**main.cpp**

```cpp
#include <vector>
#include <iostream>
#include <chrono>

/*
Purpose: My friend is a mathematician who works extensively with prime numbers.
As prime numbers are not exactly predictable by any mathematical function, he
has requested help in making a program than can compute the indices within the
set of primes.

This program, titled "Prime Number Indexer" finds the index of the nth prime in
the set of primes.

First, the set of primes is generated up to a specified index.
Then, the user specifies the value of the nth prime, and the program returns
its index n.
*/

// global variables
std::vector<int> primeSet;
int upperLimit;

// forward declarations
void primeCheck(int number);
void printPrimes();
int searchPrimeSet(int prime);
void indexingNth();


/**
 * Timer class using <chrono>
 * Written by Alex
 * Source: https://www.learncpp.com/cpp-tutorial/timing-your-code/
 */
class Timer {
private:
  // Type aliases to make accessing nested type easier
  using Clock = std::chrono::steady_clock;
  using Second = std::chrono::duration< double, std::ratio<1> >;

  std::chrono::time_point<Clock> m_beg { Clock::now() };

public:
  void reset() {
    m_beg = Clock::now();
  }

  double elapsed() const {
    return std::chrono::duration_cast<Second>(Clock::now() - m_beg).count();
  }
};


```

```cpp
// program
int main() {

  // user input for number of primes
  std::cout << "How many prime numbers would you like?  ";
  std::cin >> upperLimit;
  std::cout << '\n' << "Generating primes..." << '\n';

  Timer generationTimer;

  // generates set of primes up to upperLimit
  for (int i = 2; primeSet.size() < upperLimit; ++i) {
    primeCheck(i);
  }

  double generationTime = generationTimer.elapsed();
  Timer printTimer;

  // prints primeSet
  printPrimes();

  double printTime = printTimer.elapsed();
  std::cout << '\n' << "Generation Time: " << generationTime << "s" << '\n';
  std::cout << "Print Time: " << printTime << "s" << '\n';

  indexingNth();

} //main()


// function definitions

// primeCheck() checks whether number is prime and adds to set
void primeCheck(int number) {

  bool isPrime { true };

  // checks if number is prime or composite
  // implementation of Erastothenes' Sieve algorithm
  for (int prime : primeSet)  {
    if (number % prime == 0) { isPrime = false; }
  };

  // adds prime number to primeSet
  if (isPrime) { primeSet.push_back(number); }
  return;

} //primeCheck()

// prints primeSet elements to terminal
void printPrimes() {

  // prints first element without comma
  std::cout << primeSet[0];

  // prints rest of elements
  for (int prime : primeSet) {
    if (prime == 2) { continue; } // handles first case
    std::cout << ", " << prime;
  }
  std::cout << '\n';

} //printPrimes()

// returns index of prime in primeSet
int searchPrimeSet(int prime) {

  int index { -1 };

  // searches for prime in primeSet
  for (int i = 0; i < upperLimit; ++i) {
    if (prime == primeSet[i]) {
      // stores index of prime
      index = i + 1;
    }
  }

  // error handling
  if (index == -1) { throw "Invalid input, not in generated prime set"; }

  return index;

} //searchPrimeSet()

void indexingNth() {

  // user input for nth prime
  int nthPrime;
  std::cout << '\n' << "Which prime number would you like to index?  ";
  std::cin >> nthPrime;

  // prints index of nth prime
  try {

    Timer searchTimer;

    int n { searchPrimeSet(nthPrime) };

    double searchTime { searchTimer.elapsed() };
    std::cout << "Search Time: " << searchTime << "s" << "\n\n";

    std::cout << nthPrime << " is " << n << "th in the set of primes." << '\n';

  } catch (const char*) {

    std::cout << "Invalid input, not in generated prime set" << '\n';
    indexingNth();

  }

} //indexingNth()

```