Experiment 3
Predictive Parsing

Grammar:

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

After removing ambiguity
(no ambiguity)

After removing left recursion
S->(L)|a
L->SA
A->,SA|epsilon

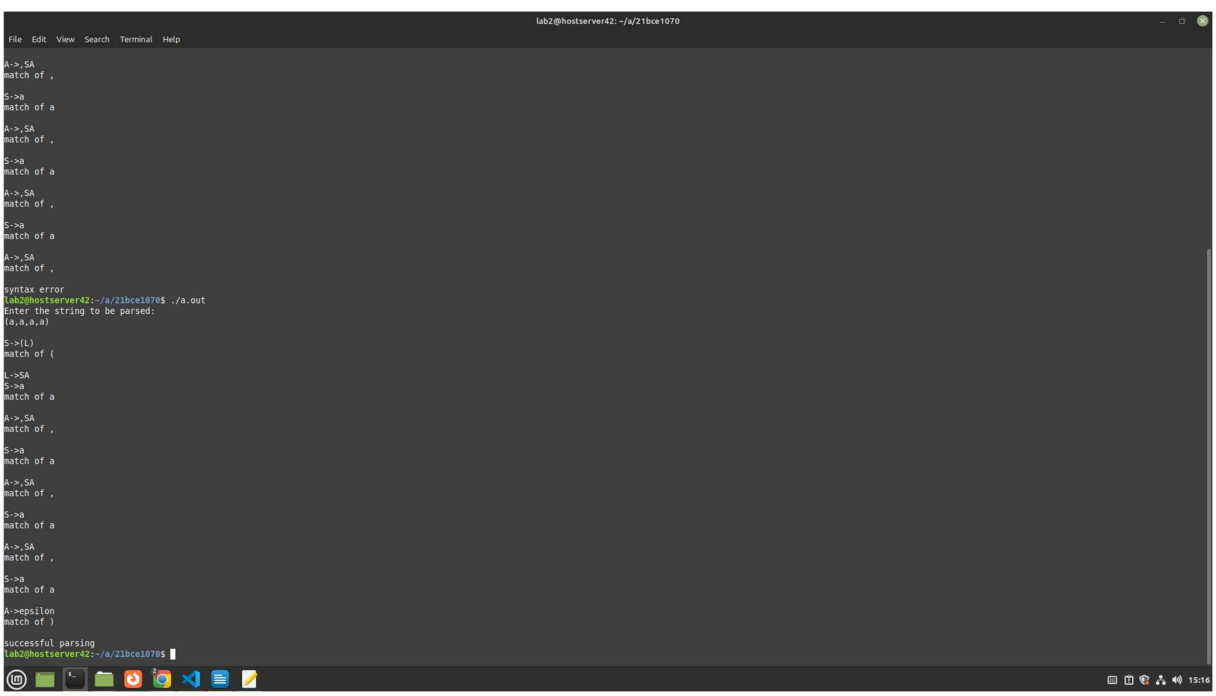
Code:

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

int i=0,top=0;
char stack[20],ip[20];
void push(char c)
{
if(top>=20)
printf("stack overflow");
else
stack[top++]=c;
}

void pop(){
if(top<0)
printf("stack underflow");
else
top--;}

void error()
{
printf("\nsyntax error\n");
```

```
exit(0);}



int main()
{
int n;

printf("Enter the string to be parsed:\n");

scanf("%s",ip);
n=strlen(ip);
ip[n]='$';
ip[n+1]='\0';
push('$');
push('S');
while(ip[i]!='\0')
{
if(ip[i]=='$' && stack[top-1]=='$')
{
printf("\nsuccessful parsing\n");
return 0;}
else if(ip[i]==stack[top-1])
{
printf("\nmatch of %c\n",ip[i]);
i++;
pop();
}


else
{
if(stack[top-1]=='S' && ip[i]=='a')
{
printf("\nS->a");
pop();
push('a');}

else if(stack[top-1]=='S' && ip[i]=='(')
{
printf("\nS->(L)");
pop();
push(')');
push('L');
push('(');
}
else if(stack[top-1]=='L' && (ip[i]=='a' || ip[i]=='('))
{
printf("\nL->SA");
pop();
push('A');
push('S');
```

```
}


else if(stack[top-1]=='A'&& ip[i]==',')
{
printf("\nA->,SA");
pop();
push('A');
push('S');
push(',');
}
else if(stack[top-1]=='A' && ip[i]==')')
{
printf("\nA->epsilon");
pop();}

else
error();
}
}
return 0;
}
```

Output:



```
lab2@hostserver42: ~/a/21bce1070
File  Edit  View  Search  Terminal  Help

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

syntax error
lab2@hostserver42:~/a/21bce1070$ ./a.out
Enter the string to be parsed:
(a,a,a,a)

S->(L)
match of (

L->SA
S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->epsilon
match of )

successful parsing
lab2@hostserver42:~/a/21bce1070$
```

```
File  Edit  View  Search  Terminal  Help
lab2@hostserver42:~$ cd /home/lab2/a/21bce1070
lab2@hostserver42:~/a/21bce1070$ g++ predictive_parser.c
lab2@hostserver42:~/a/21bce1070$ ./a.out
Enter the string to be parsed:
(a)

