CHAPTER 4

LISTS

All the programs in this file are selected from

Ellis Horowitz, Sartaj Sahni, and Susan Anderson-Freed
“Fundamentals of Data Structures in C”,

Introduction

- Array

  successive items locate a fixed distance
- disadvantage
  - data movements during insertion and deletion
  - waste space in storing *n* ordered lists of varying size
- possible solution

  Linked List

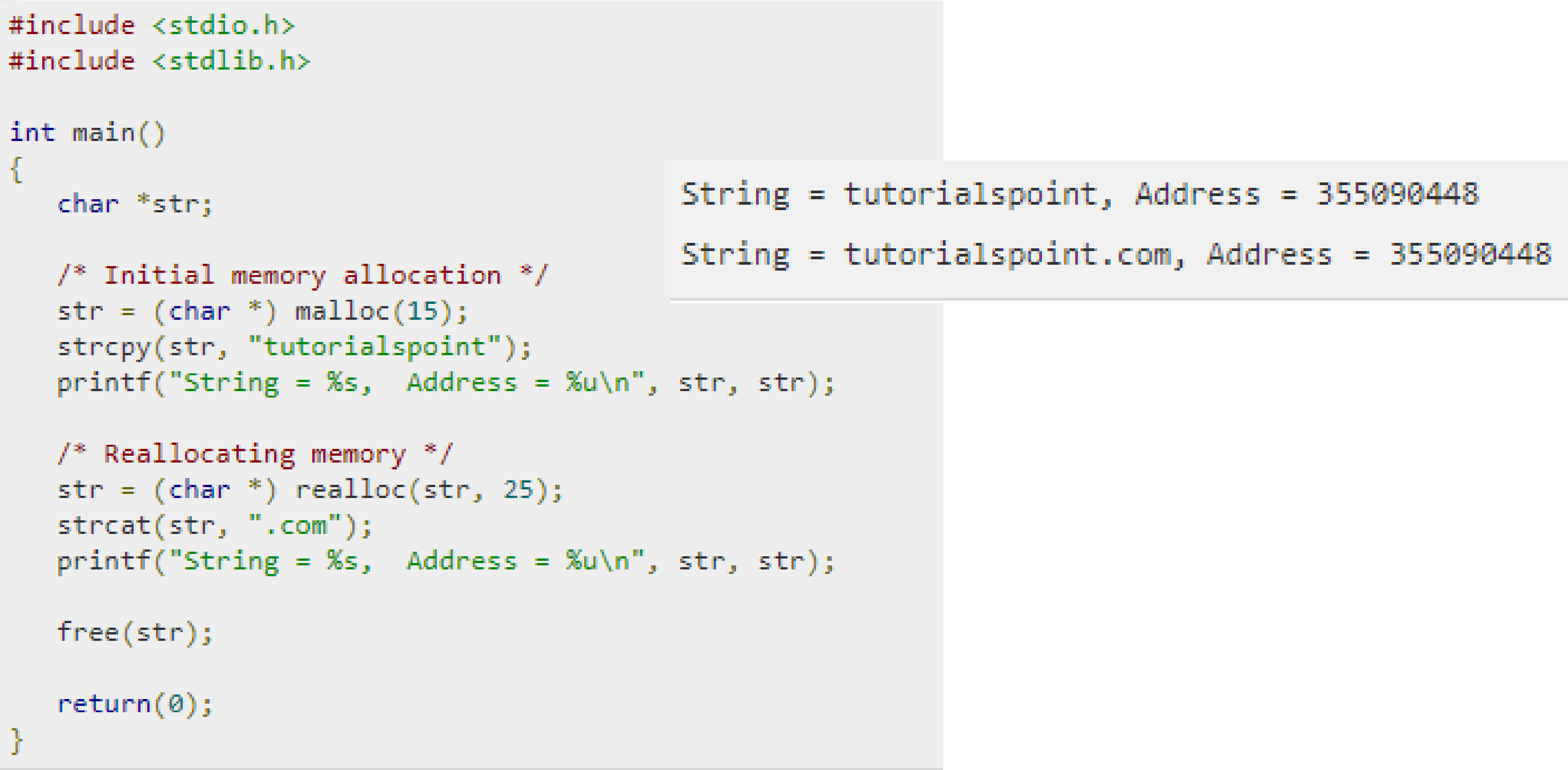
Pointer

pointer

```
  int i, *pi;
  pi= (int *) malloc(sizeof(int));
    /* assign to pi a pointer to int */
  pi = &i;
  i=10;    *pi=10
  pf=(float *) pi;
   /* coverts an int pointer to a float pointer */
```

malloc()

- The C library function **void *malloc(size_t size)** allocates the requested memory and returns a pointer to it.



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
   char *str;

   /* Initial memory allocation */
   str = (char *) malloc(15);
   strcpy(str, "tutorialspoint");
   printf("String = %s,  Address = %u\n", str, str);

   /* Reallocating memory */
   str = (char *) realloc(str, 25);
   strcat(str, ".com");
   printf("String = %s,  Address = %u\n", str, str);

   free(str);

   return(0);
}
```

```
String = tutorialspoint, Address = 355090448

String = tutorialspoint.com, Address = 355090448
```

Using Dynamically Allocated Storage

```
int i, *pi;
float f, *pf;
pi = (int *) malloc(sizeof(int));          →    request memory
pf = (float *) malloc (sizeof(float));
*pi =1024;
*pf =3.14;
printf(”an integer = %d, a float = %f\n”, *pi, *pf);
free(pi);
                     →    return memory
free(pf);
```

Singly Linked Lists

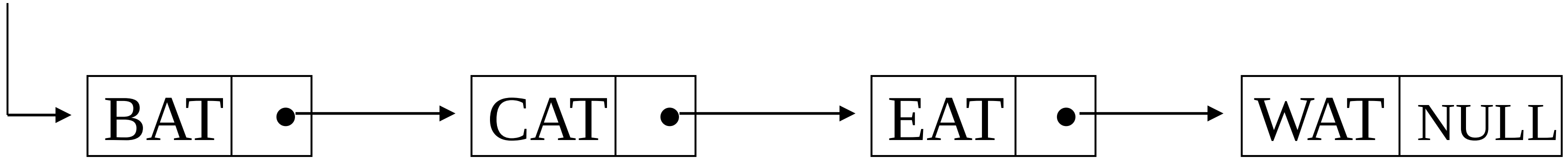


***Figure 4.2:** Usual way to draw a linked list*

Insert

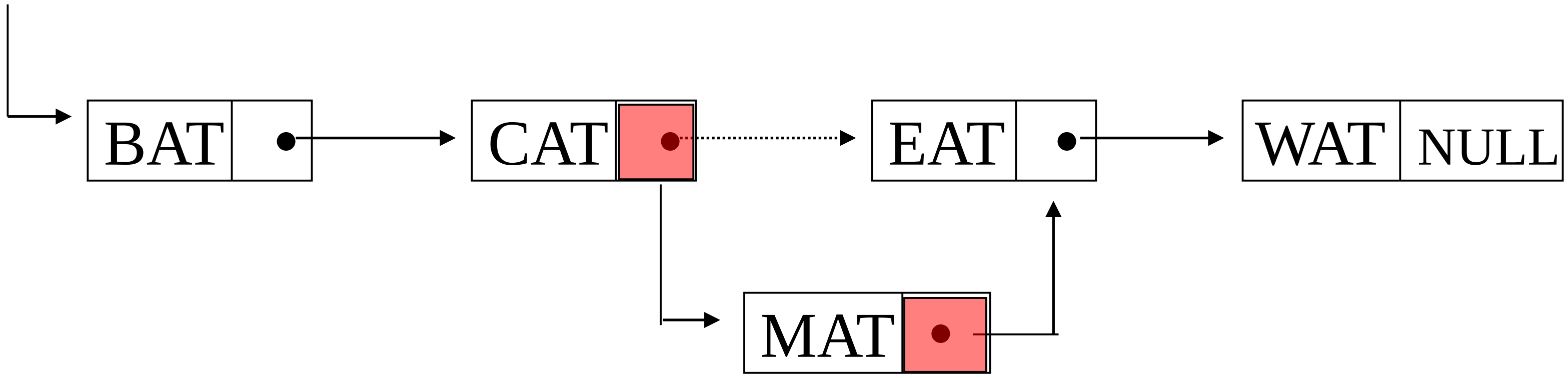
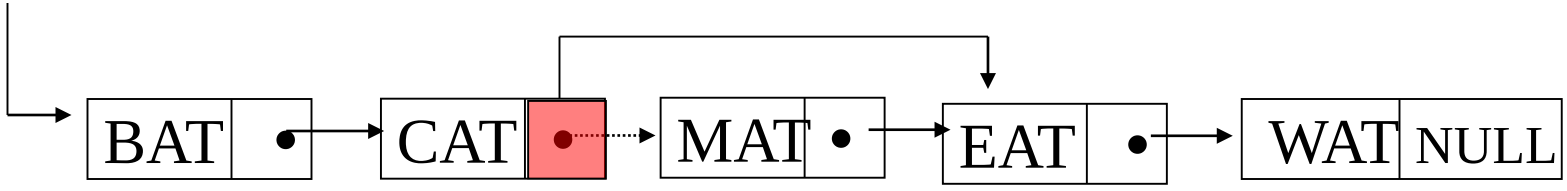


Figure 4.3: Insert MAT after CAT

Delete



dangling reference

***Figure 4.4:** Delete MAT from list*

Example 4.1: Create a linked list of words

Declaration

