CHANDIGARH UNIVERSITY
Discover. Learn. Empower.

PROJECT REPORT
ON
The Simple Calculator

Program Name: BCA
Subject Name/Code : Computer aprogramming(24CAH101)

Submitted by:

Name: Niyati Rastogi

UID: 24bca10366

Section: 6

Group: B

Submitted to:

Name: Mrs.Sonia

Designation:

1. Aims:

The Simple Calculator in C++ is a basic application that performs arithmetic operations such as addition, subtraction, multiplication, and division.

- Understanding how to take input from the user and display the output.
- Using control structures (such as **if-else** or **switch**) to handle various operations.
- Managing division by zero and other potential input errors.
- Defining and organizing functions for different operations to promote code reusability and readability.

1.2, Objective: -

To develop a basic C++ program that takes two input numbers and performs arithmetic operations based on user selection.
To understand the structure and syntax of C++ programming, including functions, control statements, and input/output handling.

1.3, Goals: -

fundamental programming constructs in C++, such as **data types, variables, operators, and control structures.**
Enhance logical thinking and problem-solving skills through the implementation of basic **arithmetic functions.**
Introduce the concept of user interaction in programs using **cin** and **cout**.

1.4, Overview: -

The Simple Calculator performs the following operations:

- **Addition:** Adds two numbers.
- **Subtraction:** Subtracts one number from another.
- **Multiplication:** Multiplies two numbers.
- **Division**: Divides one number by another, with a check to prevent division by zero.

2.Task to be done:

1. **Set Up the Program Structure: -**
   - Start with including the necessary header files, such as **<iostream>** for input and output operations.
   - Use the **std namespace** to simplify syntax.
2. **Define the Arithmetic Functions: -**
   - Define functions for each operation (**addition, subtraction, multiplication, and division**).
   - Each function should accept two numbers as parameters and return the result.
3. **Create the User Interface: -**
   - Display a menu with options for the user to choose an operation.
   - Use **cin** to accept user input for the operation and operands.
4. **Implement Input Validation: -**
   - Check for invalid inputs (e.g., **division by zero**).
   - Display error messages for invalid inputs to make the program more robust.
5. **Execute the Selected Operation :-**
   - Use a **switch statement** or conditional logic to perform the selected operation based on user choice.
   - Display the result to the user.
6. **Loop the Program (Optional) :-**
   - Allow the user to perform another calculation or exit the program.
   - Use a loop (**like while or do-while**) to keep the program running until the user decides to exit

3. Algorithm:

Step 1. Start

Step 2. Display a menu of operations (Addition, Subtraction, Multiplication, Division).

Step 3. select an operation.

Step 4. Read the user's choice.

Step 5. enter two numbers.

Step 6. Perform the selected operation based on the user's choice:

If the operation is addition, add the two numbers.

If the operation is subtraction, subtract the second number from the first.

If the operation is multiplication, multiply the two numbers.

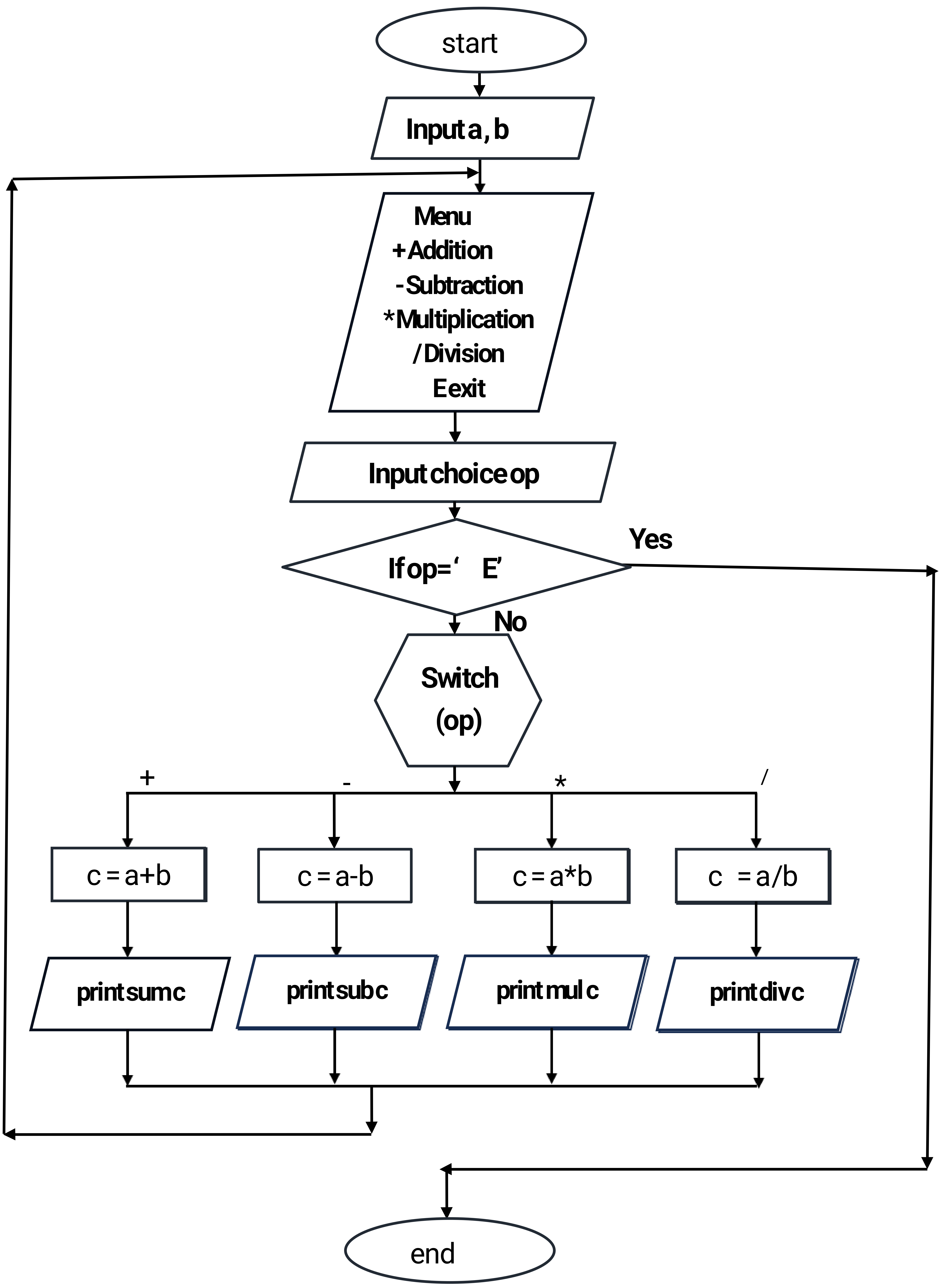
If the operation is division, check if the second number is not zero (to avoid division by zero); if not, divide the first number by the second.

