

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7  )
8
9  func Pop (pstack *[]rune) (rune, error) {
10     if Empty(*pstack) {
11         return 0, errors.New("")
12     } else {
13         c:= (*pstack)[len(*pstack)-1]
14         *pstack = (*pstack)[:len(*pstack)-1]
15         return c, nil
16     }
17 }
18
19 func Push (pstack *[]rune, r rune) {
20     *pstack = append(*pstack, r)
21 }
22
23 func Empty(stack []rune) bool {
24     return len(stack) == 0
25 }
26
27 func check(brackets string) bool {
28     var stack []rune
29     for _, b := range []rune(brackets) {
30         switch b {
31             case '(', '[', '{':
32                 Push(&stack, b)
33             case ')':
34                 if top, err:= Pop(&stack); err != nil || top != '(' {
35                     return false
36                 }
37             case ']':
38                 if top, err:= Pop(&stack); err != nil || top != '[' {
39                     return false
40                 }
41             case '}':
42                 if top, err:= Pop(&stack); err != nil || top != '{' {
43                     return false
44                 }
45             default:
46                 return false
47         }
48     }
49     return Empty(stack)
50 }
51
52 func main() {
53     f, err := os.Open("brackets.dat")
54     if err != nil {
55         return
56     }
57     defer f.Close()
58     var line string
59     for {
60         _, err := fmt.Fscanf(f, "%s\n", &line)
61         if err != nil { break }
62         fmt.Println(line, check(line))
63     }
64 }

```

**Brackets\_01a.go**

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7  )
8
9  type (
10     node struct {
11         bracket rune
12         next *node
13     }
14     stack *node
15 )
16
17 func Pop (ps *stack) (rune, error) {
18     if Empty(*ps) {
19         return 0, errors.New("")
20     } else {
21         c:= (*ps).bracket
22         *ps = (*ps).next
23         return c, nil
24     }
25 }
26
27 func Push (ps *stack, r rune) {
28     p:= new(node)
29     (*p).bracket = r
30     (*p).next = *ps
31     *ps = p
32 }
33
34 func Empty(s stack) bool {
35     return s == nil
36 }
37
38 func check(brackets string) bool {
39     var s stack
40     for _, b := range []rune(brackets) {
41         switch b {
42             case '(', '[', '{':
43                 Push(&s, b)
44             case ')':
45                 if top, err:= Pop(&s); err != nil || top != '(' { return false }
46             case ']':
47                 if top, err:= Pop(&s); err != nil || top != '[' { return false }
48             case '}':
49                 if top, err:= Pop(&s); err != nil || top != '{' { return false }
50             default:
51                 return false
52         }
53     }
54     return Empty(s)
55 }
56
57 func main() {
58     f, err := os.Open("brackets.dat")
59     if err != nil {
60         return
61     }
62     defer f.Close()
63     var line string
64     for {
65         _, err := fmt.Fscanf(f, "%s\n", &line)
66         if err != nil { break }
67         fmt.Println(line, check(line))
68     }
69 }

```

**Brackets\_01b.go**

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7  )
8
9  type stack []rune
10
11 func (ps *stack) Pop () (rune, error) {
12     if (*ps).Empty() {
13         return 0, errors.New("Pop from empty stack")
14     } else {
15         c:= (*ps)[len(*ps)-1]
16         *ps = (*ps)[:len(*ps)-1]
17         return c, nil
18     }
19 }
20
21 func (ps *stack) Push (r rune) {
22     *ps = append(*ps, r)
23 }
24
25 func (s stack) Empty() bool {
26     return len(s) == 0
27 }
28
29 func check(brackets string) bool {
30     var s stack
31     for _, b := range []rune(brackets) {
32         switch b {
33             case '(', '[', '{':
34                 s.Push(b)
35             case ')':
36                 if top, err:= s.Pop(); err != nil || top != '(' { return false }
37             case ']':
38                 if top, err:= s.Pop(); err != nil || top != '[' { return false }
39             case '}':
40                 if top, err:= s.Pop(); err != nil || top != '{' { return false }
41             default:
42                 return false
43         }
44     }
45     return s.Empty()
46 }
47
48 func main() {
49     f, err := os.Open("brackets.dat")
50     if err != nil {
51         return
52     }
53     defer f.Close()
54     var line string
55     for {
56         _, err := fmt.Fscanf(f, "%s\n", &line)
57         if err != nil { break }
58         fmt.Println(line, check(line))
59     }
60 }

```

**Brackets\_02a.go**

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7  )
8
9  type lmnt struct {
10     bracket rune
11     next *lmnt
12 }
13
14 type stack struct {
15     head *lmnt
16 }
17
18 func (s *stack) Push(bracket rune) {
19     s.head = &lmnt{bracket, s.head}
20 }
21
22 func (s *stack) Pop() (rune, error) {
23     if s.head == nil {
24         return 0, errors.New("List is empty")
25     }
26     c := s.head.bracket
27     s.head = s.head.next
28     return c, nil
29 }
30
31 func (s stack) Empty() bool {
32     return s.head == nil
33 }
34
35 func NewStack() stack {
36     return stack{nil}
37 }
38
39 func check(brackets string) bool {
40     s := NewStack()
41     for _, b := range []rune(brackets) {
42         switch b {
43             case '(', '[', '{':
44                 s.Push(b)
45             case ')':
46                 if top, err := s.Pop(); err != nil || top != '(' { return false }
47             case ']':
48                 if top, err := s.Pop(); err != nil || top != '[' { return false }
49             case '}':
50                 if top, err := s.Pop(); err != nil || top != '{' { return false }
51             default:
52                 return false
53         }
54     }
55     return s.Empty()
56 }
57
58 func main() {
59     f, err := os.Open("brackets.dat")
60     if err != nil {
61         return
62     }
63     defer f.Close()
64     var line string
65     for {
66         _, err := fmt.Fscanf(f, "%s\n", &line)
67         if err != nil { break }
68         fmt.Println(line, check(line))
69     }
70 }

