TypeDecl` 은 타입 선언 노드로, 내부 `type` 필드를 통해 실제 식별자 타입이나 포인터 구조로 연결된다.

`IdentifierType` 은 `int`, `void` 등 기본 타입을 문자열 배열로 저장한다.

```
44  /* 2. 리턴 타입 추출 함수 */
45
46  void resolve_type(json_value type, char* buffer, int depth) {
47      // type 객체에서 "_nodetype" 필드 추출
48      json_value nodetype = json_get(type, "_nodetype");
49
50      // "_nodetype" 값을 문자열로 변환
51      const char* type_str = json_get_string(nodetype);
52
53      // 1) PtrDecl : 포인터 타입
54      if (strcmp(type_str, "PtrDecl") == 0) {
55          strcat(buffer, "*");
56          resolve_type(json_get(type, "type"), buffer, depth+1);
57      }
58
59      // 2) ArrayDecl : 배열 타입
60      else if (strcmp(type_str, "ArrayDecl") == 0) {
61          strcat(buffer, "[]");
62          resolve_type(json_get(type, "type"), buffer, depth+1);
63      }
64
65      // 3) TypeDecl : 내부 type 필드로 연결
66      else if (strcmp(type_str, "TypeDecl") == 0) {
67          resolve_type(json_get(type, "type"), buffer, depth + 1);
68      }
69
70      // 3) IdentifierType : 기본 타입
71      else if (strcmp(type_str, "IdentifierType") == 0) {
72          // names 배열 추출
73          json_value names = json_get(type, "names");
74
75          // names 배열 길이 계산
76          int name_count = json_len(names);
77
78          // 배열 순회하며 버퍼에 타입 이름 추가
79          for (int i = 0; i < name_count; i++) {
80              json_value name = json_get(names, i);
81              strcat(buffer, json_get_string(name));
82              // 복합 타입일 경우 공백 추가
83              if (i != name_count-1) strcat(buffer, " ");
84          }
85      }
86  }
87
88 }
```

## 함수들의 파라미터 타입, 변수명 추출하기

함수 파라미터는 `ast.json` 의 `FuncDecl` 노드 내부 `args` 필드에 `ParamList` 형태로 정의되어 있으며, 그 안의 `params` 배열이 각 파라미터 정보를 포함하고 있다.

```
"type": {
  "_nodetype": "FuncDecl",
  "args": {
    "_nodetype": "ParamList",
    "coord": "target.c:0:1",
    "params": [
      {
        "_nodetype": "Typename",
        "align": null,
        "coord": "target.c:0:1",
        "name": null,
        "quals": [],
        "type": {
          "_nodetype": "TypeDecl",
          "align": null,
          "coord": null,
          "declname": null,
          "quals": [],
          "type": {
            "_nodetype": "IdentifierType",
            "coord": "target.c:1:11",
            "names": [
              "int"
            ]
          }
        }
      }
    ]
  }
},
```

각 파라미터는 `Typename` 노드로 표현되며, 그 내부에 `type` 필드와 `name` 필드를 갖는다. 이러한 구조에서 함수 파라미터 정보를 다음과 같이 추출할 수 있다.

```
90 /* 3. 파라미터 추출 함수 */
91
92 void get_parameters(json_value params, char* result) {
93
94     // 파라미터 개수
95     int param_count = json_len(params);
96
97     // 모든 파라미터 순회
98     for (int i = 0; i < param_count; i++) {
99         json_value param = json_get(params, i);
100
101         // 파라미터 타입을 저장할 버퍼
102         char type_buf[256] = {0};
103
104         // 파라미터 이름을 저장할 버퍼
105         char name_buf[128] = {0};
106
107         // 타입 추출
108         json_value type = json_get(param, "type");
109         resolve_type(type, type_buf, 0);
110
111         // 이름 추출
112         json_value name = json_get(param, "name");
113         if (name.type == JSON_STRING) {
114             strncpy(name_buf, json_get_string(name), sizeof(name_buf)-1);
115         } else {
116             sprintf(name_buf, sizeof(name_buf), "param%d", i); // 이름 없으면 기본 이름 사용
117         }
118         // 버퍼가 넘치지 않을 경우에만 결과 문자열에 추가
119         if (strlen(result) + strlen(type_buf) + strlen(name_buf) < 510) {
120             // 형식: "타입 이름" 추가
121             sprintf(result + strlen(result), "%s %s", type_buf, name_buf);
122             // 마지막 파라미터가 아니면 필요 추가
123             if (i != param_count-1) strcat(result, ", ");
124         }
125     }
126 }
127
```

