# Implementing Abstract Data Types

The general mechanism in C++ for implementing abstract data types is to define a class.

```
class my_type {
\\ Declare variables required to represent values of the type.
\\ These may be of built_in types or classes defined earlier.
\\ A user of my_type cannot access the variables defined here.

public:

\\ Define the methods needed to perform operations on this type.
\\ These are the only ways a user can manipulate these values.

   my_type const operator+(mytype  const &);
   my_type const operator=(mytype const &);

\\ Declares operators + and = for this type
\\ The assignment operator = has a default meaning for any class,
\\ which essentially copies the bits, but it may be necessary to
\\ redefine it in some cases.
};
```

A user could just write

```
my_type x, y;
x = x + y;
```

Write classes to implement the following abstract data types. When implementing abstract data types, it is assumed that the user will use the operations correctly, as specified by the formal definitions. If used incorrectly, the implementation may behave arbitrarily, including possibly crashing.

1. Dates: The possible values are valid dates. The operations to be performed are:
   (a) void set(string): sets the date from a string in the form DD/MM/YYYY.
   (b) string get(): returns the date as a string in the form DD/MM/YYYY.
   (c) date operator+(int i) : returns the date after i days from the date to which the + operator is applied. Note that i may be negative in which case it should return the date $-$i days before.
   (d) int operator$-$(date d): Returns the number of days between the two dates as an integer. This is positive if the subtracted date is earlier.

It can be assumed that valid dates will have year at least zero and at most 9999. In many early programs, only two digits were used for the year. This created a major scare in 1999, called the Y2K bug, as the next year would have become 00, and all arithmetic with dates would have gone wrong. In this problem it can be assumed that a year YYYY is a leap year iff it is divisible by 4 but not by 100 or is divisible by 400. Date is a built_in class in some languages.

2. The int type in C++ is supposed to be an implementation of the abstract data type integer. However, it puts a limit on the possible values. Typically, the value is less than $2^{31}$ in magnitude (32 bits). You can get a bigger set by using the type long long int, which gives up to $2^{63}$. Suppose you do not want any such restriction and want to work with integers having any number of bits. Implement a new type called my_int with basic arithmetic operations and input/output. You can implement simple methods for multiplying and dividing two numbers, but doing it better is a challenging problem. The fastest method for multiplying two large integers was found just a few years ago. There are many libraries that give implementations of this, with fast algorithms for multiplication/division etc. One such freely available one is gmp (gnu multiprecision arithmetic library), which also supports arbitrary precision floating point numbers.

   Arbitrary precision here means that there is no fixed bound on the number of digits. This will always be limited in reality by the amount of memory available, but the program itself does not put any restriction.

3. Triangles: The possible values of this data type are triangles in the plane. Assume a triangle is specified by the coordinates of its 3 corners, but there can be other ways of doing it. Some possible operations on triangles are rotation, reflection, scaling, and checking for congruence or similarity. Another useful operation is checking whether a given point is inside, on or outside a triangle, and finding its area. Try to implement as many of these as possible.