```
typedef struct list_node *list_pointer;
typedef struct list_node {
        char data [4];
        list_pointer link;
        };
```

Creation

```
list_pointer first =NULL;
```

Testing

```
#define IS_EMPTY(first) (!(first))
```

Allocation

```
first=(list_pointer) malloc (sizeof(list_node));
```

```
strcpy(first -> data, “BAT”);
first -> link = NULL;
```

first -> data ⇨ (*first).data

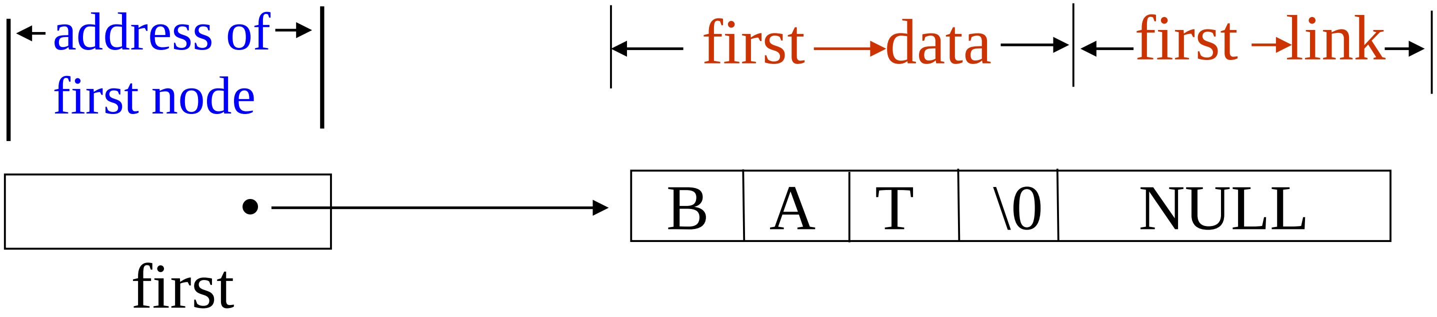


***Figure 4.5:**Referencing the fields of a node

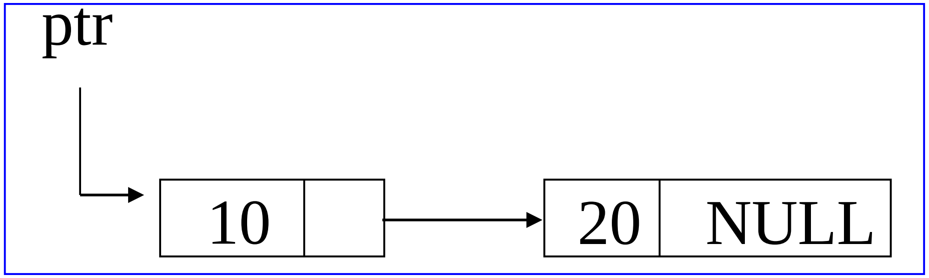
Create a linked list pointer

```
typedef struct list_node *list_pointer;
typedef struct list_node {
            int data;
            list_pointer link;
            };
list_pointer ptr =NULL
```

ptr → NULL

Create a two-node list

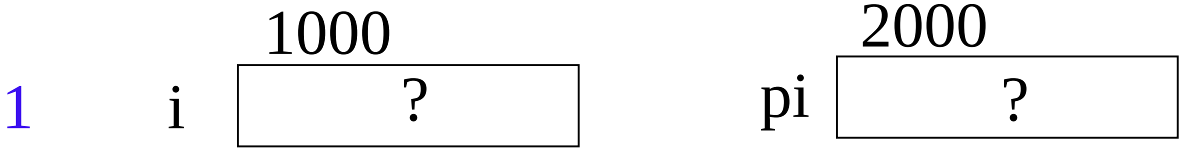
```
list_pointer create2( )
{
/* create a linked list with two nodes */
    list_pointer first, second;
    first = (list_pointer) malloc(sizeof(list_node));
    second = ( list_pointer) malloc(sizeof(list_node));
    second -> link = NULL;
    second -> data = 20;
    first -> data = 10;
    first ->link = second;
    return first;
}
```



***Program 4.1:**Create a two-node list*

Pointer Review (1)

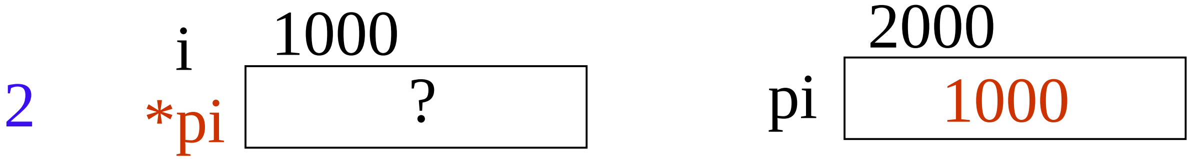
int i, *pi;



1000 2000

1 i ? pi ?

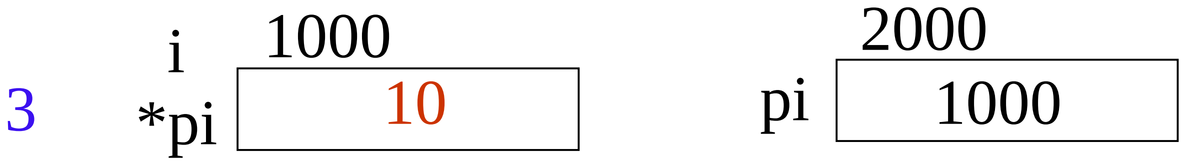
pi = &i;



i 1000 2000

2 *pi ? pi 1000

i = 10 or *pi = 10

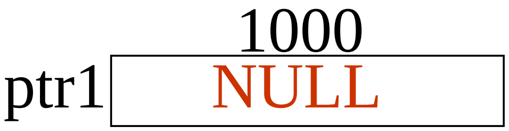


i 1000 2000

3 *pi 10 pi 1000

Pointer Review (2)

```
typedef struct list_node *list_pointer;
typedef struct list_node {
                    int data;
                    list_pointer link;
                  }
list_pointer ptr1 = NULL;
```

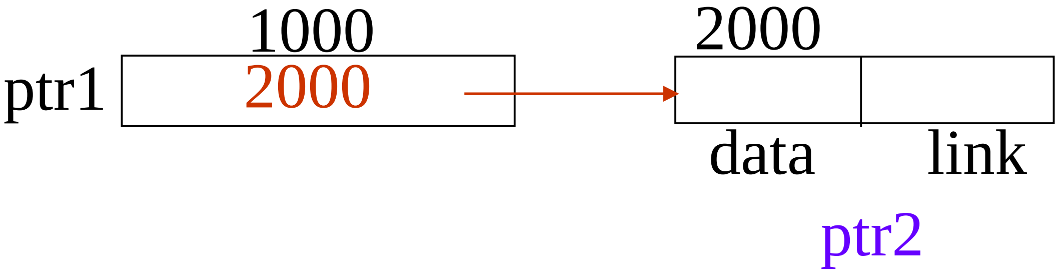


1000

ptr1 NULL

ptr1->data⇨(*ptr1).data

