1.Write a C program to represent a Grammar in (VN, ∑, P, S ) format. Also check a) whether production exists for all non-terminals b) Terminals used in productions are in ∑

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define SIZE 20

int main(){

    char terminals[SIZE],non\_terminals[SIZE];

    printf("\nEnter the set of terminals : ");

    gets(terminals);

    printf("\nEnter the set non-terminals: ");

    gets(non\_terminals);

    int num;

    printf("\nEnter the number of production rules for the grammer: ");

    scanf("%d",&num);

    char production[num][SIZE];

    getchar();

    for(int i=1;i<=num;i++){

        printf("\nEnter the %d production rule: ",i);

        gets(production[i-1]);

    }

    char start\_symbol;

    printf("\nEnter the stat symbol of the grammer: ");

    scanf("%c",&start\_symbol);

    printf("\nGiven Gramme is = ");

    printf("\nSet of non-terminals: ");

    printf("{ ");

    for(int i=0;i<strlen(non\_terminals);i++){

        printf("%c, ",non\_terminals[i]);

    }

    printf("}");

    printf("\nSet of terminals: ");

    printf("{ ");

    for(int i=0;i<strlen(terminals);i++){

        printf("%c, ",terminals[i]);

    }

    printf("}");

    printf("\nProduction rules: \n");

    for(int i=0;i<num;i++){

        puts(production[i]);

    }

    printf("\nStart Symbol: %c",start\_symbol);

    int valid=0;

    for(int i=0;i<strlen(non\_terminals);i++){

        valid=0;

        for(int j=0;j<num;j++){

            if(non\_terminals[i]==production[j][0]){

                valid=1;

                break;

            }

        }

        if(!valid){

            printf("\n No production exists for the non-terminal: %c",non\_terminals[i]);

            return 1;

        }

    }

    printf("\nProduction rule exists for all non-teminals!!");

    for(int i=0;i<num;i++){

        valid=0;

        for(int j=3;j<strlen(production[i]);j++){

            valid=0;

            if(production[i][j]=='|') continue;

            for(int k=0;k<strlen(non\_terminals);k++){

                if(production[i][j]==non\_terminals[k]){

                    valid=1;

                    break;

                }

            }

            if(valid) continue;

            for(int k=0;k<strlen(terminals);k++){

                if(production[i][j]==terminals[k]){

                    valid=1;

                    break;

                }

            }

            if(!valid){

                printf("\n%c is not in the set of non-terminals: ",production[i][j]);

                return 1;

            }

        }

    }

    printf("\nAll the terminals present in producation rules are also present in the given set of terminals!!");

    return 0;

}

2.If there exist multiple productions from a single non-terminal group them into one and represent as N α1/ α2/ ....../ αn

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define SIZE 20

typedef struct production{

    char\* str;

    int len;

}production;

int isNT(char c) {

    return c >= 'A' && c <= 'Z';

}

int nonterminal[26];

int main(){

    int num\_of\_prod;

    printf("Enter the number of production rules: ");

    scanf("%d",&num\_of\_prod);

    production prod[26];

    for(int i=0;i<26;i++){

        prod[i].str=NULL;

        prod[i].len=0;

    }

    char buff[256];

    getchar();

    for(int i=0;i<num\_of\_prod;i++){

        printf("\nEnter production %d :",i+1);

        scanf("%s",buff);

        if(nonterminal[buff[0]-'A']){

            int length=prod[buff[0]-'A'].len+strlen(buff)+1-3;

            prod[buff[0]-'A'].str=realloc(prod[buff[0]-'A'].str,length\*sizeof(char));

            int idx=prod[buff[0]-'A'].len-1,j=3;

            prod[buff[0]-'A'].str[idx]='|';

            idx++;

            while(buff[j]){

                prod[buff[0]-'A'].str[idx]=buff[j];

                idx++;

                j++;

            }

            prod[buff[0]-'A'].str[idx]='\0';

            prod[buff[0]-'A'].len=length;

        }else{

            int length=strlen(buff)+1;

            prod[buff[0]-'A'].str=malloc(length\*(sizeof(char)));

            prod[buff[0]-'A'].len=length;

            strcpy(prod[buff[0]-'A'].str,buff);

            nonterminal[buff[0]-'A']=1;

        }

    }

    printf("\nProduction rules after grouping:");

    for(int i=0;i<26;i++){

        if(!prod[i].len) continue;

        printf("\n%s",prod[i].str);

    }

    return 0;

}

4.C program to Calculate the FIRST for each non-terminal symbol in the grammar.

