

## Using the Array2D class

The Array2D class is a templated class that creates objects that can be used as 2-dimensional arrays. The purpose is to avoid some of the problems that arise when trying to use the normal array mechanism in C++ to build 2-dimensional arrays.

The public interface of the Array2D class is the following:

```
template <class T>
class Array2D {
public:
    Array2D(); // Create empty array
    Array2D(int R, int C); // Create R*C array
    Array2D(const Array2D& Source);
    Array2D& operator=(const Array2D& Source);

    bool set(int R, int C, T Value); //Assign Value to location
    T get(int R, int C) const; // Get Value from location
    T& operator() (int R, int C); // Get reference for value
    ~Array2D();
    class InvalidCoord; // exception class
};
```

**Declaring objects:** To declare an Array2D object, you need to provide a type for the template parameter T. For instance, to create an Array2D object named `intarray` to hold `int` values, you would declare the variable as

```
Array2D<int> intarray(10,20);
```

The type parameter between the brackets is part of the type name for the `intarray` variable. If you want to pass this object as a parameter you would use the type `Array2D<int>` instead of just `Array2D`.

We can also declare Array2D variables for objects that can hold other types of objects. For instance, if we want the array to hold `GamePiece` objects for project 1 we would declare the array as

```
Array2D<GamePiece> brd1(15,32);
```

Or, we might want it to hold pointers to `GamePiece` objects as in

```
Array2D<GamePiece*> board(45,60);
```

**Using the functions:** There are two ways to use the objects as arrays once they are allocated. The first is with the `set` and `get` functions, and the second is with the `operator()` function. The `set` function assigns the value parameter to the proper location in the array, and the `get` function returns the value at the coordinates. If the coordinates given are not valid for the array, the `set` function will not insert the value and will return false, otherwise it returns true. The `get` function throws an exception if the coordinates are not valid. Exceptions are used to indicate a serious error that may affect the execution of a program, and if one is thrown and not caught it will cause the program to abort. Obviously, you can avoid this situation by making sure that your coordinates are valid, but you can also put any code that might raise the exception in a try block followed by a catch clause (an *exception handler*).

For example, if we want to access the `(row, col)` entry in `intarray` as declared above and assign the value to a variable `x`, the code would look like:

```
int x;
try {
    x = intarray.get(row,col);
}
catch (Array2D<int>::InvalidCoord ic) {
    //executes if exception occurs
    cerr << "Array index out of range: ("
        << ic.row << ", " << ic.col << ")" << endl;
}
```

Of course, the best way to deal with the exception is to make sure that you don't try to access a location that is not in the array!

The other way to use the array objects is more like regular array notation. The function `operator()` allows the object to be used like an array, except instead of writing `a[i][j]` we will write `a(i, j)`. (The reason that we do not use the standard array notation has to do with the array notation convention in C++. Basically, we can only define an `operator[]` and not an `operator[][]`.)

Because the `operator()` function returns a reference to a `T` object, we can use this function to assign to an array location, and get values from the location. This is more like what we are used to with regular arrays. For instance, here is a fragment of code that manipulates the object `intarray`.

```
int i, j, tmp;
for (i = 0; i < 10; i++)
    for (j = 0; j < 20; j++)
        intarray(i,j) = i*j;
for (j = 0; j < 20; j++) {
    for (i = 0; i < 10; i++) {
        tmp = intarray(i,j);
        cout << setw(4) << tmp << " ";
    }
    cout << endl;
}
```

Like the `get` function, the `operator()` function also raises an exception if the coordinates are out of range, so if you are sloppy for error checking in your parser you should include an exception handler (try block with catch clause). (In principle, you should always use an exception handler when using classes that can throw exceptions, but we don't plan to check for their use in your programs.)