3a.
3a.i
The overall purpose of this program is to simulate dice rolling. The user can select the amount of sides they want on their dice, and then select how many times they would like to roll the dice. This is done using a random number generator, and then various amounts of data related to the specified dice rolling are output. This program can be used to gather statistical data about dice, such as how the average roll changes with an increasing number of rolls.

3a.ii
The video showcases the input and output process the program provides to roll dice and get data. The user specifies the amount of sides they want (in this case 4), and then they are taken to a query about the number of rolls they want (in this case 125), and the program spits out the data. Finally, the program asks if the user wants the data saved to a text file, and provides some options for doing so. Once this process is finished, the program reruns, awaiting a new input.

3a.iii
The user will input a positive integer to represent the amount of sides they want their dice to have. Then the user will input another positive integer to represent the amount of rolls they would like. The program will then output a list of all the possible rolls with the amount of times it rolled each number adjacent, and then the sum of all the rolls, the average roll, and the amount of sides and rolls as originally inputted.

3b.
3b.i

```
106     map<int, int> results;
107     random_device rd_seed;
108     uniform_int_distribution<int> dist(1, dice_sides);
109
110     for (int i = 0; i < rolls; ++i) {
111       ++results[dist(rd_seed)];
112     }
```

3b.ii

```
114     int sum = 0;
115     for (auto [key, val] : results) {
116       sum += key * val;
117     }
```

3b.iii
On line 106, the ordered hashmap named **results** is created and is used on line 111.

3b.iv
The data contains a mapping of “keys” to “values”. In this case the keys would be the possible numbers a dice could roll, and the values would be the amount of times that number has been rolled thus far.

3b.v
On lines 110, 111, and 112, a for loop is run in which for every “roll” of the dice, a random number is created (within the number of sides) and then is used as a unique key in the map to access the number of times this number has been rolled before. The **++** operator at the front increases this count within the map by one. This keeps all of the results neatly organized, is able to store an arbitrary amount of keys and values, and is also easy to use. If only variables were used, it would be impossible to predict the amount of variables that need to be set, and they would not be easily iterable or usable. The hashmap solves all of these problems easily.

3c.
3c.i

```cpp
void save_to_file(char query, const auto& results, const auto& other_data) {
  if (query == 'n') return;
  else if (query != 'y') return;

  cout << "Would you like to save the hits per number in a seperate file from total/average/sides/rolls? y/n\n";
  char seperate_files;
  cin >> seperate_files;

  if (seperate_files == 'y') {
    cout << "Hits file name:\n";

    string hits_filename;
    cin >> hits_filename;

    ofstream hits_file;
    hits_file.open(hits_filename + ".txt");

    for (auto [key, val] : results) {
      hits_file << key << ":\t" << val << '\n';
    }

    hits_file.close();


    cout << "Other data file name:\n";

    string other_filename;
    cin >> other_filename;

    ofstream other_file;
    hits_file.open(other_filename + ".txt");


    hits_file << "Total:\t" << get<0>(other_data)
      << "\nAverage:\t" << get<1>(other_data)
      << "\nSides:\t" << get<2>(other_data)
      << "\nRolls:\t" << get<3>(other_data) << '\n';

    hits_file.close();
  } else {
    cout << "File name:\n";
    string filename;
    cin >> filename;

    ofstream output;
    output.open(filename + ".txt");

    for (auto [key, val] : results) {
      output << key << ":\t" << val << '\n';
    }

    output << "\nTotal:\t" << get<0>(other_data)
      << "\nAverage:\t" << get<1>(other_data)
      << "\nSides:\t" << get<2>(other_data)
      << "\nRolls:\t" << get<3>(other_data) << '\n';

    output.close();
  }
  return;
}
```

3c.ii

```
133     cout << "\n\nWould you like to save your data? y/n\n";
134     char query;
135     cin >> query;
136
137     save_to_file(query, results, other_data);
138   }
139 }
```

3c.iii

The identified procedure, **save_to_file** implements the file saving functionality, such that the outputted data may be saved to an external text file at the user's request.

3c.iv

If the user were to reply ‘y’ to the saving data query, then the procedure would ask if the user would like to split the data into separate files. Then, the algorithm will iterate through the hashmap **results**, piping the data into the specified text file (the name is also user selected), and then selects data inside the **other_data** data structure (it is a “list” which holds 4 specific pieces of information) to be piped into its specified file as well. If the user decided not to split the data, **results** and **other_data** share the same “specified file.”

3d.

3d.i

results and **other_data** will be a sample set of data derived from having a 3 sided dice roll 10 times, and be passed in for both calls as the specifics of their data have no impact on how the function behaves.

First call:
User inputs ‘y’ for **query**.

Second call:
User inputs ‘a’ for **query**.

3d.ii

Condition(s) tested by first call:
Line 10 checks if **query** is equal to ‘n’, if not, it checks if **query** is not equal to ‘y’. Since both checks are false (as **query** here is ‘y’), it continues on past the if statements to line 13 to ask for specific write settings.

Condition(s) tested by second call:

Line 10 checks if **query** is equal to ‘n’, which ‘a’ is not. Due to this, it now runs line 11 to check if **query** is not equal to ‘y’, which is true. This triggers a return which ends the procedure’s execution early.

3d.iii

Results of the first call:

Due to the first call passing the checks, the procedure is able to continue executing. As a result, the user is prompted to select how their data should be saved, and the name(s) of the file(s). Finally, the data is written and saved.

Results of the second call:

Due to the second call failing the checks, the procedure returned and ended early. What this logically means is that either the user entered ‘n’ and decided not to save the data, or that in this case they gave an erroneous input (‘a’) which will default to not saving. Thus, the data was not saved to any separate files.