CD Lab #5

Question 1

// Write a recursive descent parser for the following simple grammars.

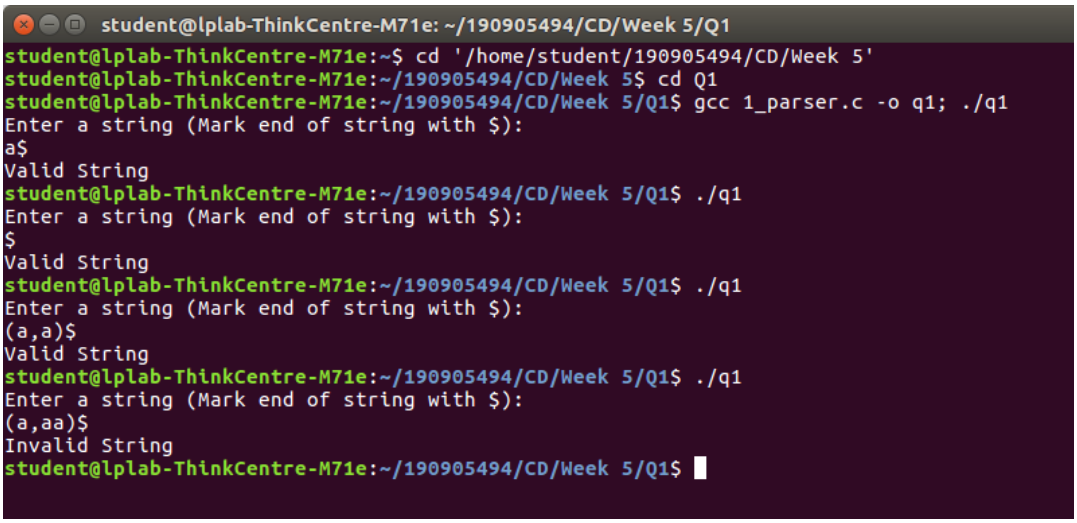
/*
Grammar:
S -> a | > | (T)
T -> T,S|S

We need to eliminate left recursion in the production of T
T -> ST'
T' -> ,ST' | ε

Therefore, the grammar is:
S -> a | > | (T)
T -> ST'
T' -> ,ST' | ε
*/

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

int curr = 0;
char str[512];
void S();
void T();
void T_();

void invalid()
{
	printf("Invalid String\n");
	exit(-1);
}

void valid()
{
	printf("Valid String\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 {
                        invalid();
                }
        }
}

void T()
{
        S();
        T_();
}

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


int main()
{
        printf("Enter a string (Mark end of string with $):\n");
        scanf("%s", str);
        S();

        if (str[curr] == '$') {
                valid();
        } else {
                invalid();
        }

        return 0;
}
```



Question 2

// Write a recursive descent parser for the following simple grammars.

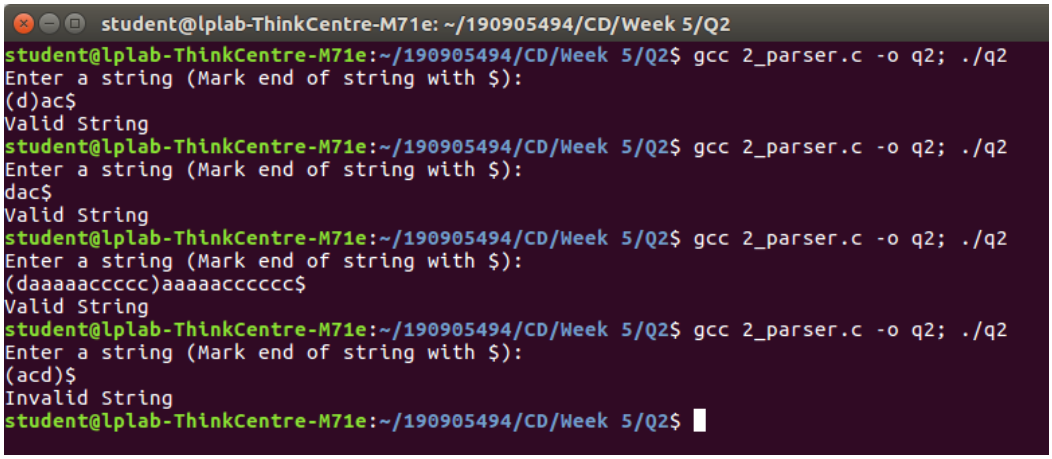
/* Grammar:
S -> UVW
U -> (S) | aSb | d
V -> aV | ε
W -> cW | ε

There is no left recurison in this grammar
*/

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


int curr = 0;
char str[512];

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

void invalid() {
        printf("Invalid String\n");
        exit(-1);
}

void valid()
```

```
{
	printf("Valid String\n");
	exit(0);
}

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

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

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

void W()
```

