

# Major Project: Public Tool Library System

CAB301 Algorithms and Complexity - Major Project

Student ID: N10489045

Student Name: Sophia Politylo

Due: 24/05/21

## Table of Contents

Introduction: .....	4
Assumptions:.....	4
Design of Classes:.....	5
Binary Search tree Class (BSTree) Implementing iMemberCollection.....	5
Member Class Implementing iMember .....	5
ToolCollection Class Implementation of the iToolCollection interface .....	6
Tool Class implementation of the iTool interface.....	6
LibraryInterface Class Implementation of the ToolLibrarySystem Interface .....	6
Design and Analysis of Algorithms:.....	7
Member Collection .....	7
Storage and sorting.....	7
Pseudocode and Analysis.....	7
Tool Collection: .....	11
Storage and Sorting:.....	11
Pseudocode and Analysis.....	11
Top 3 Tools Sorting Algorithm .....	18
Software Testing: .....	20
Renting cases: .....	20
A User is already borrowing 3 tools:.....	20
A User want to borrow 2 of the same tool: .....	21
Admin wants to remove a tool from the system currently being rented:.....	22
Admin wants to delete a user that is renting a tool: .....	23
A User want to rent a tool that is unavailable: .....	24
A User wants to return a tool when they have none rented:.....	25
A User want to return a rented tool: .....	26
A User wants to rent a new tool: .....	27
Updating fields: .....	28
Admin want to add a new tool: .....	28
Admin want to add another tool of the same type: .....	29
Admin only want to remove a few tools from a tool class: .....	30
Admin wants to remove all the tools from a class: .....	31
Admin wants to add a new user: .....	32
Admin want to remove a user from the system: .....	33
Search Functions: .....	34

Searching for a user's phone number that does not exist:.....	34
User viewing their currently rented tools when they have non out .....	35
Descending times borrowed merge sort on a Tool Collection .....	36
User checking the Top rented tools when none or only 1or 2 have been rented out: .....	37
User checking the Top rented Tools .....	38
Updating top3 rented tools .....	39
Normal operation of the program (menu screens) .....	40

## Introduction:

The following documentation has been created to help explain and demonstrate the functionality of the created Tool Library System C# program that has been created as per the clients brief. This document will explain and explore the created algorithm and storage structures that have been used within this program, explaining why each sorting algorithm and storage type was chosen. Along with this the functionality of the program will also be displayed and analysed through test cases and explanation of the operation of the tool library program.

## Assumptions:

There have been a few assumptions made to help in the completion of this task, that will influence the final product.

The first assumption made was that no tools could be deleted that were currently being rented out. This means that the max tools that an admin can delete is the number of available tool.

The next assumption made was that a user can be deleted from the member tree, when they are still renting out tools. This has a stipulation however in that they must have returned the tools to an admin, as all tools they were currently renting will be returned. To facilitate this any user that is renting out tools at time of deletion will prompt a popup, that will ask the admin if they have returned the tools, with the user being deleted if the admin answers yes.

The next assumption that was made was the way that the Tool Collection arrays will be sorted. With the chosen method sorting method being number of borrowing in descending order. This was chosen as it provides the user's of the program an easy way to see the most popular tool of each tool type. Allowing the admins to easily see what tools they should get more of. While also allowing the normal members to see what tools are the most popular and thus hopefully the most reliable and best tools in each tool type.

Another assumption that was made was in regards to how tools could be selected to be displayed. As within the Display all tools of a tool type, an all selection has also been added to allow the user to view all tools within a category or the whole tool library.

The next assumption made was that when a admin is added, they will not be instantly added to the member array, with them needing to be added separately to the member tree to access the normal member functions.

## Design of Classes:

### Binary Search tree Class (BSTree) Implementing iMemberCollection

The BSTree Class is the class implementation of the iMemberCollection interface provide with it holding the functions that allows, the program to search the member collection for a user, add a new user and delete an existing user. This functions also allows the program to get an array of members constructed via an in-order traversal of the tree, organising the members in alphabetical order using their last names.

To implement the binary search A new class was created called Bsnode that stored an imember class as a node, and 2 Bsnode classes as the left and right child of that node. This class was necessary to add as it allowed the BSTree class to effectively be implemented allowing for each node of the tree to be neatly packaged together, allowing for easy manipulation of the tree. Allowing for members to be added and removed, via only changing a single node, Right or left child, via inserting a new node.

Within the BSTree class a few extra functions were added to help add functionality to class. This was mainly in the implementation of new search functions. The added search functions differ from the original one given in two ways. The first being that the input variable is now a String array and that it returns a imember instead of a Boolean. This function was deemed necessary to add as it one allowed for user's to be easily pulled out from the array, when they are need and two allows the function to take the raw input of the user's first name, last name, or phone number. This means that less pre-processing is need when feeding across the user's inputs into the console, while also saving on memory, given that a new temporary imember class does not need to be constructed to search the array.

### Member Class Implementing iMember

The member class has not been modified to much when compared to the original imember interface with all methods being implemented as they were given. One small change has been added however though with the imember interface now extending the IComparable interface. This was done as it was important to make sure that the Member Class could be effectively compared against each other for storing in the binary search tree. Given that the IComparable interface allows the use of CompareTo method, extending the Member class to include it gave an easy way to implement this method in the desired way.

The CompareTo method was set up in such a way that it took an imember object, and then compared the imember passed to it and the member it was called from, Last name and then first name if last names match. With it than returning 0 if both members are the same, -1 if the imember passed goes before the member or 1 if the member goes before the passed member.

One last Function was also added to help in the binary search tree searching function this method was called search\_string, this method would then take a string array passed to it, which it would then use to check if all the values in the string array existed as either the first or last name or the phone number of that given user. Return true if all string in the array was present in the user else it would return false.

### ToolCollection Class Implementation of the iToolCollection interface

The ToolCollection interface is the one of the methods with the largest amount of new methods add to help with managing the tool collection. Many of these functions however have been added as private helper functions that allow similar overload functions to perform the same task in a slightly different manner. An example of one such function is the updatetool private function, that allows for the handling of finding and updating a tool within the collection when it already exists. This was made a separate private function as it allowed for a cleaner add tool function, while also allowing the add tool function to terminate if the tool was updated since it returns True or False depending on if a tool's quantity was updated.

The major functions that were added to this class was the merge sort functions, with their being an added 4 in total for just the sorting functionality, however 2 of those are just overloads, with 3 of the 4 all being private methods that are called recursively from the public sort of method.

### Tool Class implementation of the iTool interface

The Tool class directly implements all the functions given to it in the iTool interface, with no new methods being added to the Tool class. This class was kept as given, since it is the simplest class given that does not interact directly with any other class meaning that all the other class handle its storing and sorting along with anything that goes beyond checking if the tool is available to rent.

### LibraryInterface Class Implementation of the ToolLibrarySystem Interface

The library interface has undergone the most changes of any of the given classes, with many new functions being added to allow it to more effectively process the information given to it by the user and display options and responses to the user. Most of these added Functions are just menu screens and helper functions for those screen, cutting down on the code need to display a readable and functioning user interface.

Examples of such methods are the IsDigitonly method, which checks that the input is only numbers when, they are need for navigating menus. The three different displaychoice methods, which are all slightly different methods that allows for a array to be passed to it to allow, a selection menu to be displayed to the user, where they are able to select a choice to preform from. The rest of the added functions full into the main menu category, in which each method, outlines a specific menu to display and the required inputs the user needs to perform, in order to use the main functionalities within the ToolLibrarySystem interface.

## Design and Analysis of Algorithms:

### Member Collection

#### Storage and sorting

The Member Collection is stored within a Binary search Tree, where the head node of the tree is the user that has half of each other user on either side of them in alphabetical order. For example if the last names were a,b,c; b would be the head of the node with the right child containing all the members above it in the alphabet and the left child holds all members with a name before it in alphabetical order. This an effective way store member within a binary search tree as it allows for faster searching within the binary search. This is since only one branch of the binary search tree will have to be explored, since it is known that the left child contains members that are before the current node in the alphabet and the right node contains members after it in the alphabet. Means that an algorithm is able trim the tree by comparing the node's member's name and the member to search for name and if it returns that the member's name comes after the node's name go down the left side of the tree, else if it comes after going down the right side of the tree.

#### Pseudocode and Analysis

For all the created methods the search functionality to find the node to delete add or return is all the same at:

$$O(\log(n))$$

This is due to the fact stated before that at worse only the max depth of the tree will ever have to be traversed. Since each node of the tree splits the new number of candidates it could possibly be down to  $n/2$ , this means that since this equation is recessive the total number of nodes it could be are trimmed at a faster and faster rate, since each time  $n$  halves. This means that the number of nodes to visit get small fast for example. Let's compare a search tree with 128 members and 64 members, with the number of times  $n$  needs to be halved to get to  $0 \approx 1$

Number of steps	64 members	128 members
1	64/2	128/2
2	32/2	64/2
3	16/2	32/2
4	8/2	16/2
5	4/2	8/2
6	2/2	4/2
7	1	2/2
8		1

Table 1 - comparing the number of times for 64 and 128 need to be divided by 2 to reach 1

From the table computed it is easy to see just how quickly halving the binary search tree after each node is drastically able to bring down the search time given that double the amount of members only increased the search time by 1 step. This trend would also hold up for 258 members, given that after the first step the number of members to search would be down back to 128 members, making the total number of steps need 9. This than means that to search a binary search tree the total number of steps is:

$$\log_2 n$$

Equation 1 – binary search tree search time

Making the search algorithm of the search algorithm  $O(\log(n))$

*Compare members method*

```
MemberCompare(m1,m2)
  If m1.Name > m2.Name
    return -1
  else if m1.Name < m2.Name
    return 1
  else if m1.Name == m2.Name
    return 0
```

*Search method*

```
BS_Search(member, node)
  If (MemberCompare(member, node.value) == -1)
    if (node.Left  $\neq$  null)
      BS_Search(member, node.Left)
  else if (MemberCompare(member, node.value) == 1)
    if (node.Right  $\neq$  null)
      BS_Search(member, node.Right)
  else if (MemberCompare(member, node.value) == 0)
    return True
  return False
```



*Delete method*

The delete search method is a little bit more complex than the normal search method, however the same basic theory applies to it with the only major difference coming from the fact that more steps need to be performed when the member is found. This means that the efficiency of this function is still  $O(\log(n))$  however the equation has now changed to include more static terms to represent the moving of node around within the array. Even though there is an added while loop within this implementation, it still does not affect the worst-case scenario as all it does is place the best case scenario back to the worse case making all cases the same for this function, given that it will only ever be computed if the node to remove is not at the bottom of the tree already or second from the bottom. Making most cases for this function the same at:

$$\log(n) + 4$$

*Equation 2 - time for binary tree delete function*

```

BS_remove(member)
    Parent ← null;
    remove ← root
    While (remove ≠ null ) AND (node ≠ remove)
        parent ← remove
        If (MemberCompare(item, remove.value) == -1)
            Remove ← parent.Left
        else
            remove ← parent.Right
        else if (MemberCompare(member, node.value) == 0)
            return node
    if remove ≠ null
        if (remove.left ≠ null) AND (remove.right ≠ null)
            if remove.Left.Left == null
                remove.item ← remove.Left.value
                remove.Left ← remove.Left.Left
            else
                child ← remove.Left
                parent ← remove
                while (child.Right ≠ null)
                    parent ← child
                    child ← parent.child

                remove.value ← parent.value
                Child.Right ← parent.Left
        else
            if (remove.left ≠ null)
                child ← remove.Left
            else
                child ← remove.Right
            if remove == root
                root ← child
            else
                if remove == parent.Left
                    parent.Left ← child
                else
                    parent.Right ← child

```

### Add method

The insert method is slightly less complex than the delete method with less cases needing to be accounted for, since it does not need to reshuffle the whole tree, due to a node being removed. This means that the equation for this function is only:

$$\log(n) + 2$$

*Equation 3 - time for binary tree add function.*

There has been no added functionality to add a new member that is the same as an existing member as there should be no overlap within the members first name and last name within the binary search tree, meaning that any member already in the tree, will break the recursion of the function, leading to them not being inserted.

```
BS_add(member,node)
    If (MemberCompare(member, node.value) == -1)
        if (node.Left == null)
            node.Left ← Node(member)
        else
            BS_add(node.Left)
    else if (MemberCompare(member, node.value) == 1)
        if (node.Right == null)
            node.right ← Node(member)
        else
            BS_add(node.Right)
```

### Tool Collection:

The tool collection class has implemented the three given functions.

### Storage and Sorting:

The tools in the Tool Collection class have been stored within an array, that gets expanded every time a new tool is added. To sort the array a merge sort algorithm was used, every time a member rented out a new tool ensuring that order was kept.

### Pseudocode and Analysis

#### *Search for tool method*

A simple full array search has been created where a simple for loop is used to iterate and check if the passed tool is in the array, with it returning true if it is found. The efficiency of this search function has the worse efficiency of  $O(n)$ , with the following equation for the worst-case scenario being:

$$c(n) = n$$

*Equation 4 - time taken to search for tool in array*

The worse case scenario for this algorithm is  $O(n)$  due to the worse case being when the tool value does not exist in the array or is the last element in the array. This then forces the search method to perform the if statement  $n$  times before finally returning a value.

```
Search(A[n], tool)
  For i ← 0 to n-1
    If A[i] == tool
      Return True
  Return False
```

*Remove tool method*

The remove tool method works via iterating over the tool array twice, with the first for loop removing the specified quantity from the desired tool and checking whether the new array needs to be shorted if the new quantity for the tool is 0. The second loop then goes back over the tool array, adding all the tools that have a quantity greater than zero into the new tool array.

The Worst efficiency for the remove function is  $O(n)$ , with the following equation:

$$c(n) = 2n + 3$$

*Equation 5 - time to remove tool from array*

The worst-case scenario for the function happens when a tool needs to be removed from the array, making the method cycle through the tool array twice, along with it needing to test every if statement within both methods.

```

Remove(A[n], tool, amount)
    n_new ← n
    i_new ← 0
    A_new
    For i ← 0 to n-1
        If A[i] == tool
            A[i].quantity ← A[i].quantity - amount
            If A[i].quantity < 1
                N_new ← N_new - 1
    If N_new < n
        A_new ← array[n_new]
        For i ← 0 to n-1
            If A[i] ≠ tool
                A_new[i_new] ← tool
                i_new++
    else
        A_new ← A[n]
    return A_new

```

This Function is the optimal solution for the given problem due to the way the array is sorted and the way C# array's work. Since the tool array is sorted via number borrowings, a binary search method can't be used, since there is no way to know where a tool will be via any midpoint methods. Makes the sequential search the best method. Two for loops are need because of how arrays work, given that they are created with a fix size, means that the first loop removing the quantity of the given tool, also checks to see if the array needs to be shorted, with the second loop then copying over the desired tools into a shorted tool array.

*Add Tool method*

The add tool method is very similar to the delete tool method, with the array in the worst case having to be iterated upon twice. This is again since the array first needs to be checked if an instance of the tool needs updated or added to the array. This means that again that the worst-case time for this algorithm is  $O(n)$ . With it again having a similar equation of:

$$c(n) = 2n + 4$$

*Equation 6 - Time taken to add tool to array*

The worst-case scenario is achieved when a tool is not in the tool array and needs to be added. This is because a full search of the array needs to be done first to make sure that the tool does not exist in the array, and then the tool array needs to be copied over into the new larger array, with a new tool being added to the end of the array after the copying.

```
Add(A[n],tool):  
    n_new ← n+1  
    A_new ← array[ n_new]  
    for i ← 0 to n-1  
        A_new[i] ← A[i]  
    A_new[n] ← tool  
    A[n] ← A_new
```

### *Merge sort tools by descending order*

The Tools within the Tool Collection array is sorted via the number of times each tool has been borrowed in descending order. To achieve this a bottom-up merge sort algorithm was implemented to sort the array every time a tool is borrowed. A merge sort algorithm was chosen due to its stability and time taken to sort. Since merge sort is a stable sorting algorithm it means that all the tools will be stored in correct order every time. Which is advantageous for the type of sorting that is needed for this tool collection as a similar sorting algorithm heap sort does not possess this quality. This means that the heap sort algorithm only sorts the head nodes, making sure they are the biggest value, meaning that the left and right child might not be sorted in the correct way. The time taken to use the merge sort algorithm is also a major advantage when compared to other sorting algorithms since it has one of the lowest worst sort and average sort time at  $n \log(n)$ . The best sorting time for the merge sort algorithm however is not the best with many sorting algorithms having a best time of  $n$ . Most of these sorting algorithms however have a fair worse best and average sorting time at  $n^2$ . With the only algorithms being better than a merge sort not being within the purview of this assignment and unit.

The Merge sort algorithm however does pose one drawback, and that is the total amount of memory that is needed, with it being  $n$ , which is by far the worst with the majority of sorting algorithms only needing the initial array to sort or 1. This drawback however is overlooked by the average sort time and the stability of the sorting algorithm, with it being the fastest sorting stable algorithm within the assignment's purview. Along with this the size used by this algorithm is not a very big concern given it is only temporary storage, the wide range of memory, available on modern machines and the relative size of the program. This means that in the context of the tool library system memory is not a big concern, due to what it will run on and the relative expected size of the final tool library system. But if the program were to be ported over to a microprocessor or the scope of the stored tools were to be increased, the memory usage would become an issue.

To implement the merge sorting for the tool collection array 3 methods were created. The first of these methods were a split method that allows for the splitting up of an array into two sub arrays. The second method is used to merge the two sub arrays together, with the third function then being called to recursively call the other two functions in a single action.

A merge sort algorithm is a  $O(n \log(n))$  algorithm due to the recursive nature of the function. Since the merge algorithm will always take  $O(n)$  and the total size of  $n$  each recursion is  $n/2$  until  $n = 1$ . This means that the total number of times that this recursion will take place is the depth needed for  $n$  to split to 1. Given that the equation for finding the max depth is:

$$d = \log(n)$$

*Equation 7 - Time taken to reach max depth of splitting an array to 1*

Using the recurrence relation general formula and basic equation for the time taken for the function can be created through considering the depth equation for number of iterations and adding an  $n$  trailing term for the merging function produces the following equation:

$$T(n) = n(\log n + 1)$$

*Equation 8 - Time taken for merge sort*

Where  $n$  was added due to the merge algorithm, where for each instance the for loop will pass over the array once. The  $\log n$  was added since it computes the max depth need to split an array down to its single elements and the place one was added to denote the processing done in calculating the midpoint of the array.

```
Split(A[n], start, length)
    A_split[length]
    for i ← 0 to length-1
        A_split[i] ← A[i+start]
    Return A_split
```

```
Merge(A1[n1], A2[n2]):
    A_merge[n1+n2]
    i ← 0
    i1 ← 0
    i2 ← 0
    while (i < n1+n2-1)
        if (i1 < n1-1) & (i2 < n2-1)
            if A1[i1] > A2[i2]
                A_merge[i] ← A1[i1]
                i1++
            else
                A_merge[i] ← A2[i2]
                i2++
        else if (i1 > n1-1)
            A_merge[i] ← A2[i2]
            i2++
        else if (i2 > n2-1)
            A_merge[i] ← A1[i1]
            i1++
        i++
```

```
MergeSort(A[n])
    If n > 1
        S1 ← 0
        L1 ← floor(n/2)
        L2 ← n-L1
        A1 ← Split(A[n], S1, L1)
        A2 ← Split(A[n], L1, L2)
        A ← Merge(A1, A2)
    return A
```

Given that this algorithm was the most complex out of the sorting algorithms created experimental data was created to ensure that the performance of the algorithm was as expected. This was done by creating the pseudo code function in C# and making them run on arrays with length 1 to 10000 with a step of 100, with the time being recorded for each array length. This produced the following graph in figure 1. Which does show a loose adherence to the  $n \log(n)$  rule, with the time taken towards the larger x axis starting to become less clustered in their time taken, starting to go up quicker than the smaller arrays. This is to be expected within a  $n \log n$  algorithm given that at smaller  $n$  these types of functions perform similarly to a  $O(n)$  function. But where these two functions start to differ are at larger  $n$  values where  $O(n \log n)$  starts to have a steeper slope.

