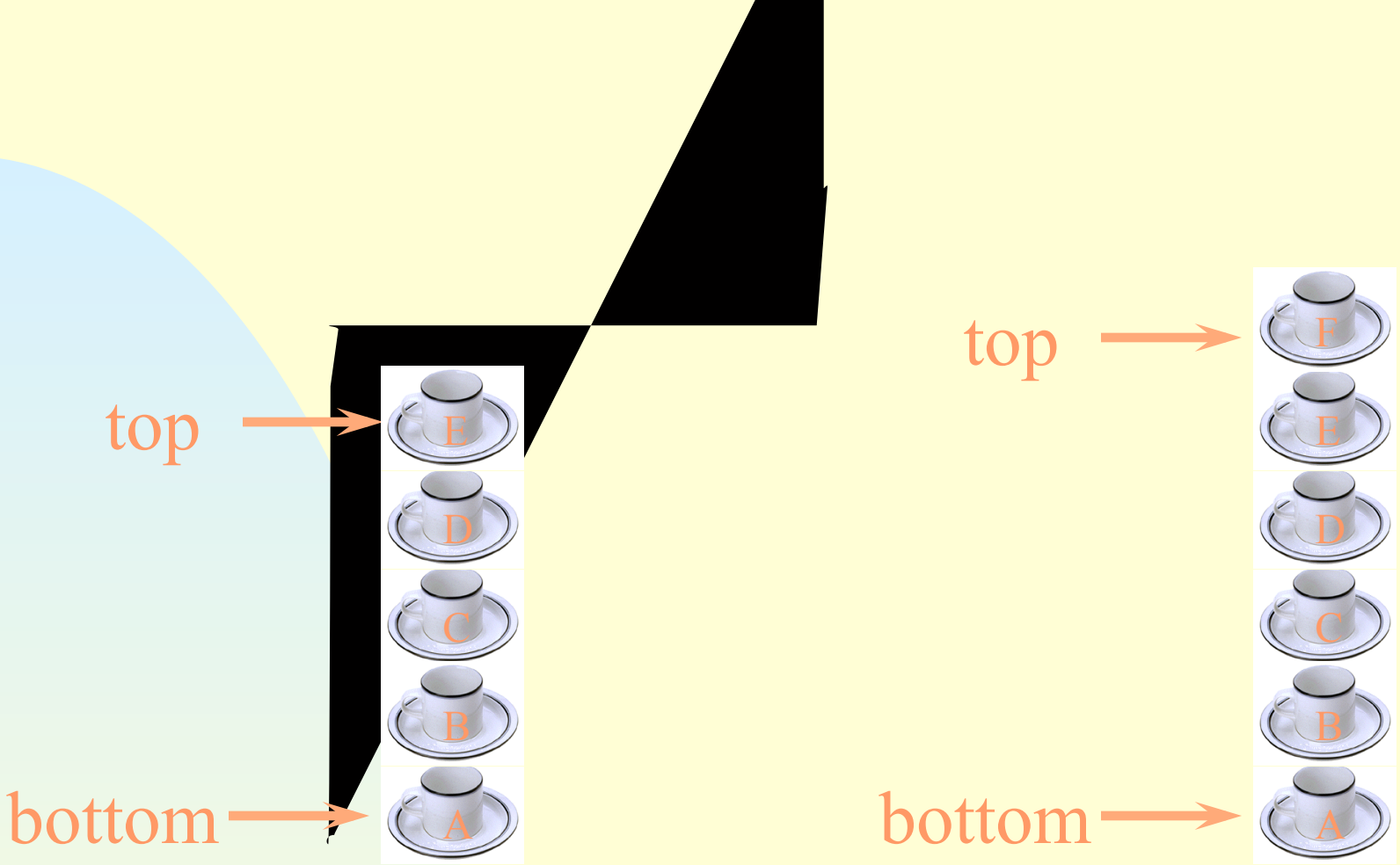
Stacks and Queues

The Stack Abstract Data t pe

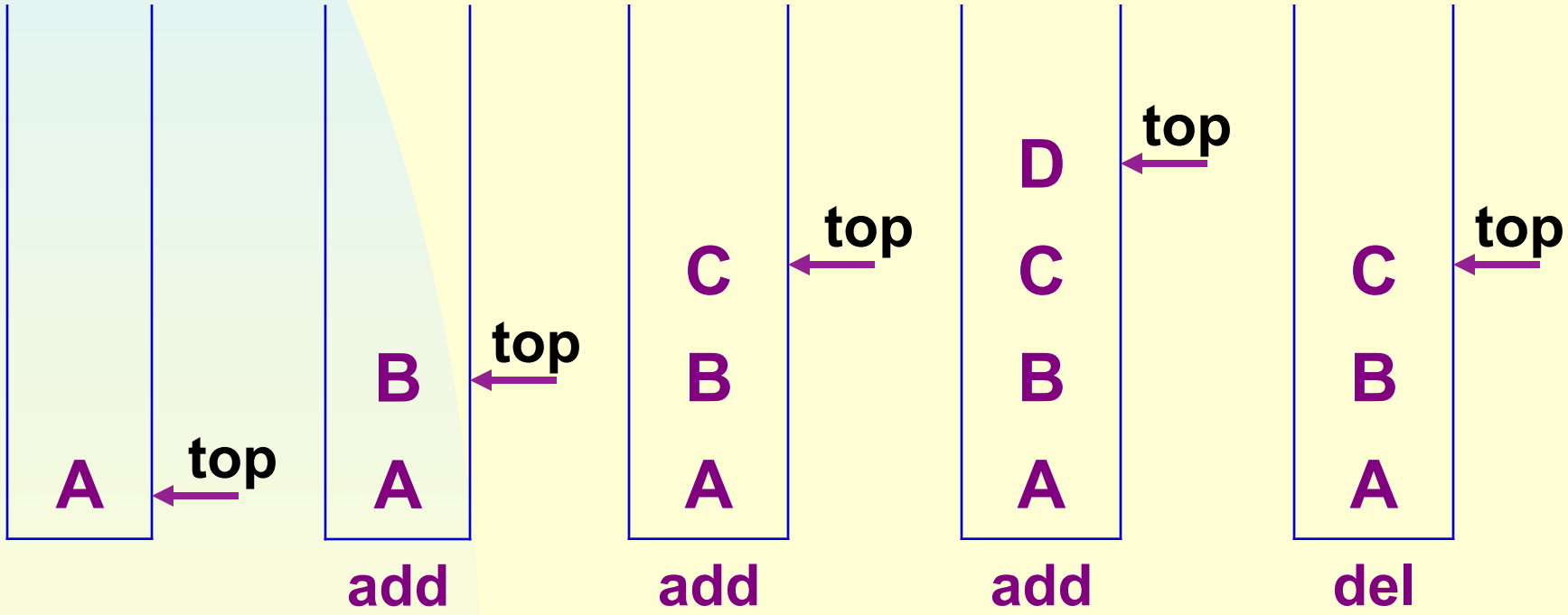


- .

Remo e a cup from ne stack.

A stack is a LIFO list.

:


A
top
B
A
top
add
C
B
A
top
add
D
C
B
A
top
add
C
B
A
top
del

ADT 3.1 Stack

```
                <        T>
     Stack
// A finite ordered list    ith   ero or more elements.
         :
 Stack (      stackCapacit   = 10);
//Creates an empt   stack    ith initial capacit   of stackCapacit

         IsEmpt  ()          ;
 //If number of elements in the stack is 0,         else

 T& Top()          ;
 // Return the top element of stack

         Push(          T& item);
 // Insert item into the top of the stack
         Pop();
 // Delete the top element of the stack.
;
```

,

1.

:

:

T* stack;
top;
capacit ;

```
<        T>
Stack<T>::Stack(     stackCapacit  ): capacit  (stackCapacit  )

   (capacit   < 1)            Stack capacit   must be > 0  ;
  stack =        T[capacit  ];
  top = -1;


<        T>
    Stack<T>::IsEmpt  ()

  (top == -1);
```

```
        <        T>
    T& Stack<T>::Top()

  (IsEmpt  )              Stack is Empt    ;
         stack[top];


         <        T>
   Stack<T>::Push(          T&    )

  (top == capacit   - 1)

   ChangeSi  e1D(stack, capacit  , 2*capacit  );
   capacit   *= 2;

stack[++top] =    ;
```

1

1-

:

```
<        T>
ChangeSi  e1D(T* a,              oldSi  e,              ne   Si  e)

  (ne   Si  e < 0)          Ne    length must be >= 0  ;
T* temp =       T[ne   Si  e];
    number = min(oldSi  e, ne   Si  e);
cop  (a, a + number, temp);
        [] a;
a = temp;
```

```
<        T>
Stack<T>::Pop()
// Delete top element of stack.
   (IsEmpt  ())            Stack is empt  , cannot delete.  ;
 stack[top--].  T();  // destructor for T
```

: 138-1, 2

Bus Stop

front rear rear rear rear

Bus Stop

front

rear

Bus Stop

front

rear

Bus Stop

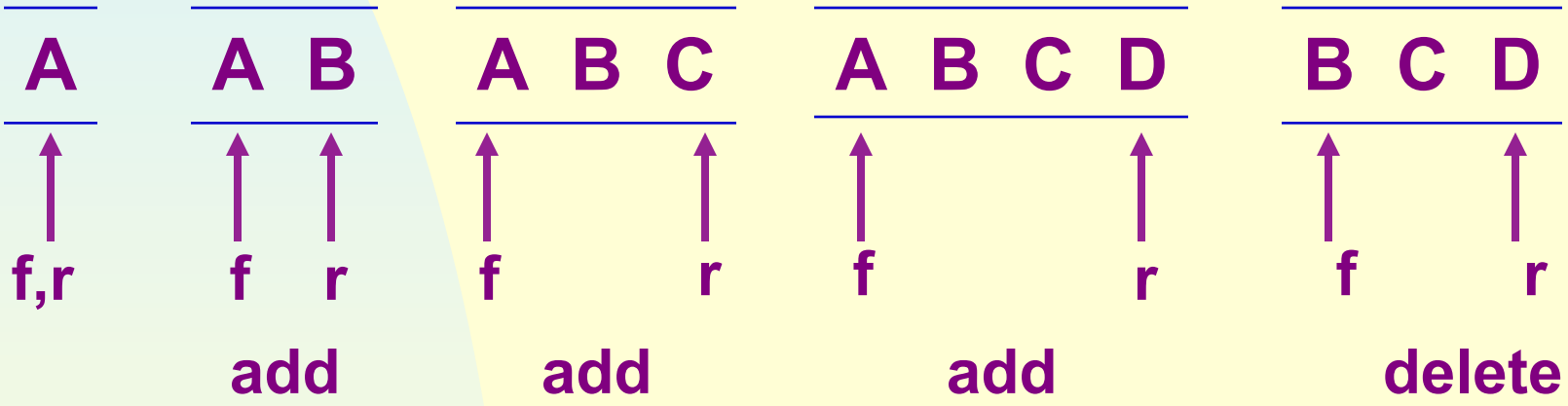
front

rear

rear

3.3 The Q e e Abstract Data T pe

3.3 The Q e e Abstract Data T pe



f = q e e front r = q e e rear

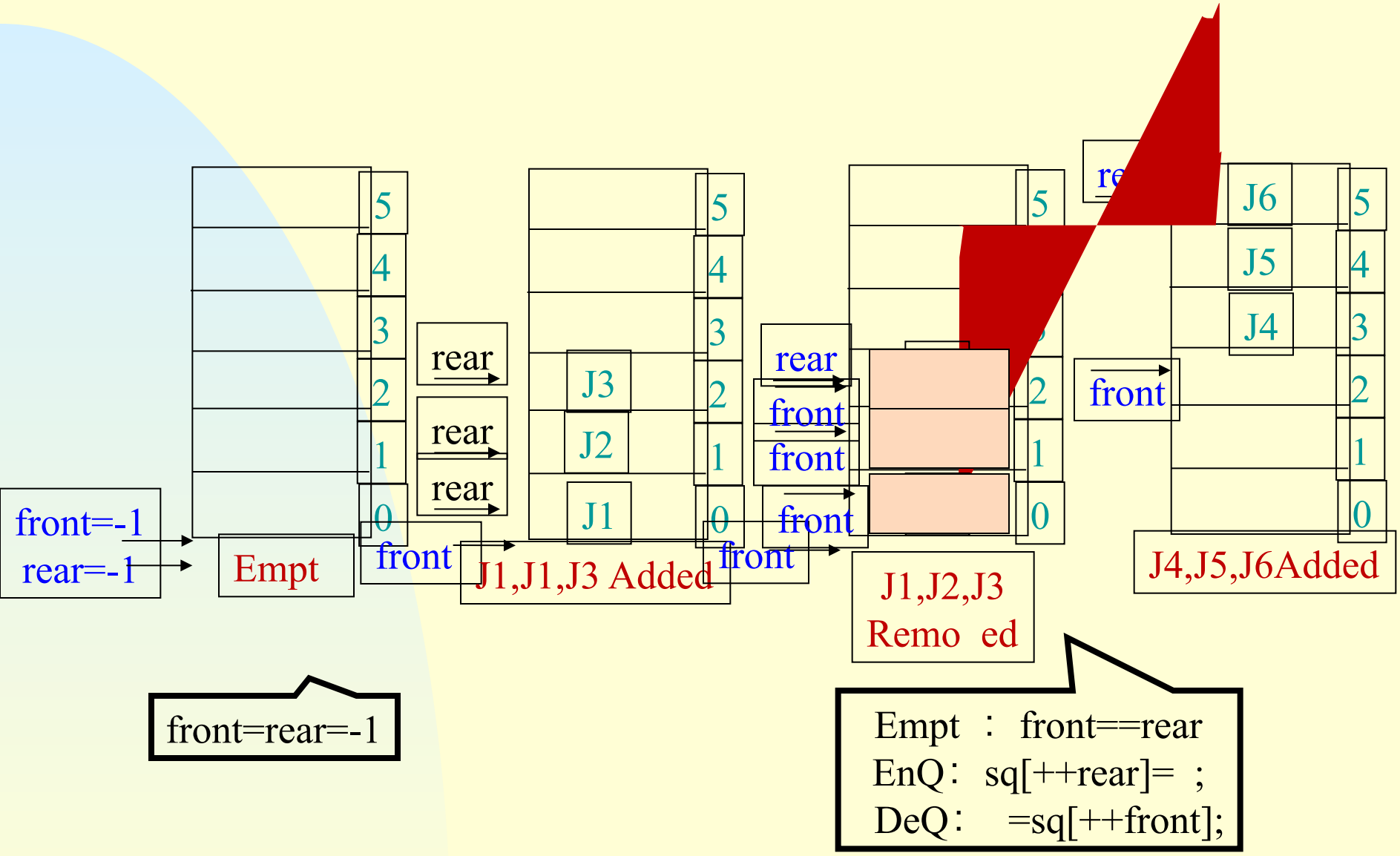
```
<        T>
Queue
// A finite ordered list    ith   ero or more elements.
:
Queue (      queueCapacit   = 10);
// Creates an empt   queue    ith initial capacit   of
// queueCapacit

IsEmpt
```

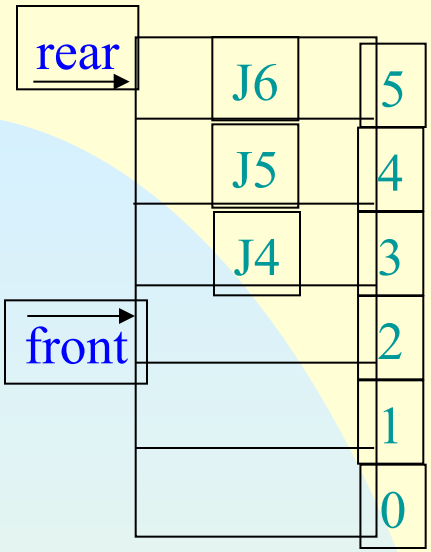
,

:

:

```
T* queue;
   front,
   rear,
   capacit  ;
```


front=-1
rear=-1
Empt
5
4
3
2
1
0
rear
rear
rear
front
J1,J1,J3 Added
J3
J2
J1
front
front
front
front
J1,J2,J3
Remo ed
re
front
J6
J5
J4
J4,J5,J6Added
front=rear=-1
Empt : front==rear
EnQ: sq[++rear]= ;
DeQ: =sq[++front];


rear
J6
J5
J4
front
5
4
3
2
1
0

■ :

◆ !

