Name ..... Branch ..... Sem .....

PROGRAM - 1

Date .....

- Aim: Program to identify whether given String is Keyword or not.

- SOURCE CODE:

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

const char *keywords[NUM_KEYWORDS] = {
    "auto", "break", "case", "char", "const", "continue", "default",
    "do", "double", "else", "enum", "extern", "float", "for",
    "goto", "if", "inline", "int", "long", "register", "restrict",
    "return", "short", "signed", "sizeof", "static", "struct",
    "switch", "typedef", "union", "unsigned", "void", "volatile",
    "while"
};

bool isKeyword(const char *str){
    for(int i=0; i< NUM_KEYWORDS; i++){
        if(strcmp(str, keywords[i]) == 0){
            return true;
        }
    }
    return false;
}

int main(){
    char str[100];
```

Name ..... Branch ..... Sem .....

Date

```
printf("Enter a string: ");
scanf("%s", str);
if (isKeyword(str)){
        printf("'%s' is a keyword.\n", str);
} else {
        printf("'%s' is not a keyword.\n", str);
    }
}
```

- Output :-

Enter a string : int
'int' is a keyword

Enter a string : hello
'hello' is not a keyword.

Name ..... Branch ..... Sem .....

Program : 2 Date

- Aim : Program to identify whether a given line is comment or not.
- Source Program :

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

bool isSingleLineComment (const char* line){
    return (strncmp(line, "//", 2)==0);
}

bool isMultiLineComment(const char* line){
    int len = strlen(line);
    return (len>2 && strncmp(line, "/*", 2)==0
    && line[len-2]=="*" && line[len-1]='/');
}

bool isComment (const char* line){
    return isSingleLineComment(line) || isMultiLineComment(line);
}

int main(){
    char line[200];
    printf("Enter a line of code: ");
    fgets(line, sizeof(line), stdin);
    line[strcspn(line, "\n")] = '\0';
```

Name ..... Branch ..... Sem .....

Date

```
if (isComment(line)){
    printf("The given line is a comment.\n");
} else{
    printf("Given line is NOT a comment.\n");
}
}
```

- Output:

Enter a line of code: // Kunal
The given line is a comment.

Name ..... Branch ..... Sem .....

PROGRAM - 3 Date

- Aim: Program to validate an identifier.
- Source Code:

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#define MAX_LENTH 100

int isValidIdentifier(char str[]){
    int length = strlen(str);
    if(length == 0){
        return 0;
    }
    if(!(isalpha(str[0]) || str[0] == '_')){
        return 0;
    }
    for(int i = 1; i< length; i++){
        if(!(isalpha(str[i]) || str[i] == '_')){
            return 0;
        }
    }
}

char *keywords[] = {"int", "char", "float", "double",
"return", "for", "while", "if", "else", "break", "continue",
"switch", "case", "void"};
int num_keywords = sizeof(keywords)/sizeof(keywords[0]
```

Name ..... Branch ..... Sem .....

```
for (int i = 0; i < num_keywords; i++) {
    if (strcmp(str, keywords[i]) == 0) {
        return 0;
    }
}
return 1;
}

int main() {
    char identifier[MAX_LENGTH];
    printf("Enter an identifier: ");
    scanf("%s", identifier);
    if (isValidIdentifier(identifier)) {
        printf("'%s' is a valid identifier.\n", identifier);
    } else {
        printf("'%s' is not a valid identifier.\n", identifier);
    }
    return 0;
}
```

- Output: Enter an identifier: myVar
  'myVar' is a valid identifier.

Name ..... Branch ..... Sem .....

PROGRAM - 4 Date

- Aim : Program to count total words and blank spaces in a string.
- Source Code :

