Contents

1 Module `Bred` : main project **1**

2 Module `Read` **1**

3 Module `Write` **1**

4 Module `Markov` **1**

1 Module `Bred` : The main project.

Entrypoint and working with command line arguments.

```
type output = Pervasives.out_channel -> bytes -> int -> int -> unit
```
Output could be to stdout or to a named file.

```
val learn : string -> string
```
Take filename and return it's content.

```
val main : int -> 'a -> string list -> 'b -> unit
```
Entrypoint for the bred program.

```
val deep : int Cmdliner.Term.t
```
CLI parameter to set deep of learning for the markov's chain.

```
val out : output Cmdliner.Term.t
```
CLI parameter to set output file/stdout.

```
val files : string list Cmdliner.Term.t
```
CLI parameter for list of files for chain's education.

```
val num : int Cmdliner.Term.t
```
CLI parameter to limit length of output.

```
val main_t : unit Cmdliner.Term.t
```
Build a CLI handler.

```
val info : Cmdliner.Term.info
```
Build an info page.

2 Module `Read` : There is module Read for transformation raw text to a convinient for markov's chains learning presentation.

```
val string_of_file : string -> string
```

Take a filename and return a string with it's content.

3 Module `Write` : The text generator module.

```
val generate_text : Markov.ptable -> string
```

Take a markov's chain and generate output.

4 Module `Markov` : There is constructing markov's chain.

```
type distribution = {
  total : int ;
  amounts : (string * int) list ;
}
```

The presentation for a word's distribution.

```
type ptable = {
  prefix_length : int ;
  table : (string list, distribution) Hashtbl.t ;
}
```

The type for a Markov's chain.

```
val is_word : char -> bool
```

Check that it's a alphanumerical symbol.

```
val is_punctuation : char -> bool
```

Check that it's a punctuation.

```
val is_sentence_separator : char -> bool
```

Chech that it's a separator between sentencies.

```
val split_word : string -> string list
```

Split a string to a list of words.

```
val start : int -> string list
```

Make the list with few "START" words according to depth of chain's learning.

`val shift : 'a list -> 'a -> 'a list`

Remove first element of a list and add new one to the end `1; 2; 3` 4 → `2; 3; 4`

`val add_to : ('a, 'b list) Hashtbl.t -> 'a -> 'b -> unit`

Add new word to a chain.

`val compute_distribution : string list -> distribution`

Construct a distribution from a list of words.

`val next_in_htable : ('a, distribution) Hashtbl.t -> 'a -> string`

Find a continuation for a given word in a distribution.

`val build_ptable : string list -> int -> ptable`

Take a list of words and a depth. Construct markov's chain.

`val walk_ptable : ptable -> string list`

Produce a random words' list from a given markov's chain.