## 함수들의 if 조건 개수 추출하기

함수 내부에서 사용된 `if` 조건문의 개수를 파악하기 위해, `FuncDef` 노드에 포함된 `body` 필드를 분석한다. `_nodetype` 값이 `if` 인 노드를 탐색할 때마다 개수가 증가하는 방식으로 구현한다.

```
128  /* 4. IF 조건 개수 추출 함수 */
129
130  // 키 존재 여부를 확인하는 json_has_key() 함수 추가
131  bool json_has_key(json_value obj, const char* key) {
132      if (obj.type != JSON_OBJECT) return false;
133      json_object* jsobj = (json_object*)obj.value;
134      for (int i = 0; i <= jsobj->last_index; i++) {
135          if (strcmp(jsobj->keys[i], key) == 0) return true;
136      }
137      return false;
138  }
139
140  void count_if_recursive(json_value node, int* count) {
141      // 현재 노드가 if인 경우
142      if (json_has_key(node, "_nodetype")) {
143          json_value nodetype = json_get(node, "_nodetype");
144          if (nodetype.type == JSON_STRING && strcmp(json_get_string(nodetype), "If") == 0) {
145              (*count)++;
146          }
147      }
148
149      // 객체인 경우: 모든 키의 값에 대해 재귀 탐색
150      if (node.type == JSON_OBJECT) {
151          json_object* obj = (json_object*)node.value;
152          for (int i = 0; i <= obj->last_index; i++) {
153              json_value child = obj->values[i];
154              count_if_recursive(child, count);
155          }
156      }
157
158      // 배열인 경우: 각 요소에 대해 재귀 탐색
159      if (node.type == JSON_ARRAY) {
160          int len = json_len(node);
161          for (int i = 0; i < len; i++) {
162              json_value child = json_get(node, i);
163              count_if_recursive(child, count);
164          }
165      }
166  }
167
168  // 주어진 함수에서 if 조건문의 개수를 계산하는 함수
169  int count_if_conditions(json_value func) {
170      int count = 0;
171      count_if_recursive(func, &count);
172      return count;
173  }
174
```

