LAB EXERCISES:

Write a recursive descent parser for the following simple grammars.

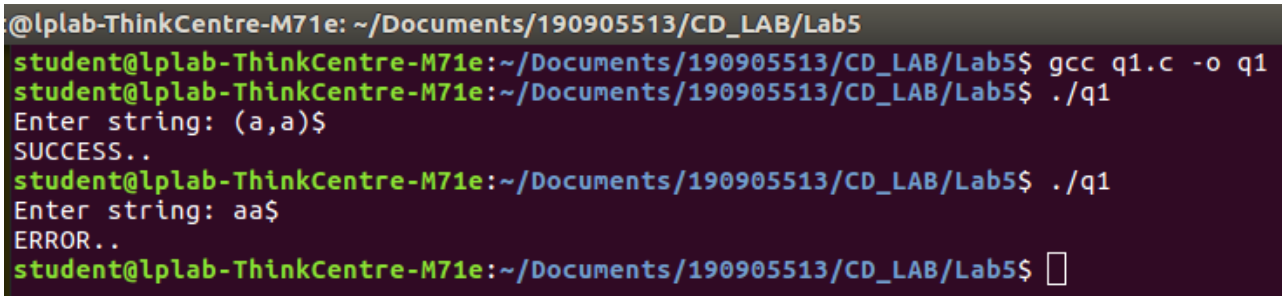
1. S-> a | > | (T)
T-> T , S | S

Program:

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

int curr=0;
char str[100];

void S();
void T();
void Tprime();

void reject()
{
  printf("ERROR..\n");
  exit(0);
}

void valid()
{
    printf("SUCCESS..\n");
    exit(0);
}

void S()
{
    if(str[curr]=='a')
    {
        curr++;
        return;
    }
    else if(str[curr]=='>')
    {
        curr++;
        return;
    }
    else if(str[curr]=='(')
```

```
    {
        curr++;
        T();
        if(str[curr]==')')
        {
            curr++;
            return;
        }
        else
            reject();
    }
    else
        reject();
}
void T()
{
    S();
    Tprime();
}

void Tprime()
{
    if(str[curr]==',')
    {
        curr++;
        S();
        return;
    }
}

int main()
{
    printf("Enter string: ");
    scanf("%s",str);
    S();
    if(str[curr]=='$')
    {
        valid();
    }
    else
         reject();
}
```

Output:



2. S-> UVW
U-> (S) | aSb | d
V-> aV | ε
W-> cW | ε

Follow(V) = {'c',')','b','$'}
Follow(W) = {')','b','$'}

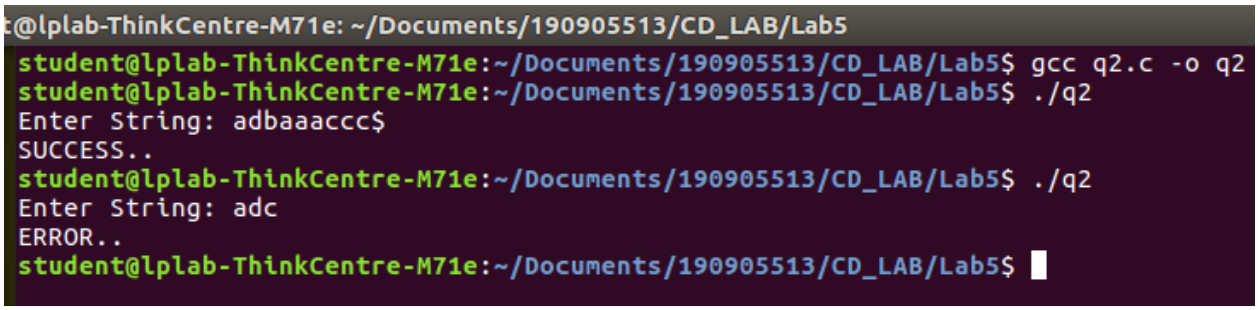
Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int curr = 0;
char str[100];

void S();
void U();
void V();
void W();

void invalid(){
     printf("ERROR..\n");
     exit(0);
}
void valid(){
     printf("SUCCESS..\n");
     exit(0);
}

void S(){
     U();
     V();
     W();
}

void U(){
     if(str[curr]=='('){
          curr++;
          S();
```

```
          if(str[curr]==')'){
               curr++;
          }
          else{
               invalid();
          }

     }
     else if(str[curr]=='a'){
          curr++;
          S();
          if(str[curr]=='b'){
               curr++;
          }
          else{
               invalid();
          }

     }
     else if(str[curr]=='d'){
          curr++;
     }
     else{
          invalid();
     }
}

void V(){
     if(str[curr]=='a'){
          curr++;
          V();

     }
     else if(str[curr]=='c'||str[curr]==')'||str[curr]=='b'||
str[curr]=='$'){
          return;
     }
     else{
          invalid();
     }
}

void W(){
     if(str[curr]=='c'){
          curr++;
          W();

     }
     else if(str[curr]==')'||str[curr]=='b'||str[curr]=='$'){
          return;
     }
     else{
```

```
          invalid();
     }
}

int main(){
     printf("Enter String: ");
     scanf("%s", str);
     S();
     if(str[curr] == '$')
          valid();
     else
          invalid();
}
```

