**Experiment No - 07**

**Aim**: **Implement a program for constructing Operator Precedence Parsing.**

**Date:**

**Competency and Practical Skills:**

* Understanding of OPG and its role in compiler construction • Ability to write precedence table for a given language
* Ability to develop OPG program using C compiler.

**Relevant CO:** CO2

**Objectives:**

By the end of this experiment, the students should be able to:

* + Understand the concept of OPG its significance in compiler construction
  + Write precedence relations for grammar
  + Implement a OPG using C compiler

**Software/Equipment:**  C compiler

**Theory:**

Operator Precedence Parsing is also a type of Bottom-Up Parsing that can be used to a class of Grammars known as Operator Grammar.

A Grammar G is Operator Grammar if it has the following properties − • Production should not contain ϵ on its right side.

• There should not be two adjacent non-terminals at the right side of production.

**Example 1** − Verify whether the following Grammar is operator Grammar or not.

E → E A E |(E)|id A → +| − |  **Solution**

No, it is not an operator Grammar as it does not satisfy property 2 of operator Grammar.

As it contains two adjacent Non-terminals on R.H.S of production E → E A E.

We can convert it into the operator Grammar by substituting the value of A in E → E A E.

E → E + E |E − E |E \* E |(E) | id. Operator Precedence Relations

Three precedence relations exist between the pair of terminals.

|  |  |
| --- | --- |
| Relation | Meaning |
| p <. q | p has less precedence than q. |
| p >. q | p has more precedence than q. |
| p =. q | p has equal precedence than q. |

Depending upon these precedence Relations, we can decide which operations will be executed or parsed first.

Association and Precedence Rules

* If operators have different precedence Since \* has higher precedence than + Example−

In a statement a + b \* c

 + <. \*

In statement a \* b + c

∴ ∗ . > +

* If operators have Equal precedence, then use Association rules.

(a) Example minus; In statement a + b + c here + operators are having equal precedence.

As '+' is left Associative in a + b + c

 (a + b) will be computed first, and then it will be added to c.

i.e., (a + b) + c

+ .> +

Similarly, '\*' is left Associative in a \* b \* c

(b) Example − In a statement a ↑ b ↑ c here, ↑ is the Right Associative operator  It will become a ↑ (b ↑ c)  (b ↑ c) will be computed first.

 ↑<. ↑

* Identifier has more precedence then all operators and symbols.

 θ <. id $ <. id

id . > θ id . > $

id . >) (<. id.

* $ has less precedence than all other operators and symbols.

$ <. ( id . > $

$ <. + ). > $

$ <.\*

**Example 2** – Construct the Precedence Relation table for the Grammar.

E → E + E | E  E/id

**Solution**

Operator-Precedence Relations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Id | + | \* | $ |
| Id |  | .> | .> | .> |
| + | <. | .> | <. | .> |
| \* | <. | .> | .> | .> |
| $ | <. | <. | <. |  |

Advantages of Operator Precedence Parsing

* It is accessible to execute.

Disadvantages of Operator Precedence Parsing

* Operator Like minus can be unary or binary. So, this operator can have different precedence’s in different statements.
* Operator Precedence Parsing applies to only a small class of Grammars.

**Program:**

#include<stdlib.h>

#include<stdio.h>

#include<string.h>

// function f to exit from the loop // if given condition is not true void f()

{

printf("Not operator grammar"); exit(0);

}

void main()

{

char grm[20][20], c;

// Here using flag variable,

// considering grammar is not operator grammar

int i, n, j = 2, flag = 0;

// taking number of productions from user

scanf("%d", &n);

for (i = 0; i < n; i++)

scanf("%s", grm[i]);

for (i = 0; i < n; i++) {

c = grm[i][2];

while (c != '&#092;&#048;') {

if (grm[i][3] == '+' || grm[i][3] == '-' || grm[i][3] == '\*' || grm[i][3] == '/')

flag = 1;

else {

flag = 0;

f();

}

if (c == '$') {

flag = 0;

f();

}

c = grm[i][++j];

}

}

if (flag == 1)

printf("Operator grammar");

}

**Observations and Conclusion:**

Input :3

A=A\*A

B=AA

A=$

Output : Not operator grammar

In the above example ,the grammar is analysed as per operator grammar rules and the output is against the rules of OPG so, it is not an operator grammar.

Input :2

A=A/A

B=A+A

Output : Operator grammar

In the above example ,the grammar is analysed as per operator grammar rules and the output favors the rules of OPG(operator present between two non terminals) so, it is not an operator grammar.

**Quiz:**

1. Define operator grammar.
2. Define operator precedence grammar.
3. What are the different precedence relations in operator precedence parser?
4. What are the different methods are available to determine relations?
5. What do you mean by precedence function

**Suggested Reference:**

1. <https://www.gatevidyalay.com/operator-precedence-parsing/>
2. <https://www.geeksforgeeks.org/role-of-operator-precedence-parser/>

**Rubric wise marks obtained:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rubrics** | **Knowledge of parsing**  **techniques**  **(2)** | | **Knowledge**  **of**  **precedence table (2)** | | **Implementat ion (2)** | | **Completeness and accuracy**  **(2)** | | **Presentation**  **(2)** | | **Total** |
| **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** | **Good**  **(2)** | **Avg.**  **(1)** |
| **Marks** |  |  |  |  |  |  |  |  |  |  |  |