```ts
import { Rule } from '../types'

// string.ts
export type StringType = 'string'

export const string = (msg?: string): Rule<StringType, string> => ({
  type: 'string',
  ...,
  func(ctx) {...}
})

// required.ts
export type RequiredType = 'required'

export const required = <Value extends any = any>(
  msg?: string
): Rule<RequiredType, Value> => ({
  type: 'required',
  ...,
  func(ctx) {...}
})
```

```
function fuji<Types extends RuleType, Value>(
  ...rules: Rule<Types, Value>[]
): Fuji<Types, Value> {
  return { rules: sortRules(rules) }
}
```

```ts
export const runner: RuleRunner = (schema, context) => {
  const { failFast } = context.config
  let runnerContext = context

  for (let i = 0; i < schema.rules.length; i++) {
    const rule = schema.rules[i]

    if (rule.canSkipCheck && shouldSkipCheck(runnerContext)) {
      continue
    }

    const nextContext = rule.func(runnerContext)

    if (failFast && nextContext.errors.length > 0) {
      return nextContext
    }

    runnerContext = nextContext
  }

  return runnerContext
}
```

```ts
export function run<Types extends RuleType, Value>(...): Result<Types, Value> {
  const configuration = createConfig(config)
  const context = createContext<Value>(value as Value, configuration)
  const output = runner<Value>(schema, context)

  if (output.errors.length > 0) {
    return {
      invalid: true,
      errors: output.errors,
      value: null
    }
  }

  return {
    invalid: false,
    value: output.current as Infer<Fuji<Types, Value>>,
    errors: null
  }
}
```

```ts
export function run<Types extends RuleType, Value>(...): Result<Types, Value> {
  const configuration = createConfig(config)
  const context = createContext<Value>(value as Value, configuration)
  const output = runner<Value>(schema, context)

  if (output.errors.length > 0) {
    return {
      invalid: true,
      errors: output.errors,
      value: null
    }
  }

  return {
    invalid: false,
    value: output.current as Infer<Fuji<Types, Value>>,
    errors: null
  }
}
```

```ts
type RuleFunc<A, B = A> = (ctx: VContext<A>) ⟹ VContext<B>

type Rule<Type extends RuleType, A, B = A> = {
  type: Type
  func: RuleFunc<A, B>
}

export type Fuji<Types extends RuleType, Value> = {
  rules: Rule<Types, Value>[]
}
```