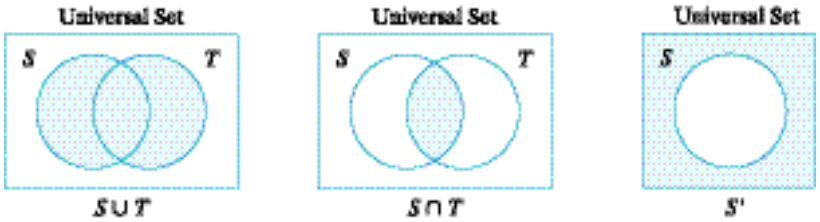
Implementing Sets with Bitsets

In mathematics and computer science, the term **set** refers to an unordered collection of objects called the **elements** or **members** of the set. A set is commonly denoted by listing the elements enclosed in braces, { and }. For example, the set of decimal digits contains the elements 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 and is denoted {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. The set of uppercase letters is {A, B, C, ··· , Z}. The set of even prime numbers {2} contains the single element 2; and the set of even prime numbers greater than 2 is the empty set, denoted ø or { }, that is, the set containing no elements.

In a problem involving sets, the elements are selected from some given set called the **universal set** for that problem. For example, if the set of vowels or the set {X, Y, Z} is being considered, the universal set might be the set of all letters. If the universal set is the set of names of months of the year, then one might use the set of summer months {June, July, August}; the set of months whose names do not contain the letter *r* {May, June, July, August}; or the set of all months having fewer than 30 days {February}

The basic relation in defining a set is the **membership** relation. Given a set *S* and any object *x* in the universal set, one must be able to determine that *x* belongs to **S**, denoted by *x* *S*, or that it does not belong to *S*, denoted by *x* *S*.

Three basic set operations are intersection, union, and set difference. The **intersection** of two sets *S* and *T*, denoted in mathematics by *S* ∩ *T*, is the set of elements that are in both sets. The **union** of *S* and *T*, *S* *T*, is the set of elements that are in *S* or in *T* or in both. The **complement** of a set *S*, *S*', consists of those elements of _the universal set that are not in *S*. The following **Venn diagrams** illustrate these basic set operations:



Although there are many other set operations that are important in mathematics, we will take these as the basic operations in our specification of a set as an abstract data type:

ADT SET

Collection of Data Elements:

An ordered collection of data items.

Basic Operations:

- Membership
- Union
- Intersection
- Complement

Sets whose elements are selected from a *finite* universal set can be represented in computer memory by bit strings in which the number of bits is equal to the number of elements in this universal set. Each bit corresponds to exactly one element of the universal set. A given set is then represented by a bit string in which the bits corresponding to the elements of that set are 1 and all other bits are 0. A `bitset` is therefore an obvious data structure to use to implement such a set.

To illustrate, suppose the universal set is the set of uppercase letters. Then any set of uppercase letters can be represented by a string of 26 bits, with one bit corresponding to the letter A, another corresponding to the letter B, and so on. Thus, the set of vowels can be represented by the bit string

```
1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
| | | | | | | | | | | | | | | | | | | | | | | | | |
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

and the empty set by

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

The `bitset` operations `&`, `|`, and `flip()` can be used to implement the basic set operations of intersection, union, and complement. Applying the & operation bitwise to the bit strings representing two sets yields a bit string representing the _intersection of these sets. For example, consider the sets S = {A, B, C, D} and T = {A, C, E, G, I}, where the universal set is the set of uppercase letters. The 26-bit string representations of these sets are as follows:

```
S:    1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

T:    1 0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      | | | | | | | | | | | | | | | | | | | | | | | | | |
      A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

Performing the & operation bitwise gives the bit string

```
      1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
      | | | | | | | | | | | | | | | | | | | | | | | | | |
      A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

which represents the set {A, C}, the intersection of *S* and *T.*

Bitwise application of the `|` operation to the bit strings representing two sets *S* and *T* yields the representation of *S* *T* . For the preceding sets, this gives the bit string

```
1 1 1 1 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
| | | | | | | | | | | | | | | | | | | | | | | | |
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

which represents {A, B, C, D, E, G, I}, the union of *S* and *T.*

Changing each bit in the representation of a set *T* with the `flip()` operation gives a bit string representing *T'*. For the preceding sets, bitwise complementation of the string for *T* gives

```
0 1 0 1 0 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
| | | | | | | | | | | | | | | | | | | | | | | | |
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
```

which represents *T'*. = {B, D, F, H, J, K, ···, Z}.

Example: Finding Prime Numbers with the Sieve of Eratosthenes

A **prime number** is an integer greater than 1 whose only divisors are 1 and the number itself. Prime numbers are important in public-key encryption/decryption methods (see Sec. 5.5), in random-number generators, in the design of hash tables (see Sec. 12.7), and in many other applications of number theory.

