*Logic Specification Template*

|  |  |  |  |
| --- | --- | --- | --- |
| **Student** | Erick Francisco González Martínez | **Program #** | 5 |

|  |  |
| --- | --- |
| **Class Name** | OutputHandler |

|  |  |
| --- | --- |
| **Method Name** | Double round |

|  |  |
| --- | --- |
| **Parameters** | Double number |

|  |  |  |
| --- | --- | --- |
| If number < 0 | | |
| Return ceil(number – 0.5) | | |
| Return floor(number + 0.5) | | |
| **Method Name** | | String convert |

|  |  |
| --- | --- |
| **Parameters** | Double number |

|  |
| --- |
| string s = to\_string ( round (x \* 100000.0) / 100000.0) |
| int length |
| while (s[s.length() - 1] == '0' && length < s.length()){ |
| length = s.find(".") + 6 |
| } |
|  |
| s.pop\_back(); |
| Return s |

|  |  |
| --- | --- |
| **Method Name** | Void OutputHandler::display |

|  |  |
| --- | --- |
| **Parameters** |  |

|  |
| --- |
| cout << " p = " << results[2] << endl; |
| cout << "dof = " << results[1] << endl; |
| cout << " x = " << convert(results[1]) << endl; |

|  |  |
| --- | --- |
| **Method Name** | OutputHandler(vector<double> temp):results(temp  {}; |

|  |  |
| --- | --- |
| **Parameters** | vector<double> temp |

|  |
| --- |
| //Inline initialization of the vector |

|  |  |
| --- | --- |
| **Class Name** | InputReader |

|  |  |
| --- | --- |
| **Method Name** | InputReader::InputReader() |

|  |  |
| --- | --- |
| **Parameters** |  |

|  |
| --- |
| Double Error = 0 |
| Results(vector<double>(11, 0)) |
| fileName("") |
| num\_seg(100000) |
| p1(0) |
| p2(0) |
| dSumXY = 0 |
| dSumX = 0 |
| dSumY = 0 |
| dSumY2 = 0 |
| dSumX2 = 0 |
| dAvgX = 0 |
| dAvgY = 0 |
| dSumXY = 0 |
| dSumX = 0 |
| dSumY = 0 |
|  |

|  |  |
| --- | --- |
| **Method Name** | InputReader::InputReader |

|  |  |
| --- | --- |
| **Parameters** | const InputReader &ir |

|  |  |  |
| --- | --- | --- |
| results = ir.results; | | |
| fileName = ir.fileName; | | |
| **Method Name** | | InputReader::handleInput |

|  |  |
| --- | --- |
| **Parameters** |  |

|  |
| --- |
| getline(cin, fileName); |

|  |  |
| --- | --- |
| **Method Name** | InputReader::openFile |

|  |  |
| --- | --- |
| **Parameters** |  |

|  |
| --- |
| file.open(fileName) |
|  |
| if (file.fail() || file.eof()) |
| file.close() |
| return false |
| End if |
| Return true |

|  |  |
| --- | --- |
| **Method Name** | getDistributionGamma |

|  |  |
| --- | --- |
| **Parameters** | double num |

|  |  |  |
| --- | --- | --- |
| If (num == 1) | | |
| return 1 | | |
| Enf if | | |
| If (num == 0.5) | | |
| return 1.77245385091 | | |
| End if | | |
| return (num - 1) \* getDistributionGamma(num - 1) | | |
| **Method Name** | getDistributionT |

|  |  |
| --- | --- |
| **Parameters** | double dof, double x |

|  |
| --- |
| part1 = (1 + x^ 2) / dof))^ (-(dof + 1) / 2)); |
| part2 = getDistributionGamma((dof + 1) / 2) / (dof \* pi^(1/2) \* getDistributionGamma (dof / 2)) |
| Return part1\*part2; |

|  |  |
| --- | --- |
| **Method Name** | calculateIteration() |

|  |  |
| --- | --- |
| **Parameters** | double x, double dof, double num\_seg |

|  |
| --- |
| width = x / num\_seg |
| p = 0 |
| it\_num\_seg = 0 |
| it\_width = 0 |
| it\_constant = width / 3 |
| width = x / num\_seg |
|  |
|  |
| while (it\_num\_seg <= num\_seg) |
| { |
| If (it\_num\_seg == num\_seg) |
| Then p += it\_constant \* distrubtionT(it\_width, dof) |
| Else if (it\_num\_seg == 0) |
| Then p += it\_constant \* distrubtionT(0, dof) |
| Else if (it\_num\_seg % 2 == 0) |
| p += 2 \* it\_constant \* distrubtionT(it\_width, dof) |
| Else p += 4 \* it\_constant \* distrubtionT(it\_width, dof) |
| End IF |
| it\_width += width |
| ++it\_num\_seg |
| } |
| Return p |

