LECTURE 17

THURSDAY NOVEMBER 7

Use of MATHMODELS: Single-Choice Principle

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 1
  imp: ARRAY[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_from_array (imp)
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make_empty ensure model.count = 0 end
  push (g: G) do imp.force(g, imp.count + 1)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.remove_tail(1)
    ensure popped: model ~ (old model.deep_twin).front end
end
```

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 2 (first as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.prepend(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[count - i.item + 1]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.put_front(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.start ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation Strategy 3 (last as top)
  imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
  model: SEQ[G]
    do create Result.make_empty
      across imp as cursor loop Result.append(cursor.item) end
    ensure
      counts: imp.count = Result.count
      contents: across 1 |..| Result.count as i all
                  Result[i.item] ~ imp[i.item]
    end
feature -- Commands
  make do create imp.make ensure model.count = 0 end
  push (g: G) do imp.extend(g)
    ensure pushed: model ~ (old model.deep_twin).appended(g) end
  pop do imp.finish ; imp.remove
    ensure popped: model ~ (old model.deep_twin).front end
end
```

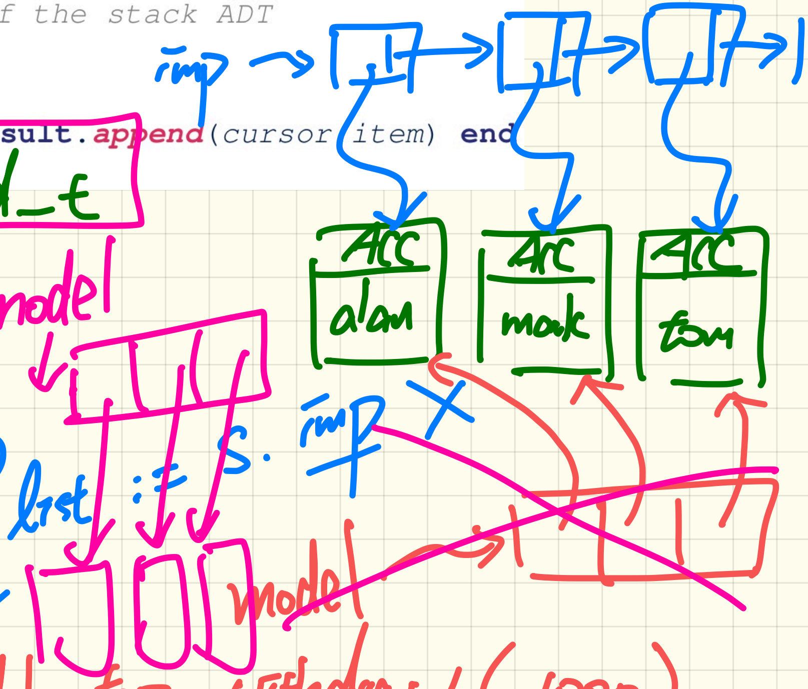
Safe Use of model by Evil Clients

```
class LIFO_STACK[G -> attached ANY] create make
feature {NONE} -- Implementation
 imp: LINKED_LIST[G]
feature -- Abstraction function of the stack ADT
 model: SEQ[G]
  do create Result.make_empty
     across imp as cursor loop Result.append(cursor.item) end
  end
```

Result := Result.d_e

Client:

```
s: STACK[ACCOUNT]
s.push(alan)
s.push(mark)
s.push(tom)
seq := s.model
```



s. model . top . withdraw (1000)

Testing REL in MATHMODELS

$$r.\textbf{overridden}(\underbrace{\{(a,3),(c,4)\}}_{t})$$

$$= \underbrace{\{(a,3),(c,4)\}}_{t} \cup \underbrace{\{(b,2),(b,5),(d,1),(e,2),(f,3)\}}_{r.\textbf{domain\_subtracted}(\underbrace{t.\textbf{domain}}_{\{a,c\}})}$$

$$= \{(a,3),(c,4),(b,2),(b,5),(d,1),(e,2),(f,3)\}$$

```
test_rel: BOOLEAN
  local
    r, t: REL[STRING, INTEGER]
    ds: SET[STRING]
  do
    create r.make_from_tuple_array (
      <<["a", 1], ["b", 2], ["c", 3],
        ["a", 4], ["b", 5], ["c", 6],
        ["d", 1], ["e", 2], ["f", 3]>>)
    create ds.make_from_array (<<"a">>)
    -- r is not changed by the query 'domain_subtracted'
    t := r.domain_subtracted (ds)
    Result :=
      t /~ r and not t.domain.has ("a") and r.domain.has ("a")
    check Result end
    -- r is changed by the command 'domain_subtract'
    r.domain_subtract (ds)
    Result :=
      t ~ r and not t.domain.has ("a") and not r.domain.has ("a")
  end
```

