

Winter 2016
COMP-250: Introduction
to Computer Science

Lecture 5, January 26, 2016

HWI

```
class Number{  
  
    short Base;  
    short[] Int;  
    short[] NonRep;  
    short[] Rep;  
  
};
```

HW1

```
class Number{  
  
//=====+  
    public Number convert(Number A, short Base) {  
        Number B=new Number();  
        B.Base=Base;  
        B.Int=A.NonRep; B.NonRep=A.Int; B.Rep=A.NonRep;  
        // my code above is just to make sure it compiles and runs  
        return B;  
    }  
//=====+  
  
    short Base;  
    short[] Int;  
    short[] NonRep;  
    short[] Rep;  
};
```

HWI

```
class Number{  
  
    public void printShortArray(short[] S) {  
        for (int i = S.length-1; i>=0; i--) {  
            System.out.print(S[i]);  
        }  
    }  
  
    public void printNumber(Number N) {  
        System.out.print("(");  
        N.printShortArray(N.Int);  
        System.out.print(".");  
        N.printShortArray(N.NonRep);  
        System.out.print("{");  
        N.printShortArray(N.Rep);  
        System.out.print("})_");  
        System.out.println(N.Base);  
    }  
  
    short Base; short[] Int, NonRep, Rep;  
};
```

HWI

```
package conv;
public class tester {
    public static void main(String[] args) {

        class Number{
            short Base; short[] Int,NonRep,Rep;
        };

        Number N1=new Number() ;
        N1.Base=10;
        short[] X = {9,1}; N1.Int=X;
        short[] Y = {7,4,2}; N1.NonRep=Y;
        short[] Z = {}; N1.Rep=Z;
        N1.printNumber(N1);

        Number N2=new Number() ;
        short R=2;

        N2=N1.convert(N1, R);
        N2.printNumber(N2);

    }
}
```

Array Algorithms

Sorting

ALGORITHM: INSERTION SORT

INPUT: an array $a[]$ with N elements that can be compared ($<$, $=$, $>$)

OUTPUT: the array $a[]$ containing the same elements, in increasing order

for $k = 1$ to $N - 1$ **do**

$tmp \leftarrow a[k]$

$i \leftarrow k$

while $(i > 0) \ \& \ (tmp < a[i - 1])$ **do**

$a[i] \leftarrow a[i - 1]$

$i \leftarrow i - 1$

end while

$a[i] = tmp$

end for

Insertion Sort

a :	[11 , 13 , 2 , 5 , 21 , 99 , 12 , 51]	tmp :	13
a :	[11 , 13 , 2 , 5 , 21 , 99 , 12 , 51]	tmp :	13
a :	[11 , 13 , 2 , 5 , 21 , 99 , 12 , 51]	tmp :	2
a :	[11 , 13 , 2 , 5 , 21 , 99 , 12 , 51]	tmp :	2
a :	[2 , 11 , 13 , 5 , 21 , 99 , 12 , 51]	tmp :	2
a :	[2 , 11 , 13 , 5 , 21 , 99 , 12 , 51]	tmp :	5
a :	[2 , 11 , 13 , 5 , 21 , 99 , 12 , 51]	tmp :	5
a :	[2 , 5 , 11 , 13 , 21 , 99 , 12 , 51]	tmp :	5
a :	[2 , 5 , 11 , 13 , 21 , 99 , 12 , 51]	tmp :	21
a :	[2 , 5 , 11 , 13 , 21 , 99 , 12 , 51]	tmp :	21
a :	[2 , 5 , 11 , 13 , 21 , 99 , 12 , 51]	tmp :	99

Insertion Sort

a : [2, 5, 11, 13, 21, 99, 12, 51] **tmp** : 99

a : [2, 5, 11, 13, 21, 99, 12, 51] **tmp** : 99

a : [2, 5, 11, 13, 21, 99, 12, 51] **tmp** : 12

a : [2, 5, 11, 13, 21, 99, 12, 51] **tmp** : 12

a : [2, 5, 11, 12, 13, 21, 99, 51] **tmp** : 12

a : [2, 5, 11, 12, 13, 21, 99, 51] **tmp** : 51

a : [2, 5, 11, 12, 13, 21, 99, 51] **tmp** : 51

a : [2, 5, 11, 12, 13, 21, 51, 99] **tmp** : 51

a : [2, 5, 11, 12, 13, 21, 51, 99] **tmp** : 51

Insertion Sort

a : [2 , 5 , 11 , 13 , 21 , 99 , **12** , 51] **tmp** : **12**

a : [2 , 5 , 11 , 13 , 21 , **99** , 99 , 51] **tmp** : **12**

a : [2 , 5 , 11 , 13 , **21** , 21 , 99 , 51] **tmp** : **12**

a : [2 , 5 , 11 , **13** , 13 , 21 , 99 , 51] **tmp** : **12**

a : [2 , 5 , **11** , **12** , 13 , 21 , 99 , 51] **tmp** : **12**

a : [2 , 5 , 11 , 12 , 13 , 21 , 99 , **51**] **tmp** : **51**

Analysis of Insertion Sort

ALGORITHM: INSERTION SORT

INPUT: an array $a[]$ with N elements that can be compared ($<$, $=$, $>$)

OUTPUT: the array $a[]$ containing the same elements, in increasing order

for $k = 1$ to $N - 1$ **do**

$tmp \leftarrow a[k]$
 $i \leftarrow k$ } **cst**

while $(i > 0) \ \& \ (tmp < a[i - 1])$ **do** } **cst~linear**
 $a[i] \leftarrow a[i - 1]$
 $i \leftarrow i - 1$

end while

$a[i] = tmp$ } **cst**

end for

Linear~quadratic

Analysis of Insertion Sort

ALGORITHM: INSERTION SORT

INPUT: an array $a[]$ with N elements that can be compared ($<$, $=$, $>$)

OUTPUT: the array $a[]$ containing the same elements, in increasing order

for $k = 1$ to $N - 1$ **do**

$tmp \leftarrow a[k]$
 $i \leftarrow k$ } **cst**

while $(i > 0) \ \& \ (tmp < a[i - 1])$ **do** } **cst**
 $a[i] \leftarrow a[i - 1]$
 $i \leftarrow i - 1$

end while

$a[i] = tmp$ } **cst**

end for

linear

$$\text{Time}(N) \geq c_1 + c_2 \times N$$

Analysis of Insertion Sort

ALGORITHM: INSERTION SORT

INPUT: an array $a[]$ with N elements that can be compared ($<$, $=$, $>$)

OUTPUT: the array $a[]$ containing the same elements, in increasing order

for $k = 1$ to $N - 1$ **do**

$tmp \leftarrow a[k]$
 $i \leftarrow k$ } **cst**

while $(i > 0) \ \& \ (tmp < a[i - 1])$ **do**

$a[i] \leftarrow a[i - 1]$
 $i \leftarrow i - 1$ } **cst**

end while

$a[i] = tmp$ } **cst**

end for

} **linear**

} **quadratic**

$$\text{Time}(N) \leq C_1 + C_2 \times N + C_3 \times N^2$$

Analysis of Algorithms

Best Case

Time (N) **is** $\Omega(N)$

Worst Case

Time (N) **is** $O(N^2)$

(Singly) Linked List

Linked Lists

List = ordered set of elements.

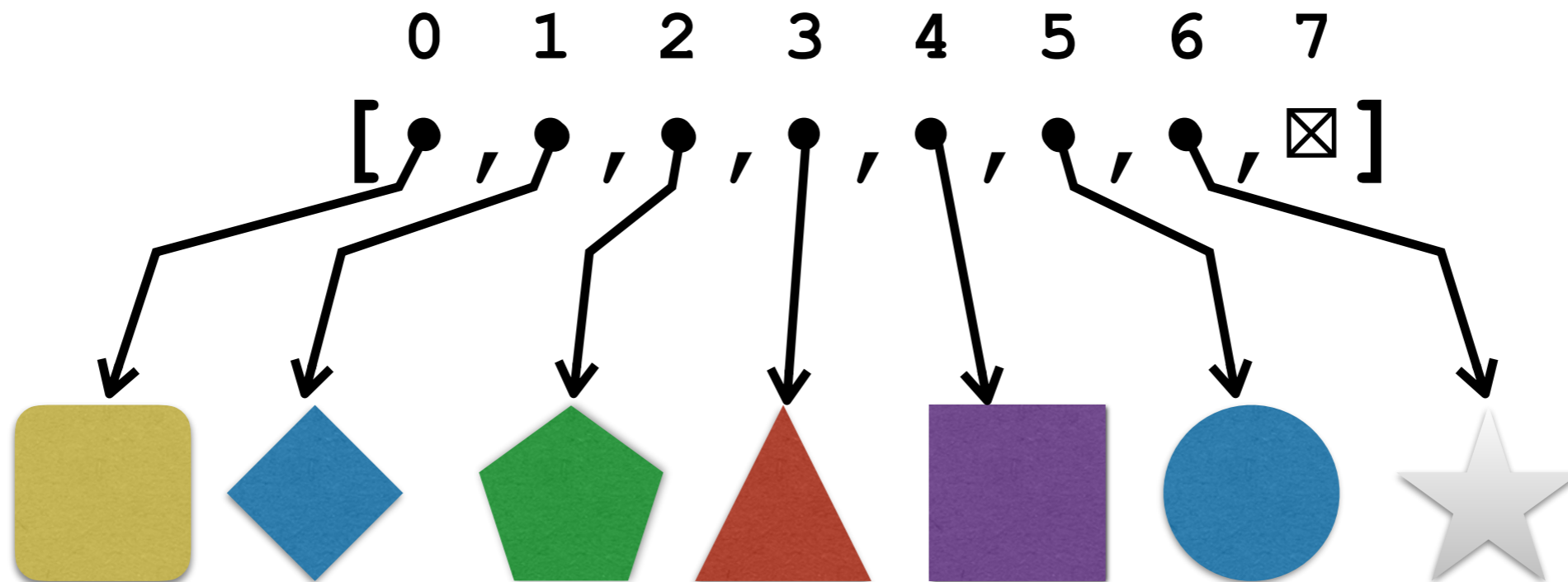
$(a_0, a_1, \dots, a_{\text{size}-1})$

Size = number of elements.

Array of integers:

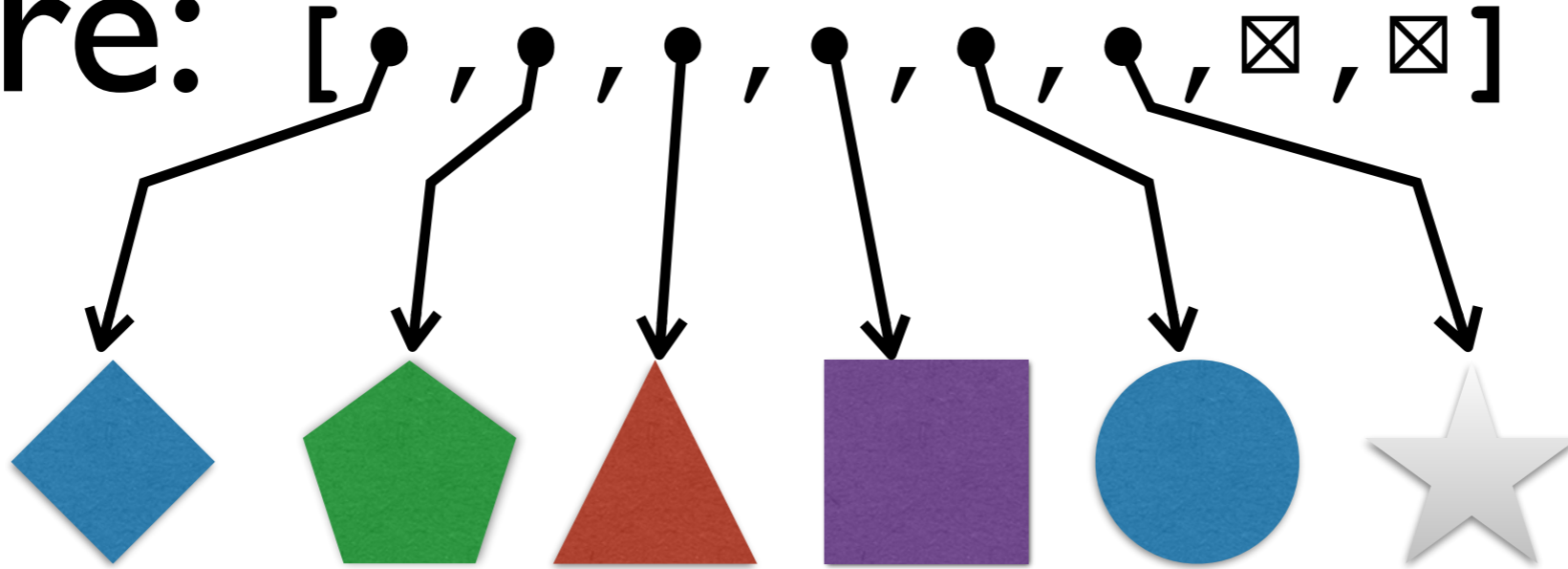
0 1 2 3 4 5 6 7
[5, 2, 9, 3, 3, 1, 7, 0]

Array of shapes:

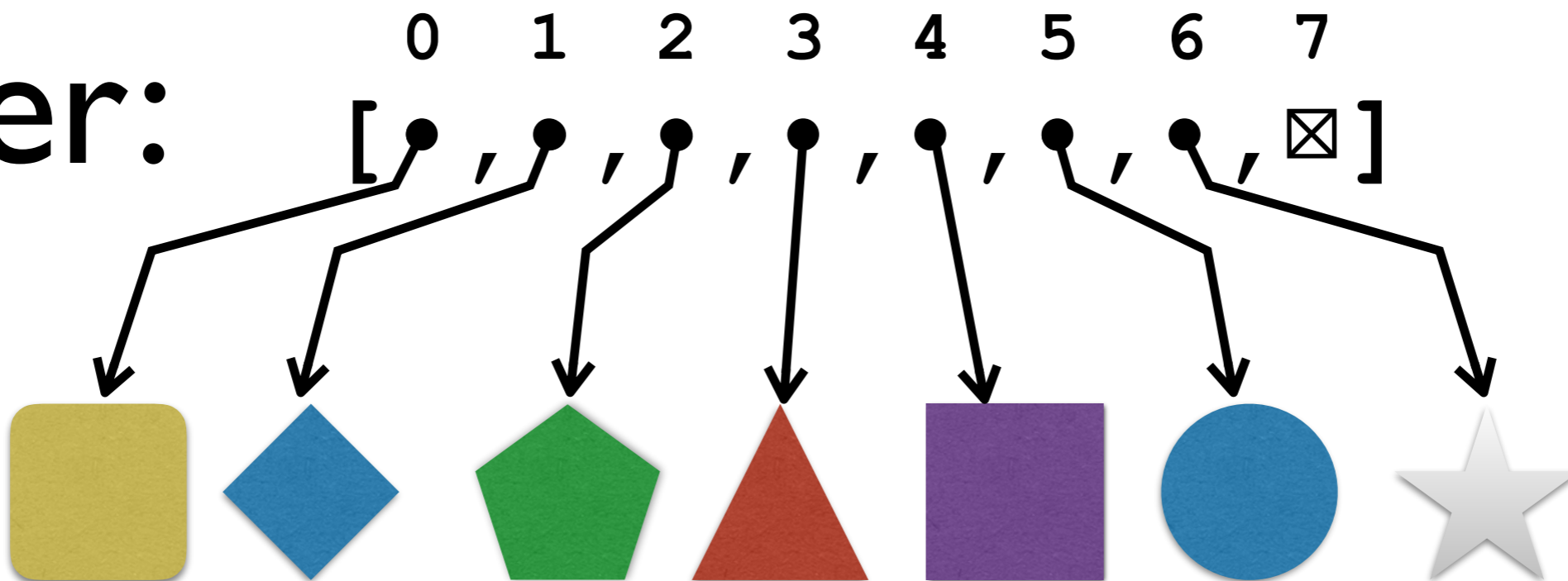


Adding an element to List

Before: [0 1 2 3 4 5 6 7
 ● , ● , ● , ● , ● , ● , ☒ , ☒]



After: [0 1 2 3 4 5 6 7
 ● , ● , ● , ● , ● , ● , ● , ☒]



Adding element to Front

```
// add new element to front of the list
// assuming that there is room left in the array
//
for (i = size; i > 0; i--)
    a[i] = a[i-1]
a[0] = new element
size = size + 1
```

Removing element at Front

```
// remove the element at front of the list
//
for (i = 1; i < size-1; i++)
    a[i-1] = a[i]
a[size-1] = null
size = size - 1
```

Adding/Removing at End

```
// add new last element to the list
// assuming that there is room left in the array
//
a[size] = new element
size = size + 1

// remove the last element from the list
//
a[size-1] = null
size = size - 1
```

Array of integers:

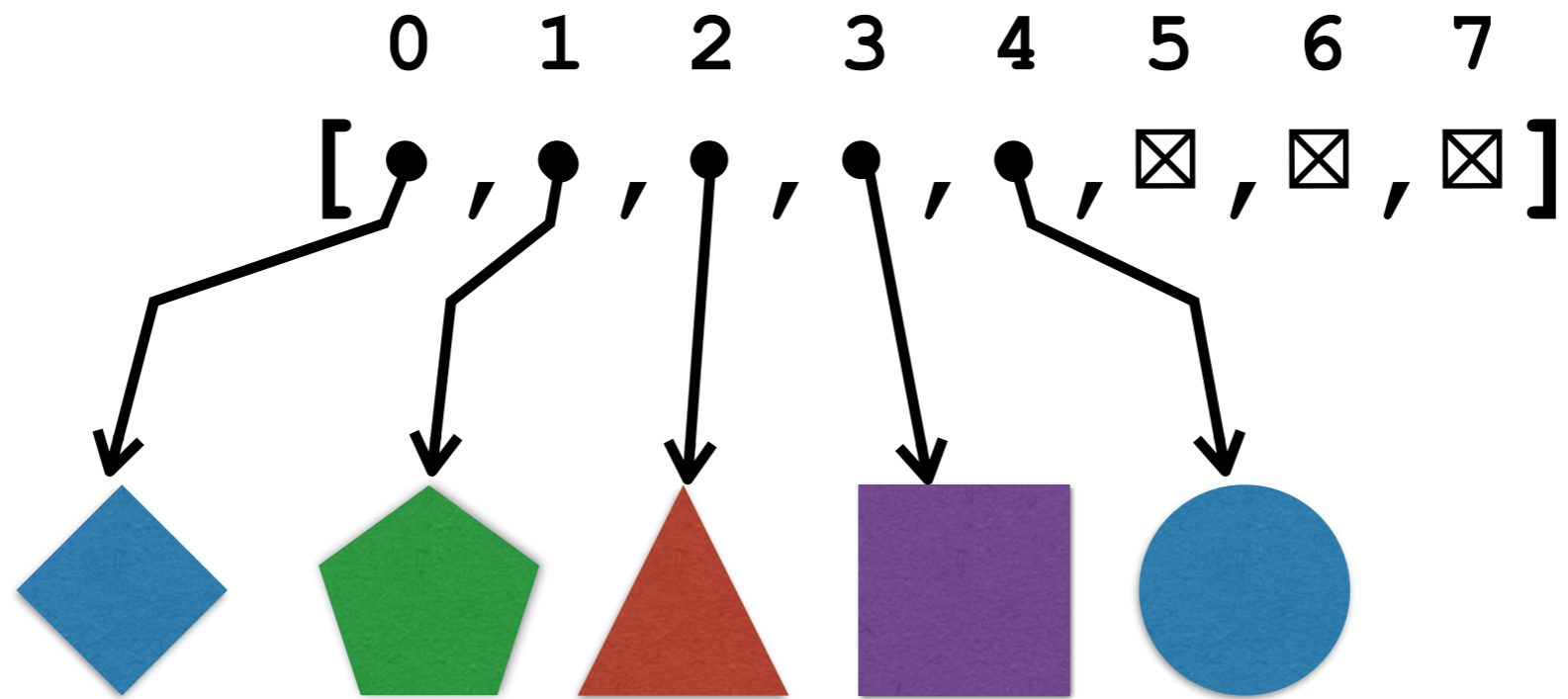
0	1	2	3	4	5	6	7
[5	,2	,9	,3	,3	,0	,0	,0]

Linked list of integers:

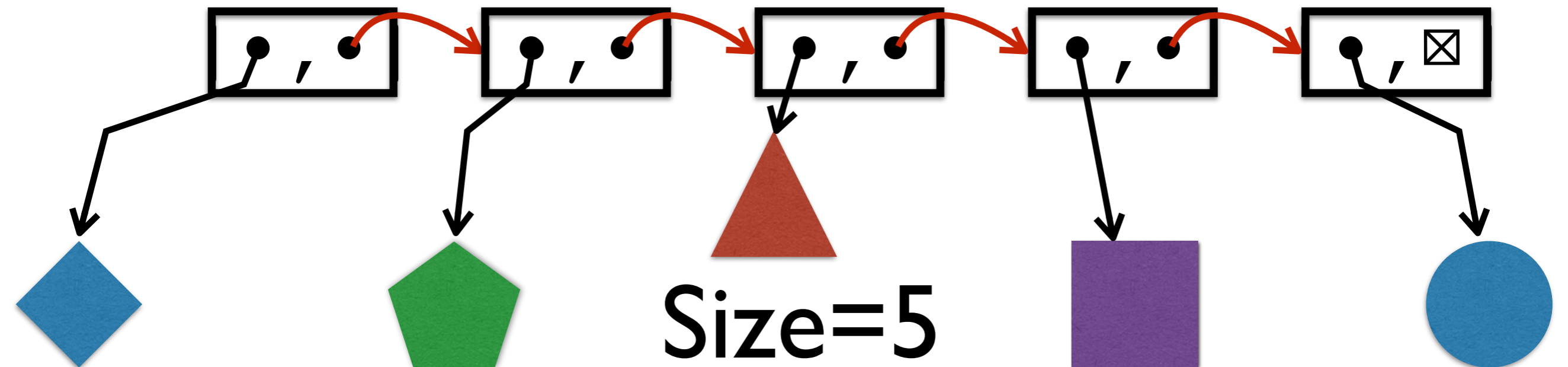


Size=5

Array of shapes:

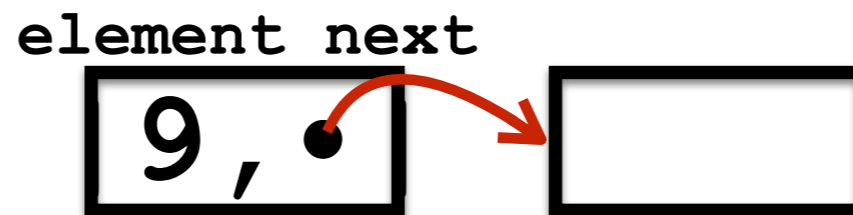


Linked list of shapes:



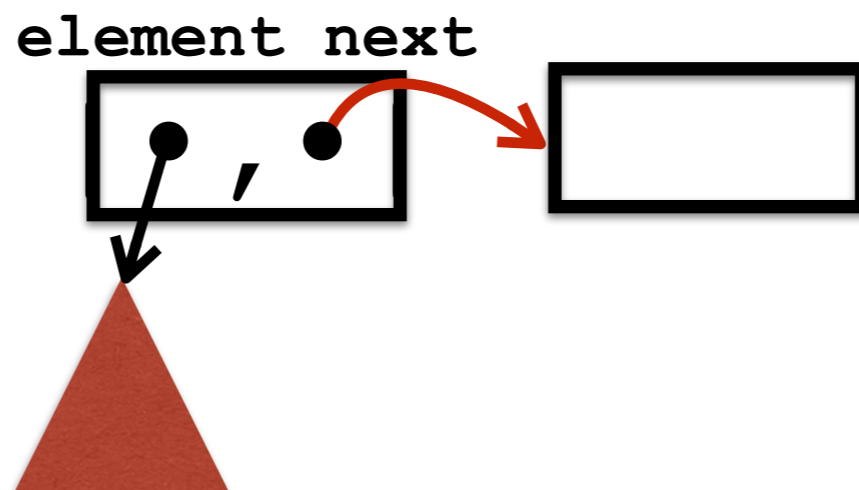
(Singly) Linked List Node

```
class SNode{  
    Type          element;  
    SNode        next;  
}
```



(Singly) Linked List Node

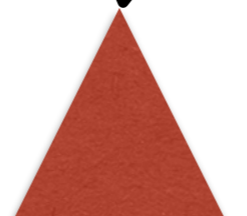
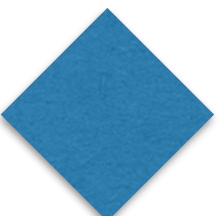
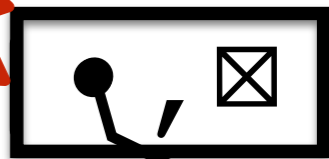
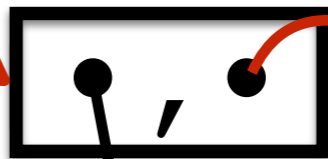
```
class SNode{  
    Type          element;  
    SNode        next;  
}
```



```
class SLinkedList{  
    SNode      head;  
    SNode      tail;  
    integer    size;  
}
```

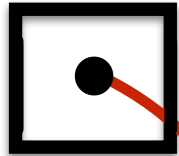
```
class SLinkedList{
    SNode    head;
    SNode    tail;
    integer  size;
}
```

head tail size

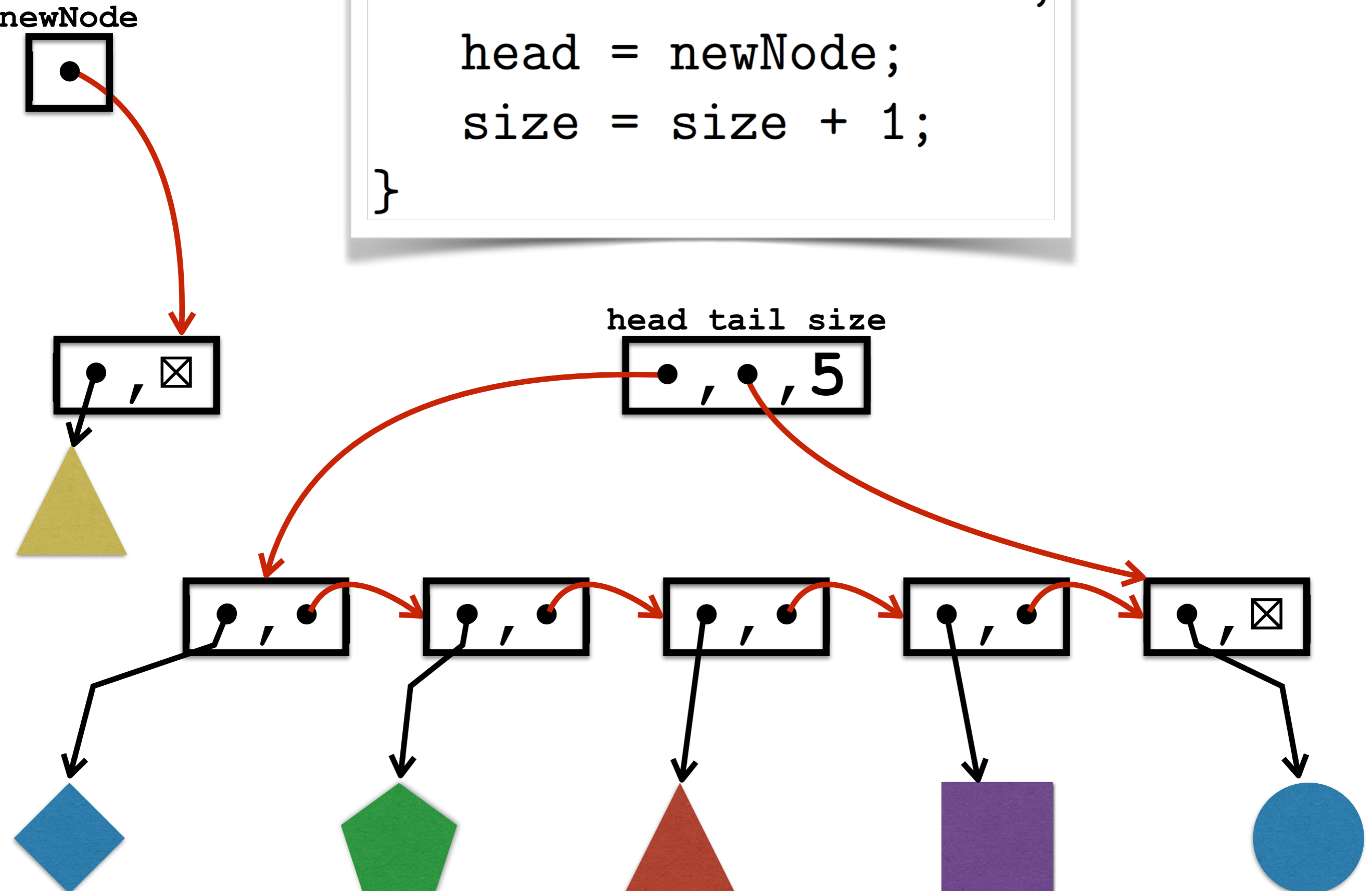
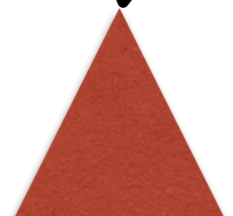
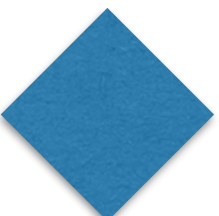
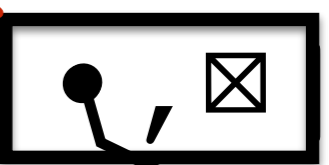
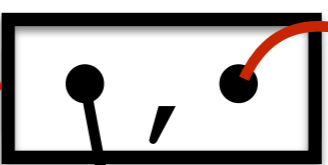
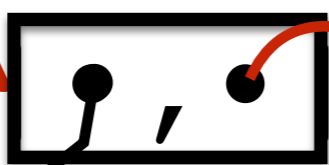
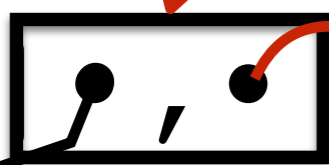
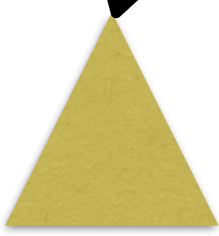


```
addFirst( newNode ){
    newNode.next = head;
    head = newNode;
    size = size + 1;
}
```