|  |  |
| --- | --- |
| **Method Name** | InputReader::storeValues() |

|  |  |
| --- | --- |
| **Parameters** |  |

|  |
| --- |
| double x, y; |
| string line; |
| getline(file, line); |
|  |
| if (line.length() != 0) |
| xK = stoi(line); |
|  |
| while (!file.eof()) |
| { |
| getline(file, line); |
| x = stod(line.substr(0, line.find(","))); |
| y = stod(line.substr(line.find(",") + 1)); |
| if (x != 0 && y != 0) |
| data.push\_back(make\_pair(x, y)); |
| } |
|  |
| N = (int)data.size(); |
|  |
| if (N != 0) |
| { |
| for (int i = 0; i < N; i++) |
| { |
| dSumXY += data[i].first \* data[i].second; |
| dSumX += data[i].first; |
| dSumY += data[i].second; |
| dSumY2 += pow(data[i].second, 2); |
| dSumX2 += pow(data[i].first, 2); |
| } |
|  |
| dAvgX = dSumX / ((double)N); |
| dAvgY = dSumY / ((double)N); |
|  |
| calculateValues(); |
| } |
| double x, y; |
| string line; |
| getline(file, line); |
|  |
| if (line.length() != 0) |
| xK = stoi(line); |
|  |

|  |  |
| --- | --- |
| **Method Name** | InputReader::calculateValues() |

|  |  |
| --- | --- |
| **Parameters** |  |

|  |
| --- |
| results[0] = N; |
| results[1] = xK; |
| results[2] = (double)((dSumXY - N \* dAvgY \* dAvgX) / (dSumX2 - N \* pow(dAvgX, 2))); |
| results[3] = (double)(dAvgY - results[2] \* dAvgX); |
| results[4] = ((double)N \* (dSumXY)-dSumX \* dSumY) / sqrt((double)(N \* (dSumX2)-pow(dSumX, 2)) \* ((double)N \* (dSumY2)-pow(dSumY, 2))); |
| results[5] = pow(results[4], 2); |
| results[6] = results[3] + results[2] \* xK; |
| results[7] = 1- 2(calculateIteration((abs(results[4])\*sqrt(N-2))/(sqrt(1-results[5])), N-2, num\_seg)) |
|  |
|  |
| //Starts calculating the range |
| Sum\_Sigma = 0, sum\_Range (0), dSigma(0) |
| Sum\_ |
| for (i = 0; i < N; i++) |
| { |
| sum\_Sigma += pow(data[i].second- results[3]-results[2]\*data[i].first,2) |
| sum\_Range += pow(data[i].first - dAvgX,2) |
| } |
| dSigma = sqrt((1/(N-2))\* sum\_Sigma) |
|  |
| bool bRight = false |
| x = 1.0 |
| delta = x / 2.0 |
| p1 = calculateIteration(x, dof, num\_seg) |
|  |
| If (p1 < 0.35) |
| x += delta |
| bRight = true |
| **End IF** |
| Else |
| x -= delta |
| **End Else** |
| p2 = calculateIteration(x, dof, num\_seg) |
| delta /= 2; |
| while (abs(p1 - p2) >= maxError) |
| if (p1 < 0.35) |
| x += delta; |
| If (!bRight) |
| delta /= 2; |
| **End if** |
| bRight = true |
| **End if** |
| Else |
| x -= delta |
| if (bRight){ |
| delta /= 2} |
| bRight = false |
| **End Else** |
| p2 = p1; |
| p1 = calculateIteration(x, dof, 10000); |
|  |
| **End While** |
| results[8] = (x\*sigma\*sqrt( 1.0 + (double)1.0/N + ((pow(xK-dAvgX,2))/sum2))) |
| results[9] = results[6]+results[8] |
| results[10] = results[6]-results[8] <= 0 ? 0: results[6]-results[8] |