ABHINAV RANJAN
RA1911003010003
CSE A1 SECTION
SRMIST , KTR

COMPILER DESIGN LAB

EXP 8 - LEADING AND TRAILING

AIM :

To create a program to find the leading and trailing terminals of a given grammar.

REQUIREMENTS :

1. Knowledge of the concepts of grammar and its types
2. Knowledge of the concepts of terminals and non terminals
3. Knowledge of the concepts of leading and trailing terminals
4. Online compiler to execute code

THEORY:

Leading and Trailing are functions specific to generating an operator-precedence parser, which is only applicable if you have an operator precedence grammar. An operator precedence grammar is a special case of an

operator grammar, and an operator grammar has the important property that no production has two consecutive non-terminals.

Given an operator grammar, the function Leading (resp. Trailing) of a non-terminal produces the set of terminals which could be (recursively) the first (resp. last) terminal in some sentential form derived from that non-terminal.

Another possibly more intuitive definition is that a terminal is in the Leading set for a non-terminal if the terminal is "visible" from the beginning of a production. Non-terminals are "transparent", so a terminal could be visible either by looking through a leading non-terminal or by looking into a visible non-terminal.

ALGORITHM:

LEADING ALGORITHM:

1. Start
2. For each nonterminal A and terminal a do L(A,a):= fal se;
3. For each production of the form A->a or A->B do INSTALL(A,a);
4. While STACK not empty repeat step 5& 6
5. Pop top pair (B,a) from STACK;
6. For each production of the form A->B do INSTALL(A,a)

7. Stop

Algorithm For INSTALL(A,a)

1. Start
2. If L(A,a) not present do step 3 and 4. 3 . Make L(A,a)=True

4 . Push (A,a) onto stack 5 . Stop

TRAILING ALGORITHM :

1. Start
2. For each non terminal A and terminal a do L(A,a):=f alse;
3. For each production of the form A->a(alpha) or A- > Ba(alpha) do INSTALL(A,a)
4. While STACK not empty repeat 5 and 6
5. Pop top pair from stack
6. For each production of the form A- > B(alpha) do INSTALL(A,a)
7. Stop

Algorithm For INSTALL(A,a)

1. Start
2. If L[A,a] not present repeat step 3 and 4
3. Make L(A,a)=True
4. Push (A,a) onto stack
5. Stop

SOURCE CODE :

1. LEADING

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

char arr[18][3] ={{'E', '+', 'F'},{'E', '*', 'F'},{'E', '(', 'F'}, {'E', ')', 'F'},{'E',
'i', 'F'},{'E', '$', 'F'},
{'F', '+', 'F'},{'F', '*', 'F'},{'F', '(', 'F'},{'F', ')', 'F'},{'F', 'i', 'F'},{'F', '$', 'F'},
{'T', '+', 'F'},
{'T', '*', 'F'}, {'T', '(', 'F'},{'T', ')', 'F'},{'T', 'i', 'F'},{'T', '$', 'F'}};

char prod[6] = "EETTFF";
char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0'}, {'T', '*', 'F'},  {'F', '\0'}, {'(', 'E',
')'}, {'i', '\0'}};
char stack [5][2];
int top = -1;

void install(char pro, char re) {
    int i;
    for (i = 0; i < 18; ++i) {
        if (arr[i][0] == pro && arr[i][1] == re) {

            arr[i][2] = 'T';
            break;
        }
    }
    ++top;
    stack[top][0] = pro;
    stack[top][1] = re;
}

void main() {
    int i = 0, j;
    char pro, re, pri = ' ';
    for (i = 0; i < 6; ++i) {
```

