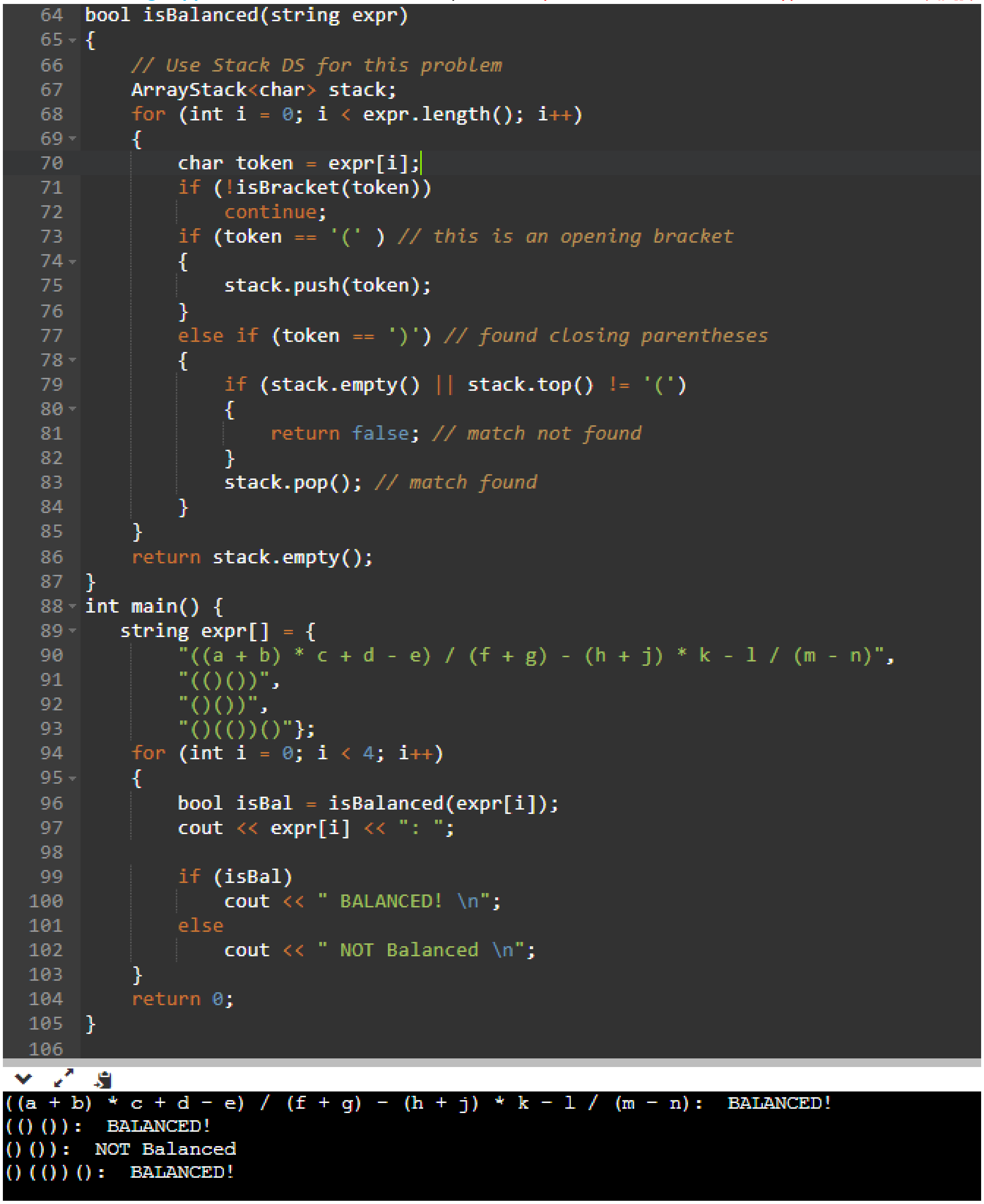
Birla Institute of Technology and Science, Pilani Hyd Campus

BITS F232: Foundations of Data Structures and Algorithms

1st Semester 2023-24 Lab No:6 (Stack ADT)