D R

→ imm. query

→ command

Say $r = \{(a,1),(b,2),(c,3),(a,4),(b,5),(c,6),(d,1),(e,2),(f,3)\}$

- $\boxed{\text{r.}\textit{domain}}$: set of first-elements from r
  - r.**domain** $= \{ d \mid (d,r) \in r \}$
  - e.g., r.**domain** $= \{a,b,c,d,e,f\}$
- $\boxed{\text{r.}\textit{range}}$: set of second-elements from r
  - r.**range** $= \{ r \mid (d,r) \in r \}$
  - e.g., r.**range** $= \{1,2,3,4,5,6\}$
- $\boxed{\text{r.}\textit{inverse}}$: a relation like r except elements are in reverse order
  - r.**inverse** $= \{ (r,d) \mid (d,r) \in r \}$
  - e.g., r.**inverse** $= \{(1,a),(2,b),(3,c),(4,a),(5,b),(6,c),(1,d),(2,e),(3,f)\}$
- $\boxed{\text{r.}\textit{domain\_restricted}\text{(ds)}}$: sub-relation of r with domain ds.
  - r.**domain_restricted**(ds) $= \{ (d,r) \mid (d,r) \in r \land d \in ds \}$
  - e.g., r.**domain_restricted**({a, b}) $= \{(\mathbf{a},1),(\mathbf{b},2),(\mathbf{a},4),(\mathbf{b},5)\}$
- $\boxed{\text{r.}\textit{domain\_subtracted}\text{(ds)}}$: sub-relation of r with domain <u>not</u> ds.
  - r.**domain_subtracted**(ds) $= \{ (d,r) \mid (d,r) \in r \land d \notin ds \}$
  - e.g., r.**domain_subtracted**({a, b}) $= \{(\mathbf{c},6),(\mathbf{d},1),(\mathbf{e},2),(\mathbf{f},3)\}$
- $\boxed{\text{r.}\textit{range\_restricted}\text{(rs)}}$: sub-relation of r with range rs.
  - r.**range_restricted**(rs) $= \{ (d,r) \mid (d,r) \in r \land r \in rs \}$
  - e.g., r.**range_restricted**({1, 2}) $= \{(a,\mathbf{1}),(b,\mathbf{2}),(d,\mathbf{1}),(e,\mathbf{2})\}$
- $\boxed{\text{r.}\textit{range\_subtracted}\text{(ds)}}$: sub-relation of r with range <u>not</u> ds.
  - r.**range_subtracted**(rs) $= \{ (d,r) \mid (d,r) \in r \land r \notin rs \}$
  - e.g., r.**range_subtracted**({1, 2}) $= \{(c,\mathbf{3}),(a,\mathbf{4}),(b,\mathbf{5}),(c,\mathbf{6})\}$

MATHMODELS

SEQ

SET

REL

↑

FUN

$(b, 6)$ $(g, 4)$

Say $r = \{(a,1), (\cancel{b,2}), (c,3), (a,4), (\cancel{b,5}), (c,6), (d,1), (e,2), (f,3)\}$

$r.\text{override}(\{(a,3), (c,4)\})$

- $\boxed{\text{r.}\textbf{\textit{domain}}}$: set of first-elements from r
  - r.**domain** $= \{\ d \mid (d, r) \in r\ \}$
  - e.g., r.**domain** $= \{a, b, c, d, e, f\}$
- $\boxed{\text{r.}\textbf{\textit{range}}}$: set of second-elements from r
  - r.**range** $= \{\ r \mid (d, r) \in r\ \}$
  - e.g., r.**range** $= \{1, 2, 3, 4, 5, 6\}$
- $\boxed{\text{r.}\textbf{\textit{inverse}}}$: a relation like r except elements are in reverse order
  - r.**inverse** $= \{\ (r, d) \mid (d, r) \in r\ \}$
  - e.g., r.**inverse** $= \{(1,a), (2,b), (3,c), (4,a), (5,b), (6,c), (1,d), (2,e), (3,f)\}$

