1 Foundations

These definitions form the foundation upon which other definitions are constructed. You should re-visit these definitions from time to time because it’s easy to forget some of the details. We will begin with mathematical sets, because these simple structures can be used to build all of the others that we will implement in this class. And, as we continue the course, more definitions will appear in this document.

**Definition (Set).** A set is an unordered collection of distinct objects called its “elements.”

Sets may be written in several ways. If the number of elements is small, the “roster” format is often used.

\[ S = \{1, 2, 3\} \]

defines the set \( S \) that contains three elements. Note also that elements contained within a set are distinct. Thus,

\[ T = \{1, 2, 3, 1, 2, 3\} \]

defines another set \( T \) which happens to be equal to \( S \) because duplicates are ignored (or disallowed in the case of programming applications).

More common, however, are sets that describe large collections, such as the set of all non-negative even integers. Typically, these sets are written in “set-builder” notation, thus:

\[ E = \{n \mid n \text{ is an non-negative integer and } n \text{ is even}\} \]

Or, more formally,

\[ E = \{n \mid n \in \mathbb{Z}^+ \text{ and even}(n)\} \]

Given the abstract nature of the definition, sets have few properties. Perhaps the most important is **cardinality**, or the number of elements contained in the set. This is written variously, \(|S|\) or \(\#(S)\) or \(n(S)\)—which is 3 in the example above.

In computer science applications, sets are generally finite, and so the number of elements contained in a set is a non-negative integer, i.e., an integer \(\geq 0\). Note that a set may be empty, which is written \(\emptyset\), or \(\{\}\), and \(|\emptyset| = 0\).
1.1 Operations with sets

Numerous conventions arise in the contexts of sets. Frequently, we wish to say that an element is contained within a set:

\[ 1 \in \{1, 2, 3\} \]

whereas, one can say that a particular object is not contained within a set:

\[ 4 \notin \{1, 2, 3\} \]

Or, more likely something like:

\[ 1 \notin \{n \mid n \in \mathbb{Z}^+ \text{ and } \text{even}(n)\} \]

which is another way of saying that the integer 1 is not in the set of non-negative even integers.

A set may be contained within another set, as in the set of all apples are contained in the set named “fruit.” In these cases, we say that one set “apples” is a “subset” of the set “fruit.”

Definition (subset). The set \( S \) is a subset of set \( T \) when all elements contained within \( S \) appear in \( T \).

The subset relation is written in several ways, we will use the \( \subseteq \) symbol as in \( S \subseteq T \) to indicate that \( S \) is a subset of \( T \). Note that two sets \( A \) and \( B \) are equal when both \( A \subseteq B \) and \( B \subseteq A \).

1.2 Sets in programming contexts

Again, in computer science applications sets are finite, comprised of discrete, well-defined elements. And, because sets appear in computational settings, sets are generally provided in a programming language with additional operations.

In the Java collections classes, the method \( \text{size()} \) is generally equivalent to cardinality in that it gives the number of elements in the set. Likewise, the method \( \text{contains} \) that appears in various interfaces and classes is equivalent to the “element of” or \( \in \) operator above. Ask: how would you implement the \( \text{equals()} \) method? As we progress through this course, we will identify similar analogs.

1.3 Subsets and finer points

Notice, I didn’t talk in detail about support in the Java language for “subset” relations—and with good reason. Generally, the notion of a “sub” object requires some mathematical sophistication on the part of the implementor as well as the user. A sub-object preserves the properties of the containing object but is in some sense “smaller.” However, it is also
admissible that a subset be the same size at its containing set, which is why the underbar appears in the symbol \( \subseteq \), and so a set may be a subset of itself. Applying this reasoning to the empty set leads us to conclude that the empty set is the subset of any set, including the empty set.

Some collections classes in Java do support sub-objects, others do not. The `String` class provides a `substring` method that captures these ideas nicely. A substring must preserve all of the properties of any String. In particular, the empty `String` is a substring any any `String`. As we proceed through this course together, we will challenge you to implement sub-objects in at least one instance.