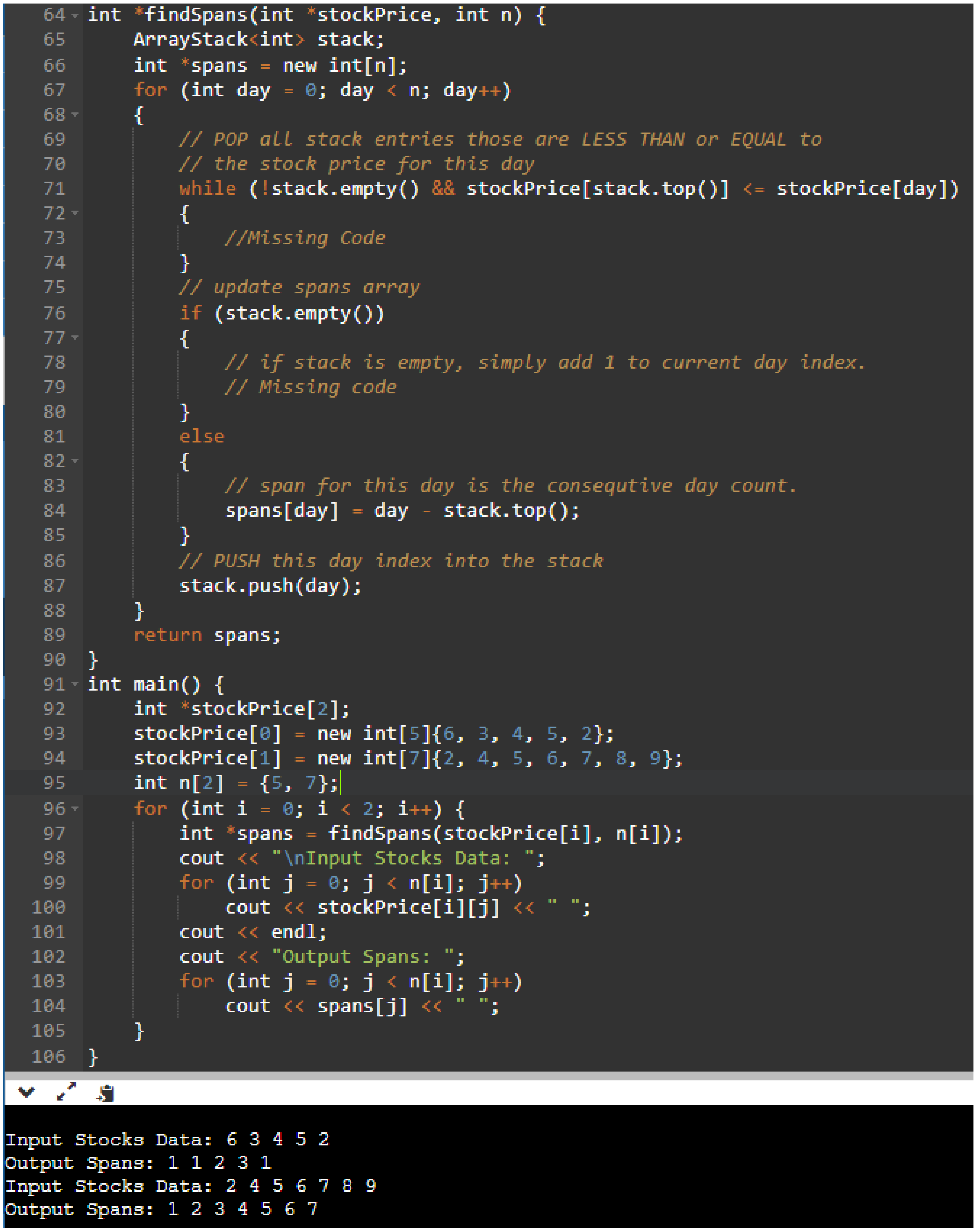
Program 1: The Balanced Parentheses problem discussed in the class is given partly below. Rest all part is given in attached Prog1.cpp. Run the code to see the output. Modify the code to handle other types of brackets ({} []).



```cpp
bool isBalanced(string expr)
{
    // Use Stack DS for this problem
    ArrayStack<char> stack;
    for (int i = 0; i < expr.length(); i++)
    {
        char token = expr[i];
        if (!isBracket(token))
            continue;
        if (token == '(' ) // this is an opening bracket
        {
            stack.push(token);
        }
        else if (token == ')') // found closing parentheses
        {
            if (stack.empty() || stack.top() != '(')
            {
                return false; // match not found
            }
            stack.pop(); // match found
        }
    }
    return stack.empty();
}
int main() {
   string expr[] = {
        "((a + b) * c + d - e) / (f + g) - (h + j) * k - l / (m - n)",
        "(()())",
        "()())",
        "()(())()"};
    for (int i = 0; i < 4; i++)
    {
        bool isBal = isBalanced(expr[i]);
        cout << expr[i] << ": ";

        if (isBal)
            cout << " BALANCED! \n";
        else
            cout << " NOT Balanced \n";
    }
    return 0;
}
```

```
((a + b) * c + d - e) / (f + g) - (h + j) * k - l / (m - n):  BALANCED!
(()()):  BALANCED!
()()):  NOT Balanced
()(())():  BALANCED!
```

