Exercises — Basic AST

version **#7be580532266ed398481e31366afcc24b1950c2a**

Assistants C/Unix 2023 <assistants@tickets.assistants.epita.fr>

Copyright

This document is for internal use at EPITA (website) only.

Contents

*https://intra.assistants.epita.fr

File Tree

```
basic_ast/
├── ast.h
├── evaluate.c  (to submit)
├── parser.c  (to submit)
├── simple_ast_evaluate_example.c
└── simple_ast_print_example.c
```

Authorized functions : You are only allowed to use the following functions

- atoi(3)
- fclose(3)
- fopen(3)
- free(3)
- getline(3)
- malloc(3)
- calloc(3)
- realloc(3)
- printf(3)
- putchar(3)
- strtok_r(3)

Authorized headers : You are only allowed to use the functions defined in the following headers

- err.h
- errno.h
- assert.h
- stddef.h

Compilation : Your code must compile with the following flags

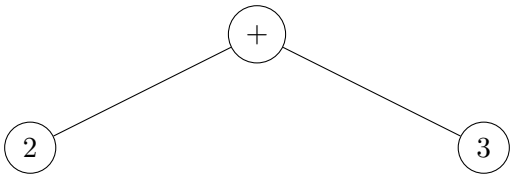
- -std=c99 -pedantic -Werror -Wall -Wextra -Wvla

Main function : None

1 Goal

AST means *abstract syntax tree*. It is a way to describe syntactic constructions as a tree data structure, which can then be used, for instance in order to evaluate expressions in an interpreter or to optimize code through transformation of the tree.

For example:



Let's take a basic arithmetic operation: 2 + 3. The AST which represent this expression will be composed of 3 nodes. The root will be a node which will represent the '+' operator and the left child will be 2 and the right child 3.

To evaluate our tree, we are going to use the first deeps traversal algorithm to visit each nodes. In our tree, it would be: + then 3 and 2.

You will have to implement an AST evaluator for arithmetic expressions. Only the sum and the product have to be handled here. Be careful, even if you do not have to handle subtractions, -2+3 is a valid expression and should return 1.

We provide you a given header: `ast.h`. This header will describe the structure of your AST and the prototype of the functions you have to implement.

Of course, if needed, you can add other functions and modify the header. But be careful, you must not modify the prototype of the functions already presents in the header! If you do not respect this rules, we will not be able to evaluate your work.

Note that we will compile your code with `-D_XOPEN_SOURCE=700`. There should therefore be no errors when you code is compiled with this option.

The following functions must be implemented in `parser.c`:

```
struct node *ast_build(const char *filename);
void ast_delete(struct node *root);
```

The following functions must be implemented in `evaluate.c`:

```
int ast_evaluate(const struct node *root);
void ast_print(const struct node *root);
```

You must not submit other C files.

In this exercise, you can consider that test files will consist of one line terminated by an EOL. You do not have to handle error in parsing and can consider that input files will be valid. The input file will not contain space characters or bad operators.

2 AST Build

The first function you will have to implement builds an AST. This function will take in parameter a filename. Once your file is open, you have to read the first line and parse it to build the AST. You *must* handle operator priority. If the filename provided is not correct (bad path or NULL pointer), this function *MUST* return `NULL`.

Going further...

You should consider checking out the Shunting Yard algorithm to build such an AST.

The prototype of the build function will be:

```
struct node *ast_build(const char *filename);
```

3 AST Print

In order to check if your AST is built correctly, you will have to implement a print function.

The prototype of the print function will be:

```
void ast_print(const struct node *root);
```

3.1 Example

```
42sh$ cat -e test.in
2*2+3$
42sh$ gcc -Wall -Werror -std=c99 -Wextra -pedantic -D_XOPEN_SOURCE=700 evaluate.c parser.c␣
↪simple_ast_print_example.c -o print_example
42sh$ ./print_example test.in | cat -e
2*2+3$
```

Be careful!

Your output must be the same as the expression stored in the input file followed by a new line.

4 AST Evaluation

Once your ast is built, you will have to evaluate it.

The prototype of the evaluate function must be as follows:

```
int ast_evaluate(const struct node *root);
```

4.1 Example

```
42sh$ cat -e test.in
2+3*6$
42sh$ gcc -Wall -Werror -std=c99 -Wextra -pedantic -D_XOPEN_SOURCE=700 evaluate.c parser.c␣
↪simple_ast_evaluate_example.c -o eval_example
42sh$ ./eval_example test.in
42sh$ echo $?
20
```

The behaviour of this function is very simple: it visits each node of your AST and returns the result of the operation.

5 AST Deletion

Now that your AST is evaluated, you do not want to leave leaks at the end of our tests. Be sure to free your tree and avoid memory leaks thanks to the last function of this exercise :

```
void ast_delete(struct node *root);
```

The way is lit. The path is clear. We require only the strength to follow it.