Compiler Project | 03 AST

By: Mónica AYALA, Davide AVESANI, Théodore PRÉVOT

AST INTRODUCTION

An Abstract Syntax Tree (AST) is a tree-like data structure used to represent a source code written with a given grammar. The purpose of an AST is to serve as an intermediate representation (IR) to apply transformations on it or to generate an output code. For instance the LLVM backend is a compiler backend for assembly and it can be used to compile multiple languages like c (clang), c++ (clang), rust, haskell, swift, ada to different target architectures (x86, arm, risc-v, …). In our case we will build a simple AST to represent our markdown document. Once the AST is built we will be able to produce an HTML document.

C IMPLEMENTATION

As reported in the previous section, we aim to implement the AST compiler in C based on a three data structure.
In particular, the HTML dom tags represent the nodes of the three data structure, whether the content of the tags, such as the text or the URL for the links, represent the leaves.
To define the DOM components in C we define a struct as reported in the following figure.

```c
typedef struct dom {
    DomElement dom_el; // The type of dom element (Paragraph, Header1, Quote, etc...)
    struct dom_list* children; // A reference to the first element of a list of dom elements. NULL if no children.
    SvgList* svg_children; // A reference to the first element of a list of svg instructions. NULL if no children.
    char* text; // A text parameter that can be used for TextElement, Header, Link and Image elements. NULL if not needed.
    char* url; // A url parameter that can be used for Link and Image elements. NULL if not needed.
} DOM;
```

We proceed therefore to create another data structure to link together the various dom elements, as reported in the following figure.

```c
typedef struct dom_list {
    DOM* dom; // An object containing a dom element.
    struct dom_list* next; // A reference to the next element of the list or NULL if it is the last element.
} DomList;
```

We proceed in a similar way to define the struct for the SVG elements.
The final step to complete the implementation of the code is to initialize the struct data types previously defined. Following a picture of the code modified for this scope.

```c
SvgCoord* new_svg_coord(int x, int y) {
    SvgCoord* svg_coord = malloc(sizeof(SvgCoord));
    svg_coord -> x = x;
    svg_coord -> y = y;
    return svg_coord;
}

SvgCoordList* new_svg_coord_list(SvgCoord* svg_coord) {
    SvgCoordList* svg_coord_list = malloc(sizeof(SvgCoordList));
    svg_coord_list -> coord = svg_coord;
    return svg_coord_list;
}

SvgInst* new_svg_inst(SvgInstKind kind, SvgCoordList* coords) {
    SvgInst* svg_inst = malloc(sizeof(SvgInst));
    svg_inst -> kind = kind;
    svg_inst -> coords = coords;
    svg_inst -> width = -1;
    svg_inst -> height = -1;
    return svg_inst;
}

SvgList* new_svg_list(SvgInst* svg) {
    SvgList* svg_list = malloc(sizeof(SvgList));
    svg_list -> svg = svg;

    return svg_list;
}

DOM* new_dom(DomElement dom_el, DomList* children) {
    DOM* dom = malloc(sizeof(DOM));
    dom->dom_el = dom_el;
    dom->children = children;

    return dom;
}

DomList* new_dom_list(DOM* dom) {
    DomList* dom_list = malloc(sizeof(DomList));
    dom_list -> dom = dom;
    dom_list -> next = dom->children;

    return dom_list;
}
```

