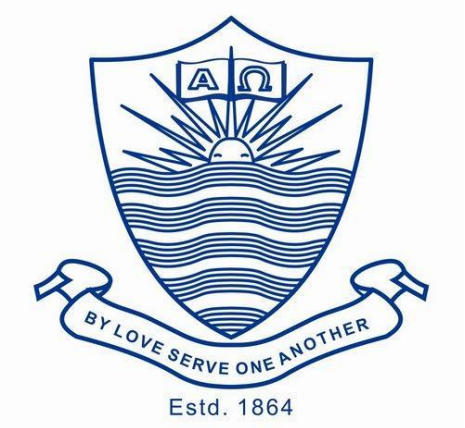
FORMAN CHRISTIAN COLLEGE

(A CHARTERED UNIVERSITY)



COMP451A

Fall 2022

Course Project

Abdullah Baig

231485698

INTRODUCTION:

The code provided is an implementation of LR(0) and LL(1) parsing algorithms. The code uses a stack-based approach to parse a given input string against a predefined grammar. The code has been implemented in C and uses several helper functions and data structures to implement the parsing algorithm.

The parse_table variable is a 2D array of strings that represents the parse table for the given grammar. The rows represent the states and the columns represent the symbols. Each entry in the table represents the next action to be taken based on the current state and symbol.

The ruleset variable is a 2D array of strings that represents the production rules for the given grammar. Each row represents a single production rule, with the left-hand side of the rule in the first column and the right-hand side in the second column.

The stack1 and stack2 variables are character arrays that represent the two stacks used in the parsing algorithm. The stack1 array is used to store the production rules and the stack2 array is used to store the input string.

The stack1_pointer and stack2_pointer variables are integers that represent the current position of the top of the stacks.

The main function takes in the input string as a command-line argument and calls the parse() function to start the parsing process. If the input string is accepted by the grammar, the program prints "String is accepted by the grammar", otherwise, it prints "String is not accepted by the grammar".

FUNCTION DECLARATIONS:

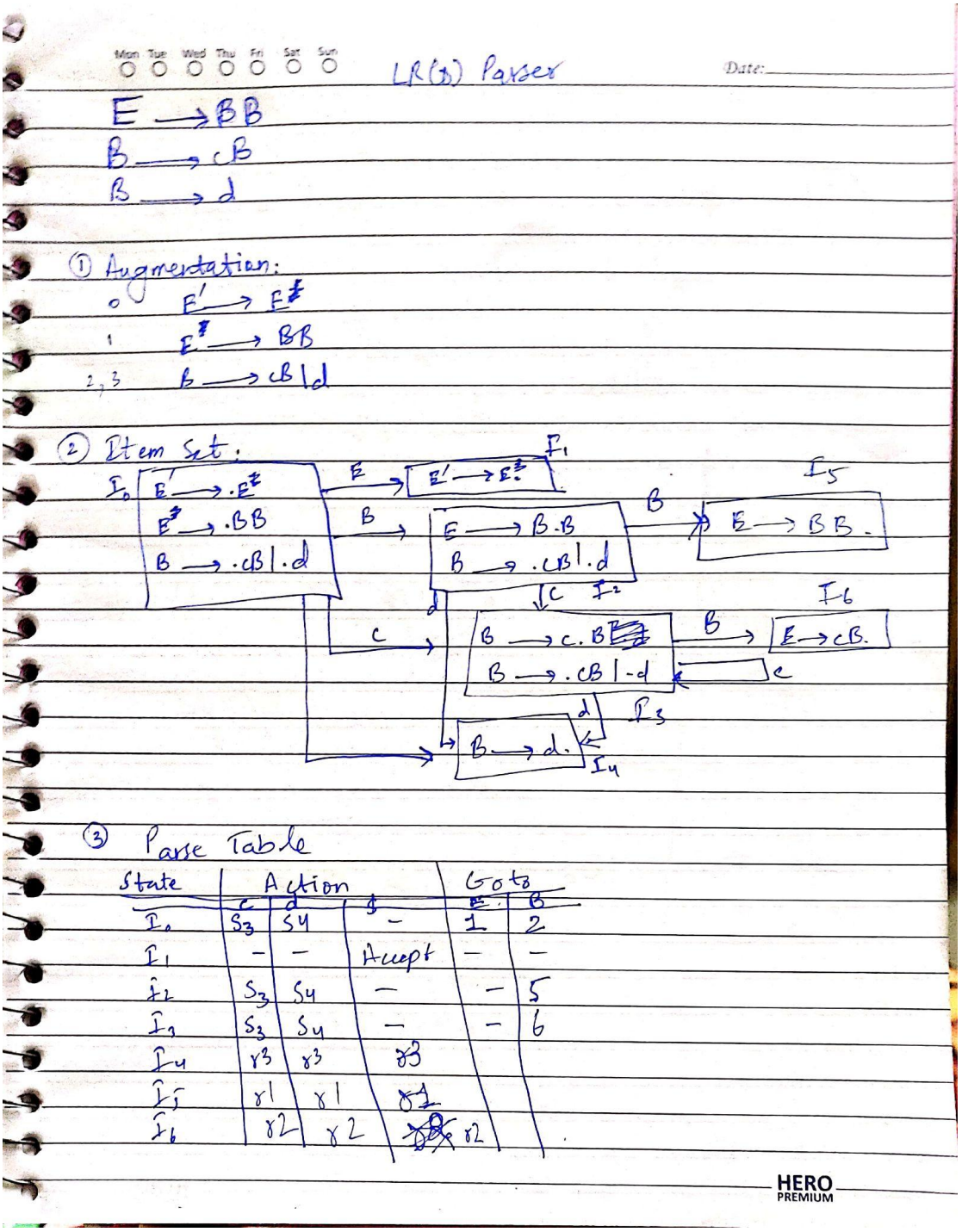
- **int parse():** This function is the main function that implements the LR(0) and LL(1) parsing algorithms. It returns 1 if the input string is accepted by the grammar and 0 otherwise.
- **char pop1():** This function pops the topmost symbol from stack1 and returns it.
- **char pop2():** This function pops the topmost symbol from stack2 and returns it.
- **void get_stack1():** This function prints the contents of stack1.