$r.\text{override}(\{(g, 4), (b, 6)\})$

1 2 3 4 5 6 7 8 9

Say $r = \{\cancel{(a,1)}, (b,2), (c,3), \cancel{(a,4)}, (b,5), (c,6), (d,1), (e,2), (f,3)\}$

- $\boxed{\text{r.}\textit{domain}}$: set of first-elements from r
  - r.**domain** $= \{\ d \mid (d,r) \in r\ \}$
  - e.g., r.**domain** $= \{a, b, c, d, e, f\}$
- $\boxed{\text{r.}\textit{range}}$: set of second-elements from r
  - r.**range** $= \{\ r \mid (d,r) \in r\ \}$
  - e.g., r.**range** $= \{1, 2, 3, 4, 5, 6\}$
- $\boxed{\text{r.}\textit{inverse}}$: a relation like r except elements are in reverse order
  - r.**inverse** $= \{\ (r,d) \mid (d,r) \in r\ \}$
  - e.g., r.**inverse** $= \{(1,a), (2,b), (3,c), (4,a), (5,b), (6,c), (1,d), (2,e), (3,f)\}$

r.domain_restrict({a}) = {(a,1), (a,4)}

r.domain_subtract({a})

1 2 3 4 5 6 7 8 9

Say $r = \{(a,1), \cancel{(b,2)}, (c,3), (a,4), (b,5), (c,6), (d,1), \cancel{(e,2)}, (f,3)\}$

r.range_subtract({2})

- $\boxed{\text{r.}\textit{domain}}$: set of first-elements from r
  - r.**domain** $= \{\ d \mid (d,r) \in r\ \}$
  - e.g., r.**domain** $= \{a, b, c, d, e, f\}$
- $\boxed{\text{r.}\textit{range}}$: set of second-elements from r
  - r.**range** $= \{\ r \mid (d,r) \in r\ \}$
  - e.g., r.**range** $= \{1, 2, 3, 4, 5, 6\}$
- $\boxed{\text{r.}\textit{inverse}}$: a relation like r except elements are in reverse order
  - r.**inverse** $= \{\ (r,d) \mid (d,r) \in r\ \}$
  - e.g., r.**inverse** $= \{(1,a), (2,b), (3,c), (4,a), (5,b), (6,c), (1,d), (2,e), (3,f)\}$

