1 Searching Concepts

- Want to find an item quickly.
- Many search methods require data to be sorted.
- Operations required for searching: comparisons.

1.1 Common Searching Techniques

Binary Searches through a reduced set of the array. Fairly easy to implement. Efficient in use.

Linear Very simple to implement. Can be very inefficient in use. Can improve the efficiency by searching in the reverse direction or caching previous results.

Many of these techniques will be covered in detail in later Computer Science courses.

1.2 Linear Search

```
/*  LinearSearch()
 *
 *  Searches for the value, keyVal, in an array.
 *  If successful, the index is returned.
 *  If unsuccessful, -1 is returned.
 */

int LinearSearch( const int A[], const int nMax,
                  const int keyVal )
{
    for( int i = 0 ; i < nMax ; i++ )
    {
        if( A[i] == keyVal )
            return i;
    }

    return  -1;
}
```

1.3 Binary Search

```
/*  BinarySearch
 *
 *  Searches for the value, keyVal, in a sorted
 *  array (values must be low to high).
 *  If successful, the index is returned.
 *  If unsuccessful, -1 is returned.
 */

int BinarySearch( const int A[], const int nMax,
                  const int keyVal )
{
    int  first = 0;
    int  last  = nMax-1;
    int  middle;

    if( (keyVal < A[first]) || (keyVal > A[last]) )
        return -1;

    while( first <= last ) {
        middle = (first+last)/2;

        if( keyVal == A[middle] )
            return  middle;
        else if( keyVal > A[middle] )
            first = middle + 1;
        else  // keyval is less than A[middle]
            last = middle - 1;
    }
    return  -1;
}
```

2 Sorting Concepts

- Want array elements ordered in some way.
- Typically non-decreasing (may have multiple values).
- Operations required for sorting: comparisons, swapping element(s).

2.1 Common Sorting Techniques

Bubble sort Iterate through the array examining adjacent pairs of elements. If necessary, swap them to put them in the desired order (Brick Sort).

Selection sort Iterate through the array putting the ith smallest element in the ith location.

Insertion sort Iterate through the array placing the ith element with respect to the $i-1$ previous elements.

Quick sort Partition the list into smaller lists such that every element in the left partition is less than or equal to the right partition. Repeat Quicksort process on the partitions—frequently done using recursion.

Merge sort Useful for sorting very large amounts of data. The basic algorithm:

1. Split the file into smaller files.
2. Sort the smaller files
3. Merge the sorted files

Many of these techniques will be covered in detail in later Computer Science courses.

2.2 Selection Sort

```
/*  SelectionSort()
 *
 *  Perform a selection sort on an array
 */

void SelectionSort( int A[], const int nA )
{
    for( int i = 0 ; i < nA ; ++i )
    {
        int  low = i;
        for( int j = i + 1 ; j < nA-1 ; ++j )
        {
            if( A[j] < A[low] )
                low = j;
        }
        if( i != low )
        {
            //Swap( A[low], A[i] );
            iTmp   = A[low];
            A[low] = A[i];
            A[i]   = iTmp;
        }
    }
}
```

Performs poorly for sorted/nearly sorted array. Can be improved by adding a swap flag, as illustrated in the Bubble Sort code that follows.

2.3 Bubble Sort

```
const int TRUE  = 1;
const int FALSE = 0;

/*  BubbleSort
 *
 *  Perform a bubble sort on an array
 */

void BubbleSort( int A[], const int nA )
{
    int  iTmp;
    int  swapFlag = TRUE;

    for( int i = 0 ; i < nA && swapFlag == TRUE ; ++i )
    {
        swapFlag = FALSE;
        for( int j = 0 ; j < nA - i ; j++ ) {
            if( A[j] > A[j+1] )   // swap them
            {
                swapFlag = TRUE;
                //Swap( A[j], A[j+1] );
                iTmp   = A[j];
                A[j]   = A[j+1];
                A[j+1] = iTmp;
            }
        }
        //  Show progress
    }
}
```

2.4 CD Database Program

One simple implementation of the CD Database. This program illustrates a number of fairly complicated tasks:

- Sorting and searching an array structure.
- String manipulation

2.4.1 Sample CD Database Input file

```
Beach Boys
Spirit of America
Surf
15.95
Gershwin
Rhapsody in Blue
Classical
8.99
Led Zeppelin
Physical Graffiti
Rock
17.99
Fleetwood Mac
The Dance
Rock
13.99
Chris Isaak
Heart Shaped World
Rock
13.99
```

2.4.2 Output file generated by CD Database

```
Initial contents of CD data base:

Beach Boys Spirit of America
Gershwin Rhapsody in Blue
Led Zeppelin Physical Graffiti
Fleetwood Mac The Dance
Chris Isaak Heart Shaped World

---------------------

CD data base contents after sorting:


Beach Boys Spirit of America
Chris Isaak Heart Shaped World
Fleetwood Mac The Dance
Gershwin Rhapsody in Blue
Led Zeppelin Physical Graffiti
---------------------

Beach Boys found at location:  1
Who not found
```

```
/*  CDDB.cpp
 *
 *  Sample CD data base program
 *
 *  Bruce M. Bolden
 *  November 17, 1997
 *  Updated:  June 30, 1998
 *
 *  http://www.cs.uidaho.edu/~bruceb/
 * ------------------------------------------------------------
 */

#include <iostream.h>
#include <iomanip.h>
#include <fstream.h>
#include <ctype.h>
#include <string.h>

   //  CD Info record structure
const int MAX_ARTIST_LENGTH = 15;
const int MAX_TITLE_LENGTH = 40;
const int MAX_CATEGORY_LENGTH = 10;

struct st_CD_Info
{
   char  artist[MAX_ARTIST_LENGTH];
   char  title[MAX_TITLE_LENGTH];
   char  category[MAX_CATEGORY_LENGTH];
   double  cost;
};
typedef struct st_CD_Info CDInfo;

const int MAX_DB_ITEMS = 20;

CDInfo currDB[MAX_DB_ITEMS];
```

```
    //  function prototypes
int  ReadCDDB( char *dbName, const int maxCDs );
void ShowCDs( ofstream& outFile, const int nCDs );

void SortByArtist( const int nCDs );
void SwapCDRecords( const int i, const int j );
void SwapString( char *s1, char *s2 );
void SwapDouble( double& d1, double& d2 );

int SearchByArtist( const char *artistName, const int nMax );
void ShowSearchResult( ofstream& o,
                       const char *aName, const int i );
```

```
int main()
{
    ofstream outFile( "CDDB.out", ios::out );
    if( !outFile ) {
        cerr << "Unable to open: \"" << "CDDB.out" << "\"" << endl;
        exit( -1 );
    }

    int nCDs = ReadCDDB( "CDDB.dat", MAX_DB_ITEMS );

    if( nCDs > 0 ) {
        outFile << "Initial contents of CD data base:\n" << endl;
        ShowCDs( outFile, nCDs );

        SortByArtist( nCDs );

        outFile << "CD data base contents after sorting:\n" << endl;
        ShowCDs( outFile, nCDs );

        int  i;      //  Index returned by search operation
        i = SearchByArtist( "Beach Boys", nCDs );
        ShowSearchResult( outFile, "Beach Boys", i );

        i = SearchByArtist( "Who", nCDs );
        ShowSearchResult( outFile, "Who", i );
    }

    outFile.close();
}
```

```
/*  Read CD Database
 */
int ReadCDDB( char *dbName, const int maxCDs )
{
        //  Open data base file
    ifstream inCDFile( dbName );
    if( !inCDFile )
    {
        cerr << "Unable to open: \"" << dbName << "\"" << endl;
        return  -1;
    }

        //  Read data base file
    int  i = 0;
    bool done = false;
    const int MAX_LINE = 80;
    char tmpLine[MAX_LINE];

    while( !done ) {
        inCDFile.getline( currDB[i].artist, MAX_ARTIST_LENGTH );
        inCDFile.getline( currDB[i].title, MAX_TITLE_LENGTH );
        inCDFile.getline( currDB[i].category, MAX_CATEGORY_LENGTH );
