Maple 16 のオブジェクト

Maple ユーザーは言語にオブジェクトを追加することで、Maple との親和性が非常に高いコードを作成することができます。オブジェクトの概要については、object ヘルプページを参照してください。新規オブジェクトの宣言については、object,create ヘルプページを参照してください。オブジェクトを使用すると、メソッドの実装 (object,methods を参照)、組込みオブジェクトの多重定義 (object,builtins を参照)、標準の Maple 構文で該当するオブジェクトを指定した場合に起動するさまざまな演算子の使用 (object,operatorsを参照) が可能になります。これにより、あたかも Maple の組込みタイプであるかのように、ユーザー定義オブジェクトを Maple に組み込むことができます。次の例では、n を法とした整数を表すオブジェクトを実装します。

```
module IntMod()
    option object;

    local base := 1;
    local value := 0;

    # implement a ModuleApply and ModuleCopy routine
    # to use IntMod() as a object constructor
    export ModuleApply::static := proc()
        Object( IntMod, _passed );
    end;

    export ModuleCopy::Static := proc( self::IntMod, proto::IntMod,
                    v::integer, b::integer, $ )
        if ( _npassed < 3 ) then
            # if a value was not given, copy the prototype
            self:-value := proto:-value;
        else
            # use the given value
            self:-value := v;
        end;

        if ( _npassed < 4 ) then
            # if a base was not given, opy the prototype
            self:-base := proto:-base;
        else
            # use the given base
            self:-base := b;
        end;

        # apply the mod
        self:-value := self:-value mod self:-base;
    end;

    # implement ModulePrint to make the display nice
    export ModulePrint::static := proc( self::IntMod )
        nprintf( "%d mod %d", self:-value, self:-base );
    end;

    # Implement Module type to enable type checking with
    # with a specified base
    export ModuleType::static := proc( self, type, b, $ )
        if ( _npassed = 2 ) then
            true;
        else
            evalb( self:-base = b );
        end;
    end;

    # accessor to get the stored value
    export getValue::static := proc( self::IntMod )
        self:-value;
    end;

    # implement the + operator to allow IntMods to work properly
    # in sum statements.  We assume that the modular arthemetic is
    # contagious and the entire addition become modular
    export `+`::static := proc( )
        local ints, imods, total, base, other;

        # remove the stuff we aren't going to manipulate
        ( ints, other ) := selectremove( type, [_passed],
                                    { 'IntMod', 'integer' } );

        ( imods, ints ) := selectremove( type, ints, 'IntMod' );

        base := imods[1]:-base;
        if ( not andmap( type, imods, 'IntMod'( base ) ) ) then
            error "all IntMods must be of the same base"
        end;

        #Add the IntMods and the integer constant, if one exists
        total := ( `if`( numelems(ints) > 0, ints[1], 0 ) +
                    add( getValue(i), i in imods ) ) mod base;

        # return a sequence
```

Create a new instance of the IntMod class using the IntMod( ) object factory.

```
> i0m5 := IntMod( 0, 5 );
```

$$i0m5 := 0 \bmod 5 \tag{1}$$

フォームがきちんとプリントされている点に注意してください。

また、既存のオブジェクトを複製することで新規オブジェクトを作成することもできます。 この場合、新しい値を指定しますが、ベースは指定したオブジェクトからコピーされます。

```
> i1m5 := Object( i0m5, 1 );
```

$$i1m5 := 1 \bmod 5 \tag{2}$$

新規作成したこれらのオブジェクトが IntMod クラスのインスタンスであることをテストできます。

```
> type( i1m5, 'IntMod' );
```

$$\text{true} \tag{3}$$

ModuleType プロシージャを使用すると、IntMod タイプがベースの指定をサポートしているかどうかを確認できます。

```
> type( i1m5, 'IntMod'(3) );
```

$$\text{false} \tag{4}$$

```
> type( i1m5, 'IntMod'(5) );
```

$$\text{true} \tag{5}$$

ModInt は '+' 演算子を実装します。この演算子を使って、残りの 5 を法とした整数を生成できます。

```
> i2m5 := i1m5 + 1;
```

$$i2m5 := 2 \bmod 5 \tag{6}$$

```
> i3m5 := i2m5 + 1;
```

$$i3m5 := 3 \bmod 5 \tag{7}$$

```
> i4m5 := i3m5 + 1;
```

$$i4m5 := 4 \bmod 5 \tag{8}$$

```
> i4m5 + 1;
```

$$0 \bmod 5 \tag{9}$$

```
> i3m5 + i4m5;
```

$$2 \bmod 5 \tag{10}$$

+ を実装すると、非整数値を処理対象外とすることで、非整数値もサポートされます。

```
> 4 + i1m5 + 18 + i2m5 + 3 + f(x) + y;
```

$$3 \bmod 5 + f(x) + y \tag{11}$$

IntMod は * と ^ 演算子を多重定義します。

```
> i3m5 * i4m5 * y * f(x);
```

$$2 \bmod 5\, y\, f(x) \tag{12}$$

式に 0 が含まれる場合、^ の呼び出しは行われません。これは自動簡単化によって呼び出しが妨げ

られるためです。

> *i2m5*^0;

$$1 \tag{13}$$

> *i2m5*^1;

$$2 \bmod 5 \tag{14}$$

> *i2m5*^2;

$$4 \bmod 5 \tag{15}$$

> *i2m5*^3;

$$3 \bmod 5 \tag{16}$$

> *i2m5*^4;

$$1 \bmod 5 \tag{17}$$

> *i2m5*^5;

$$2 \bmod 5 \tag{18}$$

ModInt オブジェクトは、変換関数の多重定義も行います。このため、IntMods を整数に変換することができます。

> *convert*(*i3m5*, '*integer*');

$$3 \tag{19}$$