**Tutorial 2**

**Question 1**

Go through the codes below. If you want you can run them in Dev C++.

*#include <iostream>*

*using namespace std;*

*int main()*

*{*

*const int A\_SCORE = 90,*

*B\_SCORE = 80,*

*C\_SCORE = 70,*

*D\_SCORE = 60,*

*MIN\_SCORE = 0, // Minimum valid score*

*MAX\_SCORE = 100; // Maximum valid score*

*int testScore;*

*cout << "Enter your numeric test score and I will\n"*

*<< "tell you the letter grade you earned: ";*

*cin >> testScore;*

*if (testScore >= MIN\_SCORE && testScore <= MAX\_SCORE)*

*{*

*// Determine the letter grade.*

*if (testScore >= A\_SCORE)*

*cout << "Your grade is A.\n";*

*else if (testScore >= B\_SCORE)*

*cout << "Your grade is B.\n";*

*else if (testScore >= C\_SCORE)*

*cout << "Your grade is C.\n";*

*else if (testScore >= D\_SCORE)*

*cout << "Your grade is D.\n";*

*else*

*cout << "Your grade is F.\n";*

*}*

*else*

*{*

*cout << "That is an invalid score. Run the program\n"*

*<< "again and enter a value in the range of\n"*

*<< MIN\_SCORE << " through " << MAX\_SCORE << ".\n";*

*}*

*return 0;*

*}*

1. Write out the possible outputs you can drive from this program
2. Discuss areas that are highlighted in yellow. Each area in paragraphs

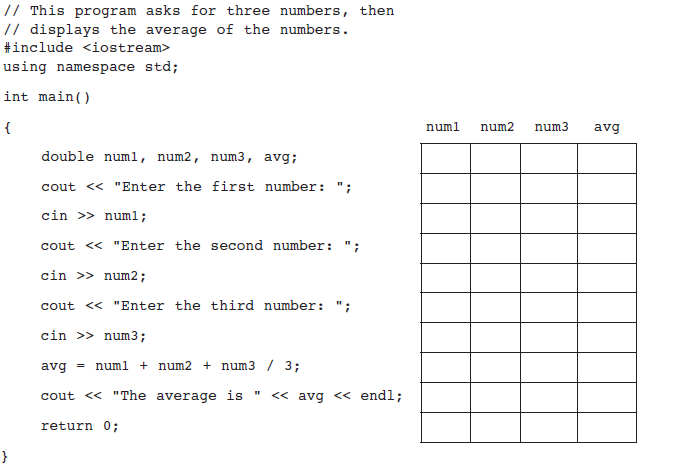
Question 2

*Hand tracing* is a debugging process where you pretend that you are the computer executing a program. You step through each of the program’s statements one by one. As you look at a statement, you record the contents that each variable will have after the statement executes.

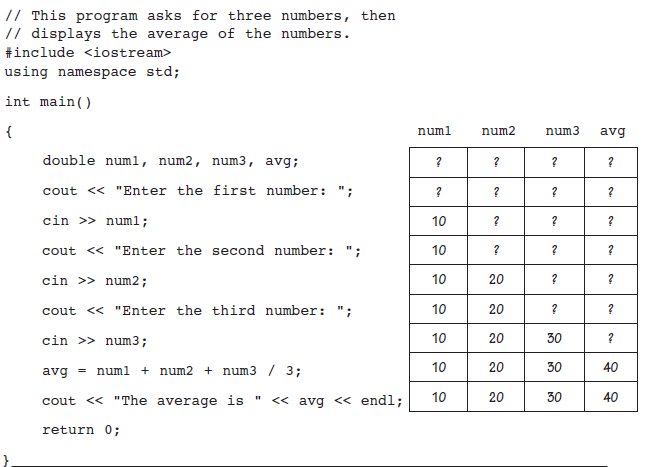
This process is often helpful in finding mathematical mistakes and other logic errors.

To hand trace a program you construct a chart with a column for each variable. The rows in the chart correspond to the lines in the program. The program uses the following four variables: num1 , num2 , num3 , and avg . Notice that the hand trace chart has a column for each variable and a row for

each line of code in function main .



Solution



Question 3

List and discuss briefly the different types of software testing

Acceptance testing: Verifying whether the whole system works as intended.

Integration testing: Ensuring that software components or functions operate together.

Unit testing: Validating that each software unit performs as expected. A unit is the smallest testable component of an application.

Functional testing: Checking functions by emulating business scenarios, based on functional requirements. Black-box testing is a common way to verify functions.

Performance testing: Testing how the software performs under different workloads. Load testing, for example, is used to evaluate performance under real-life load conditions.

Regression testing: Checking whether new features break or degrade functionality. Sanity testing can be used to verify menus, functions and commands at the surface level, when there is no time for a full regression test.

Stress testing: Testing how much strain the system can take before it fails. Considered to be a type of non-functional testing.

Usability testing: Validating how well a customer can use a system or web application to complete a task.

Question 4

Indicate what types issues that can be discovered when software testing is done.

* Architectural flaws
* Poor design decisions
* Invalid or incorrect functionality
* Security vulnerabilities
* Scalability issues