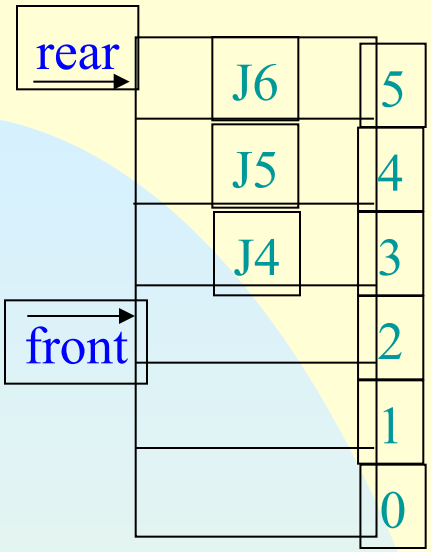
◆ ! →

✦

■

◆

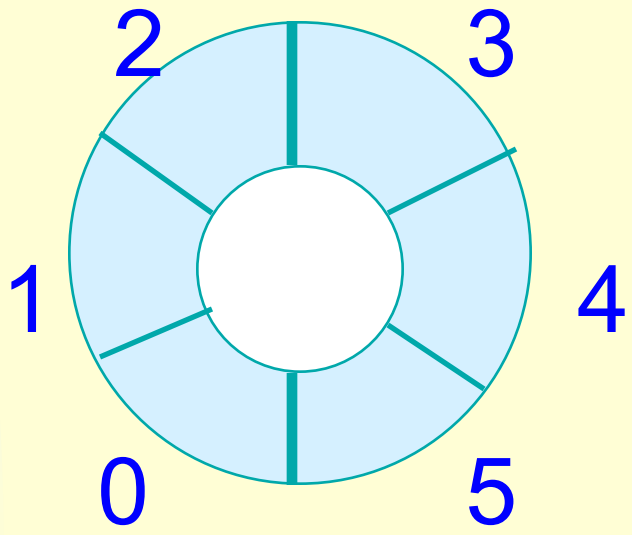
◆ ?



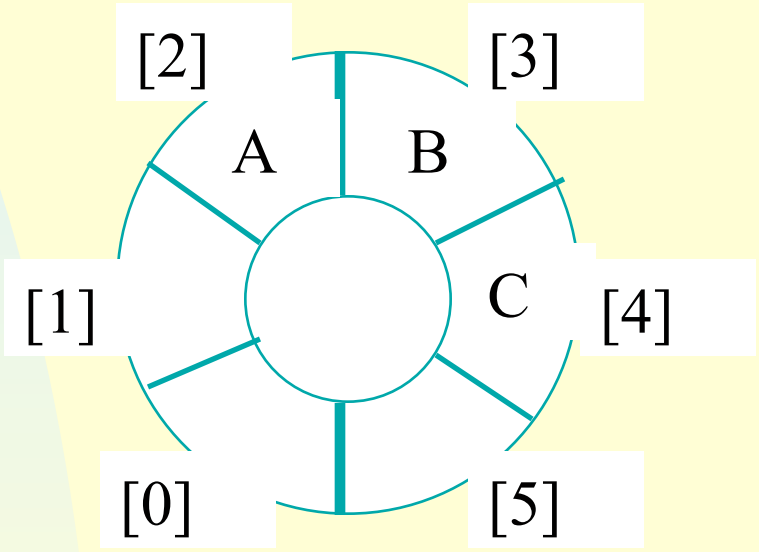
-
-
  - 6 → 2
  - 6 → 1
  - 6 → 0
- 6
- 6 % 6 0

1

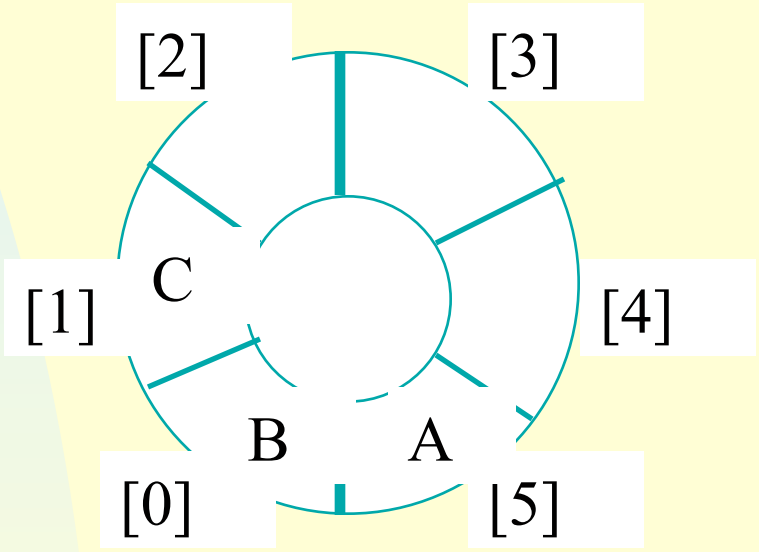
queue[]



Possible configuration ith 3 elements.



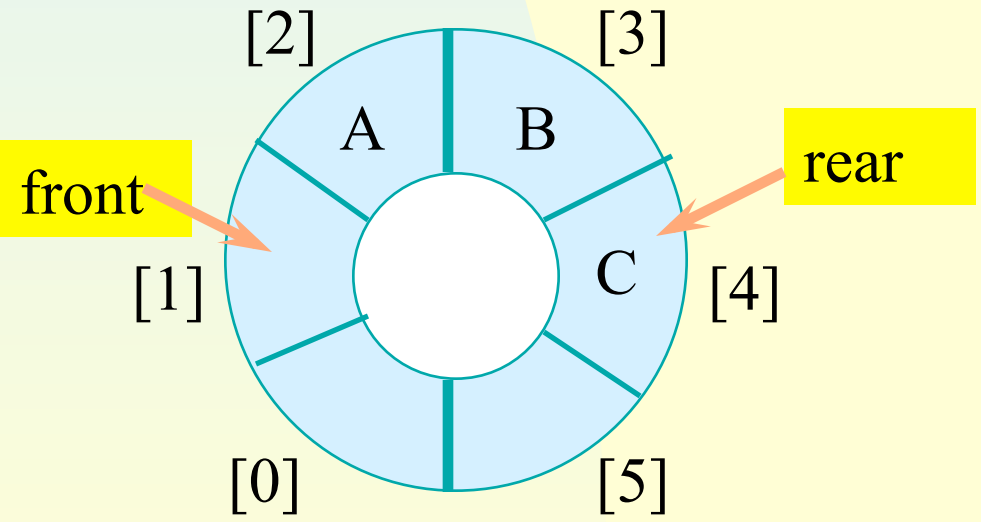
Another possible configuration ith 3 elements.

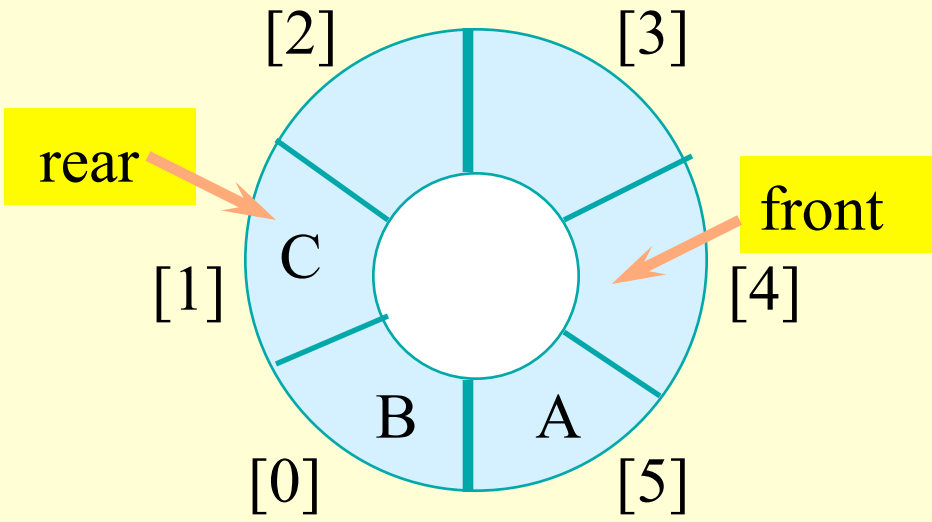


Use integer ariables front and rear.

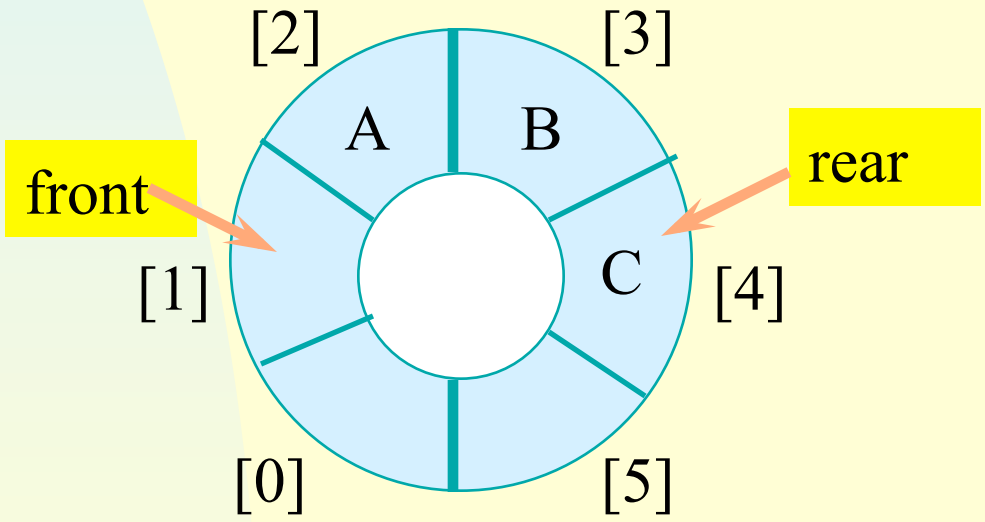
front is one position counterclock ise from first element

rear gi es position of last element



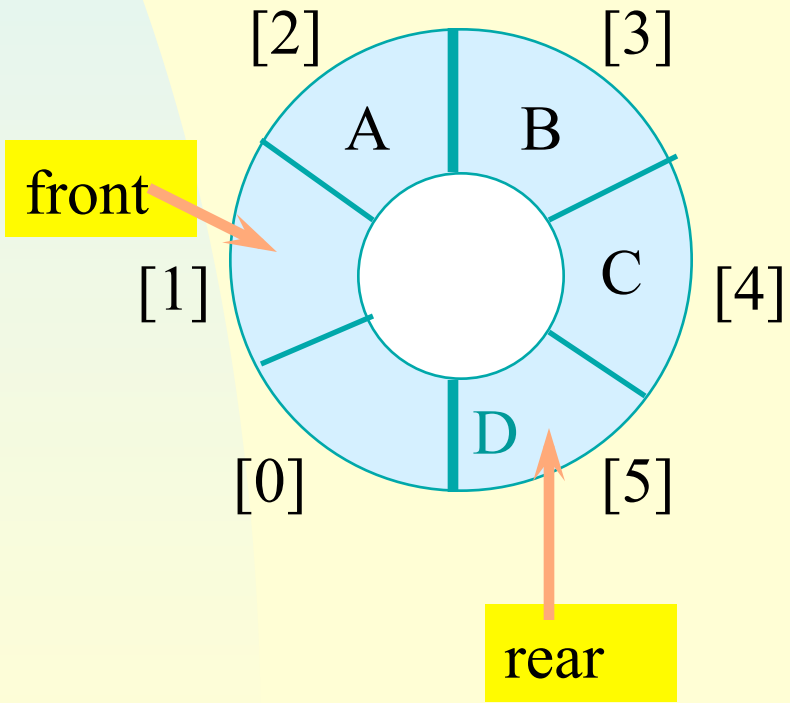


Mo e rear one clock ise.

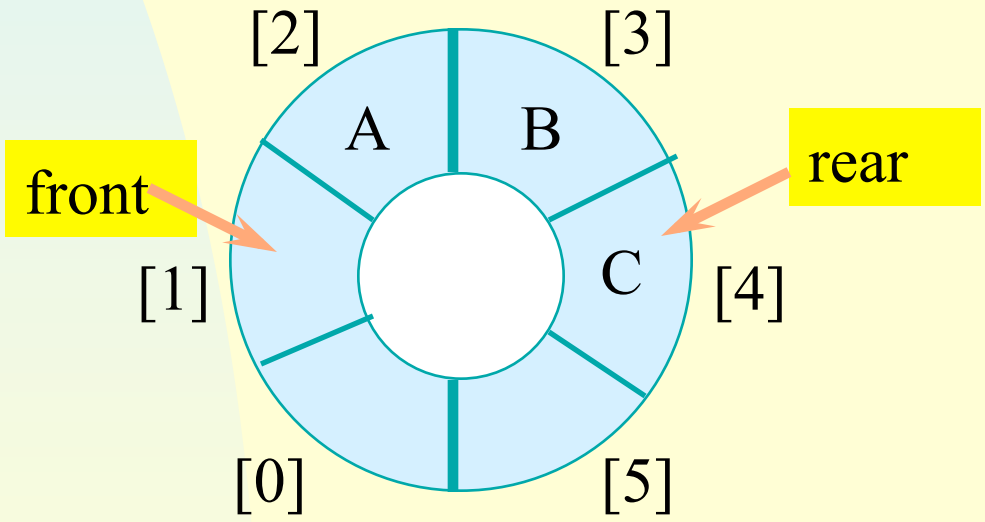


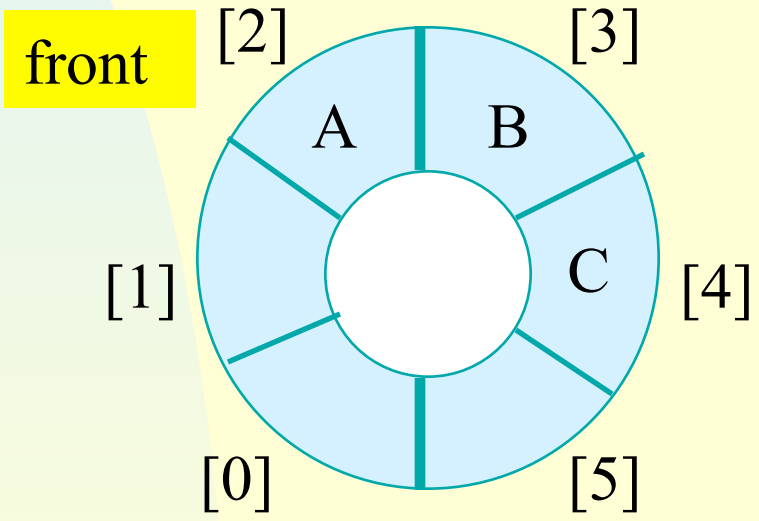
Mo e rear one clock ise.

Then put into queue[rear].

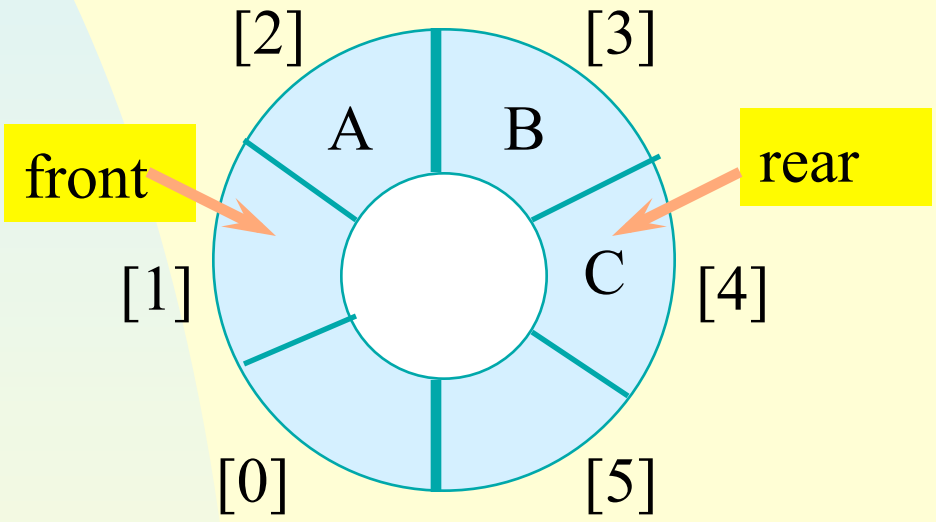


Mo e front one clock ise.

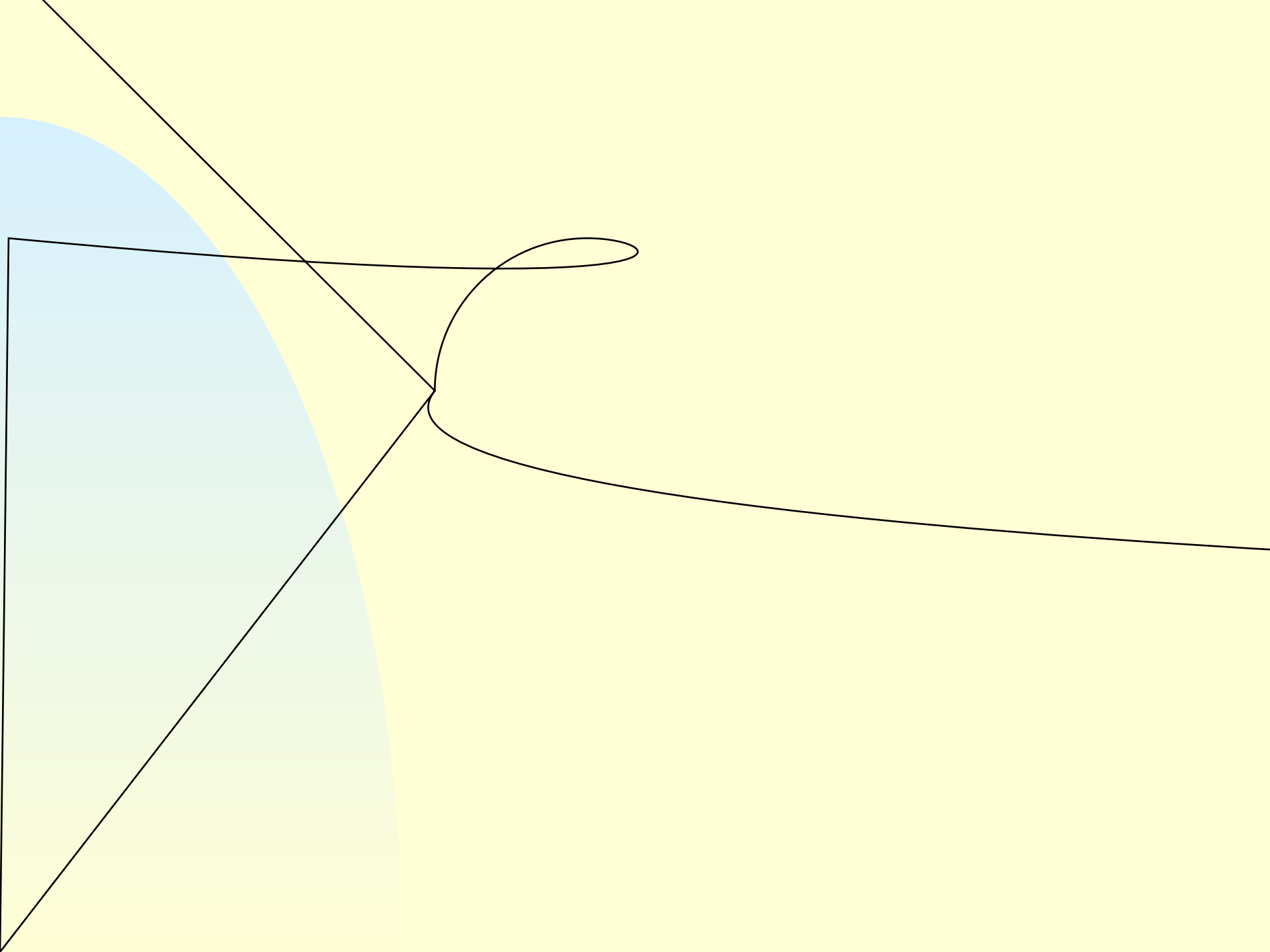


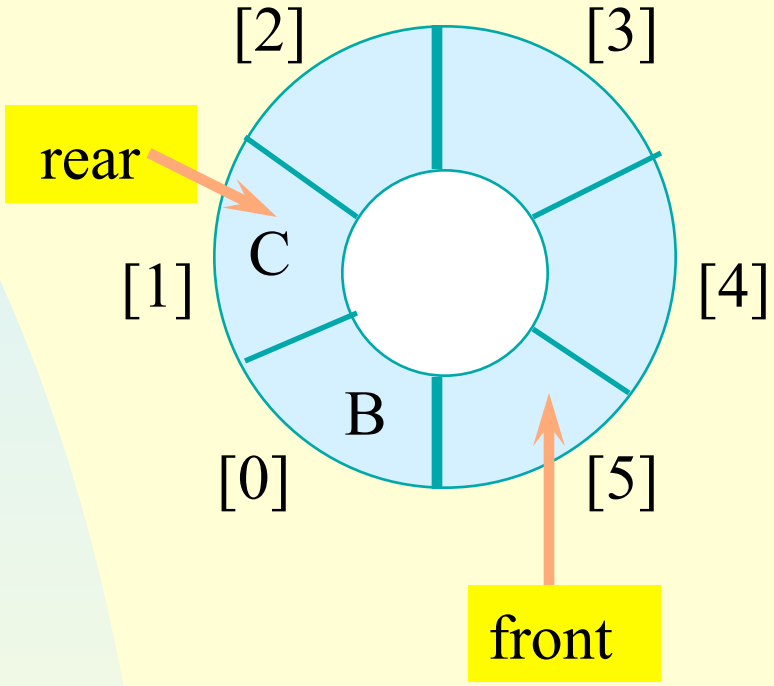

front
[2]
[3]
A
B
C
[1]
[4]
[0]
[5]

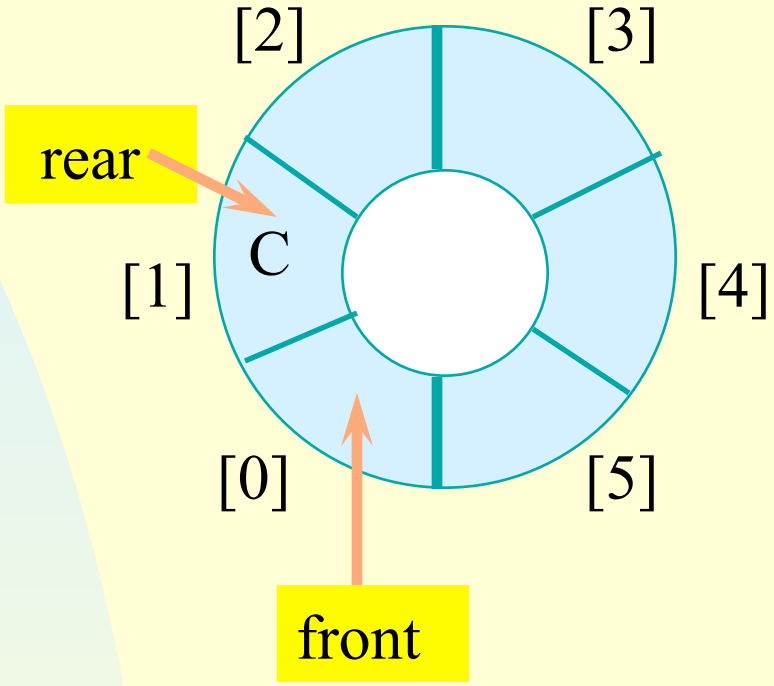
```
rear++;
if (rear = = queue.length) rear = 0;
```

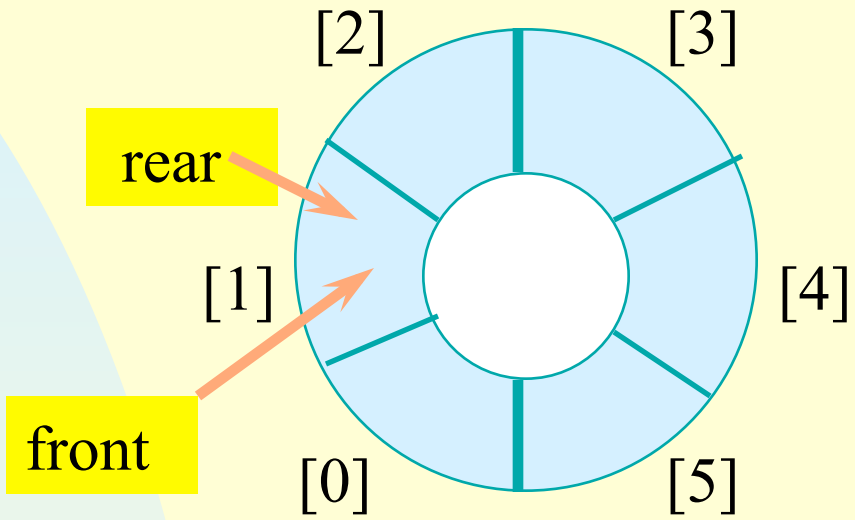



```
rear = (rear + 1) % queue.length;
```

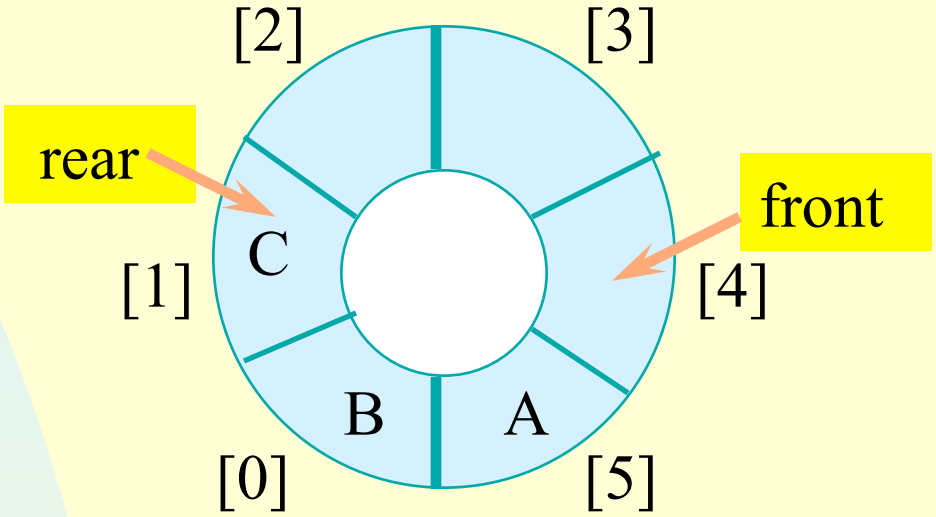


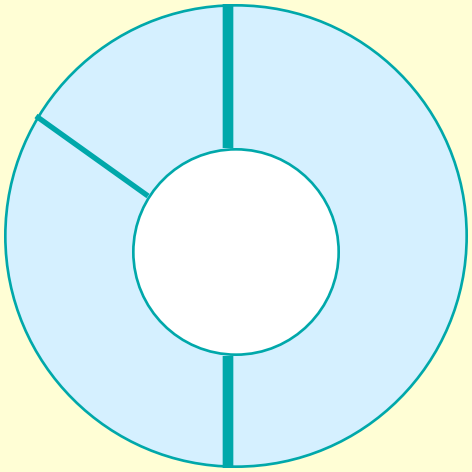

[2]
[3]
rear
C
[1]
[4]
B
[0]
[5]
front

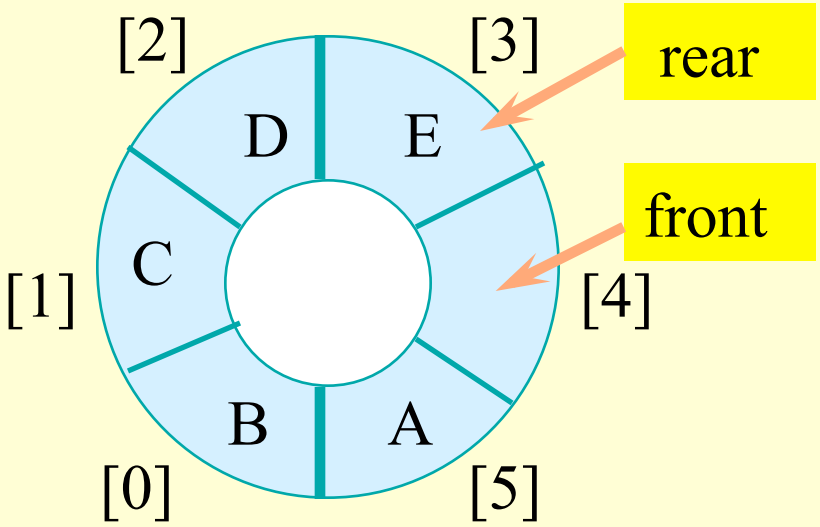

[2]
[3]
rear
C
[1]
[4]
[0]
[5]
front

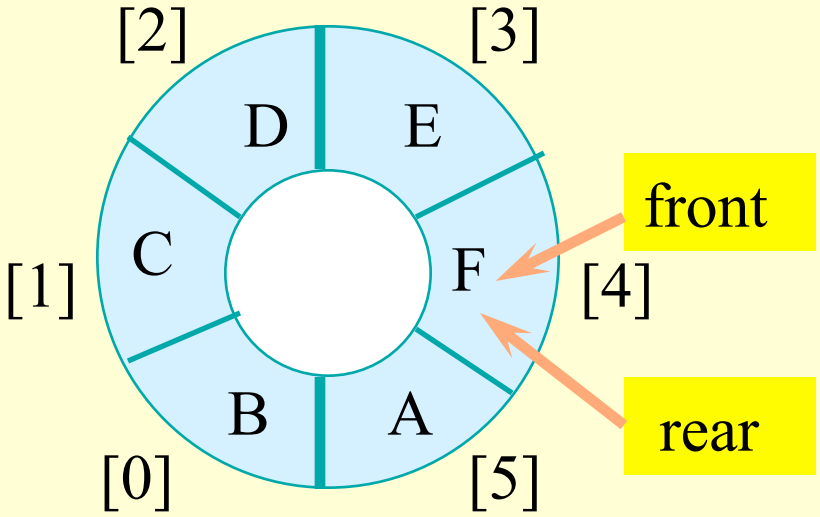


-
-
- 0.


[2]
[3]
rear
front
C
[1]
[4]
B
A
[0]
[5]




[2]
[3]
rear
D
E
front
C
[1]
[4]
B
A
[0]
[5]



When a series of adds causes the queue to become full, front = rear.

So e cannot distinguish bet een a full queue and an empt queue!

( ) && !

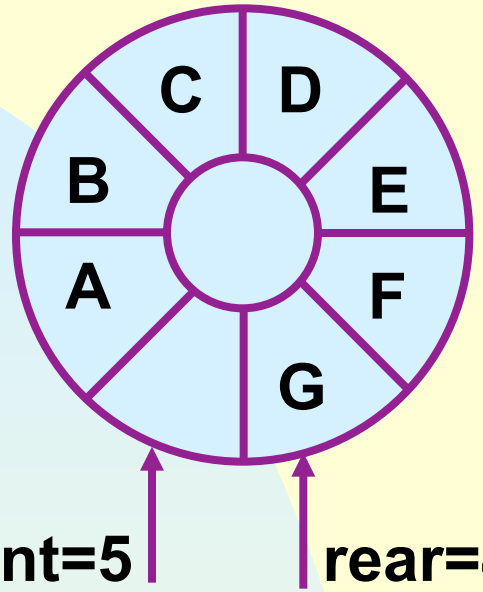
( ) &&

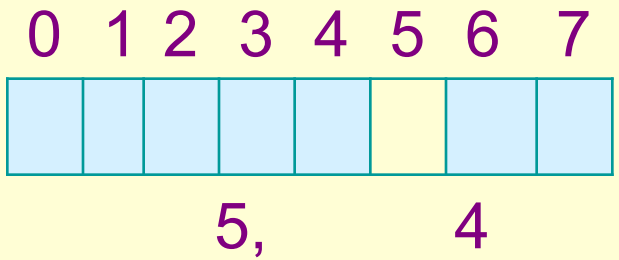
```
                <          T>
Queue<T  pe>::Queue(      queueCapacit  ):
                                        capacit  (queueCapacit  )

    (capacit   < 1)             Queue capacit   must > 0  ;
   queue =          T[capacit  ];
   front = rear = 0;
```

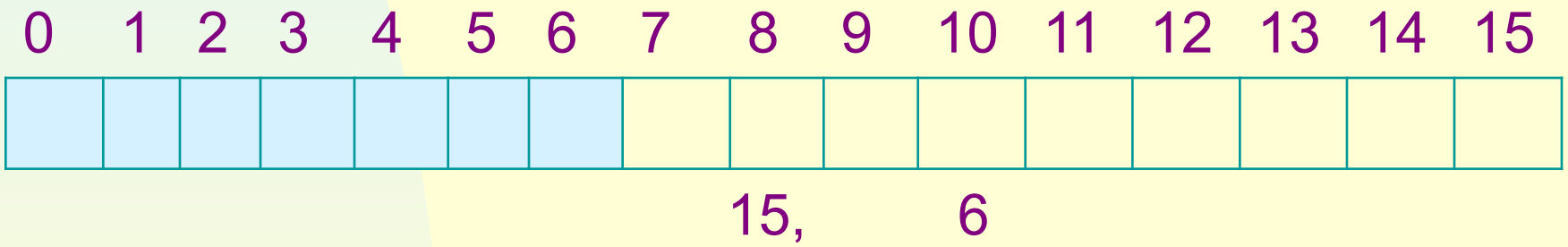
```
<        T>
   Queue<T>::IsEmpt  ()
front==rear  ;

<        T>
T& Queue<T>::Front()

(IsEmpt  ())          Queue is empt  . No front element  ;
      queue[(front+1)%capacit  ];


<        T>
T& Queue<T>::Rear()

(IsEmpt  ())          Queue is empt  . No rear element  ;
      queue[rear];
```

```
<        T>
Queue<T>::Push(          T&   )
// add    at rear of queue
    ((rear+1)%capacit   == front)
    // queue full, _____________
    // code to double queue capacit   comes here

 rear = (rear+1)%capacit  ;
 queue[rear] =   ;
```

:


C
D
B
E
A
F
G
front=5
rear=4


0 1 2 3 4 5 6 7
5,
4


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
15,
6

:

(1) .

(2)

0.

(3)

- -1.

:

```
// allocate an arra      ith t   ice the capacit
T*        Queue =         T[2*capacit  ];

// cop   from queue to ne   Queue
     start = (front+1)%capacit  ;
   (start < 2)
     // no    rap around
     cop  (queue+start, queue+start+capacit  -1, ne   Queue);

   // queue    raps around
     cop  (queue+start, queue+capacit  , ne   Queue);
     cop  (queue, queue+rear+1, ne   Queue+capacit  -start);


// s   itch to ne   Queue
front = 2*capacit  -1;  rear =  capacit  -2; capacit   *= 2;
          [] queue;
queue = ne   Queue;
```

```
<        T>
Queue<T>::Pop()
// Delete front elemnet from queue
   (IsEmpt  ())            Queue is empt  . Cannot delete  ;
front = (front+1)%capacit  ;
queue[front].  T;
```

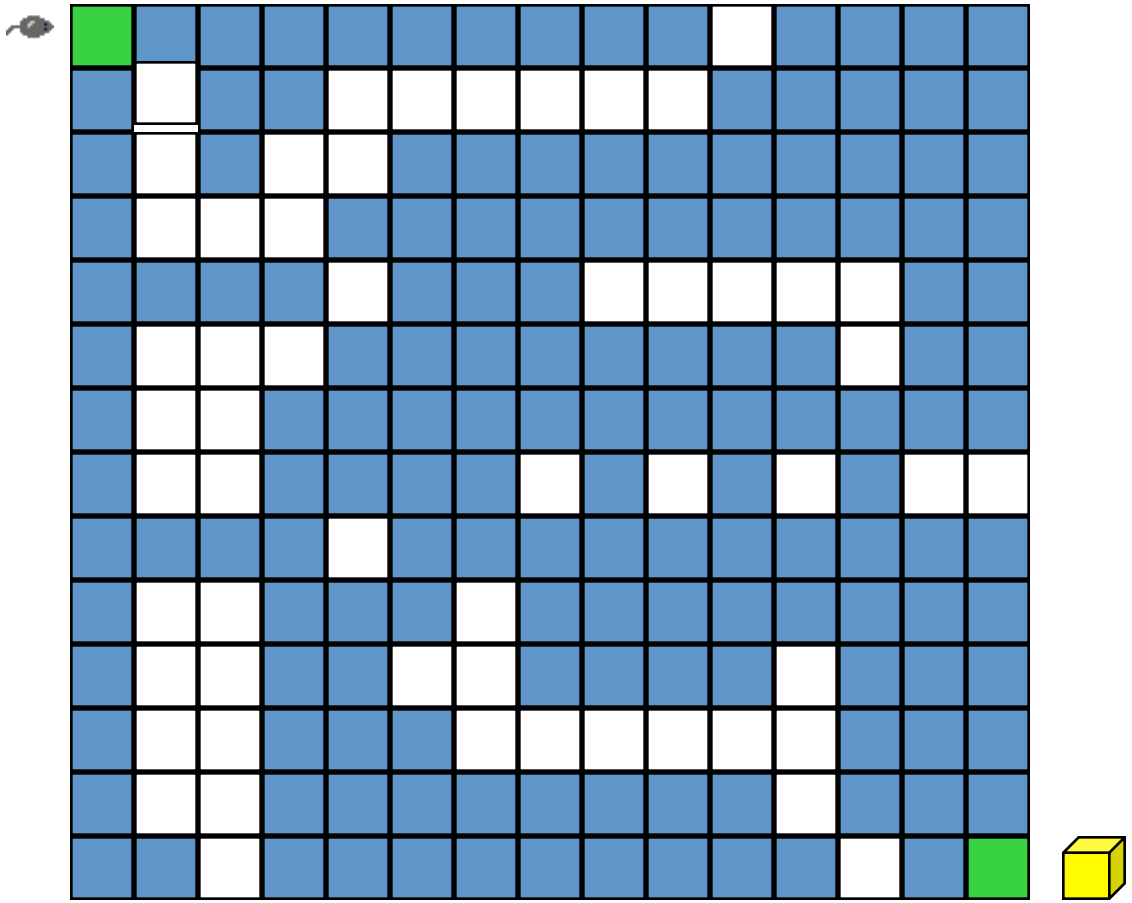
, -

( )

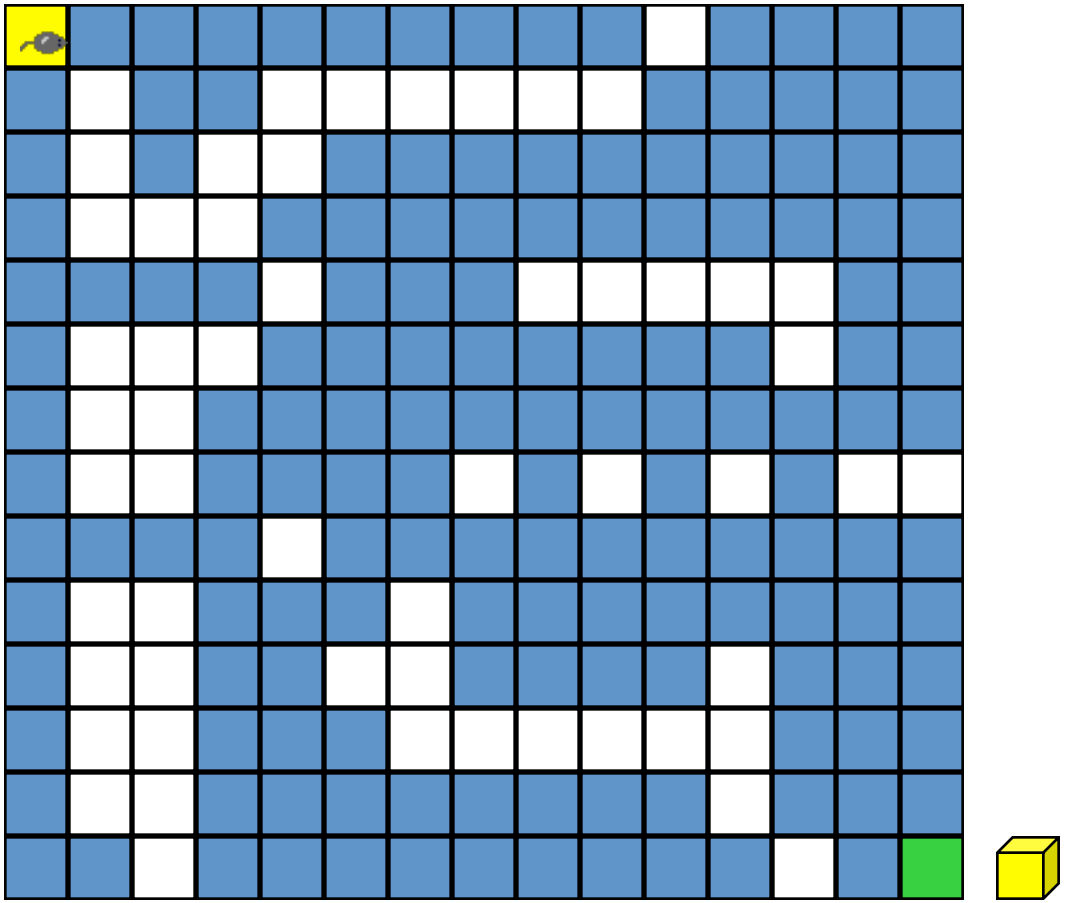
(1).

E ercises: P147-1, 3.

