REPORT

REVERSE POLISH CALCULATOR

DATA STRUCTURE &ALGORITHMS

CTEC22033

CT/2016/012 Dilujan P.

CT/2016/013 Dissanayake J.B.A.C.

CT/2016/015 Fernando K.D.N.

CT/2016/028 Jayawardena H.M.D.P.

CT/2016/048 Pirashanthy B.

CT/2016/070 Vinojith N.

**INTRODUCTION**

Reverse Polish Notation is a way of expressing arithmetic expressions that avoids the use of brackets to define priorities for evaluation of operators. In the Reverse Polish calculator when we are using any expression, no matter how complicated as it can be specified without the use of parentheses. There are two best known alternatives which we can write the operator before or after its operands known as prefix or postfix notation. The only real difference between the two notations is the direction that you read them, left to right or right to left. In our Reverse Polish Calculator, we used postfix method. In postfix method operators are written in after the operands.

Most calculators are designed to recognize basic operations the same way they are written. Reverse Polish Calculator is a faster method of inputting equations. It can also help with performing complex operations on a calculator. In here we have implemented this Reverse Polish Calculator using Stack data structure. When the expression will be executed then the operands are pushed onto a stack. When an operation is performed, it pops its operands from the stack and pushes its result back onto the stack.

This may be achieved by parsing the bracketed expression before carrying out the calculation. But it is more likely that the calculator logic will be pushing numbers down onto the stack every time a pair of brackets is opened or is implied by the operator precedence. So in RPN calculator is offloading this work to the user, resulting in simpler logic design in the calculator. The technical barriers to using conventional bracket notation in an electronic calculator no longer exist, and yet users of RPN calculators rarely seem to want to move over to the more conventional algebraic logic.

**METHOD**

Let’s consider about the approach which we have implemented this Reverse Polish Calculator. “?” denote an instruction to read an operand and push it onto the stack. As the operators which are *+* , *−*, \* , and / represent arithmetic operations. Within these operators “=” is an instruction to print the top of the stack. Further we have used a, b, c, andd to denote numerical values such as 2.5 , 6 or -4.

As the format for this RPC we used **?a?b+=** instead of **a+b =**

Let’s consider the procedure through an example from one of the instruction types.

**(a+b) \* (c+d)**

According to the above expression by using the Postfix notation, initially we have to insert operands respectively which are a,b,c and d.

In this notation the above expression would be ;

**?a?b+?c?d+\*=**

Reading from left to right, this is interpreted as follows:

* First we have to push **a** onto the stack.
* Then we push **b** onto the stack. Reading from the top, the stack now contains (b,a).
* By applying the **+** operation. Take the top two operands off the stack, add them together, and put the result back on the stack. The stack now contains just the result of the **a** and **b**.
* Then push **c** onto the stack.
* After that push **d** onto the stack. It now contains (d, c, result of the first part).
* Then by applying the **+** operation. Take the top two operands off the stack, add those two operands together. And put the result back on the stack. The stack now contains (result of the c and d, result of the a and b).
* Then we have to apply the **\*** operation. Take the top two operands off the stack, multiply them together, by applying the **=** operator and put the result back on the stack. The stack now contains just the final result.

A close up of a map

Description automatically generated**ALGORITHM FOR THE SCENARIO**

**Pseudocode**

Begin

input expression

while(next operator or operand is not end of file indicator)

if (operand)

push in stack

else if (operator)