```
ptr2 = malloc(sizeof(list_node));
ptr1 = &ptr2;
```



1000 2000

ptr1 2000

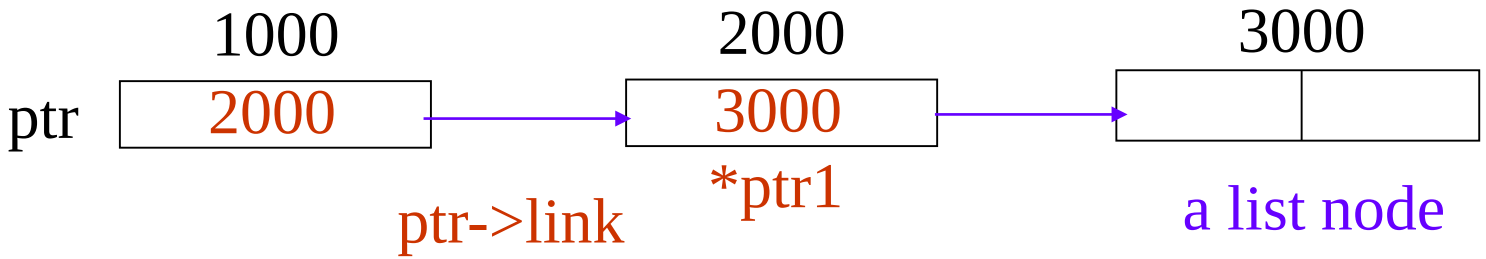
data link

ptr2

Pointer Review (3)

void delete(list_pointer *ptr, list_pointer trail, list_pinter node)

ptr: a pointer point to a pointer point to a list node



1000

2000

3000

ptr

2000

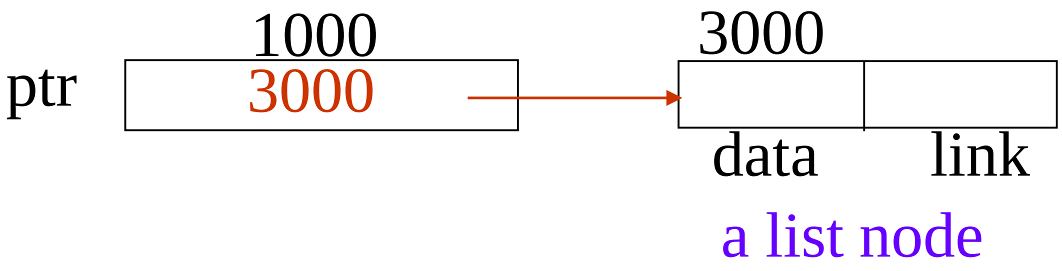
3000

*ptr1

ptr->link

a list node

ptr = & node; (a pointer point to a list node)



1000

3000

ptr

3000

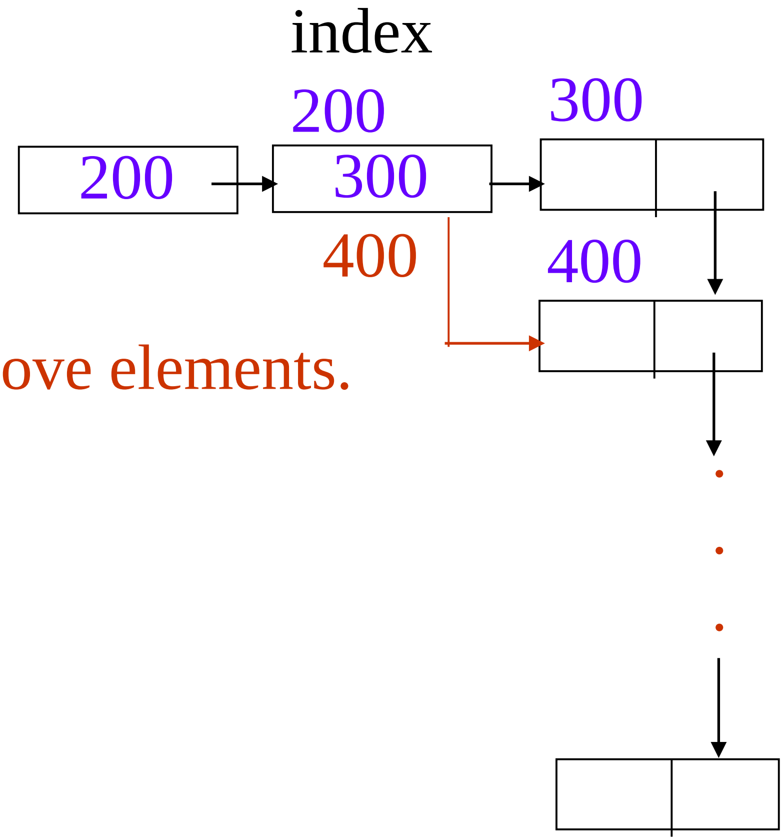
data

link

a list node

Pointer Review (4)

element delete(node_pointer *index)



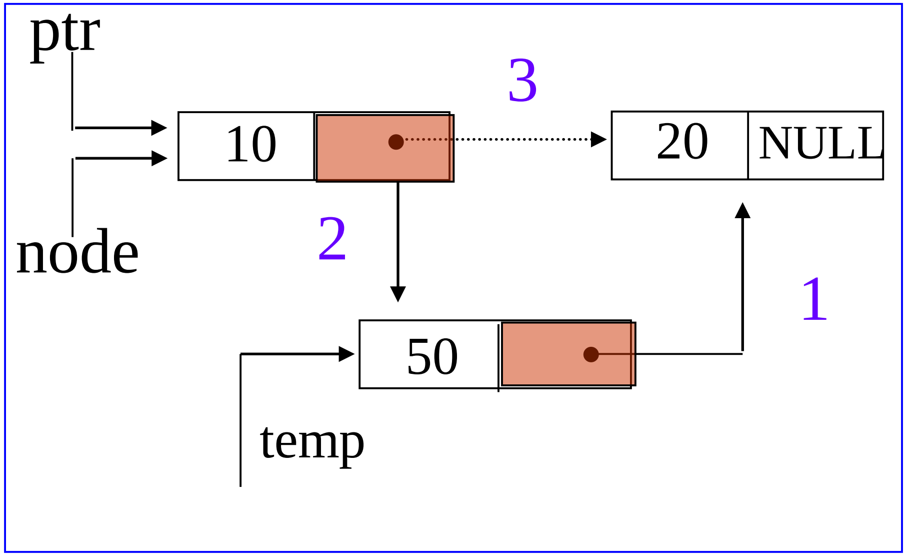
Does not move elements.

List Insertion

Insert a node after a specific node

```
void insert(list_pointer *ptr, list_pointer x)
{
/* insert a new node with data = 50 into the list ptr after node */
    list_pointer temp;
    temp = (list_pointer) malloc(sizeof(list_node));
    if (IS_FULL(temp)){
       fprintf(stderr, “The memory is full\n”);
       exit (1);
    }
```

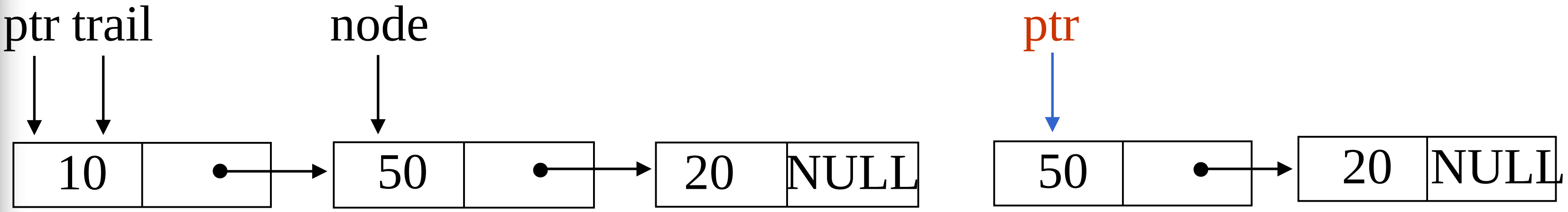
```
  temp->data = 50;
  if (*ptr) {  //noempty list
      temp->link =node ->link;
      node->link = temp;
 }
 else {    //empty list
     temp->link = NULL;
     *ptr =temp;
  }
}
```



***Program 4.2:**Simple insert into front of list

List Deletion

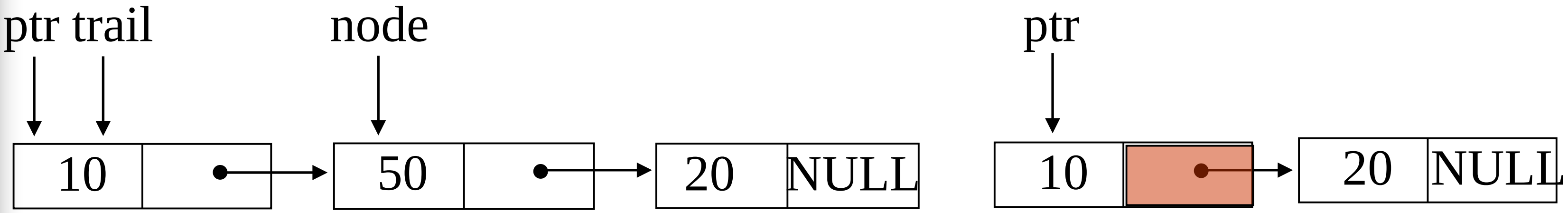
1: Delete the first node.



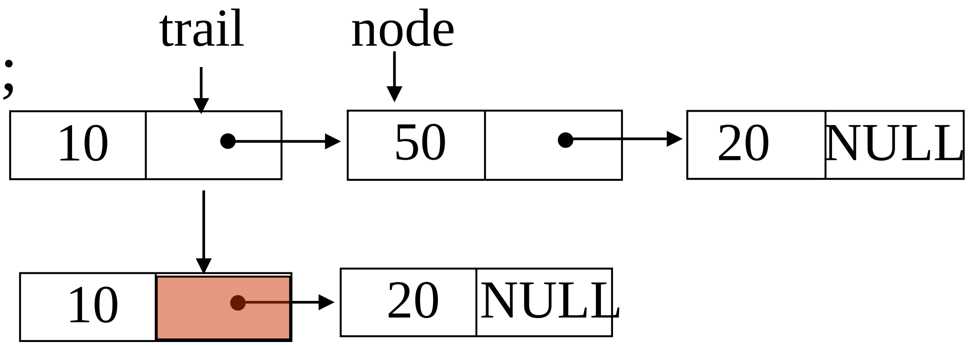
(a) before deletion

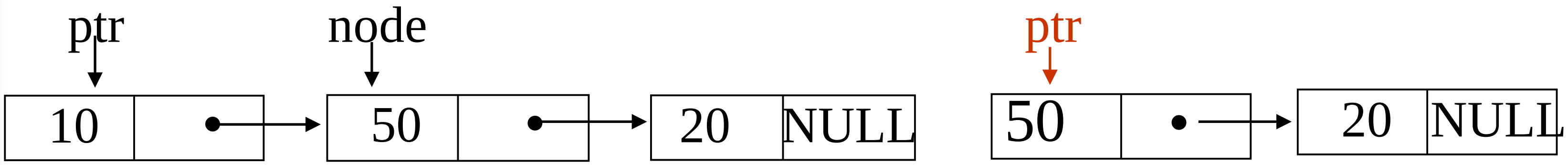
(b) after deletion

2: Delete node other than the first node.



```
void delete(list_pointer *ptr, list_pointer trail,
                                                  list_pointer node)
{
/* delete node from the list, trail is the preceding node
   ptr is the head of the list */
   if (trail)
      trail->link = node->link;
   else
      *ptr = ptr ->link; //head
    free(node);
}
```



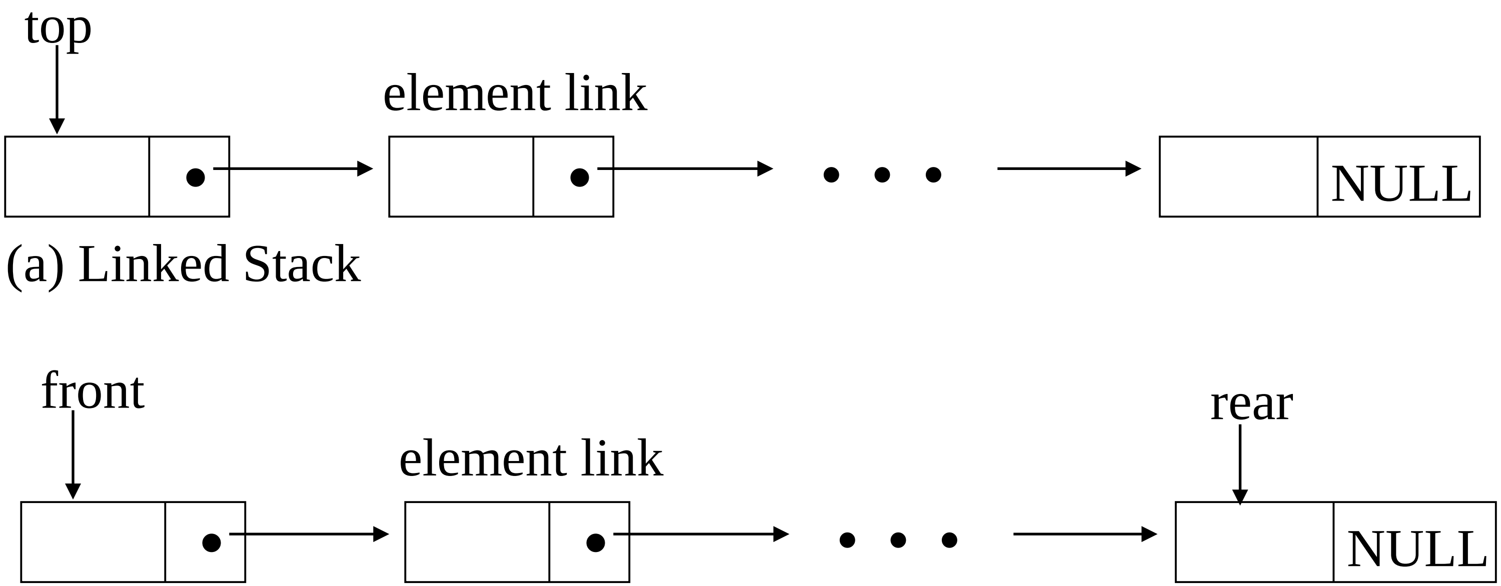


Print out a list (traverse a list)

```
void print_list(list_pointer ptr)
{
    printf(“The list ocntains: “);
    for ( ; ptr; ptr = ptr->link)
       printf(“%4d”, ptr->data);
    printf(“\n”);
 }
```

***Program 4.4:** Printing a list

Linked Stacks and Queues



(a) Linked Stack

(b) Linked queue

***Figure 4.11:** Linked Stack and queue*

Represent n stacks

```
#define MAX_STACKS 10 /* maximum number of stacks */
typedef struct {
        int key;
        /* other fields */
       } element;
typedef struct stack *stack_pointer;

typedef struct stack {
        element item;
        stack_pointer link;
        };
stack_pointer top[MAX_STACKS];
```

Represent n queues

```
#define MAX_QUEUES 10 /* maximum number of queues */
typedef struct queue *queue_pointer;

typedef struct queue {
            element item;
            queue_pointer link;
            };
queue_pointer front[MAX_QUEUE], rear[MAX_QUEUES];
```

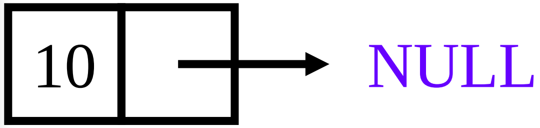
Implementation stack by linked lists

- 用list實作stack的插入和刪除功能
- 實作步驟

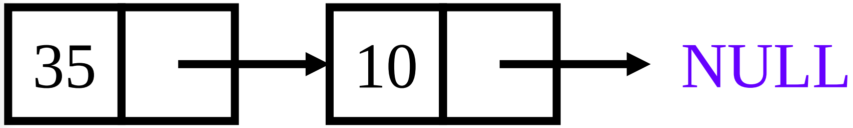
1. 建立node的結構(struct)
2. 主要function
   - 插入: push()
   - 刪除: pop()

Implementation stack by linked lists

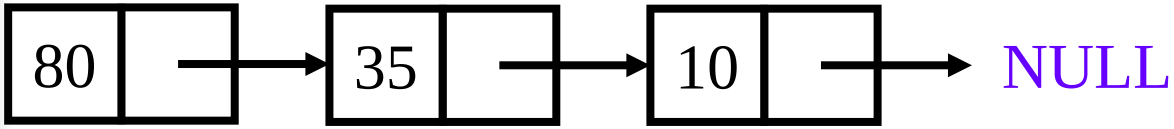
- push 10



- push 35

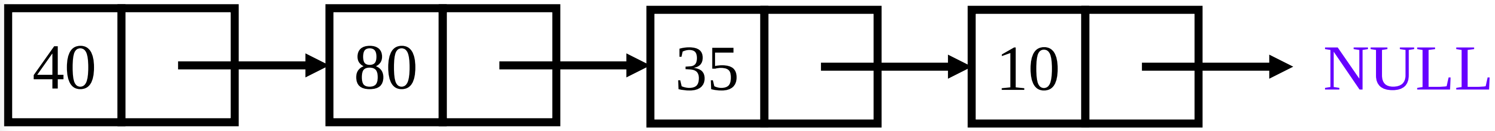


- push 80



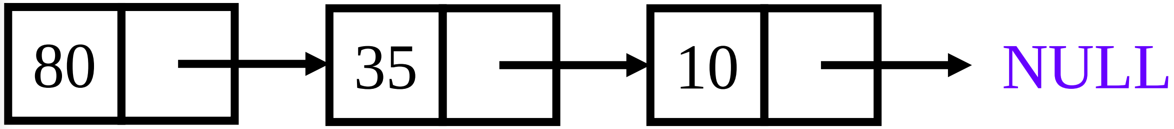
Implementation stack by linked lists

- push 40



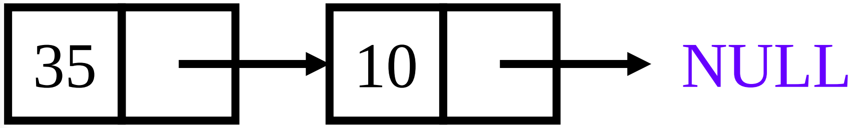
40
80
35
10
NULL

- pop



80
35
10
NULL

- pop



35
10
NULL

push in the linked stack