! "#$%&$ ' $ (") *$



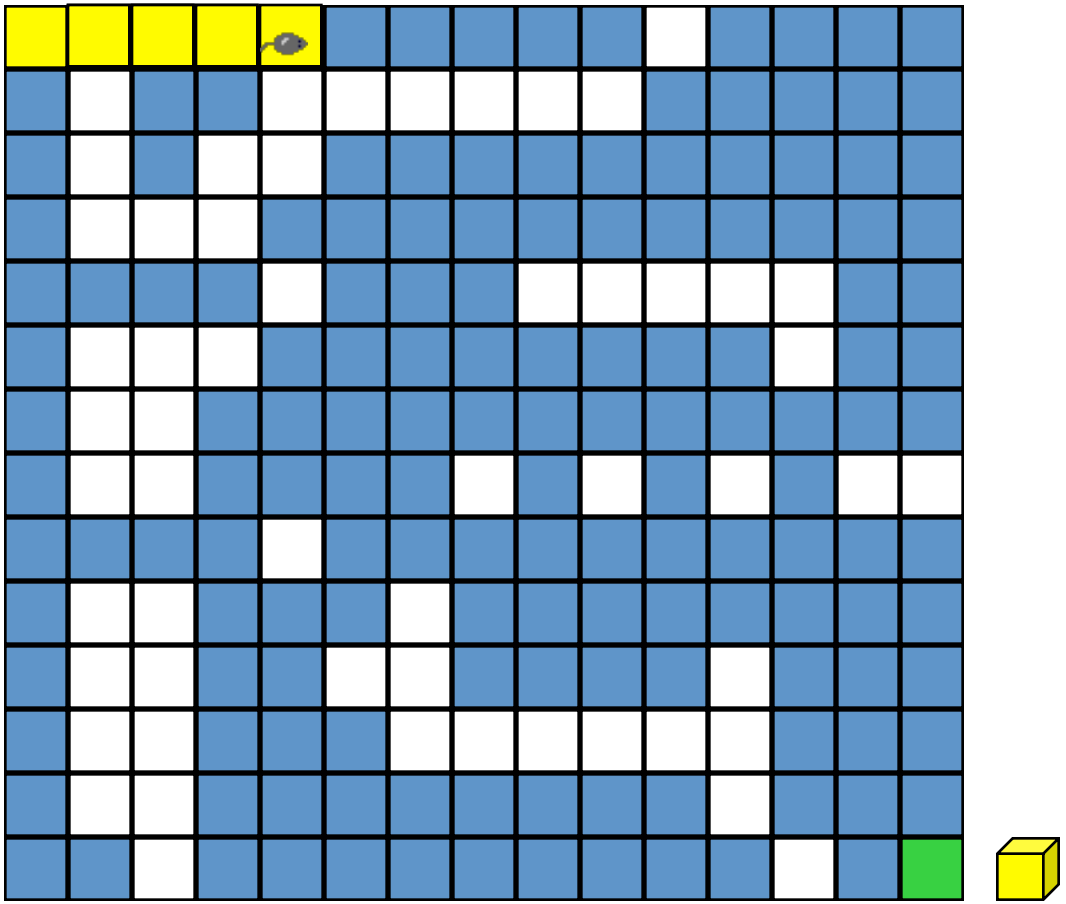
! "#$%&$ ' $ (") *$



(+, *$+- . *-$/01$-/23#$. +5 & $6*7 89

: 6+ ; <$9+0/=+&0$#+$" , +/ . $-* , /0/#>$

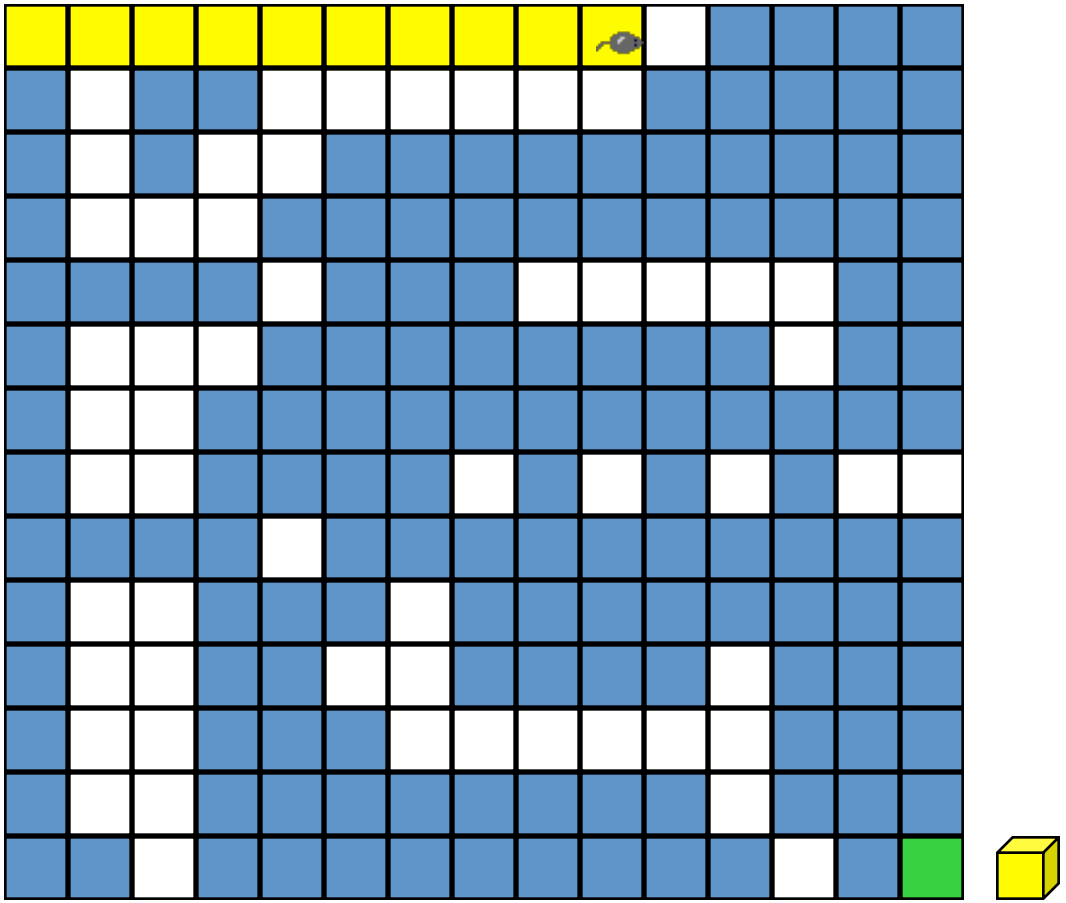
! "#$%&$ ' $ (") *$



(+, *$+- . *-$/01$-/23#$. +5&$6*7 89

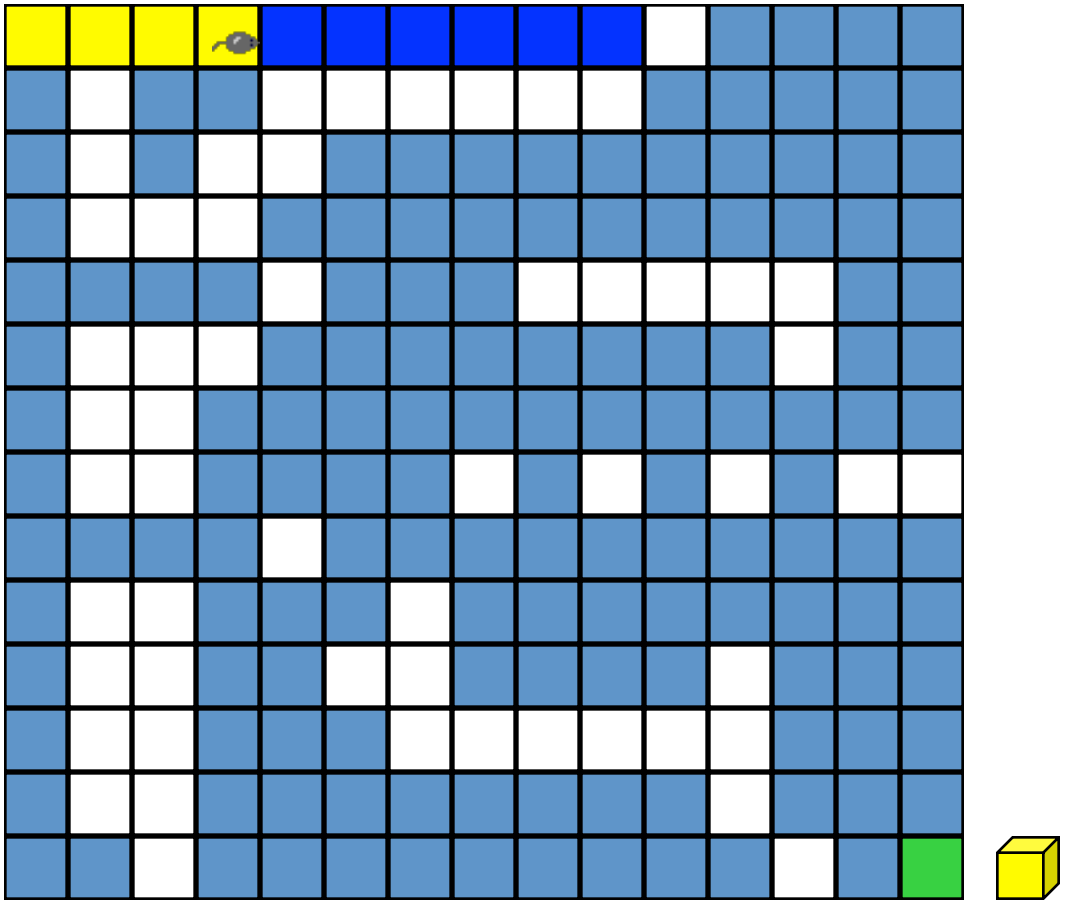
: 6+ ; <$9+0/=+&0$#+$" , +/ . $-* , /0/#>$

! "#$%&$ ' $ (") *$



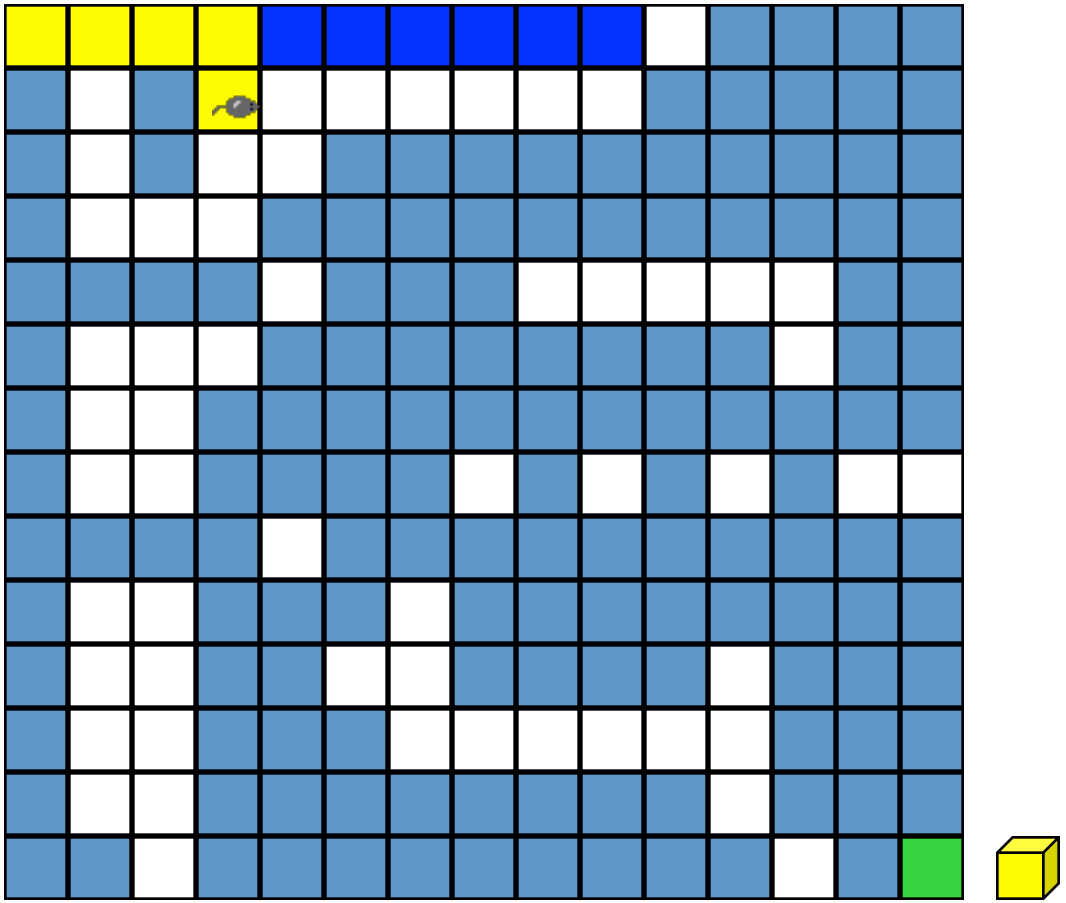
(+, *$?";<5"-.$8&=6$5*$-*";3$"$0@8"-*$A-+B$53/;3$"$
A+-5"-.$B+, *$/0$9+00/?6*>$

! "#$%&$ ' $ (") *$



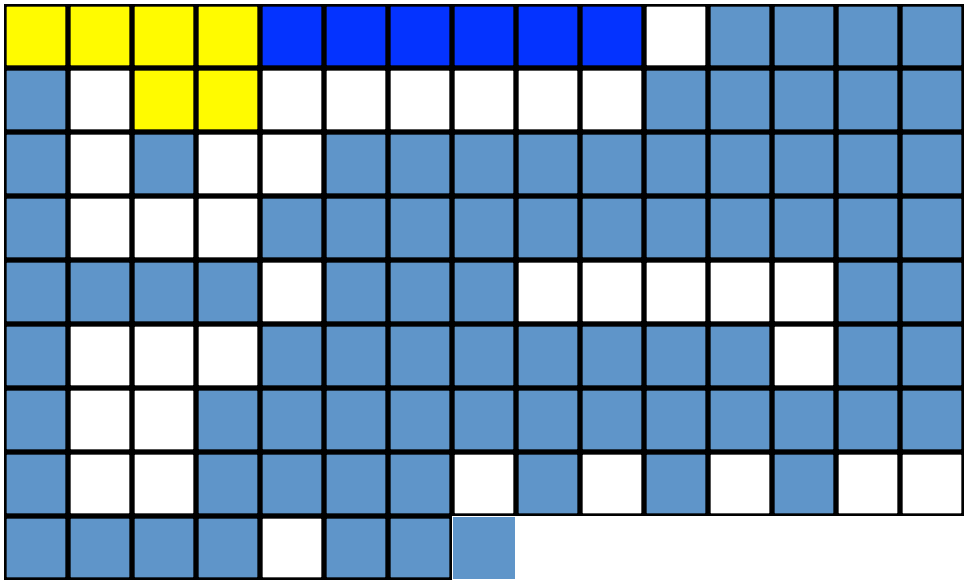
(+, *$. +5&>$

! "#$%&$ ' $ (") *$



(+, *$6*7>$

! "#$%&$ ' $ (") *$

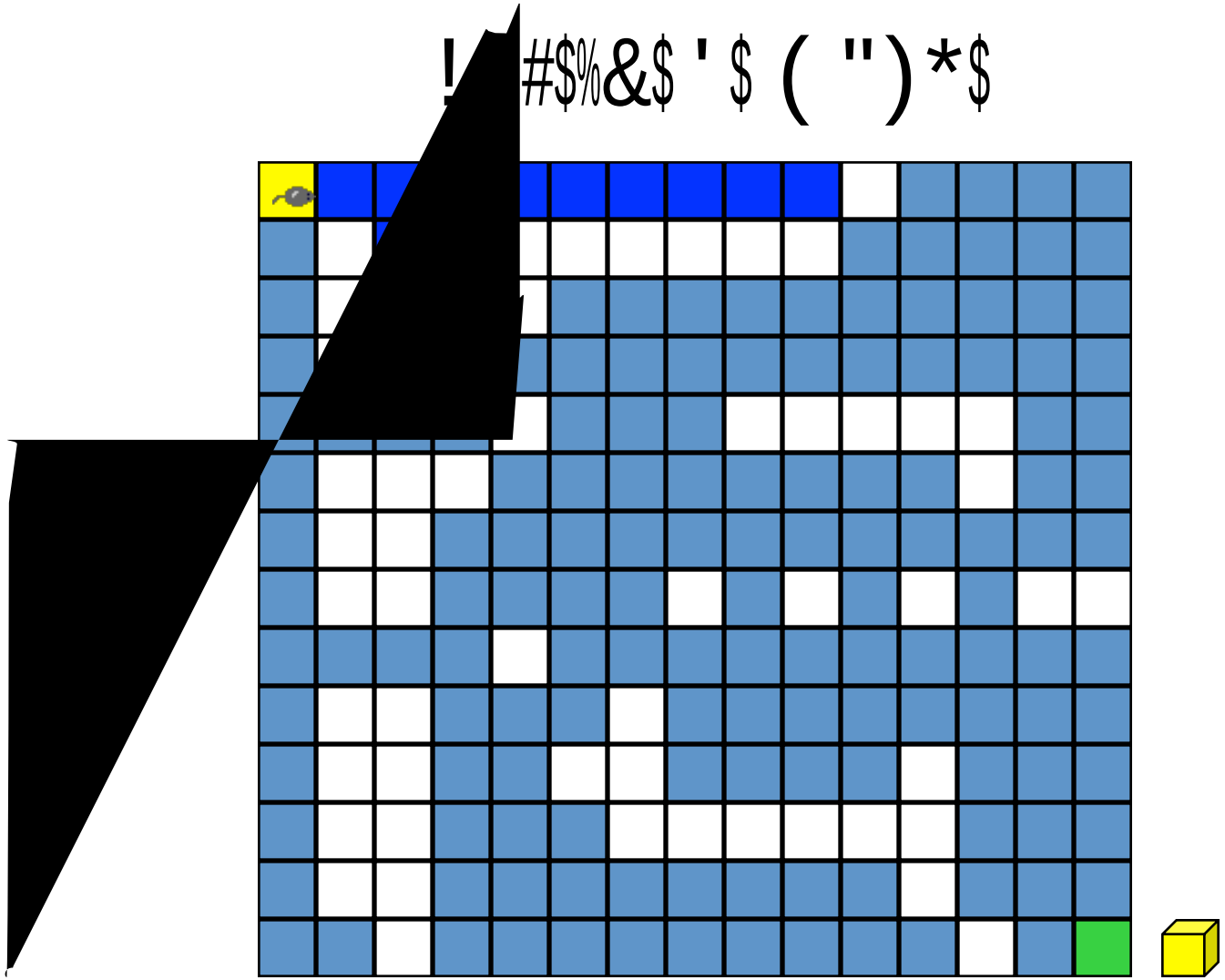


(+, *$. +5&›$

! "#$%&$ ' $ (") *$

(+, *$?";<5"-.$8&=6$5*$-*";3$"$0@8"-*$A-+B$53/;3$"$
A+-5"-.$B+, *$/0$9+00/?6*›$

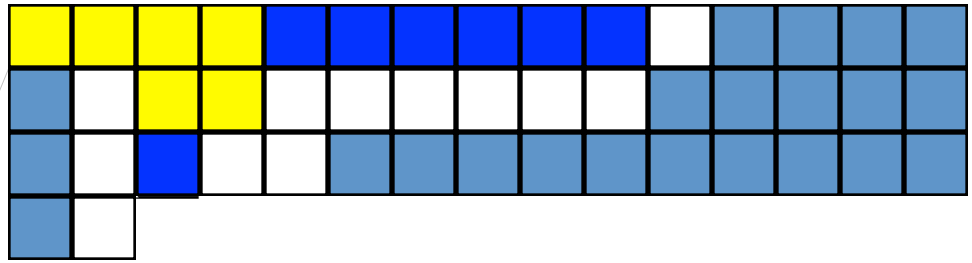
! #$%&$ ' $ (") *$



(+, *$?";<5"-.$8&=6$5*$-*";3$"$0@8"-*$A-+B$53/;3$"$
A+-5"-.$B+,*$/0$9+00/?6*>$

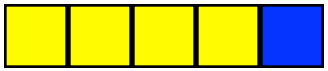
Mo e do n ard.