## main 함수

main 함수는 다음과 같이 작성하였다.

```
175
176 int main(void) {
177     /* JSON 파일 읽기 */
178
179     FILE *fp = fopen("ast.json", "r");
180     if (fp == NULL) {
181         printf("파일을 열 수 없습니다.\n");
182         return 1;
183     }
184
185     /* 파일 크기 측정 및 메모리 할당 */
186
187     fseek(fp, 0, SEEK_END); // 파일 끝으로 이동
188     long file_size = ftell(fp); // 파일 크기 가져오기
189     rewind(fp); // 파일 포인터 다시 처음으로
190
191     char *file_buff = (char *)malloc(file_size + 1);
192     if (!file_buff) {
193         fclose(fp);
194         return 1;
195     }
196
197     /* 파일 읽기 */
198
199     fread(file_buff, 1, file_size, fp);
200     file_buff[file_size] = '\0';
201
202     /* 파일 읽기 */
203
204     fclose(fp);
205
206     /* JSON 파싱 시작 */
207
208     json_value json = json_create(file_buff);
209     if (json.type != JSON_OBJECT) {
210         printf("잘못된 JSON 형식\n");
211         free(file_buff);
212         return 1;
213     }
214
215     /* 1. 함수 개수 출력 */
216
217     int total_func = count_functions(json);
218     printf("Number of Functions : %d\n\n", total_func);
219
220
```

```
/* 파라미터 처리 */
char params[512] = "";
int has_params = 0;

json_value args = (0); // args 정보 구조체

if (strcmp(nodetype_str, "FuncDef") == 0) {
    json_value decl = json_get(func, "decl");
    json_value decl_type = json_get(decl, "type");

    json_value decl_type_tag = json_get(decl_type, "_nodetype");
    if (strcmp(json_get_string(decl_type_tag, "FuncDecl") == 0) {
        args = json_get(decl_type, "args");
    }
} else {
    json_value func_type = json_get(func, "type");
    json_value func_type_tag = json_get(func_type, "_nodetype");
    if (func_type_tag == JSON_STRING &&
        strcmp(json_get_string(func_type_tag, "FuncDecl") == 0) {
        args = json_get(func_type, "args");
    }
}

/* 파라미터 목록 추출 */
if (args.type == JSON_OBJECT) {
    json_value params_val = json_get(args, "params");
    if (params_val.type == JSON_ARRAY) {
        get_parameters(params_val, params);
        has_params = 1;
    }
}

/* IF 조건문 개수 출력 */
int if_count = 0;
if (nodetype_str && strcmp(nodetype_str, "FuncDef") == 0) {
    if_count = count_if_conditions(func);
}

/* 결과 출력 */
printf("Function Name : %s\n", func_name);
printf("Return Type : %s\n", return_type);
printf("Parameter : %s\n", has_params ? params : "None");
printf("IF Count : %d\n\n", if_count);
}

/* 메모리 해제 */
json_free(json); // JSON 파싱을 끝낸 후 해제
free(file_buff); // 파일 읽기를 위한 메모리 해제

return 0;
}
```

```
/* 2. 함수 이름, 반환 타입, 파라미터 추출 */
124 json_value ext = json_get(json, "ext"); // 함수 목록을 'ext' 배열 안에 저장
125 if (ext.type == JSON_ARRAY) {
126     int ext_len = json_get_length(ext); // 배열 길이 구하기
127
128     // 배열 내 모든 함수 정보 또는 파라미터 정보
129     for (int i = 0; i < ext_len; i++) {
130         json_value func = json_get(ext, i);
131         if (func.type != JSON_OBJECT) continue; // 객체가 아닌 경우 skip
132
133         json_value nodetype = json_get(func, "_nodetype");
134         const char *nodetype_str = NULL;
135         if (nodetype.type == JSON_STRING) {
136             nodetype_str = json_get_string(nodetype);
137         }
138
139         char func_name[128] = "미함"; // 기본 함수 이름
140         char return_type[128] = ""; // 반환 타입 초기화
141
142         json_value type_node = (0); // 반환 타입 추출을 위한 노드
143
144         /* 함수 정보 (Function) */
145         if (strcmp(nodetype_str, "FuncDef") == 0) {
146             json_value decl = json_get(func, "decl"); // decl 안에 함수 정보
147             json_value name = json_get(decl, "name"); // 이름 추출
148             if (name.type == JSON_STRING) {
149                 sprintf(func_name, sizeof(func_name), "%s", json_get_string(name));
150             }
151             type_node = json_get(decl, "type"); // 반환 타입을 나타내는 노드
152         }
153         /* 함수 선언 (Decl) */
154         else {
155             json_value name = json_get(func, "name");
156             if (name.type == JSON_STRING) {
157                 sprintf(func_name, sizeof(func_name), "%s", json_get_string(name));
158             }
159             type_node = json_get(func, "type");
160         }
161
162         /* 반환 타입 처리 */
163         if (type_node.type != JSON_UNDEFINED) {
164             json_value nodetype = json_get(type_node, "_nodetype");
165             const char *ntype = json_get_string(nodetype);
166
167             if (strcmp(nodetype, "FuncDecl") == 0) {
168                 type_node = json_get(type_node, "type"); // 내부 반환 타입을 나타내기
169             }
170
171             resolve_type(type_node, return_type, 0); // 반환 타입을 결정
172         } else {
173             strcpy(return_type, "void"); // 기본값을 void 처리
174         }
175     }
176 }
```