2 Building with sets

Sets can be combined by various rules to construct new sets. Three operators are commonly defined over sets. The union of two (or more) sets is a new set that contains the elements that appear in either. More formally:

\[
A \cup B = \{ e \mid e \in A \text{ or } e \in B \}.
\]

Restricting our combination logic to require that an element appear in both sets gives us the definition of the intersection of two sets:

\[
A \cap B = \{ e \mid e \in A \text{ and } e \in B \}.
\]

And, we can express the notion of “difference” as the complement or set-difference of two sets to be that set that contains elements of one but not the other:

\[
A - B = \{ e \mid e \in A \text{ and } e \notin B \}.
\]

Generally, these operators are not provided by the Java language—which means that we will implement these ourselves.

3 Sequences and their relation to sets

Sets are restrictive in the sense that they do not admit duplicate members, nor do they distinguish order. In other words: \{a, b, c\} = \{b, b, a, c\}. But, many commonly used collections do allow duplications and do respect order: think about `Strings`. Let’s define the term:

**Definition** (sequence). A sequence is an ordered collection of objects in which repetition is allowed.
In computer science, sequences often begin at 0 and written as

\[ a_0, a_1, a_2, \ldots, a_{n-1} \]

which would denote the sequence of terms \( a \) from 0 through \( n - 1 \), which contains \( n \) terms. More formally, sequences be defined in terms of functions. First, we need to define what we mean by “function.”

**Definition (function).** A *function* is a *unitary mapping* from one set, called the function’s *domain*, to another set, called the function’s *codomain*.

Functions can be visualized by internal diagrams, tables, or other kinds of descriptions, and these are useful when we wish to focus on the particular “rule” by which this mapping is done. In this, and in a majority of documents relating to computer science, however, we are more interested in the properties of a function’s domain and codomain. I use *external diagrams* for this purpose: \( f: A \rightarrow B \) is an external diagram of the function \( f \) whose domain is the set \( A \) and its codomain the set \( B \).

A sequence can now be visualized as a function from the non-negative integers, \( 0, 1, \ldots, n \) to a set of objects. Rewriting the last definition, more formally,

**Definition (Sequence).** A *sequence* is a function from the natural numbers: \( f: \mathbb{N} \rightarrow C \), where \( \mathbb{N} \) denotes the natural numbers, \( \{0, 1, \ldots\} \).

**Example (Text Searching).** A *snippet* is a map from \( T \), a non-empty, finite set called “search terms,” to a “document,” \( D \), which is a sequence of *Strings*: \( s: T \rightarrow D \). Observe that \( T \), by definition, contains no duplicates and is unordered.

### 3.1 Properties of functions

Although this is not a mathematics class, it’s instructive to focus on some properties of common functions that we will encounter in this course. For one, we will focus on “total functions,” which are functions that map *all* points in their domains. In other words, total functions leave no point in their domains unmapped. The actual rule is unimportant.

**Example (Equality).** A very important example of total functions is Java’s *equals()* method:

\[ \text{equals}: O \times O \rightarrow \mathbb{B}, \]

where \( O \) denotes the set of all objects, \( O \times O \) denotes the product of all pairings of objects, and \( \mathbb{B} \) the set containing only \{false, true\}.

Other properties of functions include “cancellation” properties, which are important in later courses.
4 Sets are used to build structures

Structures may be typically represented as sets equipped with particular operations. One important structure that we discussed in class was the “graph,” which is defined

**Definition** (Graph). A graph $G = (V, E)$ is a structure containing two sets, $V$, which a set of vertices and $E$ edges.

Vertices appear as “points” or “nodes” and edges as lines to connect them. Think of a roadmap: vertices are cities and edges highways. Many common data-structures can be represented as various types of graphs. And, we will make a more systematic study of various types of graphs in the second half of the class.