Brooke Slampak

CS-300

5/18/2025

**Module Two Assignment Reflection**

The code is organized into a set of functions and a struct, all working together to read, process, sort, and display bid data from a CSV file. It starts with a function called strToDouble, which takes string data from the CSV file and converts it into numeric values that the program can work with. At the center of the program is a Bid struct, which holds the details of each bid. These bid records are stored in a vector, which can be sorted and displayed.

To show the contents of the vector, the displayBid function sends the bid information to the console. There’s also a getBid function in the code, but it’s not actually used anywhere in the current version. The loadBids function reads a CSV file from a provided file path, or a default one if none is given, pulls out key pieces of information, and creates Bid objects from them. These are then added to the vector for later use.

When it comes to sorting the bids, there are two options, which are quick sort and selection sort. The partition function helps with quick sort by splitting the vector into two parts and making sure elements are in the right order relative to a pivot. The quickSort function then uses this partitioning logic recursively to sort the entire list. On the other hand, selectionSort works by scanning the list with two loops, looking for out-of-place elements, and swapping them into the right position.

Finally, the main function brings everything together. It presents a simple switch menu that allows the user to load bid data, view it, sort it using either algorithm, or exit the program. It also keeps track of how long each sorting method takes, using the time.h library, so users can see which one performs better. Overall, the code is a clean and functional example of working with structured data, applying sorting algorithms, and measuring their performance.

One of the biggest challenges I faced during this assignment was that I hadn’t used Visual Studio or written C++ code in nearly a year. Coming back to it after such a long break made it difficult to remember syntax rules, standard library functions, and even basic project setup steps within Visual Studio. Simple tasks took longer than expected as I had to re-familiarize myself with the language and tools. I had to spend extra time reviewing syntax and watching Visual Studio tutorial videos before I could complete the assignment.

**Pseudocode**

**Function: main()**

- Read command-line arguments

- If provided, store the argument as the CSV file path

- If not, use a default CSV file path

- Loop while user choice is not '9':

- Display menu options

- Get and validate user input

- If input is not 1–4 or 9, display an error

- If choice == '1': // Load Bids

- Start timer

- Call loadBids() to read CSV and store data in 'bids' vector

- Display number of records loaded

- Stop timer and output elapsed time

- If choice == '2': // Display Bids

- Loop through all bids in the vector

- Call displayBid() for each

- If choice == '3': // Selection Sort

- Start timer

- Call selectionSort(bids)

- Stop timer and output elapsed time

- If choice == '4': // Quick Sort

- Start timer

- Call quickSort(bids, 0, bids.size() - 1)

- Stop timer and output elapsed time

- If choice == '9':

- Exit the application

- Output "Goodbye"

**End**

**Function: selectionSort(bids)**

- For i from 0 to bids.size() - 2:

- Set indexSmallest = i

- For j from i + 1 to bids.size() - 1:

- If bids[j].title < bids[indexSmallest].title:

- Set indexSmallest = j

- Swap bids[i] with bids[indexSmallest]

**End**

**Function: quickSort(bids, begin, end)**

- If begin >= end:

- Return

- Set lowEndIndex = partition(bids, begin, end)

- Recursively call:

- quickSort(bids, begin, lowIndex)

- quickSort(bids, lowIndex + 1, end)

**End**

**Function: partition(bids, begin, end)**

- Set midPoint = begin + (end - begin) / 2

- Set pivot = bids[midPoint].title

- Declare tempSwap

- Set done = false

- While not done:

- While bids[begin].title < pivot:

- Increment begin

- While pivot < bids[end].title:

- Decrement end

- If begin >= end:

- Set done = true

- Else:

- Set tempSwap = bids[begin]

- Set bids[begin] = bids[end]

- Set bids[end] = tempSwap

- Increment begin

- Decrement end

- Return end

**End**