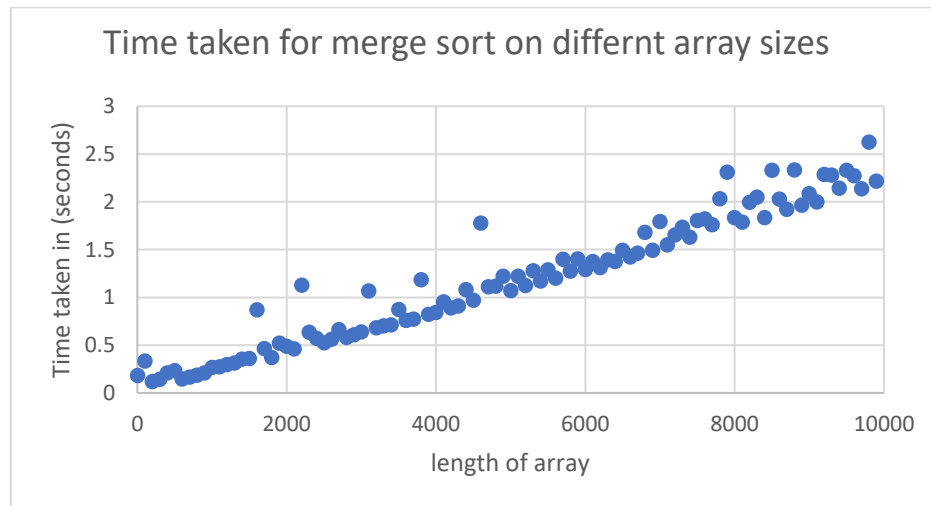


Figure 1 - experimental data to find the time taken for the created merge sort method

To ensure that this is indeed a  $O(n \log(n))$  function a quick for loop and a for loop in a for loop were tested to just have some data points to compare the merge sort algorithm against. From this the graph in figure 2, was produced which showed that the sorting algorithm outperforming the  $O(n^2)$  remarkably well.

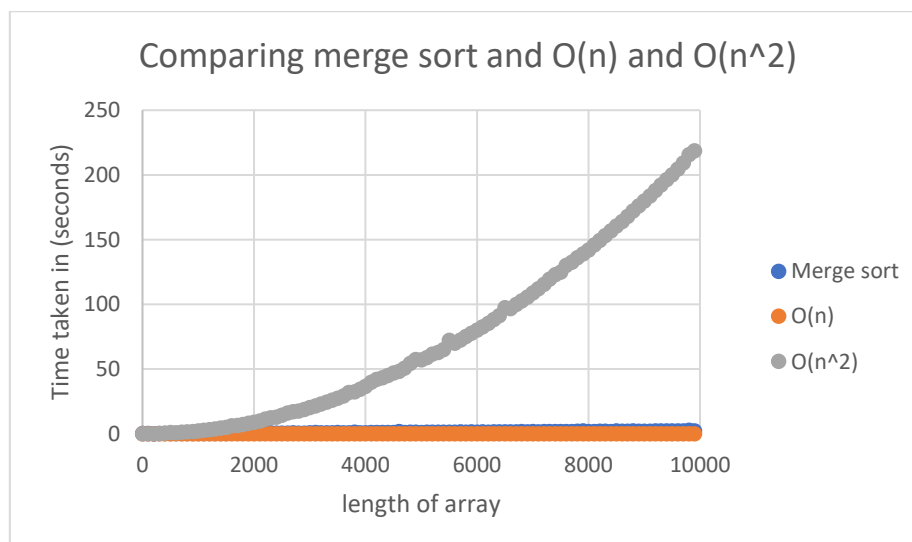


Figure 2 - Comparing merge sort to  $O(n)$  and  $O(n^2)$

This graph however does make it hard to view the  $O(n)$  comparison, to fix this a third graph was created that just shows the merge sort and  $O(n)$ , which can be seen in figure 3. From this graph it



becomes obvious that the merge sort is worse than a  $O(n)$  algorithm, with the axis still being too big for any noticeable slope to be seen on the  $O(n)$  data set. This however does confirm that the merge sort algorithm is not a  $O(n^2)$  or a  $O(n)$ , given that a  $O(n \log n)$  has a fairly constant slope, once  $n$  reaches a certain it can safely be said that the created merge sort algorithm does adhere to a  $O(n \log n)$  notation, with its slope being constant after passing  $n = 200$ , unlike the  $O(n^2)$  which has a increasing slope as  $n$  increases. Along with this the slope for the merge sort is far greater than that of the  $o(n)$  algorithm.

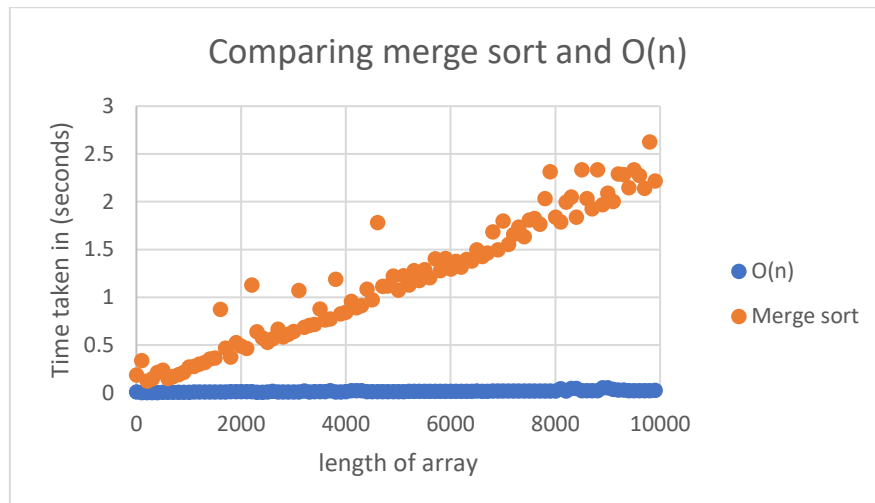


Figure 3 - Comparing merge sort to  $O(n)$

From this analysis of this array, it is dreadfully obvious that this function is not the most effective sorting method but given that the way the arrays are sorted are highly dependent to change, the merge sort algorithm is the best one for this task. This is because merge sort can be performed after insertion of a new object unlike a sequential search and add function, meaning that the created algorithm can be used when a user rents out a new tool, keeping the arrays up to date.

The merge sort was mainly chosen to allow the tools to be displayed in order of popularity as it will help add functionality to the program. Through showing the user quickly and easily the most popular tools within each tool type making it easy for them to choose the tool that is used the most and thus would most likely be the best tool for the job.

### Top 3 Tools Sorting Algorithm

To sort the top 3 tools an array has been created within the LibraryInterface class, that has the capacity to store 3 tools. To populate this array a slightly modified serial search was created. This search algorithm created might seem to be more complex than just  $O(n)$  but given that the nested for loops are just flattening the input array into the components and the arrays is not being iterated upon with a for loop any  $n^2$  or greater notations are removed given the array is looping across itself twice. The equation for the worse case however for this search algorithm is:

$$(N)3 + 1$$

Where  $N = n + m + 3$

$$3(n + m + 3) + 1$$

This equation is a bit more complex than just a normal sequential search, but it can basically be seen as the program will iterate through tool categories and then iterate through that categories tool types and then check the top 3 tools in that tool type. This is only in the worst-case scenario however as the algorithm created does always iterate through the full top3 array or top 3 tools in each tool type category, with it discarding any tool type as soon as it is known that the current tool in the tool category will not be better than what is already in the top 3. An example would be if the first tool in a tool type had 3 total borrowings and the third place tool in the top 3 had 4 total borrowing, since it knows that the top tool for that tool type has only been borrowed out 3 times there is no need to check any more variables in the top 3 array or the tool type array, allowing it to move one after, looking at the best tool in that tool type array and the worst tool in the top 3 array.

This search algorithm can also be displayed as the following equation:

$$3(n + m + 3) + 1 + n(\log n + 1)$$

This is mainly due to the fact that for this search algorithm to work as intended the merge sort in each tool type must be completed, making it an important part of this algorithm. This would then make the  $O$  notation for this search algorithm  $O(n \log n)$  which is far worse than the  $O(n)$  found before and worse than just a search function that checks all the tools in each tool type, without pre-sorting them. This part of the equation however was removed as it is not performed via the top 3 method, and is instead part of another sorting algorithm, that is performed at the time of borrowing. This means that it is separate to the time taken for this searching algorithm and thus can be removed when computing the total time taken. Since the merge sort was added in for a quality-of-life improvement for the user's in helping them find the best tool, there is no need to justify it within the context of this search function. With the only purpose it is serving in helping to minimize the minimal time taken by this search function.

```

Top3(A[n][m][y])
  Top3 ← array[3]
  for i ← 0 to n - 1
    for j ← 0 to m - 1
      l ← 0
      while l < y or l < 2
        insert ← -1
        if top3[0] < A[i][j][l]
          insert ← 0
        else if top3[1] < A[i][j][l] & top[0] ≠ A[i][j][l]
          insert ← 1
        else if top3[2] < A[i][j][l] & top[0] ≠ A[i][j][l] & top[1] ≠ A[i][j][l]
          insert ← 2
        If insert > -1
          Temp3 ← top3
          for t ← insert to 2
            top3[t+1] = temp3[t]
          Top3[insert] ← A[i][j][l]
        else if (insert == -1) OR insert == 2
          l = 3
      l ← l + 1

```

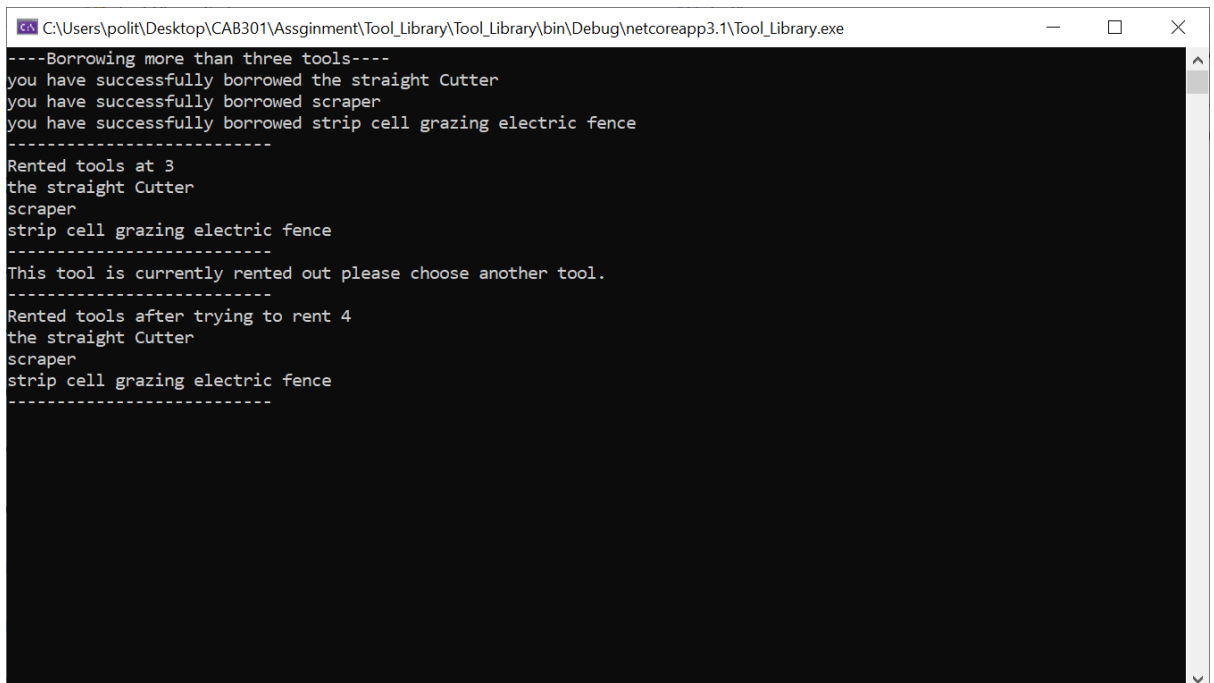
## Software Testing:

To test the functionality of the program and algorithm a range of test cases were created within the program, that is accessible through, inputting 3 on the login screen. These test cases have been constructed to ensure that full range of possible expected and unexpected user inputs are handled correctly. A Range of test cases have also been created to ensure that the sorting algorithms for the member binary search tree, tool collection array and top 3 rented tools all return the explained variables in the correct order.

### Renting cases:

A User is already borrowing 3 tools:

This test has been created to ensure that a user is unable to rent any more tools after they already have 3 tools. From the output in figure 4 the expected output has been displayed with the rented tools by the user not increasing after renting out 3 tools, with the user being told to return a tool before renting out a new tool.

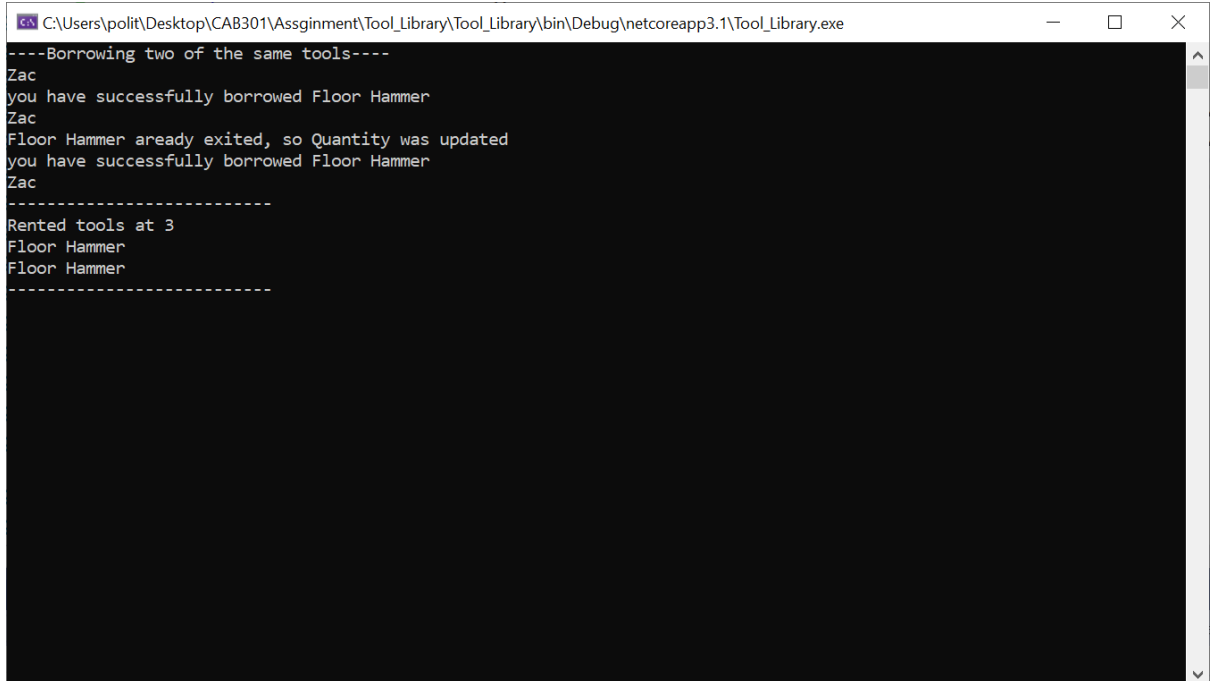


```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Borrowing more than three tools----
you have successfully borrowed the straight Cutter
you have successfully borrowed scraper
you have successfully borrowed strip cell grazing electric fence
-----
Rented tools at 3
the straight Cutter
scraper
strip cell grazing electric fence
-----
This tool is currently rented out please choose another tool.
-----
Rented tools after trying to rent 4
the straight Cutter
scraper
strip cell grazing electric fence
-----
```

Figure 4 - Borrowing more than three tools output

A User want to borrow 2 of the same tool:

This test has been created to ensure that a user is able to rent out two tools of the same type if there is more than one of that tool available. From figure 5, the expected output has been displayed with the user being able to borrow 2 of the same tools, with both instances of the tools being accounted for in the rented tool screen.

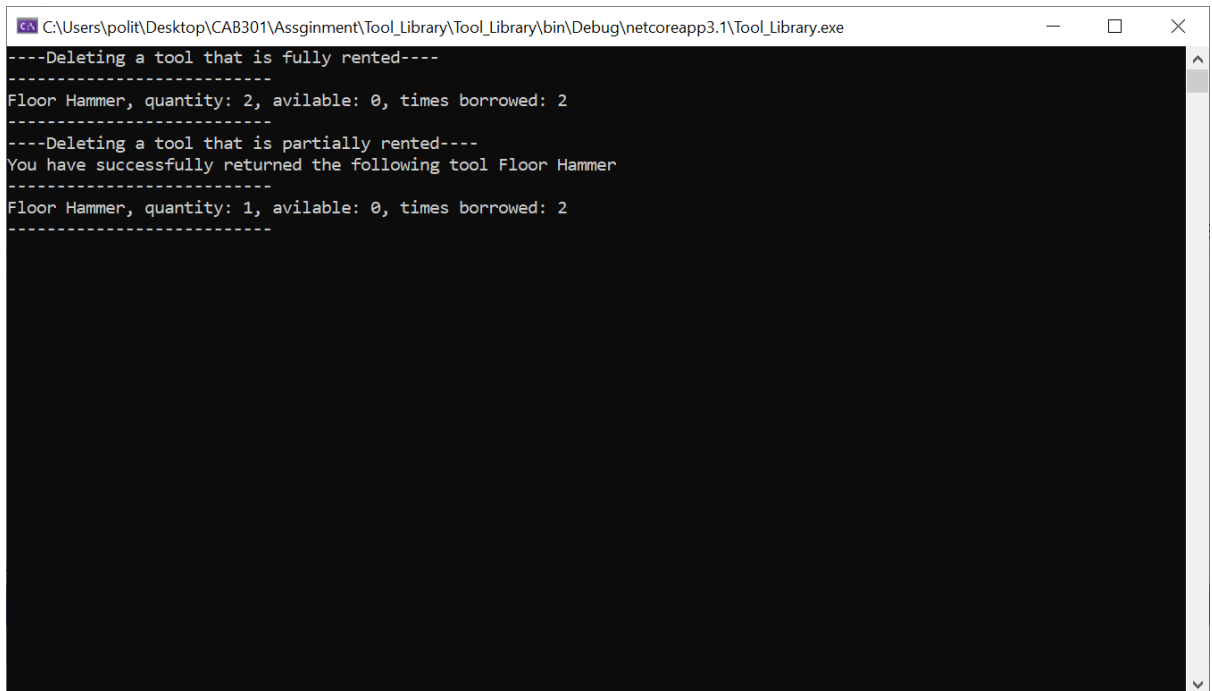


```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Borrowing two of the same tools----
Zac
you have successfully borrowed Floor Hammer
Zac
Floor Hammer already exited, so Quantity was updated
you have successfully borrowed Floor Hammer
Zac
-----
Rented tools at 3
Floor Hammer
Floor Hammer
-----
```

Figure 5 – borrowing two tools of the same type

Admin wants to remove a tool from the system currently being rented:

This test has been created to ensure that a tool that is being rented will not be removed from the tool collection before it is returned. From figure 6, two cases have been displayed. With the first case being all tools being currently rented out and the second test case having only a single tool still rented out. Both of these cases have displayed the desired results with the first case not removing any tools since all tools are unavailable. The second test case was also correct with only the newly returned tool being removed from the tool quantity bring it down to 1.

