# Lab 14
# SERACHING
## Dr. John P. Abraham

We can only do a linear search on an unsorted array, but on a sorted array we can do a binary search. In this chapter we will study two different algorithms for searching, linear search and binary search. In linear search we look through an unordered list until we find the item we are looking for. The length of time it takes to look up something using a linear search will depend upon the size of the list. Imagine trying to look up a name in a telephone directory if it was not alphabetized (sorted). If we were to use linear search numerous times, it is possible for us to get lucky once in a while and find it at the beginning of the list. It is also equally possible for us to find it at the end of the list. So, the average time will depend upon the number of the items in the list, such that (n-1)/2, where is the number of items in the list.

In computer science, we use a notation called Big-0. To explain it, I will use an analogy. The other day I saw the news about the most expensive car in the world, Bugatti Veyron, costing $1,700,000. I thought to myself, a person buying that car probably don't worry about gasoline prices; it would be negligible. So, in this case the Big-O would be the price of the car, O(price). When several variables contribute to the calculation of some factor, the one that impacts the most would be the Big-O. In $f(N)=N^4 + N^2 + 10N + 100$, the big-O is $O(N^4)$. The Big-O for linear search is O(N), where as for linear search it is, $O(\log_2 N)$.

Let us do a linear search program first. We need to make sure to exit the search when the item is found. If the list is exhausted, and did not find the item, an error message will be printed out. Suppose you want to search through the names of the month for a particular month and print out its numeric month value (for example, for April print out 4). The program is given below in 14-1.

Program 14-1

```
#include <iostream>
#include <string>
using namespace std;
```

```
//linear search

int main()
{
 int  monthNum;
 bool found=false;
 string oneMonth;
 string month[] = {"","JAN","FEB","MAR","APR","MAY","JUN",

      "JUL","AUG","SEP","OCT","NOV","DEC"};

 cout<<"\nEnter a month name using uppercase three character format
('QUIT' to exit the program) ";
 cin>>oneMonth;
  while (oneMonth !="QUIT")
  {
      monthNum=1;
      while (monthNum <=12 && !found)
      {
        if (oneMonth==month[monthNum])
        {
               cout <<"\nThe number of the month you entered is:
"<<monthNum;
               found=true;
        }
        else monthNum++;
      }
if (monthNum>12) cout<<"\nPlease recheck the spelling!";
 cout<<"\nEnter a month name using uppercase three character format
('QUIT' to exit the program) ";
 cin>>oneMonth;
 found=false;
  }


return 0;
}
```

Program Run 14-1

```
Enter a month name using uppercase three character format ('QUIT' to exit the program)
APR

The number of the month you entered is: 4
Enter a month name using uppercase three character format ('QUIT' to exit the program) JAN

The number of the month you entered is: 1
Enter a month name using uppercase three character format ('QUIT' to exit the program)
DEC
```

```
The number of the month you entered is: 12
Enter a month name using uppercase three character format ('QUIT' to exit the program)
ERR

Please recheck the spelling!
Enter a month name using uppercase three character format ('QUIT' to exit the program)
```

In this program (14-1), a Boolean variable 'found' is initialized to false each time the outer loop is entered. The inner loop compares the oneMonth string with each of the array string elements to look for a match. If matched, the integer value for the month is printed out and the found variable is changed to true. Since the inner loop condition is set to check for the month number <=12 && found==false (same as !found), changing the found to true will exit the inner loop. If the month number is > 12, a match was not found and the user must have spelled the month incorrectly.

Let us now turn to another program. Here, we will make two lists, names and telephone numbers, and search for a name to find the appropriate telephone number. In this program I introduce a new concept, defining a type. Until now we only used types that were provided to us by C++ compiler. Since most other languages have built in string type, I thought it would appropriate for us to define a string data type. A string is made up of an array of characters. **typedef char string[255];** This new type **string** can be used within this program as any other types; we can make any number of variables of this type. This type of string is called a c-string.