```
        for (j = 0; j < 3 && res[i][j] != '\0'; ++j) {
            if (res[i][j] == '+' || res[i][j] == '*' || res[i][j] == '(' || res[i][j] == ')'
|| res[i][j] == 'i' || res[i][j] == '$') {
                install(prod[i], res[i][j]);
                break;
            }
        }
    }
    while (top >= 0) {
        pro = stack[top][0];
        re = stack[top][1];
        --top;
        for (i = 0; i < 6; ++i) {
            if (res[i][0] == pro && res[i][0] != prod[i]) {
                install(prod[i], re);
            }
        }
    }
    for (i = 0; i < 18; ++i) {
        printf("\n\t");
        for (j = 0; j < 3; ++j)
            printf("%c\t", arr[i][j]);
    }
    getch();
    printf("\n\n");
    for (i = 0; i < 18; ++i) {
        if (pri != arr[i][0]) {
            pri = arr[i][0];
            printf("\n\t%c -> ", pri);
        }
        if (arr[i][2] == 'T')
            printf("%c ", arr[i][1]);
    }
    getch();
}
```

2. TRAILING

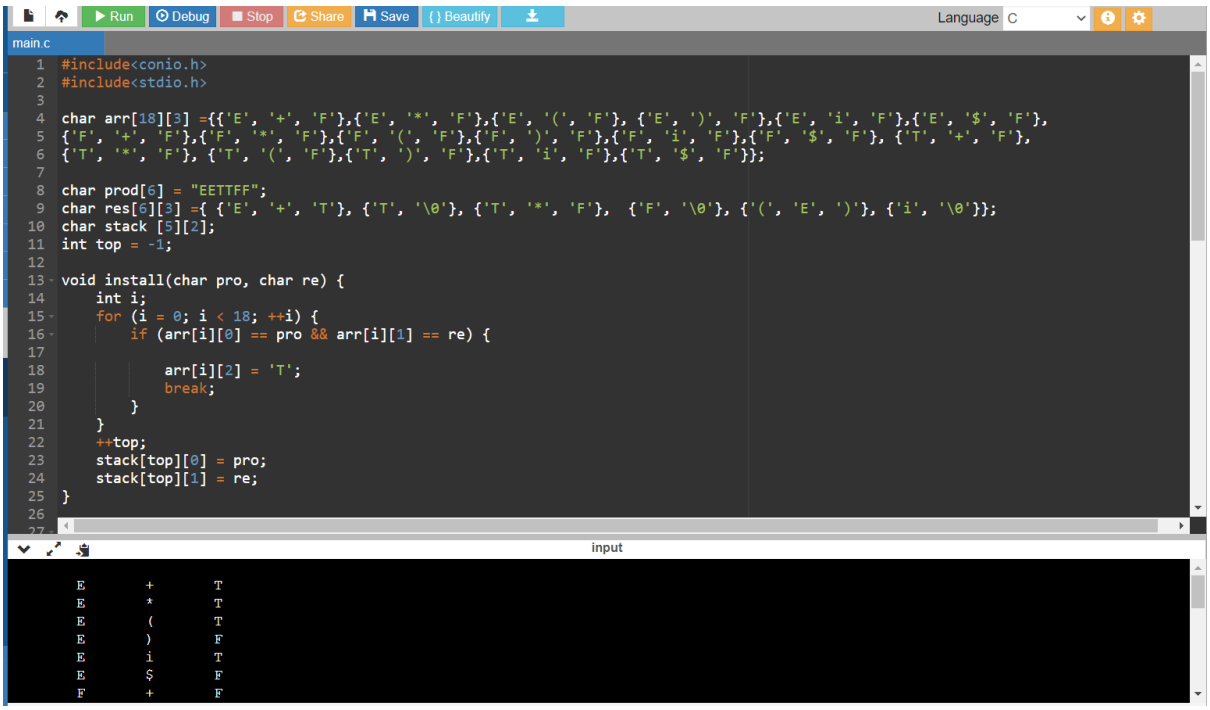
```
#include<conio.h>
#include<stdio.h>
char arr[18][3] ={{'E', '+', 'F'},  {'E', '*', 'F'}, {'E', '(', 'F'},    {'E', ')', 'F'},
{'E', 'i', 'F'},
   {'E', '$', 'F'},    {'F', '+', 'F'},    {'F', '*', 'F'},    {'F', '(', 'F'},    {'F', ')',
'F'},    {'F', 'i', 'F'},
   {'F', '$', 'F'},    {'T', '+', 'F'},    {'T', '*', 'F'},    {'T', '(', 'F'},    {'T', ')',
'F'},    {'T', 'i', 'F'},
   {'T', '$', 'F'},
};
char prod[6] = "EETTFF";
char res[6][3] ={    {'E', '+', 'T'},    {'T', '\0', '\0'},    {'T', '*', 'F'},    {'F',
'\0', '\0'},    {'(', 'E', ')'},   {'i', '\0', '\0'},};
char stack [5][2];
int top = -1;

void install(char pro, char re) {
   int i;
   for (i = 0; i < 18; ++i) {
      if (arr[i][0] == pro && arr[i][1] == re) {
     ++top;
   arr[i][2] = 'T';
   break;
      }