pop two operands

do operation

push into stack

else

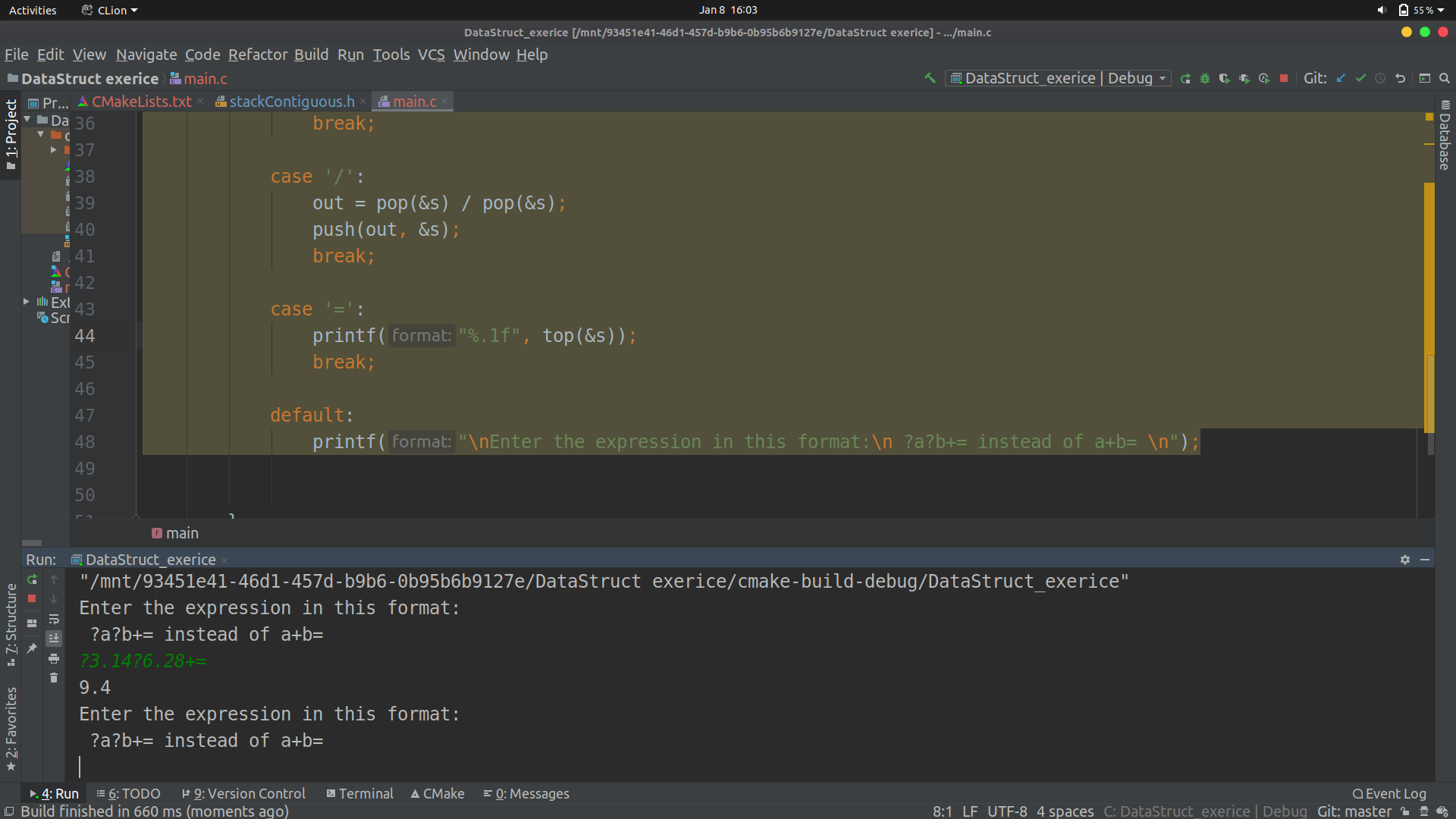
pop

print top of stack

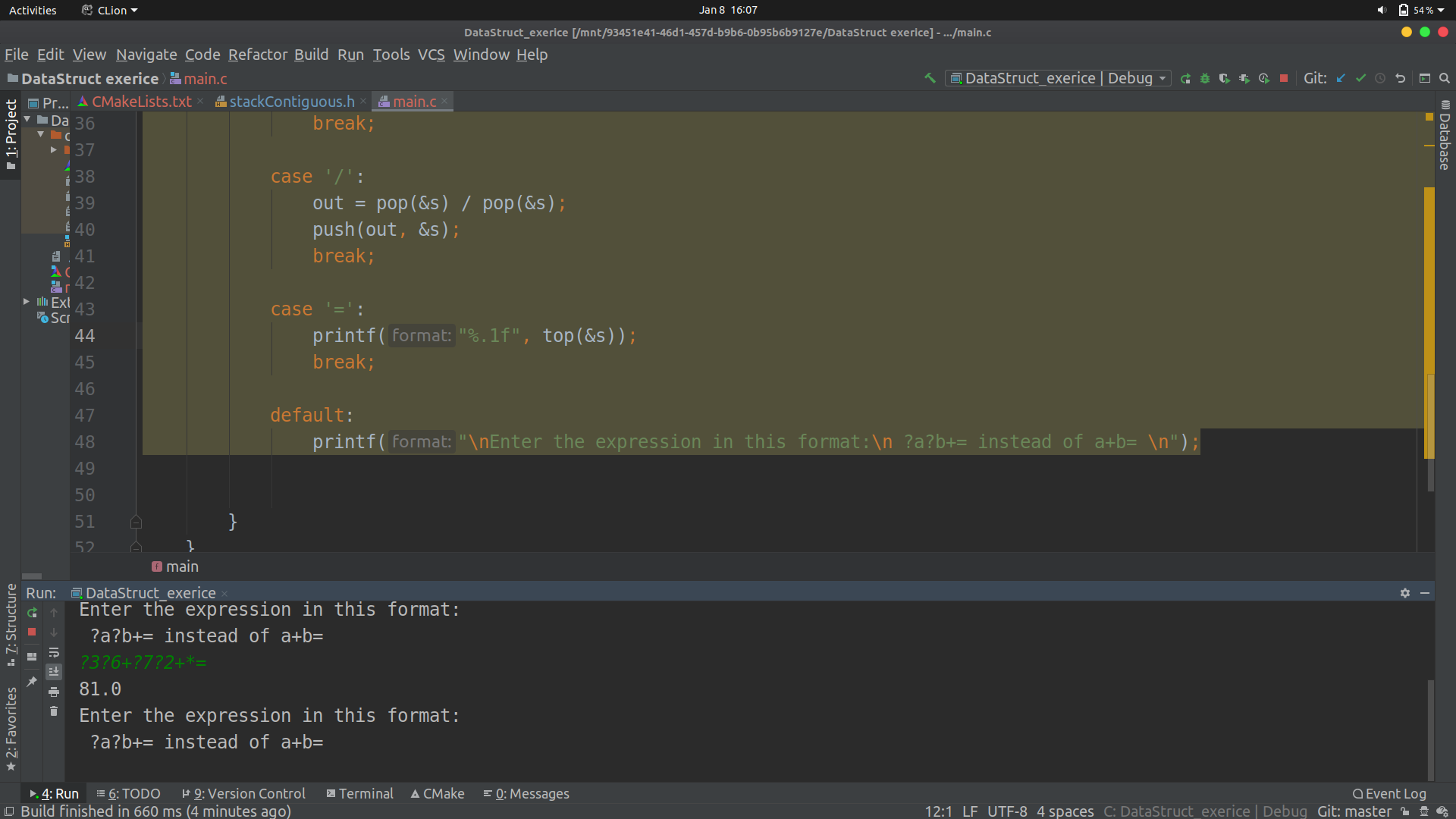
End

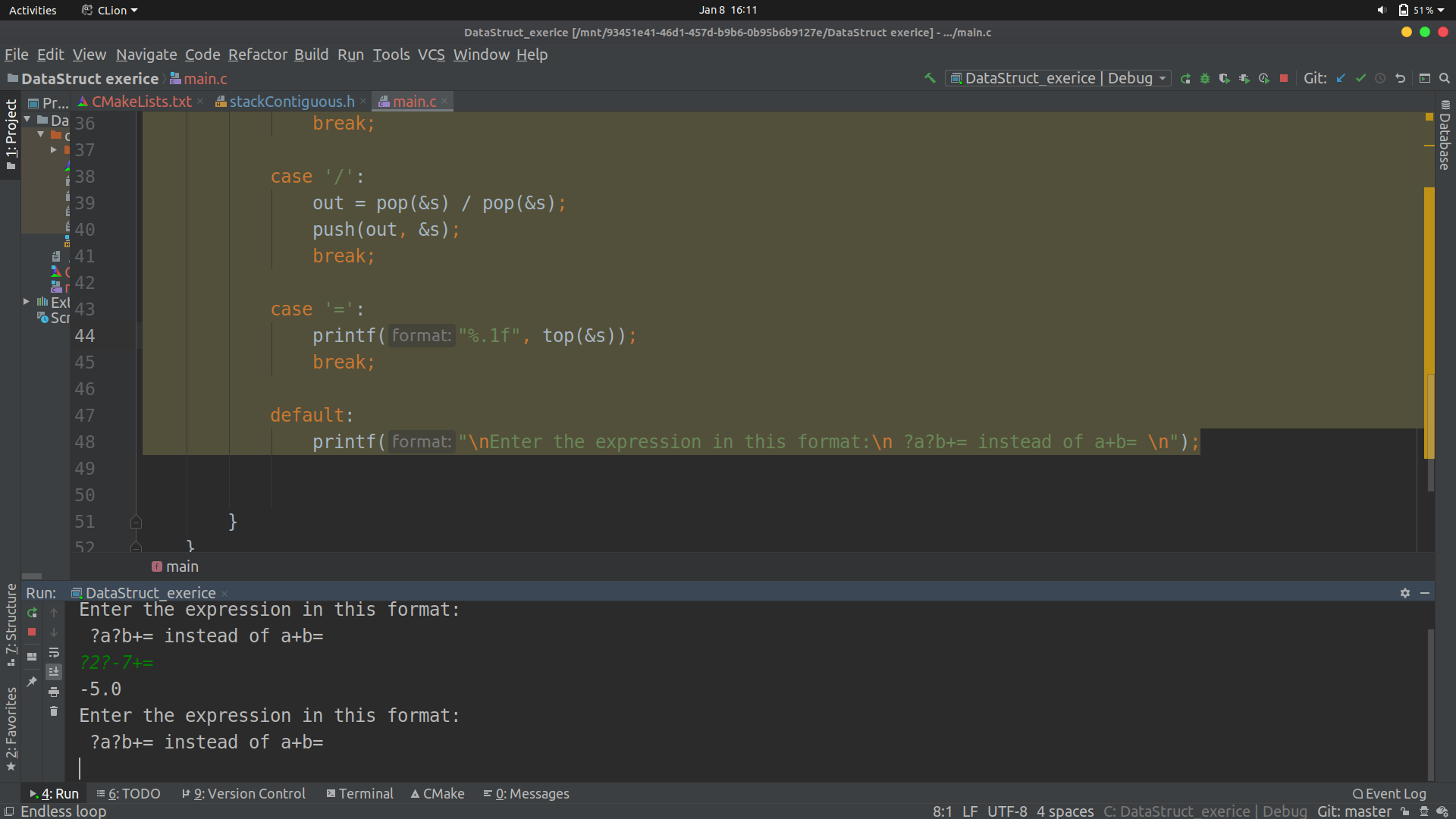
**RESULTS AND DISCUSSION**

1. As the first example our expression is **6.28+3.14=** So we are going to add two double operands as 3.14 and 6.28 according to the given format below. User must insert the operands in order to the postfix notation. It means initially ?3.14 then ?6.28 and the operator (+) finally the = sign. So we will get the answer as follows.