Program 14-2

```
/************************************************
Searching -linear
Teaching objective  -Linear Search, unordered list.
By Dr. John Abraham
user defined data type is introduced.
Created for 1380 students
************************************************/

#include <iostream>
using namespace std;

typedef char cstring[15]; // type string defined.

int getNames (cstring names[], cstring tele[]);
int lookup (int,cstring, cstring names[]);

int main ()
```

```
{
      cstring names[10], tele[10], lookfor;
      int n, location;
      n = getNames(names, tele);
      cout << "\n*****************************\n";
      cout << "Enter a name to lookup telephone number ";
      cin >> lookfor;
      while (strcmp(lookfor,"quit")!=0)
      {
            location = lookup (n, lookfor, names);
            if (location > 0)
            cout << "Name found at location " << location
                  << ", Telephone# " << tele[location] << endl;
            else cout << "Name not in list! \n";
            cout << "\nEnter a name to lookup telephone number ";
            cin>>lookfor;
      }
      return 0;
}

int getNames (cstring names[], cstring tele[])
{
      int n=1;
      cout << "Enter a name or 'quit' to stop entry-> ";
      cin >> names[n];
      while (strcmp (names[n], "quit") !=0 && n <10)
      {
            cout << "Enter Telephone for " << names[n] << "->  ";
            cin >> tele[n];
            cout << "--------------------\n";
            n++;
            if (n<10) {
            cout << "Enter a name or 'quit' to stop entry-> ";
            cin >> names[n];}
      }
    return n-1;
}
int lookup (int n,cstring lookfor, cstring names[])
{
      int i=1;
      bool found=false;
      while (!found && i <=n)
            if (strcmp(names[i], lookfor)==0)
                  found = true;
            else i++;

if (found) return i;
else return 0;
}
```

Program Run 14-2

```
Enter a name or 'quit' to stop entry-> chen
```

```
Enter Telephone for chen->  3520
--------------------
Enter a name or 'quit' to stop entry-> abraham
Enter Telephone for abraham->  3550
--------------------
Enter a name or 'quit' to stop entry-> brazier
Enter Telephone for brazier->  3455
--------------------
Enter a name or 'quit' to stop entry-> egle
Enter Telephone for egle->  3518
--------------------
Enter a name or 'quit' to stop entry-> tsai
Enter Telephone for tsai->  7329
--------------------
Enter a name or 'quit' to stop entry-> ng
Enter Telephone for ng->  2472
--------------------
Enter a name or 'quit' to stop entry-> quit

******************************
Enter a name to lookup telephone number egle
Name found at location 4, Telephone# 3518

Enter a name to lookup telephone number tsai
Name found at location 5, Telephone# 7329

Enter a name to lookup telephone number ng
Name found at location 6, Telephone# 2472

Enter a name to lookup telephone number this
Name not in list!

Enter a name to lookup telephone number
```

In program 14-2, the look up function returns the array location where the name was found. The look up function is given the array of names, total number of names in the array and the name to look for. We initialize Boolean variable found to false. The while loop is exited when found or the list is exhausted (i <=n). The strcmp is used to compare two c-strings. Please notice the comparison is different for string objects and c-strings.

As explained in the beginning of this lab, linear lookup is very slow. When working with a small list like this one, the slowness is not noticeable. However, when the list grows it becomes painfully slow. In a sorted telephone directory we can go to the middle of the directory and compare the name there with the name we are looking for. What we are looking for is greater than what we found in the middle, we can abstractly through away the first half of the book. We can go the middle of the remaining pages, and repeat the process. Each time we look up, we are cutting down the number of pages we have to look through otherwise in a linear lookup. This is why the Big-O is $O(\log_2 N)$.