newNode

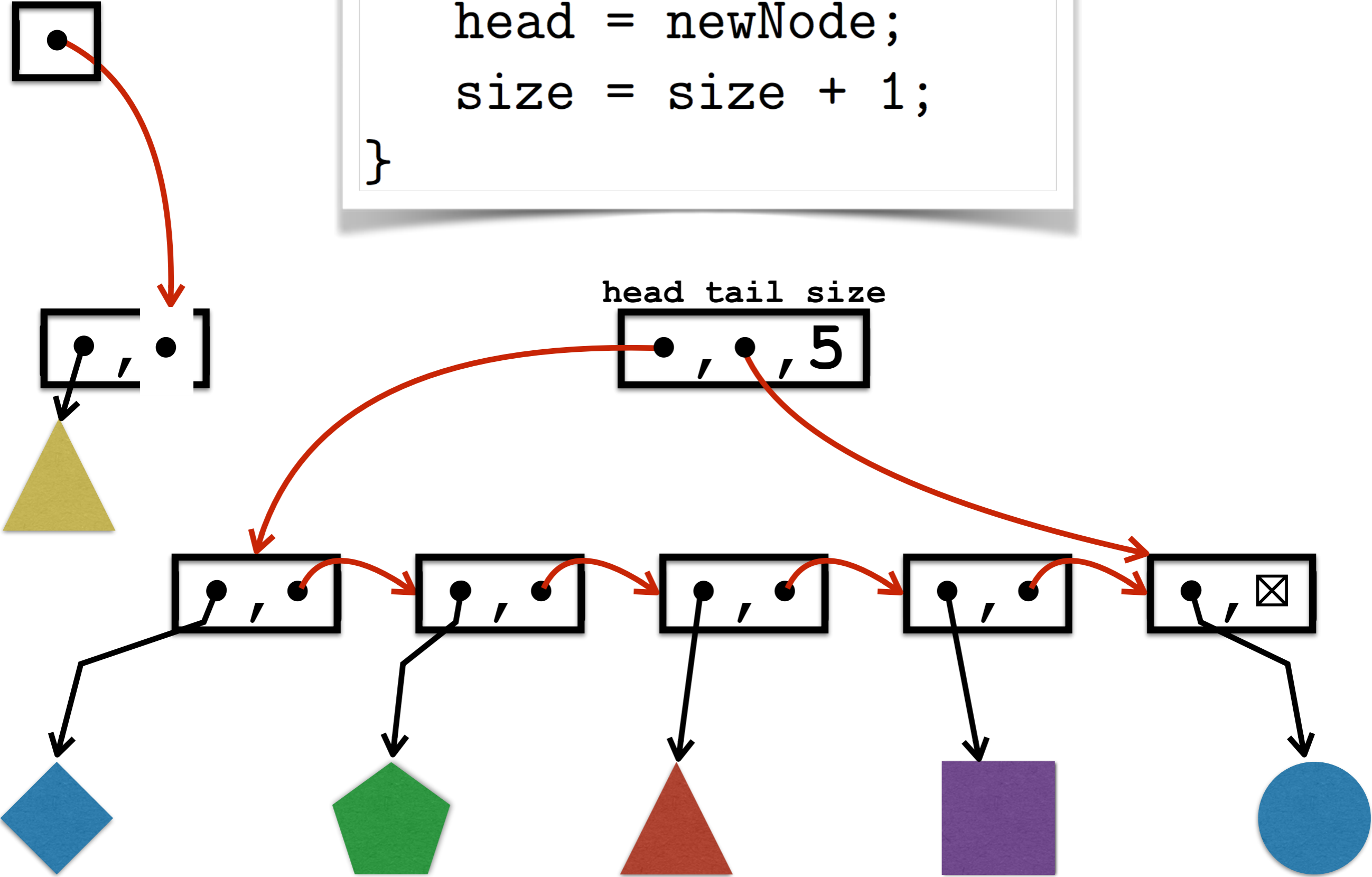


head tail size



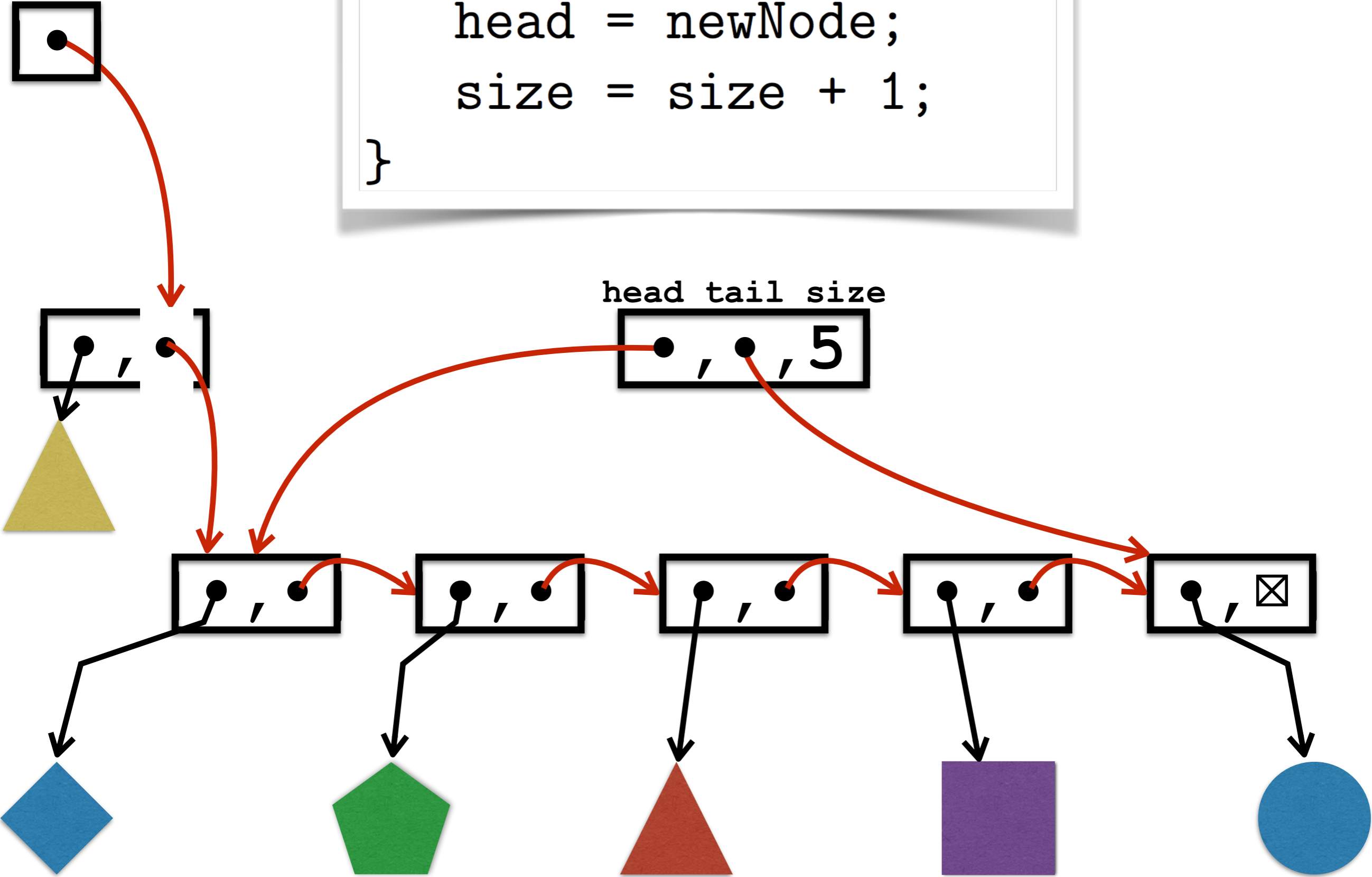
```
addFirst( newNode ){  
    newNode.next = head;  
    head = newNode;  
    size = size + 1;  
}
```

newNode



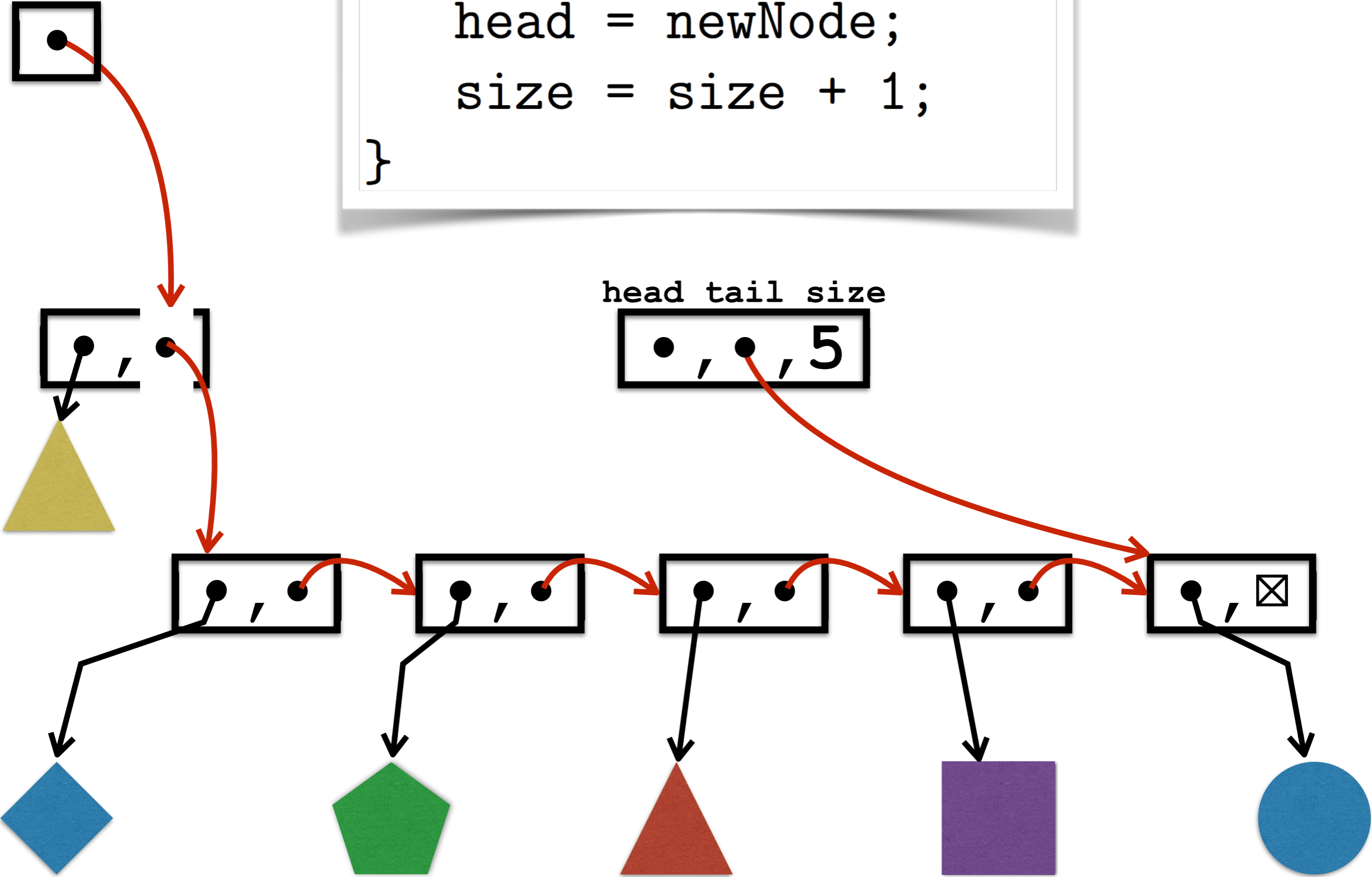
```
addFirst( newNode ){  
    newNode.next = head;  
    head = newNode;  
    size = size + 1;  
}
```

newNode



```
addFirst( newNode ){  
    newNode.next = head;  
    head = newNode;  
    size = size + 1;  
}
```

newNode

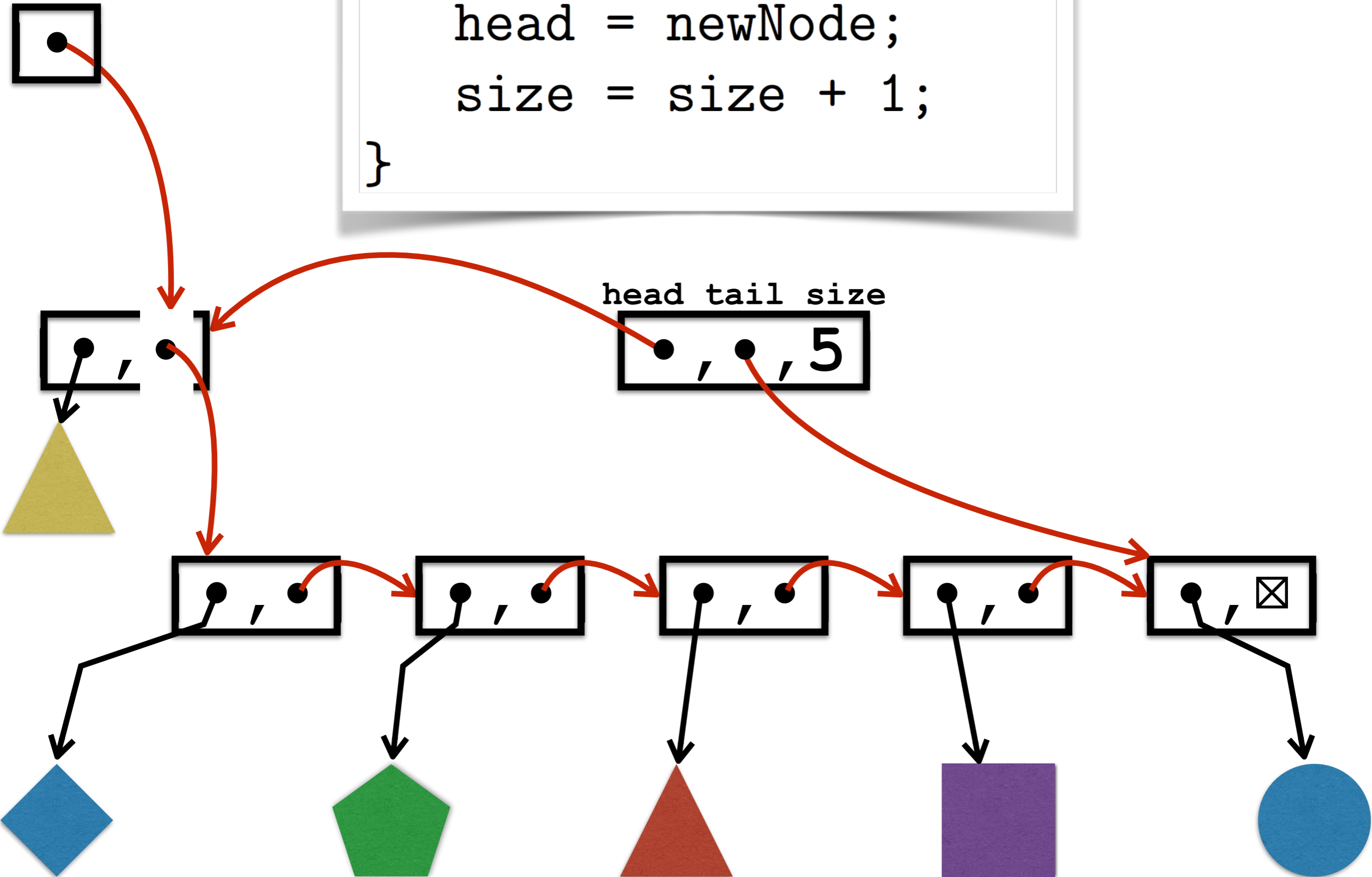


head tail size



```
addFirst( newNode ){
    newNode.next = head;
    head = newNode;
    size = size + 1;
}
```

newNode

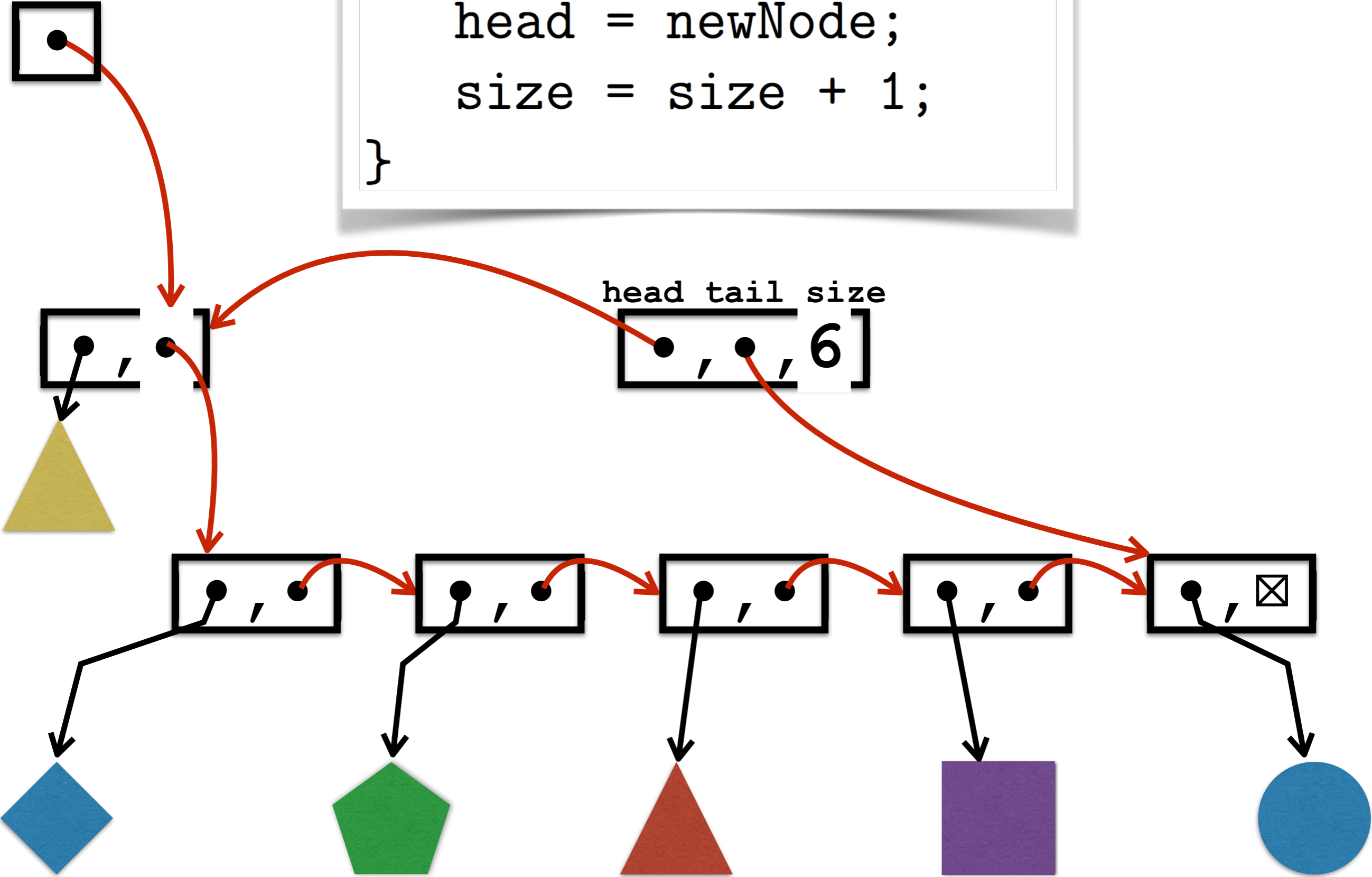


head tail size




```
addFirst( newNode ){
    newNode.next = head;
    head = newNode;
    size = size + 1;
}
```

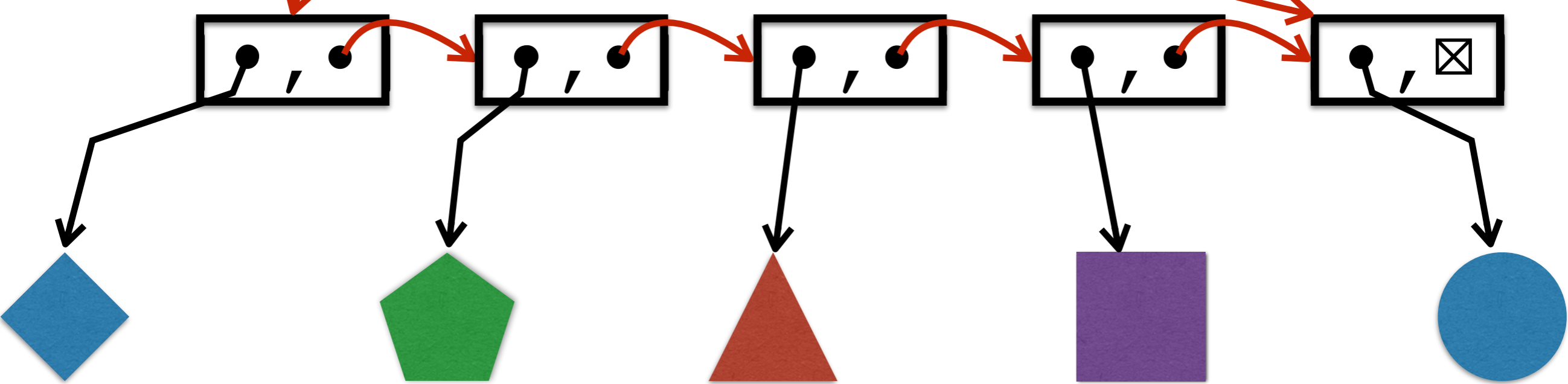
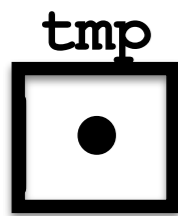
newNode



head tail size

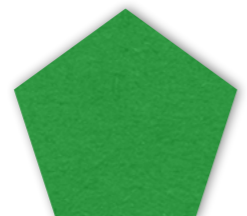
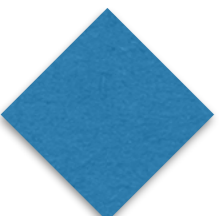
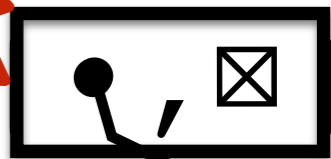
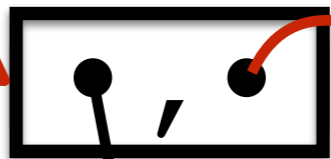
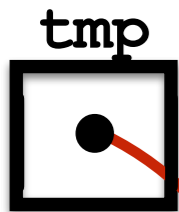


```
removeFirst(){
    tmp = head;
    head = head.next;
    tmp.next = null;
    size = size - 1;
}
```



```
removeFirst(){  
    tmp = head;  
    head = head.next;  
    tmp.next = null;  
    size = size - 1;  
}
```

head tail size

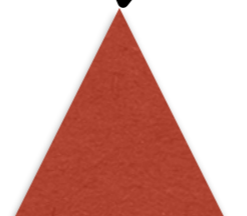
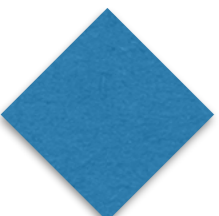
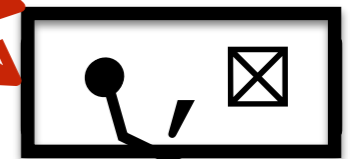
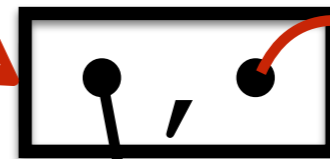
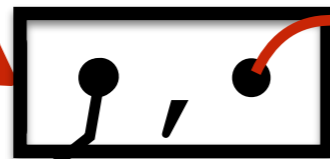
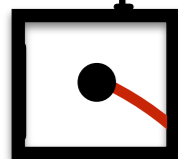


```
removeFirst(){  
    tmp = head;  
    head = head.next;  
    tmp.next = null;  
    size = size - 1;  
}
```

head tail size

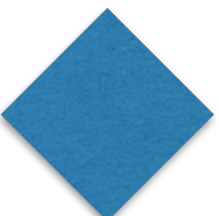
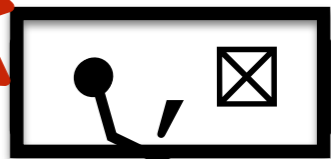
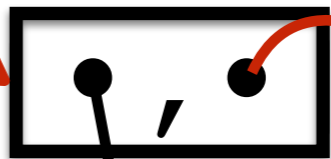
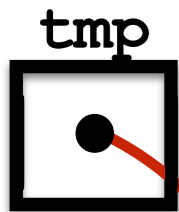
• , • , 5

tmp



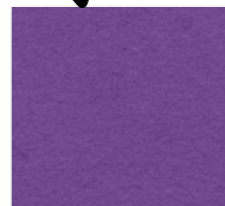
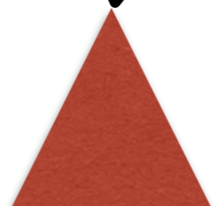
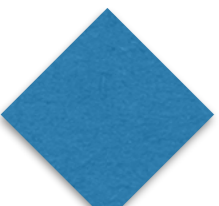
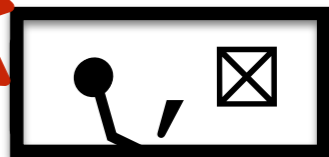
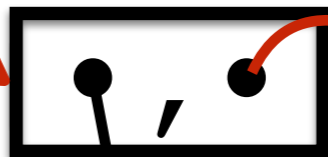
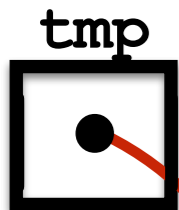
```
removeFirst(){  
    tmp = head;  
    head = head.next;  
    tmp.next = null;  
    size = size - 1;  
}
```

head tail size



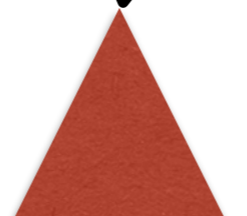
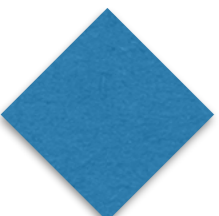
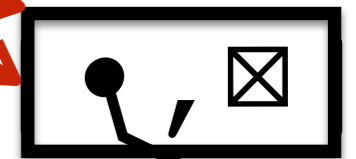
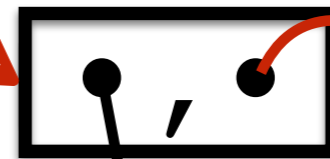
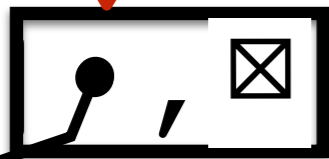
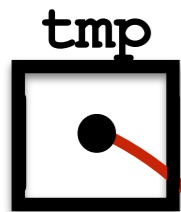
```
removeFirst(){
    tmp = head;
    head = head.next;
    tmp.next = null;
    size = size - 1;
}
```

head tail size



```
removeFirst(){
    tmp = head;
    head = head.next;
    tmp.next = null;
    size = size - 1;
}
```

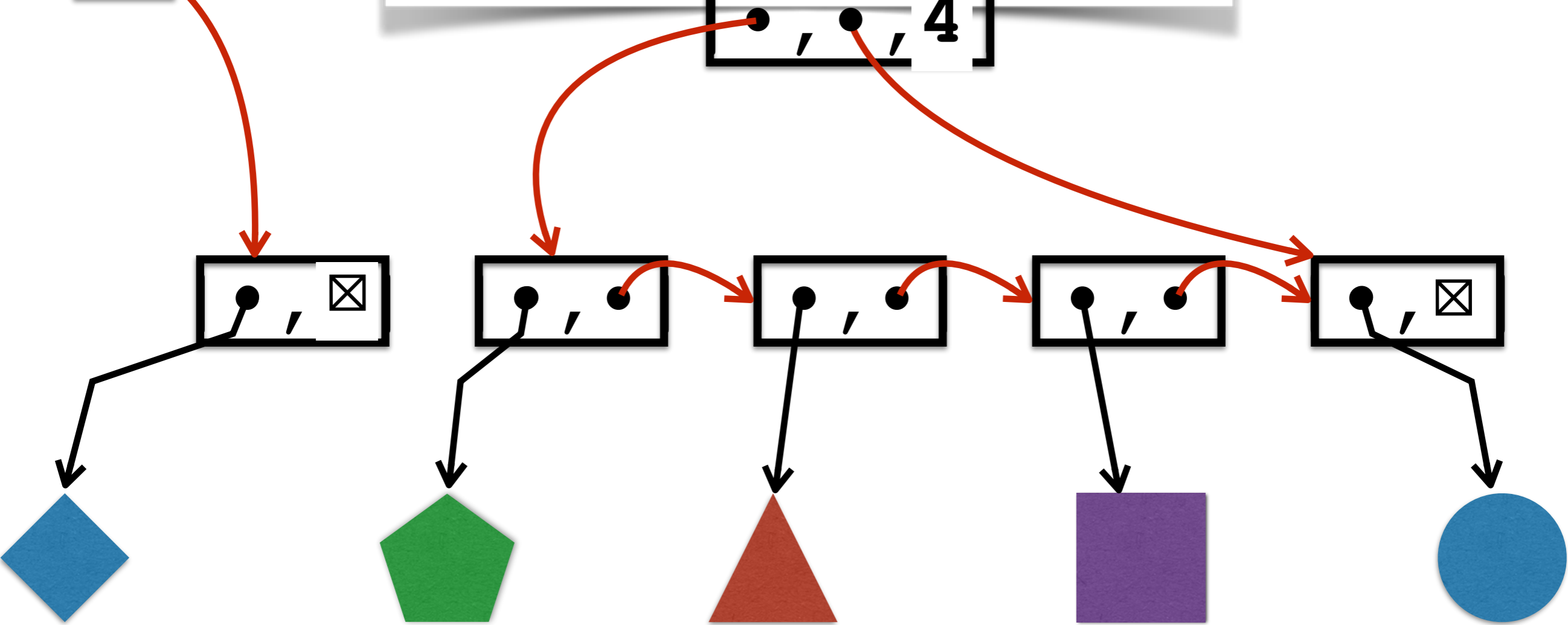
head tail size



```
removeFirst(){
    tmp = head;
    head = head.next;
    tmp.next = null;
    size = size - 1;
}
```

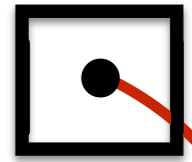
head tail size
[. , . , 4]

tmp

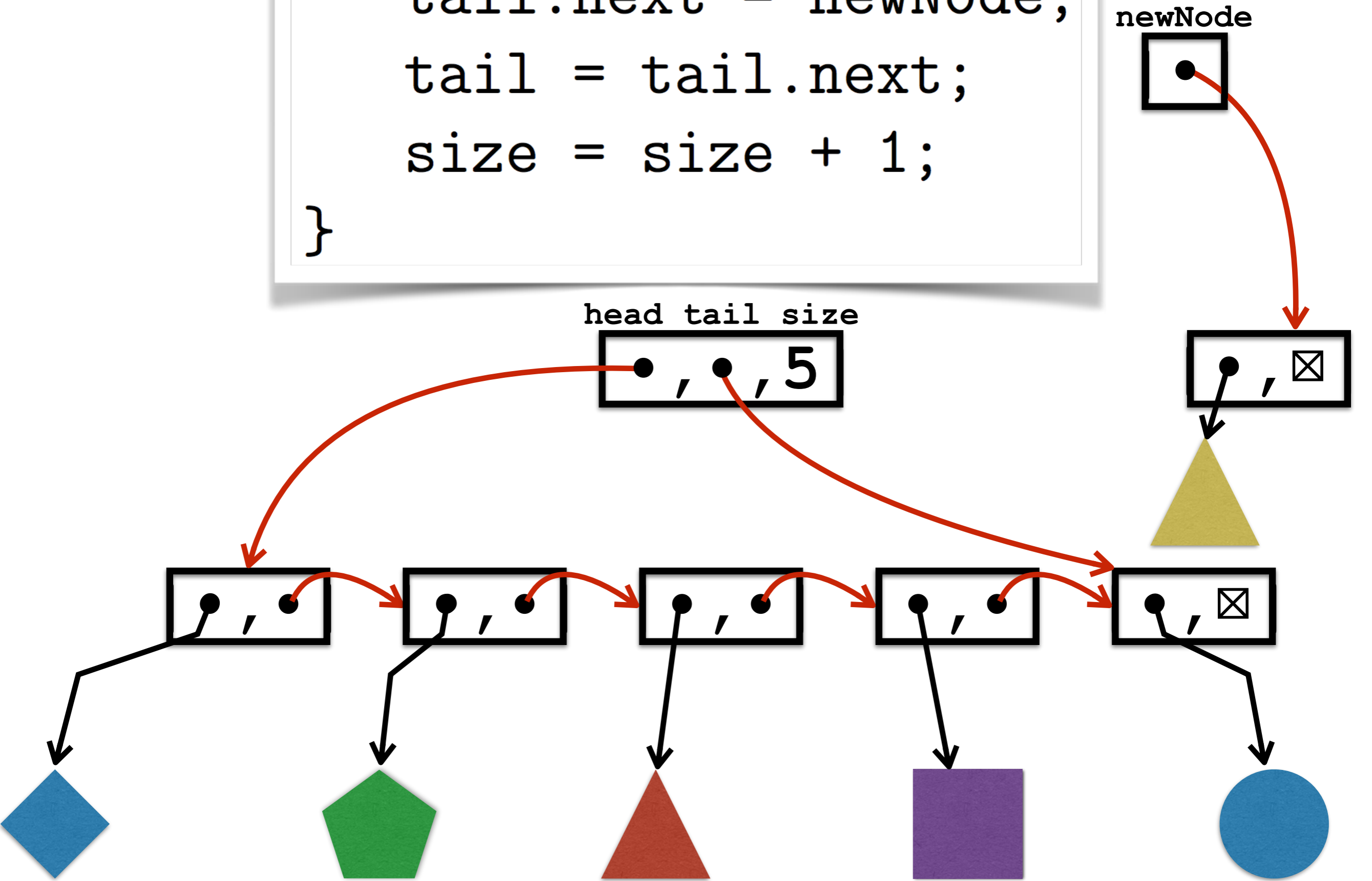
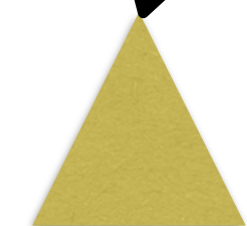
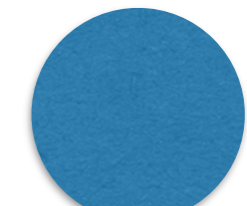
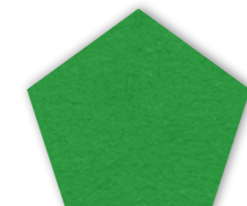
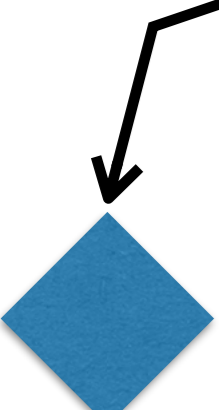
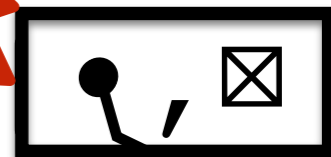
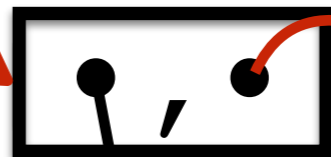



```
addLast( newNode ){  
    tail.next = newNode;  
    tail = tail.next;  
    size = size + 1;  
}
```

newNode

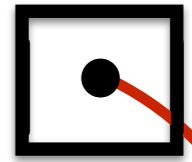


head tail size

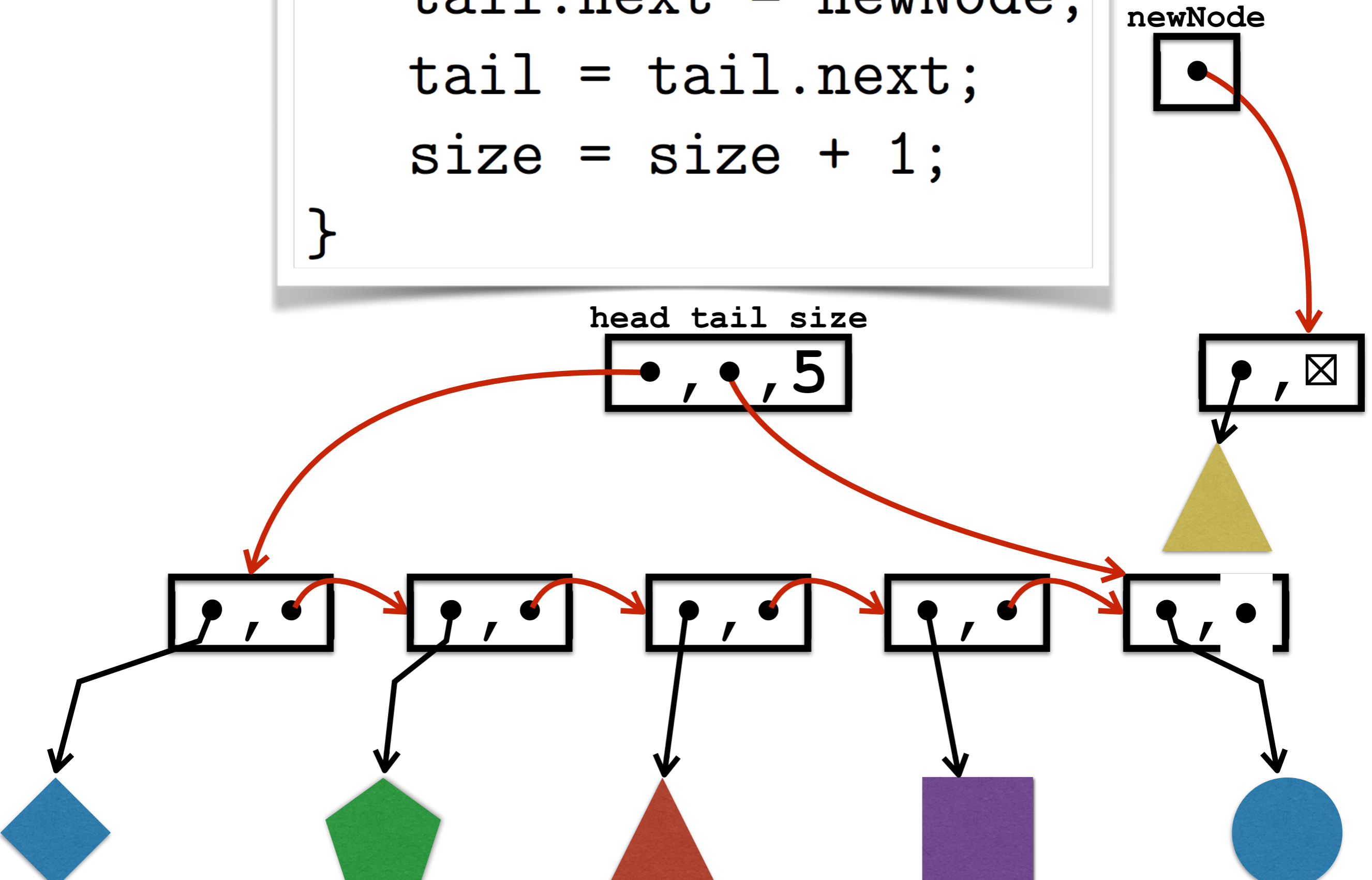
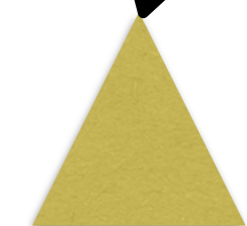
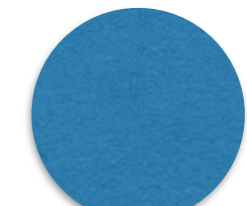
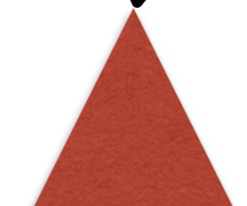
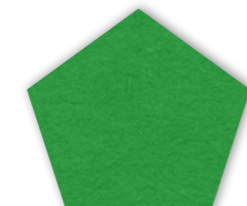
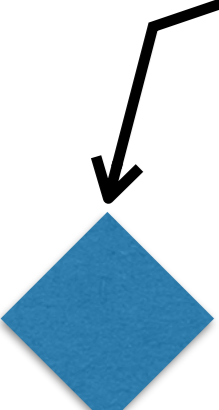
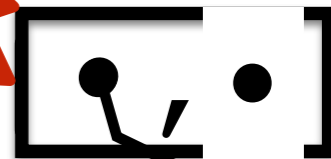
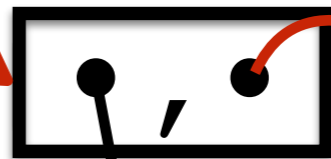


```
addLast( newNode ){  
    tail.next = newNode;  
    tail = tail.next;  
    size = size + 1;  
}
```