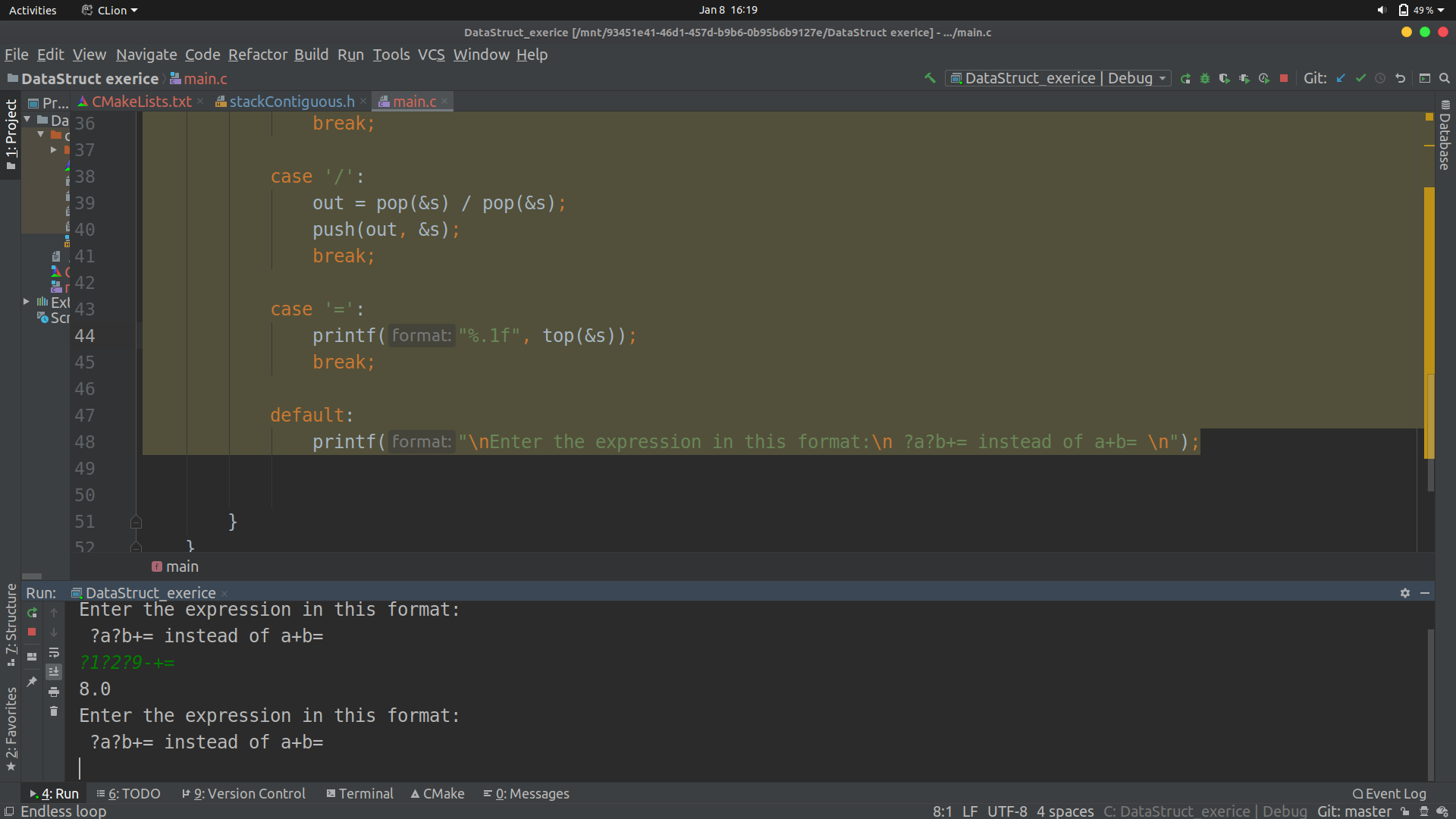
2. In our second example there are two parts of the expression as **(6+3)\*(7+2)=** In the first part we have added two integer operands as 3 and 6. And the second part also we have added two integers as 7 and 2. Then the both of the results of first and second part will be multiplied as following.

As the procedure of the whole execution, initially 3 and 6 operands will be pushed on the stack and the result which we will get by adding those two operands will be pushed back again to the stack. In the second part 7 and 2 operands will be pushed on the stack. Then the stack will contains result of the first part (3+6) , 7 and 2. Then top two of the operands in the stack will be added and that result also will be pushed back again to the stack. Finally, by entering the = sign two parts of the results will be multiplied.