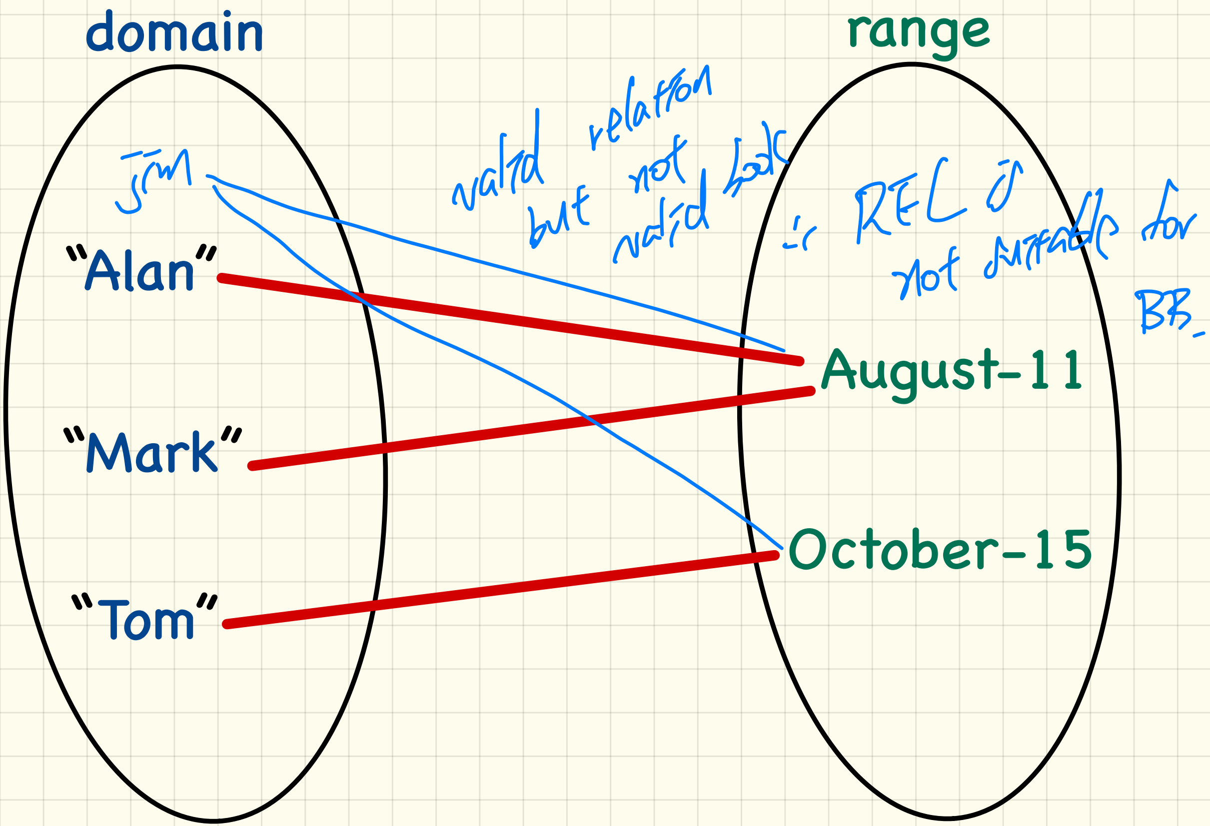
r. override (..)

Command

r. overridden_by (..)

↳ immutable query

Model of an Example Birthday Book



Birthday Book: Design

BIRTHDAY_BOOK

model: FUN[NAME, BIRTHDAY]
-- abstraction function

model: ~~BIRTHDAY~~

count: INTEGER
-- number of entries

put(n: NAME; d: BIRTHDAY)
ensure
model_operation: model ~ (**old** model.deep_twin).overriden_by ([n,d])
-- infix symbol for override operator: @<+

fn. query

remind(d: BIRTHDAY): ARRAY[NAME]
NAME
ensure
nothing_changed: model ~ (**old** model.deep_twin)
same_counts: **Result**.count = (model.range_restricted_by(d)).count
same_contents: ∀ name ∈ (model.range_restricted_by(d)).domain: name ∈ **Result**
-- infix symbol for range restriction: model @> (d)

invariant:
consistent_book_and_model_counts: count = model.count

BIR.DAY

model: FUN[NAME, ..]

FUN[NAME, ..]

model: . .

BIRTHDAY

day: INTEGER
month: INTEGER

invariant
1 ≤ month ≤ 12
1 ≤ day ≤ 31

ARRAY[~~NAME~~]

remind: ARRAY[~~NAME~~]

NAME

item: STRING

invariant
item[1] ∈ A..Z

Imp.

ns → | alan | mark | tom | "jim" |

birthdays → | jan5 | aug6 | jan5 | dec4 |

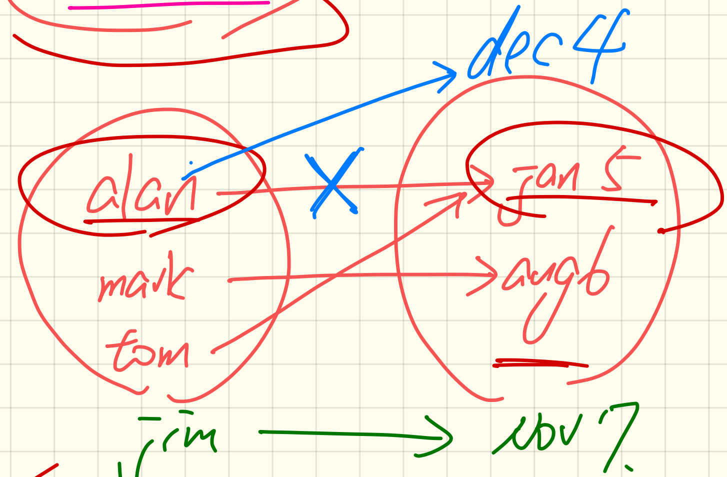
put(alan, dec4)

put(jim, nov7)

→ design decision

: if the name exists, overwrite the birthday.

