Deliverable III: CGF with Lex and Yacc

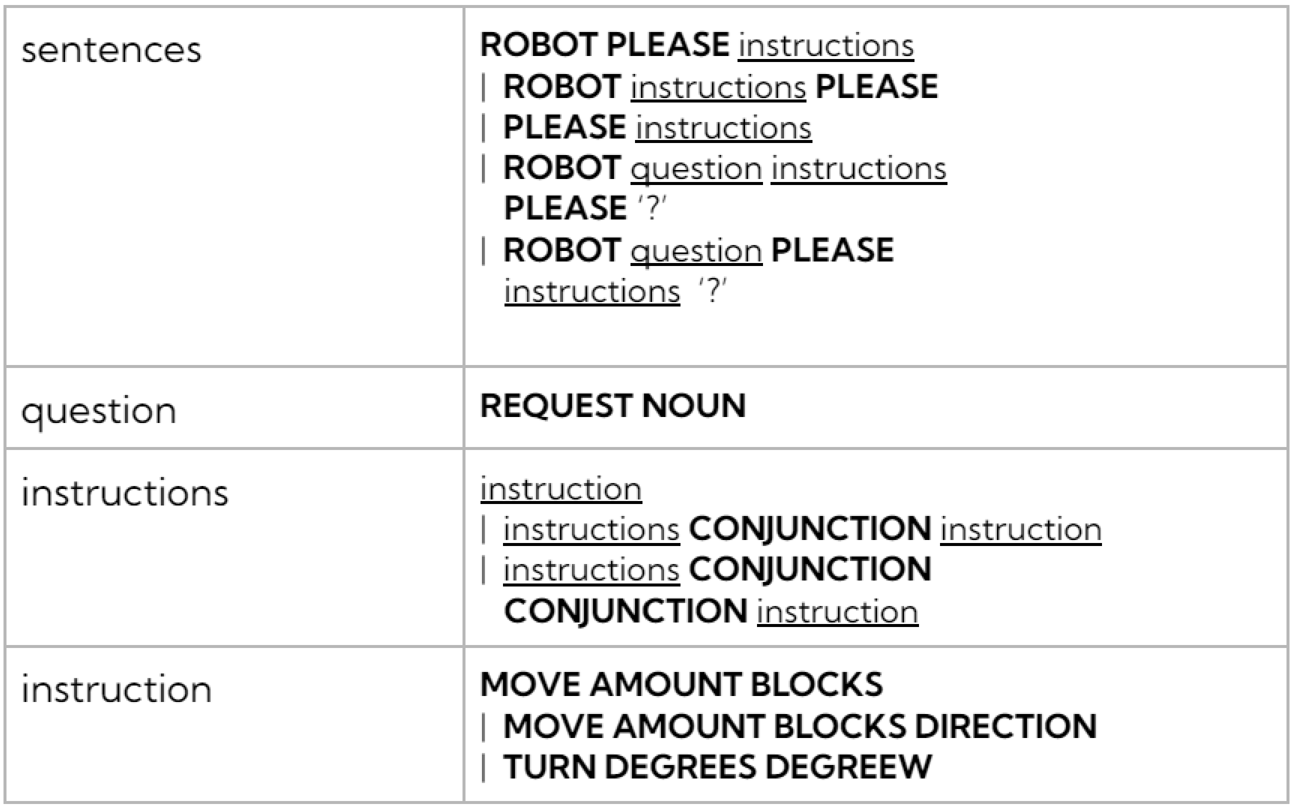
After generating the tokens that simplify human language, a Context Free Grammar is needed to finish processing the instruction. The CFG will combine different tokens to form new variables that can compose a final variable that is accepted by the compiler and CPU. This way, different phrasing can lead to the same result.

Taking under consideration that the robot can only accomplish the instructions:

1. MOV X (X being an integer)
2. TURN Y (Y being 90,180,270,360)

The following CFG was generated.

Context Free Grammar



sentences	**ROBOT PLEASE** instructions | **ROBOT** instructions **PLEASE** | **PLEASE** instructions | **ROBOT** question instructions **PLEASE** '?' | **ROBOT** question **PLEASE** instructions '?'
question	**REQUEST NOUN**
instructions	instruction | instructions **CONJUNCTION** instruction | instructions **CONJUNCTION CONJUNCTION** instruction
instruction	**MOVE AMOUNT BLOCKS** | **MOVE AMOUNT BLOCKS DIRECTION** | **TURN DEGREES DEGREEW**

The Yacc is composed by the different variables available, and then it tries to form a sentence (final variable). If the program suceeds, it can then translate the sentence into the Robots language, if not, it will show an error as the input was not valid.

