# CSC252 Programming in C++

## How to submit your Assignment

After filling all the parts in this file, please follow the following steps.

1. Add your name and ID to the first page.
2. Save the file in the original format (Docx or Doc). **Do NOT** convert to other file formats e.g. PDF, ZIP, RAR, ….
3. Rename the file as

*YOUR* ***First*** *Name - YOUR* ***Last*** *Name- YOUR student ID-* CSC252*.docx*

**Example:** John – Smith - 234566435 - CSC252.docx

1. Upload the file and submit it (only using Blackboard)

# Problem: Design and Implement the Signature Assignment (200 points)

Write a program addresses all of the functionality of the Signature Assignment.

Design, edit, run, test, and debug your program. Enter the completed C++ code here:

|  |
| --- |
| C++ code for the lab project: |
| //Vincent Hernandez, Clint Woods, Jeremy Sherman  using namespace std;  #include <iostream>  #include <fstream>  #include <random>  #include <string>  #include <algorithm>  const int ARRAY\_SIZE = 1000;  class BinaryReader {  int\* arrInt; //A pointer to hold integer array from file  int arrSize;  void readValues(const string& filename) {  ifstream readFile(filename, ios::binary);  if (!readFile) {  cerr << "Error: could not open file for reading.\n";  arrInt = nullptr;  arrSize = 0;  return;  }  readFile.read(reinterpret\_cast<char\*>(&arrSize), sizeof(arrSize)); //gets size of the array  arrInt = new int[arrSize]; //Dynamically allocates memory for the array  readFile.read(reinterpret\_cast<char\*>(arrInt), sizeof(int) \* arrSize); //Reads the actual array of integers into arrInt  }  public:  BinaryReader(const string& filename) {  readValues(filename); //immediatly reads data from file  }  ~BinaryReader() {  delete[] arrInt; // Deletes the data read from the file  arrInt = nullptr;  }  //Creates an array of 1000 integers (0-999). Saves it to binary.dat. Then deallocates the temp array. USES writeBinary()  static void createBinaryFile(string filename) {  int\* arr = new int[ARRAY\_SIZE];  srand(static\_cast<unsigned int>(time(NULL)));  for (int i = 0; i < ARRAY\_SIZE; i++) arr[i] = rand() % ARRAY\_SIZE;  writeBinary(arr, ARRAY\_SIZE, filename);  delete[] arr;  }  //Opens file, writes size and array to file. USED IN createBinaryFile() FUNCTION TO WRITE RANDOM INTO FILE  static void writeBinary(int\* values, int length, string& path) {  ofstream File(path, ios::binary);  if (!File) {  cerr << "Error: Cannot open output file: " << path << "\n";  return;  }  File.write(reinterpret\_cast<char\*>(&length), sizeof(length));  File.write(reinterpret\_cast<char\*>(values), sizeof(int) \* length);  File.close();  }  int\* getvalues() {  return arrInt;  }  int getSize() {  return arrSize;  }  //bonus method for testing  void printData() {  cout << "The Rest of the data is below the array" << endl;  cout << "The size of the array is " << arrSize << endl;  cout << "------------------------------------------------------------------------- " << endl;  for (int i = 0; i < arrSize; i++) {  cout << arrInt[i] << " ";  }  cout << "\n---------------------------------------------------------------------- " << endl;    }  };  class Analyzer {  private: //Only accessible by Analyzer  //Copys the array and returns a Copys  int\* cloneValues(int\* original, int s) {  int\* copy = new int[s];  for (int i = 0; i < s; ++i)  copy[i] = original[i];  return copy;  }  protected: //Accessible by all derived classes  int\* arr;  int size;  public:  Analyzer(int\* a, int s) : size(s) {  // 1. Allocate space for the copy  arr = new int[size];  // 2. Copy the values  for (int i = 0; i < s; ++i)  arr[i] = a[i];  }  ~Analyzer() {  delete[] arr;  arr = nullptr;  }  virtual string analyze() = 0;  };  class SearchAnalyzer : public Analyzer {  public:  SearchAnalyzer(int\* a, int s) : Analyzer(a, s) {  selection\_sort(this->arr, this->size);  }  string analyze() override {  int random[100];  int count = 0;    for (int i = 0; i < 100; i++) random[i] = rand() % 1000;  for (int i = 0; i < 100; i++) {  if (binary\_search(arr, random[i], size)) { count++; }  }  string results = "There were " + to\_string(count) + " out of 100 random values found";  return results;  }  bool binary\_search\_recursive(int\* arr, int