Switch

Spec: https://go.dev/ref/spec#Switch_statements

Go's `switch` statements are pretty neat. For one thing, you don't need to break at the end of each case.

```
switch c {
case '&':
    esc = "&amp;"
case '\'':
    esc = "&apos;"
case '<':
    esc = "&lt;"
case '>':
    esc = "&gt;"
case '"':
    esc = "&quot;"
default:
    panic("unrecognized escape character")
}
```

src/pkg/html/escape.go

Not just integers

Switches work on values of any type.

```
switch syscall.OS {
case "windows":
    sd = &sysDir{
        Getenv("SystemRoot") + `\system32\drivers\etc`,
        []string{
            "hosts",
            "networks",
            "protocol",
            "services",
        },
    }
case "plan9":
    sd = &sysDir{
        "/lib/ndb",
        []string{
            "common",
            "local",
        },
    }
default:
    sd = &sysDir{
        "/etc",
        []string{
```

```
            "group",
            "hosts",
            "passwd",
        },
    }
}
```

Missing expression

In fact, you don't need to switch on anything at all. A switch with no value means "switch true", making it a cleaner version of an if-else chain, as in this example from Effective Go:

```
func unhex(c byte) byte {
    switch {
    case '0' <= c && c <= '9':
        return c - '0'
    case 'a' <= c && c <= 'f':
        return c - 'a' + 10
    case 'A' <= c && c <= 'F':
        return c - 'A' + 10
    }
    return 0
}
```

Break

Go's `switch` statements `break` implicitly, but `break` is still useful:

```
command := ReadCommand()
argv := strings.Fields(command)
switch argv[0] {
case "echo":
    fmt.Print(argv[1:]...)
case "cat":
    if len(argv) <= 1 {
        fmt.Println("Usage: cat <filename>")
        break
    }
    PrintFile(argv[1])
default:
    fmt.Println("Unknown command; try 'echo' or 'cat'")
}
```

Fall through

To fall through to a subsequent case, use the `fallthrough` keyword:

```
v := 42
switch v {
```

```
case 100:
    fmt.Println(100)
    fallthrough
case 42:
    fmt.Println(42)
    fallthrough
case 1:
    fmt.Println(1)
    fallthrough
default:
    fmt.Println("default")
}
// Output:
// 42
// 1
// default
```

Another example:

```
// Unpack 4 bytes into uint32 to repack into base 85 5-byte.
var v uint32
switch len(src) {
default:
    v |= uint32(src[3])
    fallthrough
case 3:
    v |= uint32(src[2]) << 8
    fallthrough
case 2:
    v |= uint32(src[1]) << 16
    fallthrough
case 1:
    v |= uint32(src[0]) << 24
}
```

src/pkg/encoding/ascii85/ascii85.go

The 'fallthrough' must be the last thing in the case; you can't write something like

```
switch {
case f():
    if g() {
        fallthrough // Does not work!
    }
    h()
default:
    error()
}
```

However, you can work around this by using a 'labeled' `fallthrough`:

```
switch {
case f():
    if g() {
        goto nextCase // Works now!
    }
    h()
    break
nextCase:
    fallthrough
default:
    error()
}
```

Note: `fallthrough` does not work in type switch.

Multiple cases

If you want to use multiple values in the same case, use a comma-separated list.

```
func letterOp(code int) bool {
    switch chars[code].category {
    case "Lu", "Ll", "Lt", "Lm", "Lo":
        return true
    }
    return false
}
```

Type switch

With a type switch you can switch on the type of an interface value (only):

```
func typeName(v interface{}) string {
    switch v.(type) {
    case int:
        return "int"
    case string:
        return "string"
    default:
        return "unknown"
    }
}
```

You can also declare a variable and it will have the type of each `case`:

```
func do(v interface{}) string {
    switch u := v.(type) {
    case int:
        return strconv.Itoa(u*2) // u has type int
    case string:
```

```
        mid := len(u) / 2 // split - u has type string
        return u[mid:] + u[:mid] // join
    }
    return "unknown"
}

do(21) == "42"
do("bitrab") == "rabbit"
do(3.142) == "unknown"
```

Noop case

Sometimes it useful to have cases that require no action. This can look confusing, because it can appear that both the noop case and the subsequent case have the same action, but isn't so.

```
func pluralEnding(n int) string {
    ending := ""

    switch n {
    case 1:
    default:
        ending = "s"
    }

    return ending
}

fmt.Sprintf("foo%s\n", pluralEnding(1))  == "foo"
fmt.Sprintf("bar%s\n", pluralEnding(2))  == "bars"
```