// Compute average grade for one student - Version 6
#include <iostream>    // for cin, cout
using namespace std;

// Function Prototypes
int compute_grade_sum(int score1, int score2, int score3);
double compute_grade_average(int score_sum);

// array argument
void output_student_report(char name[], int test1, int test2, int test3,
                             int test_sum, double test_average);
void set_fp_format();

int main()
{
    int grade1, grade2, grade3, grade_sum;
double average_grade;
char student_name[80];
    // Input student's data
    cout << "Midterm Grade Report\n";
cout << "Enter student's name ==> ";
cin.getline(student_name, 80);
cout << "Enter first grade =====> ";
cin >> grade1;
cout << "Enter second grade ====> ";
cin >> grade2;
cout << "Enter third grade =====> ";
cin >> grade3;

    // Function Calls to
    // compute sum and average of the three grades
    grade_sum = compute_grade_sum(grade1, grade2, grade3);
    average_grade = compute_grade_average(grade_sum);

    // Function Call to output the student report
    output_student_report(student_name, grade1, grade2, grade3,
                           grade_sum, average_grade);
    return 0;
}

// Function Definitions
int compute_grade_sum(int score1, int score2, int score3)
{
    return score1 + score2 + score3;
}

double compute_grade_average(int score_sum)
{
    const double NUMBER_OF_SCORES = 3.0;
    return score_sum / NUMBER_OF_SCORES;
}
// array argument
void output_student_report(char name[], int test1, int test2, int test3,
                          int test_sum, double test_average)
{
    set_fp_format();
    cout << "\nStudent's name: " << name;
    cout << "\nTest grades: " << test1 << " " << test2 << " " << test3;
    cout << "\nGrade sum: " << test_sum;
    cout << "\nGrade average: " << test_average;
    cout << "\n\nEnd of Midterm Grade Report\n";
} // return; is optional in a void function

void set_fp_format()
{
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);
}

Run 1:
Midterm Grade Report
Enter student's name ==> Carol Wong
Enter first grade =====> 100
Enter second grade =====> 76
Enter third grade =====> 83
Student's name: Carol Wong
Test grades:  100 76 83
Grade sum:  259
Grade average:  86.3
End of Midterm Grade Report

Run 2:
Midterm Grade Report
Enter student's name ==> Jim Garcia
Enter first grade =====> 100
Enter second grade =====> 100
Enter third grade =====> 100
Student's name: Jim Garcia
Test grades:  100 100 100
Grade sum:  300
Grade average:  100.0
End of Midterm Grade Report
Structure Chart:

- Produce Grade Report
  - Compute Grade Sum
  - Compute Grade Average
  - Output Student Report
    - Set Floating-point Format
Functions for all Subtasks

Stack at return of compute_grade_sum():

<table>
<thead>
<tr>
<th>Copy of grade</th>
<th>Score</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade1</td>
<td>100</td>
<td>259</td>
</tr>
<tr>
<td>grade2</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>grade3</td>
<td>83</td>
<td>compute_grade_sum()</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Student Name</th>
<th>Average Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Carol Wong</td>
<td>garbage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grade</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade3</td>
<td>83</td>
</tr>
<tr>
<td>grade2</td>
<td>76</td>
</tr>
</tbody>
</table>

Stack at return of compute_grade_average():

<table>
<thead>
<tr>
<th>Copy of grade sum</th>
<th>Score sum</th>
<th>Return Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade sum</td>
<td>259</td>
<td>86.3333333333333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Student Name</th>
<th>Average Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Carol Wong</td>
<td>garbage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grade</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade3</td>
<td>83</td>
</tr>
<tr>
<td>grade2</td>
<td>76</td>
</tr>
</tbody>
</table>

Stack during output_student_report():

<table>
<thead>
<tr>
<th>Copy of student name</th>
<th>Name</th>
<th>Copy of grade</th>
<th>Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>student name</td>
<td>Carol Wong</td>
<td>grade1</td>
<td>100</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grade2</td>
<td>76</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grade3</td>
<td>83</td>
</tr>
<tr>
<td></td>
<td></td>
<td>grade sum</td>
<td>259</td>
</tr>
<tr>
<td></td>
<td></td>
<td>average grade</td>
<td>86.3333333333333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Student Name</th>
<th>Average Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Carol Wong</td>
<td>86.3333333333333</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grade</th>
<th>Copy</th>
</tr>
</thead>
<tbody>
<tr>
<td>grade3</td>
<td>83</td>
</tr>
<tr>
<td>grade2</td>
<td>76</td>
</tr>
</tbody>
</table>
1. \[\text{int } a = 2, \ b = 3;\]

\[
\begin{array}{c|c|c}
\text{Before} & \text{After} \\
\hline
b & 3 & b \\
a & 2 & a \\
\end{array}
\]