! "#$%&$ ' $ (") *$



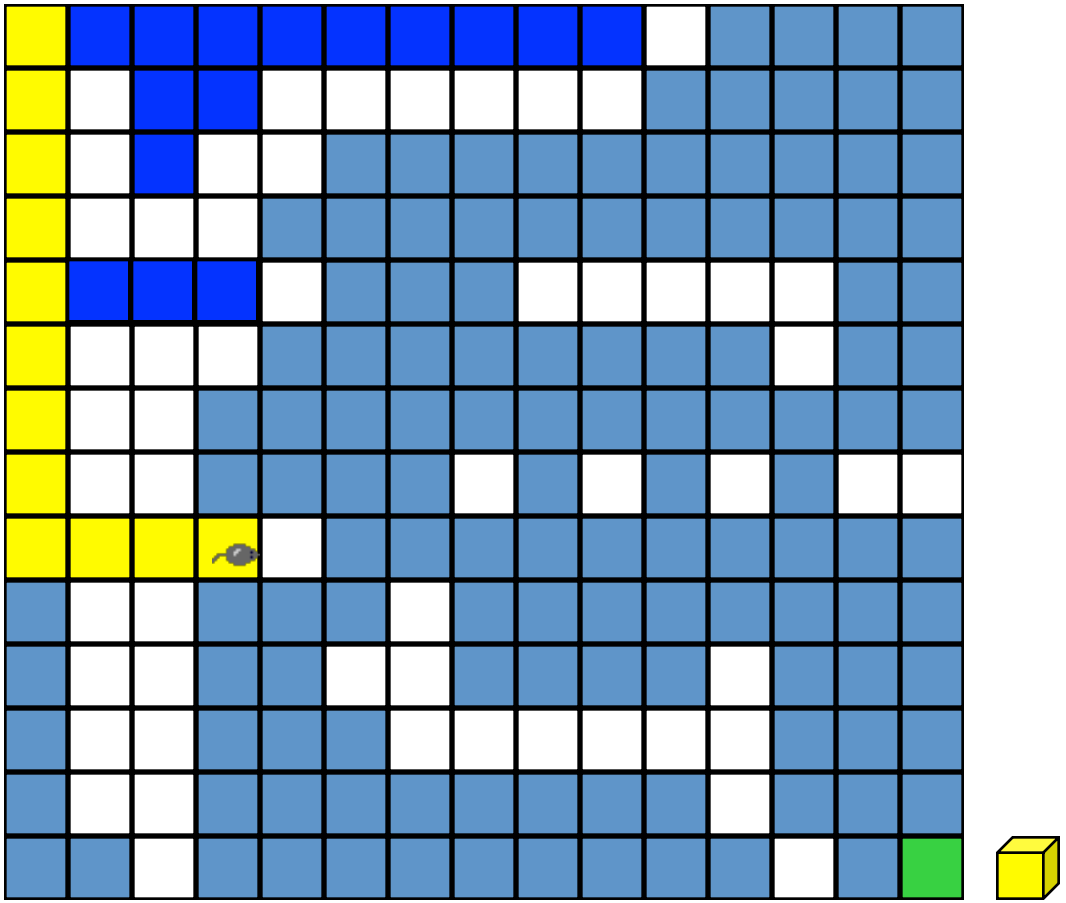
(+ , *$-/23#›$

! "#$%&$ ' $ (") *$



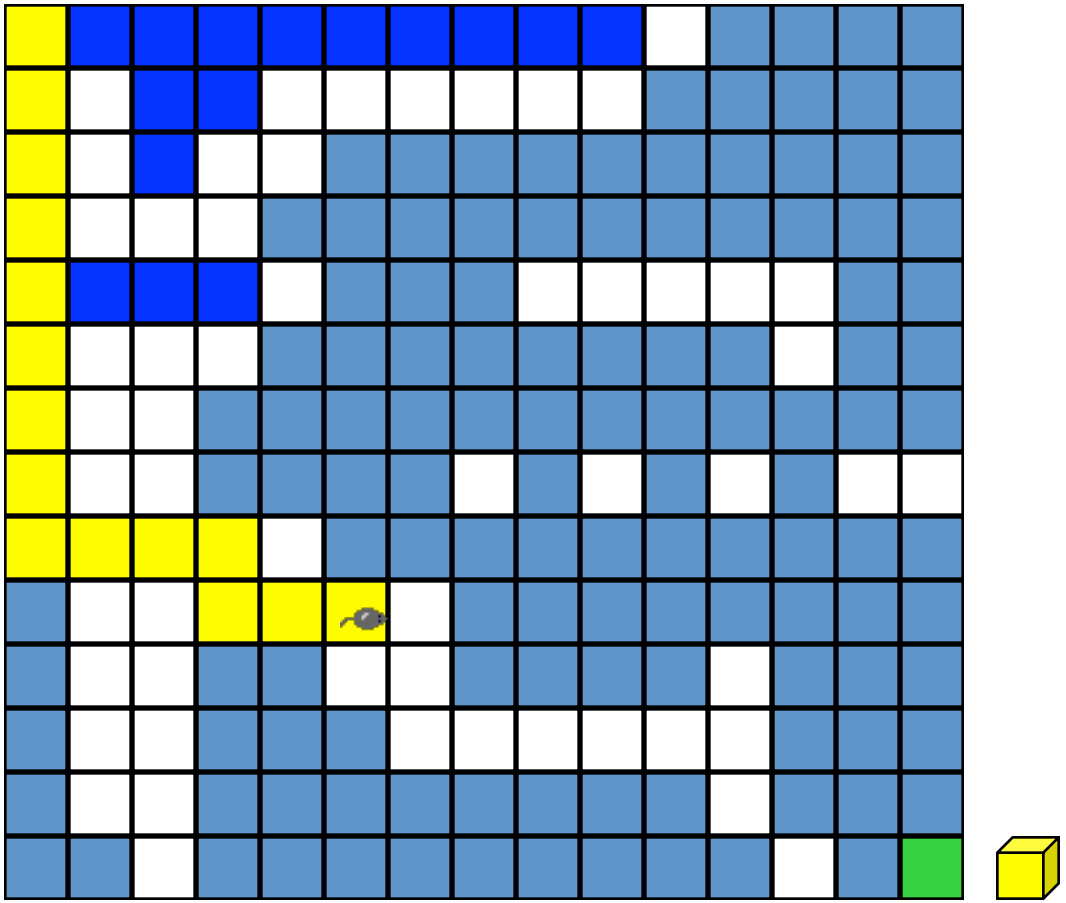
(+, *$-/23#›$

! "#$%&$ ' $ (") *$



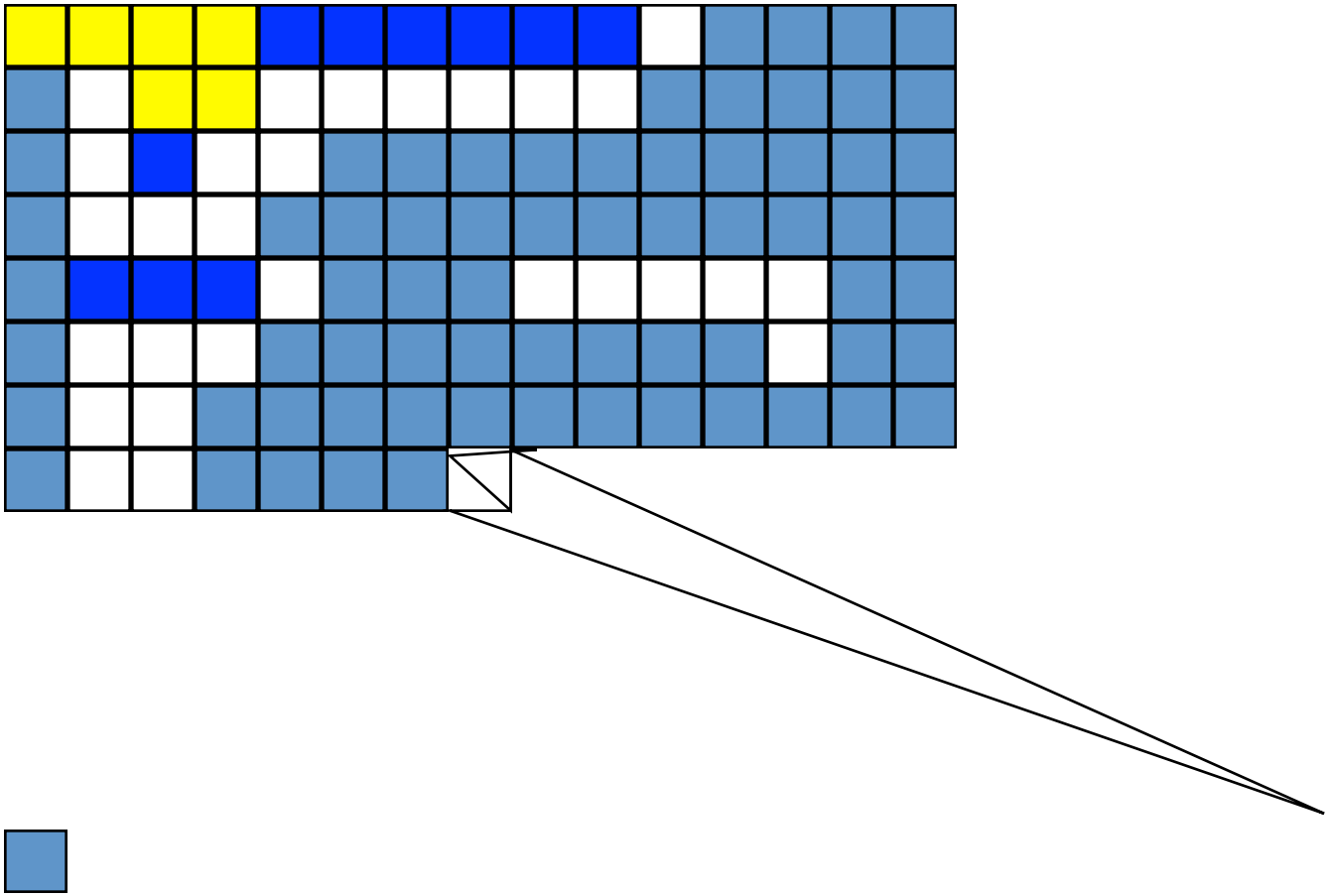
(+, *$+&*$. +5&$"&. $#3*&$-/23#›$

! "#$%&$ ' $ (") *$



(+, *$+&*$89$"&. $#3*&$-/23#›$

! "#$%&$ ' $ (") *$



(+, *$. +5&$#+$*C/#$"&. $*"#$;3**0*›$

D#"&./&2E$F+&.*-/&2E

(+,*$A+-5"-.$53*&*,*-$

G+$5"66$H$&+#$,/0/#*.$

(+,*$?";<$IIII$JKFL$

!*B*B?*-$#3*$A++#9-/&#0$

K!EE:*M*-L$

GNOP$9+00/?6*$B+,*$A-+B$9-*,/+80$9+0/=+&$

D#+-"2*L$

DP'QR$

!

%#S0$''$TKGU$6/A*$E

J+5$#+$98#$''&$*&.$#+$#3/0$B/0*-VL$!%W$

U+.$?6*00$/#X$

Y''B*$/#X$

F3*&*,*-$*C/0#$''$9+00/?6*$B+,*$A-+B$9-*,/+80$9+0/=+&0$

F3*&*,*-$#3*$0#'';<$/0$&+#$*B9#V$

0	1	0	0	1	1	0	1	1
1	**0**	**0**	1	0	0	1	1	1
0	1	1	**0**	1	1	1	0	1
1	1	0	**0**	1	**0**	**0**	1	0
1	0	0	1	**0**	1	1	**0**	1
0	0	1	1	0	1	**0**	1	1
0	1	0	0	1	1	**0**	**0**	**0**

:

, 1 ≤ ≤ , 1 ≤ ≤ .

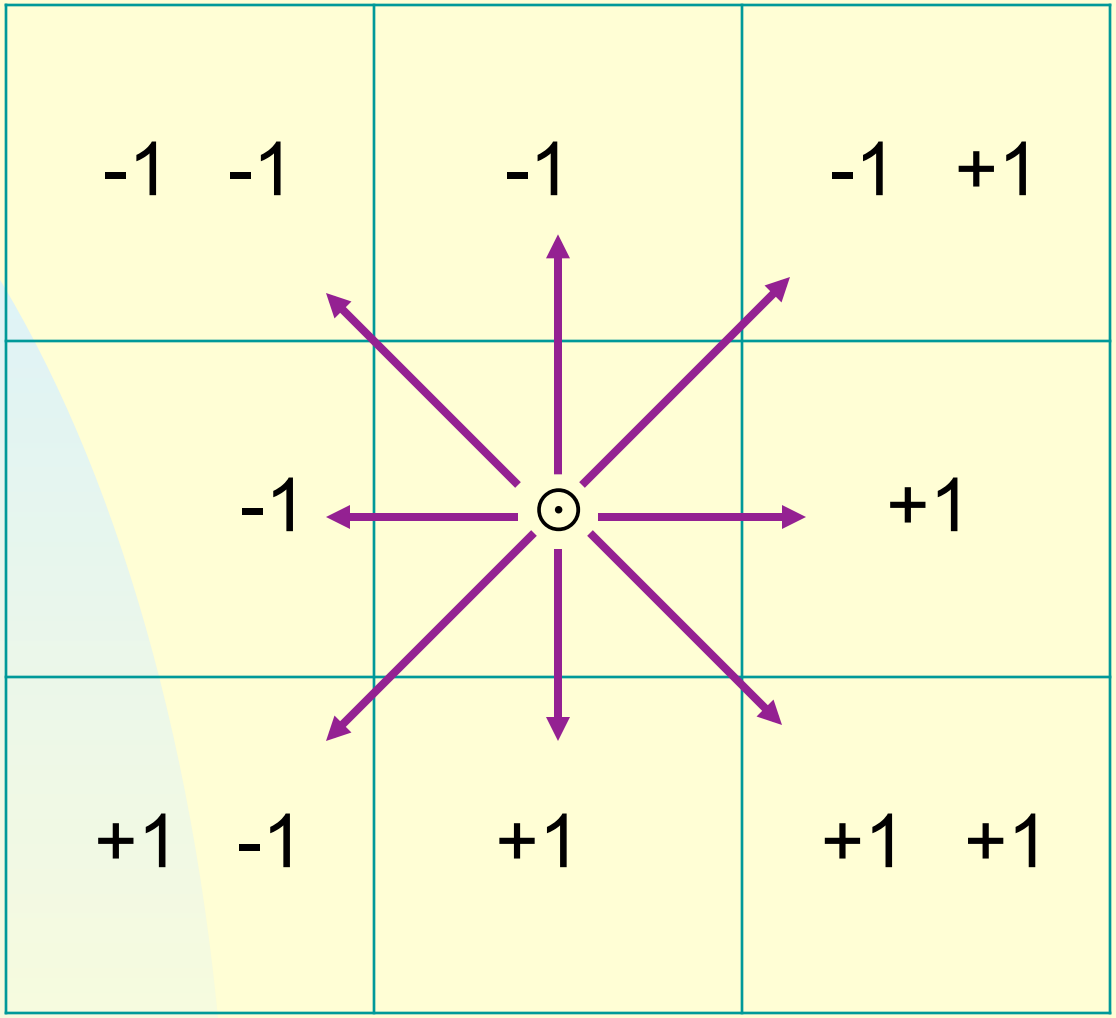
1--- , 0 --- .

