```cpp
#include <vector>
#include "Sieves.h"
using namespace std;

vector<long> Sieves::SlowGetAtkinPrimes(long limit)
{
    vector<bool> bsieve;
    vector<long> primes;

    for (long i = 0; i < limit + 1; i++)
         bsieve.push_back(false);

    for (long a = 1; a * a <= limit; a++)
    {
        for (long b = 1; b * b <= limit; b++)
        {
            // Main part of Sieve of Atkin

            long n = (4 * a * a) + (b * b);

            if (n <= limit && (n % 12 == 1 || n % 12 == 5))
                bsieve[n] = bsieve[n] ^ true;

            n = (3 * a * a) + (b * b);

            if (n <= limit && n % 12 == 7)
                bsieve[n] = bsieve[n] ^ true;

            n = (3 * a * a) - (b * b);

            if (a > b && n <= limit && n % 12 == 11)
                bsieve[n] = bsieve[n] ^ true;
        }
    }

    for (long r = 5; r * r <= limit; r++)
    {
        if (bsieve[r])
        {
            for (int i = r * r; i < limit; i += r * r)
                bsieve[i] = false;
        }
    }

    primes.push_back(2);
    primes.push_back(3);

    for (int x = 5; x <= limit; x++)
        if (bsieve[x])
            primes.push_back(x);

    return primes;
```

```
 53  }
 54
 55  vector<long> Sieves::SlowGetEratoPrimes(long limit)
 56  {
 57      // Sieve of Eratosthenes
 58      // find all prime numbers
 59      // less than or equal limit
 60
 61      vector<bool> bsieve;
 62      vector<long> primes;
 63
 64      for (long i = 0; i < limit + 1; i++)
 65          bsieve.push_back(false);
 66
 67      int c = 3, i, inc;
 68      bsieve[2] = true;
 69
 70      for (i = 3; i <= limit; i++)
 71          if (i % 2 == 1)
 72              bsieve[i] = true;
 73      do
 74      {
 75          i = c * c;
 76          inc = c + c;
 77
 78          while (i <= limit)
 79          {
 80              bsieve[i] = false;
 81
 82              i += inc;
 83          }
 84
 85          c += 2;
 86
 87          while (!bsieve[c])
 88              c++;
 89      } while (c * c <= limit);
 90
 91      for (i = 2; i <= limit; i++)
 92          if (bsieve[i])
 93              primes.push_back(i);
 94
 95      return primes;
 96  }
 97
 98  long Sieves::GetBit(long i, long sieve[])
 99  {
100      long b = i % BITS_PER_LONG;
101      long c = i / BITS_PER_LONG;
102      return (sieve[c] >> (BITS_PER_LONG_1 - b)) & 1L;
103  }
104
```

```
105 void Sieves::SetBit(long i, long v, long sieve[])
106 {
107     long b = i % BITS_PER_LONG;
108     long c = i / BITS_PER_LONG;
109     long mask = 1L << (BITS_PER_LONG_1 - b);
110 
111     if (v == 1)
112         sieve[c] |= mask;
113     else
114         sieve[c] &= ~mask;
115 }
116 
117 vector<long> Sieves::FastGetAtkinPrimes(
118     long limit, long sieve[])
119 {
120     long nlongs = limit / BITS_PER_LONG + 2;
121     vector<long> primes;
122 
123     for (long i = 0; i < nlongs; i++)
124         sieve[i] = 0;
125 
126     for (long a = 1; a * a <= limit; a++)
127     {
128         for (long b = 1; b * b <= limit; b++)
129         {
130             // Main part of Sieve of Atkin
131 
132             long n = (4 * a * a) + (b * b);
133 
134             if (n <= limit && (n % 12 == 1 || n % 12 == 5))
135             {
136                 long bit = GetBit(n, sieve);
137                 SetBit(n, bit ^ 1, sieve);
138             }
139 
140             n = (3 * a * a) + (b * b);
141 
142             if (n <= limit && n % 12 == 7)
143             {
144                 long bit = GetBit(n, sieve);
145                 SetBit(n, bit ^ 1, sieve);
146             }
147 
148             n = (3 * a * a) - (b * b);
149 
150             if (a > b && n <= limit && n % 12 == 11)
151             {
152                 long bit = GetBit(n, sieve);
153                 SetBit(n, bit ^ 1, sieve);
154             }
155         }
156     }
```

```
157
158     for (long r = 5; r * r <= limit; r++)
159     {
160         if (GetBit(r, sieve))
161         {
162             for (long i = r * r; i <= limit; i += r * r)
163                 SetBit(i, 0, sieve);
164         }
165     }
166
167     primes.push_back(2);
168     primes.push_back(3);
169
170     for (long x = 5; x <= limit; x++)
171         if (GetBit(x, sieve))
172             primes.push_back(x);
173
174     return primes;
175 }
176
177 vector<long> Sieves::FastGetEratoPrimes(
178     long limit, long sieve[])
179 {
180     long c, i, inc, nlongs = limit / BITS_PER_LONG + 1;
181     vector<long> primes;
182
183     for (long i = 0; i <  nlongs; i++)
184         sieve[i] = 0;
185
186     SetBit(0, 0, sieve);
187     SetBit(1, 0, sieve);
188     SetBit(2, 1, sieve);
189
190     for (i = 3; i < limit; i++)
191         SetBit(i, i & 1, sieve);
192
193     c = 3;
194
195     do {
196         i = c * c, inc = c + c;
197         while (i < limit) {
198             SetBit(i, 0, sieve);
199             i += inc;
200         }
201         c += 2;
202         while (!GetBit(c, sieve)) c++;
203     } while (c * c <= limit);
204
205     for (i = 2; i <= limit; i++)
206         if (GetBit(i, sieve))
207             primes.push_back(i);
208
```

```
209     return primes;
210 }
```