Program 14-3

```
/***********************************************
Searching
Teaching objective  -Binary Search sorted list.
By Dr. John Abraham
Created for 1380 students
***********************************************/

#include <iostream>
using namespace std;

typedef char cstring[15]; // type string defined.

int getNames (cstring names[], cstring tele[]);
void sort (int, cstring names[], cstring tele[]);
int lookup (int,cstring, cstring names[]);
void Display(int n, cstring names[], cstring tele[]);

int main ()
{
      cstring names[10], tele[10];
      int  n;
      n = getNames(names, tele);
      cout << "\n*****************************\n";
      sort(n,  names, tele);
      Display( n,  names,  tele);
      return 0;
}

int getNames (cstring names[], cstring tele[])
{
      int n=1;
      cout << "Enter a name or 'quit' to stop entry-> ";
      cin >> names[n];
      while (strcmp (names[n], "quit") !=0 && n <10)
      {
            cout << "Enter Telephone for " << names[n] << "->  ";
            cin >> tele[n];
            cout << "--------------------\n";
            n++;
            if (n<10) {
```

```
            cout << "Enter a name or 'quit' to stop entry-> ";
            cin >> names[n];}
      }
    return n-1;
}
int lookup (int n,cstring lookfor, cstring names[])
{
      int first=1, last=n, middle;
      bool found=false;
      while (!found && first <= last)
      {
            middle = (first+last)/2;
            if (strcmp(names[middle], lookfor)==0)
                  found = true;
            else if (strcmp(names[middle], lookfor) < 0) first =
middle+1;
            else last = middle-1;
      }

if (found) return middle;
else return 0;
}

void sort(int n, cstring names[], cstring tele[])
{
      int i;
      cstring temp, ttemp;
      bool sorted = false;
      while (!sorted)
      {
            sorted = true;
            for (i=1;i <= n-1; i++)

              if (strcmp (names[i], names[i+1]) >0)
              {
               strcpy (temp,names[i]);       strcpy(ttemp,tele[i]);
               strcpy (names[i],names[i+1]); strcpy(tele[i],
tele[i+1]);
               strcpy (names[i+1],temp);   strcpy(tele[i+1],ttemp);
               sorted = false;
              }
      n--;
      }

}

void Display(int n, cstring names[], cstring tele[])

{

      cstring lookfor;
      int i, location;
      cout << "Sorted names \n";
      for (i=1; i <=n; i++) cout << i <<"  "<< names[i] <<"  "
<<tele[i] <<endl;
      cout << "SEARCH ROUTINE. ENTER quit TO STOP\n";
      cout << "Enter a name to lookup telephone number ";
```

```
      cin >> lookfor;
    while (strcmp(lookfor,"quit")!=0)
      {
            location = lookup (n, lookfor, names);
            if (location > 0)
            cout << lookfor <<" found at location " << location
                  << ", Telephone# " << tele[location] << endl;
            else cout << "Name not in list! \n";
            cout << "Enter a name to lookup telephone number ";
            cin >> lookfor;
      }
}
```

Program Run 14-3

```
Enter a name or 'quit' to stop entry-> tsai
Enter Telephone for tsai->  7229
--------------------
Enter a name or 'quit' to stop entry-> abraham
Enter Telephone for abraham->  3550
--------------------
Enter a name or 'quit' to stop entry-> chen
Enter Telephone for chen->  3520
--------------------
Enter a name or 'quit' to stop entry-> liu
Enter Telephone for liu->  2932
--------------------
Enter a name or 'quit' to stop entry-> fowler
Enter Telephone for fowler->  3453
--------------------
Enter a name or 'quit' to stop entry-> brazier
Enter Telephone for brazier->  3455
--------------------
Enter a name or 'quit' to stop entry-> quit

******************************
Sorted names
1  abraham  3550
2  brazier  3455
3  chen  3520
4  fowler  3453
5  liu  2932
```

```
6  tsai  7229
SEARCH ROUTINE. ENTER quit TO STOP
Enter a name to lookup telephone number chen
chen found at location 3, Telephone# 3520
Enter a name to lookup telephone number brazier
brazier found at location 2, Telephone# 3455
Enter a name to lookup telephone number error
Name not in list!
Enter a name to lookup telephone number
```

The first time we look up, we find the middle of the entire list. First is set to 1 and last is set to the very last index. First + Last divided by two should give us the middle. Compare the name at that middle location with the name we are looking for. If they are the same, we found it. If the name in the middle is smaller than what we are looking for, what we are looking for is in the upper half of the last, we can disregard the first half of the list. So we change the value of the variable first to middle +1. We will be looking again in from middle+1 to the last.

On the other hand, if the name we are looking for is smaller than what we found in the middle, we should disregard the second half and concentrate on searching in the first half. Another words, the first will remain 1, but the last will change to middle-1.

Assignment:
Re-write the program 14-3 to use string class instead of c-string.
.