Output:



3. S-> aAcBe
A-> Ab|b
B-> d

After removing left recursion the new grammer is:
S -> aAcBe
A -> bAprime
Aprime -> bAprime|e
B -> d

Follow(Aprime) = {'c'}

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int curr = 0;
char str[100];

void S();
void A();
void Aprime();
void B();

void invalid(){
```

```
     printf("ERROR..\n");
     exit(0);
}
void valid(){
     printf("SUCCESS..\n");
     exit(0);
}

void S(){
     if(str[curr]=='a'){
          curr++;
          A();
          if(str[curr]=='c'){
               curr++;
               B();
               if(str[curr]=='e'){
                    curr++;
               }
               else{
                    invalid();
               }
          }
          else{
               invalid();
          }
     }
     else{
          invalid();
     }
}


void A(){
     if(str[curr]=='b'){
          curr++;
          Aprime();

     }
     else{
          invalid();
     }
}

void Aprime(){
     if(str[curr]=='b'){
          curr++;
          Aprime();

     }
     else if(str[curr]=='c'){
          return;
     }
```

```
     else{
          invalid();
     }
}

void B(){
     if(str[curr]=='d'){
          curr++;

     }
     else{
          invalid();
     }
}

int main(){
     printf("Enter String: ");
     scanf("%s", str);
     S();
     if(str[curr] == '$')
          valid();
     else
          invalid();
}
```

Output:

```
@lplab-ThinkCentre-M71e: ~/Documents/190905513/CD_LAB/Lab5
student@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LAB/Lab5$ gcc q3.c -o q3
student@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LAB/Lab5$ ./q3
Enter String: abbbbbbbbbbcde$
SUCCESS..
student@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LAB/Lab5$ ./q3
Enter String: acbde$
ERROR..
student@lplab-ThinkCentre-M71e:~/Documents/190905513/CD_LAB/Lab5$
```

4. S-> (L) | a (s,s,s,s,s)
L-> L,S | S

After removing left recursion the new grammer is:
S -> (L)|a
L -> SLprime
Lprime -> ,SLprime|e

Follow(Lprime) = {')'}

Program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int curr = 0;
```

```
char str[100];

void S();
void L();
void Lprime();

void invalid(){
     printf("ERROR..\n");
     exit(0);
}
void valid(){
     printf("SUCCESS..\n");
     exit(0);
}

void S(){
     if(str[curr]=='('){
          curr++;
          L();
          if(str[curr]==')'){
               curr++;
          }
          else{
               invalid();
          }
     }
     else if(str[curr]=='a'){
          curr++;
     }
     else{
          invalid();
     }
}


void L(){
     S();
     Lprime();
}

void Lprime(){
     if(str[curr]==','){
          curr++;
          S();
          Lprime();

     }
     else if(str[curr]==')'){
          return;
     }
     else{
          invalid();
```

```
     }
}

int main(){
     printf("Enter String: ");
     scanf("%s", str);
     S();
     if(str[curr] == '$')
          valid();
     else
          invalid();
}
```

Output: