1. **Problem Statement**

Program reads a file of user’s choice and checks if the file is empty or does not exist. The program reads the file until the end storing data in an array.

1. **Requirements**
   1. **Assumptions**
      1. The user types in an appropriate file name
   2. **Specifications**
      1. Add method *readIntFile* to read the whole file and store data in an array
      2. Add method *printFileValues* to print data in the array to the screen
      3. Make an array to store values in
      4. Add assertions where possible to check for errors, especially when there may be no values in the array
2. **Decomposition Diagram**

|  |  |  |
| --- | --- | --- |
| **Main** | | |
| **Input** | **Process** | **Output** |
| Data from the file | Check if the file is empty |  |
|  | Store values from the data file to the array | Print data is array is not empty |
| File name | Checks if the file name exists | If file empty, outputs nothing inside file.  If no file exists, outputs no such file exists |

1. **Test Strategy**
   1. Valid Data
   2. Invalid Data
2. **Test Plan Version 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in appropriate file name | “in.dat” |  |  |  |
| Invalid | 1 | User types in inappropriate file name | “indat” |  |  |  |
| Valid | 2 | File is empty | Empty file: “in.dat” |  |  |  |

1. **Initial Algorithm**
   1. In main function
      1. Initialize an array of 20 integers
      2. Prompt the user to give file to be opened and stored in an array
         1. Check if the file exists
            1. If file exists, check if it is empty or not
      3. Pass the file, length, and size of the array to the function *readIntFile*.
      4. Call function *printFileValues* to print values of the array
   2. In *readIntFile* function:
      1. Initialize a counter for the number of integers stored, starting from 0;
      2. Using a while loop to count until 20, store each data from the file passed in to the array passed in.
         1. Continue the loop while the file has not reached the end of file using a test x.eof();
      3. Return the counter variable.
   3. In *printFileValues* function
      1. Using the total number of data in the array actually stored, print all the data from the array on the screen in an organized manner.
         1. Use for loop and continue until the length that is passed in the function
2. **Test Plan Version 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in appropriate file name | “in.dat” | Program continues and prints out the data is file exists and has data inside |  |  |
| Invalid | 1 | User types in inappropriate file name | “indat” | Program displays error message that no such file exists |  |  |
| Valid | 2 | File is empty | Empty file: “in.dat” | Program displays that nothing is inside the file |  |  |

1. **Code**

// CIS200\_Lab3\_Program2 - Home.cpp : Defines the entry point for the console application.

//

#include <iostream>

#include <string>

#include <fstream>

#include <assert.h>

using namespace std;

void readIntFile(ifstream &x, int intArray[], int size, int &length);

void printFileValues(int intArray[], int &length);

int main()

{

string userInput;

ifstream x;

int length;

const int size = 20;

int intArray[size];

cout << "File name: " << endl;

cin >> userInput;

x.open(userInput);

assert(x.good());

readIntFile(x, intArray, size, length);

printFileValues(intArray, length);

cout << "Length: " << length << endl;

system("pause");

return 0;

}

void readIntFile(ifstream &x, int intArray[], int size, int &length)

{

length = 0;

while (!x.eof())

{

x >> intArray[length];

length++;

}

}

void printFileValues(int intArray[], int &length)

{

for (int i = 0; i < length - 1; i++)

{

cout << intArray[i] << ", ";

}

cout << intArray[length - 1] << endl;

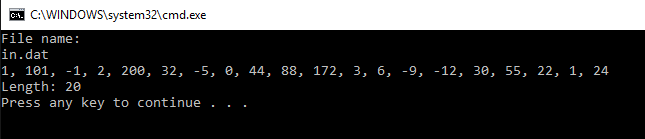
}

1. **Updated Algorithm**
   1. In main function
      1. Initialize an array of 20 integers through the variable *size*
      2. Prompt the user to give file to be opened and stored in an array
      3. Open file
         1. Check if the file exists (assert)
            1. If file exists, check if it is empty or not
      4. Pass the file, length, and size of the array to the function *readIntFile*.
      5. Call function *printFileValues* to print values of the array
      6. Print the length of total number of values stored in the array
      7. Close file
   2. In *readIntFile* function:
      1. Initialize a length for the number of integers stored, starting from 0;
      2. Using a while loop to count until end of file, store each data from the file passed in to the array passed in.
         1. Continue the loop while the file has not reached the end of file using a test x.eof();
         2. Increment length variable by 1 each time a value is added in the array
      3. Return the ~~counter variable~~ length variable
   3. In *printFileValues* function
      1. Using the total number of data in the array actually stored, print all the data from the array on the screen in an organized manner.
         1. Use for loop and continue until the length that is passed in the function
2. **Test Plan Version 3**

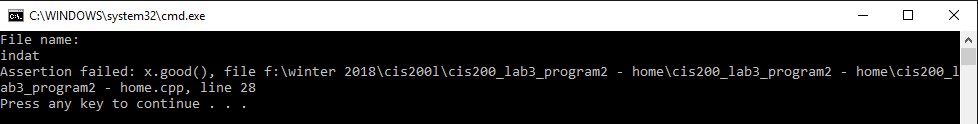
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
| Valid | 1 | User types in appropriate file name | “in.dat” | Program continues and prints out the data is file exists and has data inside | Program continues and prints out the data is file exists and has data inside | Pass |
| Invalid | 1 | User types in inappropriate file name | “indat” | Program displays error message that no such file exists | Program displays error message of assertion | Pass |
| Valid | 2 | File is empty | Empty file: “in.dat” | Program displays that nothing is inside the file | -858993460 | Fail |

1. **Screenshots**

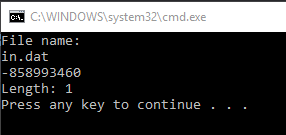
Valid Test Case 1:



Invalid Test Case 1:



Valid Test Case 2:



1. **Status**

Program works perfectly with assumptions in mind.