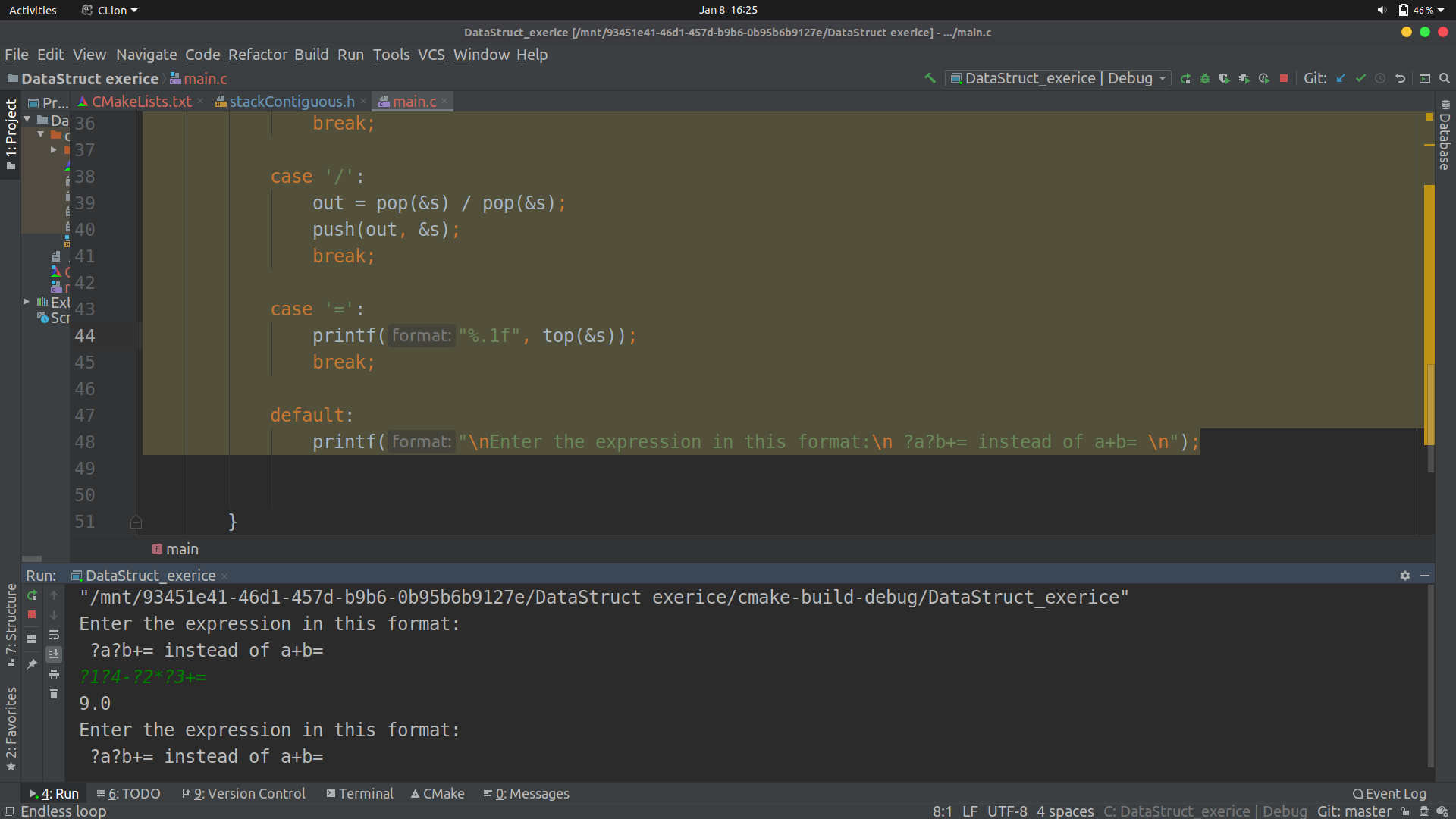


3.As the third example, **-7+2 =** we are going to add two integer operands as 2 and -7. According to the given format below. User must insert the operands in order to the postfix notation. It means initially ?2 then ?-7 and the operator (+) finally the = sign. So we will get the answer as follows.

