Table of Contents

bool

This package provides utility functions for working with boolean values in MoonB
it, primarily focused on type conversions that are useful in systems programming
, bitwise operations, and numerical computations.

Overview

Boolean values in MoonBit can be seamlessly converted to numeric types, followin
g the standard convention where true maps to 1 and false maps to 0. This is part
icularly useful for:

- Conditional arithmetic and accumulation
- Interfacing with C libraries or low-level code
- Implementing boolean algebra with numeric operations
- Converting logical results to flags or indices

Basic Integer Conversion

Convert boolean values to standard integers for arithmetic operations:

```
1
2   test "bool to integer conversions" {
3
4     inspect(true to_int(), content="1")
5     inspect(false to_int(), content="0")
6
7
8     let score = 100
9     let bonus_applied = true
10    let final_score = score + bonus_applied to_int() * 50
11    inspect(final_score, content="150")
12
13
14    let conditions = [true, false, true, true, false]
15    let count = conditions fold(init=0, fn(acc, cond) { acc + cond to_int(
16    inspect(count, content="3")
17  }
```

Specialized Integer Types

For specific use cases requiring different integer widths and signedness:

```
1
2   test "bool to specialized integer types" {
3     let flag = true
4     let no_flag = false
5
6
7     inspect(flag.to_uint(), content="1")
8     inspect(no_flag.to_uint(), content="0")
9
10
11    inspect(flag.to_int64(), content="1")
12    inspect(no_flag.to_int64(), content="0")
13
14
15    inspect(flag.to_uint64(), content="1")
16    inspect(no_flag.to_uint64(), content="0")
17  }
```

Practical Use Cases

Boolean Indexing and Selection

```
1
2   test "boolean indexing" {
3
4     let options = ["default", "enhanced"]
5     let use_enhanced = true
6     let selected = options[use_enhanced.to_int()]
7     inspect(selected, content="enhanced")
8
9
10    let base_value = 10
11    let multiplier = 2
12    let apply_multiplier = false
13    let result = base_value * (1 + apply_multiplier.to_int() * (multiplier
14    inspect(result, content="10")
15  }
```

Bit Manipulation and Flags

```
1
2   test "flags and bit operations" {
3
4     let read_permission = true
5     let write_permission = false
6     let execute_permission = true
7     let permissions = (read_permission.to_uint() << 2) |
8       (write_permission.to_uint() << 1) |
9       execute_permission.to_uint()
10    inspect(permissions, content="5")
11  }
```

Statistical and Mathematical Operations

```

  test "statistical operations" {

    let test_results = [true, true, false, true, false, true, true]
    let successes = test_results fold(init=0, fn(acc, result) {
      acc + result to_int()
    })
    let total = test_results length()
    let success_rate = successes to_double() / total to_double()
    inspect(success_rate > 0.7, content="true")


    let feature_enabled = [true, false, true]
    let weights = [0.6, 0.3, 0.1]


    let score1 = feature_enabled[0] to_int() to_double() * weights[0]
    let score2 = feature_enabled[1] to_int() to_double() * weights[1]
    let score3 = feature_enabled[2] to_int() to_double() * weights[2]
    let weighted_score = score1 + score2 + score3
    inspect(weighted_score == 0.7, content="true")
  }
```

This package provides the essential bridge between MoonBit's boolean logic and n
umeric computations, enabling elegant solutions for conditional arithmetic, flag
operations, and data processing workflows.