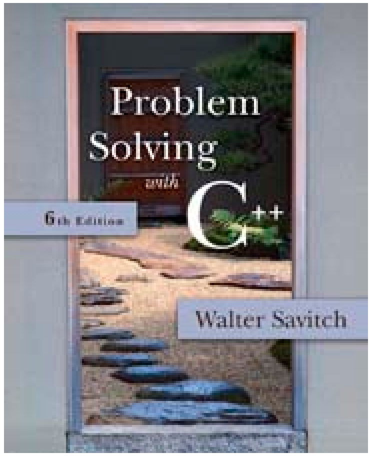
APS105: Lecture 28

Wael Aboelsaadat

wael@cs.toronto.edu

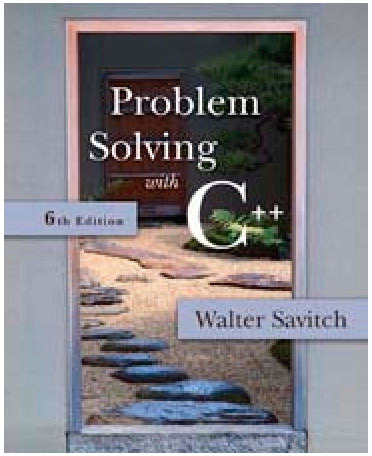
http://ccnet3.utoronto.ca/20079/aps105h1f/

Acknowledgement: These slides are a modified version of the text book slides as supplied by Addison Wesley