## 전체 코드(Github 링크)와 실행 결과

[https://github.com/inyeongjang/WHS\\_AST/blob/main/analyzer.c](https://github.com/inyeongjang/WHS_AST/blob/main/analyzer.c)

```
Number of Functions : 42

Funtion Name : exit
Return Type : void
Parameter : int param0
IF Count: 0

Funtion Name : getchar
Return Type : int
Parameter : void param0
IF Count: 0

Funtion Name : malloc
Return Type : *void
Parameter : int param0
IF Count: 0

Funtion Name : putchar
Return Type : int
Parameter : int param0
IF Count: 0

Funtion Name : main1
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : main
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : my_realloc
Return Type : *char
Parameter : *char old, int oldlen, int newlen
IF Count: 0
```

```
Funtion Name : nextc
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : token
Return Type : *char
Parameter : None
IF Count: 0

Funtion Name : token_size
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : error
Return Type : void
Parameter : None
IF Count: 0

Funtion Name : i
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : takechar
Return Type : void
Parameter : None
IF Count: 1

Funtion Name : get_token
Return Type : void
Parameter : None
IF Count: 7
```

```
Funtion Name : table_pos
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : stack_pos
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : sym_lookup
Return Type : int
Parameter : *char s
IF Count: 1

Funtion Name : sym_declare
Return Type : void
Parameter : *char s, int type, int value
IF Count: 1

Funtion Name : sym_declare_global
Return Type : int
Parameter : *char s
IF Count: 1

Funtion Name : sym_define_global
Return Type : void
Parameter : int current_symbol
IF Count: 1

Funtion Name : number_of_args
Return Type : int
Parameter : None
IF Count: 0
```

```
Funtion Name : sym_get_value
Return Type : void
Parameter : *char s
IF Count: 5

Funtion Name : be_start
Return Type : void
Parameter : None
IF Count: 0

Funtion Name : be_finish
Return Type : void
Parameter : None
IF Count: 0

Funtion Name : promote
Return Type : void
Parameter : int type
IF Count: 2

Funtion Name : expression
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : primary_expr
Return Type : int
Parameter : None
IF Count: 9

Funtion Name : binary1
Return Type : void
Parameter : int type
IF Count: 0
```

```

Funtion Name : peek
Return Type : int
Parameter : *char s
IF Count: 0

Funtion Name : accept
Return Type : int
Parameter : *char s
IF Count: 1

Funtion Name : expect
Return Type : void
Parameter : *char s
IF Count: 1

Funtion Name : code
Return Type : *char
Parameter : None
IF Count: 0

Funtion Name : code_size
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : codepos
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : code_offset
Return Type : int
Parameter : None
IF Count: 0

```

```

Return Type : void
Parameter : *char p, int n
IF Count: 0

Funtion Name : load_int
Return Type : int
Parameter : *char p
IF Count: 0

Funtion Name : emit
Return Type : void
Parameter : int n, *char s
IF Count: 1

Funtion Name : be_push
Return Type : void
Parameter : None
IF Count: 0

Funtion Name : be_pop
Return Type : void
Parameter : int n
IF Count: 0

Funtion Name : table
Return Type : *char
Parameter : None
IF Count: 0

Funtion Name : table_size
Return Type : int
Parameter : None
IF Count: 0

```

```

Funtion Name : bitwise_or_expr
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : expression
Return Type : int
Parameter : None
IF Count: 2

Funtion Name : type_name
Return Type : void
Parameter : None
IF Count: 0

Funtion Name : statement
Return Type : void
Parameter : None
IF Count: 8

Funtion Name : program
Return Type : void
Parameter : None
IF Count: 4

Funtion Name : main1
Return Type : int
Parameter : None
IF Count: 0

```

```

Funtion Name : binary2
Return Type : int
Parameter : int type, int n, *char s
IF Count: 0

Funtion Name : postfix_expr
Return Type : int
Parameter : None
IF Count: 3

Funtion Name : additive_expr
Return Type : int
Parameter : None
IF Count: 2

Funtion Name : shift_expr
Return Type : int
Parameter : None
IF Count: 2

Funtion Name : relational_expr
Return Type : int
Parameter : None
IF Count: 0

Funtion Name : equality_expr
Return Type : int
Parameter : None
IF Count: 2

Funtion Name : bitwise_and_expr
Return Type : int
Parameter : None
IF Count: 0

```

## C언어 함수로 변환하기

앞선 단계에서 각 함수에 대한 핵심 정보를 추출하였다. 이를 종합하여 상위 5개 요소에 대해 C 언어 함수 형태로 재구성하는 작업을 수행한다.