newNode

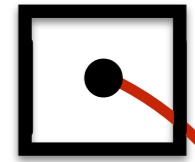


head tail size

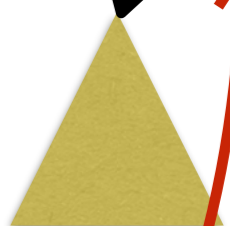
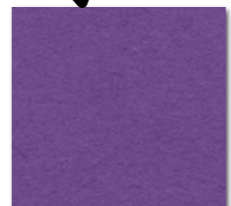
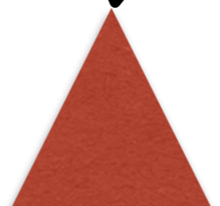
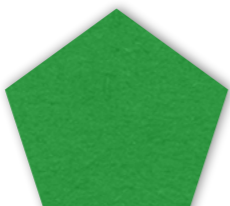
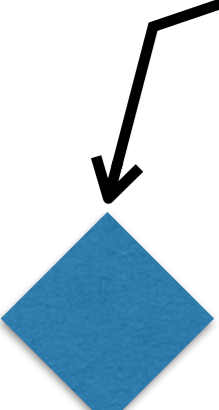
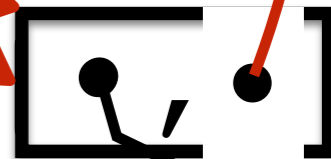
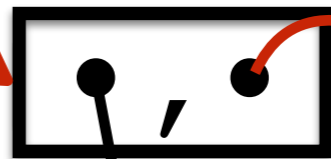
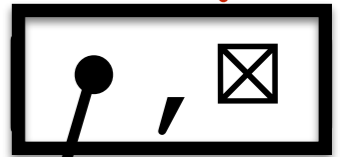


```
addLast( newNode ){
    tail.next = newNode;
    tail = tail.next;
    size = size + 1;
}
```

newNode

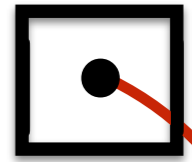


head tail size

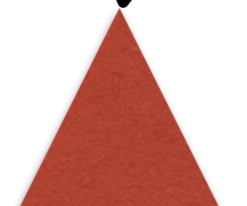
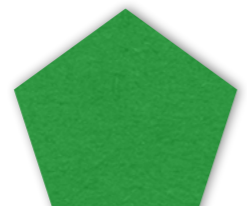
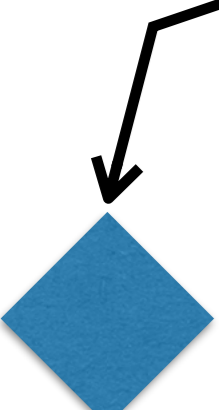
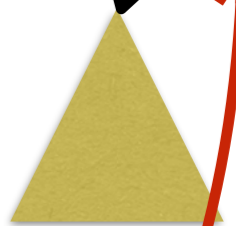
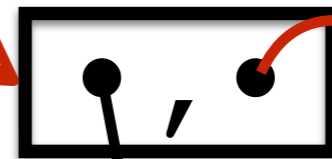
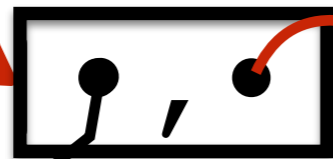


```
addLast( newNode ){  
    tail.next = newNode;  
    tail = tail.next;  
    size = size + 1;  
}
```

newNode

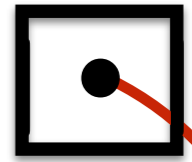


head tail size

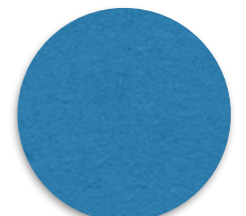
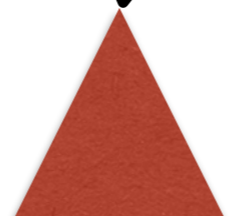
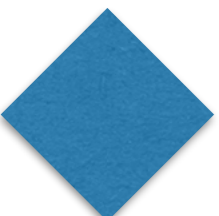
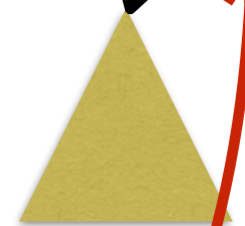
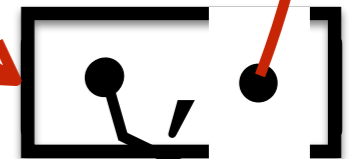
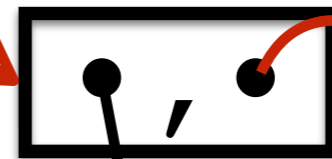
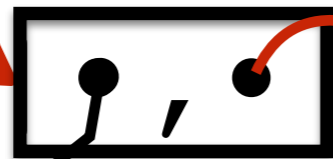


```
addLast( newNode ){
    tail.next = newNode;
    tail = tail.next;
    size = size + 1;
}
```

newNode

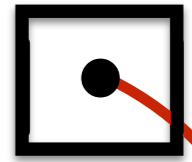


head tail size

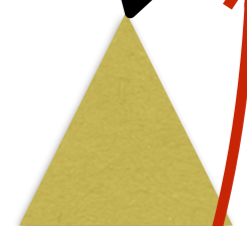
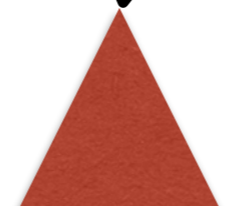
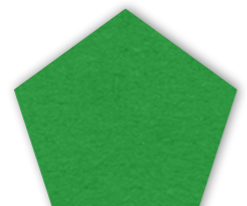
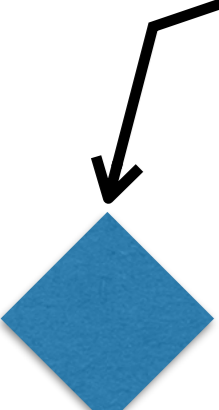
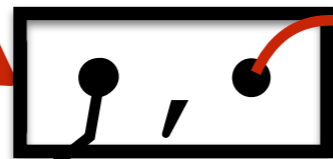
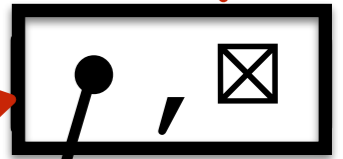
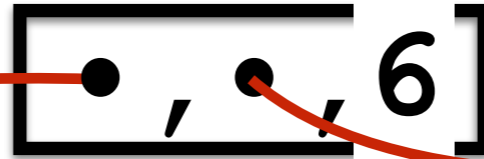


```
addLast( newNode ){  
    tail.next = newNode;  
    tail = tail.next;  
    size = size + 1;  
}
```

newNode

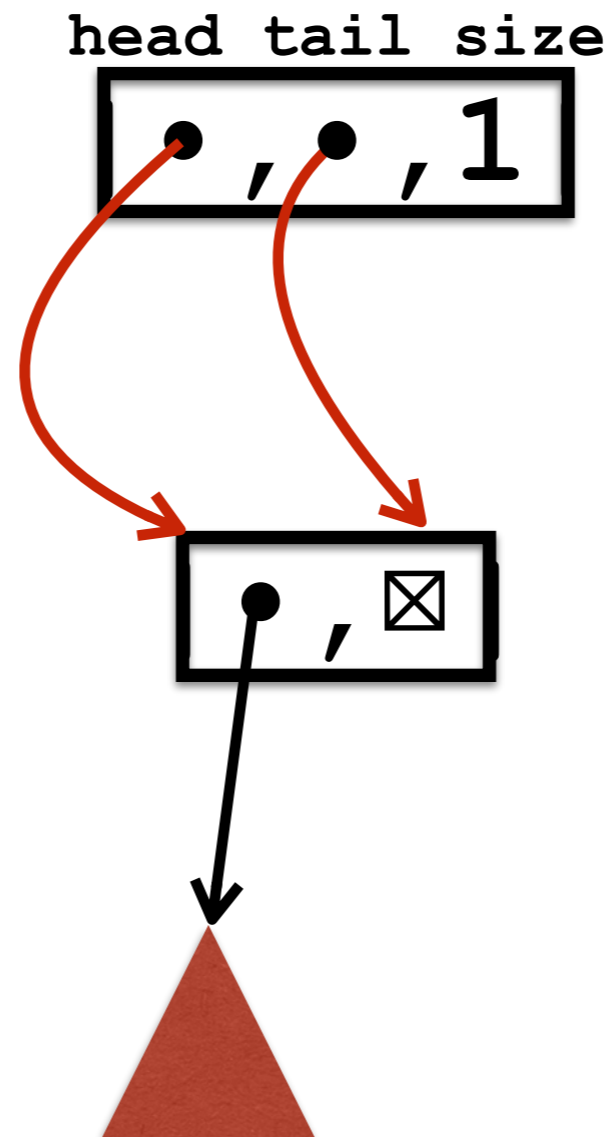


head tail size



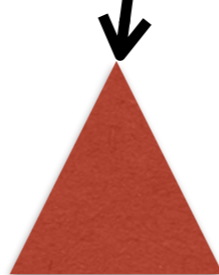
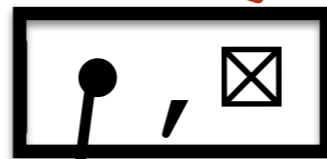
```
removeLast(){
    if (head == tail){
        head = null;
        tail = null;
        size = 0;
    }
    else{
        tmp = head;
        while (tmp.next != tail){
            tmp = tmp.next;
        }
        tmp.next = null;
        tail = tmp;
        size = size - 1;
    }
}
```

```
removeLast(){
  if (head == tail){
    head = null;
    tail = null;
    size = 0;
  }
}
```



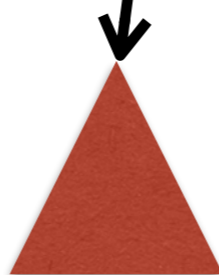
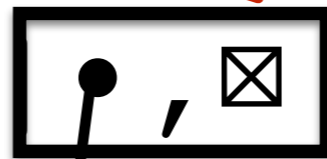

```
removeLast(){  
    if (head == tail){  
        head = null;  
        tail = null;  
        size = 0;  
    }  
}
```

head tail size



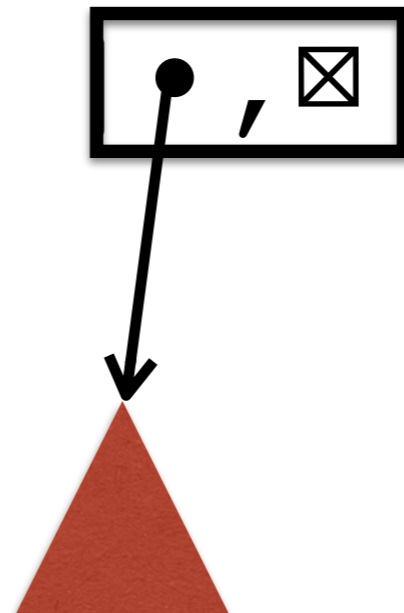
```
removeLast(){  
    if (head == tail){  
        head = null;  
        tail = null;  
        size = 0;  
    }  
}
```

head tail size



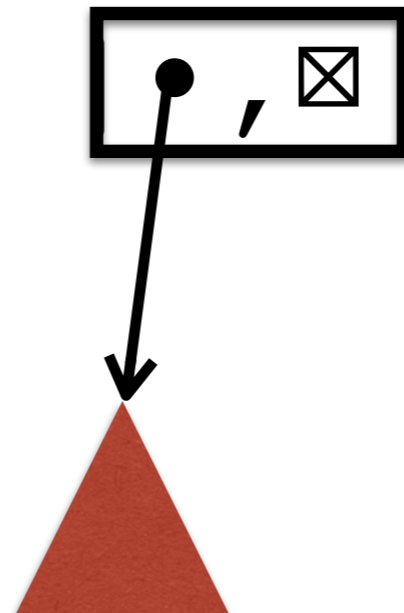
```
removeLast(){  
    if (head == tail){  
        head = null;  
        tail = null;  
        size = 0;  
    }  
}
```

head tail size



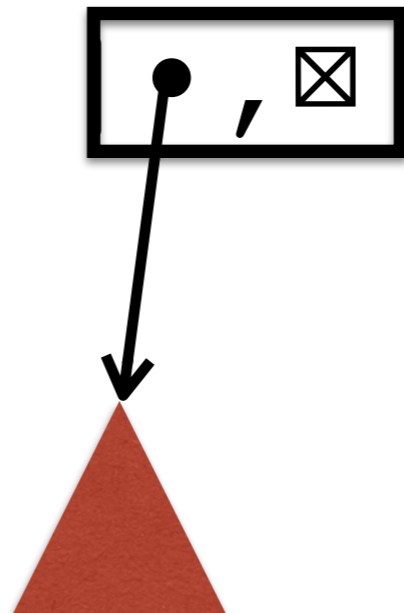
```
removeLast(){
    if (head == tail){
        head = null;
        tail = null;
        size = 0;
    }
}
```

