

Consider the simple calculator program on the following slides. The program reads simple integer expressions from an input file and evaluates them:

Input:

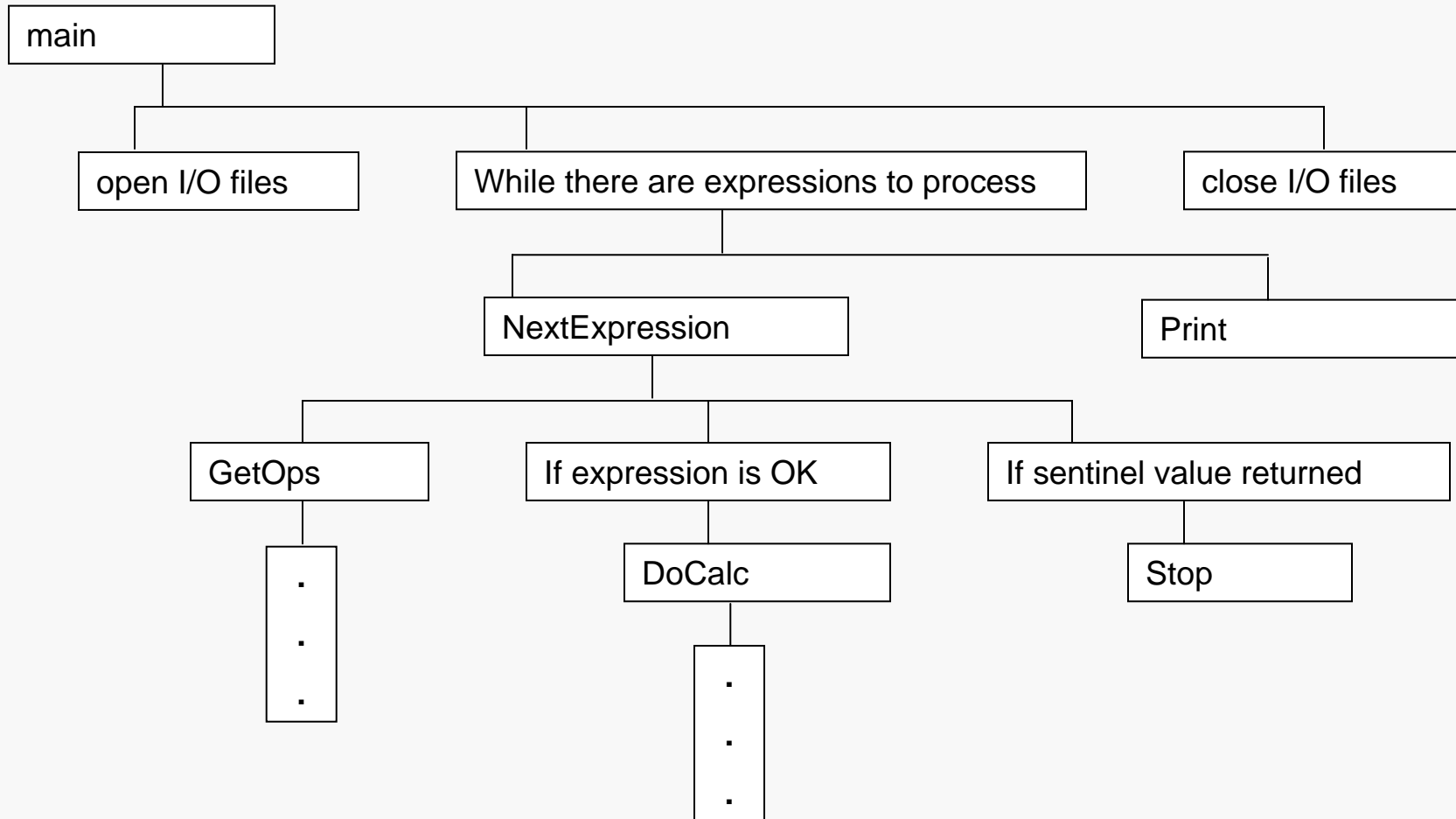
17	+	43
67	-	14
89	*	12
43	/	7

Output:

17	+	43	=	60
67	-	14	=	53
89	*	12	=	1068
43	/	7	=	6
No more expressions.				

This program also illustrates using the placement of function prototypes to restrict access so that only functions that need to make calls can do so.

An analysis of the formal specification leads to the following design:



```
. . .
int main( ) {
    int  NextExpression(ifstream& In, int& Op1, char& Op, int& Op2);
    void Print(int Op1, char Op, int Op2, int Val);
    int  Operand1, Operand2, Value = 0;
    char Operator;
    ifstream iFile;

    iFile.open("Calculator.data");
    while (iFile && Value != INT_MIN) {
        Value = NextExpression(iFile, Operand1, Operator, Operand2);
        if (Value == INT_MIN) {
            cout << "No more expressions." << endl;
            iFile.close();
            return 0;
        }
        Print(Operand1, Operator, Operand2, Value);
    }
    iFile.close(); // probably never executed, included for safety
    return 0;
}
```

```
int NextExpression(istream& In, int& Op1, char& Op, int& Op2) {
    int DoCalc(int Op1, char Op, int Op2);
    void GetOps(istream& In, int& Op1, char& Op, int& Op2);

    GetOps(In, Op1, Op, Op2);                // get expression

    if (In)
        return DoCalc(Op1, Op, Op2);        // evaluate if OK
    else
        return INT_MIN;                     // return flag if not
}

void GetOps(istream& In, int& Op1, char& Op, int& Op2) {

    In >> Op1 >> Op >> Op2;                // read expression
    In.ignore(80, '\n');                    // skip to beginning of next line
    return;
}
```

```
void Print(int Op1, char Op, int Op2, int Val) {
    cout << setw(5) << Op1 << ' ' << Op << ' '
         << setw(5) << Op2 << " = "
         << setw(5) << Val << endl;
}

int DoCalc(int Op1, char Op, int Op2) {
    int Result;

    switch ( Op ) {                // consider operation
    case '+': Result = Op1 + Op2;
              break;
    case '-': Result = Op1 - Op2;
              break;
    case '*': Result = Op1 * Op2;
              break;
    case '/': Result = Op1 / Op2;   // no protection against Op2 == 0
              break;
    default:  Result = INT_MIN;     // return flag if operation
    }                                // is invalid
    return Result;
}
```