```
void push(stack_pointer *top, element item)
{
  /* add an element to the top of the stack */
  stack_pointer temp =
                     (stack_pointer) malloc (sizeof (stack));
  if (IS_FULL(temp)) {
    fprintf(stderr, “ The memory is full\n”);
    exit(1);
    }
    temp->item = item;
    temp->link = *top;
    *top= temp;
}
```

***Program 4.5:**Add to a linked stack

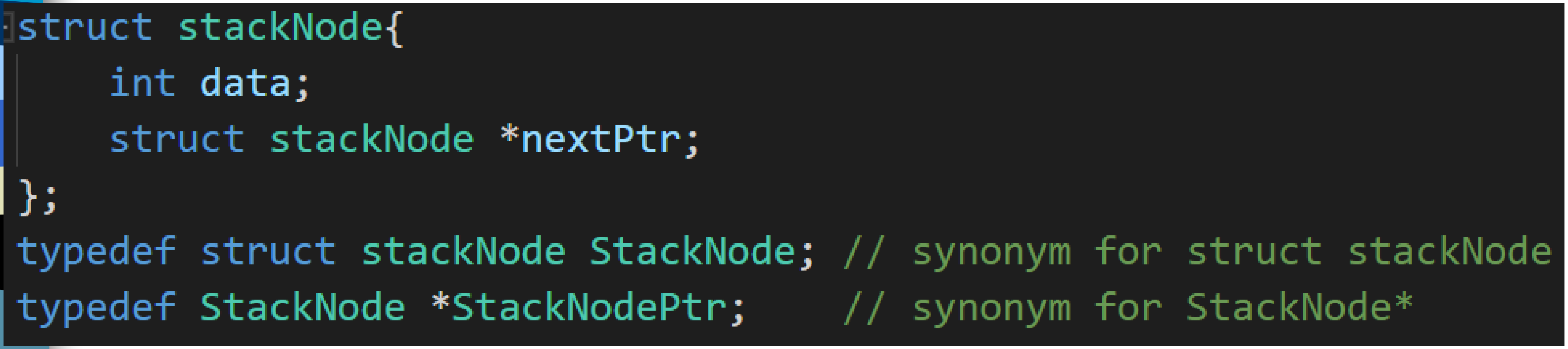
pop from the linked stack

```
element pop(stack_pointer *top) {
/* delete an element from the stack */
    stack_pointer temp = *top;
    element item;
    if (IS_EMPTY(temp))  {
       fprintf(stderr,  “The stack is empty\n”);
       exit(1);
    }
    item = temp->item;
    *top = temp->link;
     free(temp);
     return item;
}
```

*Program 4.6: Delete from a linked stack

Implementation stack by linked lists

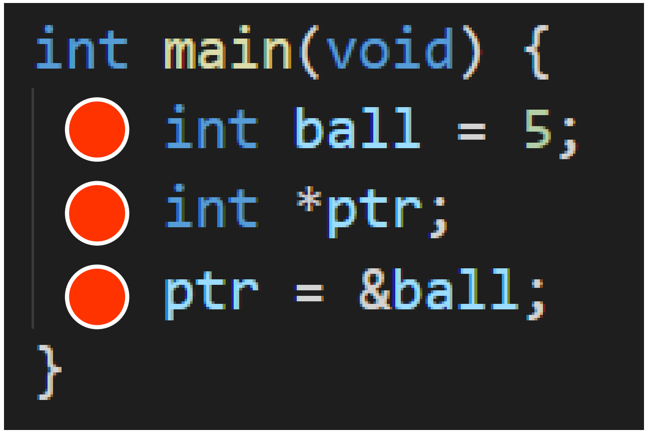
- 建立node的結構(struct)



```
struct stackNode{
    int data;
    struct stackNode *nextPtr;
};
typedef struct stackNode StackNode; // synonym for struct stackNode
typedef StackNode *StackNodePtr;    // synonym for StackNode*
```

Pointer

做法說明:



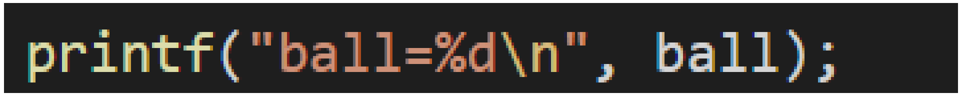
```
int main(void) {
    int ball = 5;
    int *ptr;
    ptr = &ball;
}
```

	值	記憶體位址
ball	5	0061FF18
ptr	0061FF18	0061FF1C

Pointer

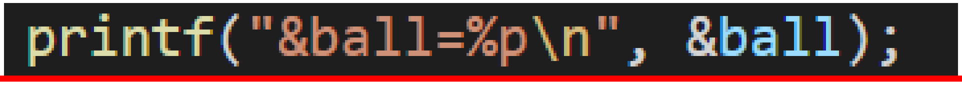
	值	記憶體位址
ball	5	0061FF18
ptr	0061FF18	0061FF1C

做法說明:



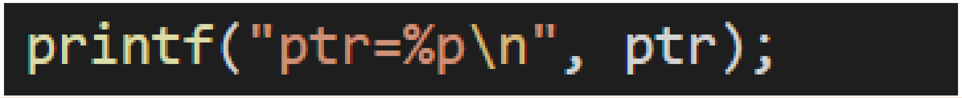
```
printf("ball=%d\n", ball);
```

5



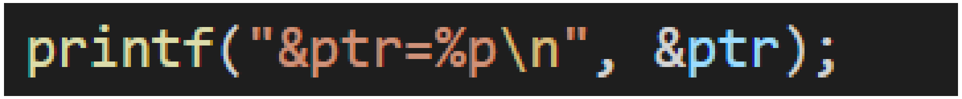
```
printf("&ball=%p\n", &ball);
```

0061FF18



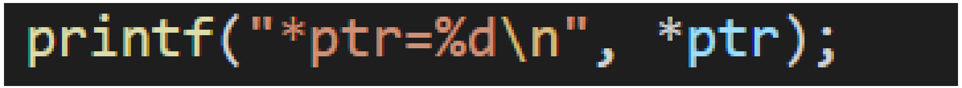
```
printf("ptr=%p\n", ptr);
```

0061FF18



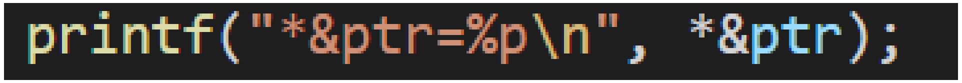
```
printf("&ptr=%p\n", &ptr);
```

0061FF1C



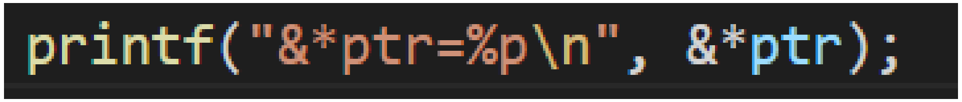
```
printf("*ptr=%d\n", *ptr);
```

5