   }

   stack[top][0] = pro;
   stack[top][1] = re;
}

void main() {
   int i = 0, j;
   char pro, re, pri = ' ';
   for (i = 0; i < 6; ++i) {
```

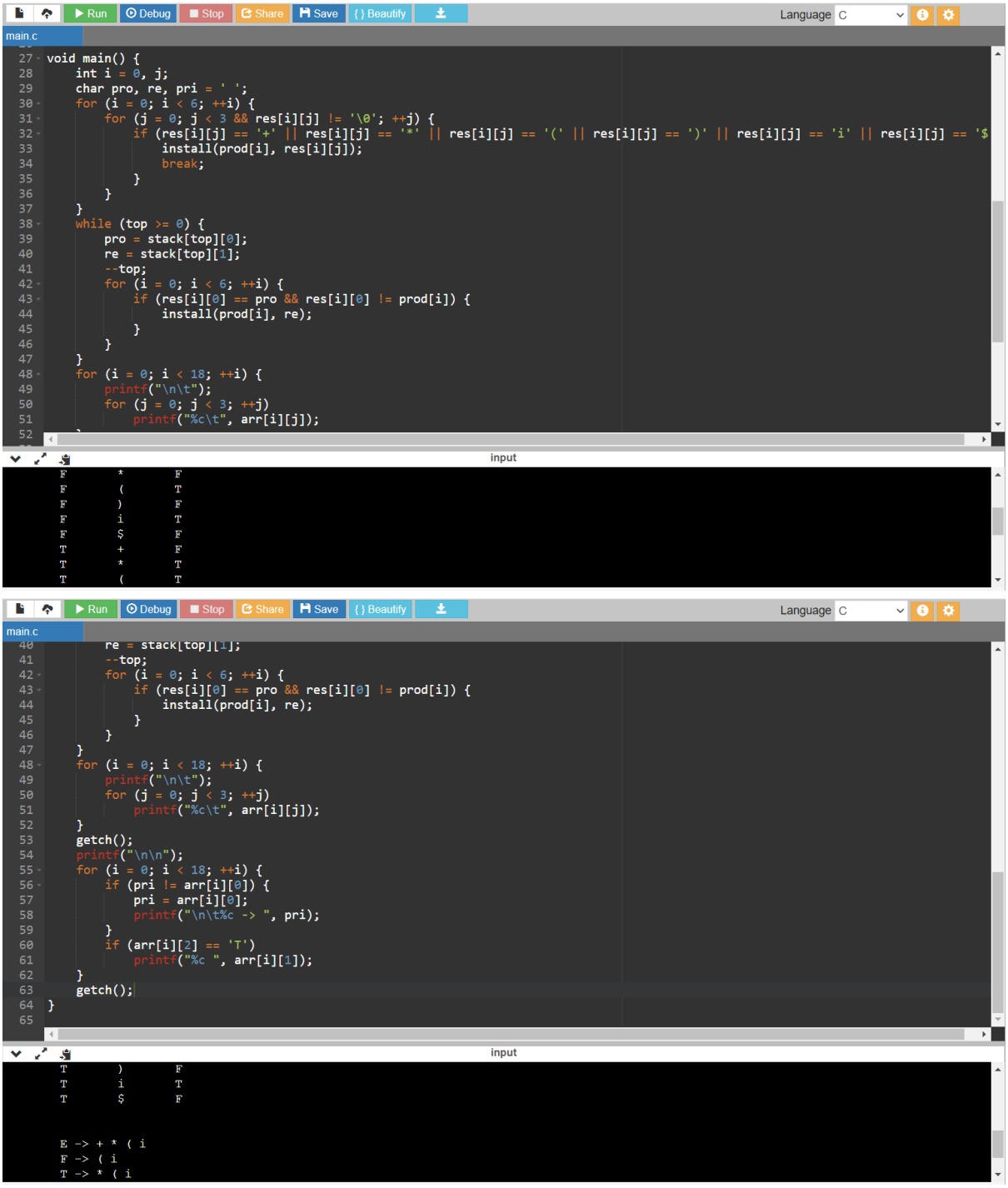
```
        for (j = 2; j >= 0; --j) {

            if (res[i][j] == '+' || res[i][j] == '*' || res[i][j] == '(' || res[i][j] == ')'
|| res[i][j] == 'i' || res[i][j] == '$') {
                install(prod[i], res[i][j]);
                break;
            } else if (res[i][j] == 'E' || res[i][j] == 'F' || res[i][j] == 'T') {
                if (res[i][j - 1] == '+' || res[i][j - 1] == '*' || res[i][j - 1] == '(' ||
res[i][j -
                        1] == ')' || res[i][j - 1] == 'i' || res[i][j - 1] == '$') {
                    install(prod[i], res[i][j - 1]);
                    break;
                }
            }
        }
    }

    while (top >= 0) {
        pro = stack[top][0];
        re = stack[top][1];
        --top;
        for (i = 0; i < 6; ++i) {
            for (j = 2; j >= 0; --j) {
                if (res[i][0] == pro && res[i][0] != prod[i]) {
                    install(prod[i], re);
                    break;
                } else if (res[i][0] != '\0') break;
            }
        }
    }
    for (i = 0; i < 18; ++i) {
        printf("\n\t");
        for (j = 0; j < 3; ++j)
            printf("%c\t", arr[i][j]);
    }
    getch();
    printf("\n\n");
```