//character = '~' is used as epsilon

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define TNUM 128

typedef struct production{

    char\* str;

    int len;

}production;

int isNT(char c) {

    return c >= 'A' && c <= 'Z';

}

int nonterminal[26],terminal[TNUM],first[26][TNUM];

void add\_firsts(char A,char B){

    for(int i=0;i<TNUM;i++){

        first[A-'A'][i]=first[A-'A'][i]||first[B-'A'][i];

    }

}

void find\_first(production \*prod,int num){

    int t=0;

    while(t<num){

        for(int i=0;i<num;++i){

            for(int j=3;j<prod[i].len;++j){

                char ch=prod[i].str[j];

                if(isNT(ch)){

                    add\_firsts(prod[i].str[0],ch);

                    if(first[ch-'A']['~']) continue;

                }else{

                    first[prod[i].str[0]-'A'][ch]=1;

                }

                break;

            }

        }

        t++;

    }

}

int main(){

    int num\_of\_prod,maxprod,idx=0,j;

    printf("Enter the number of production rules: ");

    scanf("%d",&num\_of\_prod);

    maxprod=num\_of\_prod;

    production\* prod=malloc(maxprod\*sizeof(production));

    char buff[256],temp[256];

    getchar();

    for(int i=0;i<num\_of\_prod;i++){

        printf("\nEnter the %d production: ",i+1);

        fgets(buff,256,stdin);

        j=0;

        nonterminal[buff[0]-'A']=1;

        for(int k=0;k<strlen(buff)-1;k++){

            if(buff[k]=='|'){

                prod[idx].str=malloc((j+1)\*sizeof(char));

                for(int len=0;len<j;len++) prod[idx].str[len]=temp[len];

                idx++;

                if(idx>=maxprod){

                    maxprod+=maxprod;

                    prod=realloc(prod,maxprod\*sizeof(production));

                    if(!prod){

                        printf("memory allocation error, exiting!!\n");

                        exit(1);

                    }

                }

                prod[idx-1].str[j]='\0';

                prod[idx-1].len=j;

                temp[0]=prod[idx-1].str[0];

                temp[1]='-';

                temp[2]='>';

                j=3;

            }else{

                temp[j]=buff[k];

                j++;

                if(!isNT(buff[k]) && buff[k]!='-' && buff[k]!='>'){

                    terminal[buff[k]]=1;

                }

            }

        }

        prod[idx].str=malloc((j+1)\*sizeof(char));

        for(int len=0;len<j;len++) prod[idx].str[len]=temp[len];

        prod[idx].str[j]='\0';

        prod[idx].len=j;

        idx++;

        if(idx>=maxprod){

            maxprod+=maxprod;

            prod=realloc(prod,maxprod\*sizeof(production));

            if(!prod){

                printf("memory allocation error, exiting!!\n");

                exit(1);

            }

        }

    }

    prod[idx].len=0;

    find\_first(prod,idx);

    for(int i=0;i<26;i++){

        if(nonterminal[i]){

            printf("\nFIRST of %c: {",i+'A');

            for(int j=0;j<TNUM;j++){

                if(first[i][j]) printf(" %c,",j);

            }

            printf(" }");

        }

    }

    return 0;

}

6. C program to Calculate the FOLLOW for each non-terminal symbol in the

grammar.

//character = '~' is used as epsilon

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define TNUM 128

typedef struct production{

    char\* str;

    int len;

}production;

int isNT(char c) {

    return c >= 'A' && c <= 'Z';

}

int nonterminal[26],terminal[TNUM],first[26][TNUM],follow[26][TNUM];

void add\_first\_to\_first(char A,char B){

    for(int i=0;i<TNUM;i++){

        if(i=='~') continue;

        first[A-'A'][i]=first[A-'A'][i]||first[B-'A'][i];

    }

}

void add\_first\_to\_follow(char A,char B){

    for(int i=0;i<TNUM;++i){

        if(i=='~') continue;

        follow[A-'A'][i]=follow[A-'A'][i]||first[B-'A'][i];

    }

}

void add\_follow\_to\_follow(char A,char B){

    for(int i=0;i<TNUM;++i){

        if(i=='~') continue;

        follow[A-'A'][i]=follow[A-'A'][i]||follow[B-'A'][i];

    }

}

void find\_first(production \*prod,int num){

    int t=0;

    while(t<num){

        for(int i=0;i<num;++i){

            for(int j=3;j<prod[i].len;++j){

                char ch=prod[i].str[j];

                if(isNT(ch)){

                    add\_first\_to\_first(prod[i].str[0],ch);

                    if(first[ch-'A']['~']) continue;

                }else{

                    first[prod[i].str[0]-'A'][ch]=1;

                }

                break;

            }

        }

        t++;

    }

}

void find\_follow(production \*prod,int num){

    int t=0;

    while(t<num){

        for(int k=0;k<26;k++){

            if(!nonterminal[k]) continue;

            char nt=k+'A';

            for(int i=0;i<num;i++){

                for(int j=3;j<prod[i].len;++j){

                    fflush(0);

                    if(nt==prod[i].str[j]){

                        int x;