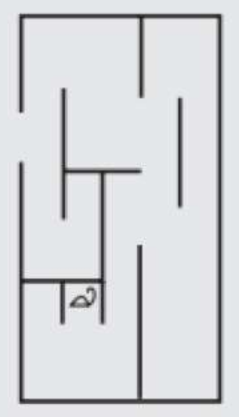
Program 2: The Stock pan example discussed in the class given partly as below. Find out the missing code at line numbers 73, and 79 and run the code to get the output as shown below. Remaining part of the code is given in attached Prog2.cpp. Run the program with different test cases and verify the results.

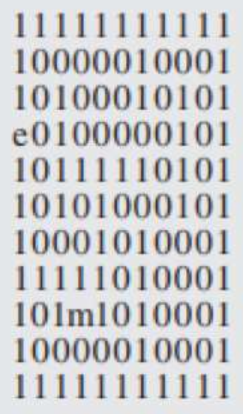


```
 64  int *findSpans(int *stockPrice, int n) {
 65      ArrayStack<int> stack;
 66      int *spans = new int[n];
 67      for (int day = 0; day < n; day++)
 68      {
 69          // POP all stack entries those are LESS THAN or EQUAL to
 70          // the stock price for this day
 71          while (!stack.empty() && stockPrice[stack.top()] <= stockPrice[day])
 72          {
 73              //Missing Code
 74          }
 75          // update spans array
 76          if (stack.empty())
 77          {
 78              // if stack is empty, simply add 1 to current day index.
 79              // Missing code
 80          }
 81          else
 82          {
 83              // span for this day is the consequtive day count.
 84              spans[day] = day - stack.top();
 85          }
 86          // PUSH this day index into the stack
 87          stack.push(day);
 88      }
 89      return spans;
 90  }
 91  int main() {
 92      int *stockPrice[2];
 93      stockPrice[0] = new int[5]{6, 3, 4, 5, 2};
 94      stockPrice[1] = new int[7]{2, 4, 5, 6, 7, 8, 9};
 95      int n[2] = {5, 7};
 96      for (int i = 0; i < 2; i++) {
 97          int *spans = findSpans(stockPrice[i], n[i]);
 98          cout << "\nInput Stocks Data: ";
 99          for (int j = 0; j < n[i]; j++)
100              cout << stockPrice[i][j] << " ";
101          cout << endl;
102          cout << "Output Spans: ";
103          for (int j = 0; j < n[i]; j++)
104              cout << spans[j] << " ";
105      }
106  }
```

```
Input Stocks Data: 6 3 4 5 2
Output Spans: 1 1 2 3 1
Input Stocks Data: 2 4 5 6 7 8 9
Output Spans: 1 2 3 4 5 6 7
```

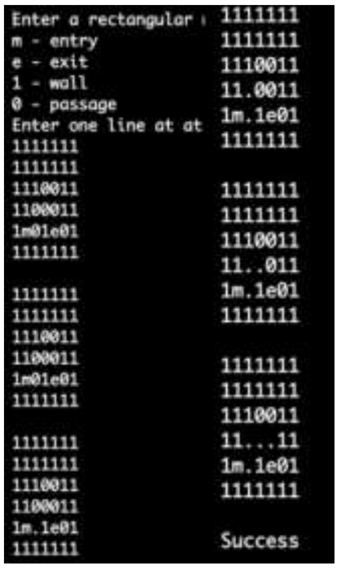
Program 3: Consider the problem of a trapped mouse (Fig. a) that tries to find its way (from position m: Fig. b) to an exit (position e: Fig. b) in a maze as discussed in the class (Fig. c). If it reaches a dead end, it retraces its steps to the last position and begins at least one more untried path using backtracking. To protect itself from falling, the mouse also has to constantly check whether it is in such a borderline position. To avoid it, the program automatically puts a frame of 1s around the maze entered by the user. The program uses two stacks: one to initialize the maze and another to implement backtracking. The user enters a maze one line at a time. The rows are pushed on the stack **mazeRows** in the order they are entered after attaching one 1 at the beginning and one 1 at the end. After all rows are entered, the size of the array store can be determined, and then the rows from the stack are transferred to the array. A second stack, **mazeStack**, is used in the process of escaping the maze. To remember untried paths for subsequent tries, the positions of the untried neighbours of the current position (if any) are stored on a stack and always in order, first upper neighbour, then lower, then left, and finally right. To avoid falling into an infinite loop of trying paths that have already been investigated, each visited position of the maze is marked with a period.