2. \[\text{a = b;} \quad \text{// Doesn't work}\]
   \[\text{b = a;}\]

3. \[\text{int } a = 2, \ b = 3, \ \text{temp;}\]
   
   ...  
   
   \[
   \begin{array}{c|c|c}
   \text{Before} & \text{After} \\
   \hline
   \text{temp} & ? & \text{temp} \\
   b & 3 & b \\
a & 2 & a \\
   \end{array}
   \]

\[
\begin{array}{c|c|c}
\text{Before} & \text{After} \\
\hline
\text{temp} & ? & \text{temp} \\
\text{b} & 3 & \text{b} \\
a & 2 & a \\
\end{array}
\]
// Passing arguments by value

#include <iostream>    // for cout
using namespace std;

void swap(int a, int b); // Function prototype

int main()
{
    int x = 2, y = 3;
    cout << "Before swap in main(), x = " << x
        << " and y = " << y << "\n\n";
    swap(x, y); // Function call
    cout << "After swap in main(), x = " << x
        << " and y = " << y << "\n\n";
    cout << "What happened??????\n";
    return 0;
}

void swap(int a, int b)       // Function definition
{
    int temp;
    cout << "Before swap in swap(), a = " << a
        << " and b = " << b << "\n\n";
    temp = a;
    a = b;
    b = temp;
    cout << "After swap in swap(), a = " << a
        << " and b = " << b << "\n\n";
}

Sample Run:
Before swap in main(), x = 2 and y = 3
Before swap in swap(), a = 2 and b = 3
After swap in swap(), a = 3 and b = 2
After swap in main(), x = 2 and y = 3
What happened??????
Structure Chart:

```
main()

swap()
```

Stack at beginning of swap():

```
<table>
<thead>
<tr>
<th>temp</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
</tr>
<tr>
<td>b</td>
</tr>
</tbody>
</table>

```

| copy of x | 2 | a |
| copy of y | 3 | b |
| y         | 3 |
| x         | 2 |

main()
// Passing arguments by reference

#include <iostream> // for cout
using namespace std;

// reference to int
void swap(int& a, int& b); // Function prototype

int main()
{
    int x = 2, y = 3;

cout << "Before swap in main(), x = " << x << " and y = " << y << "\n";

swap(x, y); // Actual argument must be variable if passed by reference

cout << "After swap in main(), x = " << x << " and y = " << y << "\n";
    return 0;
}

void swap(int& a, int& b) // Function definition
{
    int temp;

cout << "Before swap in swap(), a = " << a << " and b = " << b << "\n";

    temp = a;
    a = b;
    b = temp;

cout << "After swap in swap(), a = " << a << " and b = " << b << "\n";
}

Sample Run:
Before swap in main(), x = 2 and y = 3
Before swap in swap(), a = 2 and b = 3
After swap in swap(), a = 3 and b = 2
After swap in main(), x = 3 and y = 2
Stack at beginning of swap():

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>temp</td>
<td>swap</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>b</td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>2</td>
</tr>
<tr>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

main()

Summary:

a. Pass by Value
   if called function (Callee) does not need to change the actual argument
   which can be a variable, constant or expression.
   Advantage: Callee works with copies of the actual arguments
b. Pass by Reference
   if called function needs to change the actual argument
   which must be a variable.
// Compute average grade for multiple students - Version 7

#include <iostream> // for cin, cout
#include <cstring> // for strcmp()

using namespace std;

// Function Prototypes
void output_welcome_message();
void set_fp_format();
void input_student_name(char name[]); // Array argument
void input_test_grades(int& test1, int& test2, int& test3);
void do_midterm_processing(int test1, int test2, int test3,
                          int& test_sum, double& test_average,
                          int& stu_count, double& average_sum);
void output_student_report(char name[], int test1, int test2, int test3,
                          int test_sum, double test_average);
void output_summary_report(int stu_count, double average_sum);

int main()
{
    int grade1, grade2, grade3, grade_sum;
    int student_count = 0;
    double average_grade;
    double sum_average_grade = 0.0;
    char student_name[80];

    output_welcome_message(); // Pre-Loop
    set_fp_format();

    while (true) // Loop for each student
    {
        input_student_name(student_name);