                        for(x=j+1;x<prod[i].len;x++){

                            fflush(0);

                            char ch=prod[i].str[x];

                            if(isNT(ch)){

                                add\_first\_to\_follow(nt,ch);

                                if(first[ch-'A']['~']) continue;

                            }else{

                                follow[nt-'A'][ch]=1;

                            }

                            fflush(0);

                            break;

                        }

                        if(x==prod[i].len) add\_follow\_to\_follow(nt,prod[i].str[0]);

                    }

                }

            }

        }

        t++;

    }

}

int main(){

    int num\_of\_prod,maxprod,idx=0,j;

    printf("Enter the number of production rules: ");

    scanf("%d",&num\_of\_prod);

    maxprod=num\_of\_prod;

    production\* prod=malloc(maxprod\*sizeof(production));

    char buff[256],temp[256];

    getchar();

    for(int i=0;i<num\_of\_prod;i++){

        printf("\nEnter the %d production: ",i+1);

        fgets(buff,256,stdin);

        j=0;

        nonterminal[buff[0]-'A']=1;

        for(int k=0;k<strlen(buff)-1;k++){

            if(buff[k]=='|'){

                prod[idx].str=malloc((j+1)\*sizeof(char));

                for(int len=0;len<j;len++) prod[idx].str[len]=temp[len];

                idx++;

                if(idx>=maxprod){

                    maxprod+=maxprod;

                    prod=realloc(prod,maxprod\*sizeof(production));

                    if(!prod){

                        printf("memory allocation error, exiting!!\n");

                        exit(1);

                    }

                }

                prod[idx-1].str[j]='\0';

                prod[idx-1].len=j;

                temp[0]=prod[idx-1].str[0];

                temp[1]='-';

                temp[2]='>';

                j=3;

            }else{

                temp[j]=buff[k];

                j++;

                if(!isNT(buff[k]) && buff[k]!='-' && buff[k]!='>'){

                    terminal[buff[k]]=1;

                }

            }

        }

        prod[idx].str=malloc((j+1)\*sizeof(char));

        for(int len=0;len<j;len++) prod[idx].str[len]=temp[len];

        prod[idx].str[j]='\0';

        prod[idx].len=j;

        idx++;

        if(idx>=maxprod){

            maxprod+=maxprod;

            prod=realloc(prod,maxprod\*sizeof(production));

            if(!prod){

                printf("memory allocation error, exiting!!\n");

                exit(1);

            }

        }

    }

    prod[idx].len=0;

    find\_first(prod,idx);

    follow[prod[0].str[0]-'A']['$']=1;

    find\_follow(prod,idx);

    for(int i=0;i<26;i++){

        if(nonterminal[i]){

            printf("\nFOLLOW of %c: {",i+'A');

            for(int j=0;j<TNUM;j++){

                if(follow[i][j]) printf(" %c,",j);

            }

            printf(" }");

        }

    }

    return 0;

}

8. C program to construct a LL(1) parsing table.

//character = '~' is used as epsilon

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

#define TNUM 128

typedef struct production{

    char\* str;

    int len;

}production;

int isNT(char c) {

    return c >= 'A' && c <= 'Z';

}

int nonterminal[26],terminal[TNUM],first[26][TNUM],follow[26][TNUM];

int first\_beta[TNUM],table[26][TNUM];

void add\_first\_to\_first(char A,char B){

    for(int i=0;i<TNUM;i++){

        if(i=='~') continue;

        first[A-'A'][i]=first[A-'A'][i]|first[B-'A'][i];

    }

}

void add\_first\_to\_follow(char A,char B){

    for(int i=0;i<TNUM;++i){

        if(i=='~') continue;

        follow[A-'A'][i]=follow[A-'A'][i]|first[B-'A'][i];

    }

}

void add\_follow\_to\_follow(char A,char B){

    for(int i=0;i<TNUM;++i){

        if(i=='~') continue;

        follow[A-'A'][i]=follow[A-'A'][i]|follow[B-'A'][i];

    }

}

void find\_first(production \*prod,int num){

    int t=0;

    while(t<num){

        for(int i=0;i<num;++i){

            for(int j=3;j<prod[i].len;++j){

                char ch=prod[i].str[j];

                if(isNT(ch)){

                    add\_first\_to\_first(prod[i].str[0],ch);

                    if(first[ch-'A']['~']) continue;

                }else{

                    first[prod[i].str[0]-'A'][ch]=1;

                }

                break;

            }

        }

        t++;