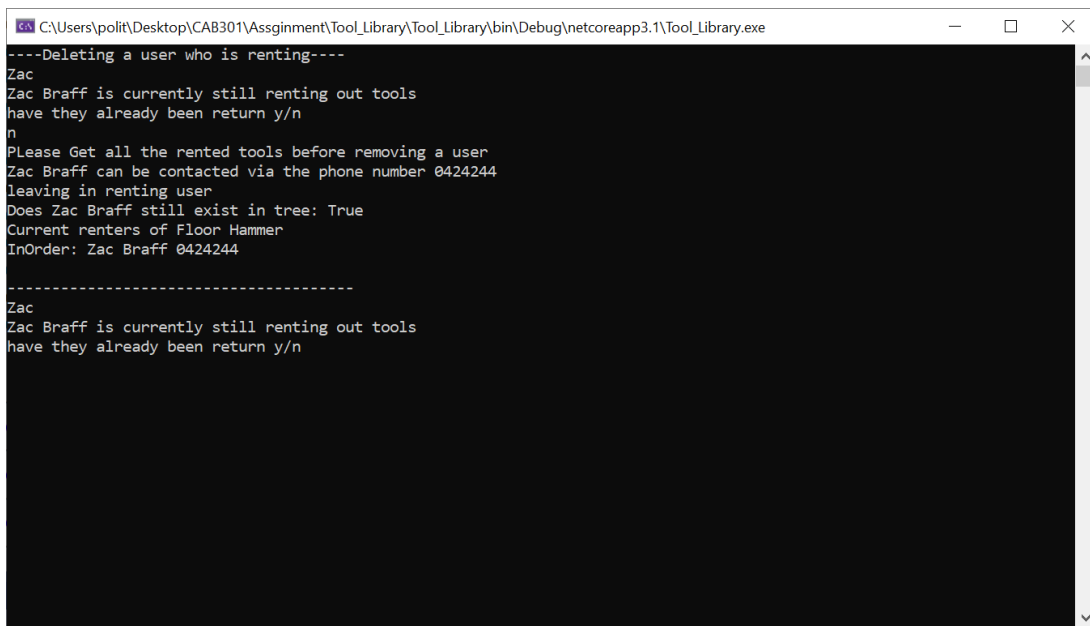


```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Deleting a tool that is fully rented----
-----
Floor Hammer, quantity: 2, avilable: 0, times borrowed: 2
-----
----Deleting a tool that is partially rented----
You have successfullly returned the following tool Floor Hammer
-----
Floor Hammer, quantity: 1, avilable: 0, times borrowed: 2
-----
```

Figure 6 - check if tools won't be removed when they are being rented

Admin wants to delete a user that is renting a tool:

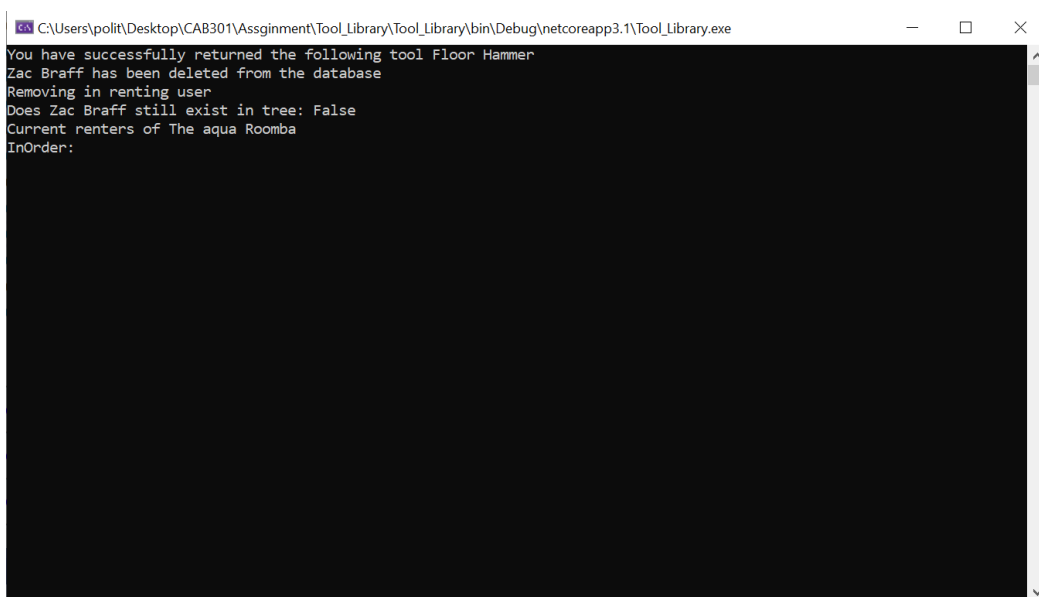
This test case is to make sure that a user renting out tools will not be instantly removed from the member tree, before they have returned all their rented-out tools. Since a member can still be removed from the member tree while having tools rented out, two test cases have been created. One where the user has returned all their tools to an admin and the second one being where the user still holds all their rented tools. The test case in figure 7. Shows what happens when a user is still renting out the tools, with them being kept in the member collection



```
C:\Users\poli\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Deleting a user who is renting----
Zac
Zac Braff is currently still renting out tools
have they already been return y/n
n
Please Get all the rented tools before removing a user
Zac Braff can be contacted via the phone number 0424244
leaving in renting user
Does Zac Braff still exist in tree: True
Current renters of Floor Hammer
InOrder: Zac Braff 0424244
-----
Zac
Zac Braff is currently still renting out tools
have they already been return y/n
```

Figure 7 - case one for removing a user renting tools

The figure bellow shows the user being removed, from the member tree, along with them being removed from the renting tree of all tools they were renting out at the time of deletion.

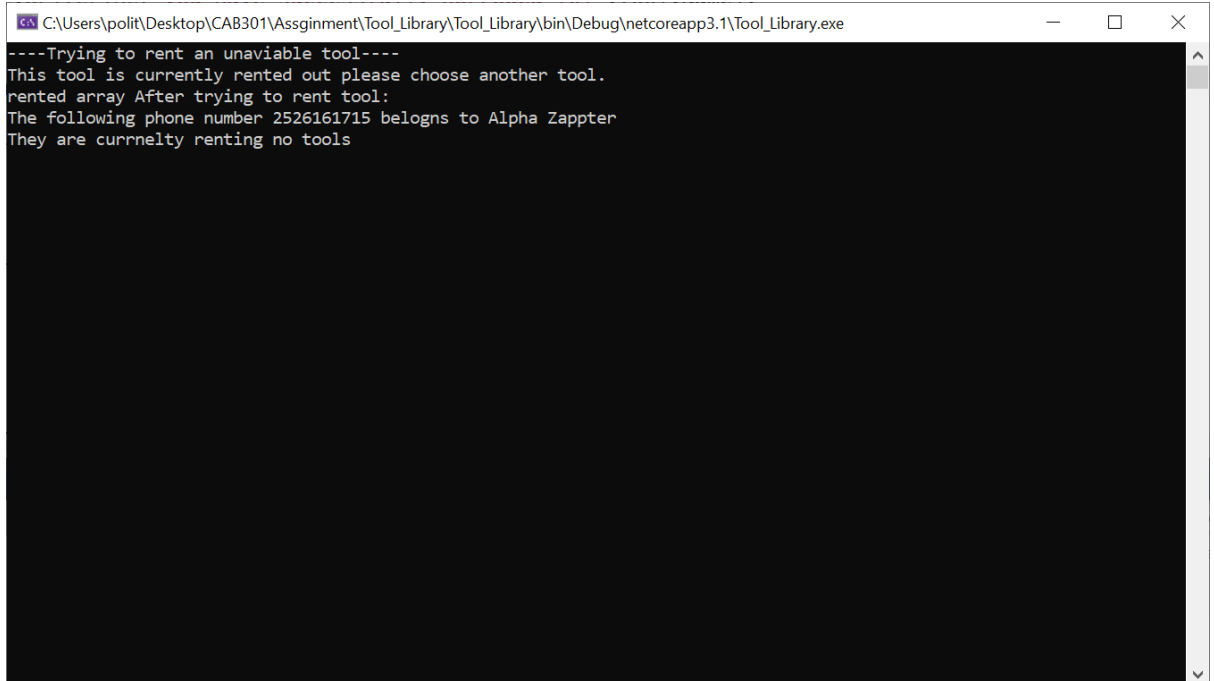


```
C:\Users\poli\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
You have successfully returned the following tool Floor Hammer
Zac Braff has been deleted from the database
Removing in renting user
Does Zac Braff still exist in tree: False
Current renters of The aqua Roomba
InOrder:
```

Figure 8 - case two for remove a user renting tools

A User want to rent a tool that is unavailable:

This test case was created to test that a user is unable to rent a tool when a tool has none available. From figure 9, the expected result has been printed with the member being unable to rent out the tool when none are available, with it not being added to the user's renting tool collection.



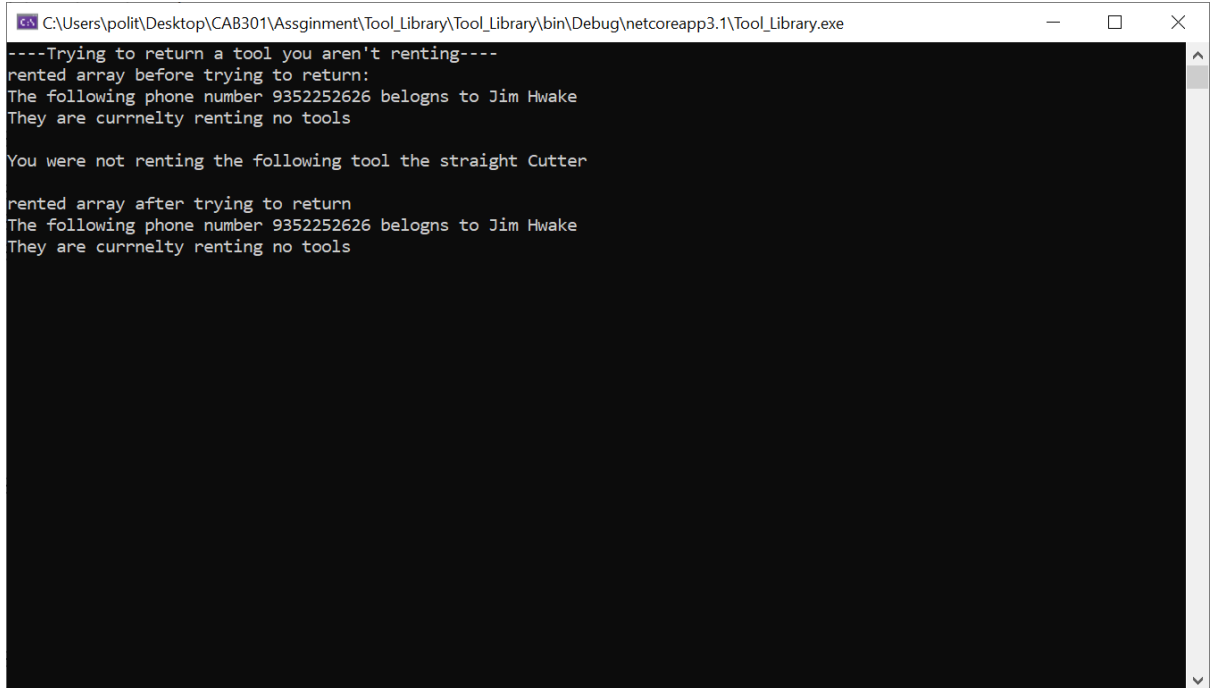
```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Trying to rent an unaviable tool----
This tool is currently rented out please choose another tool.
rented array After trying to rent tool:
The following phone number 2526161715 belogns to Alpha Zappter
They are currnelty renting no tools
```

*Figure 9 - User trying to rent out a tool that is not available*



A User wants to return a tool when they have none rented:

This class has been created to ensure that a user can not break the program via trying to return a tool they are not renting, from the output in figure 10. it can be seen that no tools have been added or removed from Jim Hawkes renting tool array, with the correct message being displayed when the user tries to return a tool they are not renting.



```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Trying to return a tool you aren't renting----
rented array before trying to return:
The following phone number 9352252626 belongs to Jim Hwake
They are currnelty renting no tools

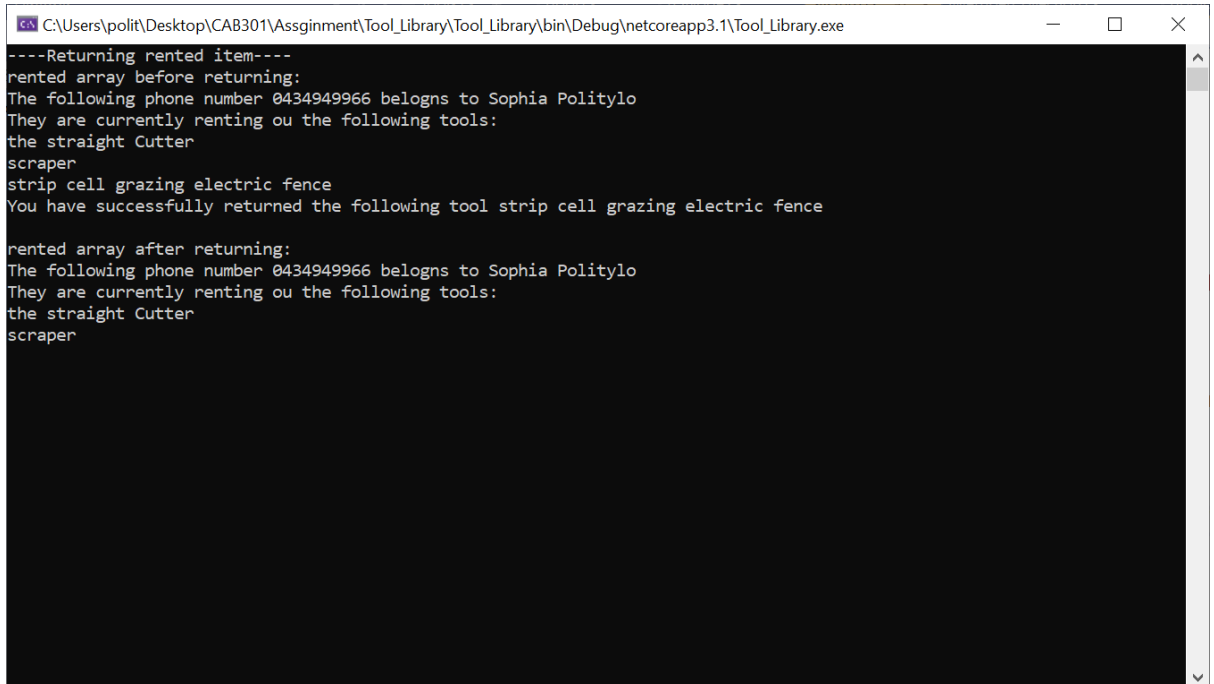
You were not renting the following tool the straight Cutter

rented array after trying to return
The following phone number 9352252626 belongs to Jim Hwake
They are currnelty renting no tools
```

Figure 10 - User trying to return a tool they are currently renting

A User want to return a rented tool:

This test case has been constructed to ensure that a user is able to return a tool under the normal renting conditions. From figure 11, the expected output has been displayed with the user being able to return a tool they are currently renting, with it being removed from their rented array, along with the available quantity being increased.



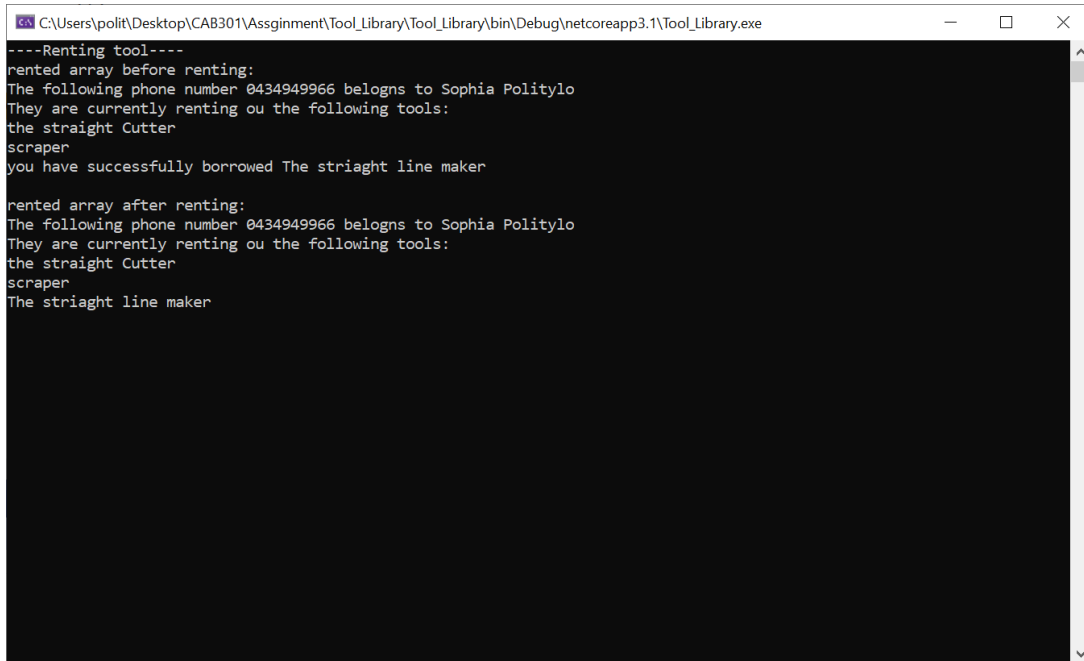
```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Returning rented item----
rented array before returning:
The following phone number 0434949966 belogns to Sophia Politylo
They are currently renting ou the following tools:
the straight Cutter
scraper
strip cell grazing electric fence
You have successfully returned the following tool strip cell grazing electric fence

rented array after returning:
The following phone number 0434949966 belogns to Sophia Politylo
They are currently renting ou the following tools:
the straight Cutter
scraper
```