```
printf("*&ptr=%p\n", *&ptr);
```

0061FF18



```
printf("&*ptr=%p\n", &*ptr);
```

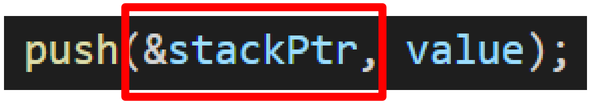
0061FF18

結論: *&ptr = &*ptr = ptr

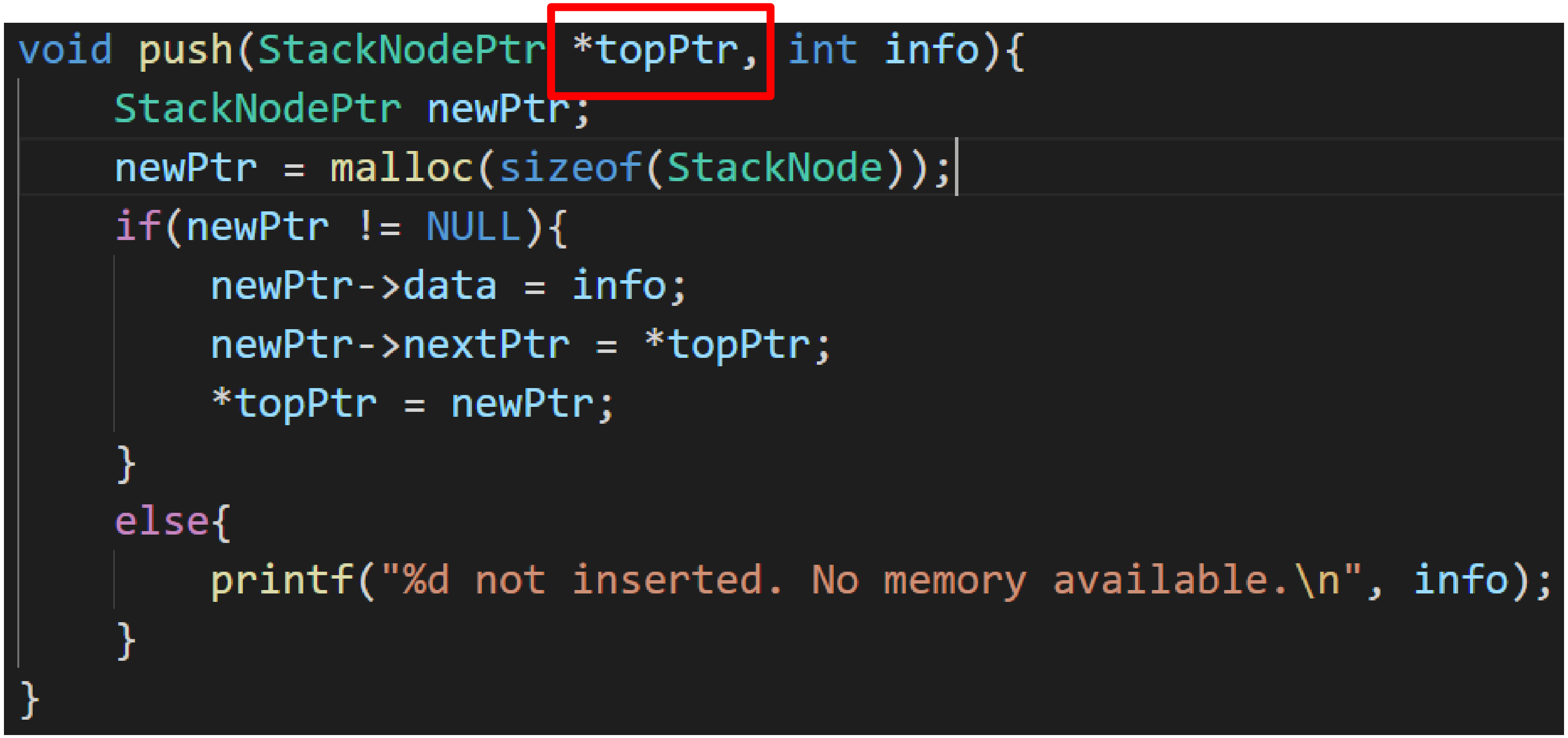
Pointer

把&stackPtr帶入
topPtr = &stackPtr
* topPtr = *&stackPtr
根據前頁結論
*&stackPtr = stackPtr
push(), pop(), enqueue(), dequeue()同理

做法說明:



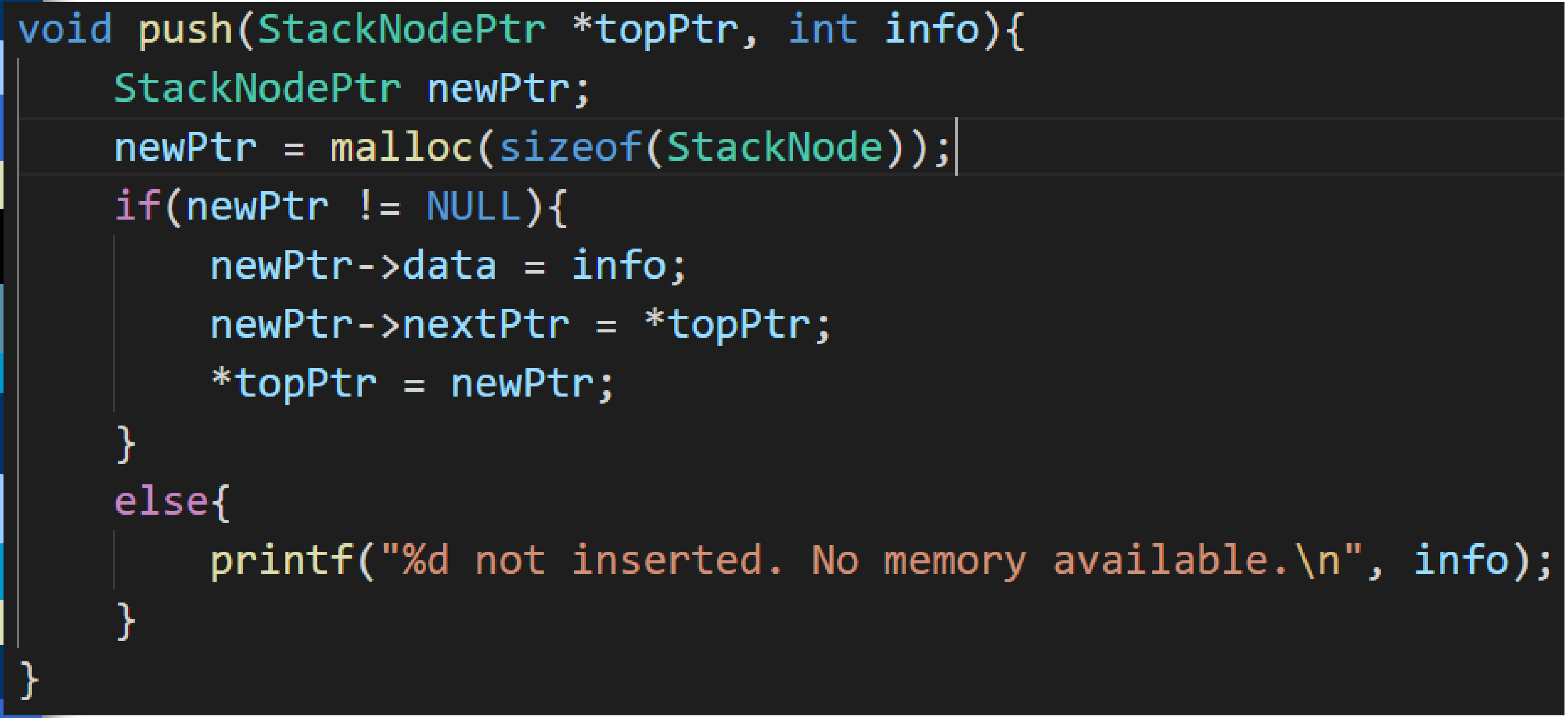
```c
push(&stackPtr, value);
```



