✅ [WHS3기][프로그래밍기초][김신아]_[9502]

🔍 AST 구조 분석기란?

- 작성한 c코드 같은 소스코드를 구조적으로 표현한 트리
- 함수 정보, 조건문, 변수 등을 구조적으로 파악할 수 있다.

🛠️ 분석기 구현 방식

- c코드 분석기를 만들 때 직접 파싱 로직을 만들 수도 있지만, 기존의 c 파서를 활용하는 것도 좋다.

- 과제 조건

1. 함수 개수 추출
2. 함수들의 리턴 타입 추출
3. 파라미터 타입과 변수명 추출
4. if 조건문 개수 추출

---

1. 헤더와 함수 정의

```
#include <stdio.h>
#include <string.h>
#include "json_c.c"
```

2. 조건문 개수 세는 재귀 함수

```
int count_ifs(json_value node) {
    int count = 0;
    if (json_get_type(node) == JSON_OBJECT) {
        char* nodetype = json_get_string(json_get(node, "_nodetype"));
        if (strcmp(nodetype, "If") == 0) count++;

        int len = json_get_last_index(node) + 1;
        for (int i = 0; i < len; i++) {
            json_value child = json_get_from_object((json_object*)node.value, ((json_object*)node.valu
            count += count_ifs(child);
        }
    } else if (json_get_type(node) == JSON_ARRAY) {
        int len = json_len(node);
        for (int i = 0; i < len; i++) {
            count += count_ifs(json_get(node, i));
        }
    }
    return count;
}
```

3. 메인 함수

```
int main() {
    json_value ast = json_read("ast.json");
    json_value ext = json_get(ast, "ext");
    int len = json_len(ext);
```

4. 표 머리 출력

```
printf("──────────────────────────────────────────────────────────────────────────────────────────
    printf("함수이름      │ 리턴타입   │ 파라미터 목록                 │ if 개수\n");
    printf("──────────────────────────────────────────────────────────────────────────────────────────
```

5. 함수 노드 순회

```
for (int i = 0; i < len; i++) {
        json_value item = json_get(ext, i);
        char* nodetype = json_get_string(json_get(item, "_nodetype"));
        if (!(strcmp(nodetype, "FuncDef") == 0 || strcmp(nodetype, "Decl") == 0)) continue;
```

6. 함수 이름 추출

```
json_value decl = strcmp(nodetype, "FuncDef") == 0 ? json_get(item, "decl") : item;
        char* fname = json_get_string(json_get(decl, "name"));
```

7. 리턴 타입 추출

```
// 리턴타입
        json_value type = json_get(decl, "type");
        while (json_get_type(type) != JSON_UNDEFINED &&
               strcmp(json_get_string(json_get(type, "_nodetype")), "IdentifierType") != 0) {
            type = json_get(type, "type");
        }
        char* rettype = json_get_string(json_get(json_get(type, "names"), 0));
```

8. 파라미터 추출

```
// 파라미터
        json_value ftype = json_get(decl, "type");
        json_value args = json_get(ftype, "args");
        char param_list[256] = "";
        if (json_get_type(args) == JSON_OBJECT) {
            json_value params = json_get(args, "params");
            int plen = json_len(params);
            for (int j = 0; j < plen; j++) {
                json_value param = json_get(params, j);
                json_value ptype = json_get(param, "type");
                while (strcmp(json_get_string(json_get(ptype, "_nodetype")), "IdentifierType") != 0) {
                    ptype = json_get(ptype, "type");
                }
                char* ptype_str = json_get_string(json_get(json_get(ptype, "names"), 0));
                char* pname = json_get_type(json_get(param, "name")) == JSON_STRING
                            ? json_get_string(json_get(param, "name")) : "anon";
                char temp[64];
                sprintf(temp, "%s %s", ptype_str, pname);
                strcat(param_list, temp);
                if (j != plen - 1) strcat(param_list, ", ");
            }
        }
```