TESTING

Inside our project there is a ast_tests.c file that defines the test cases for the AST implementation. In the main function of this file we can find the following code:

```
int main(void) {
    SvgCoord *coords = new_svg_coord(50, 15);

    if (coords == NULL) {
        print_error("new_svg_coord", "The coordinates object is
NULL!");
    }
    if (coords != NULL && coords->x != 50) {
        print_error("new_svg_coord", "The coordinate X should be 50,
get: %d", coords->x);
    }
    if (coords != NULL && coords->y != 15) {
        print_error("new_svg_coord", "The coordinate Y should be 15,
get: %d", coords->y);
    }

    SvgCoordList *coords_list = new_svg_coord_list(coords);

    if (coords_list == NULL) {
        print_error("new_svg_coord_list", "The coordinates list is
NULL!");
    }
    if (coords_list != NULL && (coords_list->coord == NULL ||
coords_list->coord != coords)) {
        print_error("new_svg_coord_list", "The coordinates list should
have the previous coordinates as element");
    }
    if (coords_list != NULL && coords_list->next != NULL) {
        print_error("new_svg_coord_list", "The coordinates list next
element should be NULL");
    }

    SvgInst *svg_instruction = new_svg_inst(Circle, coords_list);

    if (svg_instruction == NULL) {
```

```
        print_error("new_svg_inst", "The svg instruction object is
NULL!");
    }
    if (svg_instruction != NULL && svg_instruction->kind != Circle) {
        print_error("new_svg_inst", "The svg instruction kind should be
Circle, get: %s", svg_tokens[svg_instruction->kind]);
    }
    if (svg_instruction != NULL && svg_instruction->coords !=
coords_list) {
        print_error("new_svg_inst", "The svg instruction coords should
be the previous coordinates list");
    }
    if (svg_instruction != NULL && svg_instruction->width != -1) {
        print_error("new_svg_inst", "The svg instruction width should
be -1 by default, get %d", svg_instruction->width);
    }
    if (svg_instruction != NULL && svg_instruction->height != -1) {
        print_error("new_svg_inst", "The svg instruction height should
be -1 by default, get %d", svg_instruction->height);
    }
    if (svg_instruction != NULL && svg_instruction->text != NULL) {
        print_error("new_svg_inst", "The svg instruction text should be
NULL by default, get %s", svg_instruction->text);
    }
    if (svg_instruction != NULL && svg_instruction->anchor != NULL) {
        print_error("new_svg_inst", "The svg instruction anchor should
be NULL by default, get %s", svg_instruction->anchor);
    }
    if (svg_instruction != NULL && svg_instruction->color_stroke !=
NULL) {
        print_error("new_svg_inst", "The svg instruction color_stroke
should be NULL by default, get %s", svg_instruction->color_stroke);
    }
    if (svg_instruction != NULL && svg_instruction->color_fill != NULL)
{
        print_error("new_svg_inst", "The svg instruction color_fill
should be NULL by default, get %s", svg_instruction->color_fill);
    }

    SvgList *svg_list = new_svg_list(svg_instruction);

    if (svg_list == NULL) {
```

```
        print_error("new_svg_list", "The svg instructions list object
is NULL!");
    }
    if (svg_list != NULL && svg_list->svg != svg_instruction) {
        print_error("new_svg_list", "The svg instructions list first
element should be the previous svg instruction");
    }
    if (svg_list != NULL && svg_list->next != NULL) {
        print_error("new_svg_list", "The svg instructions list next
element should be NULL");
    }

    DOM *dom = new_dom(Paragraph, NULL);

    if (dom == NULL) {
        print_error("new_dom", "The dom object is NULL!");
    }
    if (dom != NULL && dom->dom_el != Paragraph) {
        print_error("new_dom", "The dom element should be Paragraph,
get: %s", dom_tokens[dom->dom_el]);
    }
    if (dom != NULL && dom->children != NULL) {
        print_error("new_dom", "The dom children should be NULL!");
    }
    if (dom != NULL && dom->svg_children != NULL) {
        print_error("new_dom", "The dom svg children should be NULL!");
    }
    if (dom != NULL && dom->text != NULL) {
        print_error("new_dom", "The dom text should be NULL by default,
get %s", dom->text);
    }
    if (dom != NULL && dom->url != NULL) {
        print_error("new_dom", "The dom url should be NULL by default,
get %s", dom->url);
    }

    DomList *dom_list = new_dom_list(dom);

    if (dom_list == NULL) {
        print_error("new_dom_list", "The dom list object is NULL!");
    }
    if (dom_list != NULL && dom_list->dom != dom) {
```

```
        print_error("new_dom_list", "The dom list first element should
be the previous dom");
    }
    if (dom_list != NULL && dom_list->next != NULL) {
        print_error("new_dom_list", "The dom list next element should
be NULL");
    }
}
```