```c
void push(StackNodePtr *topPtr, int info){
    StackNodePtr newPtr;
    newPtr = malloc(sizeof(StackNode));
    if(newPtr != NULL){
        newPtr->data = info;
        newPtr->nextPtr = *topPtr;
        *topPtr = newPtr;
    }
    else{
        printf("%d not inserted. No memory available.\n", info);
    }
}
```

Implementation stack by linked lists

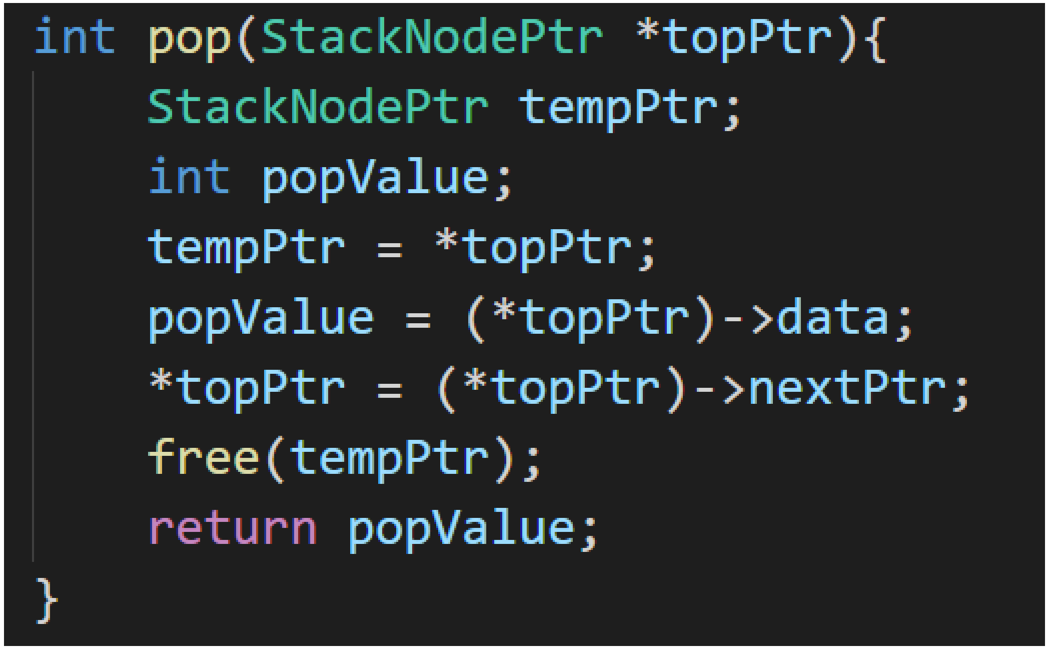
- 插入: push()



```c
void push(StackNodePtr *topPtr, int info){
    StackNodePtr newPtr;
    newPtr = malloc(sizeof(StackNode));
    if(newPtr != NULL){
        newPtr->data = info;
        newPtr->nextPtr = *topPtr;
        *topPtr = newPtr;
    }
    else{
        printf("%d not inserted. No memory available.\n", info);
    }
}
```

Implementation stack by linked lists

- 插入: pop()



```c
int pop(StackNodePtr *topPtr){
    StackNodePtr tempPtr;
    int popValue;
    tempPtr = *topPtr;
    popValue = (*topPtr)->data;
    *topPtr = (*topPtr)->nextPtr;
    free(tempPtr);
    return popValue;
}
```

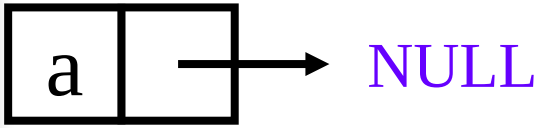
Implementation queue by linked lists

- 用list實作Queue的插入和刪除功能
- 實作步驟

1. 建立node的結構(struct)
2. 主要function
   - 插入: enqueue()
   - 刪除: dequeue()

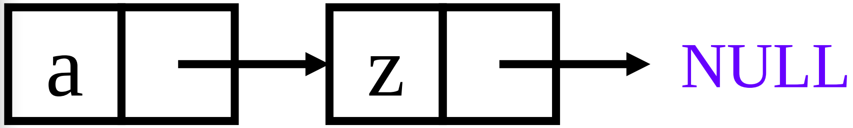
Implementation queue by linked lists

- enqueue a



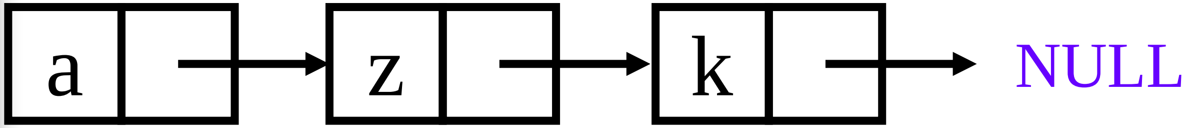
a NULL

- enqueue z



a z NULL

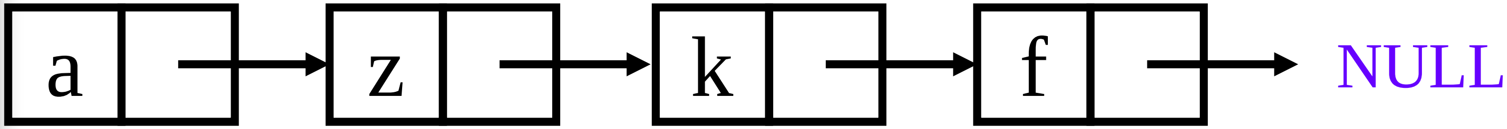
- enqueuer k



a z k NULL

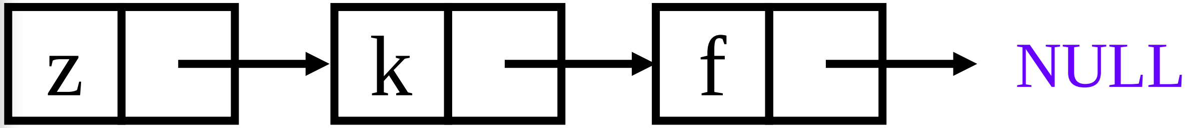
Implementation queue by linked lists

- enqueue f



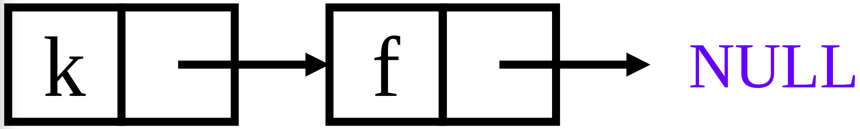
a → z → k → f → NULL

- dequeue



z → k → f → NULL

- dequeue



k → f → NULL

enqueue in the linked queue

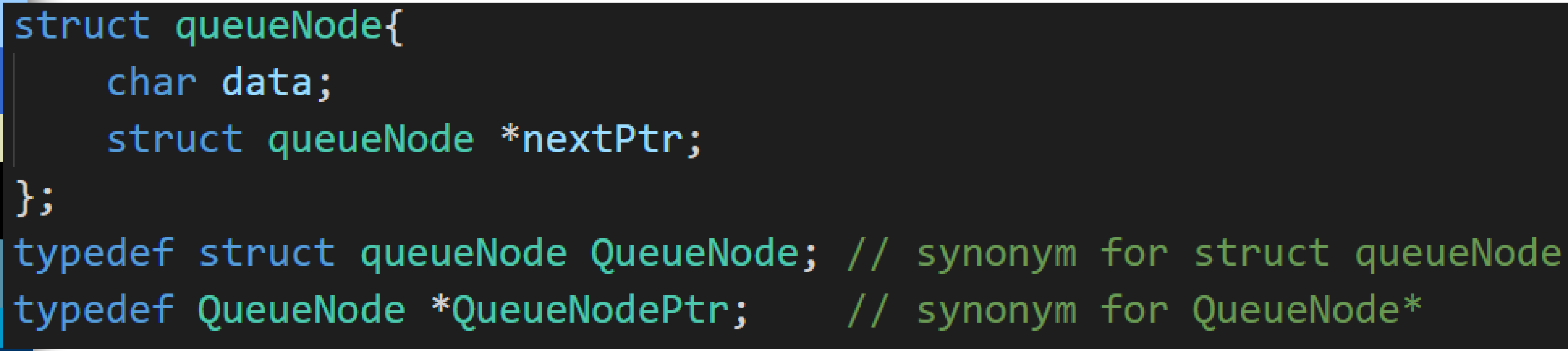
```
void addq(queue_pointer *front, queue_pointer *rear, element
item)
{  /* add an element to the rear of the queue */
   queue_pointer temp =
                      (queue_pointer) malloc(sizeof (queue));
   if (IS_FULL(temp)) {
      fprintf(stderr, “ The memory is full\n”);
      exit(1);
      }
      temp->item = item;
      temp->link = NULL;
      if (*front)
             rear -> link = temp;
      else *front = temp;
      *rear = temp;   }
```

dequeue from the linked queue

