```
#ifndef ARRAY_H
#define ARRAY_H

// forward declcarations
template <typename T> class array_t;
template <typename T> std::ostream& operator<<(std::ostream&, const array_t<T>&);

template <typename T>
class array_t {
  public:
  // constructors (overloaded)
  array_t(int isz = 5);
  array_t(T *array, int isz);

  // copy constructor
  array_t(const array_t& rhs);

  // destructors
  ~array_t()                              { delete [] arr; }

  // friends -- note the extra <> telling the compiler to instantiate
  // a templated version of the operator<< -- <T> is also legal, i.e.,
  // friend std::ostream& operator<< <T>(std::ostream& s, const array_t&);
friend  std::ostream& operator<< <>(std::ostream& s, const array_t& rhs);
friend  std::ostream& operator<<(std::ostream& s, array_t *rhs)
                                          { return(s << (*rhs)); }

  // operators
  const T& operator[](int i) const      { return arr[i]; }
  T& operator[](int i)                  { return arr[i]; }

  // assignment operator
  const array_t& operator=(const array_t&);

  // members
  int   size(void) const                { return sz; }

  void  randseed();
  void  randfill();

  T     min();
  T     max();

  int   find(T val) const;

  // private: only available to this class
  private:
  int                   sz;
  T                     *arr;
};

#endif
```