NAME: Jinyi Xia
STUDENT ID: 2021212057
CLASS NUMBER: 2021211802
CONTAINER NUMBER: a4d4f6a5498a4e912a4d1a76a7fa36cf23af64b416688d222a8968a38b29a711

REPORT ON PROJECT 1

1 Overview

In this project, I implemented a parser for BPL (BUPT Programming Language), a subset of the C programming language. It can generate an abstract syntax tree (AST) for a given BPL program, and recover from both lexical and syntax errors. Besides, it can recognize both single-line and multi-line comments.

2 Implementation Details

2.1 AST generation

I used the following data structures to represent the AST.

```c
typedef struct Head {
    char *type;
    size_t line_no;
    char *property;
    struct Head *child, *sibling;
} *Node, *Tree;
```

Meanwhile, I used the following functions to generate the AST.

```c
Node new_node(
    const char *type,
    size_t line_no,
    const char *property,
    Node child,
    Node sibling
);
void print_tree(
    FILE *restrict stream,
    Tree tree,
    size_t indent
);
void empty_tree(Tree tree);
```

Among these functions,

- `new_node` is used to create a new node, where `type` is the type of the node, `line_no` is the line number of the node's location, `property` is the property of the node, `child` is the eldest child of the node, and `sibling` is the next sibling of the node;

- `print_tree` uses depth-first traversal to print the AST to a given stream, where `stream` specifies the output stream, `tree` is the tree to print, and `indent` is the number of spaces to indent;
- `empty_tree` uses a recursive method to free the memory allocated for the AST, where `tree` is the tree to free.

The implementation of these functions can be found in `tree.c`.

2.2 Lexical Analysis

I used flex to generate a lexical analyzer for BPL. It can recognize lexemes and generate tokens for the parser. Tokens will be passed to the parser as nodes. The implementation of the lexical analyzer can be found in `lex.l`.

2.3 Syntax Analysis

I used bison to generate a parser for BPL. It can generate an AST for a given BPL program by the given rules, and recover from syntax errors. The implementation of the parser can be found in `syntax.y`.

3 Optional Feature

Except for the required features, I also implemented the following optional feature.

3.1 Comments

My parser can recognize both single-line and multi-line comments. Relevent regular expressions are defined in `lex.l`.

```
"//".*$        { /* ignore comments */ }
"/*"([^*]|[*][^/]|[*][*]+[^*/])*"*/" { for (size_t i = 0; i < strlen(yytext); i++)
    if (yytext[i] == '\n') line++; }
```

For the single-line comments, I simply ignored them. For the multi-line comments, I counted the number of newlines in the comment and added it to the line number.

e.g. The following code can be parsed correctly.

```
/*
    This is a multi-line comment.
    It can span multiple lines.
*/
int square(int num) {
    return num * num; // This is a single-line comment.
}
```

My parser will generate the following AST for the above code.

```
Program (5)
  ExtDefList (5)
    ExtDef (5)
      Specifier (5)
        TYPE: int
      FunDec (5)
        ID: square
        LP
        VarList (5)
          ParamDec (5)
            Specifier (5)
              TYPE: int
            VarDec (5)
              ID: num
        RP
      CompSt (5)
        LC
        StmtList (6)
          Stmt (6)
            RETURN
            Exp (6)
              Exp (6)
                ID: mum
              MUL
              Exp (6)
                ID: mum
            SEMI
        RC
```

4 Usage

Detailed usage can be found in `README.md`.

4.1 Build

To build the project, simply run `make` or `make bplc` in the root directory of the project. The executable file `bplc` will be generated in the `bin` directory.

4.2 Run

To run the parser, run `./bin/bplc <input_file>` in the root directory of the project, where `<input_file>` is the BPL program to parse.

4.3 Test

To easily go through all the test cases, I wrote a shell script `test.sh` in the root directory of the project. Simply run `bash ./test.sh` or `make test` in the root directory of the project, and all the cases will be tested.