# The Flix Programming Language

## The Core Datatypes

## Opt

An option Opt is either None or Some(v). *Options cannot be nested[[1]](#footnote-1).*

|  |
| --- |
| null(o: Opt[A]): Bool  Returns true iff o is None. |
| get(o: Opt[A]): A [Unsafe]  Returns v if o is Some(v). Otherwise the semantics is undefined. |
| getWithDefault(o: Opt[A], a: A): A  Returns v if o is Some(v). Otherwise returns a. |
| exists(f: A => Bool, o: Opt[A]): Bool  Returns the value of f(v) if o is Some(v). Otherwise false. |
| forall(f: A => Bool, o: Opt[A]): Bool  Returns the value of f(v) if o is Some(v). Otherwise true. |
| filter(f: A => Bool, o: Opt[A]): Opt[A]  Returns o if o is Some(v) and f(v) is true. Otherwise returns None. |
| map(f: A => B, o: Opt[A]): Opt[B]  Returns Some(f(v)) if o is Some(v). Otherwise returns None. |
| map2(f: (A, B) => C, o1: Opt[A], o2: Opt[B]): Opt[C]  Returns Some(f(v1, v2)) if o1 is Some(v1) and Some(v2).  Otherwise returns None. |
| flatMap(f: A => Opt[B], o: Opt[A]): Opt[B]  Returns f(v) if o is Some(v). Otherwise returns None. |
| flatMap2(f: (A, B) => Opt[C],  o1: Opt[A], o2: Opt[A]): Opt[C]  Returns f(v1, v2) if o1 is Some(v1) and o2 is Some(v2).  Otherwise returns None. |
| toList(o: Opt[A]): List[A]  Returns a one-element list of the value v if o is Some(v).  Otherwise returns the empty list. |
| toSet(o: Opt[A]): Set[A]  Returns a one-element set of the value v if o is Some(v).  Otherwise returns the empty set. |
| withDefault(o1: Opt[A], o2: Opt[A]): Opt[A]  Returns o1 if it is Some(v) otherwise returns o2. |

## List

A list is either the empty list Nil or a cons cell v :: vs.

### Basic Operations

|  |
| --- |
| Nil: List[A]  The empty list. |
| v :: vs: List[A]  The list with the element v followed by the list vs. |
| null(xs: List[A]): Bool  Returns true iff xs is the empty list, i.e. Nil. |
| head(xs: List[A]): A [Unsafe]  Returns the first element of the list.  If the list is empty the semantics are undefined. |
| tail(xs: List[A]): List[A] [Unsafe]  Returns the list consisting of everything but the first element of the list.  If the list is empty the semantics are undefined. |
| init(xs: List[A]): List[A] [Unsafe]  Returns the list with all the elements except the last.  If the list is empty the semantics are undefined. |
| last(xs: List[A]): A [Unsafe]  Returns the last element of the list.  If the list is empty the semantics are undefined. |
| length(xs: List[A]): Int  Returns the length of the list. |
| append(xs: List[A], ys: List[A]): List[A]  Returns the length of the list. |
| at(position: Int, xs: List[A]): A  Returns the element at position in the list. |
| range(Int, Int)  Returns a list |
| repeat(A, Int) |
| permutations(List[A]): List  fdsfds |
|  |
| map(f: A => B, xs: List[A]): List[B]  Foo |
| flatMap(f: A => List[B], xs: List[A]): List[B] |
|  |
|  |
|  |

### Index Operations

### Sub Lists

|  |
| --- |
| filter(f: A => Bool, xs: List[A]): List[A]  Returns a list of every element x in xs for which f(x) is true. |
| slice(b: Int, e: Int, xs: List[A]): List[A]  Returns the sublist from position b (inclusive) until position e (exclusive). |
| take(n: Int, xs: List[A]): List[A]  Returns the first n elements of xs. |
| takeWhile(f: A => Bool, xs: List[A]): List[A]  Returns the longest prefix of xs for which the predicate f holds. |
| drop(n: Int, xs: List[A]): List[A]  fdsffs |
| dropWhile(f: A => Bool, xs: List[A]): List[A]  fsfa |

### Zipping and Unzipping

|  |
| --- |
| zip(xs: List[A], ys: List[B]): List[(A, B)] [Unsafe]  Returns a list where each element at index i is the pair (a, b) where a is the element at position i in xs and b is the element at position i in ys.  If xs and ys are of unequal length the semantics are undefined. |
| zipWith(f: (A, B) => C,  xs: List[A], ys: List[B]): List[C] [Unsafe]  Like zip but uses the function f to compute each element in the result.  Functionally equivalent to: map(f, zip(xs, ys)). |
| unzip(xs: List[(A, B)]): (List[A], List[B])  Returns a pair where the first component is a list of all first components in xs and the second component is a list of all second components in xs. |

### Order and Lattice Operation

|  |
| --- |
| leq(xs: List[A], ys: List[A]): Bool  Pairwise compares the elements of the two lists xs and ys according to the partial order. Returns true iff every element in the list xs is smaller than its corresponding element in ys. Functionally equivalent to to the expression zipWith(leq, xs, ys).and.  If the lists xs and ys are of unequal lengths the semantics are undefined. |
| null(o: Opt[A]): Bool |

## Set

### Basic Operations

|  |
| --- |
| null(s: Set[A]): Bool  Returns true iff s is the empty set. |

## Map

### Basic Operations

|  |
| --- |
| null(m: Map[K, V]): Bool  Returns true iff m is the empty map. |
| get(k: K, m: Map[K, V]): Opt[V]  Optionally returns the value the key k is mapped to in the map m.  Returns None if m has no mapping for k. |
| getWithDefault(k: K, v: V, m: Map[K, V]): V  Returns the value the key k is mapped to in the map m. Returns v if the key has no mapping. |
| memberOf(k: K, m: Map[K, V]): Bool  Returns true iff k is key of the map m. |
|  |

### Insert, Update and Delete Operations

|  |
| --- |
| insert(k: K, v: V, m: Map[K, V]): Map[K, V]  Returns the map m updated with the key k mapped to the value v. |
| insertWith(f: (V, V) => V, k: K,  v: V, m: Map[K, V]): Map[K, V]  Returns the map m updated with the key k mapped to the value v. If the key already exists in the map and is mapped to v2, then the map is updated with the value of f(v, v2). |
| insertWithKey(f: (K, V, V) => V, k: K,  v: V, m: Map[K, V]): Map[K, V]  Returns the map m updated with the key k mapped to the value v. If the key already exists in the map and is mapped to v2, then the map is updated with the value of f(k, v, v2). |
| update(f: V => Opt[V], k: K, m: Map[K, V]): Map[K, V]  Returns ??? |

## Debug

The following *compiler directives* may aid debugging of Flix programs.

*Note: The Flix compiler and runtime is free to ignore these directives.*

|  |
| --- |
| abort!(m: String): Unit  Immediately aborts execution and prints the given message m. |
| print!(a: A): A  Prints the value a and returns it. |
| time!(f: () => A): A  Evaluates the function f while measuring its execution time.  Returns the result of evlauating the function. |

1. This ensures efficient representation of options. [↑](#footnote-ref-1)