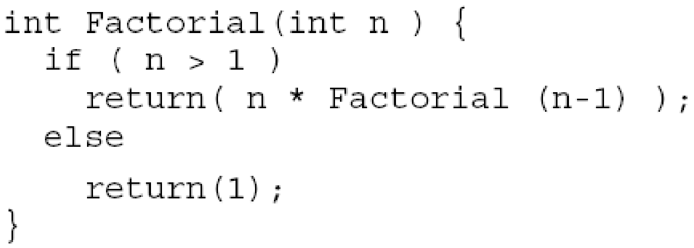
Chapter 14

Recursion



Factorial using Recursion

$$N! = 1 * 2 * \ldots * N$$

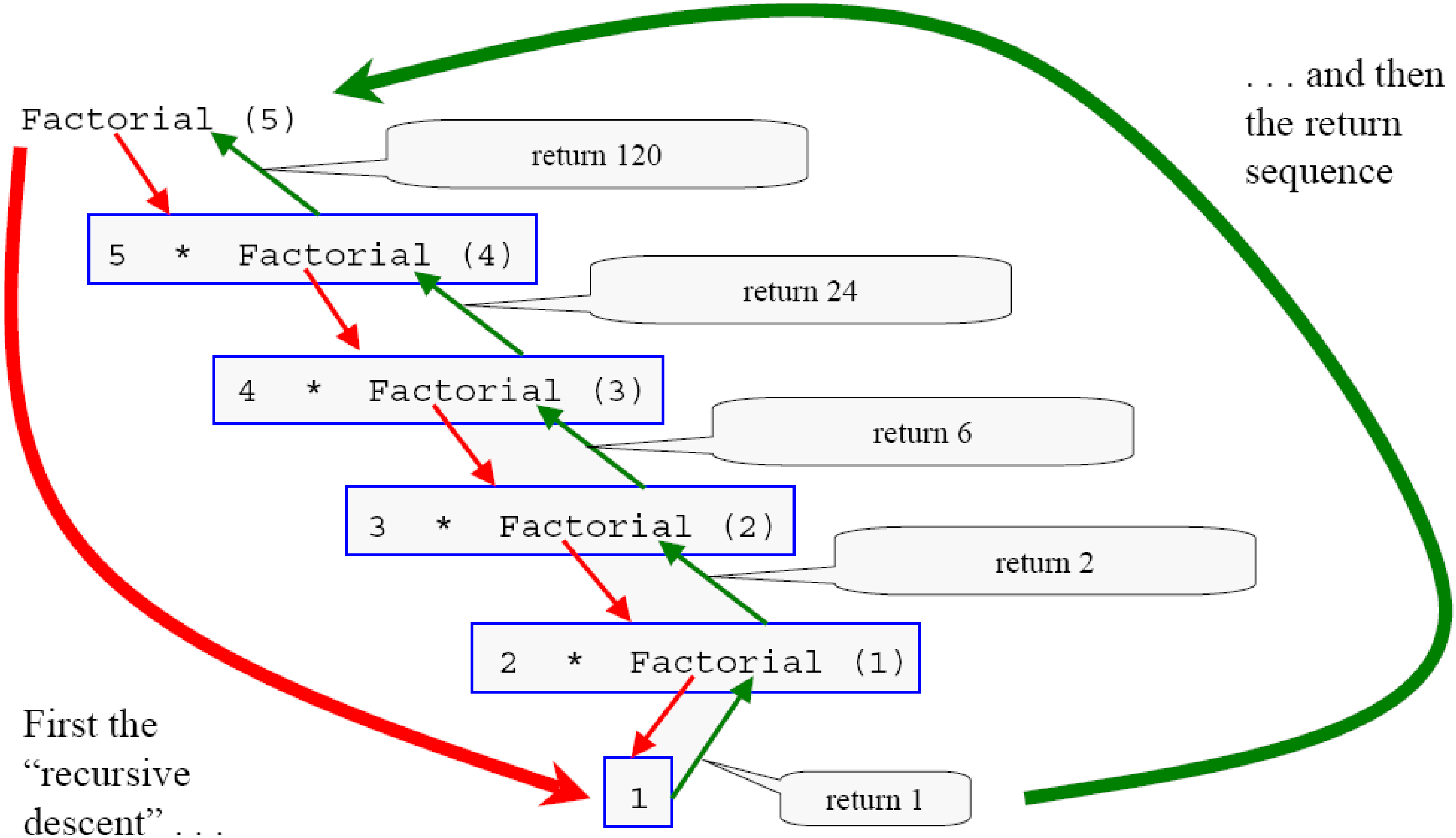


```
int Factorial(int n ) {
  if ( n > 1 )
    return( n * Factorial (n-1) );
  else
    return(1);
}
```

Factorial using Recursion

$$N! = 1 * 2 * \ldots * N$$

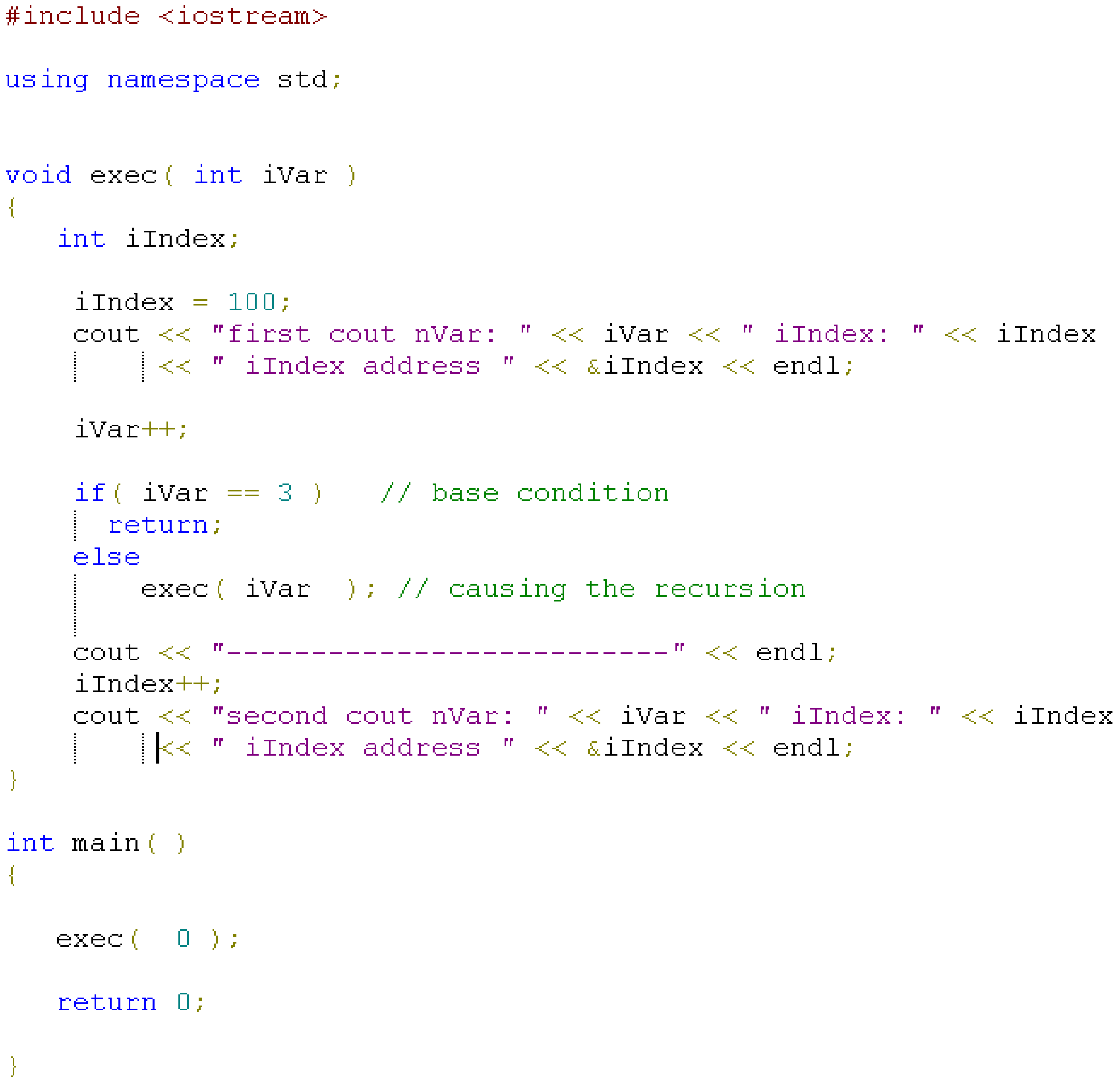
Factorial 4 = 4 x Factorial 3
Factorial 3 = 3 x Factorial 2
Factorial 2 = 2 x Factorial 1
Factorial 1 = 1



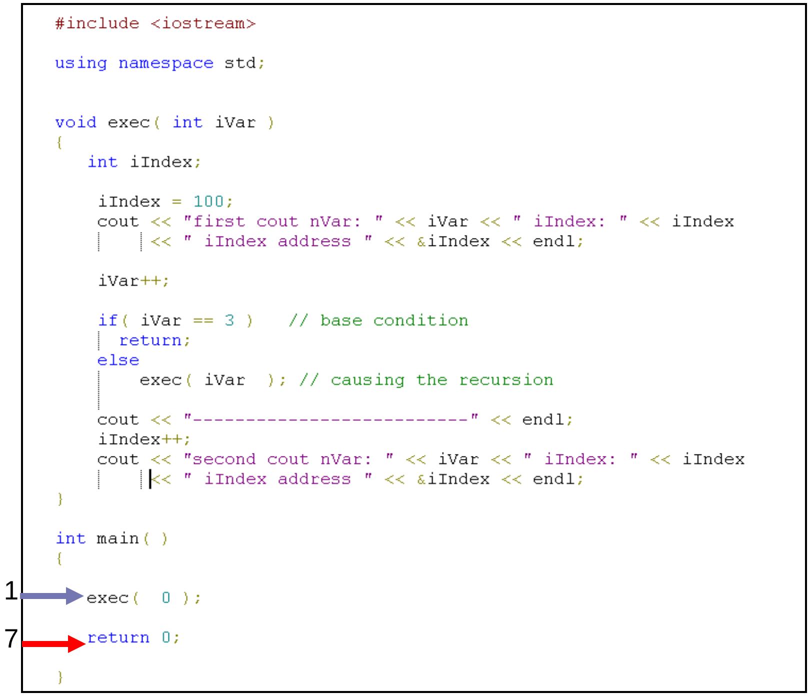
```
Factorial (5)
5 * Factorial (4)
4 * Factorial (3)
3 * Factorial (2)
2 * Factorial (1)
1
```

. . . and then the return sequence

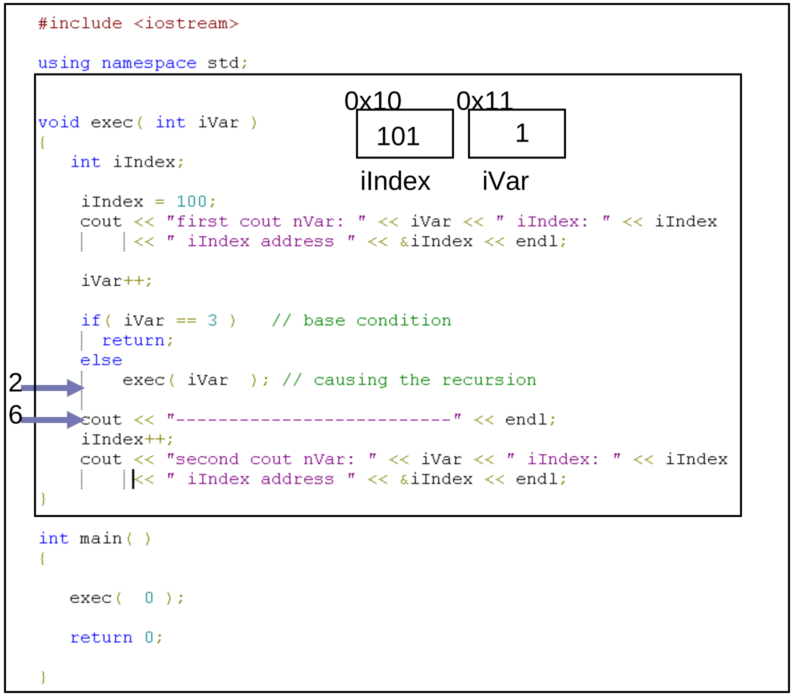
First the “recursive descent” . . .



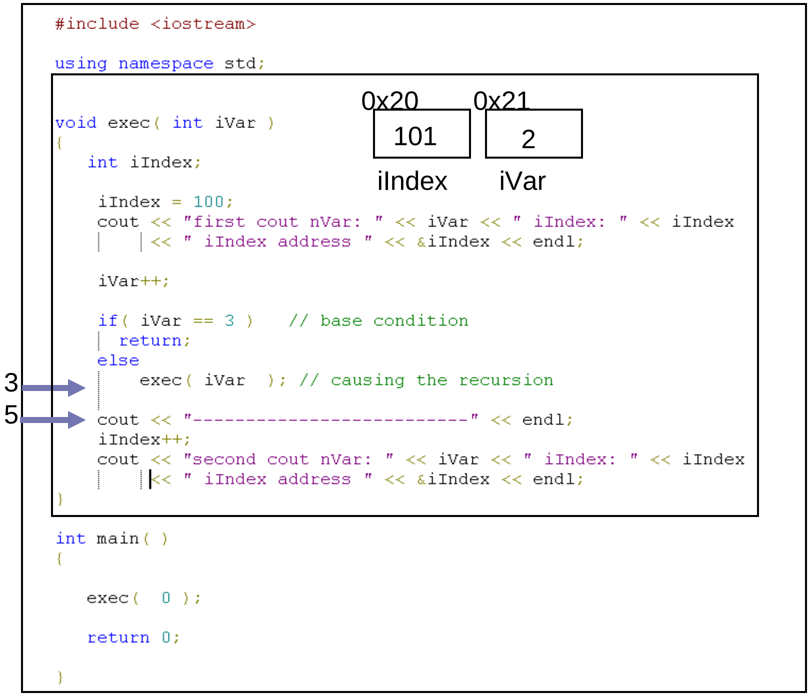
```cpp
#include <iostream>

using namespace std;


void exec( int iVar )
{
   int iIndex;

    iIndex = 100;
    cout << "first cout nVar: " << iVar << " iIndex: " << iIndex
         << " iIndex address " << &iIndex << endl;

    iVar++;

    if( iVar == 3 )    // base condition
      return;
    else
        exec( iVar  ); // causing the recursion

    cout << "--------------------------" << endl;
    iIndex++;
    cout << "second cout nVar: " << iVar << " iIndex: " << iIndex
         << " iIndex address " << &iIndex << endl;
}

int main( )
{

   exec(  0 );

   return 0;

}
```



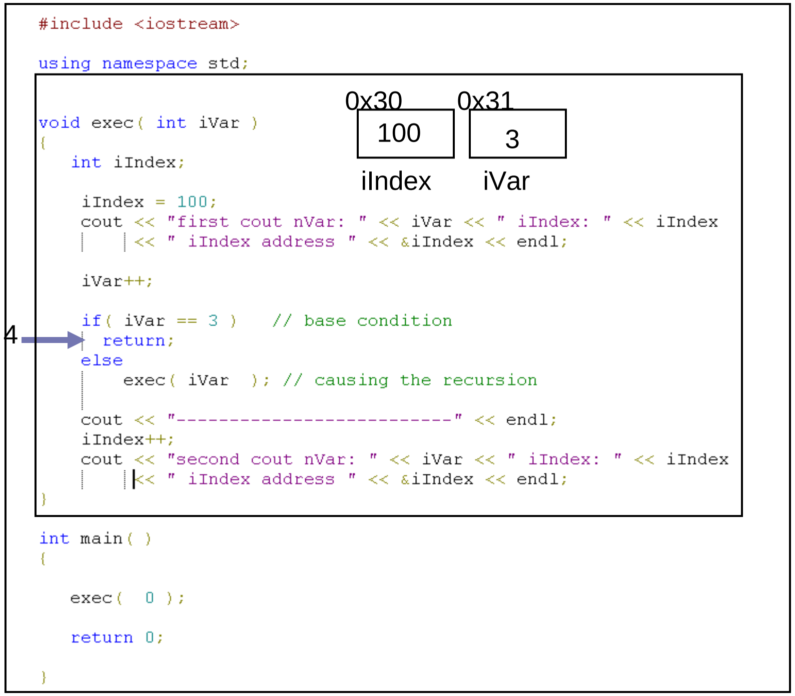
```
#include <iostream>

using namespace std;


void exec( int iVar )
{
   int iIndex;

    iIndex = 100;
    cout << "first cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;

    iVar++;

    if( iVar == 3 )    // base condition
      return;
    else
        exec( iVar  ); // causing the recursion

    cout << "--------------------------" << endl;
    iIndex++;
    cout << "second cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;
}

int main( )
{

   exec(  0 );

   return 0;

}
```



```
#include <iostream>

using namespace std;


void exec( int iVar )
{
   int iIndex;

    iIndex = 100;
    cout << "first cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;

    iVar++;

    if( iVar == 3 )    // base condition
      return;
    else
        exec( iVar  ); // causing the recursion

    cout << "--------------------------" << endl;
    iIndex++;
    cout << "second cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;
}

int main( )
{

   exec(  0 );

   return 0;

}
```

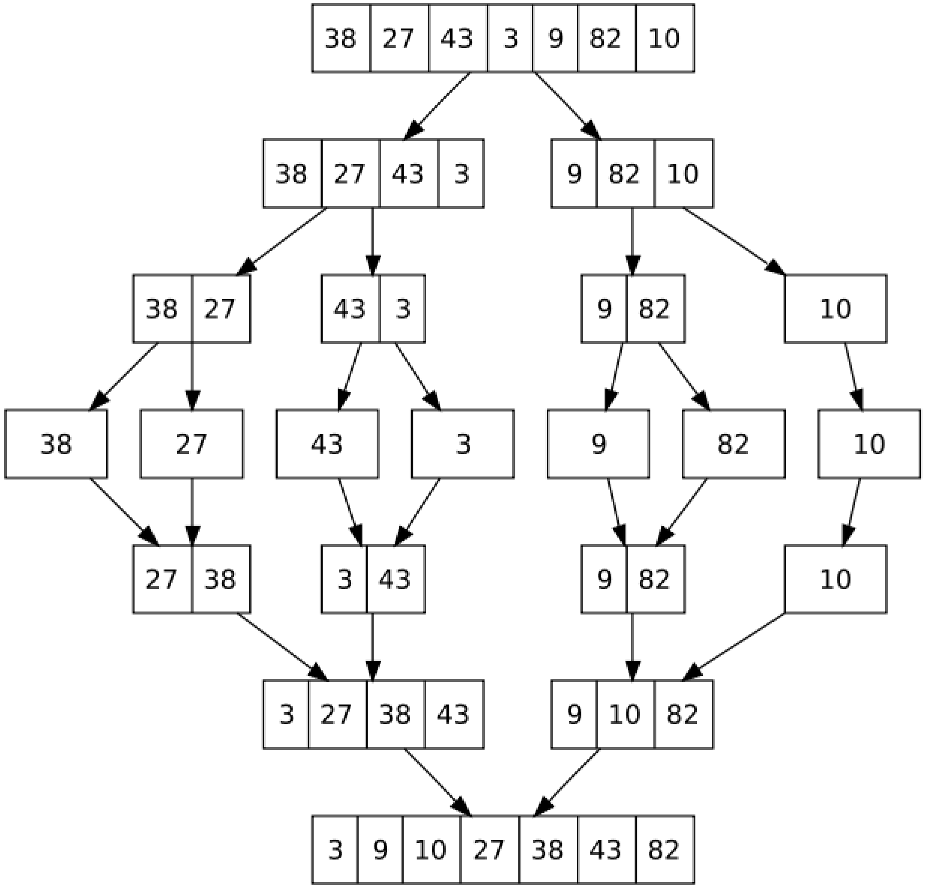


```
#include <iostream>

using namespace std;


void exec( int iVar )
{
   int iIndex;

    iIndex = 100;
    cout << "first cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;

    iVar++;

    if( iVar == 3 )    // base condition
      return;
    else
        exec( iVar  ); // causing the recursion

    cout << "--------------------------" << endl;
    iIndex++;
    cout << "second cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;
}

int main( )
{

   exec(  0 );

   return 0;

}
```



```
#include <iostream>

using namespace std;


void exec( int iVar )
{
   int iIndex;

    iIndex = 100;
    cout << "first cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;

    iVar++;

    if( iVar == 3 )    // base condition
      return;
    else
        exec( iVar  ); // causing the recursion

    cout << "--------------------------" << endl;
    iIndex++;
    cout << "second cout nVar: " << iVar << " iIndex: " << iIndex
        << " iIndex address " << &iIndex << endl;
}

int main( )
{

   exec(  0 );

   return 0;

}
```

