BF instructions

The basic concept of BF is very simple. **Imagine a very large array of integers.** You can…

...**move left and right on it** with	`<` and `>`,
...**increment and decrement** the current cell with	`+` and `-`,
...**loop** until zero with	`[` and `]`, and
...**do input and output** (one character at a time) with	`,` and `.`.

(Some of the) EBF additions

Variables are named locations on the tape. Define them with `:var_name` and go to them with `$var_name`.

Macros are reusable snippets of code. Define them with `{mac_name body}` and insert them with `&mac_name`.

Self-balancing brackets: this EBF code `(>>>)  :a:b $a(-$b+)` compiles into this BF code `[>>><<<]  [->+<]`.

Multiplicative notation curbs repetition and aids readability. `3+` compiles into `+++`.

More details, examples, and language features can be found at https://code.google.com/p/ebf-compiler/.

EBF++ additions

Libraries and other files can be included with this syntax:

```
!'filename'
```

Macros have been extended with arguments.

```
{mac_name \ arg1 \ arg2... \\
  body (can contain %arg1...)}
&{mac_name / arg1 / arg2...}
```

Arrays and structs have been added, though in this version of EBF++, they can only exist together. Specifically:

- you **cannot** create structs by themselves, and
- **every** array is an array of some kind of struct.

You can declare a struct type like this:

```
:=Point { x y }
```

Then you can define an array, either by...

size:

```
::square Point 4
```

or by supplying initial values:

```
::triangle Point
  { 0 0 / 0 1 / 1 0 }
```

You can go to an index in an array, either by...

specifying a numerical index:

```
$:triangle 2
```

or a variable containing the index:

```
$!triangle index_var
```

Now you're in a struct. You can access a member like this:

```
$$x
```

And you can iterate over array elements like this:

```
~triangle {
  body (can contain $$x...)}
```