These are all of the tests with their corresponding error should the tests fail, making use of the print_error function.The tests basically test 6 sections of the AST implementation: SVG coordinates, SVG coordinates list, SVG instance, SVG list, DOM and the DOM list.

Initially, these were the tests that failed and passed, together with the output from using the commands *$> make ast_test* and *$> ./out/ast_tests*.

SVG Coordinates

- The coordinates object shouldn't be null **pass**
- The coordinate X should be 50 **fail**
- The coordinate Y should be 15 **fail**

SVG Coordinates List

- Coordinates list object shouldn't be null **pass**
- Coordinates list should have previous coordinate as element **fail**
- The coordinates list next element should be null **pass**

SVG Instruction

- SVG Instruction object is null **pass**
- SVG Instruction kind should be circle **fail**
- SVG Instruction coords should be the previous coordinates list **fail**
- SVG Instruction width should be -1 by default **fail**
- SVG Instruction height should be -1 by default **fail**
- SVG Instruction text should be null by default **pass**
- SVG Instruction anchor should be null by default **pass**
- SVG Instruction color_stroke should be null by default **pass**

- SVG Instruction color_fill should be null by default **pass**

SVG List

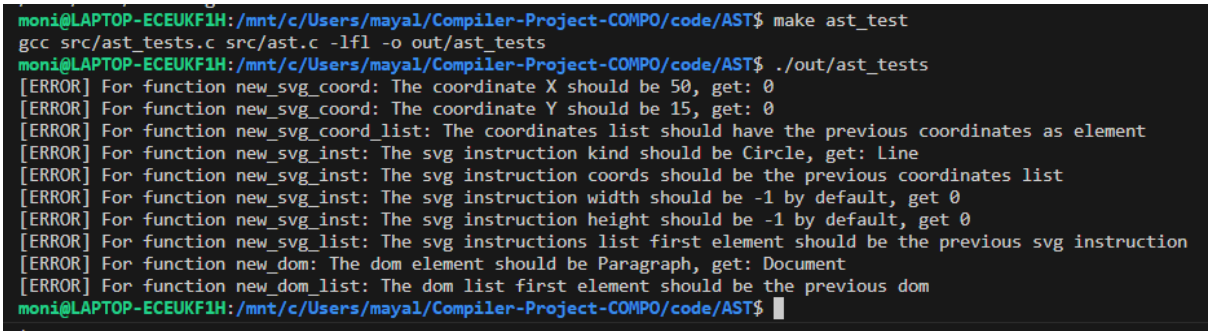
- SVG Instruction list shouldn't be null **pass**
- SVG Instruction list's first element should be last svg instruction **fail**
- SVG Instruction list's next element should be null **pass**

DOM

- Creating a non null DOM object **pass**
- DOM object correctly adds paragraph **fail**
- DOM children should be null **pass**
- DOM SVG children should be null **pass**
- DOM text should be null by default **pass**
- DOM URL should be null by default **pass**

DOM List

- Creating a non null DOM List object **pass**
- First element of the DOM List should be the previous DOM **fail**
- The next element of the DOM List should be null **pass**



After implementing the missing functions in the ast.c file and running again the *$> make ast_test* and *$> ./out/ast_tests* commands in the terminal we get no errors, which means our implementation passed all of the test cases. As we implemented the missing AST functions the test cases helped guide us and debug the errors we faced.

Testing is something necessary whenever we are coding something and automated test cases can be very time efficient if correctly implemented. During all of the previous deliveries we have always used test cases for the software we have been developing.