head tail size



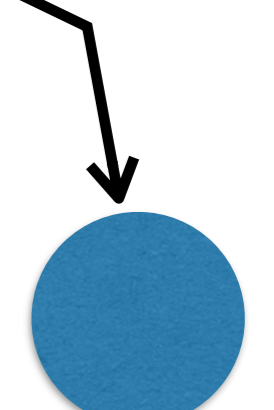
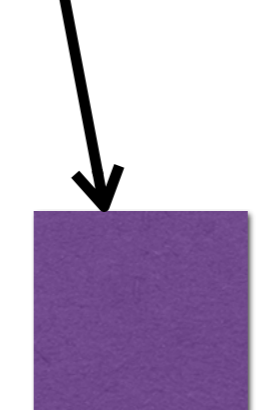
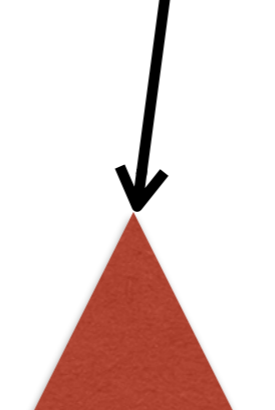
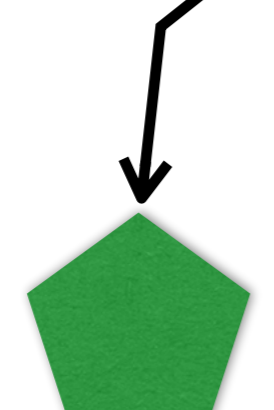
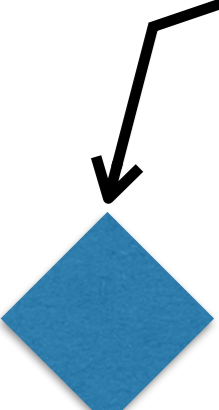
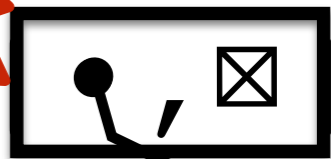
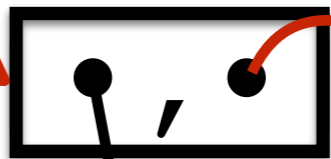
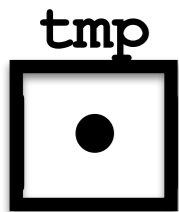
```
removeLast(){
    if (head == tail){
        head = null;
        tail = null;
        size = 0;
    }
}
```

head tail size
[⊗ , ⊗ , 0]



```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

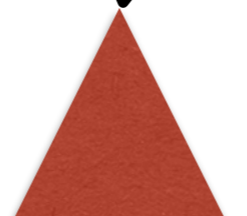
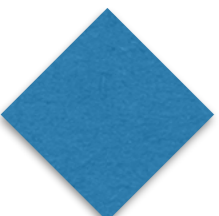
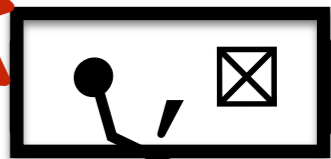
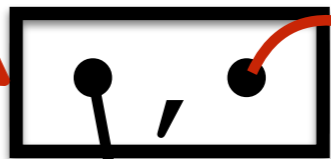
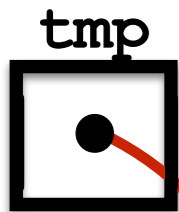
head tail size



```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

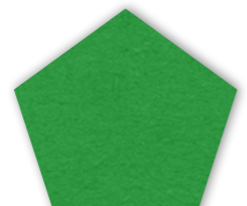
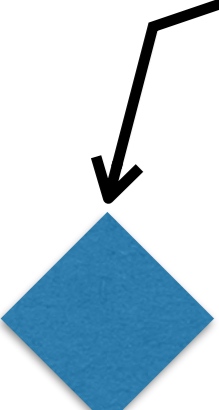
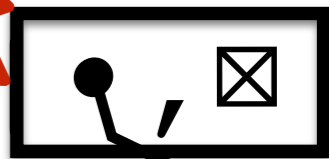
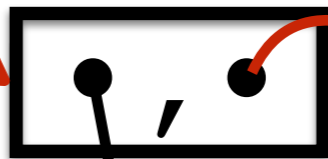
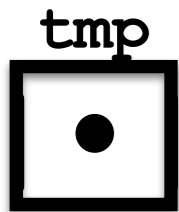
head tail size

• , • , 5



```
else{
    tmp = head;
    while (tmp.next != tail){
        tmp = tmp.next;
    }
    tmp.next = null;
    tail = tmp;
    size = size - 1;
}
```

head tail size

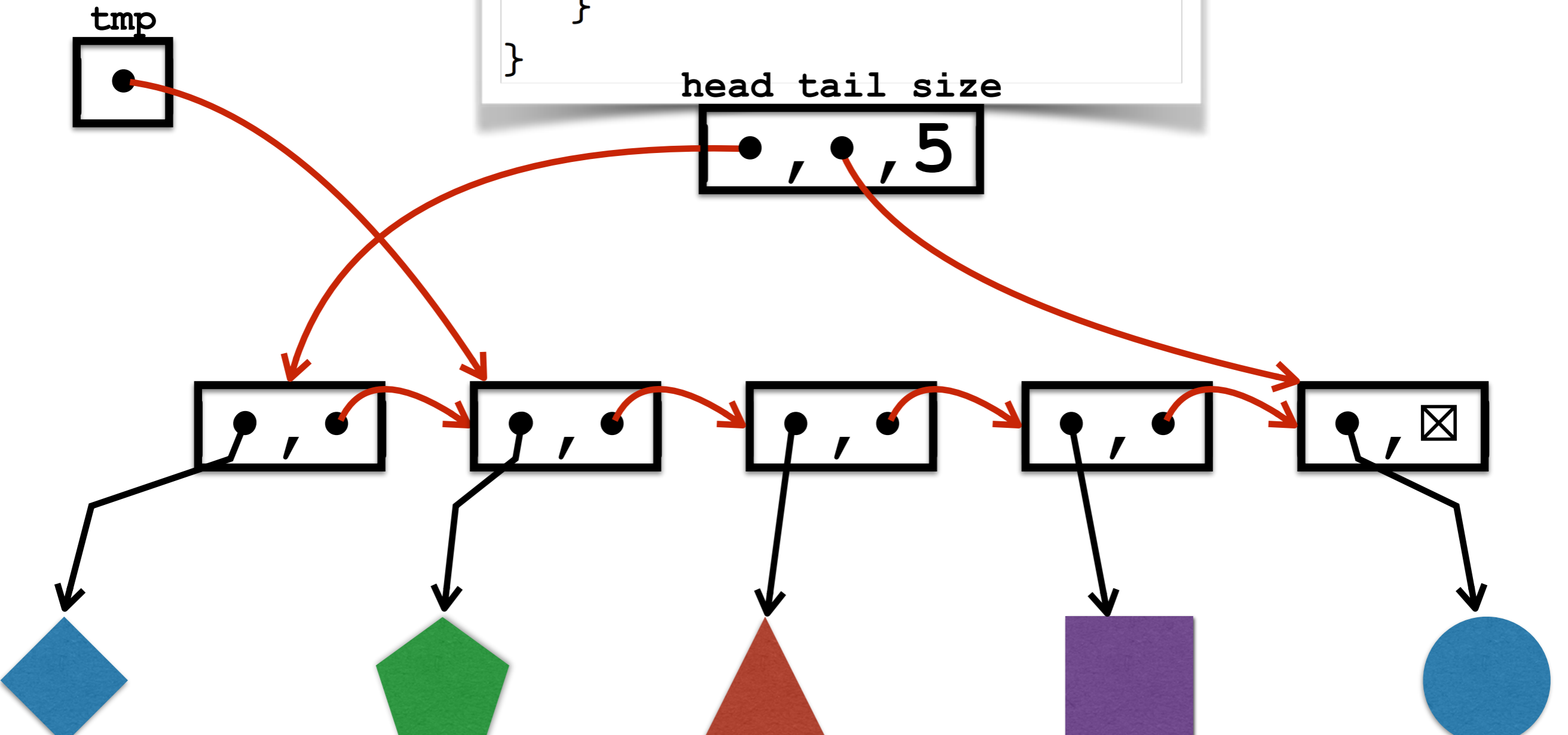
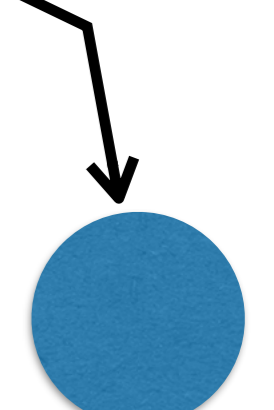
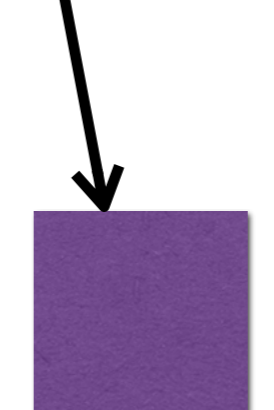
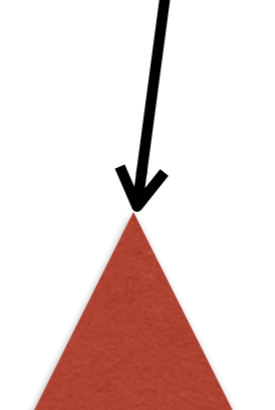
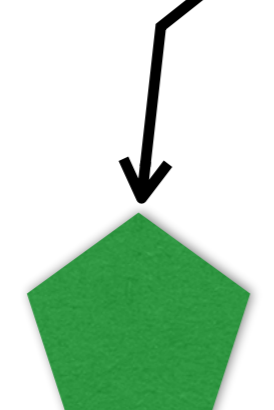
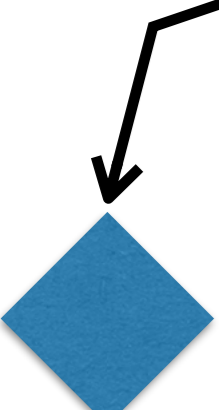
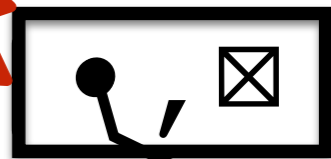
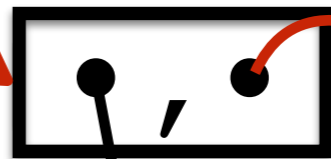
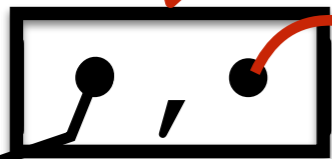
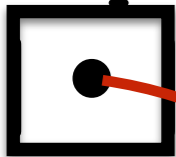



```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

head tail size

• , • , 5

tmp

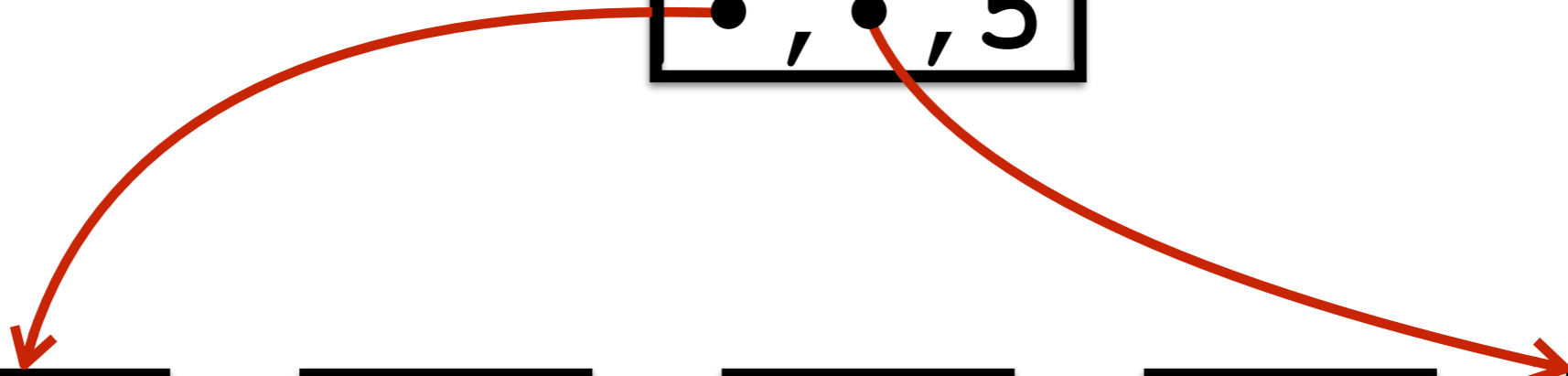
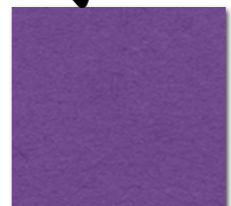
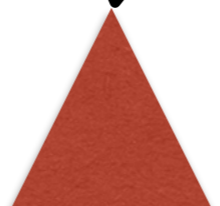
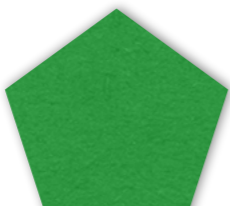
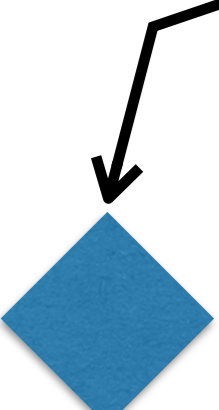
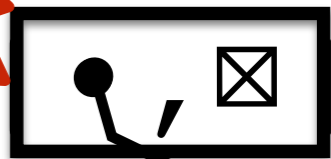
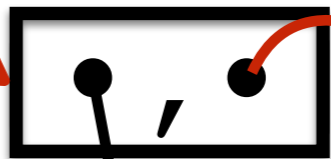
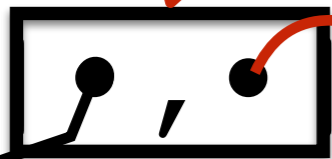
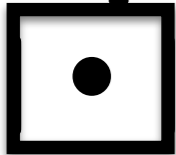