Inputs

The following lists show both VALID and INVALID inputs:

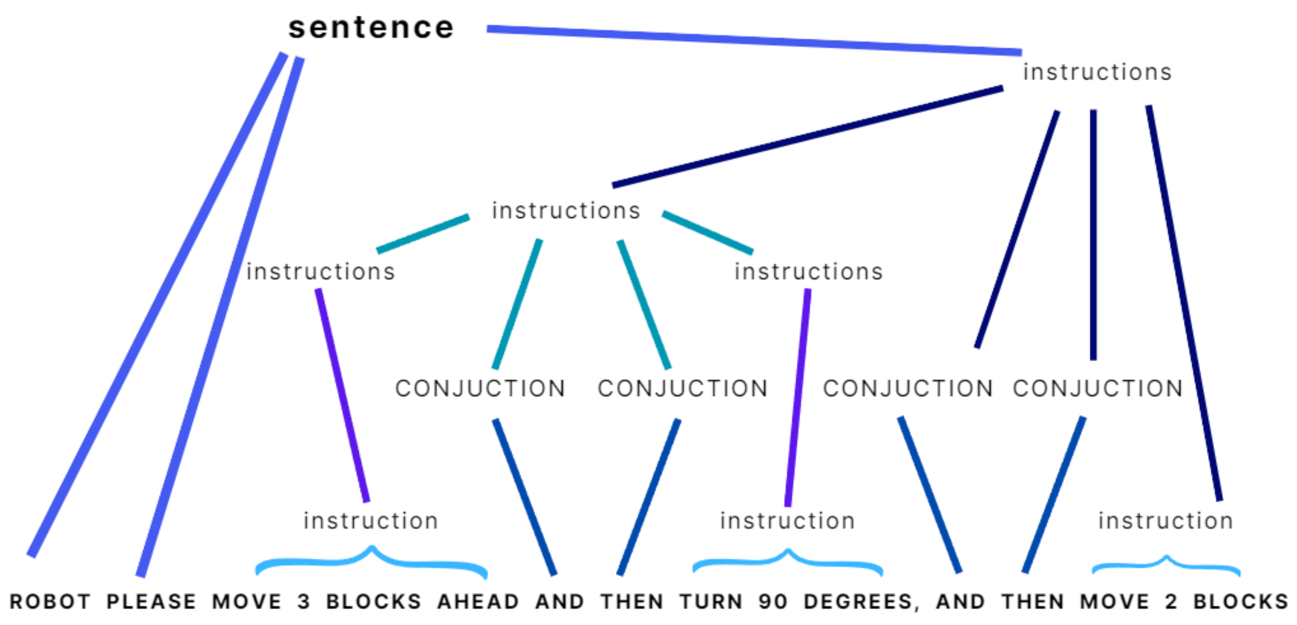
Valid Sentences

1. Robot please move 2 blocks ahead
2. Robot please move 2 blocks
3. Robot please move 3 blocks ahead and the turn 90 degrees, and then move 2 blocks
4. Robot please turn 270 degrees and then move 2 blocks
5. Robot could you please move 3 blocks?
6. Robot could you move 3 blocks please?
7. Robot move 3 blocks please
8. Please move 3 blocks