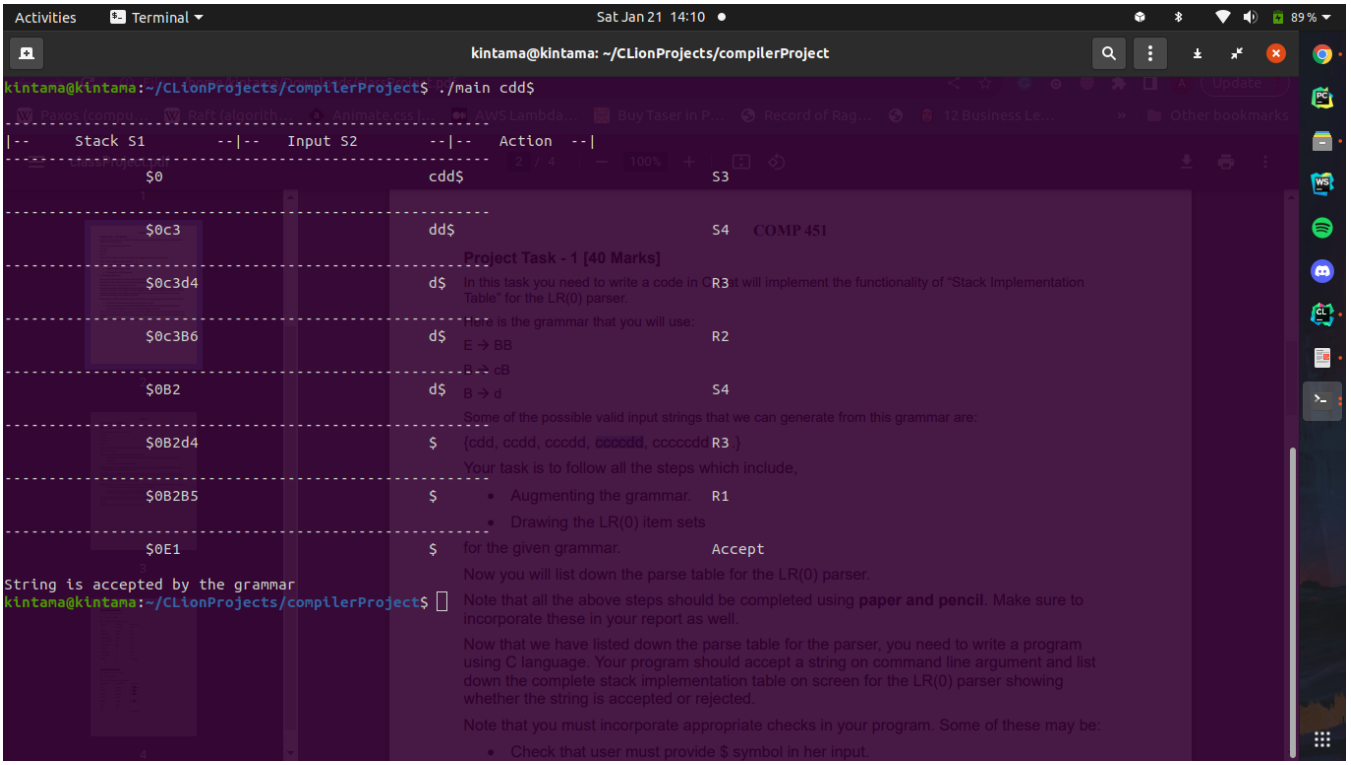
- **void get_stack2():** This function prints the contents of stack2.
- **void push1(char to_push):** This function pushes the given character to stack1.
- **void push2(char to_push):** This function pushes the given character to stack2.
- **int convert_char(char chr):** This function converts the given character to its corresponding integer value, which is used to look up the parse table entry. The function returns -1 if the character is not a valid symbol in the grammar.
- **char get_top1():** This function returns the topmost symbol in stack1.
- **char get_top2():** This function returns the topmost symbol in stack2

