26/02/2021

A4: Recursive Descent Parser using C

185001188
Vanathi G
CSE-C

PROGRAMS -

```
/* G:   E -> E+T|T
     T -> T*F | F
     F -> i
*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXLEN 40

typedef struct
{
        int ptr; // points to "lookahead" basically
        char string[MAXLEN];
}buffer;

void E(buffer *inp); // need pointers as inp because we will have to modify lookahead ptr
void T(buffer *inp);
void EPrime(buffer *inp);
void TPrime(buffer *inp);
void F(buffer *inp);

void main()
{
        buffer *inp;
        inp = malloc(sizeof(buffer));
        inp->ptr = 0;
        scanf("%s", inp->string);
        strcat(inp->string, "$");

        E(inp);
        if(inp->string[inp->ptr] == '$')
```

```
                printf("Success\n");
        else
                printf("Not derived by this grammar\n");
}

// E -> TE'
void E(buffer *inp)
{
        T(inp);
        EPrime(inp);
}

//T -> FT'
void T(buffer *inp)
{
        F(inp);
        TPrime(inp);
}

//E' -> +TE' | epsilon
void EPrime(buffer *inp)
{
        // if the current symbol is + we need to call T and E'
        if(inp->string[inp->ptr] == '+')
        {
                inp->ptr++;
                T(inp);
                EPrime(inp);
        }
        // otherwise for epsilon we just return
        return;
}

//T' -> *FT' | epsilon
void TPrime(buffer *inp)
{
        if(inp->string[inp->ptr] == '*')
        {
                inp->ptr++;
                F(inp);
                TPrime(inp);
        }
        return;
}
```

```
//F -> i
void F(buffer *inp)
{
	if(inp->string[inp->ptr] == 'i')
		inp->ptr++;
	else
	{
		printf("Not derived by this grammar\n");
		exit(0);
	}
	return;
}
```

/* G: E -> E+T|E-T|T

T -> T*F | T/F|F

F -> (E)|i

***/**

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define MAXLEN 40

typedef struct
{
	int ptr; // points to "lookahead" basically
	char string[MAXLEN];
}buffer;

void E(buffer *inp); // need pointers as inp because we will have to modify lookahead ptr
void T(buffer *inp);
void EPrime(buffer *inp);
void TPrime(buffer *inp);
void F(buffer *inp);

void main()
{
	buffer *inp;
	inp = malloc(sizeof(buffer));
	inp->ptr = 0;
	scanf("%s", inp->string);
	strcat(inp->string, "$");
```

```
	E(inp);
	if(inp->string[inp->ptr] == '$')
		printf("Success\n");
	else
		printf("Not derived by this grammar\n");
}

// E -> TE'
void E(buffer *inp)
{
	T(inp);
	EPrime(inp);
}

//T -> FT'
void T(buffer *inp)
{
	F(inp);
	TPrime(inp);
}

//E' -> +TE' | epsilon
void EPrime(buffer *inp)
{
	// if the current symbol is + we need to call T and E'
	if(inp->string[inp->ptr] == '+' || inp->string[inp->ptr] == '-')
	{
		inp->ptr++;
		T(inp);
		EPrime(inp);
	}
	// otherwise for epsilon we just return
	return;
}

//T' -> *FT' | epsilon
void TPrime(buffer *inp)
{
	if(inp->string[inp->ptr] == '*' || inp->string[inp->ptr] == '/')
	{
		inp->ptr++;
		F(inp);
		TPrime(inp);
```

```
	}
	return;
}

//F -> i
void F(buffer *inp)
{
	if(inp->string[inp->ptr] == 'i')
		inp->ptr++;
	else if(inp->string[inp->ptr] == '(')
	{
		inp->ptr++;
		E(inp);
		if(inp->string[inp->ptr] == ')')
			inp->ptr++;
		else
		{
			printf("Not derived by this grammar\n");
			exit(0);
		}

	}
	else
	{
		printf("Not derived by this grammar\n");
		exit(0);
	}
	return;
}
```

I/O SNAPSHOTS -