Figure 11 - User returning a tool that they are renting

#### A User wants to rent a new tool:

This test case has been constructed to ensure that a user is able to rent out a new tool when they have under three tools. From figure 12, the expected output has been displayed with the user's rented tools being increased, with the user now having the newly rented tool in the array.



```
C:\Users\polity\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Renting tool----
rented array before renting:
The following phone number 0434949966 belongs to Sophia Politylo
They are currently renting ou the following tools:
the straight Cutter
scraper
you have successfully borrowed The striaght line maker

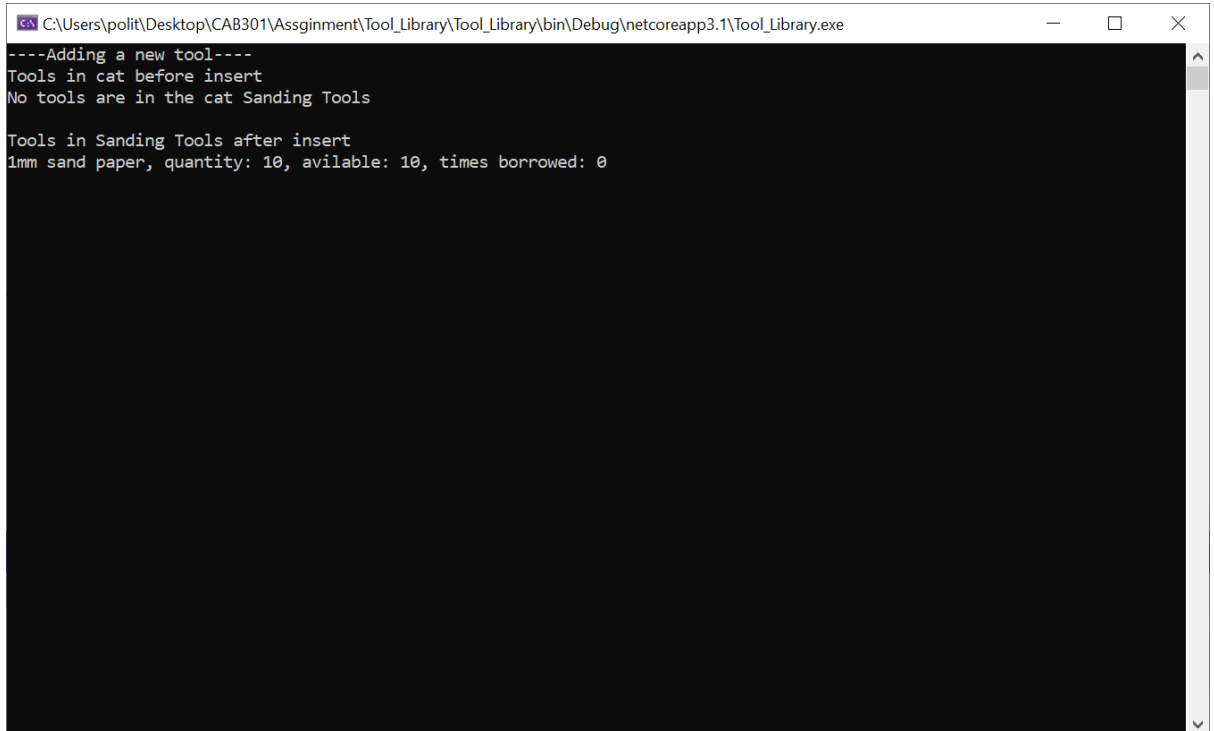
rented array after renting:
The following phone number 0434949966 belongs to Sophia Politylo
They are currently renting ou the following tools:
the straight Cutter
scraper
The striaght line maker
```

Figure 12 - User renting out a new tool

### Updating fields:

Admin want to add a new tool:

This test case has been created to ensure that the user is able to add a new tool to the tool collection class. The output from this test case can be seen inf figure 13, form this output this test case passed with the new tool being added to the Sanding tools tool collection.



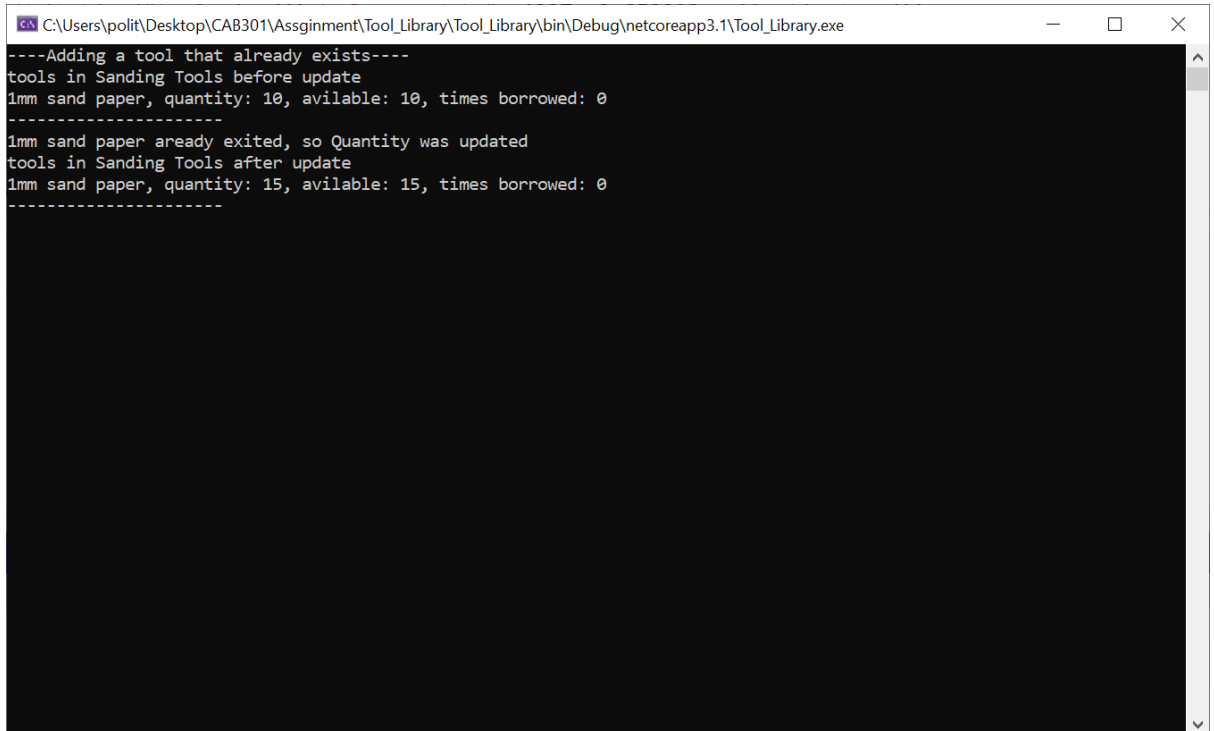
```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Adding a new tool----
Tools in cat before insert
No tools are in the cat Sanding Tools

Tools in Sanding Tools after insert
1mm sand paper, quantity: 10, avilable: 10, times borrowed: 0
```

Figure 13 - Admin adding a new tool to a tool class

Admin want to add another tool of the same type:

This test case has been created to ensure that the admins can add a tool of the same type, via increasing the quantity of an already existing tool. The out put from this test case is in figure 14, which shows that they admin is able to increase the quantity of a pre-existing tool.

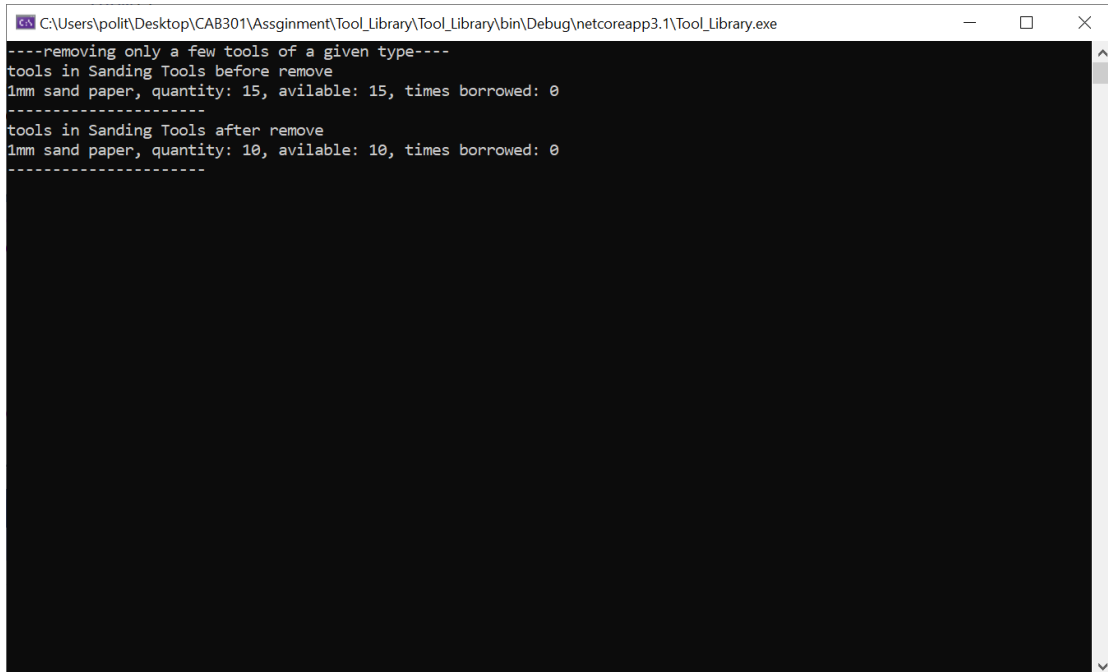
A screenshot of a Windows command prompt window. The title bar shows the file path: C:\Users\polit\Desktop\CAB301\Assginment\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe. The window contains the following text:

```
----Adding a tool that already exists----  
tools in Sanding Tools before update  
1mm sand paper, quantity: 10, avilable: 10, times borrowed: 0  
-----  
1mm sand paper aready exited, so Quantity was updated  
tools in Sanding Tools after update  
1mm sand paper, quantity: 15, avilable: 15, times borrowed: 0  
-----
```

Figure 14 - Admin adding to the quantity of an already existing tool

Admin only want to remove a few tools from a tool class:

This test case has been created to ensure that the quantity of a tool can be decreased without removing the tool from the tool collection class. Figure 15, shows the output from this test case with all the outputs being as expected. With the quantity of the 1mm sandpaper being decrease by 5, while still keeping it with the tool collection array.

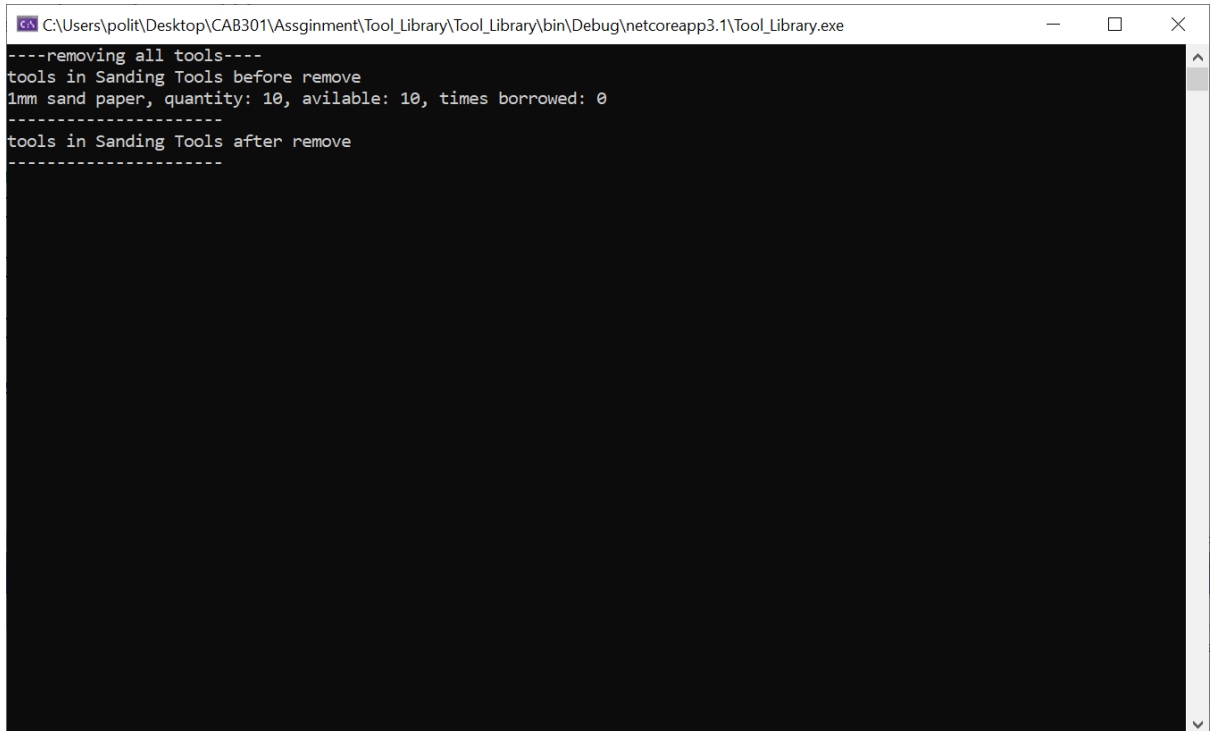


```
C:\Users\polity\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----removing only a few tools of a given type----
tools in Sanding Tools before remove
1mm sand paper, quantity: 15, avilable: 15, times borrowed: 0
-----
tools in Sanding Tools after remove
1mm sand paper, quantity: 10, avilable: 10, times borrowed: 0
-----
```

Figure 15 -admin remove 5 tools from a given tool, without removing the tool from the tool collection

Admin wants to remove all the tools from a class:

This test case has been created to ensure that a tool can be fully removed from a tool collection array. This test case can be seen in figure 16, with the 1mm sanding paper, no longer being present after its quantity is decreased to zero.

A screenshot of a Windows command prompt window. The title bar shows the file path: C:\Users\polit\Desktop\CAB301\Assginment\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe. The command prompt has a black background with white text. The text displayed is: 

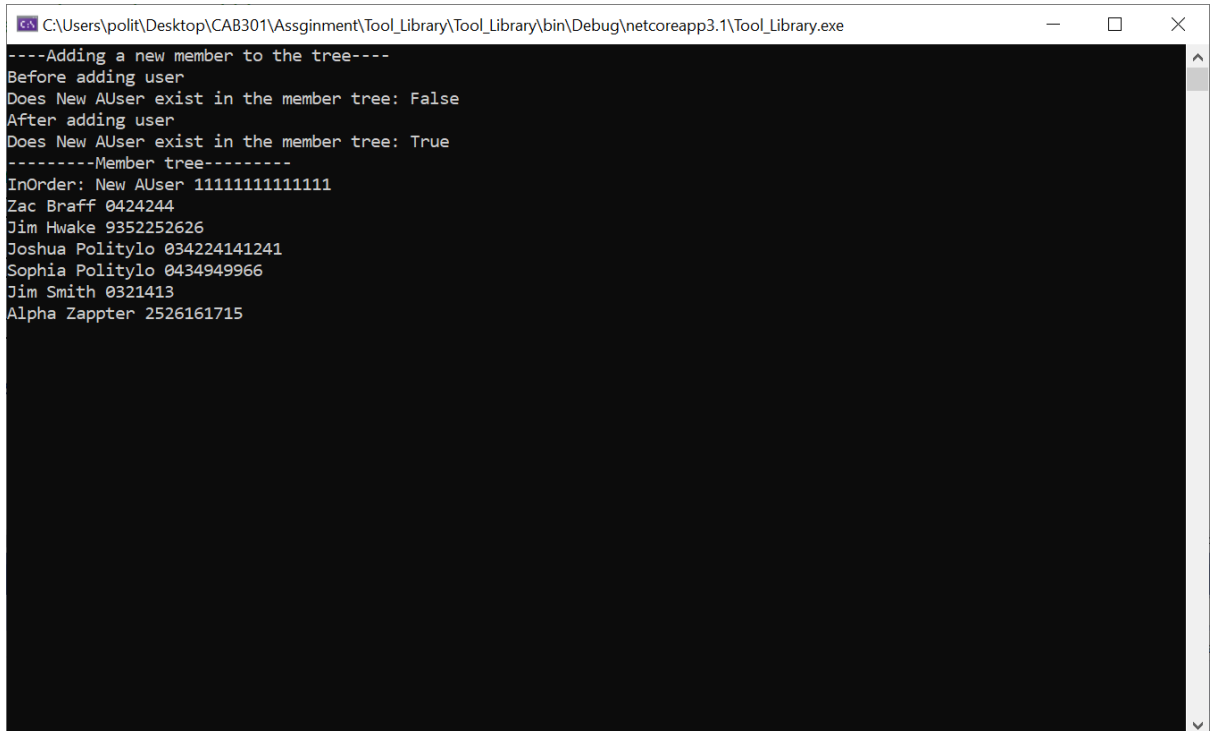
```
----removing all tools----  
tools in Sanding Tools before remove  
1mm sand paper, quantity: 10, avilable: 10, times borrowed: 0  
-----  
tools in Sanding Tools after remove  
-----
```

 The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

Figure 16 - removing a tool fully from a tool collection

Admin wants to add a new user:

This test case has been created to ensure that a new user can be inserted into the member tree, while keeping the tree sorted. From this test the output in figure 17, was generated showing that the user New Auser, was not originally in the binary search tree but after adding them, they have now been inserted into the search tree, at the correct position with them being at the top of the sorted array, given that their last name starts with an A.



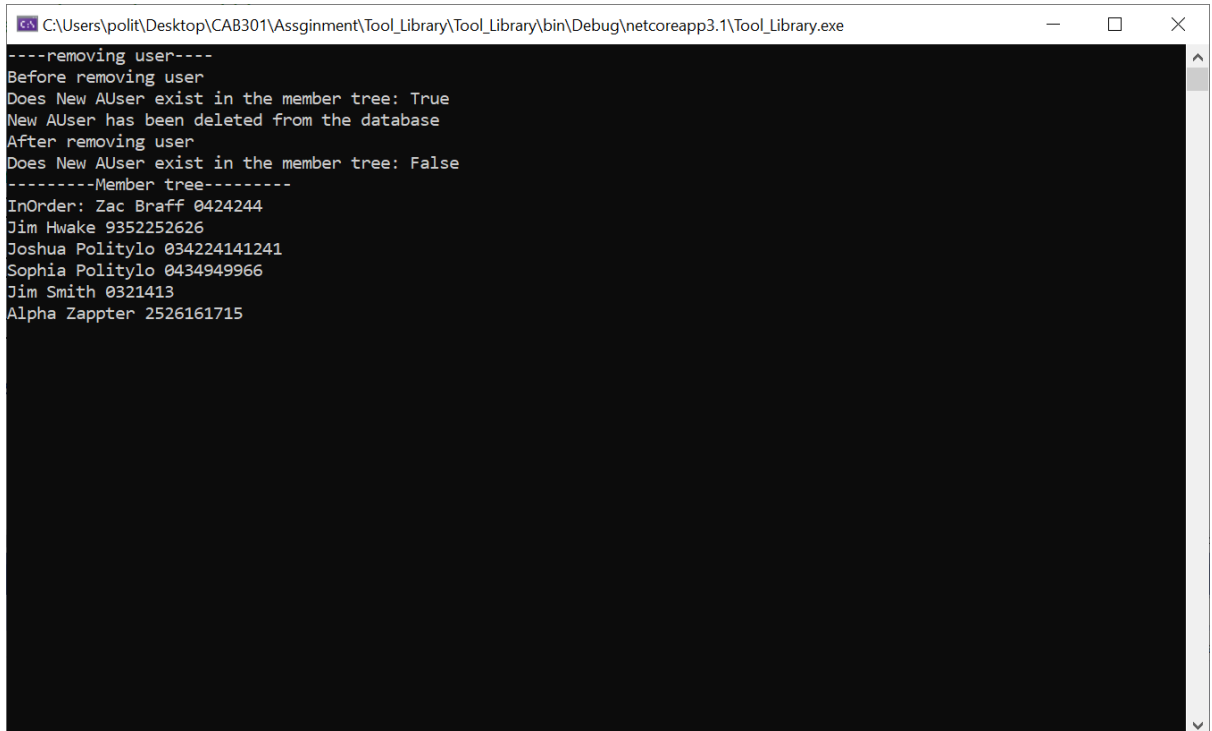
```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----Adding a new member to the tree----
Before adding user
Does New Auser exist in the member tree: False
After adding user
Does New Auser exist in the member tree: True
-----Member tree-----
InOrder: New Auser 11111111111111
Zac Braff 0424244
Jim Hwake 9352252626
Joshua Politylo 034224141241
Sophia Politylo 0434949966
Jim Smith 0321413
Alpha Zappter 2526161715
```

Figure 17 - adding a new member to the member tree



Admin want to remove a user from the system:

This test case was created to show the removal of a member that is not renting any tools from the members tree, works as expected with the member no longer being present within the tree. The output from this test is shown in figure 18, which displays that the user New Auser, is originally in the binary search tree, but after being removed is no long able to be found in it.