### 1. `void exit(int);`

`"_nodetype": "Decl"` 이므로 함수 선언 노드이고, `"name": "exit"` 이므로 함수의 이름은 `exit` 이다. 파라미터는 `int` 타입으로 정의되어 있으며, 반환 타입은 `void` 이다. 따라서, C언어로 나타내면 `void exit(int);` 와 같다.

```
{
  /* 선언 */
  "_nodetype": "Decl",

  "align": [],
  "bitsize": null,
  "coord": "target.c:1:6",
  "funccspec": [],
  "init": null,

  /* 함수 이름 : exit */
  "name": "exit",

  "quals": [],
  "storage": [],

  /* 함수의 타입 정보 */
  "type": {

    /* 함수 선언 */
    "_nodetype": "FuncDecl",
    "args": {
      "_nodetype": "ParamList",
      "coord": "target.c:0:1",

      /* 파라미터 */
      "params": [
        {
          "_nodetype": "Typename",
          "align": null,
```

```

        "coord": "target.c:0:1",
        "name": null,
        "quals": [],
        "type": {
            "_nodetype": "TypeDecl",
            "align": null,
            "coord": null,
            "declname": null,
            "quals": [],

            /* int 타입 */
            "type": {
                "_nodetype": "IdentifierType",
                "coord": "target.c:1:11",
                "names": [
                    "int"
                ]
            }
        }
    ]
},
"coord": "target.c:1:6",

/* 반환 타입 */
"type": {
    "_nodetype": "TypeDecl",
    "align": null,
    "coord": "target.c:1:6",
    "declname": "exit",
    "quals": [],

    /* void */
    "type": {
        "_nodetype": "IdentifierType",
        "coord": "target.c:1:1",
        "names": [
            "void"
        ]
    }
}

```

```

    }
  }
}

```

## 2. `int getchar(void);`

`"_nodetype": "Decl"` 이므로 함수 선언 노드이고, `"name": "getchar"` 이므로 함수의 이름은 `getchar` 이다. 파라미터는 `void` 타입으로 정의되어 있으며, 반환 타입은 `int` 이다. 따라서, C언어로 나타내면 `int getchar(void);` 와 같다.

```

{
  /* 선언 */
  "_nodetype": "Decl",
  "align": [],
  "bitsize": null,
  "coord": "target.c:2:5",
  "funccspec": [],
  "init": null,

  /* 함수 이름 : getchar */
  "name": "getchar",
  "quals": [],
  "storage": [],

  /* 함수의 타입 정보 */
  "type": {

    /* 함수 선언 */
    "_nodetype": "FuncDecl",
    "args": {
      "_nodetype": "ParamList",
      "coord": "target.c:0:1",

      /* 파라미터 */
      "params": [
        {
          "_nodetype": "Typename",
          "align": null,
          "coord": "target.c:0:1",

```

```

        "name": null,
        "quals": [],
        "type": {
            "_nodetype": "TypeDecl",
            "align": null,
            "coord": null,
            "declname": null,
            "quals": [],

            /* void 타입 */
            "type": {
                "_nodetype": "IdentifierType",
                "coord": "target.c:2:13",
                "names": [
                    "void"
                ]
            }
        }
    },
    "coord": "target.c:2:5",

    /* 반환 타입 */
    "type": {
        "_nodetype": "TypeDecl",
        "align": null,
        "coord": "target.c:2:5",
        "declname": "getchar",
        "quals": [],

        /* int 타입 */
        "type": {
            "_nodetype": "IdentifierType",
            "coord": "target.c:2:1",
            "names": [
                "int"
            ]
        }
    }
}