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


int main()
{
        printf("Enter a string (Mark end of string with $):\n");
        scanf("%s", str);
        S();
        if (str[curr] == '$') {
                valid();
        }
        else {
                invalid();
        }
        return 0;
}
```



Question 3

// Write a recursive descent parser for the following simple grammars.

/*
Grammmar:
S -> aAcBe
A -> Ab|b
B -> d

The production rule for A has left recursion which needs to be eliminated
A -> bA'
A' -> bA' | ε

Therefore, the rules are:
S -> aAcBe
A -> bA'
A' -> bA' | ε
B -> d
*/

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


int curr = 0;
char str[512];

void S();
void A();
void A_();
void B();

void invalid()
{
	printf("Invalid String\n");
	exit(-1);
}

void valid()
{
	printf("Valid String\n");
	exit(0);
}

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

```
	else {
		invalid();
	}
}

void A()
{
	if (str[curr] == 'b') {
		curr++;
		A_();
		return;
	}
	else {
		invalid();
	}
}

void A_()
{
	if (str[curr] == 'b') {
		curr++;
		A_();
	}
	return;
}

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


int main()
{
	printf("Enter a string (Mark end of string with $):\n"); scanf("%s", str);
	S();
	if (str[curr] == '$') {
		valid();
	}
	else {
		invalid();
	}
	return 0;
}
```

Question 4

// Write a recursive descent parser for the following simple grammars.

/*
Grammar:
S -> (L) | a
L -> L,S | S

The production rule for L is left recursive. This needs to be eliminated
L -> SL'
L' -> ,SL' | ε

Therefore, the grammar is:
S -> (L) | a
L -> SL'
L' -> ,SL' | ε
*/

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


int curr = 0;
char str[512];

void S();
void L();
void L_();

void invalid()
{
	printf("Invalid String\n");
	exit(-1);
}

void valid()
{
```

```
	printf("Valid String\n");
	exit(0);
}

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

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

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

int main()
{
	printf("Enter a string (Mark end of string with $):\n");
	scanf("%s", str);
	S();
	if (str[curr] == '$') {
		valid();
	}
	else {
		invalid();
```

```
    }
    return 0;
}
```


student@lplab-ThinkCentre-M71e: ~/190905494/CD/Week 5/Q4
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 5/Q4$ gcc 4_parser.c -o q4; ./q4
Enter a string (Mark end of string with $):
a$
Valid String
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 5/Q4$ gcc 4_parser.c -o q4; ./q4
Enter a string (Mark end of string with $):
(a,a)$
Valid String
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 5/Q4$ gcc 4_parser.c -o q4; ./q4
Enter a string (Mark end of string with $):
(a,a,a,a,a)$
Valid String
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 5/Q4$ gcc 4_parser.c -o q4; ./q4
Enter a string (Mark end of string with $):
(a,,a$
Invalid String
student@lplab-ThinkCentre-M71e:~/190905494/CD/Week 5/Q4$