# Searching

Searching generally means finding one particular item in a list. If you are looking for everyone who worked overtime, or all items that need to be reordered, this is referred to as an *exception report*, not a search. An exception report is straightforward: loop through every item: if it matches the criteria, print it out.

Consider the list below. Only names are shown here, but there is probably a parallel array with some information about these people, such as phone numbers. The information about Karen is in position 0, John in position 1, etc. We would search for Bill and find a match in position 2 of the name array, then print the phone number or other information from position 2.

| | |
|---|---|
| 0 | Karen |
| 1 | John |
| 2 | Bill |
| 3 | Dave |
| 4 | Sam |
| 5 | Jay |

# Sequential Search

The names in the list above are not in alphabetical order. If we want to find Bill, there is no choice but to start with item 0 and loop until we find Bill.
We might use an algorithm such as this:

```
For N = 0 to 5
{  If Name[N] = search_name
     Print Info[N]
} //end of for loop
```

There are two problems with this algorithm:

- It keeps looking after it finds that name;
- It does not print anything if the name is not found

This has a big problem:

```
For N = 0 to 5
{  If Name[N] = search_name
     print Info[N]
   Else print "not found"
} //end of for loop
```

This will print
not found
not found
*Bill's phone number*
not found
not found
not found

A better algorithm is shown below:

```
Found =-1
For N = 0 to 5
{  If Name[N] = search_name
     Found = N
} //end of for loop
If Found = -1
  Print "Not found"
Else
  Print Info[Found]
```

This solves the problem of printing not found, but the algorithm still continues after finding the item. In the next algorithm, the For loop is changed to a While loop.

```
Found =-1
N = 0
While N < 6
{  If Name[N] = search_name
     Found = N
   N = N +1
} //end of for loop
If Found = -1
  Print "Not found"
Else
  Print Info[Found]
```

Once we have the While loop, we can add to the while condition:

```
Found =-1
N = 0
While N < 6 And Found = -1
{  If Name[N] = search_name
     Found = N
   N = N +1
} //end of for loop
If Found = -1
  Print "Not found"
Else
  Print Info[Found]
```

**Experiment**: Before you look at actual code on the next page, try implementing a small array of names and some piece of information. (Year born, phone, etc.). When you find the person, print the information.

It is a very good idea to get a small program working first, add the search when you know the rest is working.

```
 0 //Initialize array and print
 1 #include <iostream.h>
 2 #include <string.h>
 3 void printAll(char names[][10], int year[], int count)
 4 //receive array of names, array of years, # in array
 5 { for(int n=0;n<count;n++)
 6     cout<<names[n]<<"\t"<<year[n]<<"\n";
 7 } //printAll
 8
 9 void main()
10 { char names[20][10]= {"Karen", "John", "Bill", "Dave",
11     "Sam", "Jay"};
12   int year[20]={1955,2000, 1972,1976, 1949,1965};
13   int count = 6; //room in the array for 20 people,
14                  //but there are actually only 6
15   printAll(names,year,count);
16 } //main
```

**Explanation**: There is room in the array for 20 people, but there are only 6 actually there. Arrays are usually a bit bigger than you think you need, just in case! It is important however, that you only print those names that are actually in the list. In this program we know that there are 6, but if you read these in from a file, until eof, you would not know the count beforehand.

1,2: We always need either iostream.h or fstream.h. We will need string.h in the next part when we compare strings.

3: printAll receives the 2 arrays and the count. The Size of the array is not necessary, but the size of each string is. C++ can not know where each name begins unless it is given the size of each element.

10: An initial list is given to get started with the program. Later we may modify this so that the names are read from a file. It is important to test one piece at a time.

15: The only thing in main right now is the call to printAll passing the array of names, years, and count.

The program below prints out the list of names then searches for one name.

```
 0 //Initialize array and print
 1 #include <iostream.h>
 2 #include <string.h>
 3 void search(char names[][10], int year[], int count)
 4 { char lookfor[10];
 5   cout<<"Enter the name to look for:";
 6   cin.getline(lookfor,10);
 7   int found=-1;
 8   int n=0;
 9   while (n<count && found==-1)
10   { if (strcmp(names[n],lookfor)==0)
11       found=n;
12     n++;
13   } //while
14   if (found==-1)
15               cout<<lookfor<<" not found.\n";
16   else
17    cout<<lookfor<<" was born in "<<year[found]<<".\n";
18 } //search
19
20 void printAll(char names[][10], int year[], int count)
21 //receive array of names, array of years, # in array
22 { for(int n=0;n<count;n++)
23             cout<<names[n]<<"\t"<<year[n]<<"\n";
24 } //printAll
25
26 void main()
27 { char names[20][10]= {"Karen", "John", "Bill", "Dave",
28     "Sam", "Jay"};
29   int year[20]={1955,2000, 1972,1976, 1949,1965};
30   int count = 6; // room in the array for 20 people,
31                  //but there are actually only 6
32   printAll(names,year,count);
33   search(names,year,count);
34 } //main
```

**Experiment**: Add each of the following one at a time:

- Print the age of the person. Ask the current year just once, in main!
- Read the names and years from a file.
- Make a menu driven program that gives a choice of print list, print all of the children (12 and under), print all of the senior citizens(60 and older), look up one person.
- Add another piece of information for each person. Adding another piece of information means adding another array, passing the additional array to the functions, and adding it to the file.

