Pseudocode — Binary Data Analyzers Program

This document contains structured pseudocode that mirrors the provided C++ program. It includes high-level overview, utilities, classes, and main flow.

## High-Level Overview

PROGRAM AnalyzeBinaryIntData  
 INPUT: binary file "data.bin" containing: [int length][length ints]  
 IF file missing → create it with 200 random ints in [0, 999]  
 LOAD values into memory  
 RUN four analyzers:  
 - Statistics (min, max, mean, median, mode)  
 - Duplicates (count equal adjacent after sort)  
 - Missing (gaps between consecutive values after sort)  
 - Search (100 random probes using binary search)  
 PRINT results  
END PROGRAM

## Utilities: selection\_sort(values, size)

FUNCTION selection\_sort(values, size)  
 FOR i FROM 0 TO size-2  
 minIndex ← i  
 FOR j FROM i+1 TO size-1  
 IF values[j] < values[minIndex] THEN  
 minIndex ← j  
 ENDIF  
 ENDFOR  
 SWAP values[i] WITH values[minIndex]  
 ENDFOR  
END FUNCTION

## Utilities: binary\_search\_recursive(values, key, start, end)

FUNCTION binary\_search\_recursive(values, key, start, end) RETURNS BOOLEAN  
 IF start > end THEN  
 RETURN FALSE  
 ENDIF  
  
 mid ← FLOOR((start + end) / 2)  
  
 IF values[mid] == key THEN  
 RETURN TRUE  
 ELSE IF key < values[mid] THEN  
 RETURN binary\_search\_recursive(values, key, start, mid - 1)  
 ELSE  
 RETURN binary\_search\_recursive(values, key, mid + 1, end)  
 ENDIF  
END FUNCTION

## Utilities: binary\_search(values, key, size)

FUNCTION binary\_search(values, key, size) RETURNS BOOLEAN  
 RETURN binary\_search\_recursive(values, key, 0, size - 1)  
END FUNCTION

## Utilities: createBinaryFile(filename)

FUNCTION createBinaryFile(filename)  
 SET random seed (current time)  
 size ← 200  
 ALLOCATE array data[size]  
 FOR i FROM 0 TO size-1  
 data[i] ← RANDOM\_INT(0..999)  
 ENDFOR  
 CALL writeBinary(filename, data, size)  
 DEALLOCATE data  
END FUNCTION

## Utilities: writeBinary(filename, values, length)

FUNCTION writeBinary(filename, values, length)  
 OPEN filename FOR binary write  
 WRITE length (int)  
 WRITE 'length' integers from values  
 CLOSE file  
END FUNCTION

## Class: BinaryReader

CLASS BinaryReader  
 PRIVATE:  
 values : int\*  
 size : int  
  
 CONSTRUCTOR(filename)  
 values ← NULL  
 size ← 0  
 CALL readValues(filename)  
 END CONSTRUCTOR  
  
 DESTRUCTOR  
 FREE values  
 END DESTRUCTOR  
  
 METHOD getValues() RETURNS int\* → RETURN values  
 METHOD getSize() RETURNS int → RETURN size  
  
 METHOD readValues(filename)  
 TRY OPEN filename FOR binary read  
 IF open fails THEN  
 PRINT "Binary file not found, creating new file..."  
 CALL createBinaryFile(filename)  
 REOPEN filename FOR binary read  
 ENDIF  
  
 READ size (int)  
 IF size ≤ 0 THEN  
 PRINT "Invalid file data."  
 RETURN  
 ENDIF  
  
 ALLOCATE values[size]  
 READ size integers INTO values  
 CLOSE file  
 END METHOD  
END CLASS

## Base Class: Analyzer

CLASS Analyzer (abstract)  
 PROTECTED:  
 values : int\* // private copy  
 size : int  
  
 CONSTRUCTOR(v, s)  
 size ← s  
 ALLOCATE values[size]  
 FOR i FROM 0 TO size-1  
 values[i] ← v[i] // deep copy  
 ENDFOR  
 END CONSTRUCTOR  
  
 DESTRUCTOR  
 FREE values  
 END DESTRUCTOR  
  
 METHOD cloneValues(src, n) RETURNS int\*  
 ALLOCATE copy[n]  
 FOR i FROM 0 TO n-1  
 copy[i] ← src[i]  
 ENDFOR  
 RETURN copy  
 END METHOD  
  