S->(L)
match of (

L->SA
S->a
match of a

A->epsilon
match of )

successful parsing
```

```
lab2@hostserver42:~/a/21bce1070$ ./a.out
Enter the string to be parsed:
(a,a,a,a,)

S->(L)
match of (

L->SA
S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

syntax error
```

```
lab2@hostserver42:~/a/21bce1070$ ./a.out
Enter the string to be parsed:
(a,a,a,a)

S->(L)
match of (

L->SA
S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->,SA
match of ,

S->a
match of a

A->epsilon
match of )

successful parsing
lab2@hostserver42:~/a/21bce1070$
```

Other Grammars:

S->SS|a|b

Ambiguity removal

S->ST|T
T->a|b

Remove Left recursion

S-> TX|
X-> TX|epsilon
T->a|b

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

int i = 0;
char input[20];

void match(char expected) {
   if (input[i] == expected) {
      i++;
   } else {
      printf("\nSyntax error: Expected '%c' but found '%c'\n", expected, input[i]);
      exit(0);
   }
}

void S();
void X();
void T();

void S() {
   if (input[i] == 'a' || input[i] == 'b') {
      T();
      X();
   } else {
      printf("\nSyntax error: Unexpected symbol '%c'\n", input[i]);
      exit(0);
   }
}

void X() {
   if (input[i] == 'a' || input[i] == 'b') {
      T();
      X();
   } else if (input[i] == '\0' || input[i] == '$') {
      // epsilon production
   } else {
      printf("\nSyntax error: Unexpected symbol '%c'\n", input[i]);
      exit(0);
   }
}

void T() {
   if (input[i] == 'a') {
      match('a');
   } else if (input[i] == 'b') {
      match('b');
   } else {
      printf("\nSyntax error: Unexpected symbol '%c'\n", input[i]);
      exit(0);
   }
```

```
}

int main() {
    printf("Enter the string to be parsed:\n");
    scanf("%s", input);

    S();  // Start the parsing process with the start symbol S

    if (input[i] == '\0') {
        printf("\nSuccessful parsing\n");
    } else {
        printf("\nSyntax error: Unexpected symbol '%c'\n", input[i]);
    }

    return 0;
}
```

```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ g++ CD.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ ./a.out
Enter the string to be parsed:
ababbab

Successful parsing

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ ./a.out
Enter the string to be parsed:
abbababa

Successful parsing

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ ./a.out
Enter the string to be parsed:
aaaaaa

Successful parsing

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ ./a.out
Enter the string to be parsed:
abcbcb

Syntax error: Unexpected symbol 'c'

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$
```

S->aA|bA
A->aA|bA|epsilon
(no ambiguity, left recursion, left factoring)

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

int i = 0, top = 0;
char stack[20], ip[20];

void push(char c) {
    if (top >= 20)
        printf("stack overflow");
    else
        stack[top++] = c;
}

void pop() {
    if (top < 0)
        printf("stack underflow");
    else
        top--;
}

void error() {
    printf("\nsyntax error\n");
    exit(0);
}

int main() {
    int n;
    printf("Enter the string to be parsed:\n");
    scanf("%s", ip);
    n = strlen(ip);
    ip[n] = '$';
    ip[n + 1] = '\0';
    push('$');
    push('S');

    while (ip[i] != '\0') {
        if (ip[i] == '$' && stack[top - 1] == '$') {
            printf("\nsuccessful parsing\n");
            return 0;
        } else if (ip[i] == stack[top - 1]) {
            printf("\nmatch of %c\n", ip[i]);
            i++;
```

```
            pop();
        } else {
            if (stack[top - 1] == 'S') {
                if (ip[i] == 'a') {
                    printf("\nS -> aA");
                    pop();
                    push('A');
                    push('a');
                } else if (ip[i] == 'b') {
                    printf("\nS -> bA");
                    pop();
                    push('A');
                    push('b');
                } else {
                    error();
                }
            } else if (stack[top - 1] == 'A') {
                if (ip[i] == 'a') {
                    printf("\nA -> aA");
                    pop();
                    push('A');
                    push('a');
                } else if (ip[i] == 'b') {
                    printf("\nA -> bA");
                    pop();
                    push('A');
                    push('b');
                } else if (ip[i] == '$' && stack[top - 1] == 'A') {
                    printf("\nA -> epsilon");
                    pop();
                } else {
                    error();
                }
            } else {
                error();
            }
        }
    }
    return 0;
}
```

```
┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ g++ CD.c

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ ./a.out
Enter the string to be parsed:
abba

S -> aA
match of a

A -> bA
match of b

A -> bA
match of b

A -> aA
match of a

A -> epsilon
successful parsing

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ ./a.out
Enter the string to be parsed:
abbb

S -> aA
match of a

A -> bA
match of b

A -> bA
match of b

A -> bA
match of b

A -> epsilon
successful parsing

┌──(kali1㉿kali)-[~/@1_DDrive/Code_Files]
└─$ ./a.out
Enter the string to be parsed:
abc

S -> aA
match of a

A -> bA
match of b

syntax error
```