key, int start, int end) {  if (end < start) {  return false;  }  int mid = (start + end) / 2;  if (arr[mid] == key) {  return true;  }  if (arr[mid] < key)  return binary\_search\_recursive(arr, key, mid + 1, end);  else  return binary\_search\_recursive(arr, key, start, mid - 1);  }  bool binary\_search(int\* arr, int key, int size) {  return binary\_search\_recursive(arr, key, 0, size-1);  }  static void selection\_sort(int\* arr, int size) {  // Outer loop iterates through all elements except the last one.  for (int i = 0; i < size - 1; ++i) {  int currentMin = i;  // Inner loop finds the minimum element in the \*unsorted\* portion  for (int j = i + 1; j < size; ++j) {  // Check if the current element arr[j] is smaller than the current minimum  if (arr[j] < arr[currentMin]) {  currentMin = j;  }  }  // If the minimum element found is not the one we started with, swap them.  if (currentMin != i) {  std::swap(arr[i], arr[currentMin]);  }  }  }  };  class StatisticsAnalyzer : public Analyzer {  public:  StatisticsAnalyzer(int\* arr, int size) : Analyzer(arr, size) {    }  string analyze() override {  int min = 0;  int max = 0;  int total = 0;  int median = 0;  int mode = 0;  int counter = 0;  int offset = 1;  double sum = 0;      SearchAnalyzer::selection\_sort(arr, size);    //Gets Sum of all values  for (int i = 0; i < size; ++i) {  sum += arr[i];  }    //gets min, max, mean  double mean = sum / size;  min = arr[0];  max = arr[size - 1];  if (size%2 == 1){  median = arr[size/2];  }  else {    median = (arr[size / 2] + arr[(size / 2) + 1])/2;  }    //LOGIC FOR MODE  for (int i = 0; i < size; ++i) {  if (arr[i] == arr[i + offset]) {  counter++;  offset++;  }  else {  if (total < counter) {  total = counter;  mode = arr[i];  }  counter = 0;  offset = 1;  }  }    string results = "The mean of the array is: " + to\_string(mean) + "\nThe min is: " + to\_string(min) + "\nThe max is: " + to\_string(max) + "\nThe median is: " + to\_string(median) + "\nThe mode is: " + to\_string(mode);  return results;  }  };  class DuplicatesAnalyzer : public Analyzer  {  public:  DuplicatesAnalyzer(int\* a, int s) : Analyzer(a, s) {}  string analyze() override  {  if (size == 0) return "The total count of duplicates: 0";  // Group the identical values  std::sort(arr, arr + size);  int NumberOfDuplicates = 0;  // Count each value in the array that appears more than once  for (int index = 1; index < size; )  {  if (arr[index] == arr[index - 1])  {  ++NumberOfDuplicates;  // Skip the remaining duplicate group to count only once  int val = arr[index];  while (index < size && arr[index] == val)  {  ++index;  }  }  else  {  ++index;  }  }  return "Total count of duplicates: " + std::to\_string(NumberOfDuplicates);  }  };  class MissingAnalyzer : public Analyzer {  public:  MissingAnalyzer(int\* a, int s) : Analyzer(a, s) {}  string analyze() override {  int counter = 0;  int compareNum;  int NestedNum;  for (int i = 0; i < size; i++) {  compareNum = arr[i];  for (int j = i + 1; j < size; j++) {  NestedNum = arr[j];  if (compareNum == NestedNum) {  counter++;  break;  }  }  }  return "Total Missing Numbers: " + to\_string(counter);  }  };  int main() {  //Set string path  string fileName = "binary.dat";  //Creates the random intedger array and saves to a binary file  BinaryReader::createBinaryFile(fileName);  BinaryReader File(fileName);  //Creates Analyzer Objects  SearchAnalyzer SearchA(File.getvalues(), File.getSize());  DuplicatesAnalyzer DupAnalyzer(File.getvalues(), File.getSize());  MissingAnalyzer MissAnalyzer(File.getvalues(), File.getSize());  StatisticsAnalyzer StatAnalyzer(File.getvalues(), File.getSize());    //Prints Arrays before and after sort  SearchA.selection\_sort(File.getvalues(), File.getSize());  File.printData();  //Gets the min max and mean  cout << StatAnalyzer.analyze() << endl;  //Get Median, Mode      //Gets Dublpicates  cout << DupAnalyzer.analyze() << endl;  //Gets Missing Numbers  cout << MissAnalyzer.analyze() << endl;  //Get total random values  cout << SearchA.analyze() << endl;  return 0;  } |

Run the code, take a screenshot of the results, and insert the screenshot here:

|  |
| --- |
| Screenshot of the results: |
|  |