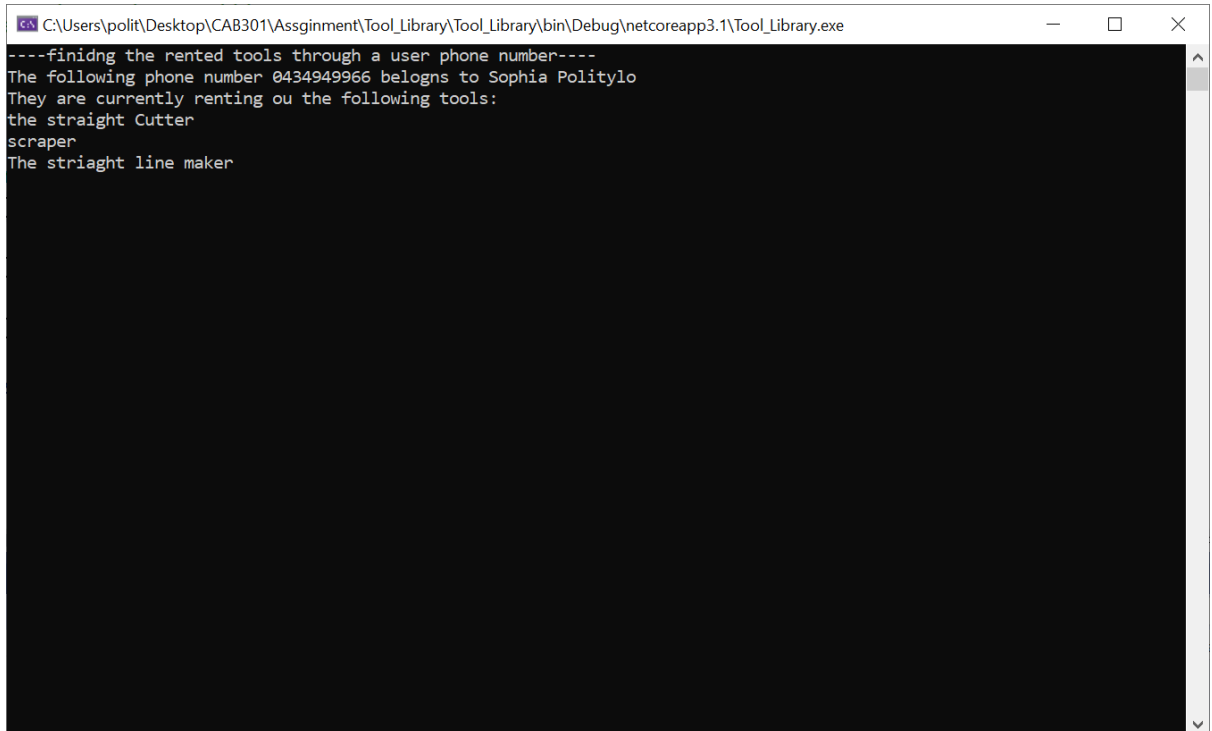
```
C:\Users\poli\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----removing user----
Before removing user
Does New AUser exist in the member tree: True
New AUser has been deleted from the database
After removing user
Does New AUser exist in the member tree: False
-----Member tree-----
InOrder: Zac Braff 0424244
Jim Hwake 9352252626
Joshua Politylo 034224141241
Sophia Politylo 0434949966
Jim Smith 0321413
Alpha Zappter 2526161715
```

Figure 18 - removing a member from the member tree

### Search Functions:

Searching for a user's phone number that does not exist:

This test case was constructed to show that displaying a user's rented out tools work correctly when a user is renting out tools. This test's output can be seen in figure 19, where the three tools that the user with the phone number 0434949966 has rented out is displayed with no more being shown.

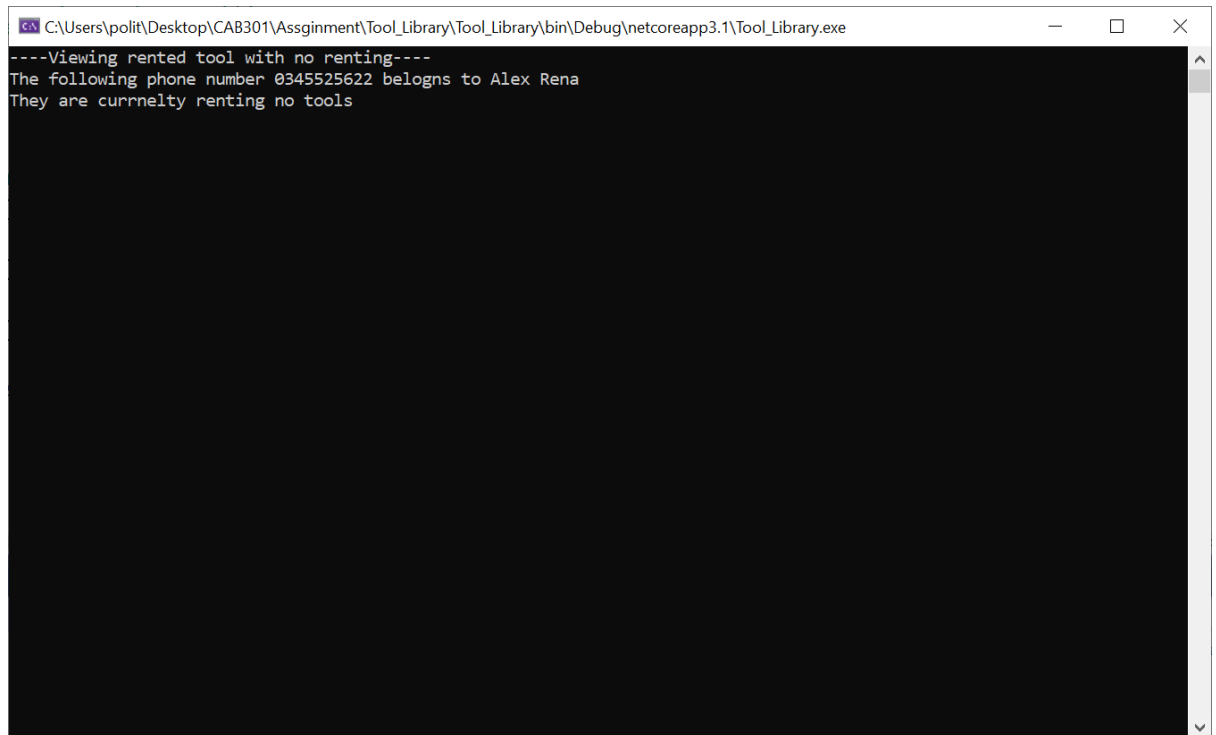


```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
----finidng the rented tools through a user phone number----
The following phone number 0434949966 belogns to Sophia Politylo
They are currently renting ou the following tools:
the straight Cutter
scraper
The striaght line maker
```

Figure 19 - display the tools that a user is currently renting out through a given phone number

### User viewing their currently rented tools when they have none out

This test has been created to make sure that a user having no tools rented out does not crash the display function when they have no tools being rented out. The output for this test can be seen in figure 20, which shows that the program functions when the user has no tools rented out and also displays the correct message telling the user they have no tools rented out.

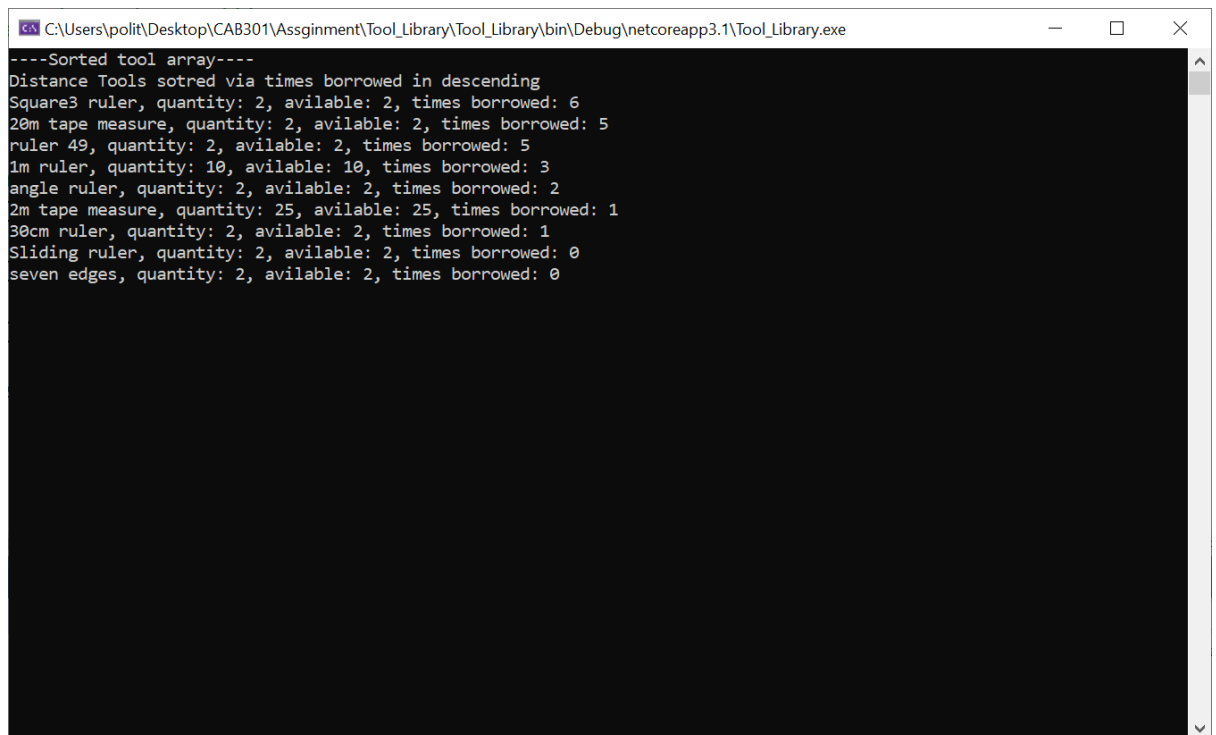
A screenshot of a Windows command prompt window. The title bar shows the file path: C:\Users\polit\Desktop\CAB301\Assginment\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe. The command prompt has a black background with white text. The output text is as follows:

```
---Viewing rented tool with no renting---  
The following phone number 0345525622 belongs to Alex Rena  
They are currnelty renting no tools
```

Figure 20 - viewing the rented tools when a user is renting out no tools

### Descending times borrowed merge sort on a Tool Collection

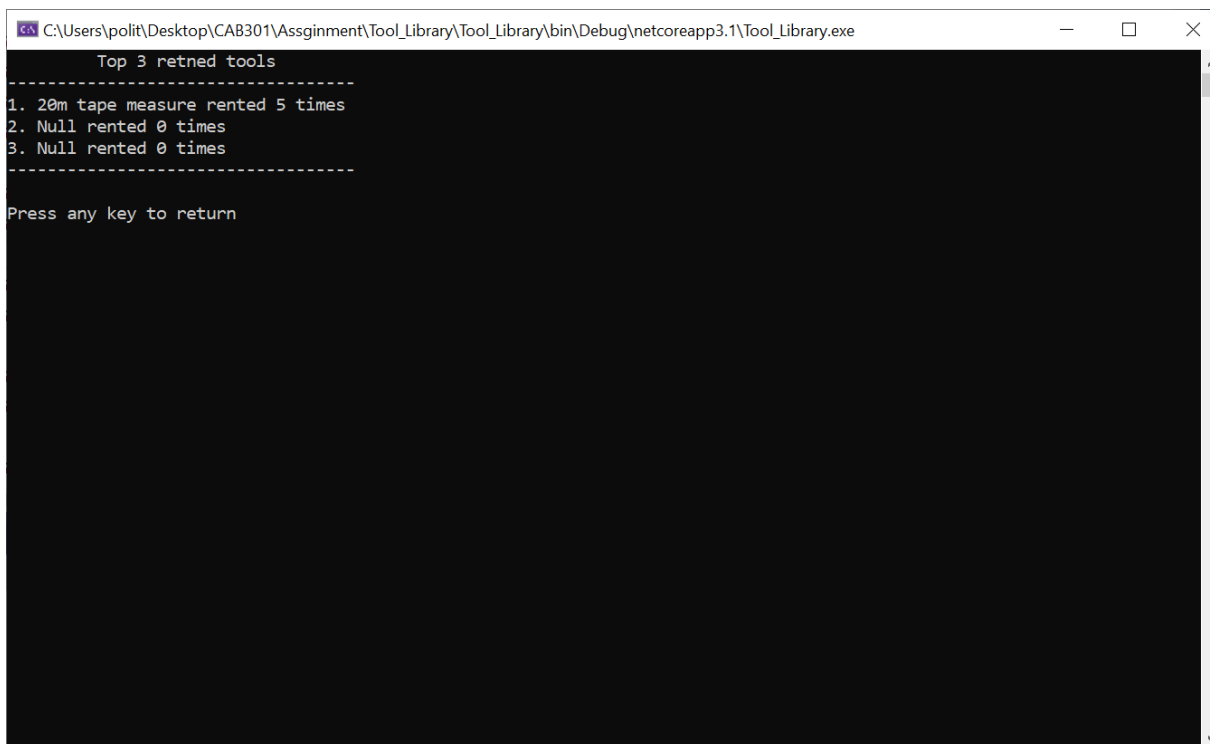
This test case was created to ensure that the merge sort functionality of the Tool collection array works correctly with the it is testing if the sort function will place all the tools in a collection in descending order in the array based on the number of times the tools have been borrowed. From the output in figure 21, this has indeed been the case with the distance measuring tool collection being sort as expected with the tools with the most borrowing being at the head of the array with each tool after it having a less number of borrowing when compared to the one before it.



```
----Sorted tool array----
Distance Tools sotred via times borrowed in descending
Square3 ruler, quantity: 2, avilable: 2, times borrowed: 6
20m tape measure, quantity: 2, avilable: 2, times borrowed: 5
ruler 49, quantity: 2, avilable: 2, times borrowed: 5
1m ruler, quantity: 10, avilable: 10, times borrowed: 3
angle ruler, quantity: 2, avilable: 2, times borrowed: 2
2m tape measure, quantity: 25, avilable: 25, times borrowed: 1
30cm ruler, quantity: 2, avilable: 2, times borrowed: 1
Sliding ruler, quantity: 2, avilable: 2, times borrowed: 0
seven edges, quantity: 2, avilable: 2, times borrowed: 0
```

Figure 21 - checking if the merge sort function correctly places the tools in descending order via times borrowed

User checking the Top rented tools when none or only 1 or 2 have been rented out:  
This test case has been created to make sure that the top 3 will only pull out tools that have been rented and leave the rest as undefined until more tools have been borrowed. From this test case the output from figure 22, was generated. Which shows that the top 3 works as expected only selecting tools that have been rented are displayed within the top 3 rented tools.

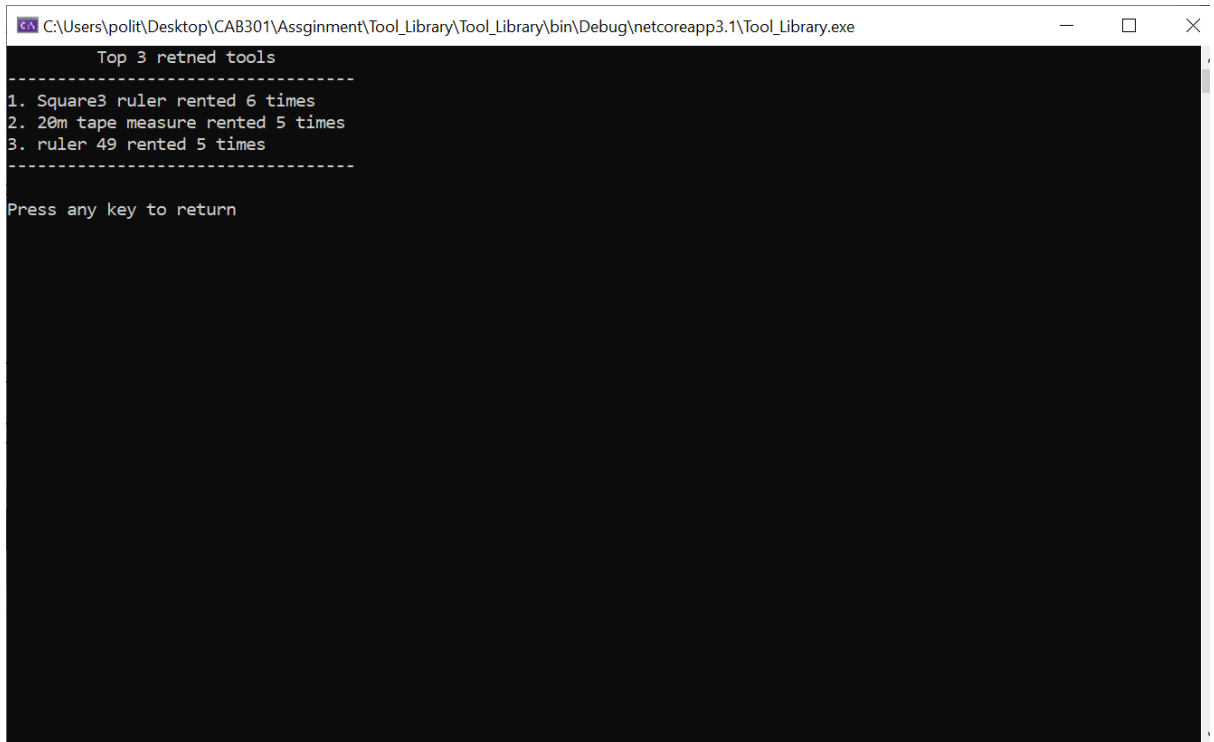


```
C:\Users\poli\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Top 3 rented tools
-----
1. 20m tape measure rented 5 times
2. Null rented 0 times
3. Null rented 0 times
-----
Press any key to return
```

Figure 22 - top 3 rented tools when only one tool has been rented out

### User checking the Top rented Tools

This test has been created to ensure that the top 3 rented tools are displayed correctly with the correct tools being selected along with all the correct tools being present within the top 3. From figure 23, this test case can be seen to have passed with the number 1 most rented tool having the greatest number of borrowing at 6 while 2 and 3 have the same number of borrowings but are for different tools, showing that this function is in fact pulling out the correct tools.