Invalid Sentences

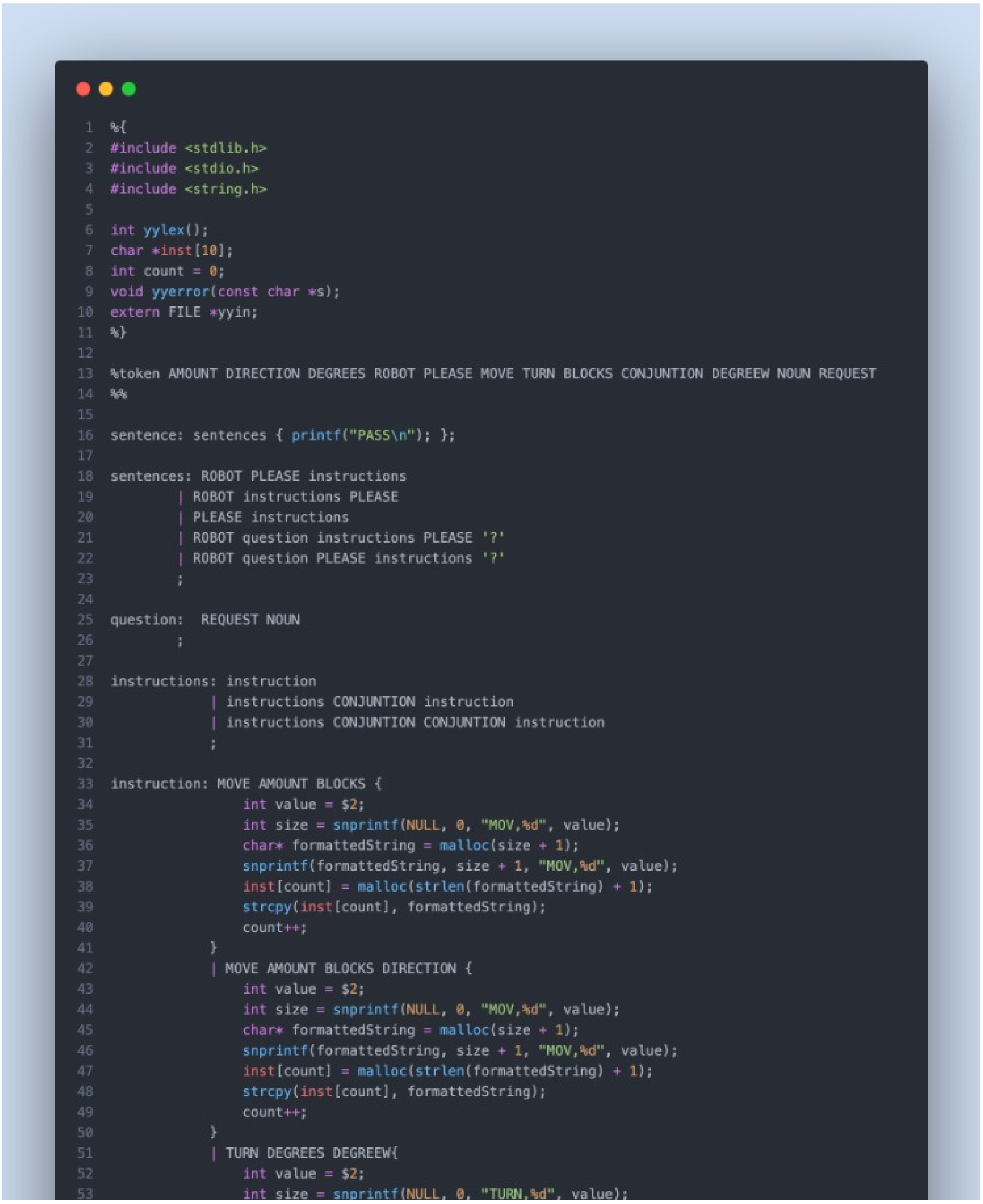
1. Robot please move 2
2. Robot please turn 90
3. Robot move 3 blocks
4. Move 2 blocks and then turn 90 degrees
5. Robot move 5 ahead and turn 360 degrees
6. Robot please turn 20 degrees
7. Turn 90 degrees and then move 3 blocks
8. move 4 blocks