```

**Brackets\_02b.go**

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7      "math"
8  )
9
10 type queue []int
11
12 func NewQueue() queue {
13     return make([]int, 0, 0)
14 }
15
16 func (q *queue) Add (n int) {
17     *q = append(*q, n)
18 }
19
20 func (q queue) Max () int {
21     res := math.MinInt64
22     for _, x := range q {
23         if x > res { res = x }
24     }
25     return res
26 }
27
28
29 func (q *queue) Remove () error {
30     if (*q).Empty() {
31         return errors.New("Attempt to remove from empty queue")
32     } else {
33         *q = (*q)[1:]
34         return nil
35     }
36 }
37
38
39 func (q queue) Empty() bool {
40     return len(q) == 0
41 }
42
43 func main() {
44     f, err := os.Open("numbers.dat")
45     if err != nil {
46         return
47     }
48     defer f.Close()
49     var k int
50     _, err = fmt.Fscanf(f, "%d\n", &k)
51     if err != nil { return }
52     var x int
53     q := NewQueue()
54     for i := 0; i < k; i++ {
55         _, err := fmt.Fscanf(f, "%d\n", &x)
56         if err != nil { break }
57         q.Add(x)
58     }
59     for {
60         fmt.Println(q.Max())
61         _, err := fmt.Fscanf(f, "%d\n", &x)
62         if err != nil { break }
63         q.Add(x)
64         q.Remove()
65     }
66 }

```

Numbers\_01.go

```

1  package main
2
3  import (
4      "fmt"
5      "errors"
6      "os"
7      "math"
8  )
9
10 type lmnt struct {
11     n int
12     next *lmnt
13 }
14
15 type queue struct {
16     head *lmnt
17     tail *lmnt
18 }
19
20 func NewQueue() queue {
21     return queue{nil, nil}
22 }
23
24 func (q *queue) Add (n int) {
25     if (*q).Empty() {
26         (*q).tail = &lmnt{n, nil}
27         (*q).head = (*q).tail
28     } else {
29         ((*q).tail).next = &lmnt{n, nil}
30         (*q).tail = ((*q).tail).next
31     }
32 }
33
34 func (q *queue) Remove () error {
35     if (*q).Empty() {
36         return errors.New("Attempt to remove from empty queue")
37     } else {
38         (*q).head = ((*q).head).next
39         return nil
40     }
41 }
42
43 func (q queue) Max () int {
44     res := math.MinInt64
45     runner := q.head
46     for runner != nil {
47         if (*runner).n > res { res = (*runner).n }
48         runner = (*runner).next
49     }
50     return res
51 }
52
53 func (q queue) Empty() bool {
54     return q.head == nil
55 }
56
57 func main() {
58     f, err := os.Open("numbers.dat")
59     if err != nil { return }
60     defer f.Close()
61     var k int
62     _, err = fmt.Fscanf(f, "%d\n", &k)
63     if err != nil { return }
64     var x int
65     q := NewQueue()
66     for i := 0; i < k; i++ {
67         _, err := fmt.Fscanf(f, "%d\n", &x)
68         if err != nil { break }
69         q.Add(x)
70     }
71     for {
72         fmt.Println(q.Max())
73         _, err := fmt.Fscanf(f, "%d\n", &x)
74         if err != nil { break }
75         q.Add(x)
76         q.Remove()
77     }
78 }

```