```
#include <stdio.h>
#include <ctype.h>
int main(){
    char str[100];
    int wordCount = 0, blankSpace = 0;
    int inWord = 0;
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    for(int i=0; str[i] != '\0'; i++){
        if(str[i] == ' '){
            blankSpaces++;
            inWord = 0;
        } else if(isalpha(str[i])){
            if(!inWord){
                wordCount++;
                inWord = 1;
            }
        }
    }
    printf("Total Words: %d\n", wordCount);
    printf("Total Spaces: %d\n", blankSpaces);
}
```

- Output : Enter a string : Hello World
  Total Words = 10
  Total Spaces = 1

Name ..... Branch ..... Sem .....

Program -5 Date

Aim: Program to Count Total no. of operators in an expression.

Source Code:

```
#include <stdio.h>
#include <ctype.h>
int main(){
    char expression[100];
    int operatorCount = 0;
    printf("Enter an expression: ");
    fgets(expression, sizeof(expression), stdin);
    for(int i=0; expression[i] != '\0'; i++){
        if(expression[i]=='+' || expression[i]=='-' ||
        expression[i]=='*' || expression[i]=='/' ||
        expression[i]=='%' || expression[i]=='=' ||
        expression[i]=='<' || expression[i]=='>' ||
        expression[i]=='&' || expression[i]=='|' ||
        expression[i]=='^' || expression[i]=='~'){
            operatorCount++;
        }
    }
    printf("Total operators: %d\n", operatorCount);
}
```

Output:

Enter an expression: a+b+c-d/e
Total operators: 4

Name ..... Branch ..... Sem .....

PROGRAM - 6 Date

- Aim: Program to Count total no. of keywords in a String.

- Source Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX_KEYWORDS 32
const char* keywords[MAX_KEYWORDS]={"auto", "break", "case", "char",
"const", "continue", "default", "do", "double", "else", "enum", "extern", "float", "for",
"goto", "if", "inline", "int", "long", "register", "restrict", "return", "short",
"signed", "sizeof", "static", "struct", "switch", "typedef", "union", "unsigned",
"void", "volatile", "while"};
int iskeyword(const char* word){
    for(int i=0; i<MAX_KEYWORDS; i++){
        if(strcmp(word, keywords[i])==0){
            return 1;
        }}
    return 0;}

int main(){
    char str[100];
    int keywordCount=0;
    char word[50];
    int i,j;
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
```

Name ..... Branch ..... Sem .....

Date

```
i=0;
while (sto[i] != '\0'){
    while(sto[i] != '\0' && !isalpha(sto[i])){
        i++;
    }

    j=0;
while(sto[i] != '\0' && (isalpha(sto[i]) || sto[i]=='_')){
    words[j++] = sto[i++];
}

words[j] = '\0';
if(j>0 && isKeyword(word)){
    keywordCount++;
    }
}
printf("Total no. of keywords: %d\n", keywordCount);
}
```

Output:

Enter a string: "int x = 10; if(x>0) return 1;

Total no. of keywords: 3

Name ..... Branch ..... Sem .....

PROGRAM - 7 Date

Aim: Program to count vowels and consonants in a string.

Source Code:

```
#include <stdio.h>
#include <ctype.h>
int main(){
    char str[100];
    int vowelCount = 0, consonantCount = 0;
    printf("Enter a string: ");
    fgets(str, sizeof(str), stdin);
    for(int i=0; str[i]!='\0'; i++){
        char ch = tolower(str[i]);
        if(ch=='a'||ch=="e"||ch=="i"||ch=='o'||ch=='u'){
            vowelCount++;
        }else{
            consonantCount++;
        }
    }
    printf("Total vowels: %d\n", vowelCount);
    printf("Total consonant: %d\n", consonantCount);
}
```

Output:

Enter a string: Hellaco World
Total vowels: 3
Total consonants: 7

Name ..... Branch ..... Sem .....

Program: 8 Date

- Aim: Program to compute the FIRST of any Grammar.