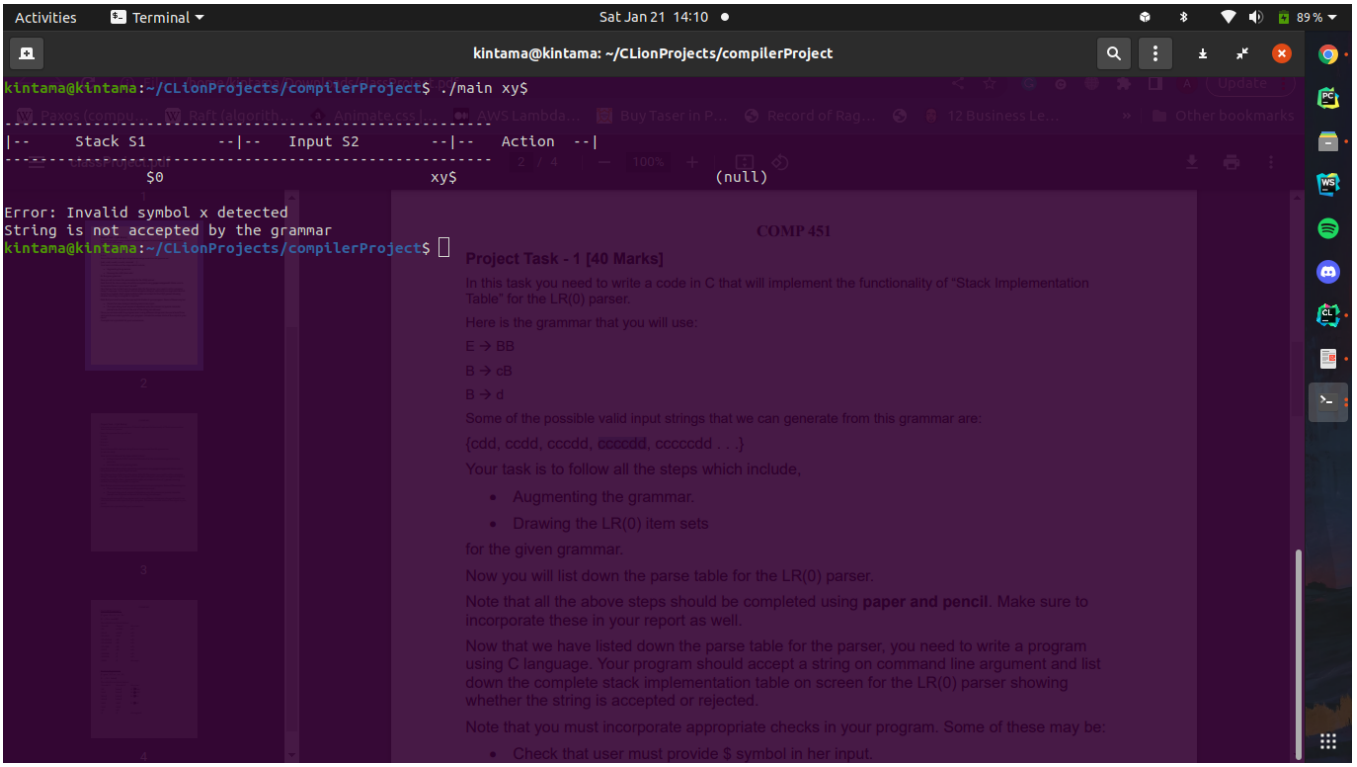
LR(0) LOGIC:

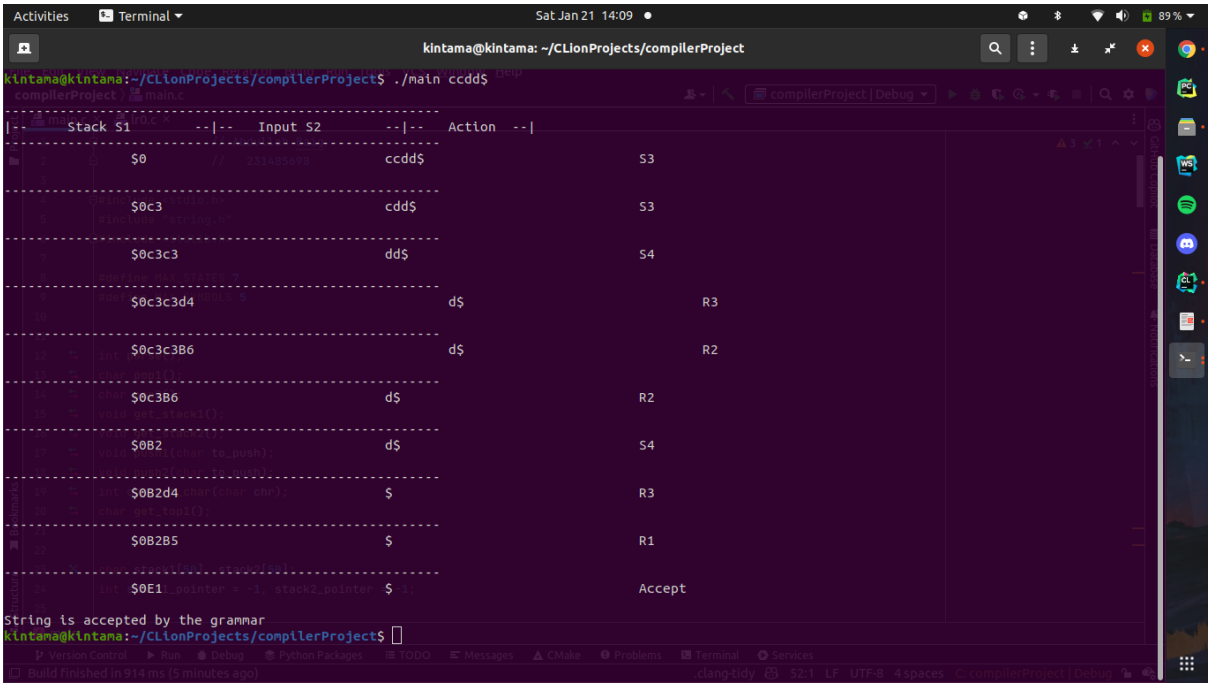
- The LR(0) parsing algorithm is implemented in the parse() function. The function starts by initializing the stack1 and stack2 pointers to -1. It then reads the input string and pushes it to stack2. The function then enters a while loop that continues until the stack1 pointer becomes -1.
- The function starts by reading the topmost symbol from stack1 and the current symbol from stack2. It then looks up the corresponding parse table entry for the state represented by the topmost symbol in stack1 and the current symbol in stack2.
- If the entry is an "Accept" symbol, the function returns 1 indicating that the input string is accepted by the grammar. If the entry is an "_" symbol, the function returns 0 indicating that the input string is not accepted by the grammar.
- If the entry is an "R" symbol, the function performs a reduction operation. The function pops the topmost symbol from stack1 and pushes the corresponding right-hand side of the production rule onto the stack.
- If the entry is an "S" symbol, the function performs a shift operation. The function pushes the current symbol from stack2 onto stack1 and pops it from stack2.