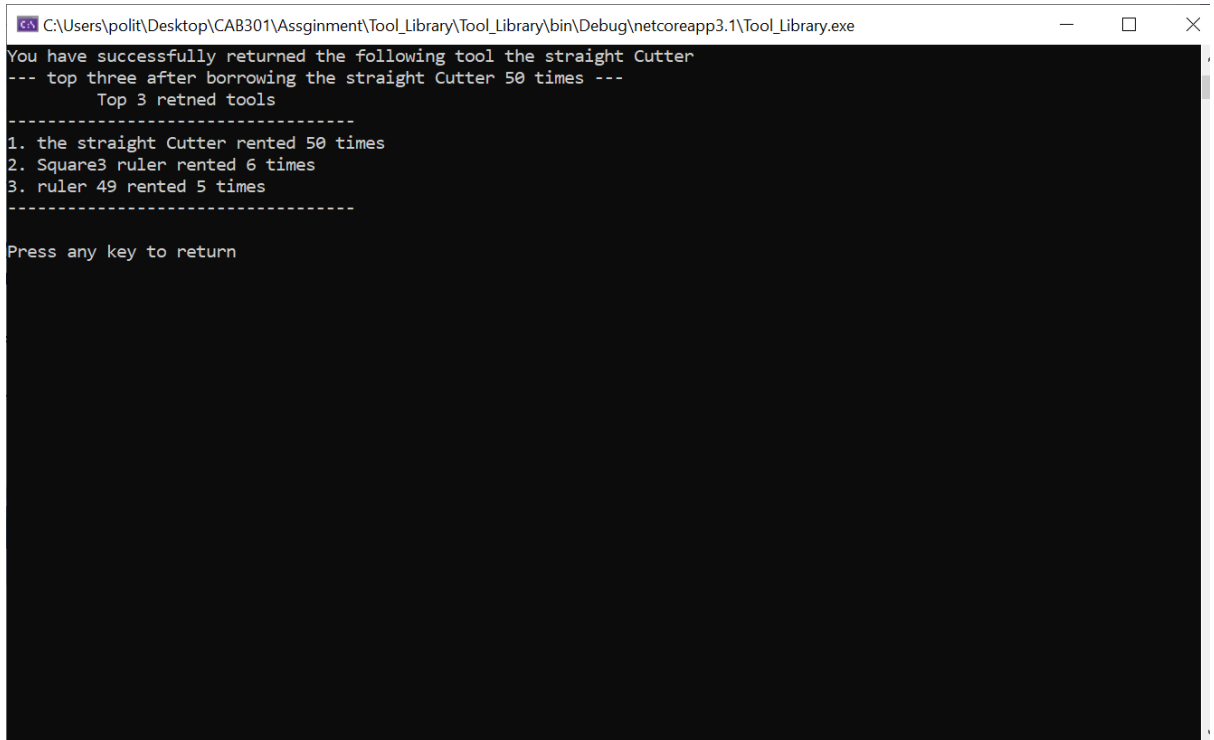


```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Top 3 retned tools
-----
1. Square3 ruler rented 6 times
2. 20m tape measure rented 5 times
3. ruler 49 rented 5 times
-----
Press any key to return
```

Figure 23 - user checking the top 3 rented tools with a range of tools having been borrowed out

### Updating top3 rented tools

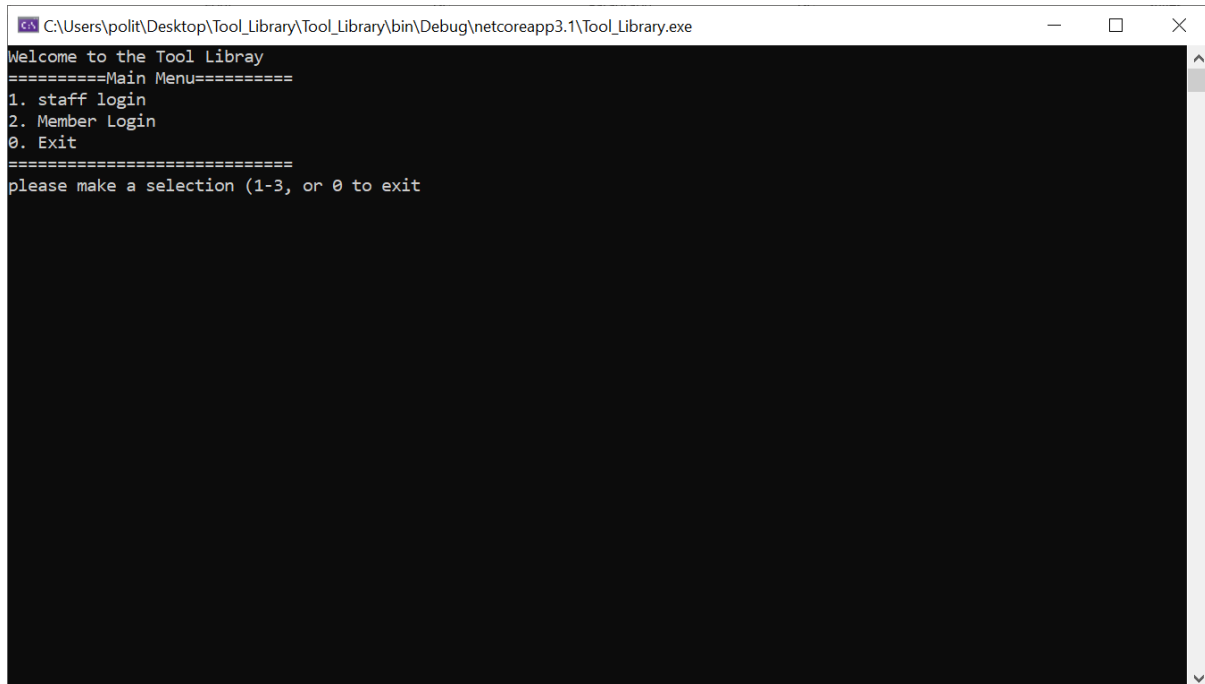
This test has been constructed to ensure that the top 3 list will be updated properly when a other tool becomes the most borrowed tool within the program. From figure24. It can be seen that the new tool is correctly placed within the top 3 array, with all the old tools being moved down to their new positions.



```
C:\Users\polit\Desktop\CAB301\Assginment\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
You have successfully returned the following tool the straight Cutter
--- top three after borrowing the straight Cutter 50 times ---
    Top 3 retned tools
-----
1. the straight Cutter rented 50 times
2. Square3 ruler rented 6 times
3. ruler 49 rented 5 times
-----
Press any key to return
```

Figure 24 - checking the top three rented tools after renting out another set of tools

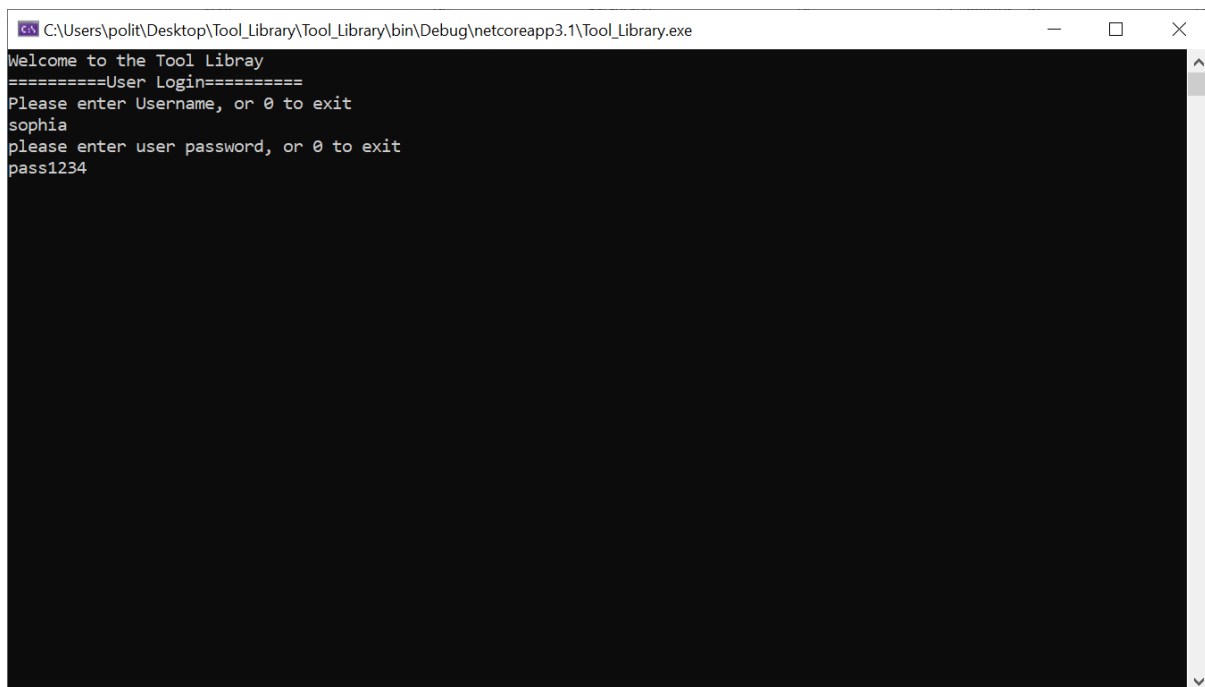
## Normal operation of the program (menu screens)



```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Library
=====Main Menu=====
1. staff login
2. Member Login
0. Exit
=====
please make a selection (1-3, or 0 to exit)
```

The screenshot shows a Windows application window titled "C:\Users\polit\Desktop\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe". The application displays a main menu with the following text: "Welcome to the Tool Library", "=====Main Menu=====", "1. staff login", "2. Member Login", "0. Exit", "=====", and "please make a selection (1-3, or 0 to exit)".

Figure 25 - Main login screen

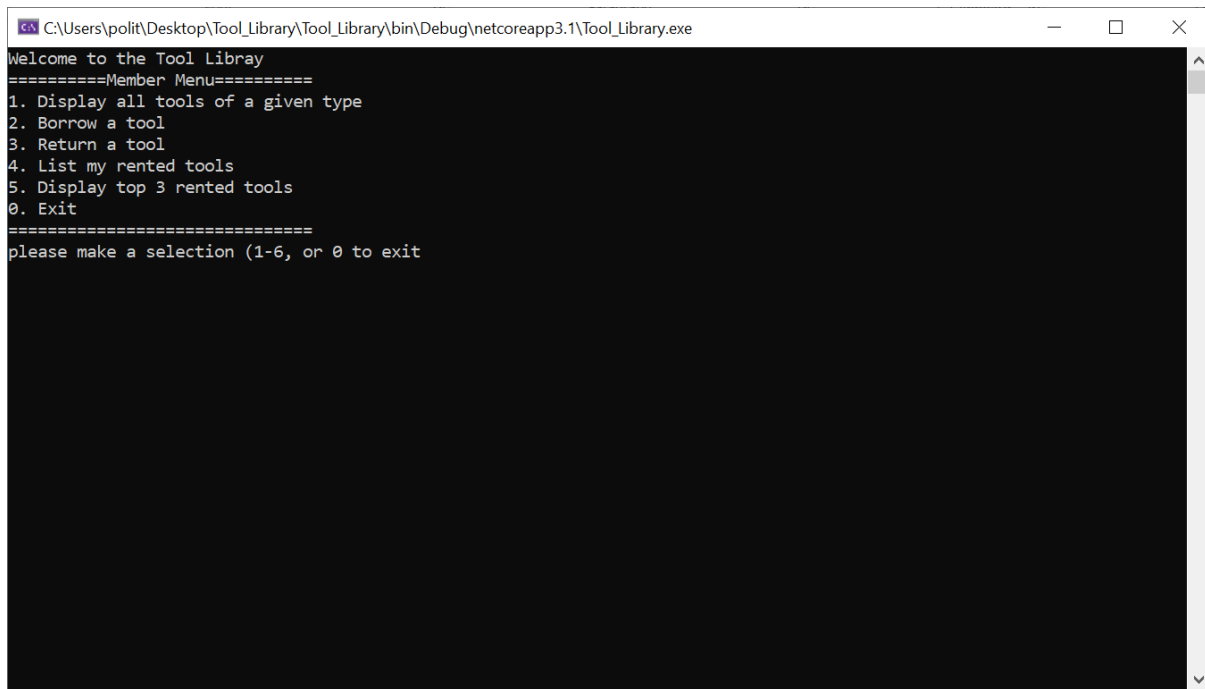


```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Library
=====User Login=====
Please enter Username, or 0 to exit
sophia
please enter user password, or 0 to exit
pass1234
```

The screenshot shows the same application window as Figure 25, but now it is in the user login screen. The text displayed is: "Welcome to the Tool Library", "=====User Login=====", "Please enter Username, or 0 to exit", "sophia", "please enter user password, or 0 to exit", and "pass1234".

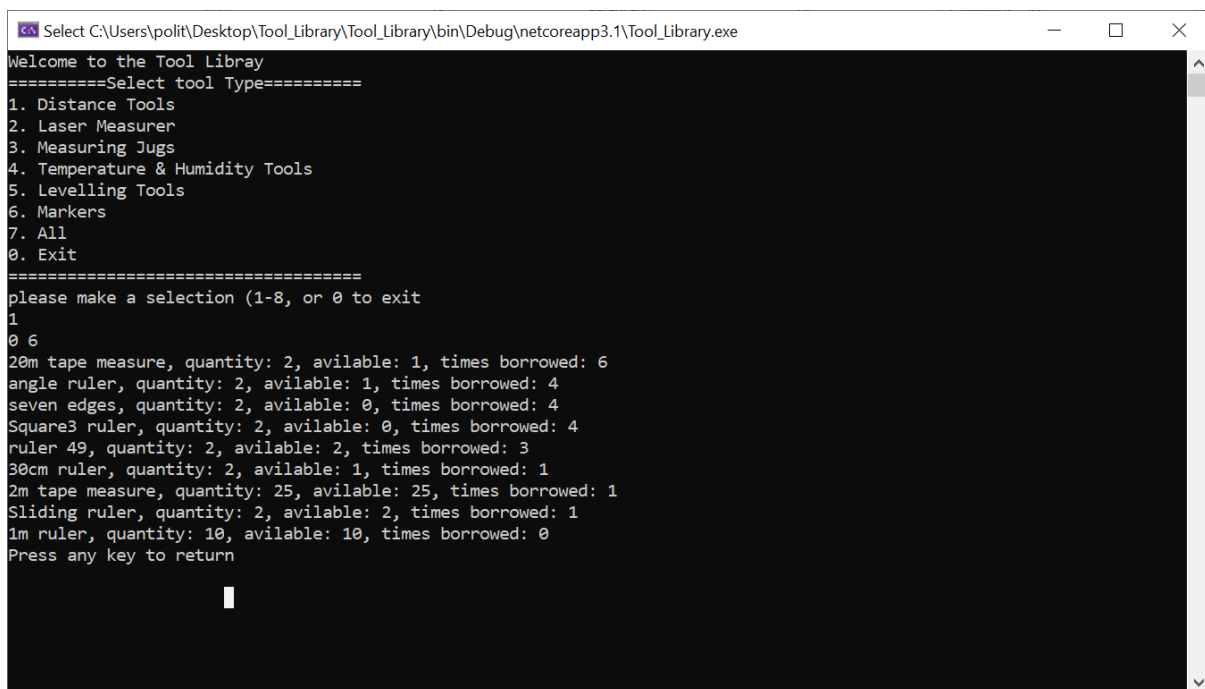
Figure 26 - Asking for login information screen



A screenshot of a Windows command prompt window titled "C:\Users\polit\Desktop\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe". The window displays a "Member Menu" with the following options: 1. Display all tools of a given type, 2. Borrow a tool, 3. Return a tool, 4. List my rented tools, 5. Display top 3 rented tools, and 0. Exit. The prompt "please make a selection (1-6, or 0 to exit)" is shown at the bottom.

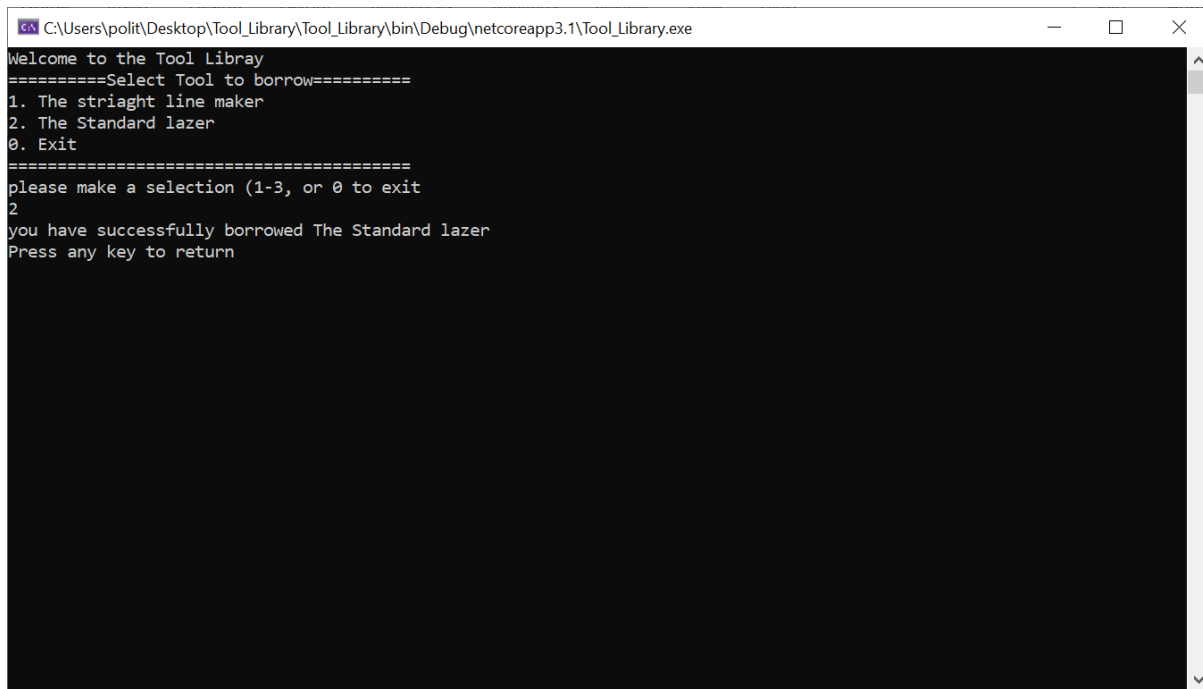
```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Library
=====Member Menu=====
1. Display all tools of a given type
2. Borrow a tool
3. Return a tool
4. List my rented tools
5. Display top 3 rented tools
0. Exit
=====
please make a selection (1-6, or 0 to exit
```

Figure 27 - member main menu

A screenshot of a Windows command prompt window titled "Select C:\Users\polit\Desktop\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe". The window displays a "Select tool Type" menu with the following options: 1. Distance Tools, 2. Laser Measurer, 3. Measuring Jugs, 4. Temperature & Humidity Tools, 5. Levelling Tools, 6. Markers, 7. All, and 0. Exit. The prompt "please make a selection (1-8, or 0 to exit)" is shown. The user has entered '1', and the program displays a list of tools with their quantities, available counts, and borrowed counts. The prompt "Press any key to return" is shown at the bottom.