- Source Code:

```
#include <stdio.h>
#include <string.h>
#define MAX 10
char grammar[MAX][MAX];
char first[MAX][MAX];
int numRules;
int isTerminal(char c){
    return (c>='a' && c<='z');
}
void findfirst(char nonTerminal){
    int i, j, k;
    for(i=0; i<numRules; i++){
        if(grammar[i][0]==nonTerminal){
            for(j=3; grammar[i][j]!='\0'; j++){
                if(isTerminal(grammar[i][j])){
                    if(!strchr(first[nonTerminal-'A'], grammar[i][j]))
                        strncat(first[nonTerminal-'A'], &grammar[i][j], 1);
                    break;
                }else{
                    findfirst(grammar[i][j]);
                    for(k=0; first[grammar[i][j]-'A'][k]!='\0'; k++){
                        if(!strchr(first[nonTerminal-'A'], first[grammar[i][j]-'A'][k]))
                            strncat(first[nonTerminal-'A'], &first[grammar[i][j]-'A'][k], 1);
                    }
                    if(!strchr(first[grammar[i][j]-'A'], '#'))
                        break;
                }
            }
        }
    }
}
```

Name ..... Branch ..... Sem .....

Date

```
            break;
        }
    }
}}}
int main(){
    printf("Enter no. of prod. rules: ");
    scanf("%d", &numRules);
    getchar();
    for(int i=0; i<numRules; i++){
        printf("Enter production %d: ", i+1);
        fgets(grammar[i], sizeof(grammar[i]), stdin);
    }
    for(int i=0; i<26; i++){
        first[i][0] = '\0';
    }
    for(int i=0; i<numRules; i++){
        findfirst(grammar[i][0]);
    }
    printf("\n frost sets:\n");
    for(int i=0; i<numRules; i++){
        printf("FIRST(%c)= {", grammar[i][0]);
        for(int j=0; first[grammar[i][0]-'A'][j]!='\0'; j++)
            printf("%c", first[grammar[i][0]-'A'][j]);
    } printf("}\n"); }}
```

- INPUT :- Enter no. of prod. rules = 2
  Enter prod. 1 : S → A
  Enter prod. 2 : A → a | ε
- Output :- FIRST(S) = {a}
  FIRST(A) = {a ε}

Name ..... Branch ..... Sem .....

PROGRAM - 9 Date

- Aim: Program to find Follow of any Grammar.
- Source Code:

```
#include <stdio.h>
#include <string.h>
#define MAX 10
char grammar[MAX][MAX];
char follow[MAX][MAX];
char first[MAX][MAX];
int numRules;
int isTerminal(char c){
    return (c>='a' && c<='z');
}
void findFirst(char nonTerminal){
    int i, j, k;
    for (i=0; i<numRules; i++){
        if(grammar[i][0]== nonTerminal){
            for (j=3; grammar[i][j]!='\0'; j++){
                if (isTerminal(grammar[i][j])){
                    if(!strchr(first[nonTerminal-'A'], grammar[i][j]))
                strncat(first[nonTerminal-'A'], &grammar[i][j], 1);
                    break;
                }else{
                findFirst(grammar[i][j]);
                    for (k=0; first[grammar[i][j]-'A'][k]!='\0', k++){
                    if(!strchr(first[nonTerminal-'A'], first[grammar[i][j]-'A'][k]))
                        strncat(first[nonTerminal-'A'], &first[grammar[i][j]-'A'][k], 1);
                    }
                }
            }
        }
    }
```

Name ..... Branch ..... Sem .....

Date

```
            break; }}}}}
void findfollow(char nonTerminal){
    int i,j,k;
    if(nonTerminal == grammar[0][0]){
      strncat(follow[nonTerminal-'A'],"$",1);
    }
    for(i=0; i<numRules; i++){
      for(j=3; grammar[i][j]!='\0'; j++){
        if(grammar[i][j]==nonTerminal){
          if(grammar[i][j+1]!='\0'){
            if(isTerminal(grammar[i][j+1])){
            if(!strchr(follow[nonTerminal-'A'],grammar[i][j+1])){
            strncat(follow[nonTerminal-'A'],&grammar[i][j+1],1);
          }else{
        for(k=0; first[grammar[i][j+1]-'A'][k]!='\0'; k++){
        if(!strchr(follow[nonTerminal-'A'],first[grammar[i][j+1]
    -'A'][k])){
        strncat(follow[nonTerminal-'A'],&first[grammar[i][j+1]-'A']
    [k],1); }}}}
    if(grammar[i][j+1]=='\0' || strchr(first[grammar[i][j+1]
    -'A'],'ε')){
        if(!strchr(follow[nonTerminal-'A'],grammar[i][0])){
          strncat(follow[nonTerminal-'A'],&grammar[i][0],1);
    }}}}}}
int main(){
    printf("Enter no. of prod. rules: ");
    scanf("%d",&numRules);
    getchar();
```

Name ..... Branch ..... Sem .....