LR(0) PARSE TABLE & OUTPUT:




Activities
Terminal
Sat Jan 21 14:10
89 %
kintama@kintama: ~/CLionProjects/compilerProject
kintama@kintama:~/CLionProjects/compilerProject$ ./main cdd$
|-- Stack S1 --|-- Input S2 --|-- Action --|
$0 cdd$ S3
$0c3 dd$ S4
$0c3d4 d$ R3
$0c3B6 d$ R2
$0B2 d$ S4
$0B2d4 $ R3
$0B2B5 $ R1
$0E1 $ Accept
String is accepted by the grammar
kintama@kintama:~/CLionProjects/compilerProject$
COMP 451
Project Task - 1 [40 Marks]
In this task you need to write a code in C that will implement the functionality of "Stack Implementation Table" for the LR(0) parser.
Here is the grammar that you will use:
E → BB
B → cB
B → d
Some of the possible valid input strings that we can generate from this grammar are:
{cdd, ccdd, cccdd, ccccdd, cccccdd . . .}
Your task is to follow all the steps which include,
Augmenting the grammar.
Drawing the LR(0) item sets
for the given grammar.
Now you will list down the parse table for the LR(0) parser.
Note that all the above steps should be completed using paper and pencil. Make sure to incorporate these in your report as well.
Now that we have listed down the parse table for the parser, you need to write a program using C language. Your program should accept a string on command line argument and list down the complete stack implementation table on screen for the LR(0) parser showing whether the string is accepted or rejected.
Note that you must incorporate appropriate checks in your program. Some of these may be:
Check that user must provide $ symbol in her input.