4. In the fourth example there are three integer operands as 1 , 2 and 9 of the expression to be performed. So the expression will be **9-2+1=** . Initially the user has to insert 1 in to the stack. And wants to insert 2 in to the stack. After that by applying the (**-**) operator, it will get the result of those previously inserted two operands, and then the result will be pushed back again to the Stack. Then the user has to push operand 9 in to the stack. Since it has no more operands, it can insert **(+)** and **(=)** operators according to the postfix method and the answer will be as following.



5. As our fifth example there are three parts of the expression. In the first part we have to subtract 1 from 4. And in the second part we have to multiply it by 2. Then add 3 to the result of the above part of the expression**, (4-1)\*2+3=**

As the procedure of the whole execution, initially 1 and 4 operands will be pushed on the stack and the result which we will get by subtracting those two operands will be pushed back again to the stack.. Then the stack will contains result of the first part (4-1). Then top two of the operands in the stack will be subtract and that result also will be pushed back again to the stack. In the second part operand 2 will be multiply with the first part and pushed on the stack . Then add 3 to the above part of the expression. Finally, by entering the = sign three parts of the result will be simplified.

**CONCLUSION**

As the conclusion we can mention that the Reverse Polish Calculator has more benefits rather than a Traditional calculator. So let’s discuss about some advantages of this type of calculators.

* RPN saves time and also less key strokes in many cases. We can avoid using and keeping track of parentheses while doing calculations.
* More flexible. Being able to swap values in the stack allows for complex partial results to be calculated first.
* RPN is logical because the user first gives the number and then tells what to do with it.
* One immediate advantage of reverse Polish is that it does generalize to number of operators where infix notation is really stuck working with two operands.
* An intermediate result allows the user to check the answer and correct errors more easily. It's easier to follow the stream of calculation. The user defines the priority of operators.
* In Reverse Polish calculator operator priorities can be represented by the order that they occur. So it doesn’t need a bracket to represent an RPN expression although they can be incorporated as operators to make conversion between infix and RPN easier.
* In RPN an operator looks just like a function which takes as arguments the two things written to its left. If you know about lambda expression this will seem very familiar.

As well as the benefits of this calculator there might be some issues also.

* This RPC is not familiar with everyone. Because it depends on the knowledge of the postfix concept.
* RPN or Reverse Polish Notation used to be a basic of the computer programmer's world, but today it is not as well known.

When we are comparing both benefits and issues we have concluded that this is suitable calculator for complex calculations.

Diary and attendance sheet

|  |  |  |  |
| --- | --- | --- | --- |
| Date | Task | Student Numbers | Signature |
| 2019-12-02 | Discuss with the group members and appointed a group leader and have discuss about the problem | CT/2016/012  CT/2016/013  CT/2016/015  CT/2016/028  CT/2016/048  CT/2016/070 |  |
| 2019-12-08 | Analyze the problem and divided each parts among the group members | CT/2016/012  CT/2016/013  CT/2016/015  CT/2016/028  CT/2016/048  CT/2016/070 |  |
| 2019-12-15 | We tried to implement the code according to the given scenario | CT/2016/012  CT/2016/013  CT/2016/015  CT/2016/028  CT/2016/048  CT/2016/070 |  |
| 2019-12-23 | Met the course instructor and got the ideas identified the issues which we had previously | CT/2016/012  CT/2016/013  CT/2016/015  CT/2016/028  CT/2016/048  CT/2016/070 |  |
| 2019-12-31 | We found some errors in our stack header file as well as main file. We discussed and again tried to implement | CT/2016/012  CT/2016/013  CT/2016/015  CT/2016/028  CT/2016/048  CT/2016/070 |  |
| 2020-01-03 | We created the final report according to the provided format | CT/2016/012  CT/2016/013  CT/2016/015  CT/2016/028  CT/2016/048  CT/2016/070 |  |
| 2020-01-06 | Finally we prepared the presentation and finalized the report | CT/2016/012  CT/2016/013  CT/2016/015  CT/2016/028  CT/2016/048  CT/2016/070 |  |