Date

```
for (int i=0 ; i< numRules; i++ ){
    printf ("Enter prod. %d ", i+1);
    fgets (grammar[i], sizeof(grammar[i]), stdin);
}

for (int i=0 ; i< 26 ; i++){
    first[i][0] = '\0';
    follow[i][0] = '\0';
}

for (int i=0 ; i< numRules ; i++){
    findfirst (grammar[i][0]);
}

for (int i=0 ; i< numRules ; i++){
    findfollow (grammar[i][0]);
}

for (int i=0 ; i< numRules ; i++){
    printf ("Follow(%c) = { ", grammar[i][0]);
    for (int j =0 ; follow[grammar[i][0]-'A'][j] != '\0' ; j++){
        printf ("%c", follow[grammar[i][0]-'A'][j]);
    }
    printf("}\n"); }}
```

- Input:

→ Enter no. of prod. rules = 3
Enter prod. 1 = S → AB
Enter prod. 2 = A → a|ε
Enter prod. 3 = B → b

- Output:

→ Follow(S) = {$}
→ Follow(A) = {b $}
→ Follow(B) = {a}

Name ..... Branch ..... Sem .....

PROGRAM - 10 Date

Aim: Program to perform Shift Reduce parsing for any grammar.

Source Code:

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define MAX 100
char stack[MAX];
char gram[2][3][3] = {{'S', 'a', 'S'}, {'S', 'e', '\0'}};
void push(char c){
    stack[++top] = c;
}
char pop(){
    return stack[top--];
}
int isTerminal(char c){
    return (c == 'a' || c == 'b');
}
void shiftReduceParse(char *input){
    int i=0;
    int n = strlen(input);
    printf("Input: %s\n", input);
    printf("Performing Shift-Reduce parsing...\n");
    while(i<n || top >= 0){
        printf("Stack: ");
        for(int j=0; j<=top; j++){
            printf("%c", stack[j]);
        }
```

```
printf(", Input: ");
for(int j=i; j<n; j++){
    printf("%c", input[j]);
}
printf("\n");
if(i<n){
    push(input[i]);
    i++;
}
int reduced = 0;
for(int rule = 0; rule < 2; rule++){
  int len = strlen(grammar[rule][1]);
  if(top >= len - 1){
    int match = 1;
    for(int k = 0; k < len; k++){
      if(stack[top - k] != grammar[rule][1][len-k-1]){
        match = 0;
        break; }}
if(match){
  printf("Reduce using: %c -> %s\n", grammar[rule][0],
    grammar[rule][1]);
  for(int k=0; k < len; k++){
        pop();
}
  push(grammar[rule][0]);
  reduced = 1;
  break;
}
```

Name ..... Branch ..... Sem .....

Date

```
    if (reduced == 0 && i >= n){
        break; }}
    if (top == 0 && stack[0] == 'S'){
        printf ("Parsing successful! \n");
    } else{
        printf ("Parsing failed! \n");
    }
int main (){
    char input[MAX];
    printf ("Enter the input string (only a's and b's): ");
    fgets (input, sizeof(input), stdin);
    input [strcspn (input, "\n")] = '\0';
    shiftReduceParse (input);
}
```

- Input :

→ Enter the input string : ab

- Output :

→ Input : ab
Performing Shift-Reduce parsing ...
Stack : , Input : ab
Stack : a, Input : b
Reduce using : S → aSb
Stack : S , Input :
Parsing successful!