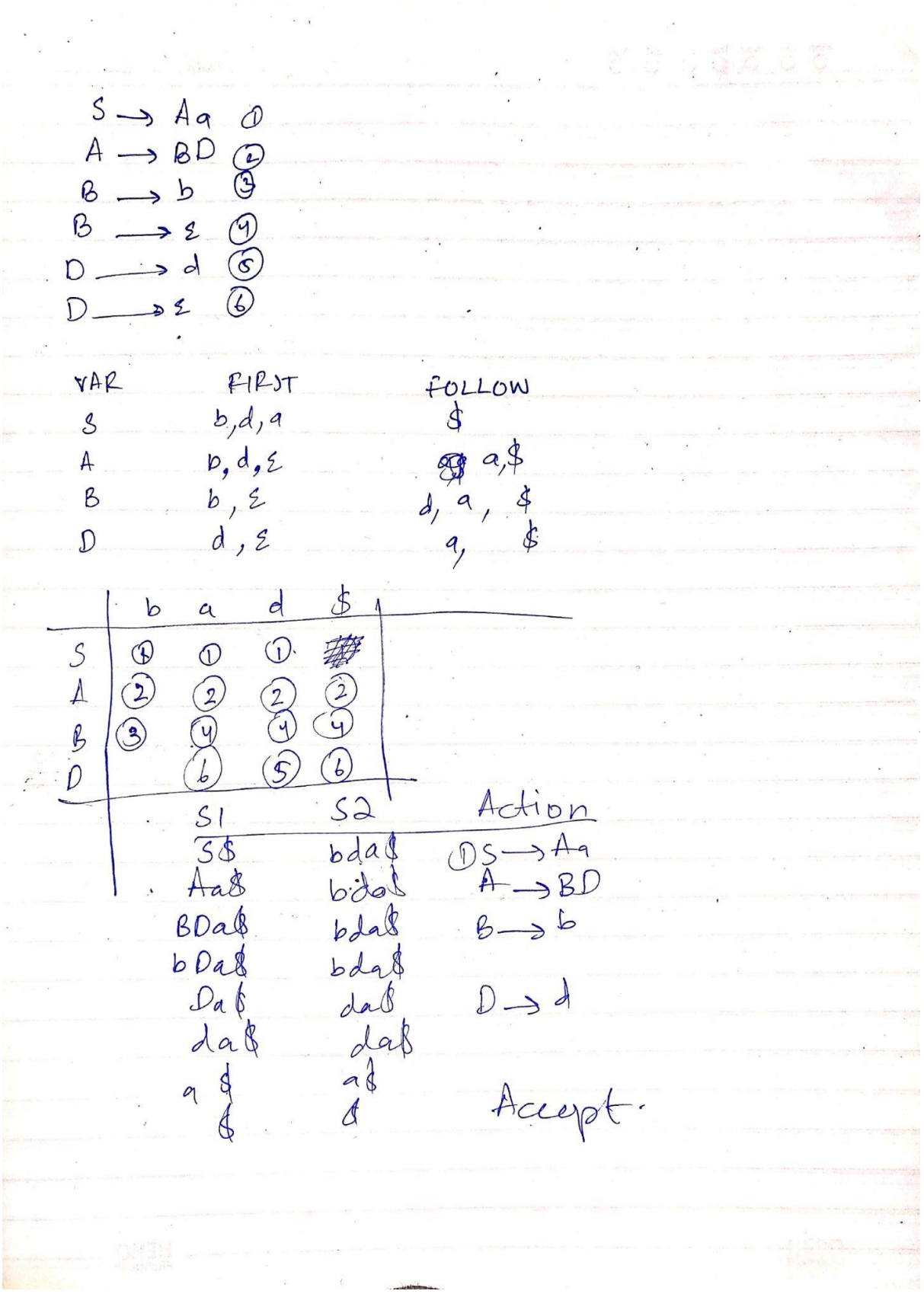

Activities
Terminal
Sat Jan 21 14:10
89 %
kintama@kintama: ~/CLionProjects/compilerProject
kintama@kintama:~/CLionProjects/compilerProject$ ./main xy$
|-- Stack S1 --|-- Input S2 --|-- Action --|
$0 xy$ (null)
Error: Invalid symbol x detected
String is not accepted by the grammar
kintama@kintama:~/CLionProjects/compilerProject$
COMP 451
Project Task - 1 [40 Marks]
In this task you need to write a code in C that will implement the functionality of "Stack Implementation Table" for the LR(0) parser.
Here is the grammar that you will use:
E → BB
B → cB
B → d
Some of the possible valid input strings that we can generate from this grammar are:
{cdd, ccdd, cccdd, ccccdd, cccccdd . . .}
Your task is to follow all the steps which include,
Augmenting the grammar.
Drawing the LR(0) item sets
for the given grammar.
Now you will list down the parse table for the LR(0) parser.
Note that all the above steps should be completed using paper and pencil. Make sure to incorporate these in your report as well.
Now that we have listed down the parse table for the parser, you need to write a program using C language. Your program should accept a string on command line argument and list down the complete stack implementation table on screen for the LR(0) parser showing whether the string is accepted or rejected.
Note that you must incorporate appropriate checks in your program. Some of these may be:
Check that user must provide $ symbol in her input.