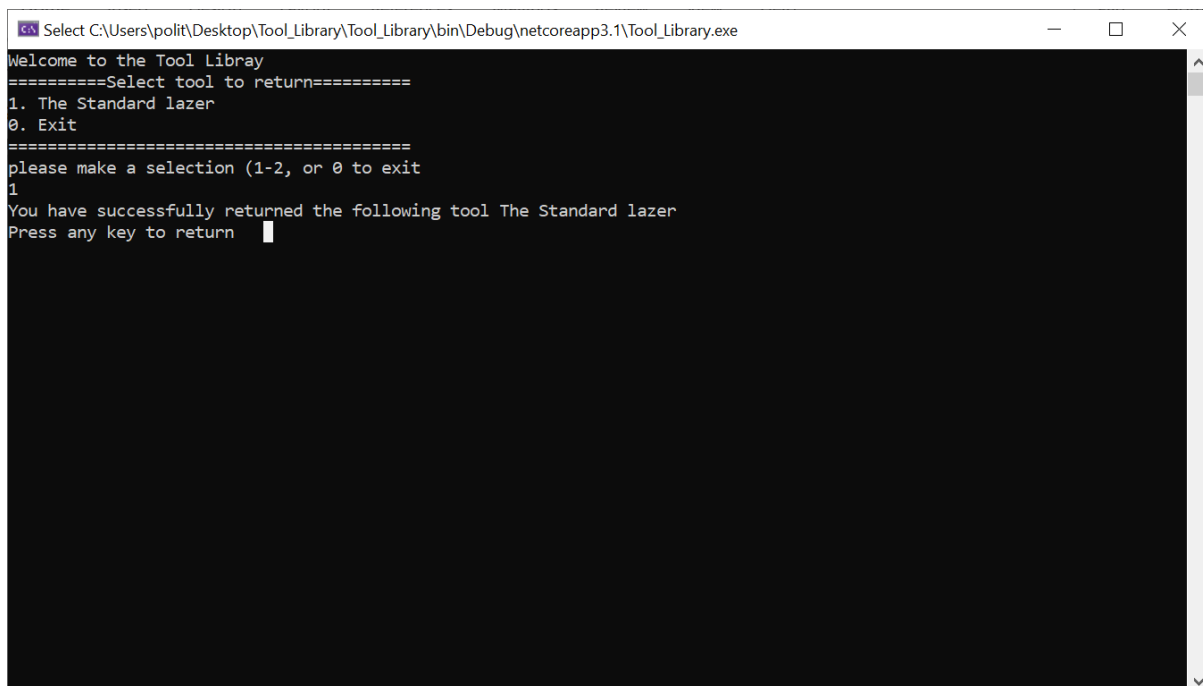
```
Select C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Library
=====Select tool Type=====
1. Distance Tools
2. Laser Measurer
3. Measuring Jugs
4. Temperature & Humidity Tools
5. Levelling Tools
6. Markers
7. All
0. Exit
=====
please make a selection (1-8, or 0 to exit
1
0 6
20m tape measure, quantity: 2, avilable: 1, times borrowed: 6
angle ruler, quantity: 2, avilable: 1, times borrowed: 4
seven edges, quantity: 2, avilable: 0, times borrowed: 4
Square3 ruler, quantity: 2, avilable: 0, times borrowed: 4
ruler 49, quantity: 2, avilable: 2, times borrowed: 3
30cm ruler, quantity: 2, avilable: 1, times borrowed: 1
2m tape measure, quantity: 25, avilable: 25, times borrowed: 1
Sliding ruler, quantity: 2, avilable: 2, times borrowed: 1
1m ruler, quantity: 10, avilable: 10, times borrowed: 0
Press any key to return
```

Figure 28 – Display Tools of a given type menu



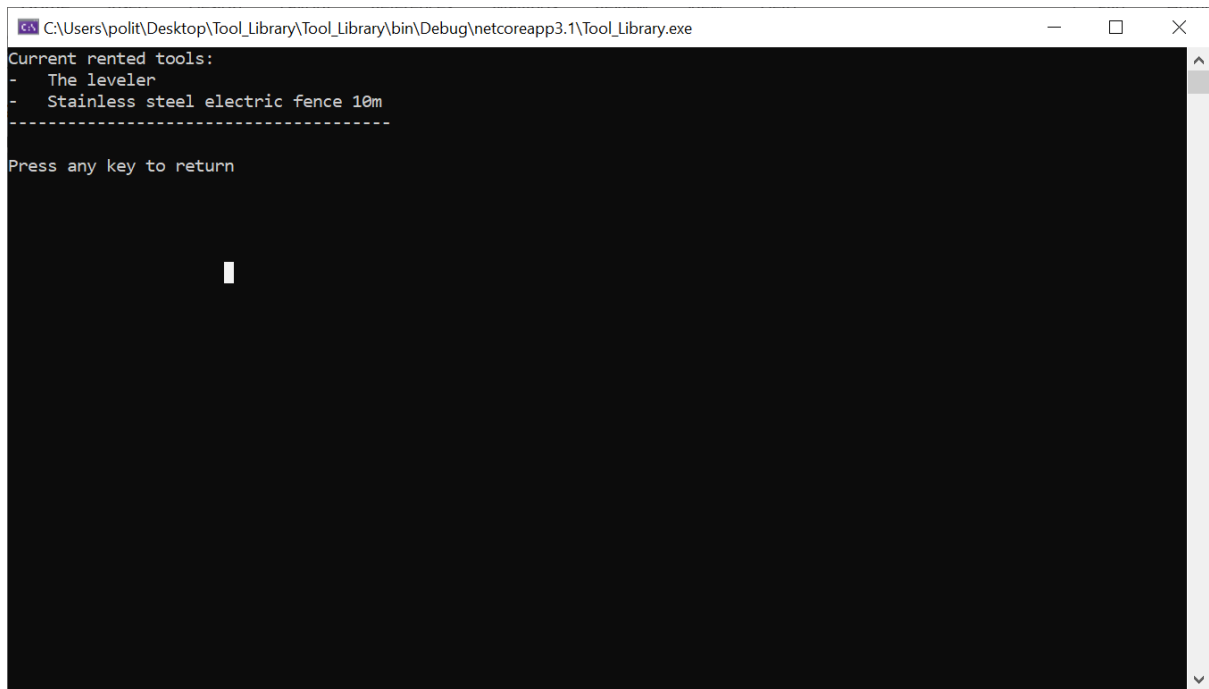
```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Library
=====Select Tool to borrow=====
1. The straight line maker
2. The Standard lazer
0. Exit
=====
please make a selection (1-3, or 0 to exit
2
you have successfully borrowed The Standard lazer
Press any key to return
```

Figure 29 - renting out a new tool



```
Select C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Library
=====Select tool to return=====
1. The Standard lazer
0. Exit
=====
please make a selection (1-2, or 0 to exit
1
You have successfully returned the following tool The Standard lazer
Press any key to return
```

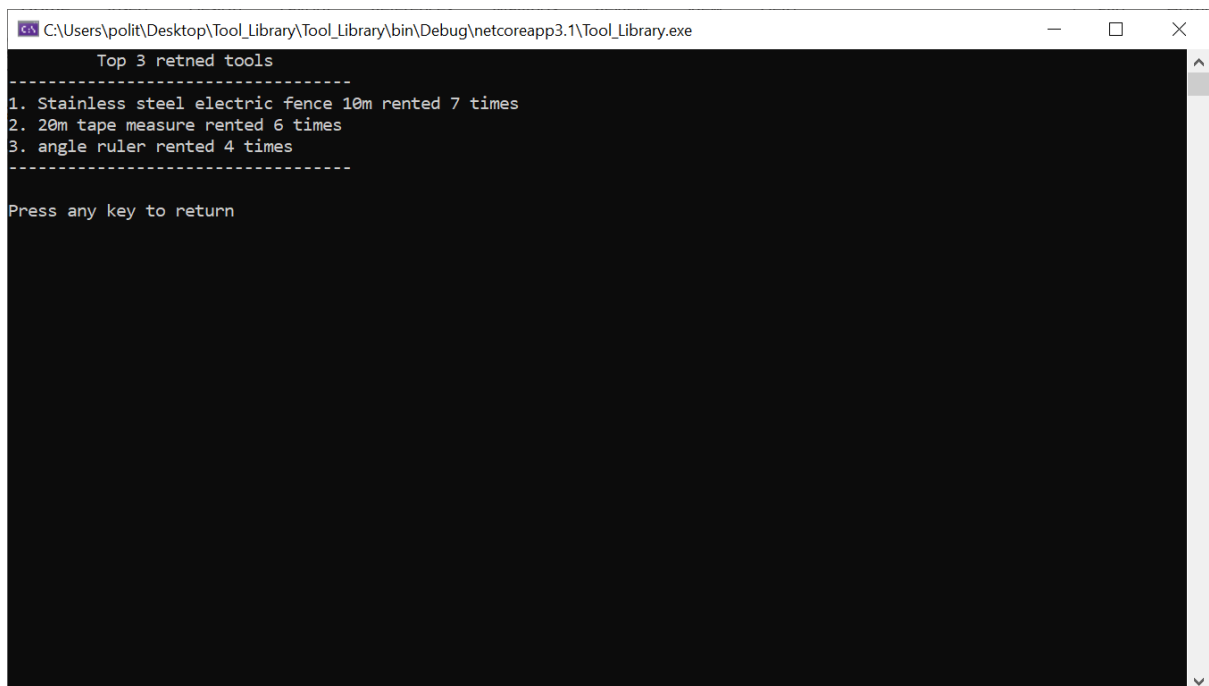
Figure 30 - returning a rented tool



A screenshot of a Windows console window titled "C:\Users\polit\Desktop\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The console output shows the text "Current rented tools:" followed by a list of two items: "- The leveler" and "- Stainless steel electric fence 10m". The list is enclosed in dashed lines. Below the list, the text "Press any key to return" is displayed. A white cursor is visible on the line following the prompt.

```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Current rented tools:
- The leveler
- Stainless steel electric fence 10m
-----
Press any key to return
```

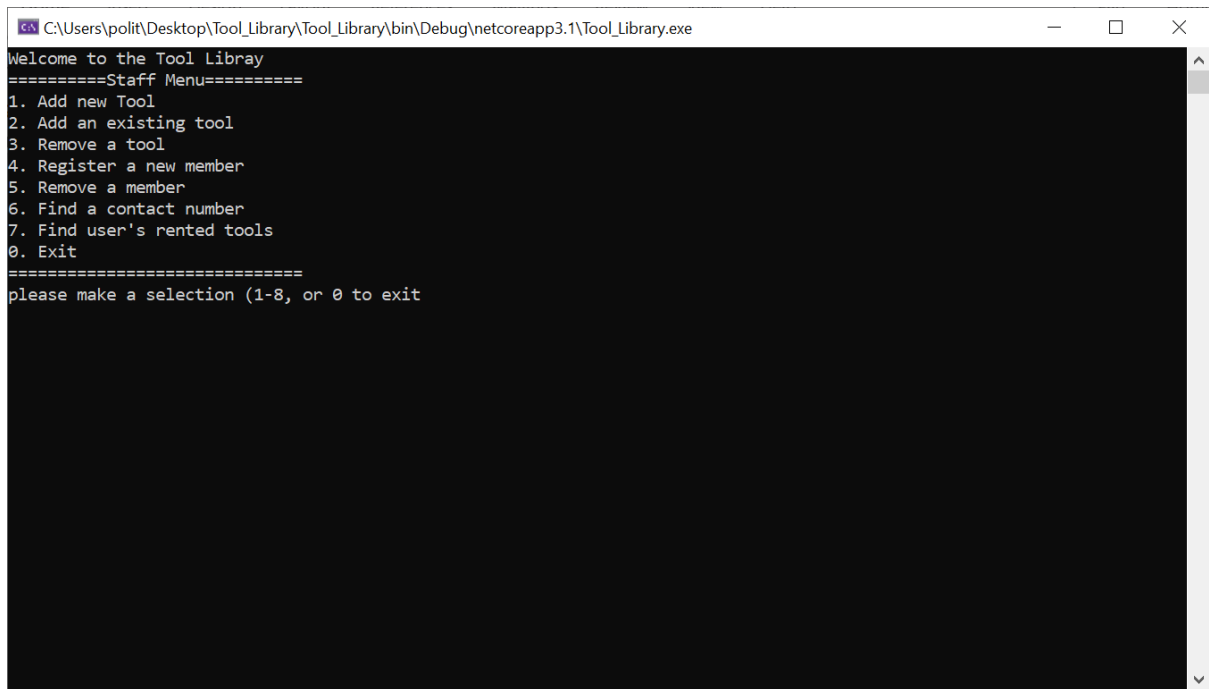
Figure 31 - viewing your rented tools



A screenshot of a Windows console window titled "C:\Users\polit\Desktop\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The console output shows the text "Top 3 rented tools" followed by a list of three items: "1. Stainless steel electric fence 10m rented 7 times", "2. 20m tape measure rented 6 times", and "3. angle ruler rented 4 times". The list is enclosed in dashed lines. Below the list, the text "Press any key to return" is displayed.

```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Top 3 rented tools
-----
1. Stainless steel electric fence 10m rented 7 times
2. 20m tape measure rented 6 times
3. angle ruler rented 4 times
-----
Press any key to return
```

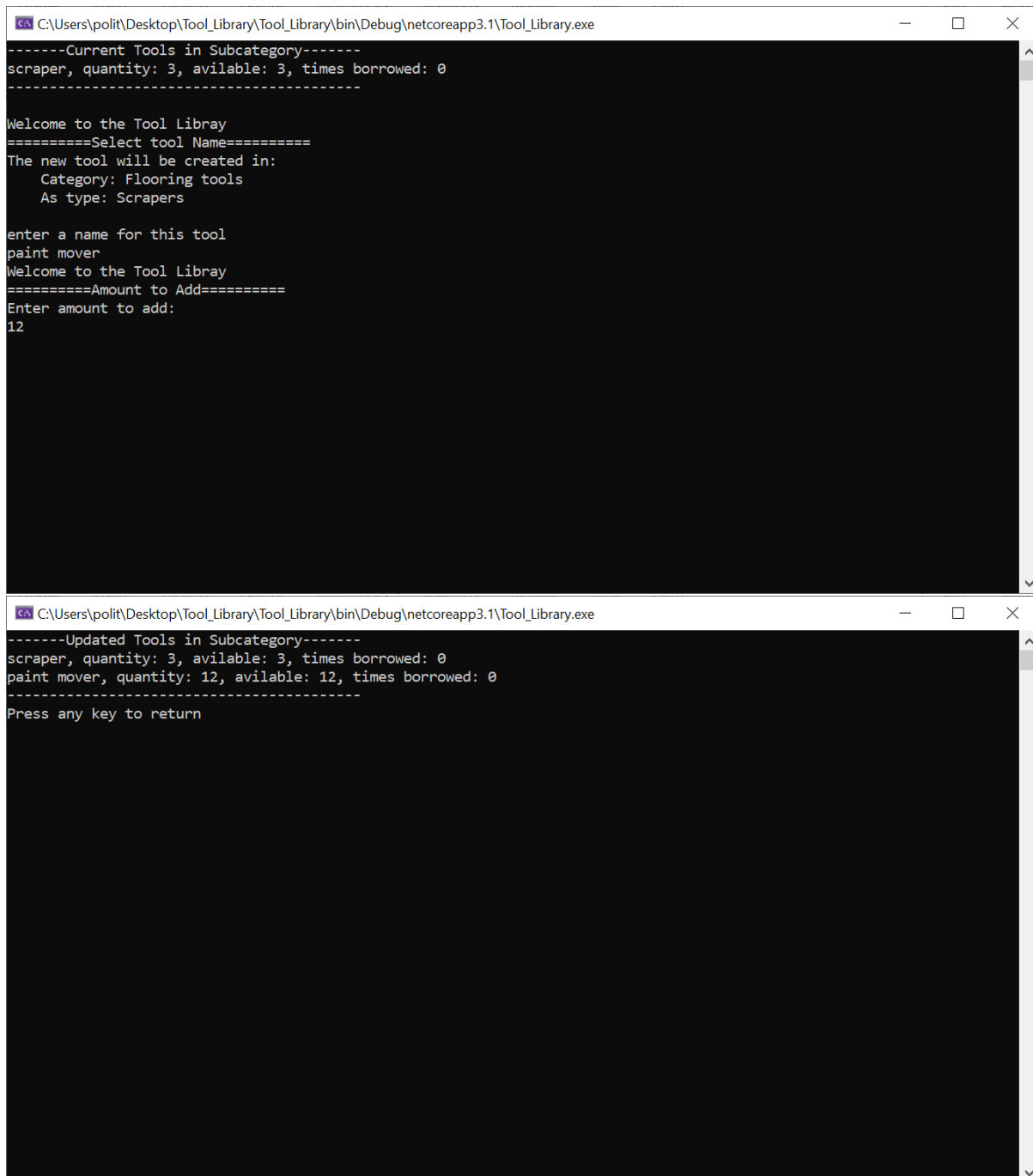
Figure 32 - viewing the top 3 rented tools



The screenshot shows a Windows command prompt window titled "C:\Users\polit\Desktop\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe". The window contains the following text:

```
Welcome to the Tool Libray
=====Staff Menu=====
1. Add new Tool
2. Add an existing tool
3. Remove a tool
4. Register a new member
5. Remove a member
6. Find a contact number
7. Find user's rented tools
0. Exit
=====
please make a selection (1-8, or 0 to exit)
```