```

```
}
}
```

### 3. `void* malloc(int);`

`"_nodetype": "Decl"` 이므로 함수 선언 노드이고, `"name": "malloc"` 이므로 함수의 이름은 `malloc` 이다. 파라미터는 `int` 타입으로 정의되어 있으며, 반환 타입은 `void*` 이다. 따라서, C언어로 나타내면 `void* malloc(int);` 와 같다.

```
{
    /* 선언 */
    "_nodetype": "Decl",
    "align": [],
    "bitsize": null,
    "coord": "target.c:3:7",
    "funccspec": [],
    "init": null,

    /* 함수 이름 : malloc */
    "name": "malloc",
    "quals": [],
    "storage": [],

    /* 함수의 타입 정보 */
    "type": {

        /* 함수 선언 */
        "_nodetype": "FuncDecl",
        "args": {
            "_nodetype": "ParamList",
            "coord": "target.c:0:1",

            /* 파라미터 */
            "params": [
                {
                    "_nodetype": "Typename",
                    "align": null,
                    "coord": "target.c:0:1",
                    "name": null,
```

```

    "quals": [],

    "type": {
        "_nodetype": "TypeDecl",
        "align": null,
        "coord": null,
        "declname": null,
        "quals": [],

        /* int 타입 */
        "type": {
            "_nodetype": "IdentifierType",
            "coord": "target.c:3:14",
            "names": [
                "int"
            ]
        }
    }
}
]
},
"coord": "target.c:3:7",

/* 포인터 타입 */
"type": {
    "_nodetype": "PtrDecl",
    "coord": "target.c:3:6",
    "quals": [],
    "type": {
        "_nodetype": "TypeDecl",
        "align": null,
        "coord": "target.c:3:7",
        "declname": "malloc",
        "quals": [],

        /* void타입 */
        "type": {
            "_nodetype": "IdentifierType",
            "coord": "target.c:3:1",
            "names": [

```



```

        "void"
      ]
    }
  }
}
}

```

4. `int main() { return main1(); }`

```

{
  /* 함수 정의 */
  "_nodetype": "FuncDef",

  /* 함수 본문 */
  "body": {
    "_nodetype": "Compound",
    "block_items": [
      {
        /* return */
        "_nodetype": "Return",

        "coord": "target.c:9:3",
        "expr": {
          "_nodetype": "FuncCall",

          /* 인자 없음 */
          "args": null,
          "coord": "target.c:9:10",

          /* 함수 이름 : main1 → return main1(); */
          "name": {
            "_nodetype": "ID",
            "coord": "target.c:9:10",
            "name": "main1"
          }
        }
      }
    ]
  }
}

```

```

    ],
    "coord": "target.c:8:1"
  },
  "coord": "target.c:7:5",

  /* 함수 선언 */
  "decl": {
    "_nodetype": "Decl",
    "align": [],
    "bitsize": null,
    "coord": "target.c:7:5",
    "funccspec": [],
    "init": null,

    /* 함수 이름 : main */
    "name": "main",
    "quals": [],
    "storage": [],

    /* 함수 타입 정의 */
    "type": {

      /* 함수 선언 */
      "_nodetype": "FuncDecl",

      /* 인자 없음 */
      "args": null,
      "coord": "target.c:7:5",

      /* main */
      "type": {
        "_nodetype": "TypeDecl",
        "align": null,
        "coord": "target.c:7:5",
        "declname": "main",
        "quals": [],

        /* int 타입 */
        "type": {
          "_nodetype": "IdentifierType",

```

```

        "coord": "target.c:7:1",
        "names": [
            "int"
        ]
    }
}
},
"param_decls": null
},

```

5. `char *my_realloc(char *old, int oldlen, int newlen) {`  
`char *new = malloc(newlen);`  
`int i = 0;`  
`while (i <= oldlen - 1) {`  
`new[i] = old[i];`  
`i = i + 1;`  
`}`  
`return new;`  
`}`

```

{
    /* 함수 정의 */
    "_nodetype": "FuncDef",

    /* 함수 본문 */
    "body": {
        "_nodetype": "Compound",
        "block_items": [

            /* char *new = malloc(newlen); */
            {
                "_nodetype": "Decl",
                "init": {
                    "_nodetype": "FuncCall",
                    "name": {
                        "_nodetype": "ID",

