Bf programming language

Minimal: 6 commands, 1 loop

```
>  increment memory pointer 1
<  decrement memory pointer 1
+  increment byte pointed at by 1
-  increment byte pointed at by 1
.  copy byte pointed at to stdout
,  copy byte from stdin to byte pointed
[  if byte pointed at is zero jump past matching ]
]  jump to matching [
```

All other characters ignored

Bf programming language

Memory initialized to all zero

Memory pointer starts at index 0

Instructions executed in sequence

Example: Add two non-negative numbers

Pseudocode:

```
a = read number
b = read number
while b != 0:
    b -= 1
    a += 1
print a
```

Bf:

```
,>,[-<+>]<.
```

Bf grammar

$$S \rightarrow CS \mid LS \mid \lambda$$
$$L \rightarrow [\ S\ ]$$
$$C \rightarrow > \mid < \mid + \mid - \mid , \mid .$$

Tree builder

$$S \rightarrow CS \mid LS \mid \lambda$$

```
def parseS(toks):
    rval = node('S')
    tok = toks.next()
    if tok in ('>', '<', '+', '-', ',', '.'):
        rval.add_child(parseC(toks))
        rval.add_child(parseS(toks))
    elif tok == '[':
        rval.add_child(parseL(toks))
        rval.add_child(parseS(toks))
    elif tok == None or tok == ']':
        rval.add_child(node(''))
    else:
        raise Exception
    return rval
```

Tree builder

$$L \rightarrow [\ S\ ]$$

```
def parseL(toks):
    rval = node('L')
    tok = toks.next()
    if tok == '[':
        toks.match('[')
        rval.add_child(node('['))
        rval.add_child(parseS(toks))
        toks.match(']')
        rval.add_child(node(']'))
    else:
        raise Exception
    return rval
```

Tree builder

C → > | < | + | - | , | .

```
def parseC(toks):
    rval = node('C')
    tok = toks.next()
    if tok in ('>', '<', '+', '-', ',', '.'):
        toks.match(tok)
        rval.add_child(node(tok))
    else:
        raise Exception
    return rval
```

Tree interpreter

```
def interpret(tree_node):
    if tree_node.data == "S":
        if len(tree_node.children) == 2:
            interpret(tree_node.children[0])
            interpret(tree_node.children[1])
    elif tree_node.data == "C":
        interpret(tree_node.children[0])
    elif tree_node.data == "L":
        while mem[midx] != 0
            interpret(tree_node.children[1])
    elif tree_node.data == "+":
        mem[midx] += 1
    elif tree_node.data == "-":
        mem[midx] -= 1
    elif tree_node.data == ">":
        midx += 1
    elif tree_node.data == "<":
        midx -= 1
    elif tree_node.data == ".":
        sys.stdin.write(mem[midx])
    elif tree_node.data == ",":
        mem[midx] = sys.stdin.read(1)
}
```