 ABSTRACT METHOD analyze() RETURNS STRING  
END CLASS

## DuplicatesAnalyser

CLASS DuplicatesAnalyser EXTENDS Analyzer  
 CONSTRUCTOR(v, s) → CALL base(v, s)  
  
 METHOD analyze() RETURNS STRING  
 CALL selection\_sort(values, size)  
 dupCount ← 0  
 FOR i FROM 1 TO size-1  
 IF values[i] == values[i-1] THEN  
 dupCount ← dupCount + 1  
 ENDIF  
 ENDFOR  
 RETURN "Duplicate count: " + STRING(dupCount)  
 END METHOD  
END CLASS

## MissingAnalyser

CLASS MissingAnalyser EXTENDS Analyzer  
 CONSTRUCTOR(v, s) → CALL base(v, s)  
  
 METHOD analyze() RETURNS STRING  
 CALL selection\_sort(values, size)  
 missing ← 0  
 FOR i FROM 1 TO size-1  
 gap ← values[i] - values[i-1]  
 IF gap > 1 THEN  
 missing ← missing + (gap - 1)  
 ENDIF  
 ENDFOR  
 RETURN "Missing numbers between min and max: " + STRING(missing)  
 END METHOD  
END CLASS

## SearchAnalyser

CLASS SearchAnalyser EXTENDS Analyzer  
 CONSTRUCTOR(v, s)  
 CALL base(v, s)  
 CALL selection\_sort(values, size)  
 END CONSTRUCTOR  
  
 METHOD analyze() RETURNS STRING  
 SET random seed (current time)  
 foundCount ← 0  
 REPEAT 100 TIMES  
 randomValue ← RANDOM\_INT(0..999)  
 IF binary\_search(values, randomValue, size) THEN  
 foundCount ← foundCount + 1  
 ENDIF  
 END REPEAT  
 RETURN "Random search: " + STRING(foundCount) + " values found in dataset."  
 END METHOD  
END CLASS

## StatisticsAnalyser

CLASS StatisticsAnalyser EXTENDS Analyzer  
 CONSTRUCTOR(v, s) → CALL base(v, s)  
  
 METHOD analyze() RETURNS STRING  
 CALL selection\_sort(values, size)  
  
 minVal ← values[0]  
 maxVal ← values[size - 1]  
  
 // Mean  
 sum ← 0 (64-bit)  
 FOR i FROM 0 TO size-1  
 sum ← sum + values[i]  
 ENDFOR  
 mean ← sum / size (as double)  
  
 // Median  
 IF size is even THEN  
 median ← (values[size/2 - 1] + values[size/2]) / 2.0  
 ELSE  
 median ← values[size/2]  
 ENDIF  
  
 // Mode (first maximal run in sorted array)  
 mode ← values[0]  
 modeCount ← 1  
 currentCount ← 1  
 FOR i FROM 1 TO size-1  
 IF values[i] == values[i-1] THEN  
 currentCount ← currentCount + 1  
 ELSE  
 currentCount ← 1  
 ENDIF  
 IF currentCount > modeCount THEN  
 modeCount ← currentCount  
 mode ← values[i]  
 ENDIF  
 ENDFOR  
  
 RETURN CONCAT(  
 "Min: ", minVal,  
 ", Max: ", maxVal,  
 ", Mean: ", mean,  
 ", Median: ", median,  
 ", Mode: ", mode,  
 " (count=", modeCount, ")"  
 )  
 END METHOD  
END CLASS

## Main Program Flow

FUNCTION main() RETURNS INT  
 filename ← "data.bin"  
  
 reader ← NEW BinaryReader(filename)  
 data ← reader.getValues()  
 size ← reader.getSize()  
  
 IF size ≤ 0 THEN  
 PRINT "No valid data to analyze."  
 RETURN 1  
 ENDIF  
  
 PRINT "=== Signature Assignment Final Output ==="  
 PRINT BLANK LINE  
  
 stats ← NEW StatisticsAnalyser(data, size)  
 PRINT stats.analyze()  
 PRINT BLANK LINE  
  
 dup ← NEW DuplicatesAnalyser(data, size)  
 PRINT dup.analyze()  
  
 miss ← NEW MissingAnalyser(data, size)  
 PRINT miss.analyze()  
  
 search ← NEW SearchAnalyser(data, size)  
 PRINT search.analyze()  
  
 PRINT BLANK LINE  
 PRINT "Program completed successfully."  
 RETURN 0  
END FUNCTION