```
for (i = 0; i < 18; ++i) {
    if (pri != arr[i][0]) {
        pri = arr[i][0];
        printf("\n\t%c -> ", pri);
    }
    if (arr[i][2] == 'T')
        printf("%c ", arr[i][1]);}
getch();}
```

SCREENSHOT OF OUTPUT:

1. LEADING



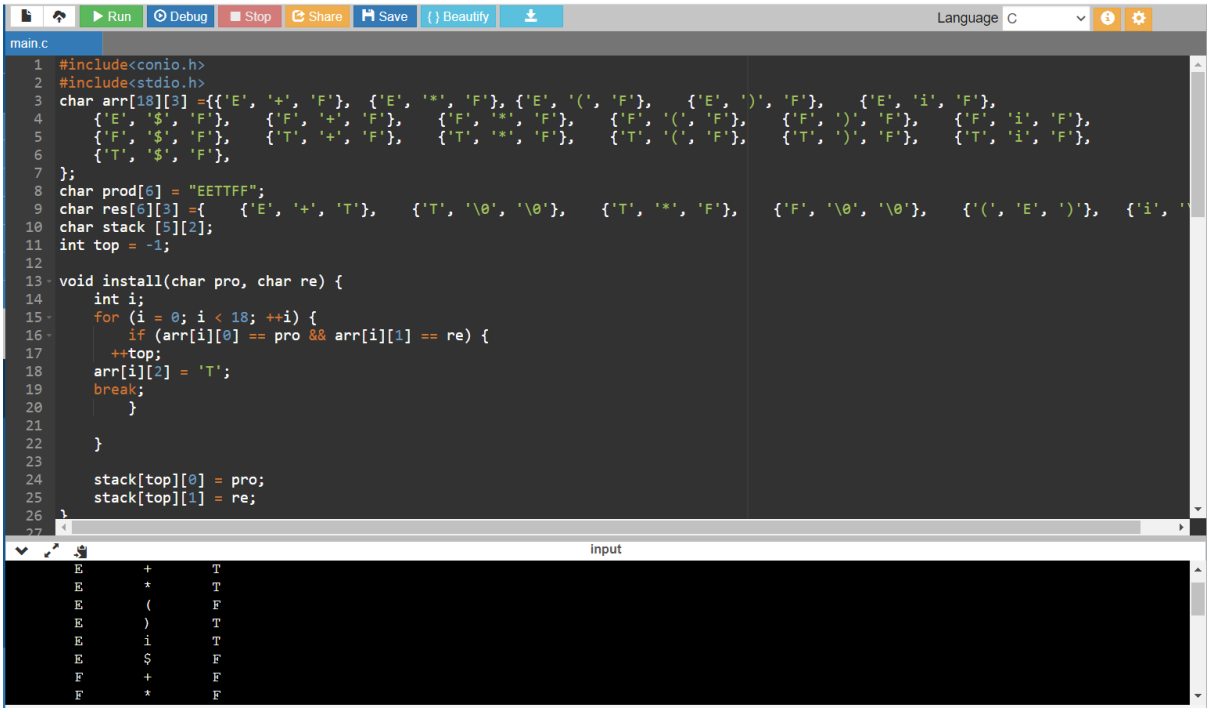


```
Run  Debug  Stop  Share  Save  { } Beautify  Language C
main.c
void main() {
    int i = 0, j;
    char pro, re, pri = ' ';
    for (i = 0; i < 6; ++i) {
        for (j = 0; j < 3 && res[i][j] != '\0'; ++j) {
            if (res[i][j] == '+' || res[i][j] == '*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' || res[i][j] == '$
                install(prod[i], res[i][j]);
                break;
            }
        }
    }
    while (top >= 0) {
        pro = stack[top][0];
        re = stack[top][1];
        --top;
        for (i = 0; i < 6; ++i) {
            if (res[i][0] == pro && res[i][0] != prod[i]) {
                install(prod[i], re);
            }
        }
    }
    for (i = 0; i < 18; ++i) {
        printf("\n\t");
        for (j = 0; j < 3; ++j)
            printf("%c\t", arr[i][j]);
input
F       *       F
F       (       T
F       )       F
F       i       T
F       $       F
T       +       F
T       *       T
T       (       T
```