Figure 33 - admin main menu



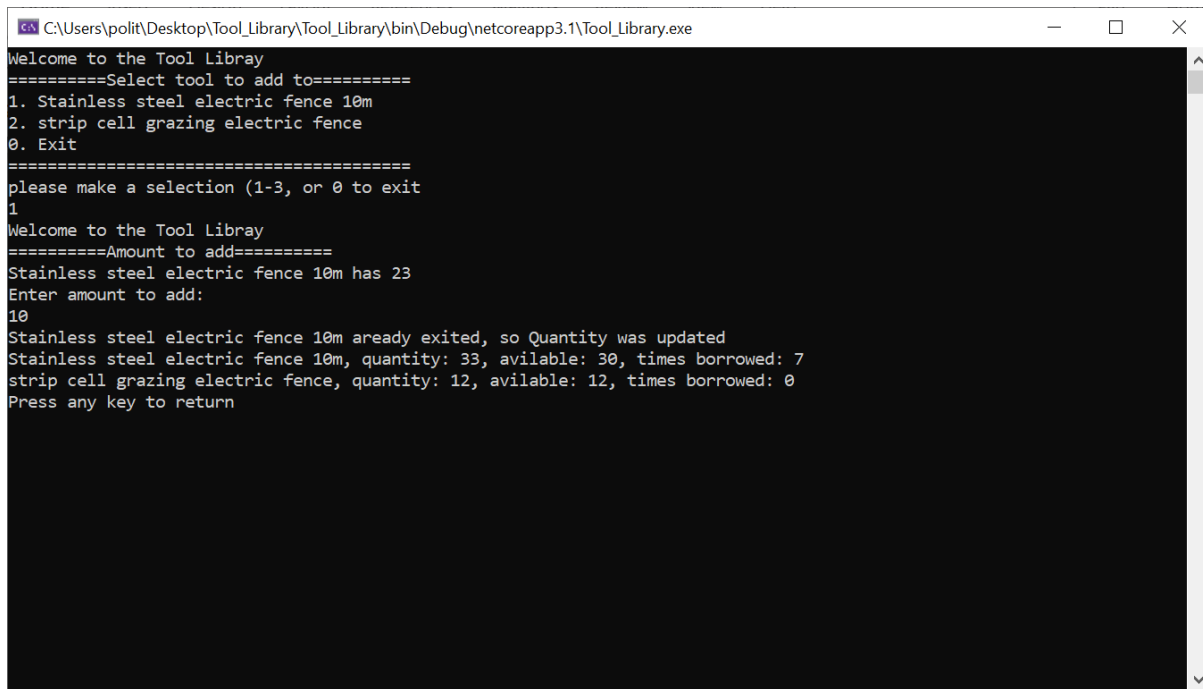
```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
-----Current Tools in Subcategory-----
scraper, quantity: 3, available: 3, times borrowed: 0
-----

Welcome to the Tool Libray
=====Select tool Name=====
The new tool will be created in:
    Category: Flooring tools
    As type: Scrapers

enter a name for this tool
paint mover
Welcome to the Tool Libray
=====Amount to Add=====
Enter amount to add:
12

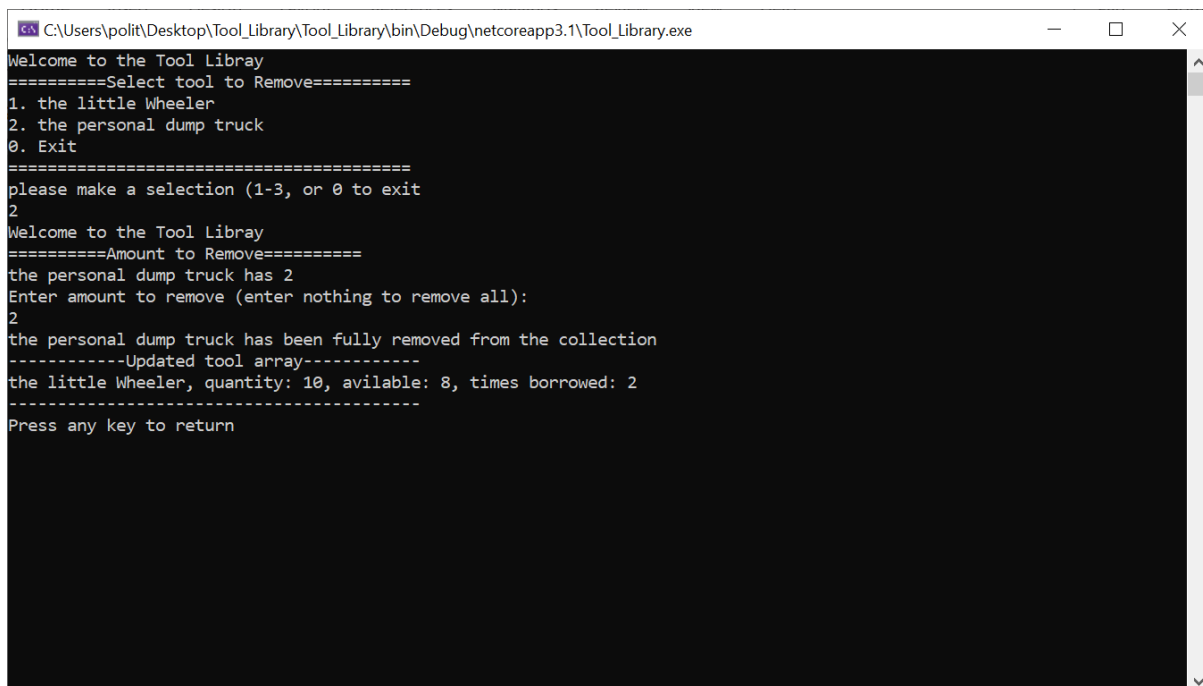
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
-----Updated Tools in Subcategory-----
scraper, quantity: 3, available: 3, times borrowed: 0
paint mover, quantity: 12, available: 12, times borrowed: 0
-----
Press any key to return
```

Figure 34 - adding a new tools from the admin menu



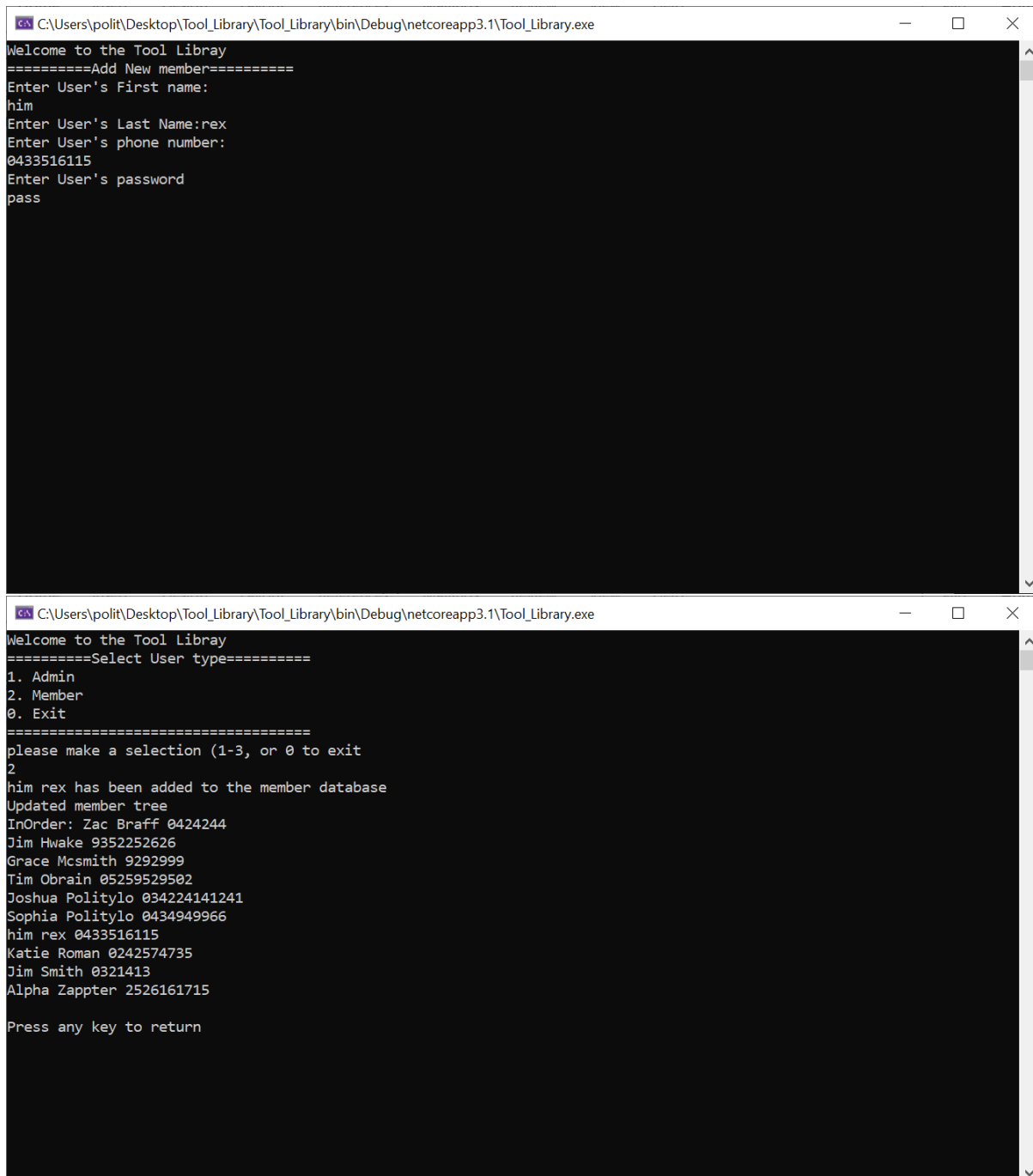
```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Libray
=====Select tool to add to=====
1. Stainless steel electric fence 10m
2. strip cell grazing electric fence
0. Exit
=====
please make a selection (1-3, or 0 to exit
1
Welcome to the Tool Libray
=====Amount to add=====
Stainless steel electric fence 10m has 23
Enter amount to add:
10
Stainless steel electric fence 10m already exited, so Quantity was updated
Stainless steel electric fence 10m, quantity: 33, available: 30, times borrowed: 7
strip cell grazing electric fence, quantity: 12, available: 12, times borrowed: 0
Press any key to return
```

Figure 35 – updating a tools quantity for an existing tool type



```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Libray
=====Select tool to Remove=====
1. the little Wheeler
2. the personal dump truck
0. Exit
=====
please make a selection (1-3, or 0 to exit
2
Welcome to the Tool Libray
=====Amount to Remove=====
the personal dump truck has 2
Enter amount to remove (enter nothing to remove all):
2
the personal dump truck has been fully removed from the collection
-----Updated tool array-----
the little Wheeler, quantity: 10, available: 8, times borrowed: 2
-----
Press any key to return
```

Figure 36 - removing a tool from a tool library using the admin menu



The image consists of two screenshots of a Windows command prompt window running a program named 'Tool\_Library.exe'. The window title is 'C:\Users\poli\Desktop\Tool\_Library\Tool\_Library\bin\Debug\netcoreapp3.1\Tool\_Library.exe'.

The first screenshot shows the initial state of the application. It displays a welcome message and a menu option to 'Add New member'. The user has entered the following information:

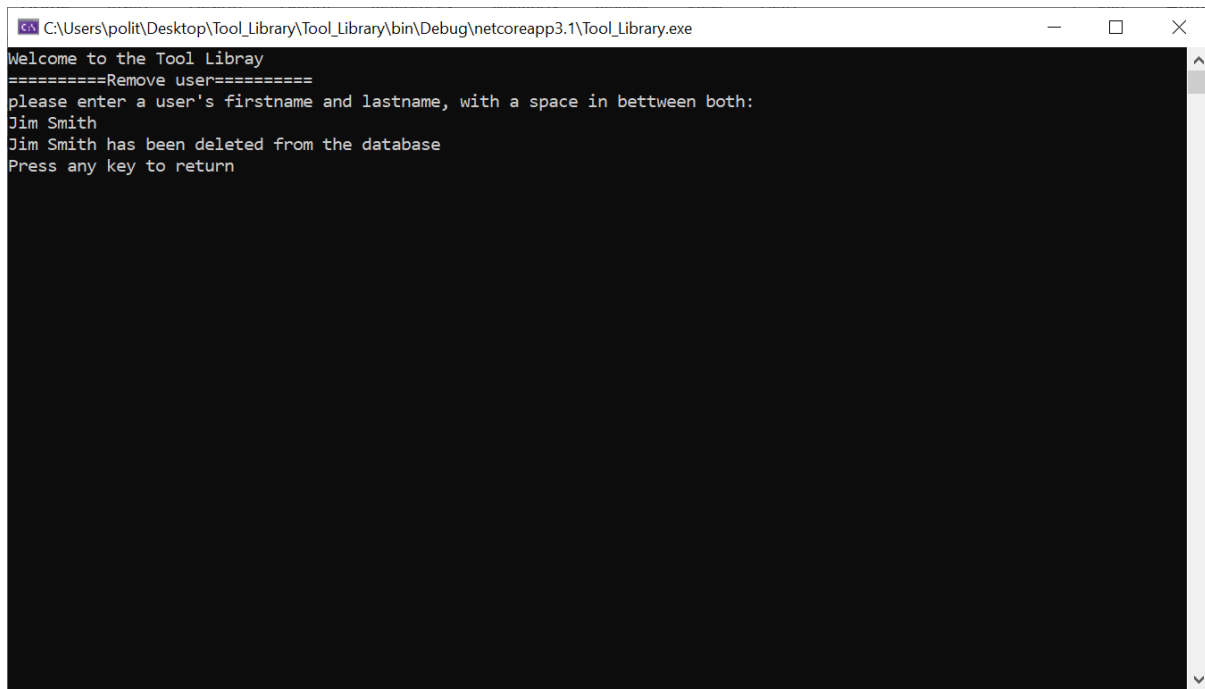
```
Welcome to the Tool Libray
=====Add New member=====
Enter User's First name:
him
Enter User's Last Name:rex
Enter User's phone number:
0433516115
Enter User's password
pass
```

The second screenshot shows the application after the user has selected the 'Admin' option from the 'Select User type' menu. It displays a list of existing members and the newly added user, 'him rex'.

```
Welcome to the Tool Libray
=====Select User type=====
1. Admin
2. Member
0. Exit
=====
please make a selection (1-3, or 0 to exit
2
him rex has been added to the member database
Updated member tree
InOrder: Zac Braff 0424244
Jim Hwake 9352252626
Grace Mcsmith 9292999
Tim Obrain 05259529502
Joshua Politylo 034224141241
Sophia Politylo 0434949966
him rex 0433516115
Katie Roman 0242574735
Jim Smith 0321413
Alpha Zappter 2526161715

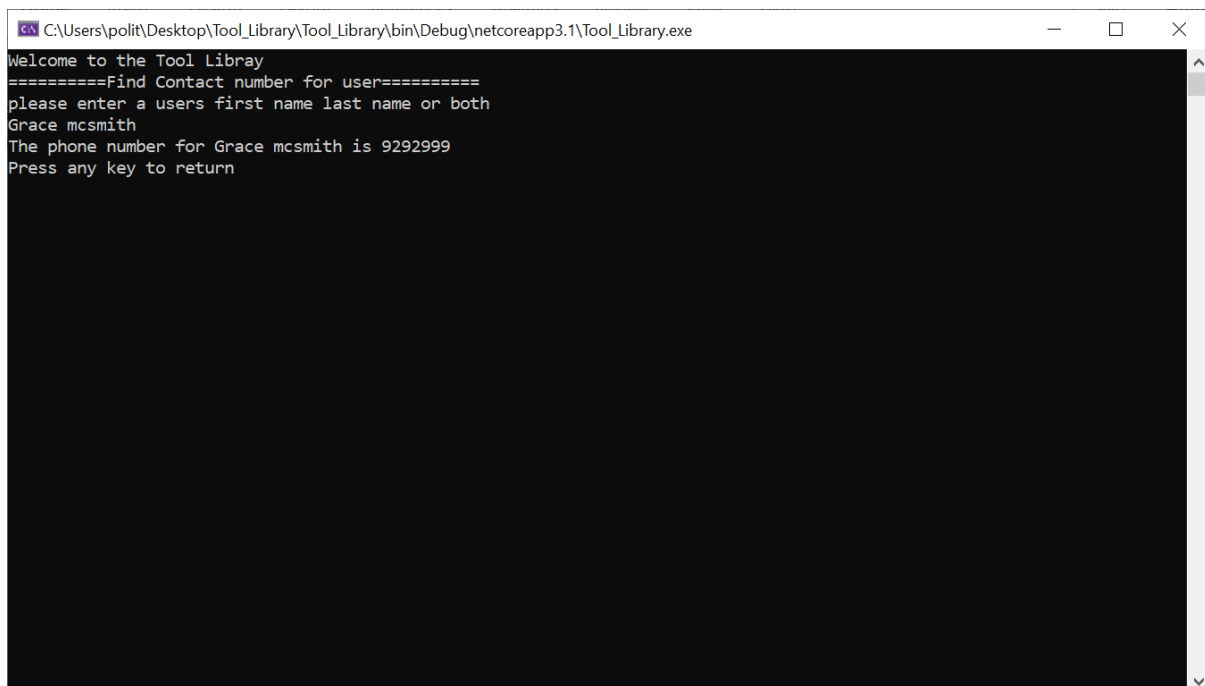
Press any key to return
```

Figure 37- adding a new user to the member tree using the admin menu



```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Libray
=====Remove user=====
please enter a user's firstname and lastname, with a space in between both:
Jim Smith
Jim Smith has been deleted from the database
Press any key to return
```

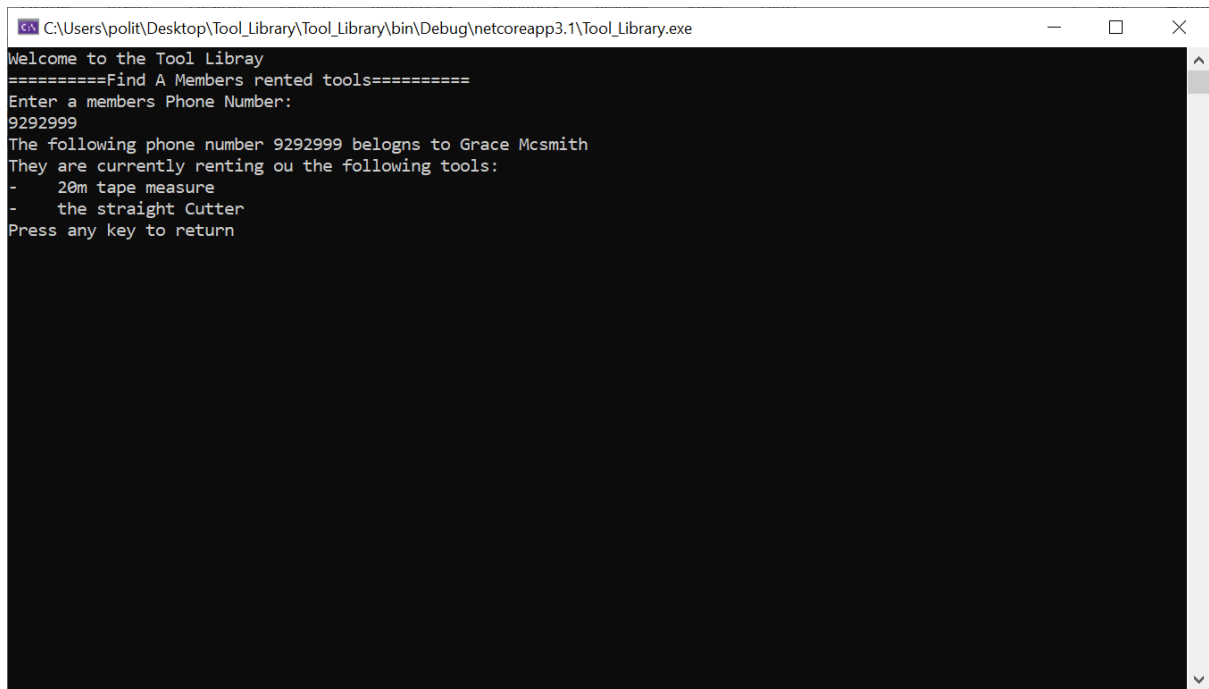
*Figure 38 - removing a member from the member tree*



```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Libray
=====Find Contact number for user=====
please enter a users first name last name or both
Grace mcsmith
The phone number for Grace mcsmith is 9292999
Press any key to return
```

*Figure 39 - finding a user's phone number*





```
C:\Users\polit\Desktop\Tool_Library\Tool_Library\bin\Debug\netcoreapp3.1\Tool_Library.exe
Welcome to the Tool Libray
====Find A Members rented tools====
Enter a members Phone Number:
9292999
The following phone number 9292999 belogns to Grace Mcsmith
They are currently renting ou the following tools:
- 20m tape measure
- the straight Cutter
Press any key to return
```

*Figure 40 - finding a user's rented tools using their phone number*