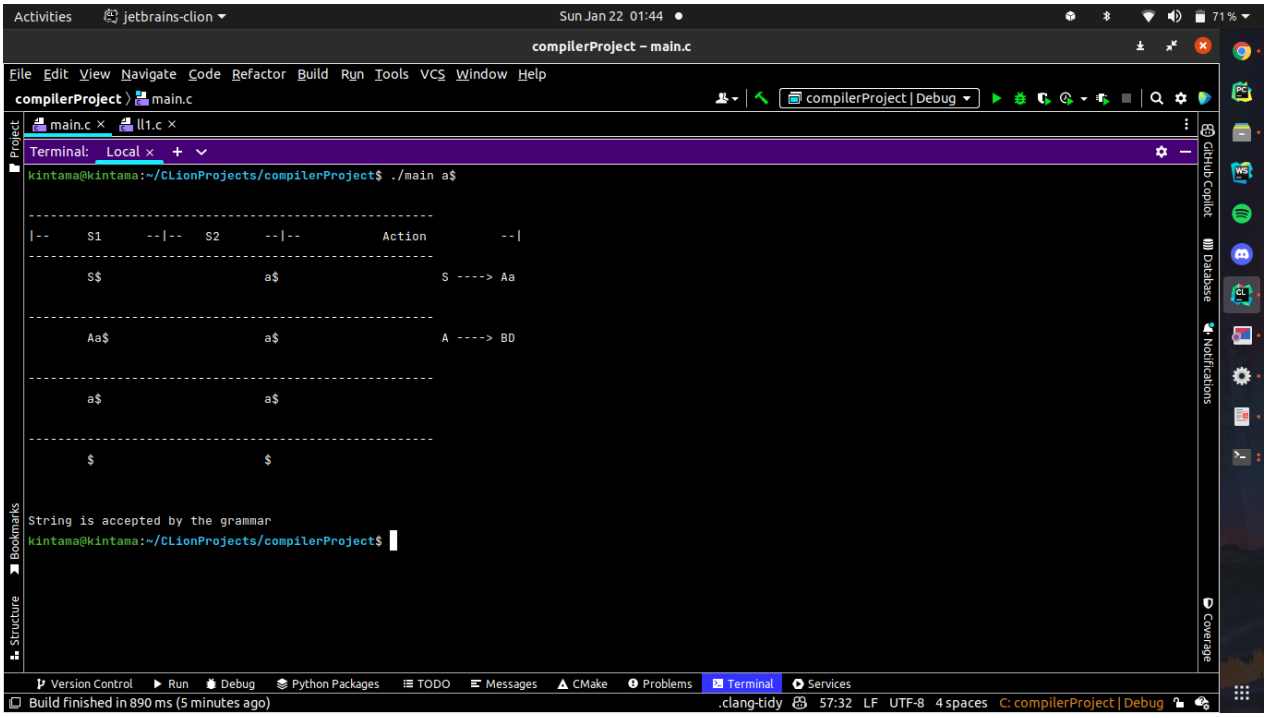


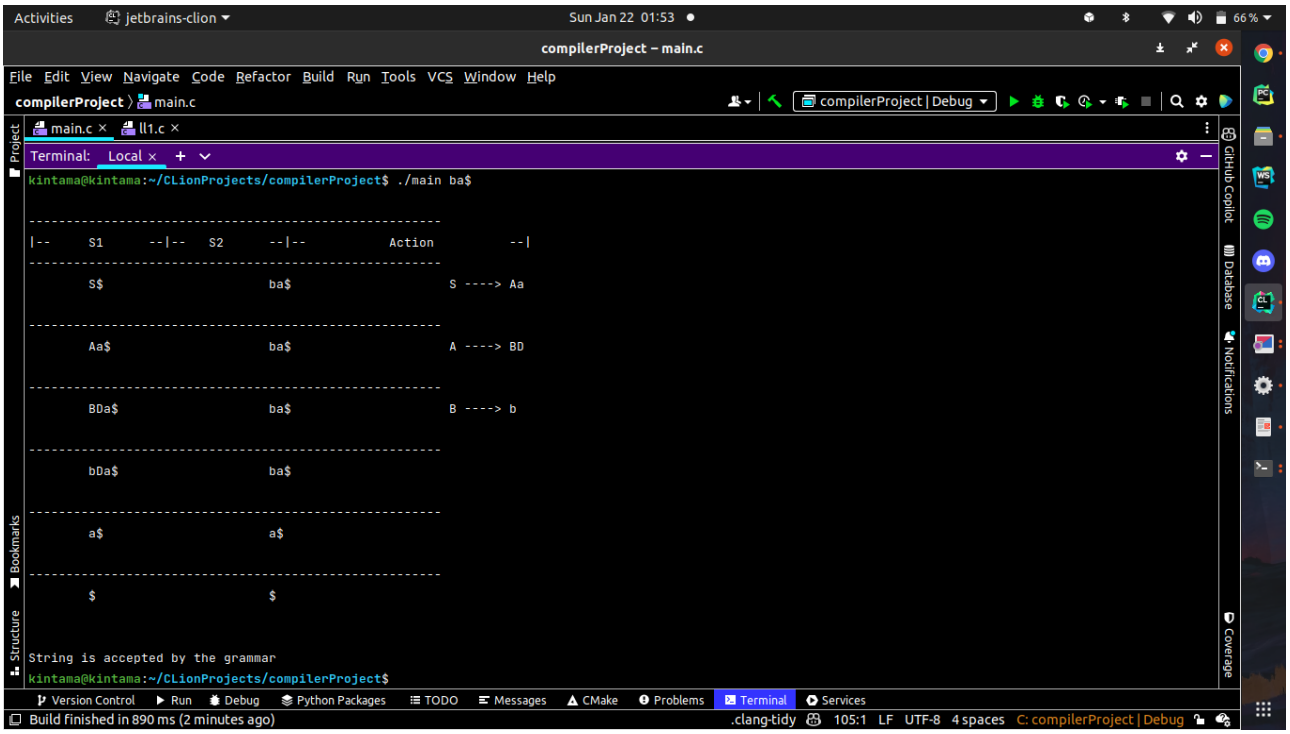
LL(1) LOGIC:

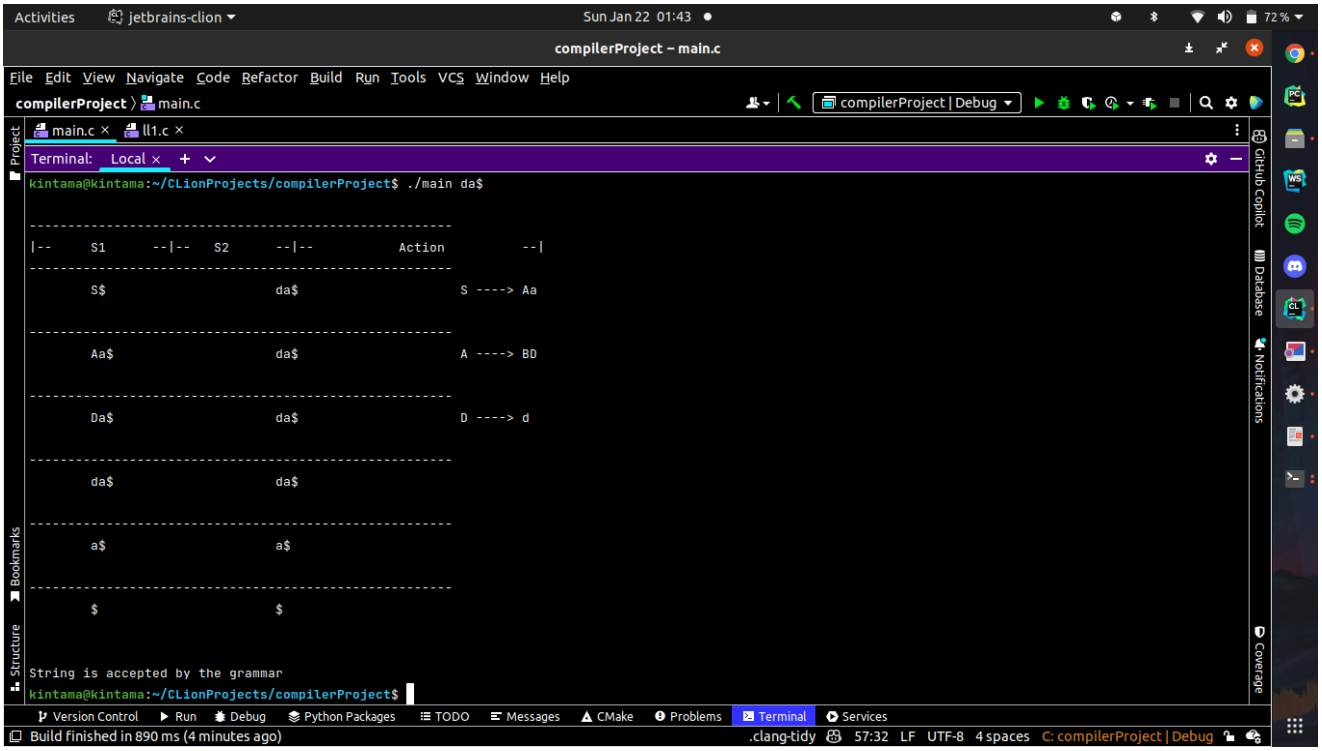
- The LL(1) parsing algorithm is implemented in the parse() function. The function starts by initializing the stack1 and stack2 pointers to -1. It then reads the input string and pushes it to stack2. The function then enters a while loop that continues until the stack1 pointer becomes -1.
- The function starts by reading the topmost symbol from stack1 and the current symbol from stack2. It then looks up the corresponding parse table entry for the state represented by the topmost symbol in stack1 and the current symbol in stack2.
- If the entry is an "Accept" symbol, the function returns 1 indicating that the input string is accepted by the grammar. If the entry is an "_" symbol, the function returns 0 indicating that the input string is not accepted by the grammar.
- If the entry is an "R" symbol, the function performs a reduction operation. The function pops the topmost symbol from stack1 and pushes the corresponding right-hand side of the production rule onto the stack.
- If the entry is an "S" symbol, the function performs a shift operation. The function pushes the current symbol from stack2 onto stack1 and pops it from stack2.