Example

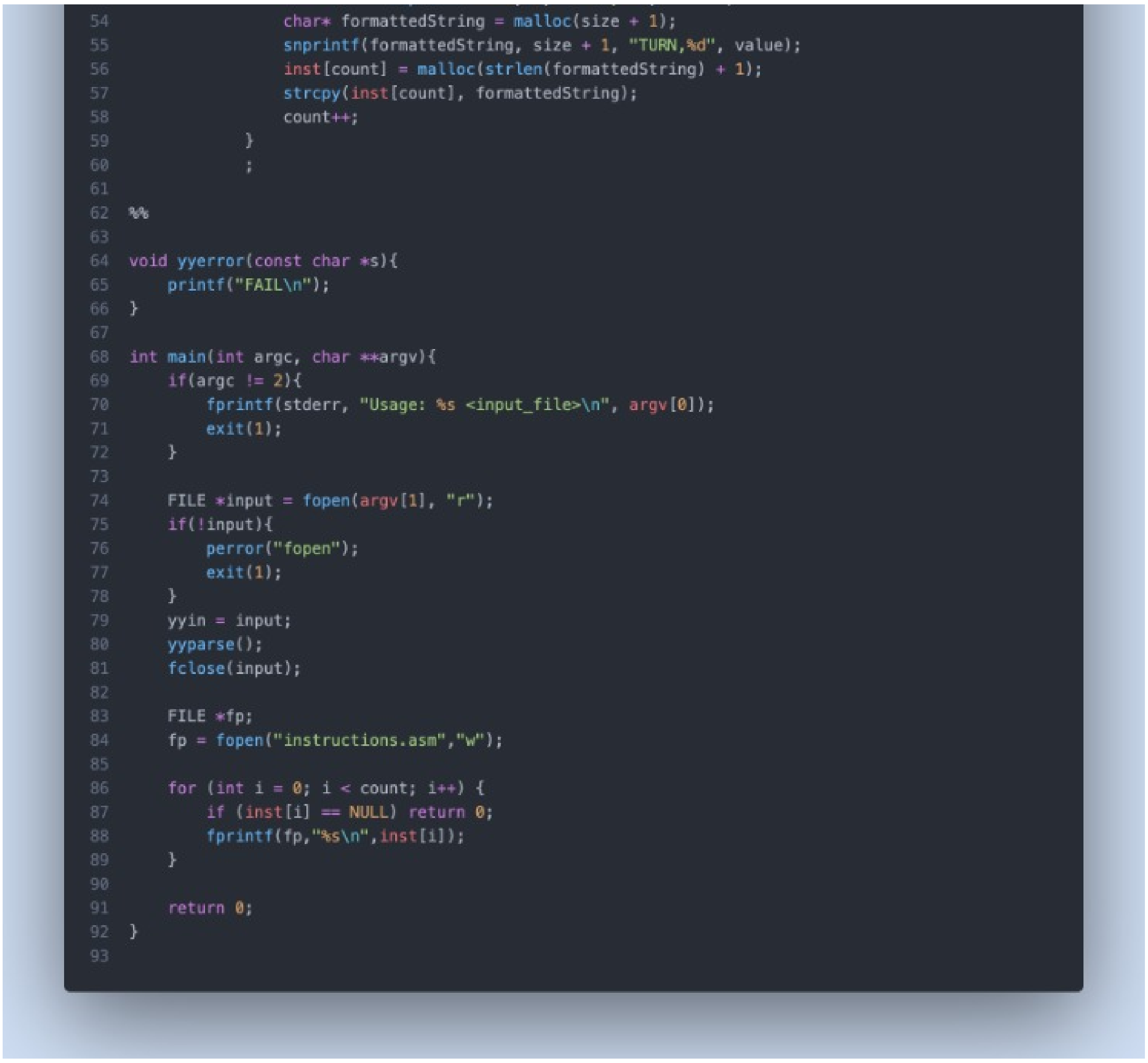


PROGRESS	INSTRUCTION
Human language	Robot please move 3 blocks ahead and then turn 90 degrees, and then move 2 blocks
Lex	ROBOT PLEASE MOVE AMOUNT BLOCKS DIRECTION CONJUNCTION CONJUNCTION TURN DEGREES DEGREEW CONJUNCTION CONJUNCTION MOVE AMOUNT BLOCKS
Yacc	MOV 3, TURN 90, MOV 2

Code



```
%{
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

int yylex();
char *inst[10];
int count = 0;
void yyerror(const char *s);
extern FILE *yyin;
%}

%token AMOUNT DIRECTION DEGREES ROBOT PLEASE MOVE TURN BLOCKS CONJUNTION DEGREEW NOUN REQUEST
%%

sentence: sentences { printf("PASS\n"); };

sentences: ROBOT PLEASE instructions
        | ROBOT instructions PLEASE
        | PLEASE instructions
        | ROBOT question instructions PLEASE '?'
        | ROBOT question PLEASE instructions '?'
        ;

question:  REQUEST NOUN
        ;

instructions: instruction
            | instructions CONJUNTION instruction
            | instructions CONJUNTION CONJUNTION instruction
            ;

instruction: MOVE AMOUNT BLOCKS {
                int value = $2;
                int size = snprintf(NULL, 0, "MOV,%d", value);
                char* formattedString = malloc(size + 1);
                snprintf(formattedString, size + 1, "MOV,%d", value);
                inst[count] = malloc(strlen(formattedString) + 1);
                strcpy(inst[count], formattedString);
                count++;
            }
            | MOVE AMOUNT BLOCKS DIRECTION {
                int value = $2;
                int size = snprintf(NULL, 0, "MOV,%d", value);
                char* formattedString = malloc(size + 1);
                snprintf(formattedString, size + 1, "MOV,%d", value);
                inst[count] = malloc(strlen(formattedString) + 1);
                strcpy(inst[count], formattedString);
                count++;
            }
            | TURN DEGREES DEGREEW{
                int value = $2;
                int size = snprintf(NULL, 0, "TURN,%d", value);
```



```
                char* formattedString = malloc(size + 1);
                snprintf(formattedString, size + 1, "TURN,%d", value);
                inst[count] = malloc(strlen(formattedString) + 1);
                strcpy(inst[count], formattedString);
                count++;
            }
            ;

%%

void yyerror(const char *s){
    printf("FAIL\n");
}

int main(int argc, char **argv){
    if(argc != 2){
        fprintf(stderr, "Usage: %s <input_file>\n", argv[0]);
        exit(1);
    }

    FILE *input = fopen(argv[1], "r");
    if(!input){
        perror("fopen");
        exit(1);
    }
    yyin = input;
    yyparse();
    fclose(input);

    FILE *fp;
    fp = fopen("instructions.asm","w");

    for (int i = 0; i < count; i++) {
        if (inst[i] == NULL) return 0;
        fprintf(fp,"%s\n",inst[i]);
    }

    return 0;
}

```

Links

Github Main Project link

Github Deliverable III link