    }

}

void find\_follow(production \*prod,int num){

    int t=0;

    while(t<num){

        for(int k=0;k<26;k++){

            if(!nonterminal[k]) continue;

            char nt=k+'A';

            for(int i=0;i<num;i++){

                for(int j=3;j<prod[i].len;++j){

                    fflush(0);

                    if(nt==prod[i].str[j]){

                        int x;

                        for(x=j+1;x<prod[i].len;x++){

                            fflush(0);

                            char ch=prod[i].str[x];

                            if(isNT(ch)){

                                add\_first\_to\_follow(nt,ch);

                                if(first[ch-'A']['~']) continue;

                            }else{

                                follow[nt-'A'][ch]=1;

                            }

                            fflush(0);

                            break;

                        }

                        if(x==prod[i].len) add\_follow\_to\_follow(nt,prod[i].str[0]);

                    }

                }

            }

        }

        t++;

    }

}

void find\_first\_beta(char \*str,int len){

    for(int i=3;i<len;i++){

        if(isNT(str[i])){

            for(int j=0;j<TNUM;j++){

                if(j=='~') continue;

                first\_beta[j]=first\_beta[j]|first[str[i]-'A'][j];

            }

            if(first[str[i]-'A']['~']) continue;

        }else first\_beta[str[i]]=1;

        break;

    }

}

int main(){

    int num\_of\_prod,maxprod,idx=0,j;

    printf("Enter the number of production rules: ");

    scanf("%d",&num\_of\_prod);

    maxprod=num\_of\_prod;

    production\* prod=malloc(maxprod\*sizeof(production));

    char buff[256],temp[256];

    getchar();

    for(int i=0;i<num\_of\_prod;i++){

        printf("\nEnter the %d production: ",i+1);

        fgets(buff,256,stdin);

        j=0;

        nonterminal[buff[0]-'A']=1;

        for(int k=0;k<strlen(buff)-1;k++){

            if(buff[k]=='|'){

                prod[idx].str=malloc((j+1)\*sizeof(char));

                for(int len=0;len<j;len++) prod[idx].str[len]=temp[len];

                idx++;

                if(idx>=maxprod){

                    maxprod+=maxprod;

                    prod=realloc(prod,maxprod\*sizeof(production));

                    if(!prod){

                        printf("memory allocation error, exiting!!\n");

                        exit(1);

                    }

                }

                prod[idx-1].str[j]='\0';

                prod[idx-1].len=j;

                temp[0]=prod[idx-1].str[0];

                temp[1]='-';

                temp[2]='>';

                j=3;

            }else{

                temp[j]=buff[k];

                j++;

                if(!isNT(buff[k]) && buff[k]!='-' && buff[k]!='>'){

                    terminal[buff[k]]=1;

                }

            }

        }

        prod[idx].str=malloc((j+1)\*sizeof(char));

        for(int len=0;len<j;len++) prod[idx].str[len]=temp[len];

        prod[idx].str[j]='\0';

        prod[idx].len=j;

        idx++;

        if(idx>=maxprod){

            maxprod+=maxprod;

            prod=realloc(prod,maxprod\*sizeof(production));

            if(!prod){

                printf("memory allocation error, exiting!!\n");

                exit(1);

            }

        }

    }

    prod[idx].len=0;

    find\_first(prod,idx);

    follow[prod[0].str[0]-'A']['$']=1;

    find\_follow(prod,idx);

    terminal['$']=1;

    terminal['~']=0;

    int p=0;

    for(int i=0;i<idx;i++){

        if(i!=0 && (prod[i].str[0]!=prod[i-1].str[0])) p++;

        for(int j=0;j<TNUM;j++) first\_beta[j]=0;

        find\_first\_beta(prod[i].str,prod[i].len);

        for(int j=0;j<TNUM;j++){

            if(first\_beta[j]&&j!='~') {

                table[p][j]=i+1;

            }

            else if(j=='~' && first\_beta[j]){

                for(int k=0;k<TNUM;k++){

                    if(follow[prod[i].str[0]-'A'][k]) table[p][k]=i+1;

                }

            }

        }

    }

    printf("\n\t LL(1) PARSING TABLE\n");

    printf("%-10s", "");

    for(int i=0;i<TNUM;i++){

        if(terminal[i]) printf("%-10c",i);

    }

    p=0;

    for(int i=0;i<idx;i++){

        if(i==0 || (prod[i].str[0]!=prod[i-1].str[0])){

            printf("\n");

            printf("%-10c",prod[i].str[0]);

            for(int j=0;j<TNUM;j++){

                if(table[p][j]) printf("%-10s",prod[table[p][j]-1].str);

                else if(terminal[j]) printf("%-10s","");

            }

            p++;

        }

    }

    return 0;

}

All the above programs have been compiled, run (on windows) and tested on some sample inputs to verify the correctness.

Programs 3,5 and 7 are subpart of program 8 and hence did not make them because of the time constraint, but those program logic is part of program 8.