```

```

    "name": "malloc"
  },
  "args": {
    "_nodetype": "ExprList",
    "exprs": [
      {
        "_nodetype": "ID",
        "name": "newlen"
      }
    ]
  }
},
"name": "new",
"type": {
  "_nodetype": "PtrDecl",
  "type": {
    "_nodetype": "TypeDecl",
    "declname": "new",
    "type": {
      "_nodetype": "IdentifierType",
      "names": ["char"]
    }
  }
}
},
/* int i = 0; */
{
  "_nodetype": "Decl",
  "init": {
    "_nodetype": "Constant",
    "type": "int",
    "value": "0"
  },
  "name": "i",
  "type": {
    "_nodetype": "TypeDecl",
    "declname": "i",
    "type": {
      "_nodetype": "IdentifierType",

```

```

        "names": ["int"]
    }
}
},

/* while (i <= oldlen - 1) { new[i] = old[i]; i = i + 1; } */
{
    "_nodetype": "While",
    "cond": {
        "_nodetype": "BinaryOp",
        "op": "<=",
        "left": { "_nodetype": "ID", "name": "i" },
        "right": {
            "_nodetype": "BinaryOp",
            "op": "-",
            "left": { "_nodetype": "ID", "name": "oldlen" },
            "right": { "_nodetype": "Constant", "type": "int", "value": "1" }
        }
    },
    "stmt": {
        "_nodetype": "Compound",
        "block_items": [

            /* new[i] = old[i]; */
            {
                "_nodetype": "Assignment",
                "op": "=",
                "lvalue": {
                    "_nodetype": "ArrayRef",
                    "name": { "_nodetype": "ID", "name": "new" },
                    "subscript": { "_nodetype": "ID", "name": "i" }
                },
                "rvalue": {
                    "_nodetype": "ArrayRef",
                    "name": { "_nodetype": "ID", "name": "old" },
                    "subscript": { "_nodetype": "ID", "name": "i" }
                }
            },

            /* i = i + 1; */

```

```

    {
      "_nodetype": "Assignment",
      "op": "=",
      "lvalue": { "_nodetype": "ID", "name": "i" },
      "rvalue": {
        "_nodetype": "BinaryOp",
        "op": "+",
        "left": { "_nodetype": "ID", "name": "i" },
        "right": { "_nodetype": "Constant", "type": "int", "value": "1" }
      }
    }
  ]
}
},

/* return new; */
{
  "_nodetype": "Return",
  "expr": { "_nodetype": "ID", "name": "new" }
}
]
},

/* 함수 선언 부분 (타입, 이름, 매개변수) */
"decl": {
  "_nodetype": "Decl",
  "name": "my_realloc",
  "type": {
    "_nodetype": "FuncDecl",
    "args": {
      "_nodetype": "ParamList",
      "params": [

        /* char* old */
        {
          "_nodetype": "Decl",
          "name": "old",
          "type": {
            "_nodetype": "PtrDecl",

```

```

    "type": {
      "_nodetype": "TypeDecl",
      "declname": "old",
      "type": {
        "_nodetype": "IdentifierType",
        "names": ["char"]
      }
    }
  },
  /* int oldlen */
  {
    "_nodetype": "Decl",
    "name": "oldlen",
    "type": {
      "_nodetype": "TypeDecl",
      "declname": "oldlen",
      "type": {
        "_nodetype": "IdentifierType",
        "names": ["int"]
      }
    }
  },
  /* int newlen */
  {
    "_nodetype": "Decl",
    "name": "newlen",
    "type": {
      "_nodetype": "TypeDecl",
      "declname": "newlen",
      "type": {
        "_nodetype": "IdentifierType",
        "names": ["int"]
      }
    }
  }
]

```

```

    },

    /* 반환 타입: char* */
    "type": {
        "_nodetype": "PtrDecl",
        "type": {
            "_nodetype": "TypeDecl",
            "declname": "my_realloc",
            "type": {
                "_nodetype": "IdentifierType",
                "names": ["char"]
            }
        }
    }
}
}
}
}
}
}
}

```