

Experiment No. 3

Aim: Write a program to implement the FIRST and Follow set for the given grammar.

Theory:

In compiler design, "FIRST" and "FOLLOW" are sets used in the context of parsing techniques, particularly for constructing predictive parser.

• FIRST set

⇒ The First set of a non-terminal in a grammar consists of the terminal symbols that can begin the strings derivable from that non-terminal.

Algorithm:

- 1) IF X is a terminal, then $\text{FIRST}(X) = \{X\}$.
- 2) IF X is a non-terminal and $X \rightarrow \epsilon$ is a production, then add ϵ to $\text{FIRST}(X)$.
- 3) IF X is a non-terminal and $X \rightarrow Y_1 Y_2 \dots Y_k$ is a production, add $\text{FIRST}(Y_1, Y_2, \dots, Y_k) - \{\epsilon\}$ to $\text{FIRST}(X)$.
- 4) Repeat steps 2-3 until no more additions can be made.

For e.g.,

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid id$$

⇒

	FIRST	FOLLOW
$E \rightarrow E + T \mid T$ $T \rightarrow T * F \mid F$ $F \rightarrow (E) \mid id$	{ (, id }	{ +, *,), \$ }

We have left recursion in above grammar.
 ⇒ After removing left recursion,

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \epsilon$$

$$T \rightarrow F T' \mid \epsilon$$

$$T' \rightarrow * F T' \mid \epsilon$$

$$F \rightarrow (E) \mid id$$

	FIRST	FOLLOW
E	{ (, id }	{ \$,) }
E'	{ +, ε }	{ \$,) }
T	{ (, id }	{ +, \$,) }
T'	{ *, ε }	{ +, \$,) }
F	{ (, id }	{ *, +, \$,) }

• Follow set

⇒ The FOLLOW set of a non-terminal in a grammar consists of the terminal symbols that can appear immediately to the right of that non-terminal in some sentential form.

• Algorithm:

- 1) Place \$ in FOLLOW(S), where S is the start symbol.
- 2) IF there is a production $A \rightarrow \alpha B \beta$, then everything in FIRST(β) except for ϵ is in FOLLOW(B).
- 3) IF there is production $A \rightarrow \alpha B \beta$ or $A \rightarrow \alpha B$ where FIRST(β) contains ϵ , then everything in FOLLOW(A) is in FOLLOW(B).
- 4) Repeat steps 2-3 until no more additions can be made.

• Conclusion:

In this experiment, we learnt about FIRST and FOLLOW of grammar, how to calculate it.

QA
29/02/2024

Code:

```
#include <iostream>
#include <string.h>
#define max 20

using namespace std;

char prod[max][10];
char ter[10], nt[10];
char first[10][10], follow[10][10];
int eps[10];
int count_var = 0;

int findpos(char ch) {
    int n;
    for (n = 0; nt[n] != '\0'; n++)
        if (nt[n] == ch)
            break;
    if (nt[n] == '\0')
        return 1;
    return n;
}

int IsCap(char c) {
    if (c >= 'A' && c <= 'Z')
        return 1;
    return 0;
}

void add(char *arr, char c) {
    int i, flag = 0;
    for (i = 0; arr[i] != '\0'; i++) {
        if (arr[i] == c) {
            flag = 1;
            break;
        }
    }
    if (flag != 1)
        arr[strlen(arr)] = c;
}

void addarr(char *s1, char *s2) {
```

```

int i, j, flag = 99;
for (i = 0; s2[i] != '\0'; i++) {
    flag = 0;
    for (j = 0;; j++) {
        if (s2[i] == s1[j]) {
            flag = 1;
            break;
        }
        if (j == strlen(s1) && flag != 1) {
            s1[strlen(s1)] = s2[i];
            break;
        }
    }
}

void addprod(char *s) {
    int i;
    prod[count_var][0] = s[0];
    for (i = 3; s[i] != '\0'; i++) {
        if (!IsCap(s[i]))
            add(ter, s[i]);
        prod[count_var][i - 2] = s[i];
    }
    prod[count_var][i - 2] = '\0';
    add(nt, s[0]);
    count_var++;
}

void findfirst() {
    int i, j, n, k, e, n1;
    for (i = 0; i < count_var; i++) {
        for (j = 0; j < count_var; j++) {
            n = findpos(prod[j][0]);
            if (prod[j][1] == (char)238)
                eps[n] = 1;
            else {
                for (k = 1, e = 1; prod[j][k] != '\0' && e == 1; k++) {
                    if (!IsCap(prod[j][k])) {
                        e = 0;
                        add(first[n], prod[j][k]);
                    } else {
                        n1 = findpos(prod[j][k]);

```

```

        addarr(first[n], first[n1]);
        if (eps[n1] == 0)
            e = 0;
    }
}
if (e == 1)
    eps[n] = 1;
}
}
}

void findfollow() {
    int i, j, k, n, e, n1;
    n = findpos(prod[0][0]);
    add(follow[n], '#');
    for (i = 0; i < count_var; i++) {
        for (j = 0; j < count_var; j++) {
            k = strlen(prod[j]) - 1;
            for (; k > 0; k--) {
                if (IsCap(prod[j][k])) {
                    n = findpos(prod[j][k]);
                    if (prod[j][k + 1] == '\\0') {
                        n1 = findpos(prod[j][0]);
                        addarr(follow[n], follow[n1]);
                    }
                    if (IsCap(prod[j][k + 1])) {
                        n1 = findpos(prod[j][k + 1]);
                        addarr(follow[n], first[n1]);
                        if (eps[n1] == 1) {
                            n1 = findpos(prod[j][0]);
                            addarr(follow[n], follow[n1]);
                        }
                    } else if (prod[j][k + 1] != '\\0')
                        add(follow[n], prod[j][k + 1]);
                }
            }
        }
    }
}

int main() {
    char s[max], i;

```

```

cout << "Enter the productions\n";
cin >> s;
while (strcmp("end", s)) {
    addprod(s);
    cin >> s;
}
findfirst();
findfollow();

// Displaying the title labels
cout << "Production\tFirst\tFollow\n";

for (i = 0; i < strlen(nt); i++) {
    cout << nt[i] << "\t\t\t" << first[i];
    if (eps[i] == 1)
        cout << ((char)238) << "\t\t";
    else
        cout << "\t\t";
    cout << follow[i] << "\n";
}
return 0;
}

```

Output:

```

/tmp/M08wSvh02h.o
Enter the productions
E->TB
B->+TB
T->FC
C->*FC
F->(E)
F->i
B->
C->
end
Production  First  Follow
E           (i    #)
B           +⌢  #)
T           (i    +#)
C           *⌢  +#)
F           (i    *+#)

```