Table of Contents

bool

This package provides utility functions for working with boolean values in MoonBit, 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, following the standard convention where true maps to 1 and false maps to 0. This is particularly 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:

```

test "bool to integer conversions" {

  inspect(true.to_int(), content="1")
  inspect(false.to_int(), content="0")


  let score = 100
  let bonus_applied = true
  let final_score = score + bonus_applied.to_int() * 50
  inspect(final_score, content="150")


  let conditions = [true, false, true, true, false]
  let count = conditions.fold(init=0, fn(acc, cond) { acc + cond.to_int(
  inspect(count, content="3")
}
```

Specialized Integer Types

For specific use cases requiring different integer widths and signedness:

```

test "bool to specialized integer types" {
  let flag = true
  let no_flag = false


  inspect(flag.to_uint(), content="1")
  inspect(no_flag.to_uint(), content="0")


  inspect(flag.to_int64(), content="1")
  inspect(no_flag.to_int64(), content="0")


  inspect(flag.to_uint64(), content="1")
  inspect(no_flag.to_uint64(), content="0")
}
```

Practical Use Cases

Boolean Indexing and Selection

```

test "boolean indexing" {

  let options = ["default", "enhanced"]
  let use_enhanced = true
  let selected = options[use_enhanced.to_int()]
  inspect(selected, content="enhanced")


  let base_value = 10
  let multiplier = 2
  let apply_multiplier = false
  let result = base_value * (1 + apply_multiplier.to_int() * (multiplier
  inspect(result, content="10")
}
```

Bit Manipulation and Flags

```

test "flags and bit operations" {

  let read_permission = true
  let write_permission = false
  let execute_permission = true
  let permissions = (read_permission.to_uint() << 2) |
    (write_permission.to_uint() << 1) |
    execute_permission.to_uint()
  inspect(permissions, content="5")
}
```

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.