Information retrieval is one of the important applications of computers. When searching we try to find a record that contains information associated with the key. To write our search program in C, we establish some notations. What we have called records will be C structures, and the C type these records we will call Item_type. One of the members (or components) of each item will be denoted key and have a type called Key_type.

**Sequential search.** Beyond doubt, the simplest way to do a search is to begin at one end of the list and scan down it until the desired key is found or the other end is reached. Here is example how sequential search can be organized.

```c
#include <iostream>
#include <conio>
#include <stdio>

const int MAXARRAY = 5;

typedef char string[];

typedef struct Item_Tag{
    int Id;
    string Name;
} Item_Type;
```
Item_Type Info[MAXARRAY];
int Key;
int location;

int Search();

int main()
{
    for (location = 0; location < MAXARRAY; location++){
        cout << "Info[" << location << "]\Id : ";
        cin >> Info[location].Id;
        cout << endl;
        cout << "Info[" << location << "]\Name : ";
        cin >> Info[location].Name;
        cout << endl;
    }
    cout << "Enter key: ";
    cin >> Key;
    cout << '/n';
    if (Search() == -1)
        printf("Enter correct value\n");
    else
        printf("Location is %d\n", location);
    getch();
    return 0;
}

int Search()
{
    for (location = 0; location < MAXARRAY; location++)
        if (Info[location].Id == Key)
            return location;
    return -1;
}

Sample run:
Info[0].Id = 987234
Info[0].Name = Ali
Info[1].Id = 987235
Info[1].Name = Hasan
Info[2].Id = 987236
Info[2].Name = Huseyin
Info[3].Id = 987237
Info[3].Name = Aylin
Info[4].Id = 987238
Info[4].Name = Enver
Key = 987236
Location is 2

**Binary search**

Sequential search is easy to write and efficient for short lists, but a disaster for long ones. If the values in the key field are arranged in ascending (or descending) order we can
use more efficient method, which is called **binary search** method.

Note that binary search is realized as function `bserach` function in `stdlib.h`. However, we illustrate the main ideology of the method using the following C program.

```c
#include <iostream>
#include <conio>
#include <stdio>

const int MAXARRAY = 5;

typedef char string[];

typedef struct Item_Tag{
    int Id;
    string Name;
} Item_Type;

Item_Type Info[MAXARRAY];
int Key;
int location, top, middle, bottom;

int Binary_Search();
int main()
{
    printf("ID values should be arranged in ascending order!!!\n\n");

    for (location = 0; location < MAXARRAY; location++){
        cout << "Info[" << location << "]\'.Id : ";
        cin >> Info[location].Id;
        cout << "\n";
        cout << "Info[" << location << "]\'.Name : ";
        cin >> Info[location].Name;
        cout << "\n";
    }

    cout << "Enter key: ";
    cin >> Key;
    cout << "\n";
    if (Binary_Search() == -1)
        printf("Key has been not found!\n");
    else
        printf("Key %d is located in position %d\n", Key, top);
    getch();
    return 0;
}
int Binary_Search()
{
    top = MAXARRAY -1;
    bottom = 0;
```
while (top > bottom){
    middle = (top + bottom)/2;
    if (Info[middle].Id < Key)
        bottom = middle +1;
    else
        top = middle;
}

if (top == -1)
    return -1;
if (Info[top].Id == Key)
    return top;
else
    return -1;
}

Sample run:
ID values should be arranged in ascending order!!!

Info[0].Id = 987234
Info[0].Name = Ali
Info[1].Id = 987235
Info[1].Name = Hasan
Info[2].Id = 987236
Info[2].Name = Huseyin
Info[3].Id = 987237
Info[3].Name = Aylin
Info[4].Id = 987238
Info[4].Name = Enver
Key = 987236

Key 987236 is location in position 2

Comparison trees

The comparison tree (also called decision tree or search tree) of an algorithm is obtained by tracing through the action of the algorithm, representing each comparison of keys by a vertex of the tree (which we draw as a circle). Inside the circle we put the index of the key against which we are comparing the target key. Branches (lines) drown down from the circle represent the possible outcomes of the comparison and are labeled accordingly. When the algorithm terminates, we put either F (for failure) or the location where the target is found at the end of the appropriate branch, which we call a leaf, and draw as a square. Leaves are also sometimes called end vertices or external vertices of the tree. The remaining vertices are called the internal vertices of the tree. The comparison tree for sequential search is especially simple; it is drown in the following figure.
The number of comparisons done by an algorithm in a particular search is the number of internal vertices traversed in going from the top of the tree, which is called its root, down the appropriate path to a leaf. The number of branches traversed to reach a vertex from the root is called the level of the vertex. Thus the root itself has level 0, the vertices immediately below it have level 1, and so on. The largest level that occurs is called the height of the tree.