## Binary Search

Consider the list below. The names are in alphabetical order. If the list of names is in alphabetical order we can use a much faster algorithm.

| | |
|---|---|
| 0 | Bill |
| 1 | Dave |
| 2 | Jay |
| 3 | John |
| 4 | Karen |
| 5 | Sam |

Let's suppose I am thinking of a number from 1 to 100. You could guess the number using a sequential search. Is it 1? Is it 2? Is it 3? Guessing the number sequentially it would, on average, take you about 50 guesses before you found it.
Usually, that guessing game is done using a binary search:
Is it 50? No, higher.
Is it 75? No, lower.

Lets look at the algorithm for this:
At the beginning, you know that the lowest value is 1 and the highest is 100. Your first guess is the average of low and high, or 50.

| Low | High |
|---|---|
| 1 | 100 |

After you guess 50 and I say No, higher, you change these values. You now know that the lowest the number can possibly be is 51.

| Low | High |
|---|---|
| 51 | 100 |

Your next guess is the average of low and high or 75. I say No, lower. You now know that the highest the number can possibly be is 74.

| Low | High |
|---|---|
| 51 | 74 |

You will continue to make guesses; each guess is the average of low and high.
If your guess is too low, you set low to guess +1. If your guess is too high, you set high to guess-1.

Let's try it with the list of names.

| | |
|---|---|
| 0 | Bill |
| 1 | Dave |
| 2 | Jay |
| 3 | John |
| 4 | Karen |
| 5 | Sam |

To start low is 0 and High is 5. We will search for Karen. The first guess is (0+5)/2=2 (Integer division!) We compare Jay and Karen. Karen comes after Jay, so we change low to guess+1 , or 3. Low is 3. High is 5. Our next guess is (3+5)/2 = 4. Eureka! There's Karen!

Now try searching for Debbie. The first guess is (0+5/2)=2. Debbie comes before Jay, so we change high to guess –1. Low is 0. High is 1. Our next guess is (0+1)/2=0. Debbie comes after Bill, so we change low to guess+1. Low is 1. High is 1. Our next guess is 1. Debbie comes after Dave, so we change low to guess+1. Low is 2. High is 1. That's impossible, so we know that when low > high the person is not on the list.

```
 0 //Binary Search
 1 #include <iostream.h>
 2 #include <string.h>
 3 void search(char names[][10], int year[], int count)
 4 { char lookfor[10];
 5   cout<<"Enter the name to look for:";
 6   cin.getline(lookfor,10);
 7   int found=-1;
 8   int low=0;
 9   int guess;
10   int high=count;
11   while (low<= high && found==-1)
12   { guess=(low+high)/2;
13     if (strcmp(names[guess],lookfor)==0)
14       found=guess;
15     else if (strcmp(names[guess],lookfor)<0)
16       low=guess+1;
17     else
18       high=guess-1;
19   } //while
20   if (found==-1)
21     cout<<lookfor<<" not found.\n";
22   else
23     cout<<lookfor<<" was born in "<<year[found]<<".\n";
24 } //search
25
26 void printAll(char names[][10], int year[], int count)
27 //receive array of names, array of years, # in array
28 { for(int n=0;n<count;n++)
```

```
29    cout<<names[n]<<"\t"<<year[n]<<"\n";
30 } //printAll
31
32 void main()
33 { char names[20][10]={"Bill", "Dave", "Jay", "John",
34     "Karen", "Sam"};
35   int year[20]={1972,1976,1965,2000,1955,1949 };
36   int count = 6; // room in the array for 20 people,
37                  //but there are actually only 6
38   printAll(names,year,count);
39   search(names,year,count);
40 } //main
```

You can not do a binary search unless the array is sorted! If the array is sorted this is a much faster search.
Consider that there are 100 names on the list. Sequentially it will take about 50 guesses to reach it. (Sometimes it will be early in the list, sometimes late, but on average 50.)
Now consider the binary search. After the first guess there are only 50 names where it could be. After the 2nd guess only 25, after the 3rd only 12. after the 4th only 6, after the 5th only 3. You should be able to find the name in only 6 tries.

**Experiment**: Have a friend think of a number from 1 to 1 million. Tell them that you can guess it in 20 guesses of less. Or have a friend pick a name from the WHITE pages of the phone book. Do the binary search by opening it up to the middle and guessing the name on that page. Just do it by eye, splitting the number of pages that you are holding between your fingers in half each time. After just 5 or 6 guesses, you will know what page the name is on. Would this work with the YELLOW pages?

.