        // if (strcmp(student_name, "END") == 0)
        if (!strcmp(student_name, "END"))
            break;

        input_test_grades(grade1, grade2, grade3);
        do_midterm_processing(grade1, grade2, grade3,
                              grade_sum, average_grade,
                              student_count, sum_average_grade);
        output_student_report(student_name, grade1, grade2, grade3,
                              grade_sum, average_grade);
    }

    output_summary_report(student_count, sum_average_grade); // Post-Loop
    return 0;
}

void output_welcome_message()
{
    cout << "Midterm Grade Report for CCSF Students\n\n" << "For each student, you will be prompted to enter the name\n" << "and three test grades. To stop, enter END for the name.\n\n";
void set_fp_format()
{
    cout.setf(ios::fixed);
    cout.setf(ios::showpoint);
    cout.precision(1);
}

void input_student_name(char name[])
{
    cout << "Enter student's name ==> ";
    cin.getline(name, 80);
}

void input_test_grades(int& test1, int& test2, int& test3)
{
    cout << "Enter first grade =====> ";
    cin >> test1;
    cout << "Enter second grade ====> ";
    cin >> test2;
    cout << "Enter third grade =====> ";
    cin >> test3;
    cin.get(); // Discard \n character
}

void do_midterm_processing(int test1, int test2, int test3,
                           int& test_sum, double& test_average,
                           int& stu_count, double& average_sum)
{
    const double NUMBER_OF_TESTS = 3.0;
    test_sum = test1 + test2 + test3;
    test_average = test_sum / NUMBER_OF_TESTS;
    stu_count++; // Count the student
    average_sum += test_average; // Accumulate test average
}

void output_student_report(char name[], int test1, int test2, int test3,
                            int test_sum, double test_average)
{
    cout << "\nStudent's name: " << name;
    cout << "\nTest grades: " << test1 << " " << test2 << " " << test3;
    cout << "\nGrade sum: " << test_sum;
    cout << "\nGrade average: " << test_average << endl << endl;
}

void output_summary_report(int stu_count, double average_sum)
{
    cout << "\nNumber of students processed: " << stu_count
    << "\nOverall grade average: "
    << average_sum / stu_count
    << "\nEnd of Midterm Grade Report\n";
}
Sample Run:
Midterm Grade Report for CCSF Students

For each student, you will be prompted to enter the name and three test grades. To stop, enter END for the name.

Enter student's name ==> Carol Wong
Enter first grade =====> 100
Enter second grade =====> 76
Enter third grade =====> 83

Student's name: Carol Wong
Test grades: 100 76 83
Grade sum: 259
Grade average: 86.3

Enter student's name ==> Jim Garcia
Enter first grade =====> 100
Enter second grade =====> 100
Enter third grade =====> 100

Student's name: Jim Garcia
Test grades: 100 100 100
Grade sum: 300
Grade average: 100.0

Enter student's name ==> Bich Tran
Enter first grade =====> 0
Enter second grade =====> 100
Enter third grade =====> 50

Student's name: Bich Tran
Test grades: 0 100 50
Grade sum: 150
Grade average: 50.0

Enter student's name ==> END

Number of students processed: 3
Overall grade average: 78.8

End of Midterm Grade Report
Control Module: function that controls the execution of other functions, which perform the actual work

main() is a control module.

Structure Chart:

Modular Approach: programming style where the logic is broken into smaller, more manageable modules

Easier to code, debug, and understand

Top-Down Design: using a structure chart to design the tasks of the various functions

Module Description: verb adjective noun
Stack at return of `input_student_name()`:

<table>
<thead>
<tr>
<th>student_name</th>
<th>Carol Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum_average_grade</td>
<td>0.</td>
</tr>
<tr>
<td>average_grade</td>
<td>garbage</td>
</tr>
<tr>
<td>student_count</td>
<td>0</td>
</tr>
<tr>
<td>grade_sum</td>
<td>garbage</td>
</tr>
<tr>
<td>grade3</td>
<td>garbage</td>
</tr>
<tr>
<td>grade2</td>
<td>garbage</td>
</tr>
<tr>
<td>main()</td>
<td>grade1</td>
</tr>
<tr>
<td></td>
<td>garbage</td>
</tr>
</tbody>
</table>

Stack at return of `input_test_grades()`:

<table>
<thead>
<tr>
<th>student_name</th>
<th>Carol Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum_average_grade</td>
<td>0.</td>
</tr>
<tr>
<td>average_grade</td>
<td>garbage</td>
</tr>
<tr>
<td>student_count</td>
<td>0</td>
</tr>
<tr>
<td>grade_sum</td>
<td>garbage</td>
</tr>
<tr>
<td>grade3</td>
<td>83</td>
</tr>
<tr>
<td>grade2</td>
<td>76</td>
</tr>
<tr>
<td>main()</td>
<td>grade1</td>
</tr>
<tr>
<td></td>
<td>100</td>
</tr>
</tbody>
</table>

