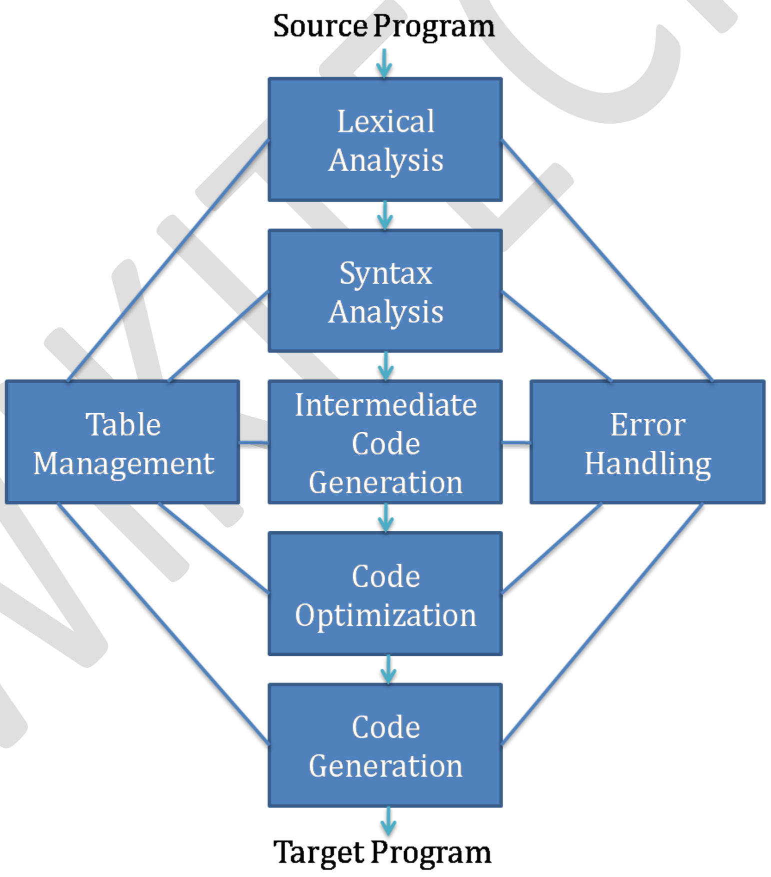
Code Optimization

Principle Sources of Optimization

- Preserve the semantics.
- Apply relatively low-level semantic transformations.
    - Algebraic identities like i + 0 = i
    - Performing the same operation on the same values yields the same result => i *1 = 1 * i = i



Quick Sort

```
void quicksort ( int m , int n )
{
/* recursively sorts a r[m] through a [n] */
  int i , j ;
  int v , x ;
     if ( n <= m) return ;
         i = m - 1 ;  j = n ; v = a [n] ;
         while ( 1 ) {
  do i = i + 1 ; while (a [i] < v) ;
  do j = j  - 1 ; while ( a [j ] > v) ;
    if ( i  >=  j ) break ;
       x = a [i] ; a [i] = a [j ] ; a [j ] = X ;
         /* swap a [i] , a [j ] */
}
       x = a [i] ; a [i] = a [n] ; a [n] = X ;
         /* swap a [i] , a [n] */
quicksort (m , j ) ;
quicksort ( i+ 1 , n) ;
```

Semantics-Preserving Transformations

- A program will include several calculations of the same value, such as an offset in an array.

- Examples of function-preserving (or semantics-preserving) transformations are,
    - Common-sub expression elimination,
    - Copy propagation,
    - Dead-code elimination, and
    - Constant folding

For More Details Click Here:

https://www.wikitechy.com/tutorials/compiler-design/code-optimization