```
Run  Debug  Stop  Share  Save  { } Beautify  Language C
main.c
        re = stack[top][1];
        --top;
        for (i = 0; i < 6; ++i) {
            if (res[i][0] == pro && res[i][0] != prod[i]) {
                install(prod[i], re);
            }
        }
    }
    for (i = 0; i < 18; ++i) {
        printf("\n\t");
        for (j = 0; j < 3; ++j)
            printf("%c\t", arr[i][j]);
    }
    getch();
    printf("\n\n");
    for (i = 0; i < 18; ++i) {
        if (pri != arr[i][0]) {
            pri = arr[i][0];
            printf("\n\t%c -> ", pri);
        }
        if (arr[i][2] == 'T')
            printf("%c ", arr[i][1]);
    }
    getch();
}
input
T       )       F
T       i       T
T       $       F


E -> + * ( i
F -> ( i
T -> * ( i
```

2. TRAILING


Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
#include<conio.h>
#include<stdio.h>
char arr[18][3] ={{'E', '+', 'F'}, {'E', '*', 'F'}, {'E', '(', 'F'}, {'E', ')', 'F'}, {'E', 'i', 'F'},
{'E', '$', 'F'}, {'F', '+', 'F'}, {'F', '*', 'F'}, {'F', '(', 'F'}, {'F', ')', 'F'}, {'F', 'i', 'F'},
{'F', '$', 'F'}, {'T', '+', 'F'}, {'T', '*', 'F'}, {'T', '(', 'F'}, {'T', ')', 'F'}, {'T', 'i', 'F'},
{'T', '$', 'F'},
};
char prod[6] = "EETTFF";
char res[6][3] ={ {'E', '+', 'T'}, {'T', '\0', '\0'}, {'T', '*', 'F'}, {'F', '\0', '\0'}, {'(', 'E', ')'}, {'i', '
char stack [5][2];
int top = -1;
void install(char pro, char re) {
int i;
for (i = 0; i < 18; ++i) {
if (arr[i][0] == pro && arr[i][1] == re) {
++top;
arr[i][2] = 'T';
break;
}
}
stack[top][0] = pro;
stack[top][1] = re;
}
input
E + T
E * T
E (F
E) T
E i T
E $ F
F + F
F * F


Run
Debug
Stop
Share
Save
{ } Beautify
Language C
main.c
void main() {
int i = 0, j;
char pro, re, pri = ' ';
for (i = 0; i < 6; ++i) {
for (j = 2; j >= 0; --j) {
if (res[i][j] == '+' || res[i][j] == '*' || res[i][j] == '(' || res[i][j] == ')' || res[i][j] == 'i' || res[i][j] == '$
install(prod[i], res[i][j]);
break;
} else if (res[i][j] == 'E' || res[i][j] == 'F' || res[i][j] == 'T') {
if (res[i][j - 1] == '+' || res[i][j - 1] == '*' || res[i][j - 1] == '(' || res[i][j -
1] == ')' || res[i][j - 1] == 'i' || res[i][j - 1] == '$') {
install(prod[i], res[i][j - 1]);
break;
}
}
}
}
while (top >= 0) {
pro = stack[top][0];
re = stack[top][1];
--top;
for (i = 0; i < 6; ++i) {
for (j = 2; j >= 0; --j) {
if (res[i][0] == pro && res[i][0] != prod[i]) {
input
F (F
F) T
F i T
F $ F
T + F
T * T
T (F
T) T

OBSERVATION :

The leading and trailing of all non terminals were found and the output displayed in the console was verified.

10-3-22

PRECEDENCE RELATION TABLE

	+	*	id	(	)	$
+	>	<	<	<	>	>
*	>	>	<	<	>	>
id	>	>	e	e	>	>
(	<	<	<	<	=	e
)	>	>	e	e	>	>
$	<	<	<	<	e	Accept

Leading (E) → { +, *, (, id }

Leading (F) → { (, id }

Leading (T) → { *, (, id }

Trailing (E) → { +, *,), id }

Trailing (F) → {), id }

Trailing (T) → { *,), id }

RESULT :

Thus we have successfully implemented a program which can identify the leading and trailing terminals in a given grammar.