Model



✓ unioned

✓ overridden

Imp

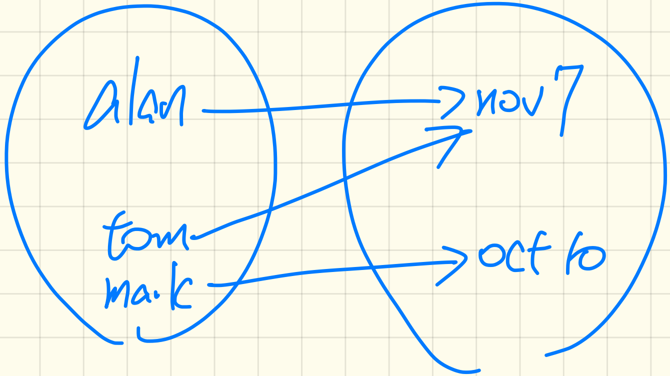
ns →

alan	mark	tom

bds →

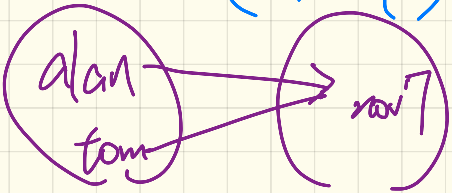
nov 7	oct 6	nov 7

Model



a : ARRAY [STRING]

model.range_restricted_by (nov 7)



book.remind (nov 7).

↳ << alan, tom >>

Birthday Book: Implementation

Imp: HASH_TABLE[BIR, NAME]

BIRTHDAY_BOOK

```
model: FUN[NAME, BIRTHDAY]
    -- abstraction function
  do
    -- promote hashtable to function
  ensure
    same_counts: Result.count = implementation.count
    same_contents: ∀ [name, date] ∈ Result: [name, date] ∈ implementation
  end

put(n: NAME; d: BIRTHDAY)
  do
    -- implement using hashtable
  ensure
    model_operation: model ~ (old model.deep_twin) @<+ [n,d]
  end

remind(d: BIRTHDAY): ARRAY[NAME]
  do
    -- implement using hashtable
  ensure
    nothing_changed: model ~ (old model.deep_twin)
    same_counts: Result.count = (model @> d).count
    same_contents: ∀ name ∈ (model @> d).domain: name ∈ Result
  end

count: INTEGER -- number of names

feature {NONE}
  implementation: HASH_TABLE[BIRTHDAY, NAME]

invariant:
  consistent_book_and_model_counts: count = model.count
  consistent_book_and_imp_counts: count = implementation.count
```

exercise

model: FUN[NAME, ..]

BIRTHDAY

```
day: INTEGER
month: INTEGER

invariant
  1 ≤ month ≤ 12
  1 ≤ day ≤ 31
```

* HASHABLE

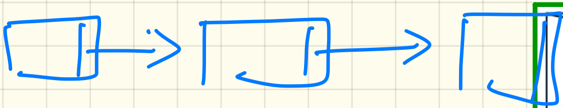
remind: ARRAY[NAME]