LL(1) PARSE TABLE & OUTPUT:




Activities
jetbrains-clion
Sun Jan 22 01:44
71 %
compilerProject – main.c
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
compilerProject 〉 main.c
compilerProject | Debug
main.c
ll1.c
Terminal: Local
kintama@kintama:~/CLionProjects/compilerProject$ ./main a$
|-- S1 --|-- S2 --|-- Action --|
S$ a$ S ----> Aa
Aa$ a$ A ----> BD
a$ a$
$ $
String is accepted by the grammar
kintama@kintama:~/CLionProjects/compilerProject$
Version Control
Run
Debug
Python Packages
TODO
Messages
CMake
Problems
Terminal
Services
Build finished in 890 ms (5 minutes ago)
.clang-tidy
57:32
LF
UTF-8
4 spaces
C: compilerProject | Debug


Activities
jetbrains-clion
Sun Jan 22 01:53
66 %
compilerProject – main.c
File Edit View Navigate Code Refactor Build Run Tools VCS Window Help
compilerProject 〉 main.c
compilerProject | Debug
main.c
ll1.c
Terminal: Local
kintama@kintama:~/CLionProjects/compilerProject$ ./main ba$
|-- S1 --|-- S2 --|-- Action --|
S$ ba$ S ----> Aa
Aa$ ba$ A ----> BD
BDa$ ba$ B ----> b
bDa$ ba$
a$ a$
$ $
String is accepted by the grammar
kintama@kintama:~/CLionProjects/compilerProject$
Version Control
Run
Debug
Python Packages
TODO
Messages
CMake
Problems
Terminal
Services
Build finished in 890 ms (2 minutes ago)
.clang-tidy
105:1
LF
UTF-8
4 spaces
C: compilerProject | Debug



