개요

test.c파일은 현재 전에 학교에서 진행이 됐던 시스템프로그래밍과제에서 풀었던 코드에 대한 정상작동 여부와 오류를 확인하는 코드이다. 따라서 이번 프로그래밍 기초 과제때 ast분석 대상이 될 파일은 test.c 파일이며, generate_ast.py은 c파일의 AST를 추출해주는 python기반 파일이고, json_c.c는 ast를 추출한 json을 분석하기 위한 프로그램인 수업때도사용했던 ast.c의 라이브러리 파일이다.

AST구조 분석

AST구조를 필요한 부분에 한에서만 조금 분석을 해본결과, 각 객체들은 처음에는 무조건 "_nodetype" 라는 데이터 이름이 존재한다는 것이다.

가장 root부분을 보자면 객체안에 _nodetype, coord, ext 데이터이름이 존재한다. _nodetype은 현재 node의 타입을 설명해주고, coord는 해당 코드의 위치를 설명해주고, ext는 나의 생각에는 extantion의 약자 확장으로 해당 node안에 존재하는 여러 객체들이 들어가 있다.

해당 과제는 함수의 이름, 리턴타입, 파라미터 타입 이름, if사용 수 에 대한 것이므로 ext안에 해당 부분에 대한 내용들만 조금 분석해보겠다.

FuncDef

ext안 객체중 _nodetype이 `FuncDef`라는 것이 있다. 해당 객체는 함수를 선언하는 객체이다. 해당 객체안에 데이터은 `body`, `coord`, `decl`이 존재한다. body는 선언한 함수안에 어떠한 내용들이 존재하는지 위에 봤던 ext와 비슷한 역할을 하고, decl은 선언한 해당 함수에대한 정보들이 담겨 있다.

해당 함수 decl객체를 더 보면 여러 데이터들이 있고, 또 많은 객체들이 존재한다. 여기서 `name`데이터가 존재하는데 여기안에 해당 함수에 대한 이름이 들어간다. 밑에 많은 객체들중에 _nodetype이 ParamList라는 객체가 보인다. 이름처럼 이곳에는 해당 함수의 파라미터들의 정보가 있다. 따라서 params라는 데이터 객들을 보면 각 객체들마다 name과 type들이 존재한다. 여기서 name은 파라미터의 이름이고 type은 해당 파라미터의 type이 있는데 해당 type객체 안 _nodetype안에 PtrDecl이라는 값이 있으면 해당 파라미터는 포인터 타입이라는 뜻이고, ArrayDecl이라는 값이 있으면 해당 파라미터는 배열이라는 뜻이다. 만약 두개 모두 존재하면 해당 파라미터는 배열의 포인터타입이라는 뜻이다. 자료형은 해당 type들을 들어가다보면 _nodetype에 IdentifierType이라는 것이 존재하는데 해당 데이터들중 names값에 파라미터의 자료형이 저장되어 있다. 그리고 return을 할 때 타입은 decl에 존재한다.

If

다시 ext객체로 돌아와 _nodetype을 보면 If라는 객체가 존재한다는 것을 확인할 수 있다. 해당 객체안에 iffalse와 iftrue라는 데이터 이름이 존재하는 것을 확인할 수 있는데 이름처럼 해당 데이터는 if문의 조건식이 참일 경우와 거짓일 경우에 대한 실행코드들의 대한 정보들이 저장되어있다. ifture의 객체에서 _nodetype이 만약 Compound라면 해당 객체안에 여러 구문들이 존재한다는 것이다. 이때 block_items라는 객체도 함께 존재하게 된다.

For

_nodetype이 For일 경우 stmt라는 객체안에 _nodetype이 Compound일 경우 위 If에서의 Compound와 똑같은 기능을 한다.

While

_nodetype이 While일 경우 For와같이 stmt라는 객체가 존재하고 해당 객체안에 _nodetype이 Compound일 경우 If에서의 Compound와 똑같은 기능을 한다.

Switch

_nodetype이 Swtich일 경우 객체 안에 stmt라는 객체가 존재하고 해당 객체안에는 무조건 Case문때문에 _nodetype은 Compund가 오게된다. Case부분의 object를 보면 stmts가 존재하는데 해당 객체안에 도 if문같은 구문들이 올 수 있다.