NAME

```
item: STRING

invariant
  item[1] ∈ A..Z
```

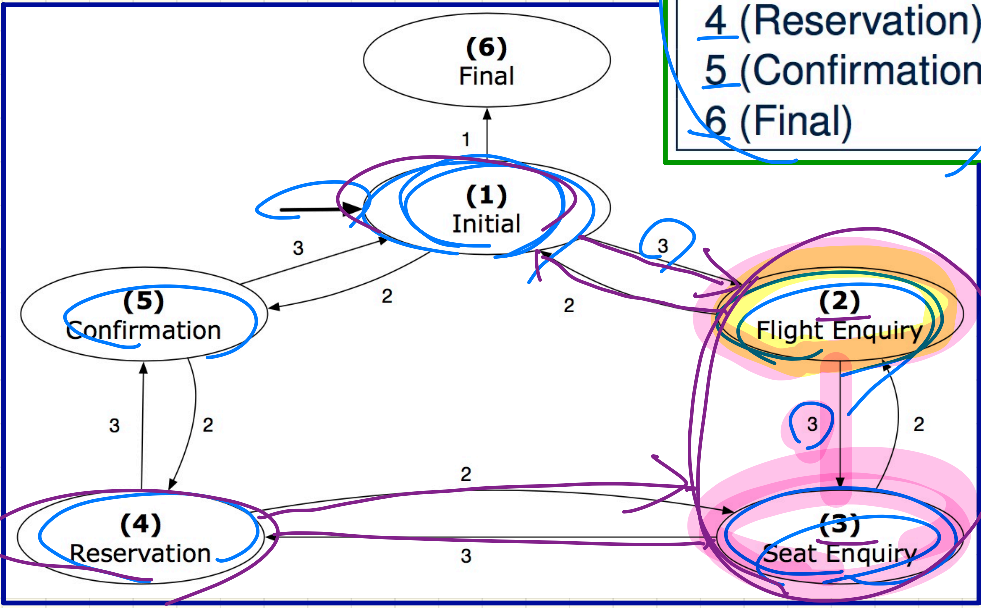
Finite State Machine (FSM)



State Transition Table

Src State \ Choice	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

State Transition Diagram



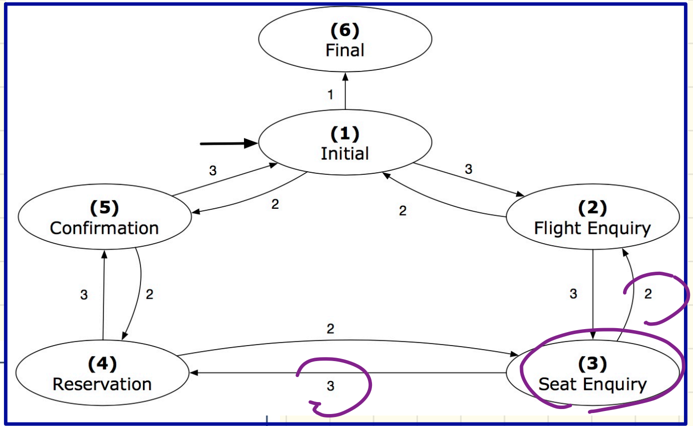
Design of a Reservation System: First Attempt

from
⋮
until
B
loop
⋮
end

as soon as B becomes true, the exit from loop.

while (B) {
⋮
} not

as long as B is true, stay.




```
1_Initial_panel:
  -- Actions for Label 1.
2_Flight_Enquiry_panel:
  -- Actions for Label 2.
3_Seat_Enquiry_panel:
  -- Actions for Label 3.
4_Reservation_panel:
  -- Actions for Label 4.
5_Confirmation_panel:
  -- Actions for Label 5.
6_Final_panel:
  -- Actions for Label 6.
```

```
3_Seat_Enquiry_panel:
from
  Display Seat Enquiry Panel
until
  not (wrong answer or wrong choice)
do
  Read user's answer for current panel
  Read user's choice C for next step
  if wrong answer or wrong choice then
    Output error messages
  end
end
Process user's answer
case C in
  2: goto 2_Flight_Enquiry_panel
  3: goto 4_Reservation_panel
end
```

not wrong ans
and
not wrong choice

while (wrong ans
or
wrong choice)
{ .. }

Design of a Reservation System: Second Attempt (1)

```
transition (src: INTEGER; choice: INTEGER): INTEGER
  -- Return state by taking transition 'choice' from 'src' state.
 require valid_source_state: 1 ≤ src ≤ 6
         valid_choice: 1 ≤ choice ≤ 3
 ensure valid_target_state: 1 ≤ Result ≤ 6
```

Examples: src choice

transition(3, 2) → 2

transition(3, 3) → 4

State Transition Table

Src State \ Choice	1	2	3
1 (Initial)	6	5	2
2 (Flight Enquiry)	–	1	3
3 (Seat Enquiry)	–	2	4
4 (Reservation)	–	3	5
5 (Confirmation)	–	4	1
6 (Final)	–	–	–

2D Array Implementation

state \ choice	1	2	3
1	6	5	2
2		1	3
3		2	4
4		3	5
5		4	1
6			

Design of a Reservation System: Second Attempt (2)

A Top-Down & Hierarchical Design