```
else{
    tmp = head;
    while (tmp.next != tail){
        tmp = tmp.next;
    }
    tmp.next = null;
    tail = tmp;
    size = size - 1;
}
```

head tail size

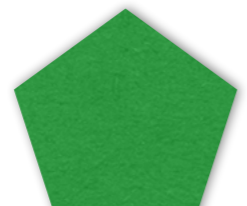
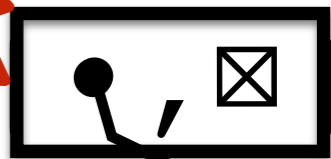
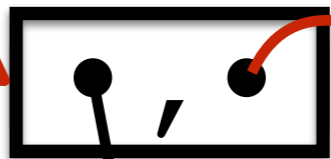
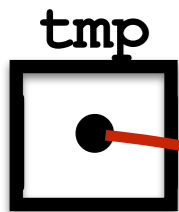
• , • , 5

tmp



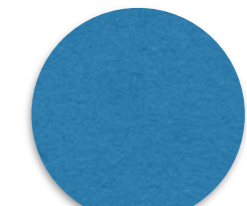
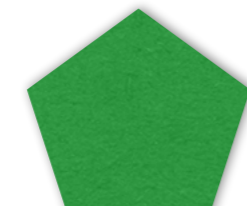
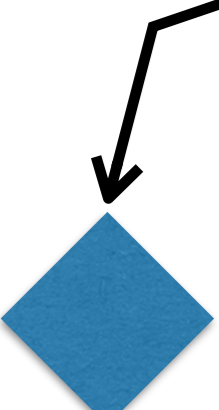
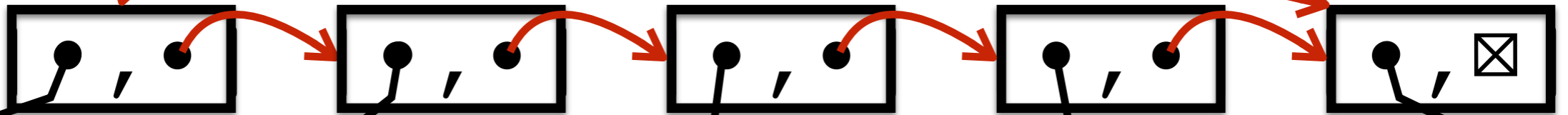
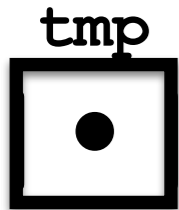
```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

head tail size



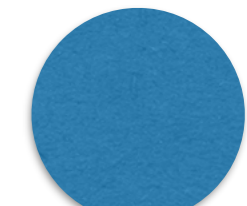
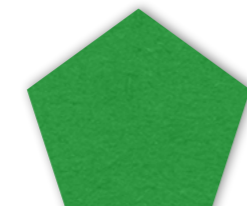
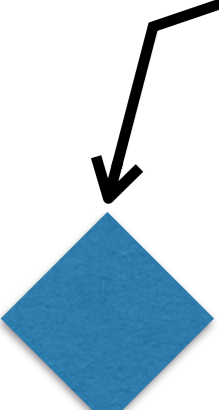
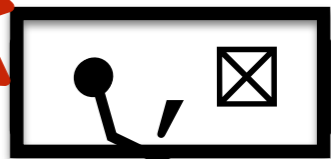
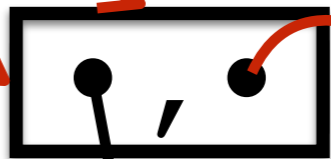
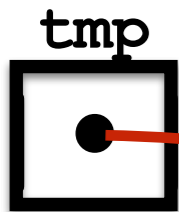
```
else{
    tmp = head;
    while (tmp.next != tail){
        tmp = tmp.next;
    }
    tmp.next = null;
    tail = tmp;
    size = size - 1;
}
```

head tail size



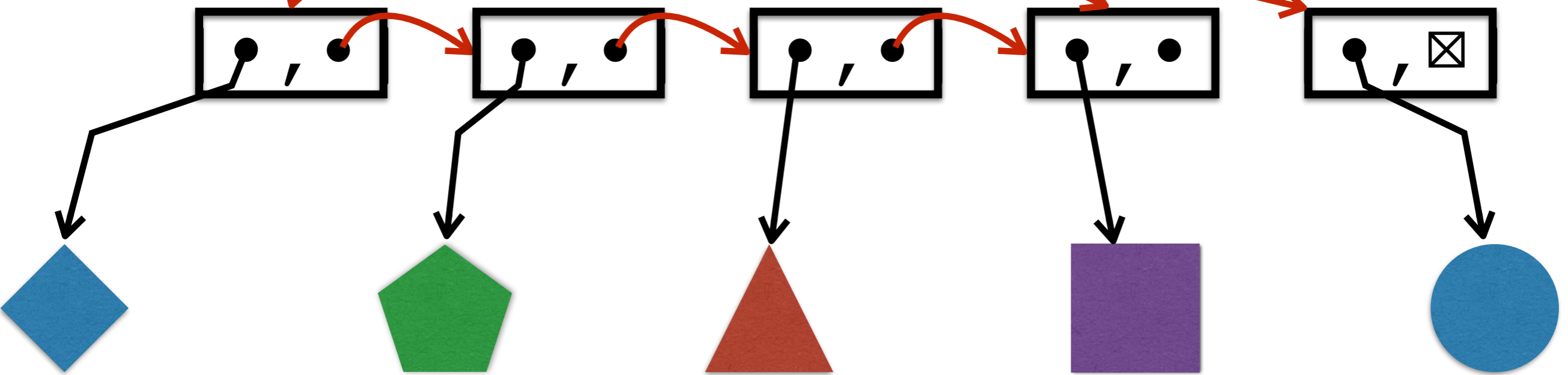
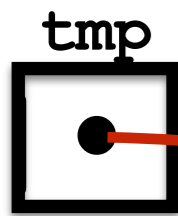
```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

head tail size



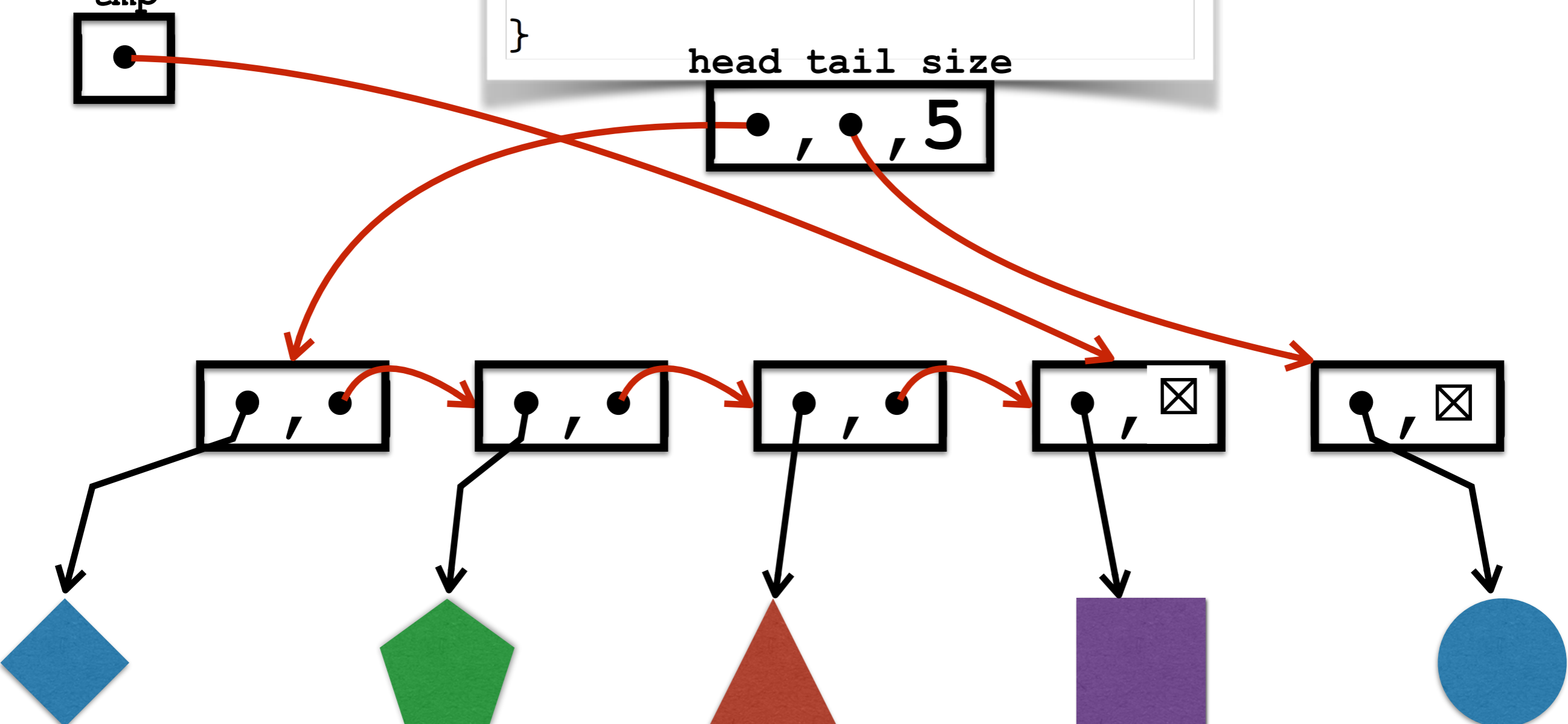
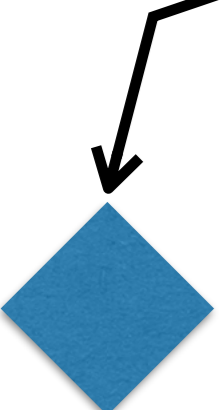
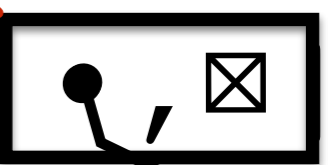
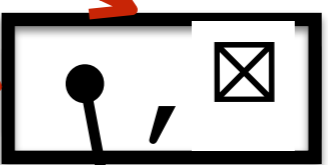
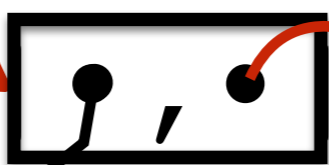
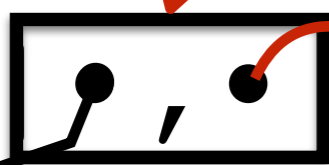
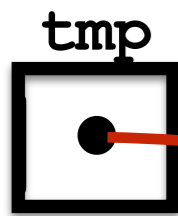
```
else{
    tmp = head;
    while (tmp.next != tail){
        tmp = tmp.next;
    }
    tmp.next = null;
    tail = tmp;
    size = size - 1;
}
```

head tail size



```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

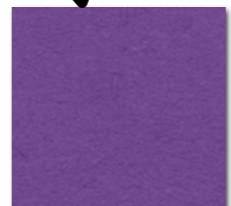
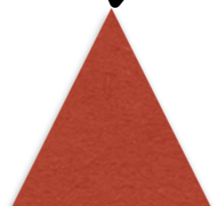
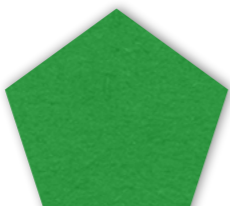
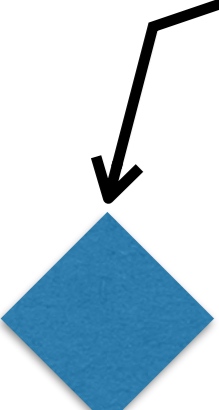
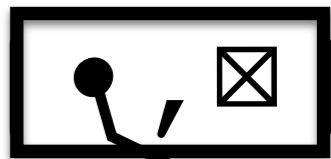
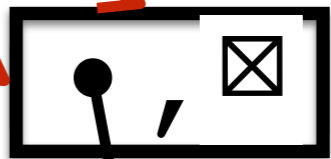
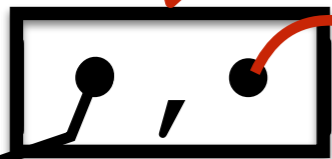
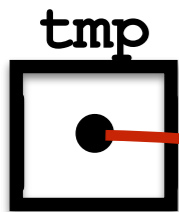
head tail size



```
else{
    tmp = head;
    while (tmp.next != tail){
        tmp = tmp.next;
    }
    tmp.next = null;
    tail = tmp;
    size = size - 1;
}
```

head tail size

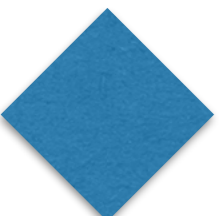
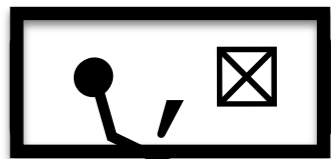
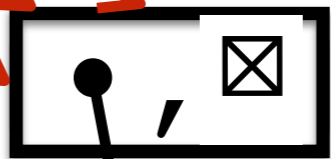
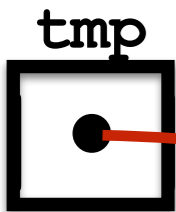
• , • , 5




```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

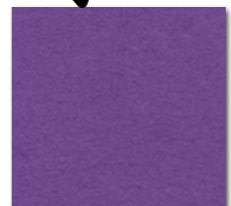
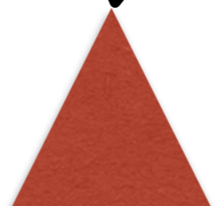
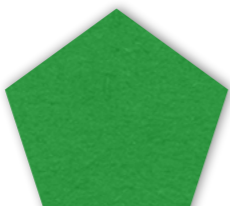
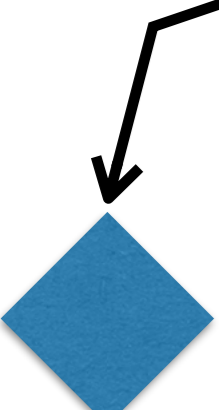
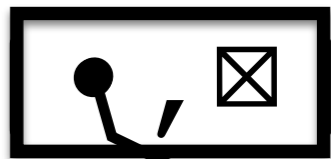
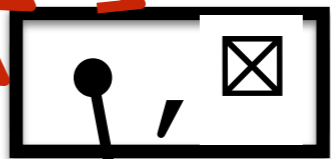
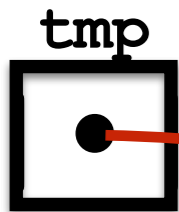
head tail size

• , • , 5



```
else{
  tmp = head;
  while (tmp.next != tail){
    tmp = tmp.next;
  }
  tmp.next = null;
  tail = tmp;
  size = size - 1;
}
```

head tail size



Winter 2016
COMP-250: Introduction
to Computer Science

Lecture 5, January 26, 2016