(a)



```
11111111111
10000010001
10100010101
e0100000101
10111110101
10101000101
10001010001
11111010001
101m1010001
10000010001
11111111111
```

(b)

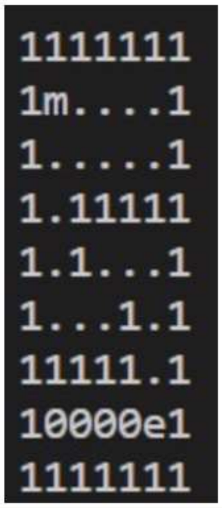


```
Enter a rectangular
m - entry
e - exit
1 - wall
0 - passage
Enter one line at at
1111111
1111111
1110011
1100011
1m01e01
1111111

1111111
1111111
1110011
1100011
1m01e01
1111111

1111111
1111111
1110011
1100011
1m.1e01
1111111
```

```
1111111
1111111
1110011
11.0011
1m.1e01
1111111

1111111
1111111
1110011
11..011
1m.1e01
1111111

1111111
1111111
1110011
11...11
1m.1e01
1111111

Success
```

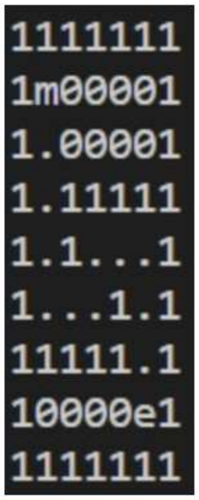
(c)

You are given with the C++ code (Prog3.cpp) to implement this maze path finding using a stack. Run this program to get similar output as shown in Fig. c. Also, try with multiple inputs. Your task in this program is to modify the given code so that it randomly selects unvisited squares/ nodes in place of top, down, left and right square sequences (being pushed onto the stack). So that, you get different paths for the same input when run multiple times as shown in below figures (Fig. d) and (Fig. e). You may also choose any other order.



```
1111111
1m....1
1.....1
1.11111
1.1...1
1...1.1
11111.1
10000e1
1111111
```

(Fig. d)



```
1111111
1m00001
1.00001
1.11111
1.1...1
1...1.1
11111.1
10000e1
1111111
```

(Fig. e)

Program 4: Imagine that you are responsible for keeping the score for a new number game invented by Recreation Activity Forum (RAF) of BITS Hyd campus with strange rules. You are to maintain a record for the game, and are given a string called operations, where operations[i] is the ith operation you must apply to the record. The operations can be any of the following:

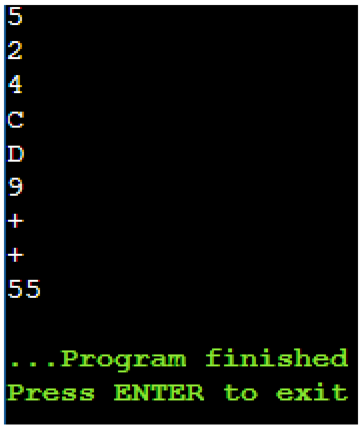
- An integer x: Record a new score of x.
- '+': Record a new score that is the sum of the previous two scores.
- 'D' : Record a new score that is the double of the previous score.
- 'C': Invalidate the previous score, removing it from the record.

Return the sum of all the scores on the record after applying all the operations.

Below are few examples:

Input: ops = "52CD+"
Output: 30
Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"C" - Invalidate and remove the previous score, record is now [5].
"D" - Add 2 * 5 = 10 to the record, record is now [5, 10].
"+" - Add 5 + 10 = 15 to the record, record is now [5, 10, 15].
The total sum is 5 + 10 + 15 = 30.

Input: ops = "524CD9++"
Output: 55
Explanation:
"5" - Add 5 to the record, record is now [5].
"2" - Add 2 to the record, record is now [5, 2].
"4" - Add 4 to the record, record is now [5, 2, 4].
"C" - Invalidate and remove the previous score, record is now [5, 2].
"D" - Add 2 * 2 = 4 to the record, record is now [5, 2, 4].
"9" - Add 9 to the record, record is now [5, 2, 4, 9].
"+" - Add 4 + 9 = 13 to the record, record is now [5, 2, 4, 9, 13].
"+" - Add 9 + 13 = 22 to the record, record is now [5, 2, 4, 9, 13, 22].
The total sum is 5+2+4+9+13+22 = 55.



```
5
2
4
C
D
9
+
+
55

...Program finished
Press ENTER to exit
```

You are given with C++ code for the above task (Prog4.cpp). Your task is to fill the code missing in the second last else if part.