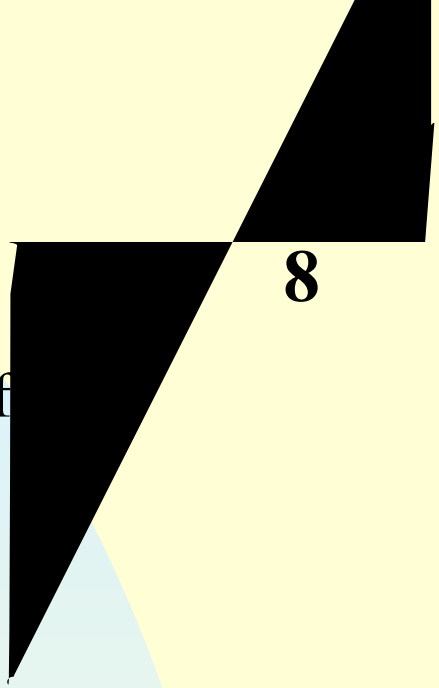
: 1 1 , : .

: .

1 , +2 +2 .

8 : , , , , , , .


-1 -1
-1
-1 +1
-1
+1
+1 -1
+1
+1 +1



```
                8        :
        off
        a,b;
;
        directions   ,   ,  ,   ,  ,   ,  ,    ;
offsets mo  e[8];
```

q	mo e[q].a	mo e[q].b
N	-1	0
NE	-1	1
E	0	1
SE	1	1
S	1	0
SW	1	-1
W	0	-1
NW	-1	-1

Table of mo es

, :

\+ . ;

\+ . ;

The basic idea:

:

+2 +2 , 0.

1 .

:

Initiali e stack to the ma e e nce ates and direction east;

(stack is not empt)

(i, j, dir)=coordinates and directi from top of stack;

pop the stack;

(there are more mo es from (i, j))

(g, h)= coordinates of ne t mo e ;

((g=l1kf3 cm BT 24 (;) Tj]TJ ETW.2 (s) -0 1 0 0 1 0 (s) -06 B

((!ma e[g][h]) **&&** (!mark[g][h])) // legal and not isited

mark[g][h]=1**;**

dir=ne t direction to tr **;**

push (i, j, dir) to stack**;**

(i, j, dir) = (g, h, N)**;**

<< No path in ma e. << **;**

```
(              m,              p)
//Output a path (if an  ) in the ma  e; ma  e[0][i] = ma  e[m+1][i]
// = ma  e[j][0] = ma  e[j][p+1] = 1, 0 ≤ i ≤ p+1, 0 ≤ j ≤ m+1.
   // start at (1,1)
   mark[1][1]=1;
   Stack<Items> stack(m*p);
   Items temp(1, 1, E);
   stack.Push(temp);
         (!stack.IsEmpt  ())

        temp= stack.Top();
       Stack.Pop();
           i=temp.  ;      j=temp.  ;      d=temp.dir;
```

```
(d<8)

  g=i+mo  e[d].a;      h=j+mo  e[d].b;
((g==m) && (h==p))   // reached e  it
// output path
        <<stack;
        << i<<    << j<<    <<d<<        ; // last t   o
        << m<<    << p<<        ;          // points
          ;
```

```
((!ma  e[g][h]) && (!m    [g][h]))   //ne   position
mark[g][h]=1;
temp.  =i; temp.  =j; temp.dir=d+1;
stack.Push(temp);
i=g ; j=h ; d=N;   // mo  e to (g, h)

    d++;  // tr   ne  t direction


<<  No path in ma  e.  <<        ;
```

<<

:

< T>

ostream& <<(ostream& os, Stack<T>& s)

os << top= <<s.top<< ;
(i=0;i<=s.top;i++);
os<<i<< : <<s.stack[i]<< ;
os;

<<

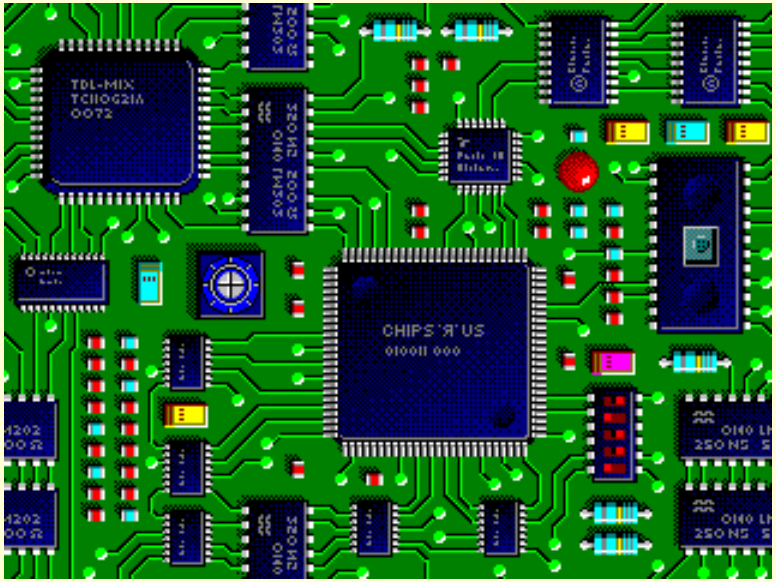
.

```
ostream&          <<(ostream& os,Items& item)

        os<<item.  <<  ,  <<item.  <<  ,  <<item.dir-1;
// note item.dir is the ne  t direction to go so the current
// direction is item.dir-1.
```

,

(*).

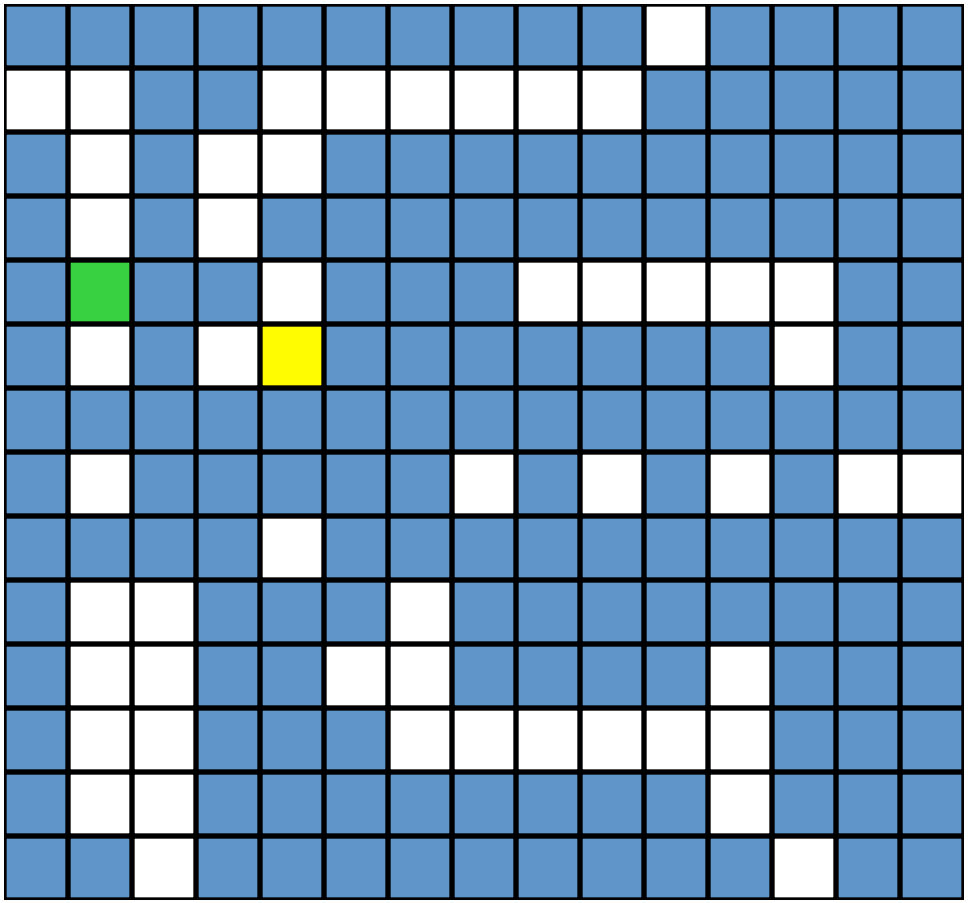
: 157-2, 3


TDL-MIX
TCHOG2IA
0072
CHIPS 'R' US

T**SO$ F/-*$!+8#*-$

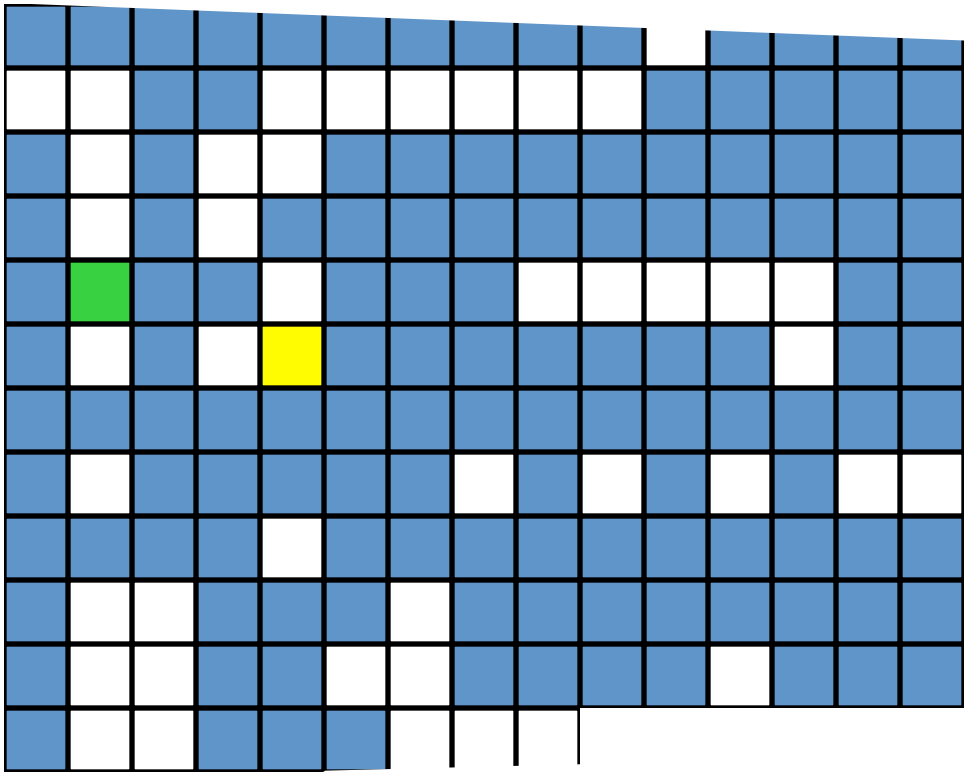
start pin

end pin



Label all reachable squares 1 unit from start.

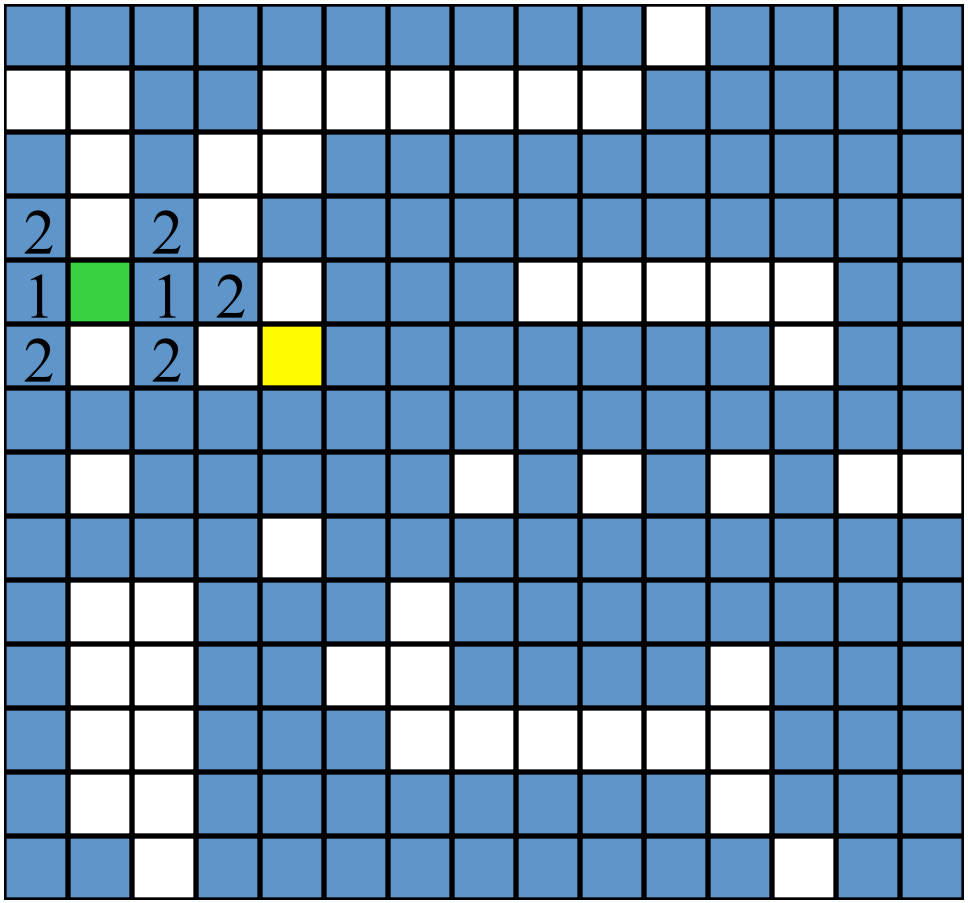
T**SO$ F/-*$!+8#*-$



T**O F/-*$!+8#*-$

start pin

end pin

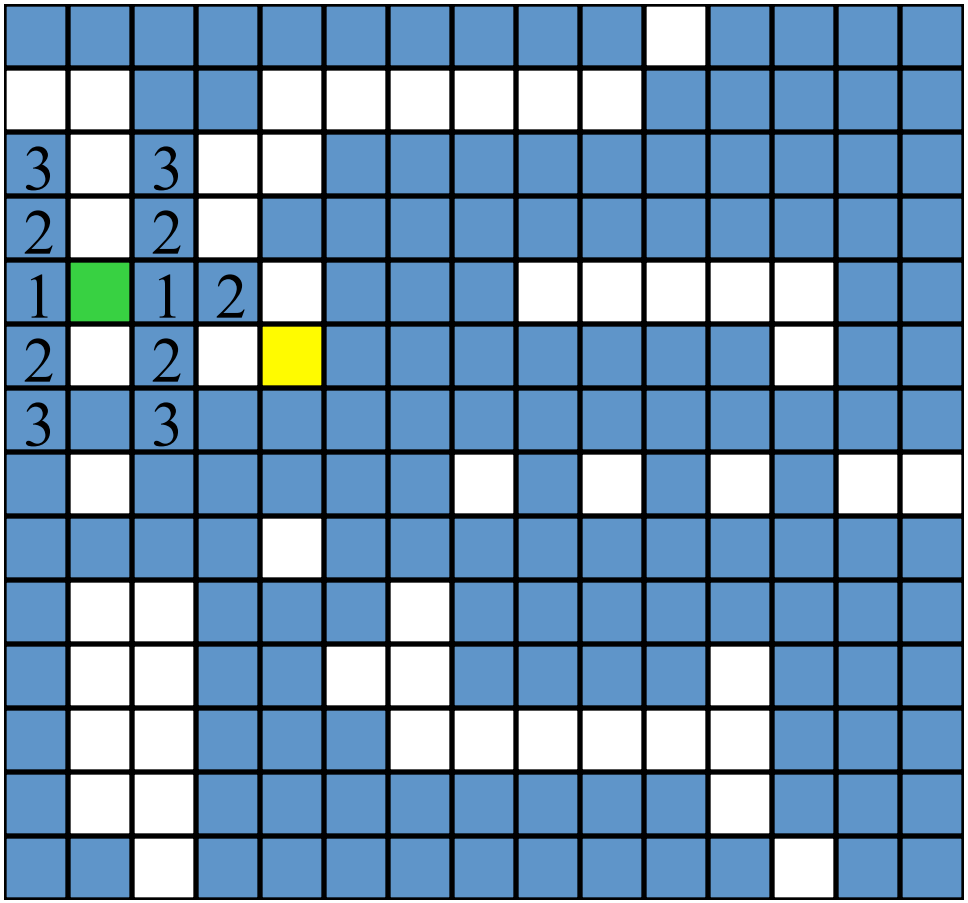


Label all reachable unlabeled squares 3 units from start.