Programming

이번 과제의 대한 목표는 해당 ast json파일에서 선언한 함수들의 리턴타입, 함수이름, 파라미터 이름과 타입, if를 사용한 개수들을 출력하는 것이다. 나는 해당 문제들을 풀기위해 파라미터 이름 타입 출력, 리턴타입 출력, if 사용 개수 출력을 함수화했다.

```
int if_count(json_value json)   // block_items
{
    int block_size = json_len(json);
    int count = 0;
    for(int i=0; i<block_size; i++)
    {
        json_value block = json_get(json, i);
        char *_nodetype = json_get_string(block, "_nodetype");
        if(strcmp(_nodetype, "If") == 0)
        {
            int ifcount = 0;
            count++;

            count += if_count_true_flase(block);
        }

        // if in for
        if(strcmp(_nodetype, "For") == 0)
        {
            json_value For = json_get(block, "stmt");
            char * _for_nodetype = json_get_string(For, "_nodetype");

            if(strcmp(_for_nodetype, "Compound") == 0)
            {
                json_value For_block_items = json_get(For, "block_items");
                count += if_count(For_block_items);
```

if개수를 알아내기 위해서 _nodetype이 If일 경우에 count를 증가시켜줬다. 현재 사진을 보면 root경로에 대해서만 분석하고 있는데 if를 사용하는 경우는 단일로 사용하거나 if문안에서 if를 사용하거나, for문, while문, switch문 안에서 사용될 수 있다. 따라서

```
int if_count_true_flase(json_value json)    // root if block
{
    int count = 0;
    json_value iftrue = json_get(json, "iftrue");
    json_value iffalse = json_get(json, "iffalse");
    if (!json_is_null(iftrue) && (strcmp(json_get_string(iftrue, "_nodetype"), "Compound") == 0) )
    {
        json_value iftrue_block_items = json_get(iftrue, "block_items");
        count += if_count(iftrue_block_items);
    }

    if(!json_is_null(iffalse))
    {
        if(strcmp(json_get_string(iffalse, "_nodetype"), "Compound") == 0)
        {
            json_value iffalse_block_items = json_get(iffalse, "block_items");
            count += if_count(iffalse_block_items);
        }

    }

    return count;
}
```

```
// if in for
if(strcmp(_nodetype, "For") == 0)
{
    json_value For = json_get(block, "stmt");
    char * _for_nodetype = json_get_string(For, "_nodetype");

    if(strcmp(_for_nodetype, "Compound") == 0)
    {
        json_value For_block_items = json_get(For, "block_items");
        count += if_count(For_block_items);

        int For_block_items_size = json_len(For_block_items);
        for(int l=0; l<For_block_items_size; l++)
        {
            json_value for_block_items_obj = json_get(For_block_items, l);
            char *for_obj_nodetype = json_get_string(for_block_items_obj, "_nodetype");
            if(strcmp(for_obj_nodetype, "Switch")==0)
            {
                count += if_count_switch(for_block_items_obj);
            }
        }
    }
}
```

```
int if_count_switch(json_value json)    // switch block
{
    int count = 0;
    json_value switch_stmt = json_get(json, "stmt");
    json_value switch_block_items = json_get(switch_stmt, "block_items");
    int switch_block_items_size = json_len(switch_block_items);
    for(int i=0; i<switch_block_items_size; i++)
    {
        json_value block_items_obj = json_get(switch_block_items, i);
        json_value obj_stmts = json_get(block_items_obj, "stmts");
        if(!json_is_null(obj_stmts))
        {
            int stmts_size = json_len(obj_stmts);
            for(int j=0; j<stmts_size; j++)
            {
                json_value stmts_block = json_get(obj_stmts, j);
                char *obj_stmts_nodetype = json_get_string(stmts_block, "_nodetype");
                if(strcmp(obj_stmts_nodetype, "If") == 0)
                {
                    count++;
                    count += if_count_true_flase(json_get(stmts_block));
                }
            }
        }
    }
    return count;
}
```

해당 사진들처럼 이에 대한 문제를 해결해줬다.

파라미터와 리턴과 관련해서는 이전에 설명했던 구조를 토대로 함수화해서 만들었다.