MergeSort



38 27 43 3 9 82 10

38 27 43 3 | 9 82 10

38 27 | 43 3 | 9 82 | 10

38 | 27 | 43 | 3 | 9 | 82 | 10

27 38 | 3 43 | 9 82 | 10

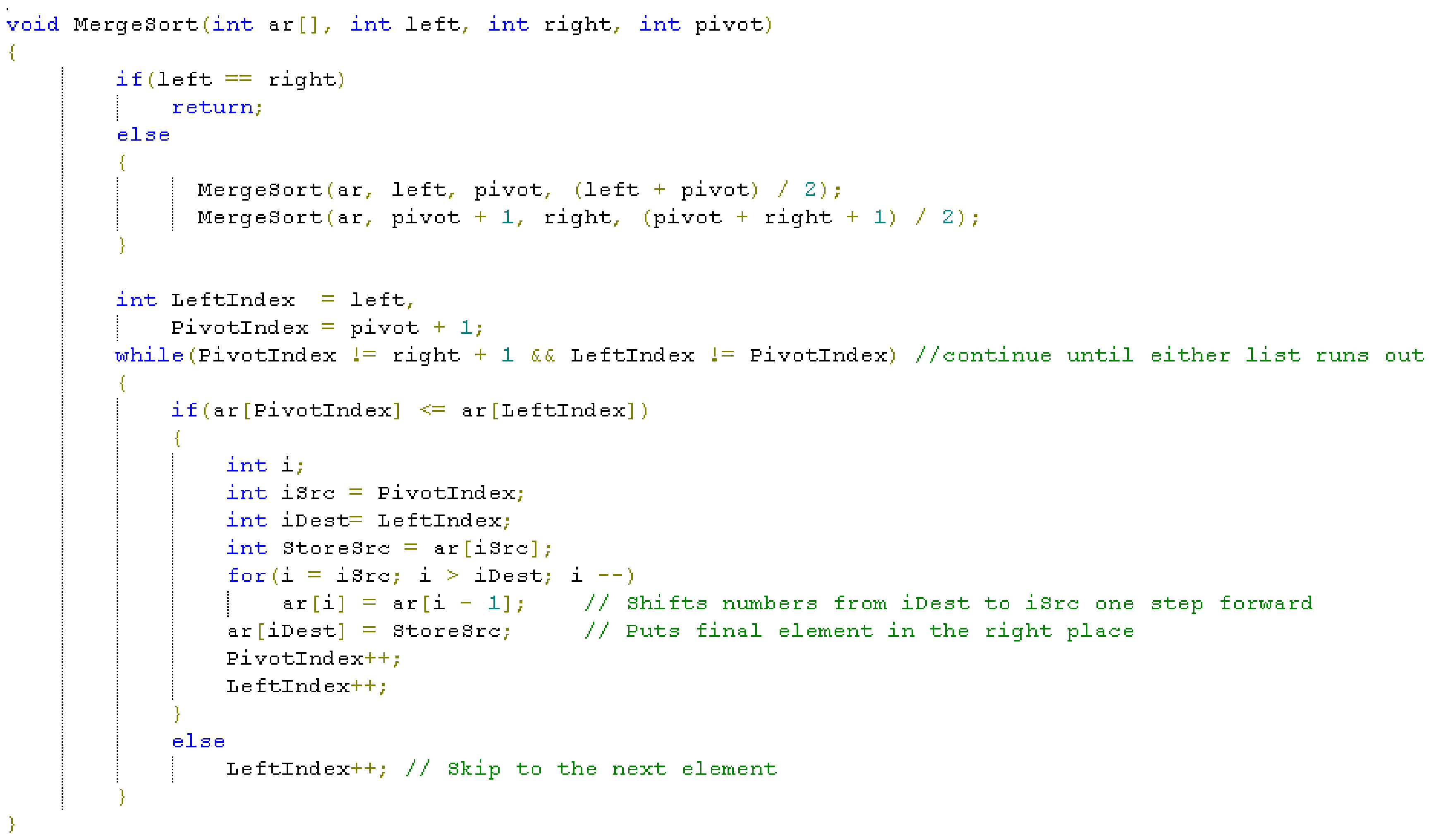
3 27 38 43 | 9 10 82

3 9 10 27 38 43 82

http://www.geocities.com/SiliconValley/Program/2864/File/Merge1/mergesort.html
http://www.iste.uni-stuttgart.de/ps/Ploedereder/sorter/sortanimation2.html

MergeSort using Recursion

- Conceptually, merge sort works as follows:
  - Divide the unsorted list into two sublists of about half the size
  - Divide each of the two sublists until we have list sizes of length 1, in which case the list itself is returned
  - Merge the two sublists back into one sorted list.



```
void MergeSort(int ar[], int left, int right, int pivot)
{
        if(left == right)
            return;
        else
        {
              MergeSort(ar, left, pivot, (left + pivot) / 2);
              MergeSort(ar, pivot + 1, right, (pivot + right + 1) / 2);
        }

        int LeftIndex  = left,
            PivotIndex = pivot + 1;
        while(PivotIndex != right + 1 && LeftIndex != PivotIndex) //continue until either list runs out
        {
            if(ar[PivotIndex] <= ar[LeftIndex])
            {
                int i;
                int iSrc = PivotIndex;
                int iDest= LeftIndex;
                int StoreSrc = ar[iSrc];
                for(i = iSrc; i > iDest; i --)
                    ar[i] = ar[i - 1];    // Shifts numbers from iDest to iSrc one step forward
                ar[iDest] = StoreSrc;     // Puts final element in the right place
                PivotIndex++;
                LeftIndex++;
            }
            else
                LeftIndex++; // Skip to the next element
        }
}
```