The problem of finding prime numbers has occupied people's attention for centuries. One of the classical algorithms was developed by the Greek mathematician Eratosthenes (c. 276–c. 194 B.C.) and is known as the **Sieve Method of Eratosthenes.** Variations of it are still used today in prime-generation algorithms.

ALGORITHM FOR THE SIEVE METHOD OF ERATOSTHENES

/* Algorithm to construct a set *sieve* of all primes in a given range.

Receives: An integer n

Returns: The set *sieve* of all primes in the range 2 through *n*

----------*/

1. Initialize the set *sieve* to contain the integers 2 through *n*.
2. Select the smallest element *prime* in *sieve.*
3. While $prime^2 \leq n$, do the following:
   a. Remove from *sieve* all elements of the form *prime* * *k* for $k \geq 1$.
   b. Replace *prime* with the smallest element in *sieve* that is greater than *prime.*

The following diagram illustrates this algorithm for $n = 30$:

sieve

{2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30}

↓

prime = 2

↓

{2,3,5,7,9,11,13,15,17,19,21,23,25,27,29}

↓

prime = 3

↓

{2,3,5,7,11,13,17,19,23,25,29}

↓

prime = 5

↓

{2,3,5,7,11,13,17,19,23,29}

↓

prime = 7; terminate since *prime*2 > 30

The main object needed for this problem is a representation of the set *sieve*. For this we will use a `bitset` in the manner described earlier:

Objects:

`sieve`: a `bitset` in which `sieve[i]` will be 1 if `i` is a prime, and 0 otherwise
`n`, `prime`: integers

Operations:

Construct `sieve` with all bits set to 1
Set selected bits in `sieve` to 0
Check whether a bit of `sieve` is 0 or 1

All of these operations are provided by `bitset`.

The following program uses these objects and operations and the sieve algorithm to find primes in a range specified by the user.

Sieve Method of Finding Primes

```
/*-------------------------------------------------------------------------------
   Program that uses the Sieve Method of Eratosthenes to find all prime numbers
   in a given range.
   Uses the standard C++ bitset class template.

   Input:  A positive integer n
   Output: Primes in the range 2 through n
 -------------------------------------------------------------------------------*/

#include <iostream>
#include <cassert>
#include <bitset>
using namespace std;
```

```
int main()
{
  const int MAX_PRIME = 1000;     // limit on size of n

  int n;
  cout << "This program finds primes in the range 2 - n."
          "\nEnter positive integer n <= " << MAX_PRIME << ": ";
  cin >> n;
  assert(n > 0 && n <= MAX_PRIME);

  // Construct bitset sieve of size MAX_PRIME + 1 containing all ones.
  // For convenience, we are not using positions 0 and 1 of sieve so
  // that bit i represents positive integer i.

  bitset<MAX_PRIME + 1> sieve;
  sieve.set();

  // Apply the Sieve Method
  int prime = 2;                  // the next prime in sieve
  while (prime * prime <= n)
  {
    // cross out multiples of prime from sieve
    for (int mult = 2*prime; mult <= n; mult += prime)
      sieve.reset(mult);

    // find next uncrossed number in sieve
    do
      prime++;
    while (!sieve.test(prime));
  }

  // Display the list of primes.
  cout << "\Primes in the range 2 through " << n << ":\";
  for (int i = 2; i <= n; i++)
    if (sieve.test(i))
      cout << i << "  ";
}
```

Sample run:

```
This program finds primes in the range 2 - n.
Enter positive integer n <= 1000: 100

Primes in the range 2 through 100:
2  3  5  7  11  13  17  19  23  29  31  37  41  43  47  53  59  61  67
71  73  79  83  89  97
```

EXERCISES

Exercises 1–7 assume the universal set is {0, 1, , 19}. Give a bitstring representation of the given set.

1. The set of odd integers
2. The set of prime integers
3. The intersection of the set of prime integers and the set of even integers
4. The union of the set of prime integers and the set of odd integers
5. The set of odd integers that are not prime integers
6. The set of integers divisible by 1
7. The set of integers not divisible by 1

Exercises 8–11 assume the universal set is the set of names (strings) Alan, Alice, Barb, Ben, Bob, Carl, Cora, Dick, Don, Dora, Dot, Fred, arranged in alphabetical order. Give a bitstring representation of the given set.

8. Set of names that begin with B
9. Set of names that begin with E
10. Set of names that begin with D and have fewer than three letters.
11. Set of names that have fewer than six letters.
12. Design a `Set` class for representing sets of integers in some given range. Use `bitset` in your implementation as described in the text. Provide (at least) the operations listed in the specification of the set ADT together with input and output operations.You should test your class with a driver program.