LR(0) CODE:

```
#include <stdio.h>
#include "string.h"
#include <stdlib.h>
#define MAX_STATES 7
#define MAX_SYMBOLS 5

int parse();
char pop1();
char pop2();
void get_stack1();
void get_stack2();
void push1(char to_push);
void push2(char to_push);
int convert_char(char chr);
char get_top1();
char stack1[50], stack2[50];
int stack1_pointer = -1, stack2_pointer = -1;
char *parse_table[MAX_STATES][MAX_SYMBOLS] = {{"S3", "S4", "_",
"1", "2"},
                                              {"_",  "_",  "Accept",
"_", "_"},
                                              {"S3", "S4", "_",
"_", "5"},
```

```
                                                {"S3", "S4", "_",
"_", "6"},
                                                {"R3", "R3", "R3",
"_", "_"},
                                                {"R1", "R1", "R1",
"_", "_"},
                                                {"R2", "R2", "R2",
"_", "_"}};

char *ruleset[4][2] = {{"E'", "E"},
                       {"E",  "BB"},
                       {"B",  "cB"},
                       {"B",  "d"}};


int main(int argc, char *argv[]) {
    if(argc != 2){
        printf("Incorrect number of arguments!\n");

        exit(0);
    }
    stack1_pointer = -1;
    stack2_pointer = -1;

    char *input_string = argv[1];

    // push input string to stack 2
    for (int i = (strlen(input_string) - 1); i >= 0; i--) {
        push2(input_string[i]);
    }

    int flag = parse();

    if (flag == 1) {
        printf("String is accepted by the grammar\n");
    } else {
        printf("String is not accepted by the grammar\n");
    }
    return 0;
}
char get_top1() {
    return stack1[stack1_pointer];
}
void push1(char to_push) {
    stack1_pointer += 1;
    stack1[stack1_pointer] = to_push;
```

```
}
void push2(char to_push) {
    stack2_pointer += 1;
    stack2[stack2_pointer] = to_push;
}
char pop1() {
    char rtn_data = stack1[stack1_pointer];
    stack1_pointer -= 1;
    return rtn_data;
}
char pop2() {
    char rtn_data = stack2[stack2_pointer];
    stack2_pointer -= 1;
    return rtn_data;
}
void get_stack1() {
    for (int i = 0; (i <= stack1_pointer); i++) {
        printf("%c", stack1[i]);
    }
}
void get_stack2() {
    for (int i = stack2_pointer; i >= 0; i--) {
        printf("%c", stack2[i]);
    }
}
int convert_char(char chr) {
    switch (chr) {
        case 'c':
            return 0;
        case 'd':
            return 1;
        case '$':
            return 2;
        case 'E':
            return 3;
        case 'B':
            return 4;
        default:
            return -1;
    }
}

int parse() {
    char current_char = 0;
    int current_state = 0;
    char *temp;
```

```
    int repeat = 0;
    char next_state;
    int printed_stack = 0;

    printf("\n---------------------------------------------------------\n"
           "|--\tStack S1\t--|--\tInput S2\t--|--\tAction\t--|\n");
    push1('$');
    push1('0');

    while (current_char != '$') {
        if (printed_stack == 0){
            current_char = pop2();
        if (temp[0] == 'S') {
            push1(current_char);
            push1(temp[1]);
            current_state = temp[1] - '0';
        } else if (temp[0] == 'R') {
            push2(current_char);
            while (temp[0] == 'R') {
                int rule_num = temp[1] - '0';
                for (int i = 0; i < (strlen(ruleset[rule_num][1]) * 2);
i++) {
                    pop1();
                }

                if (repeat == 1) {
                    char reduce_by = ruleset[rule_num][0][0];
                    char stack1_top = get_top1() - '0';
                    push1(reduce_by);
                    next_state =
parse_table[stack1_top][convert_char(reduce_by)][0];
                    push1(next_state);
                    repeat = 0;
                } else {
                    for (int i = 0; i < (strlen(ruleset[rule_num][1]));
i++) {
                        char reduce_by = ruleset[rule_num][0][i];
                        char stack1_top = get_top1() - '0';
                        push1(reduce_by);
                        next_state =
parse_table[stack1_top][convert_char(reduce_by)][0];
                        push1(next_state);
                        current_state = next_state - '0';
                    }
                }
                printf("\t\t\t\t%s\t\t\n\n", temp);
```

```
                printed_stack = 1;
                if (temp[0] == 'R') {
                    repeat = 1;
                }
            }
        } else if (strcmp(temp, "Accept") == 0) {
            return 1;
        } else if (temp[0] == '_') {
            return 0;
        } else {
            printf("Error: Invalid symbol %c detected\n", current_char);
            return 0;
        }
    }
    return 1;
}
```

LL(1) CODE:

```
#include <stdio.h>
#include "string.h"
#include <stdlib.h>
#define MAX_STATES 4
#define MAX_SYMBOLS 4

int parse();
char pop1();
char pop2();
void get_stack1();
void get_stack2();
void push1(char to_push);
void push2(char to_push);
int convert_char(char chr);
char get_top1();
char get_top2();
int check_tops();
void print_stacks(char* temp, char* temp2);
char stack1[50], stack2[50];
int stack1_pointer = -1, stack2_pointer = -1;
// change parse_table and ruleset to match your grammar
int parse_table[MAX_STATES][MAX_SYMBOLS] = {{0, 0, 0, -1},
                                            {1, 1, 1, 1},
                                            {2, 3, 3, 3},
                                            {-1, 5, 4, 5}};
```

```
char *ruleset[6][2] = {{"S", "Aa"},
                       {"A",  "BD"},
                       {"B",  "b"},
                       {"B",  "epsilon"},
                       {"D",  "d"},
                       {"D",  "epsilon"}};

char* action;
char next_state;

int main(int argc, char *argv[]) {
    if(argc != 2){
        printf("Incorrect number of arguments!\n");

        exit(0);
    }

    stack1_pointer = -1;
    stack2_pointer = -1;

    char *input_string = argv[1];

    // push input string to stack 2
    for (int i = (strlen(input_string) - 1); i >= 0; i--) {
        push2(input_string[i]);
    }

    int flag = parse();

    if (flag == 1) {
        printf("\nString is accepted by the grammar\n");
    } else {
        printf("\nString is not accepted by the grammar\n");
    }
    return 0;
}


int parse() {
    char stack1_top;
    char stack2_top;

    printf("\n-------------------------------------------------------\n"
           "|--\tS1\t--|--\tS2\t--|--\t\tAction\t\t--|\n");
```

```
    push1('$');
    push1('S');

    goto start;

    start:
    if (check_tops() == 1) {
        goto accepted;
    }
    stack1_top = get_top1();
    stack2_top = get_top2();

    if (stack1_top == stack2_top) {
        goto start;
    }

    next_state =
parse_table[convert_char(stack1_top)][convert_char(stack2_top)];
    if (next_state == -1) {
        goto rejected;
    }
    action = ruleset[next_state][1];
    if (strcmp(action, "epsilon") == 0) {
        pop1();
        goto start;
    }
    goto start;
    rejected:
    return 0;
    accepted:
    print_stacks("", "");
    return 1;}
int check_tops() {
    char top1 = get_top1();
    char top2 = get_top2();
    if (top1 == '$' && top2 == '$') {
        return 1;
    } else if (top1 == top2) {
        print_stacks("", "");
        pop1();
        pop2();
        return 2;
    }
    return 0;
}
```