Mathematical Background

It will be useful to review these topics:

- Set concepts and notation
- Recursion
- Proof techniques:
  - Induction
  - Contradiction
- Logarithms
- Summations
- Recurrence relations

Mathematics Review

- Sets
  - A set is a collection of distinguishable members (aka elements)
  - Elements are typically drawn from a large population called a base type
  - Each member is a primitive element or also a set
  - There is no duplication of elements
  - Example:
    - R contains elements 3, 4, and 5
    - Members are 3, 4, and 5
    - Base type is integer
    - Notation: \( R = \{3, 4, 5\} \)

Mathematics Review

- Set Notation
  - Set membership:
    - \( x \in A \): “\( x \) is an element of \( A \)”
    - \( x \not\in A \): “\( x \) is not an element of \( A \)”
  - \( \emptyset \) denotes null set or empty set
  - \( |A| \) denotes set cardinality
  - Relationships and operations:
    - \( A \subseteq B \): “\( A \) is a subset of \( B \)” or “\( A \) is included in \( B \)”
    - \( B \supseteq A \): “\( B \) is a superset of \( A \)”
    - \( A \cup B \): A union \( B \)
    - \( A \cap B \): A intersect \( B \)
    - \( A - B \): all elements of \( A \) but not in \( B \)

Mathematics Review

- Types of sets
  - Linear order: a set with the following properties:
    1. For any elements \( a, b \) in set \( S \), exactly one of \( a < b, a = b, \) or \( a > b \) is true
    2. For all elements \( a, b, c \in S \), if \( a < b \) and \( b < c \), then \( a < c \) (transitivity property)
  - Finite Sequence: similar to a set but order is imposed
  - A finite sequence of length \( n \) is a function \( f \) whose domain is the set \( \{0, 1, \ldots, n - 1\} \)
  - Elements of a sequence have an order
  - A sequence may contain duplicates that are distinct members of the sequence
### Mathematics Review

#### Exponents

- \[ X^A X^B = X^{A+B} \quad [E-1] \]
- \[ \frac{X^A}{X^B} = X^{A-B} \quad [E-2] \]
- \[ (X^A)^B = X^{AB} \quad [E-3] \]
- \[ X^N + X^N \neq 2X^N \quad [E-4] \]
- \[ 2^N + 2^N = 2^{N+1} \quad [E-5] \]

#### Logarithms:

- \[ \log nm = \log n + \log m \quad [L-1] \]
- \[ \log \frac{n}{m} = \log n - \log m \quad [L-2] \]
- \[ \log n^r = r \log n \quad [L-3] \]
- \[ \log_a n = \frac{\log_b n}{\log_b a} \quad [L-4] \]

---

### Mathematics Review

#### Modular Arithmetic

- English: A is congruent to B modulo N
- Mathematically: \[ A \equiv B \pmod{N} \]
- Meaning “N divides A – B”
- Alternatively:
  - Compute the remainder \( R_A \) from dividing \( N \) by \( A \)
  - Compute the remainder \( R_B \) from dividing \( N \) by \( B \)
  - then \( R_A = R_B \) if \( A \equiv B \pmod{N} \)
- Example:
  - \( 81 \equiv 61 \equiv 1 \pmod{10} \)
- Relations: If \( A \equiv B \pmod{N} \), then
  - \( A + C \equiv B + C \pmod{N} \)
  - \( AD \equiv BD \pmod{N} \)

#### Proofs

- By induction
  - Example: the sum of the first n positive integers is \( \frac{n(n+1)}{2} \)
- By counterexample
  - Example: If \( F_k \) is the kth Fibonacci number, then \( F_k \leq k^2 \) is false
- By contradiction
  - Example: “There is no largest integer.”
Recursion

An algorithm is *recursive* if it calls itself to do part of its work.

- Recursive Functions
  - Example (Fig. 1.2):
    ```c
    int f(int x) {
      if(x == 0) // 1
        return 0; // 2
      else
        return 2*f(x-1) + x*x; // 3
    }
    ```
  - Example (Fig. 1.3):
    ```c
    int bad(int n) {
      if(n == 0) // 1
        return 0; // 2
      else
        return bad(n/3+1) + n-1; // 3
    }
    ```

Mathematics Review

- Mathematical Series
  - \[
  \sum_{i=0}^{n} i = \frac{n(n+1)}{2} \quad [S-1]
  \]
  - \[
  \sum_{i=1}^{n} i^2 = \frac{2n^3 + 3n^2 + n}{6} \quad [S-2]
  \]
  - \[
  \sum_{i=1}^{\log n} n = n \log n \quad [S-3]
  \]
  - \[
  \sum_{i=0}^{\infty} a^i = \frac{1}{1-a} \text{ for } 0 < a < 1 \quad [S-4]
  \]
  - \[
  \sum_{i=1}^{n} \frac{i}{2^i} = 2 - \frac{n+2}{2^n} \quad [S-5]
  \]

- Mathematical Series (cont.)
  - \[
  \sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1} \text{ for } a > 1 \quad [S-6]
  \]
  - \[
  \sum_{i=1}^{n} \frac{1}{2^i} = 1 - \frac{1}{2^n} \quad [S-7]
  \]
  - \[
  \sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \quad [S-8]
  \]
  - \[
  \sum_{i=0}^{\log n} 2^i = 2^{\log(n+1)} - 1 = 2n - 1 \quad [S-9]
  \]
  - \[
  H_n = \sum_{i=1}^{n} \frac{1}{i}; \log_e n < H_n < 1 + \log_e n \quad [S-10]
  \]
C++ Review