        inCDFile >> currDB[i].cost;
        inCDFile.getline( tmpLine, MAX_LINE );

        char ch; //  skip newline
        //inCDFile.get( ch );
        if( !inCDFile.good() && i < maxCDs )
            done = true;

        ++i;
    }

    inCDFile.close();

    return i;
}
```

```
/* Show CDs
 */
void ShowCDs( ofstream& o, const int nCDs )
{
    for( int i = 0 ; i < nCDs ; i++ )
    {
        o << currDB[i].artist << "\t";
        o << currDB[i].title;
        o << endl;
    }

    o << "----------------------\n" << endl;
}
```

```
/*  SortByArtist
 *
 *  Perform a bubble sort on the CD data base.
 */
void SortByArtist( const int nCDs )
{
    int  iTmp;
    int  swapFlag = TRUE;

    cout << "\nIn SortByArtist():" << endl;

    for( int i = 0 ; i < nCDs && swapFlag == TRUE ; i++ )
    {
        swapFlag = FALSE;
        for( int j = 0 ; j < nCDs - i - 1 ; j++ )
        {
            if( strcmp(currDB[j].artist,
                       currDB[j+1].artist) > 0 )   // swap them
            {
                swapFlag = TRUE;
                SwapCDRecords( j, j+1 );
            }
        }
    }

    cout << "Leaving SortByArtist()\n" << endl;
}
```

```
/*  SwapCDRecords
 *
 *  Swap the contents of two CD records.
 */
void SwapCDRecords( const int i, const int j )
{
    CDInfo  tmpRec;

    tmpRec = currDB[i];
    currDB[i] = currDB[j];
    currDB[j] = tmpRec;

    /*
    SwapString( currDB[i].artist, currDB[j].artist );
    SwapString( currDB[i].title,  currDB[j].title );
    SwapString( currDB[i].category, currDB[j].category );
    SwapDouble( currDB[i].cost, currDB[j].cost );
    */
    /*
    CDInfo  tmpRec;

    strcpy( tmpRec.artist,   currDB[i].artist   );
    strcpy( tmpRec.title,    currDB[i].title    );
    strcpy( tmpRec.category, currDB[i].category );
    tmpRec.cost = currDB[i].cost;

    strcpy( currDB[i].artist,   currDB[j].artist   );
    strcpy( currDB[i].title,    currDB[j].title    );
    strcpy( currDB[i].category, currDB[j].category );
    currDB[i].cost = currDB[j].cost;

    strcpy( currDB[j].artist,   tmpRec.artist   );
    strcpy( currDB[j].title,    tmpRec.title    );
    strcpy( currDB[j].category, tmpRec.category );
    currDB[j].cost = tmpRec.cost;
    */
}
```

```
/*  SwapString
 *
 *  Swap two strings.
 */
void SwapString( char *s1, char *s2 )
{
    char  tmpStr[MAX_TITLE_LENGTH];     //  longest string

    strcpy( tmpStr, s1 );
    strcpy( s1, s2 );
    strcpy( s2, tmpStr );
}



/*  SwapDouble
 *
 *  Swap two doubles.
 */
void SwapDouble( double& d1, double& d2 )
{
    double dTmp;

    dTmp = d1;
    d1 = d2;
    d2 = dTmp;
}
```

```
/*  SearchByArtist
 *
 *  Searches for the value, artistName, in the CD data base.
 *  If successful, the index is returned.
 *  If unsuccessful, -1 is returned.
 */
int SearchByArtist( const char *artistName, const int nMax )
{
    for( int i = 0 ; i < nMax ; i++ )
    {
        if( strcmp( currDB[i].artist, artistName) == 0 )
            return i;
    }

    return  -1;
}



/*  ShowSearchResult
 *
 *  Show results of a search for a given artistName
 */
void ShowSearchResult( ofstream& o,
                       const char *aName, const int i ) {
    if( i < 0 )
    {
        o << aName << " not found";
    }
    else
    {
        o << aName << " found at location:  " << i;
    }
    o << endl;
}
```