9. if 조건문 개수 계산

```
        // if 개수
        int if_count = 0;
        if (strcmp(nodetype, "FuncDef") == 0) {
```

```
            if_count = count_ifs(json_get(item, "body"));
        }

        printf("%-13s| %-10s| %-30s| %d\n", fname, rettype, param_list, if_count);
    }
```

analyzer.c 파일

```
#include <stdio.h>
#include <string.h>
#include "json_c.c"

int count_ifs(json_value node) {
    int count = 0;
    if (json_get_type(node) == JSON_OBJECT) {
        char* nodetype = json_get_string(json_get(node, "_nodetype"));
        if (strcmp(nodetype, "If") == 0) count++;

        int len = json_get_last_index(node) + 1;
        for (int i = 0; i < len; i++) {
            json_value child = json_get_from_object((json_object*)node.value, ((json_object*)node.valu
            count += count_ifs(child);
        }
    } else if (json_get_type(node) == JSON_ARRAY) {
        int len = json_len(node);
        for (int i = 0; i < len; i++) {
            count += count_ifs(json_get(node, i));
        }
    }
    return count;
}

int main() {
    json_value ast = json_read("ast.json");
    json_value ext = json_get(ast, "ext");
    int len = json_len(ext);

    printf("──────────────────────────────────────────────────────────────────────────────────────
    printf("함수이름      | 리턴타입   | 파라미터 목록                | if 개수\n");
    printf("──────────────────────────────────────────────────────────────────────────────────────

    for (int i = 0; i < len; i++) {
        json_value item = json_get(ext, i);
        char* nodetype = json_get_string(json_get(item, "_nodetype"));
        if (!(strcmp(nodetype, "FuncDef") == 0 || strcmp(nodetype, "Decl") == 0)) continue;

        json_value decl = strcmp(nodetype, "FuncDef") == 0 ? json_get(item, "decl") : item;
```

```
        char* fname = json_get_string(json_get(decl, "name"));

        // 리턴타입
        json_value type = json_get(decl, "type");
        while (json_get_type(type) != JSON_UNDEFINED &&
               strcmp(json_get_string(json_get(type, "_nodetype")), "IdentifierType") != 0) {
            type = json_get(type, "type");
        }
        char* rettype = json_get_string(json_get(json_get(type, "names"), 0));

        // 파라미터
        json_value ftype = json_get(decl, "type");
        json_value args = json_get(ftype, "args");
        char param_list[256] = "";
        if (json_get_type(args) == JSON_OBJECT) {
            json_value params = json_get(args, "params");
            int plen = json_len(params);
            for (int j = 0; j < plen; j++) {
                json_value param = json_get(params, j);
                json_value ptype = json_get(param, "type");
                while (strcmp(json_get_string(json_get(ptype, "_nodetype")), "IdentifierType") != 0) {
                    ptype = json_get(ptype, "type");
                }
                char* ptype_str = json_get_string(json_get(json_get(ptype, "names"), 0));
                char* pname = json_get_type(json_get(param, "name")) == JSON_STRING
                            ? json_get_string(json_get(param, "name")) : "anon";
                char temp[64];
                sprintf(temp, "%s %s", ptype_str, pname);
                strcat(param_list, temp);
                if (j != plen - 1) strcat(param_list, ", ");
            }
        }

        // if 개수
        int if_count = 0;
        if (strcmp(nodetype, "FuncDef") == 0) {
            if_count = count_ifs(json_get(item, "body"));
        }

        printf("%-13s| %-10s| %-30s| %d\n", fname, rettype, param_list, if_count);
    }

    return 0;
}
```

결과 도출

함수이름	리턴타입	파라미터 목록	if 개수
exit	void	int anon	0
getchar	int	void anon	0
malloc	void	int anon	0
putchar	int	int anon	0
main1	int		0
main	int		0
my_realloc	char	char old, int oldlen, int newlen	0
nextc	int		0
token	char		0
token_size	int		0
error	void		0
i	int		0
takechar	void		1
get_token	void		7
peek	int	char s	0
accept	int	char s	1
expect	void	char s	1
code	char		0
code_size	int		0
codepos	int		0
code_offset	int		0
save_int	void	char p, int n	0
load_int	int	char p	0
emit	void	int n, char s	1
be_push	void		0
be_pop	void	int n	0
table	char		0
table_size	int		0
table_pos	int		0
stack_pos	int		0
sym_lookup	int	char s	1
sym_declare	void	char s, int type, int value	1
sym_declare_global	int	char s	1
sym_define_global	void	int current_symbol	1
number_of_args	int		0
sym_get_value	void	char s	5
be_start	void		0
be_finish	void		0
promote	void	int type	2
expression	int		0
primary_expr	int		9
binary1	void	int type	0
binary2	int	int type, int n, char s	0
postfix_expr	int		3
additive_expr	int		2

```
| shift_expr          | int        |                          |       2 |
| relational_expr     | int        |                          |       0 |
| equality_expr       | int        |                          |       2 |
| bitwise_and_expr    | int        |                          |       0 |
| bitwise_or_expr     | int        |                          |       0 |
| expression          | int        |                          |       2 |
| type_name           | void       |                          |       0 |
| statement           | void       |                          |       8 |
| program             | void       |                          |       4 |
```