Golang defines a benchmark using a function which receives a pointer to testing.B as its argument.

Eg;

becomes a Benchmark

```
func BenchmarkFib(b *testing.B){
    for(i:=0; i<b.N; i++){
        fibonacci(50)
    }
}
```

Each benchmark is represented as InternalBenchmark.

Each instance is represented by a name & a benchmark function.

```go
// InternalBenchmark is an internal type but exported because it is cross-package;
// it is part of the implementation of the "go test" command.
Implement interface
type InternalBenchmark struct {
    Name string
    F    func(b *B)
}
```

"go test" scans all the benchmarks & creates instances of InternalBenchmark.

benchmark files & fns.

Relationship between Internal-Benchmark & testing.B.

InternalBenchmark can be treated as benchmark definition & testing.B represents a runtime instance which facilitates benchmark execution.

Prior to benchmark execution, benchmarks are filtered, & an instance of testing.B is created which wraps all the filtered benchmarks in `benchFunc`.

Technically, InternalBenchmark instances are contained in a pointer to testing.B.

```go
var bs []InternalBenchmark
for _, Benchmark := range benchmarks {
    if _, matched, _ := bstate.match.fullName( c: nil, Benchmark.Name); matched {
        bs = append(bs, Benchmark)
        benchName := benchmarkName(Benchmark.Name, maxprocs)
        if l := len(benchName) + bstate.extLen + 1; l > bstate.maxLen {
            bstate.maxLen = l
        }
    }
}
main := &B{
    common: common{
        name:  "Main",
        w:     os.Stdout,
        bench: true,
    },
    importPath: importPath,
    benchFunc: func(b *B) {
        for _, Benchmark := range bs {
            b.Run(Benchmark.Name, Benchmark.F)
        }
    },
    benchTime: benchTime,
    bstate:    bstate,
}
```

→ filter benchmarks

| → wrap filtered benchmarks in a pointer lō B.