T**SO$ F/-*$!+8#*-$

start pin

end pin

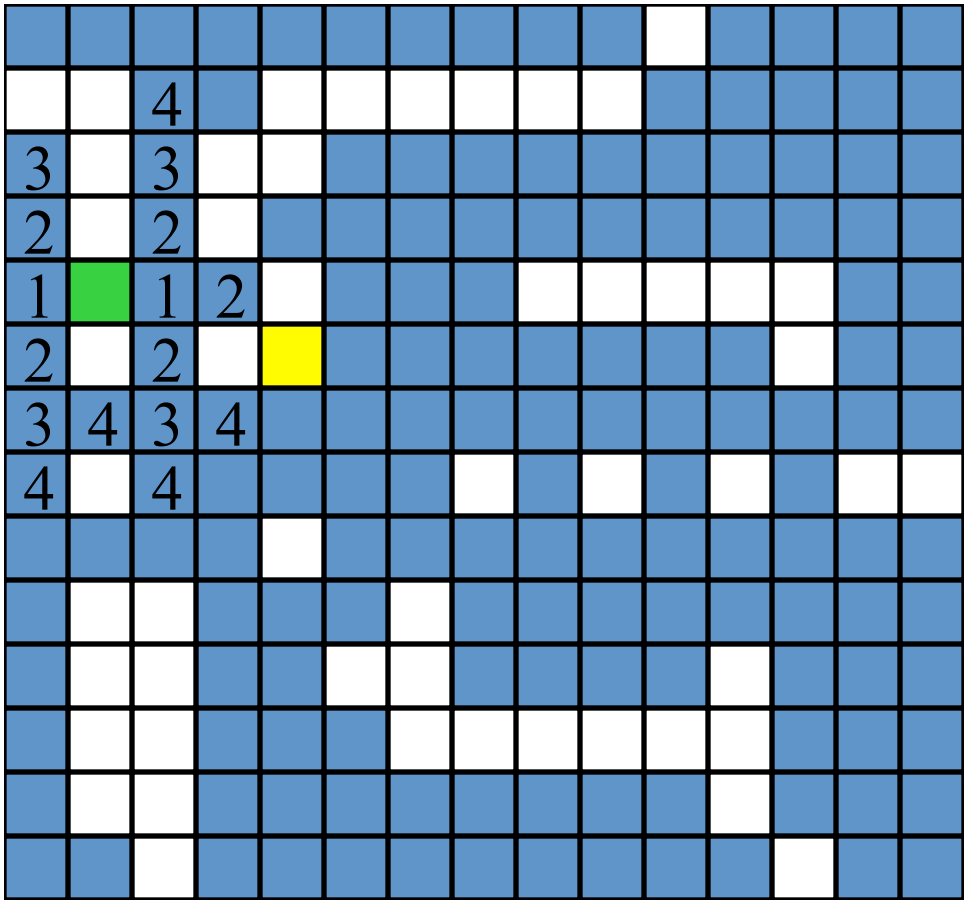


Label all reachable unlabeled squares 4 units from start.

T**SO$ F/-*$!+8#*-$

start pin

end pin

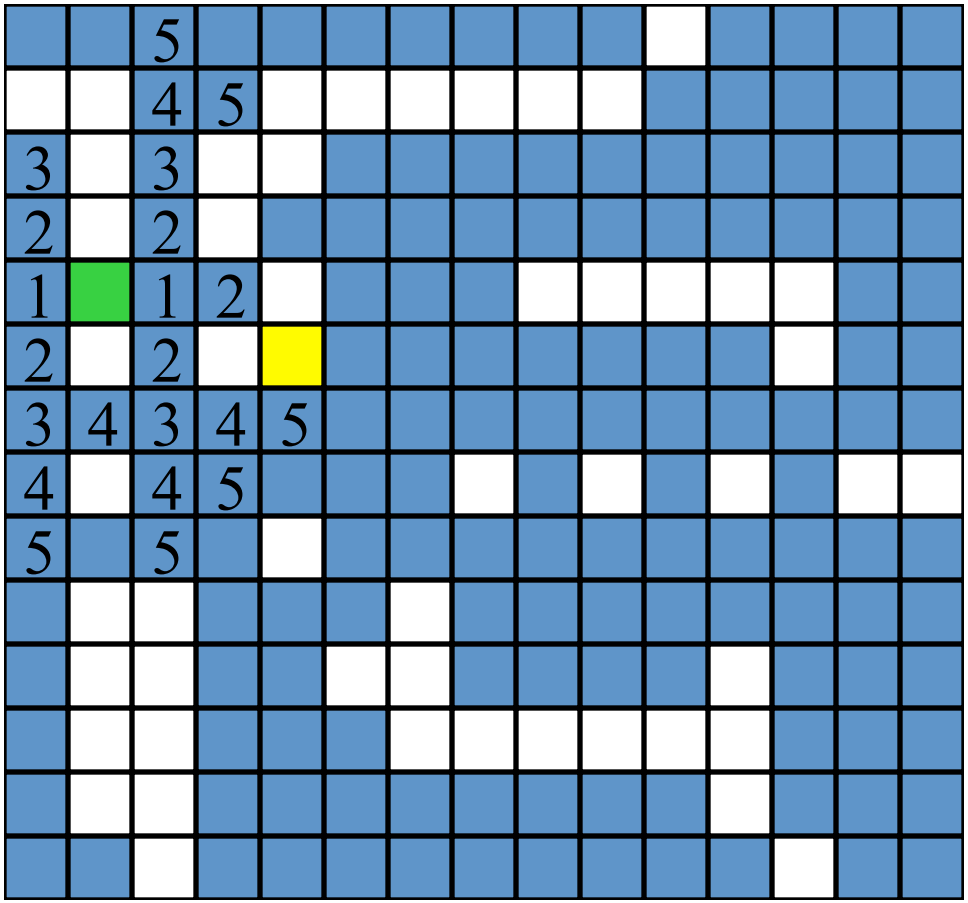


Label all reachable unlabeled squares 5 units from start.

T**O F/-*$!+8#*-$

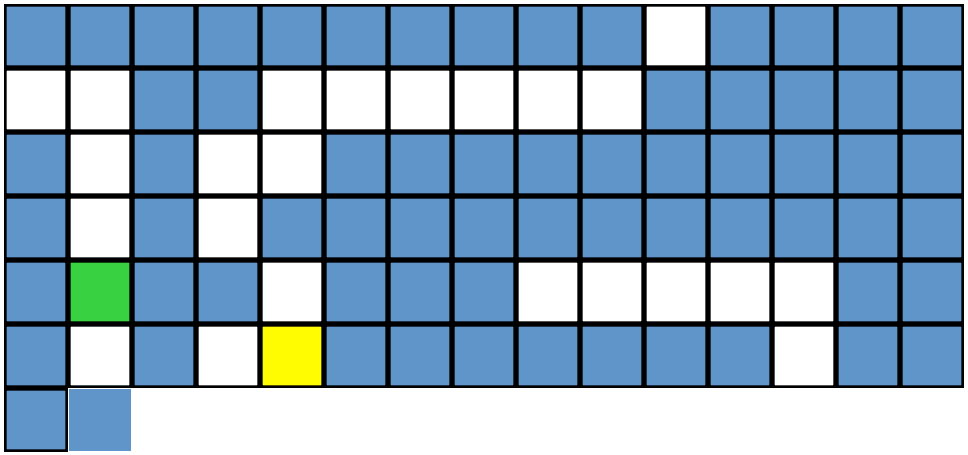
start pin

end pin



Label all reachable unlabeled squares 6 units from start.

T**SO$ F/-*$!+8#*-$



, ,

. ,

: / + * *

: / * + *

: - (

).

: .

1	, !
2	*, /, %
3	+, -
4	, , ,
5	, !
6	&&
7	

Problem:

ho to e al ate an e pression?

: / * + *

,

, ≥1.

:

A B / C D E * + A C *

T_6 is the res lt.

:

:

- ✓
- ✓
- ✓

```
E  al(E  pression e)
// e  aluate  tfi   e  pression e. It is assumed that the
// last token  s  # . A function Ne  tToken is used to get
// the ne  t t  n from e. Use stack.
  Stack<To  n> stack;  //initiali  e stack
       (Token    = Ne  tToken(e);   != # ;   =Ne  tToken(e))
       (   is an operand) stack.Push(  );
            // operator
       remo  e the correct number of operands for operator
       from stack; perform the operation    and store the result
       (if an  ) onto the stack;
```

Infi to Postfi

Idea: note the order of the operands in both infi and postfi

infi : A / B C + D * E A * C

postfi : A B / C D E * + A C *

the same!

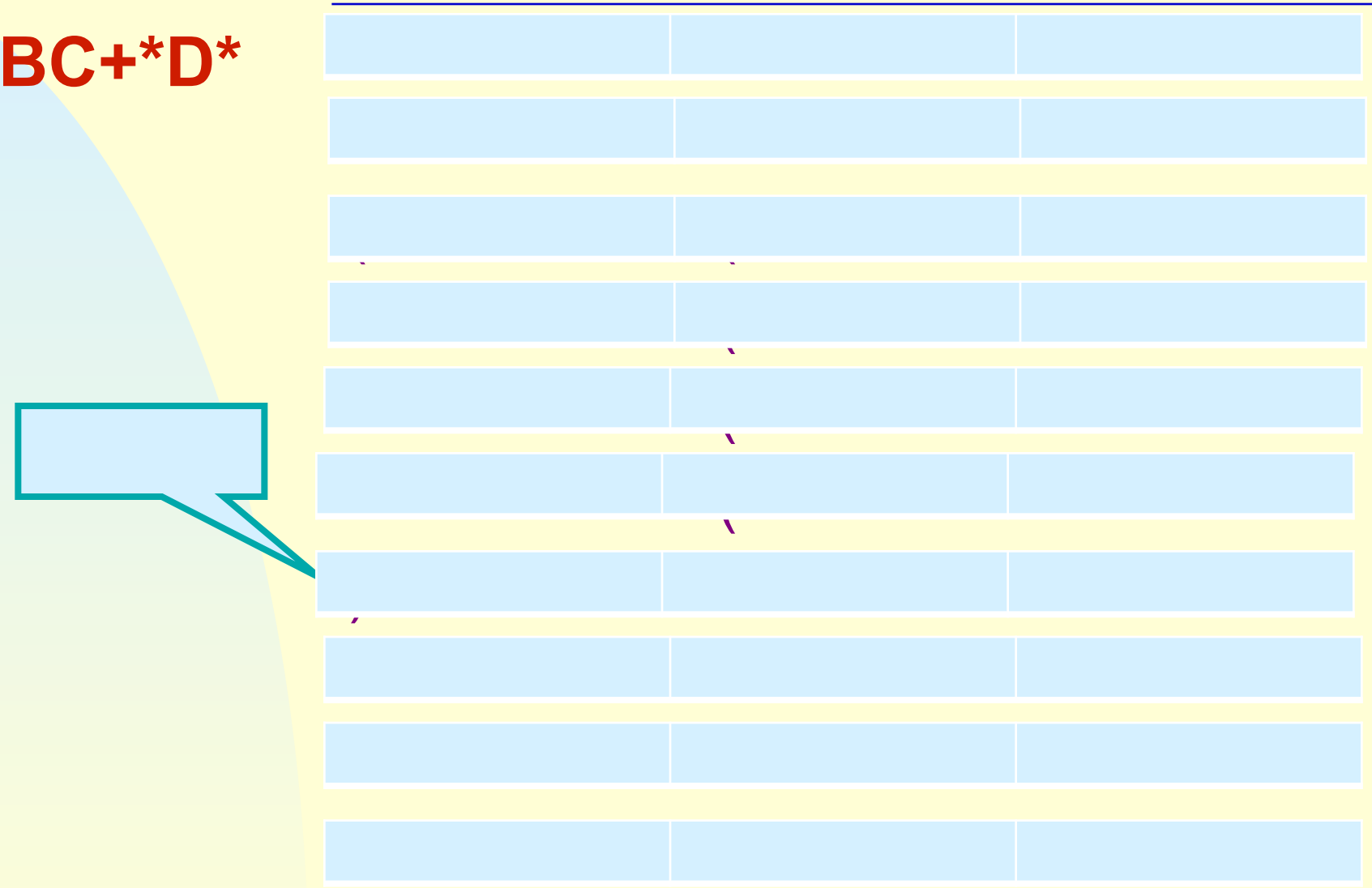
immediatel passing an operands to the o tp t store the operators in a **stack** ntil the right time.

e.g.

A*(B+C)*D → ABC+*D*

A*(B+C)*D

→ABC+*D*



,

, ,

.

- (-)
- (-)
- . 3.15

- (( ) 8, (( ) 0, (#) 8

Hence the r le:

Operators are taken o t of stack as long as their isp is n mericall less than or eq al to the icp of the ne operator.

```
Postfi   (E  pression e)
// output the postfi   of the infi   e  pression e. It is assumed
// that the last token in e is  # . Also,  #  is used at the bottom
// of the stack.
  Stack<Token> stack;  //initiali  e stack
  stack.Push( # );
```

```
(Token   =Ne  tToken(e);   != # ;   =Ne  tToken(e))
(   is an operand)        <<  ;
      (  == ) )
  // unstack until  (
        (; stackTop()!= ( ; stack.Pop())
            <<stack.Top();
   stack.Pop();  // unstack  (

      //    is an operator
       (; isp(stack.Top()) <= icp(  ); stack.Pop())
           <<stack.Top();
  stack.Push(  );

// end of e  pression, empt   the stack
    (; !stack.IsEmpt  ());       <<stack.Top(), stack.Pop());
     <<        ;
```

:

- :

, ( ).

- 1 (#) +

.

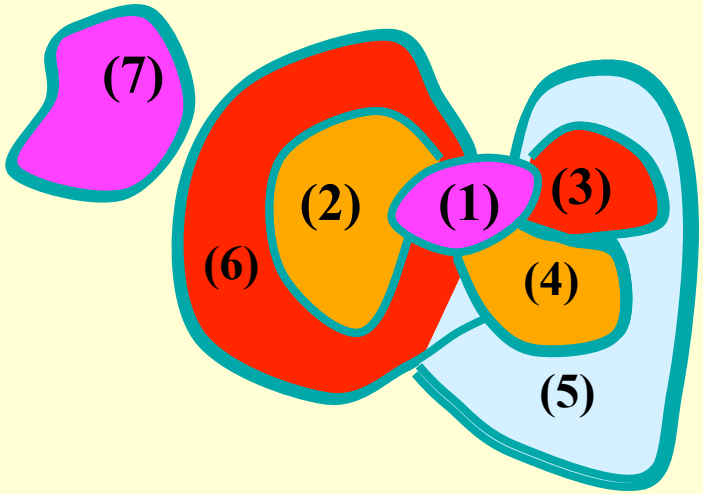
E ercises: P165-1,2

Can e e al ate infi e pressions directl ?

infi : A / B C + D * E A * C

7 7

	1	2	3	4	5	6	7
1	0	1	1	1	1	1	0
2	1	0	0	0	0	1	0
	1	0	0	1	1	0	0
	1	0	1	0	1	1	0
	1	0	1	1	0	1	0
	1	1	0	1	1	0	0
	0	0	0	0	0	0	0



1	2	3	4	5	6	7
1	2	2				1