Stack at return of `do_midterm_processing()`:

| copy of grade1     | 100         |
| copy of grade2     | 76          |
| copy of grade3     | 83          |
| student_name       | Carol Wong  |
| sum_average_grade  | 86.33333333 |
| average_grade      | 86.33333333 |
| student_count      | 1           |
| grade_sum          | 259         |
| grade3             | 83          |
| grade2             | 76          |
| main()             | grade1     |
|                    | 100         |
Stack during `output_student_report()`:

<table>
<thead>
<tr>
<th>Copy of grade1</th>
<th>100</th>
<th>test1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy of grade2</td>
<td>76</td>
<td>test2</td>
</tr>
<tr>
<td>Copy of grade3</td>
<td>83</td>
<td>test3</td>
</tr>
<tr>
<td>Copy of grade_sum</td>
<td>259</td>
<td>test_sum</td>
</tr>
<tr>
<td>Copy of average grade</td>
<td>86.3333333</td>
<td>test_average</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student name</th>
<th>Carol Wong</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum_average_grade</td>
<td>86.3333333</td>
</tr>
<tr>
<td>Average grade</td>
<td>86.3333333</td>
</tr>
<tr>
<td>Student count</td>
<td>1</td>
</tr>
<tr>
<td>Grade_sum</td>
<td>259</td>
</tr>
<tr>
<td>Grade3</td>
<td>83</td>
</tr>
<tr>
<td>Grade2</td>
<td>76</td>
</tr>
</tbody>
</table>

Stack during `output_summary_report()`:

<table>
<thead>
<tr>
<th>Copy of student_count</th>
<th>3</th>
<th>stu_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Copy of sum_average_grade</td>
<td>236.333333</td>
<td>average_sum</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Student name</th>
<th>Bich Tran</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sum_average_grade</td>
<td>236.333333</td>
</tr>
<tr>
<td>Average grade</td>
<td>50.0</td>
</tr>
<tr>
<td>Student count</td>
<td>3</td>
</tr>
<tr>
<td>Grade_sum</td>
<td>150</td>
</tr>
<tr>
<td>Grade3</td>
<td>50</td>
</tr>
<tr>
<td>Grade2</td>
<td>100</td>
</tr>
</tbody>
</table>

| Main() | grade1 | 0 |
1. Short Circuit Principle of evaluating compound conditions:
   a. Simple conditions are evaluated left to right
   b. Next simple condition is evaluated only if need be

2. 
   ```
   int grade;
   if (grade >= 0 && grade <= 100)
      cout << "In range";
   else
      cout << "Out of range";
   
   If `grade` is negative, the rightmost condition is not evaluated.
   If `grade` is greater than or equal to zero, the rightmost condition is evaluated.
   ```

3. 
   ```
   char ans;
   do
   {
      ...                
      cout << "Again? ";
      cin >> ans;
   } 
   while (ans == 'Y' || ans == 'y');
   
   If `ans` contains an uppercase `Y`, the rightmost condition is not evaluated.
   If `ans` does not contain an uppercase `Y`, the rightmost condition is evaluated.
   ```

4. 
   ```
   if (num != 0 && 100 / num > 2) // prevents division by 0
      or
      if (num && 100 / num > 2) // zero is false; non-zero is true
   
   If `num` is equal to zero, the rightmost condition is not evaluated.
   If `num` is not equal to zero, the rightmost condition is evaluated.
1. **Editing Loop**: a loop that validates data entered at the keyboard

   General logic:
   
   ```
   do
   {
      input data from the keyboard
   } while (data is Invalid);
   // Exit loop when data is Valid
   ```

2. ```
   int num;
   ...
   do
   {
      cout << "Enter a positive integer: ";
      cin >> num;
   } while (num <= 0);  // Loop while num is invalid
   ```

   a. if (num <= 0) is true, then data is Invalid (loop again)

   b. if (num <= 0) is false, then data is Valid (exit loop)

   c. if !(num <= 0) is !false, then data is Valid (exit loop)

   d. if (num > 0) is true, then data is Valid (exit loop)```
3.  

```cpp
90  int grade;
100  ...
110  do
120  {
130      cout << "Enter a grade in the range of 0 to 100, inclusive: ";
140      cin >> grade;
150  }
160  while (grade < 0 || grade > 100);  // Loop while grade is invalid
```

a. if (grade < 0 || grade > 100) is true, then data is Invalid (loop again)

b. if (grade < 0 || grade > 100) is false, then data is Valid (exit loop)

c. if !(grade < 0 || grade > 100) is !false, then data is Valid (exit loop)

d. if !(grade < 0) && !(grade > 100) is true, then data is Valid (exit loop)

e. if (grade >= 0) && (grade <= 100) is true, then data is Valid (exit loop)

**DeMorgan's Laws:**

Let a and b represent any conditions,

1. !(a || b)  ==  !(a)  &&  !(b)

2. !(a && b)  ==  !(a)  ||  !(b)