numeration-bases

This project implements conversions between numeration bases, as well as operations in arbitrary numeration bases (for bases between 2 and 16).

Interface

The program uses a console-based interface that is meant to replicate the mathematical notation of the performed operations.

As such, in order to convert a number between two bases, the user should input `n(b) = ?(h)`, where `n` is the number to convert, `b` is the source base, and `h` is the destination base. (e.g. `2A(16) = ?(10)`)

In order to perform an arithmetic operation, the user should input `x(b) <op> y(b)`, where `<op>` is one of `+`, `-`, `*` or `/`, `x` and `y` are the operands, and `b` is the base in which the operation is performed. (e.g. `44(8) + 56(8)`)

Supported operations

The following arithmetic operations are supported for any base between 2 and 16:

- Addition (e.g. `44(8) + 56(8)`)
- Subtraction (e.g. `5FA(16) - DD(16)`)
- Multiplication by 1 digit (e.g. `2A(12) * 2(12)`)
- Division by 1 digit (e.g. `3C(16) / 4(16)`)
    - Both the quotient and the remainder of the division are computed

Supported conversion methods

The program supports multiple conversion methods, and automatically selects the optimal one based on the source and destination base:

- The substitution method: Chosen when the source base is lower than the destination base
- The method of successive divisions: Chosen when the source base is greater than the destination base
- The method using an intermediary base 10: Chosen when both bases are strictly lower than 10. The number is converted from the source base to base 10 using the substitution method, then the result is converted into the destination base using the method of successive divisions
- Rapid conversions: Chosen when both bases are powers of 2 (takes precedence over using 10 as intermediary base)

Implementation

The project is implemented in Haskell, a purely functional programming language. Because Haskell is a purely functional language, it does not support looping constructs. Their behaviour is achieved using recursive functions.

Used data structures

In the program, numbers in an arbitrary base are represented as strings. In Haskell, strings are just linked lists of characters, and the recursive nature of linked lists makes using them with recursive algorithms very easy. Also, because lists in Haskell grow from the front, not the back, the numbers' digits are stored in reverse order.

In the `Operations.hs` file, we can see the definition of the data types used to represent numbers:

```
type Digit = Char
type Digits = String
type Base = Int
```

The types are just aliases for intrinsic Haskell types: `Digits` is the type of numbers with multiple digits, which is an alias for `String` (which is, in turn, an alias for a list of characters, i.e `[Char]`), and `Digit`, which is used for single-digit numbers, is an alias for a single `Char`.

Used algorithms

More details on the used algorithms can be found in the specifications and comments of functions in the source code.

Building from source

A compiled executable is also distributed along with the source code.

In order to build the source code, you will need to install the Haskell platform, then simply run `stack build` in the project's directory.