Prime Factorization #numbertheory

$12 = 2^2 \times 3$

```
int List[128];  // saves the List
int listSize;   // saves the size of List
bitset<1000000> bs;
vector<int> prime;

void sieve(long long upper_bound) {
    bs.set();
    bs[0] = bs[1] = 0;
    prime.push_back(2);
    for(long long i = 3; i <= upper_bound + 1; i += 2) {
        if(bs[i]) {
            for(long long j = i * i; j <= upper_bound + 1; j += i)
                bs[j] = 0;
            prime.push_back((int) i);
        }
    }
}
void primeFactorize( int n ) {
    listSize = 0;   // Initially the List is empty, we denote that size = 0
    int sqrtN = int( sqrt(n) + 1); // find the sqrt of the number
    sieve(sqrtN + 1);
    for( int i = 0; prime[i] * prime[i] <= n; i++ ) { // we check up to the sqrt
        if( n % prime[i] == 0 ) { // n is multiple of prime[i]
            // So, we continue dividing n by prime[i] as long as possible
            while( n % prime[i] == 0 ) {
                n /= prime[i]; // we have divided n by prime[i]
                List[listSize] = prime[i]; // added the ith prime in the list
                listSize++; // added a prime, so, size should be increased
            }
            // we can add some optimization by updating sqrtN here, since n
            // is decreased. think why it's important and how it can be added
        }
    }
    if( n > 1 ) {
        // n is greater than 1, so we are sure that this n is a prime
        List[listSize] = n; // added n (the prime) in the list
        listSize++; // increased the size of the list
    }
}
int main() {
//    freopen("input", "r", stdin);
    primeFactorize(100);
    for (int i = 0; i < listSize; ++i) cout << List[i] << endl;
}
```