Step 7. Display the result.

Step 8. Ask the user if they want to perform another calculation.

If yes, repeat from step 3. If no, proceed to step 9.**Step 9**. **End.**

4. Flowchart of arithmetic operators:



5. Dataset:

Dataset Structure:

Each entry in the dataset can include:

Input 1: The first number for the calculation.

Input 2: The second number for the calculation.

Operation: The arithmetic operation to perform (+, -, *, /).

Expected Output: The expected result of the calculation.

Error Message: An error message for operations that may cause errors (like division by zero).

Sample dataset:

Input 1	**Input 2**	**operators**	**Expected output**
43	30	+	73
76	32	-	44
40	20	*	800
48	6	/	8
10	0	Error: Division by zero	Division by zero

Dataset:

1. **Testing Functionality:** Use these inputs in your calculator program to test if it produces the expected outputs.
2. **Validating Error Handling:** Specifically test the cases involving division by zero and other edge cases to ensure the calculator responds appropriately.
3. **Improving the Code:** If the outputs do not match the expected results, debug your code to fix any issues.

6. procedure Code:

```
#include <iostream>
using namespace std;

void displaymenu()  {
cout << "Simple Calculator" << endl;
cout << "1. Adition (+)" << endl;
cout << "2. Subtraction (-)" << endl;
cout << "3. Multiplicatin (*)" << endl;
cout << "4. Division (/)" << endl;
}
double add(double a, double b) {
  return a + b;
}
double subtract(double a, double b) {
  return a - b;
}
double multiply(double a, double b) {
  return a * b;
}
double divide(double a, double b) {
  if (b == 0) {
throw invalid_argument("   division by zero not allowed."   ):
}
Return a/b;
}
int main() {
double num1, num2;
char operation;
bool running = true;
while (running) {
   displaymenu();
int choice;
cin>>choice;

if (choose==5) {
running==false;
break;

cout << "Enter the number: ";
 cout >> num1 >> num2 ;
try {
switch (choice) {
      case 1:
      cout << " result " << add (num1, num2) << endl;
       break;
       case 2:
      cout << " result " << sub (num1,num2) << endl;
      break;
       case 3:
       cout << " result " << multiply (num1,num2) << endl;
      break;
      case 4:
       cout << " result " << divide (num1, num2) << endl;
       break;
       deafult:
        cout << " invalid input! "   << endl;
         break;
     }
} catch (const invalid argument& e) {
  Cout << e. what() << endl;
}
}
Cout << "  calculator  closed. "  << endl;
Return 0;
}
```

7. Learning outcomes (What I have learnt):

1. **Basic C++ Syntax and Structure**:

   Gained familiarity with C++ syntax, including variable declarations, **data types**, and **control structures like loops** and **conditionals.**

2. **Function Creation and Utilization**:
   Learned how to define and implement functions to perform specific tasks (such as **arithmetic operations**), improving code organization and readability.

3. **User Input and Output Management**:
   Developed skills in handling user input and displaying output effectively using **the console,** enhancing user interaction with the application.

4. **Error Handling Techniques**:
   Understood the significance of error handling, especially for operations **like division**, and implemented mechanisms to manage exceptions and prevent program crashes.

5. **Problem-Solving and Algorithmic Thinking**:
   Improved problem-solving abilities by breaking down complex tasks into simpler steps, fostering an understanding of how algorithms work in programming