You are expected to understand the following C++ concepts:

- **C++ Classes**
  - Basic class syntax
  - Extra constructor syntax and accessors
  - Separation of interface and implementation
  - Vector and string

- **C++ Details**
  - Pointers
  - Parameter passing
  - Return passing
  - Reference variables
  - Destructor, copy constructor, operator=
  - C-style constructs

---

Methods to Define Generic Routines

- The `typedef` statement is a simple mechanism allowing a new type name to become a synonym for an old one

- Example: `typedef double real;`

- `typedef` can allow generic routines to be built.

- Example: simple swap routine.

  ```c++
  typedef double Type;
  void Swap(Type &lhs, Type &rhs) {
    Type tmp = lhs;
    lhs = rhs;
    rhs = tmp;
  }
  ```

- What are the disadvantages to this approach?

- What is desirable?

---

Templates

Templates are similar in some ways to `typedef`

- A template is a design for an object, as opposed to an actual object.

- Templates allow polymorphism through multiple separate instantiations of a defined template.

- Two major kinds of templates:
  - Template functions
  - Template classes

- A template function is a pattern for an actual function.

- Multiple instantiations are possible by declaring it using different types.
Use of Template Functions

- Example: templated swap function and main that calls it:

```cpp
template <class Type>
void swap (Type & lhs, Type & rhs) {
    Type tmp = lhs;
    lhs = rhs;
    rhs = tmp;
}

main() {
    int x = 5;
    int y = 7;
    double a = 21;
    double b = 34;
    swap(x, y);     // Instantiate swap w/int.
    swap(x, y);     // Use prior instance
    swap(a, b);     // Instantiate swap w/float

    // swap(x, b);  // This is illegal.
}
```

Template Classes

- A template class allows multiple object instantiations.
- The compiler automatically creates code for any necessary versions of template classes.
- Syntax is more involved than that of template functions.
- Example: a non-templated memory cell class.

```cpp
// MemoryCell class
// int Read() -> returns the stored value
// void Write(int X) -> X is stored

class MemoryCell {
    private:
        int StoredValue;
    public:
        int Read() { return StoredValue; }
        void Write(int X) { StoredValue = X; }
};
```

Template Classes (cont.)

- Example: Using the MemoryCell class in main

```cpp
main() {
    MemoryCell M;
    M.Write(6);
    cout << "Cell contents are ";
    cout << M.Read() << endl;
}
```

- How can we declare MemoryCell objects of other types?

Template Classes (cont.)

- Example: a float MemoryCell

```cpp
// MemoryCell2 class
// float Read() -> returns the stored value
// void Write(float X) -> X is stored

class MemoryCell2 {
    private:
        float fStoreValue;
    public:
        float Read() { return fStoreValue; }
        void Write(float X) { fStoreValue = X; }
};

main() {
    MemoryCell2 M2;
    M2.Write(6.3);
    cout << "Cell contents are ";
    cout << M2.Read() << endl;
}
Templated MemoryCell Class

- Note the use of the template keyword:

```
// MemoryCell class
// Stype Read() -> returns the stored value
// void Write(Stype X) -> X is stored

template <class Stype>
class MemoryCell {
  private:
    Stype StoredValue;
  public:
    const Stype & Read() const {
      return StoredValue;
    }
    void Write(const Stype & X) {
      StoredValue = X;
    }
}
```

Templated MemoryCell Class (cont.)

- Note how MemoryCells of different types are declared below:

```
main() {
  MemoryCell<int> Mi;
  MemoryCell<float> Mf;
  Mi.Write(5);  // Write 5
  Mf.Write(5.34); // Write 5.34
  cout << "int contents: ";
  cout << Mi.Read() << endl;
  cout << "float contents: ";
  cout << Mf.Read() << endl;
}
```

Friends

The *friend* declaration allows you to grant access to private class members.

- Motivation:

  - sometimes functions/classes are used in conjunction with other classes
  - Can reduce overhead (runtime)

- Types of friends:

  - Friend functions: friend keyword precedes function prototype in class definition
  - Friend operators (these are really functions, also)
  - Friend classes

Friend Functions

- Example:

```
class Euro {
  private:
    long data;
  public:
    Euro operator/(double x) {
      return (*this * (1.0/x));
    }
    friend Euro operator+ (const Euro& e1, const Euro& e2);
    friend Euro operator- (const Euro& e1, const Euro& e2);
    friend Euro operator* (const Euro& e, double x) {
      Euro temp( ((double)e.data/100.0) * x);
      return temp;
    }
    friend Euro operator* (double x, const Euro& e) {
      return e * x;
    }
};
```
Friend Classes

- All methods in the friend class become friend functions in the class containing the friend declaration

- The class containing the friend declaration decides who its friends are

- Example:

```cpp
class Result {
private:
    double val;
    DayTime time;
public:
    friend class ControlPoint;
};

class ControlPoint {
private:
    string name;
    Result measure[100];
public:
    bool statistic(); // Can access private members // of measure[1]
};
```

- You will see more examples of this in the book and course notes

CSC 375-Turner, Page 25