# Assessment Task 2 – Debugging Problems

And also re-writing like most of the code I guess.

## Issue 1: Getting the name to display correctly

The name is displaying with random characters added in.

A picture containing text, receipt, algebra, design

Description automatically generated A picture containing receipt, text, algebra, design

Description automatically generated

There are two solutions: one is dumb, the other is much smarter  
Dumb Solution: while (r->name.size() != nameSize) { r->name.pop\_back(); }   
This solution is a workaround as it pops back the characters until the length is equal to the nameSize int. it does cause it to display correctly, but is still a hacky workaround.

A close-up of a logo

Description automatically generated with low confidence A picture containing text, white, design, algebra

Description automatically generated

The better solution is to use a Null Terminator to denote the end of the string.  
In the creation of the char\*:

char\* name = new char[nameSize + 1];   
name[nameSize] = '\0';

in declaring the char\*, we add 1 to the name size. This creates an extra byte to the string which gets picked up by the next line, terminating the string at the end, so it displays correctly.

A close-up of a logo

Description automatically generated with low confidence A picture containing text, tool, design, algebra

Description automatically generated

## Issue 2: Crashing program

The program crashes when trying to go past the last index.

A screenshot of a computer error

Description automatically generated with medium confidence 🡨 Error received  
“Vector subscript out of range” effectively means the program is trying to access data that doesn’t exist, therefore we need to find where this crash is originating from.  
Because this is a issue when we press the right arrow to go to the next index, we’ll start there.  
A screen shot of a computer program

Description automatically generated with low confidence

Using a breakpoint on the inside of the if loop checking for an invalid index, we can place a breakpoint here and use F10 to forward the program by one line.  
Using this method, the program crashes upon line 72:   
currentRecord = data.GetRecord(currentRecordIdx);  
CTRL + clicking GetRecord leads us to the function in question, which is just returning the index sent to it via the records vector;  
A picture containing text, screenshot, font, line

Description automatically generated  
This code is working without issue for the moment, so our problem lies with the if method trying to catch bad values on line 70 in main.cpp.  
currentRecordIdx = data.GetRecordCount();   
this code is returning supposed to keep you on the final record, but breaks because it tries sending  
you to an invalid record due to how Arrays start at 0, not 1. So the final record is on index 4.

This is a simple fix, as we just need to subtract 1 from the value returned by GetRecordCount().  
  
this prevents the program from crashing.

## Issue 3: Random Access Creation and Rewriting

Holding the entire data file in memory is not great, so we need to make a random access algorithm that grabs the data when it needs it and holds that in memory.

To do this, we have to remove some of the old code that is holding all that data in memory, primary one being the records vector. After this, we can remove other things we don’t need such as the Save and AddRecord functions.

A picture containing text, font, screenshot

Description automatically generated

After this, we need information on how large the variables are. We can do this easily by using breakpoints.

A screenshot of a computer program

Description automatically generated with medium confidence

On the bottom of the screen, you can select auto’s / watch 1 to watch certain variables, or you can hover over them while the program is paused to see the values. In our case, we want to watch the size of each value from the record so we can calculate the size of each record.

We’ll modify the Load function to run through the file and calculate the size of each record, where it is located before storing that in a vector of ints, storing the location in bytes of each Record so we can access them quickly, which is more efficient than the vector of Record\* from before that was holding everything in memory.

Because the first int within the file shows how many records we have in the file, we need to skip past those bytes. This is done by setting the first entry in the vector to be the size of an int, so it skips past that first int. After this, we can loop through and set the location of each of the records.

int recordSize = imageSize + nameSize + ageSize;

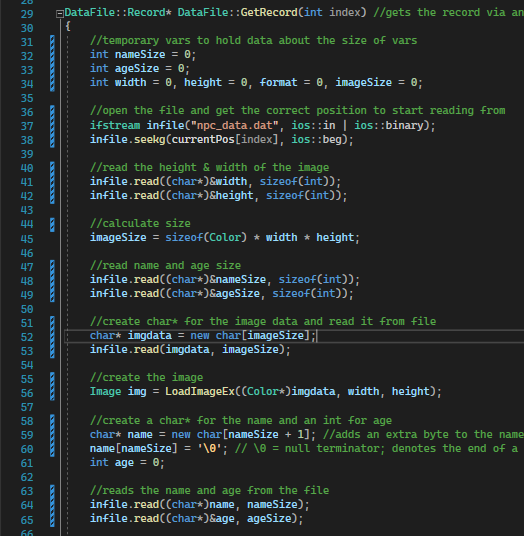
int location = infile.tellg();

infile.seekg(recordSize, infile.cur);

currentPos[j] = location + recordSize;

this code is within a for loop where j starts at 1, so it never touches the first record. This loops through using tellg to set the position to the location that was found by seekg from the previous loop through, which incremented by the size of the record.

After this, we can modify the GetRecord function to calculate which record to read and load, re-using a large portion of the code from the Load function previously with some, using seekg to set the position we want to read from, then creating a record to hold in memory and load our data from.



We also need to delete the data that is just existing.

A screen shot of a computer code

Description automatically generated with low confidence