```
element deleteq(queue_pointer *front) {
/* delete an element from the queue */
    queue_pointer temp = *front;
    element item;
    if (IS_EMPTY(*front))  {
       fprintf(stderr,  “The queue is empty\n”);
       exit(1);
    }
    item = temp->item;
    *front = temp->link;
     free(temp);
     return item;
}
```

Implementation queue by linked lists

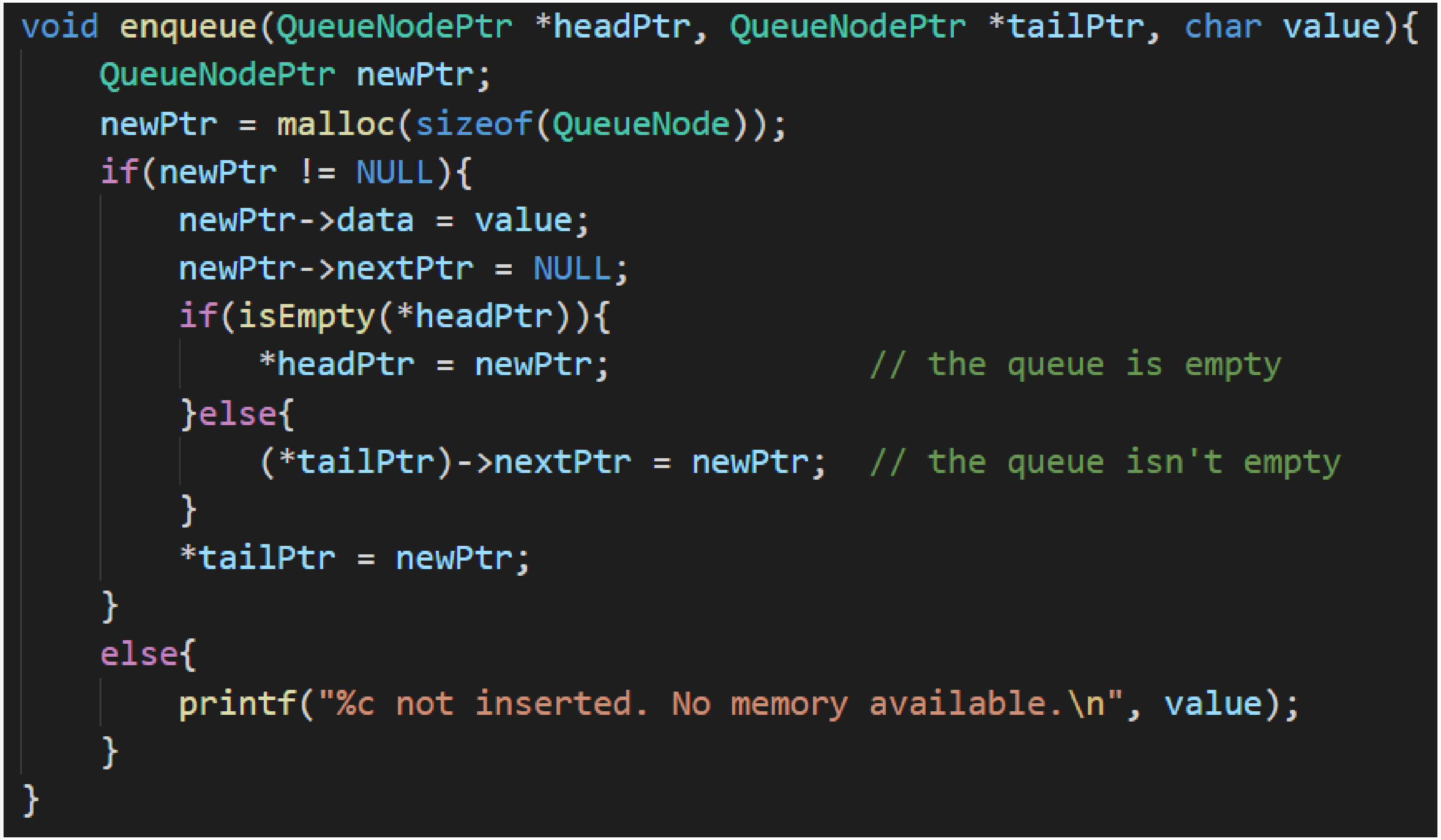
- 建立node的結構(struct)



```
struct queueNode{
    char data;
    struct queueNode *nextPtr;
};
typedef struct queueNode QueueNode; // synonym for struct queueNode
typedef QueueNode *QueueNodePtr;     // synonym for QueueNode*
```

Implementation queue by linked lists

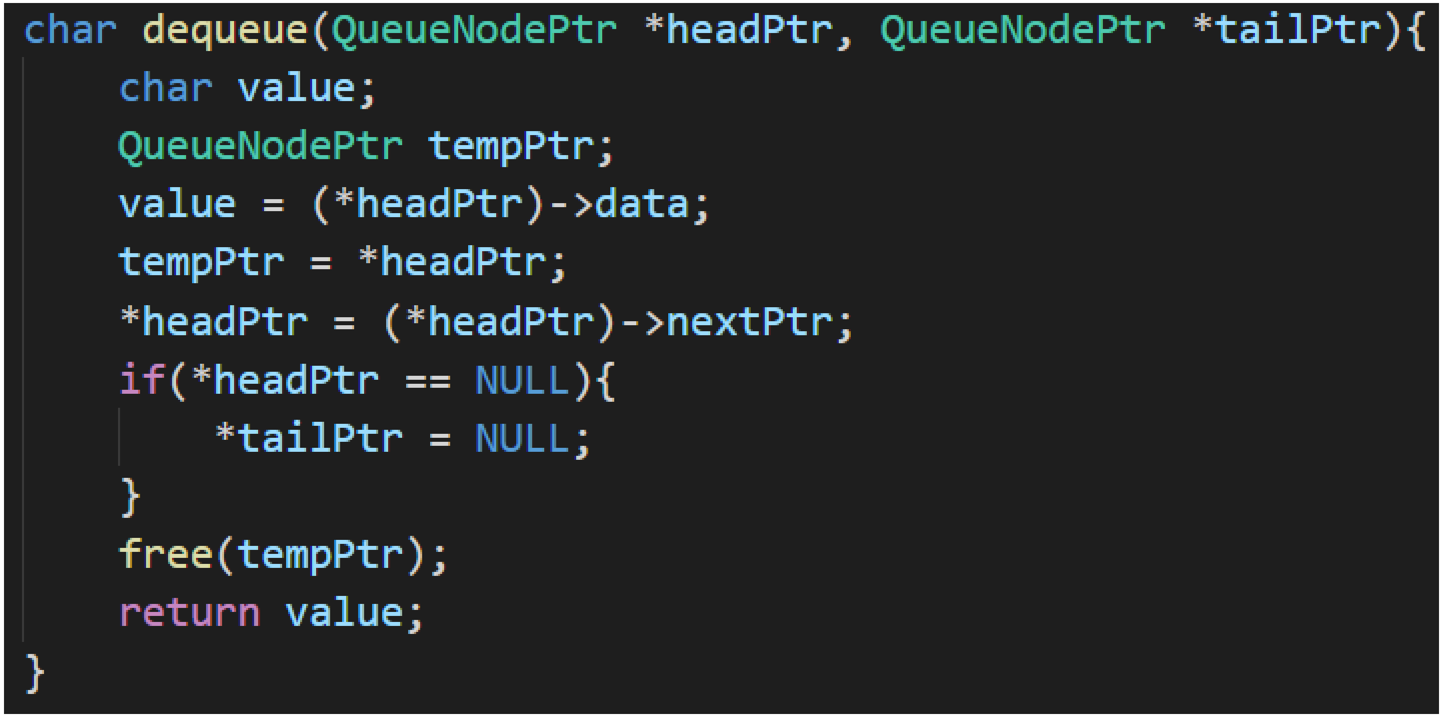
- 插入: enqueue()



```c
void enqueue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr, char value){
    QueueNodePtr newPtr;
    newPtr = malloc(sizeof(QueueNode));
    if(newPtr != NULL){
        newPtr->data = value;
        newPtr->nextPtr = NULL;
        if(isEmpty(*headPtr)){
            *headPtr = newPtr;             // the queue is empty
        }else{
            (*tailPtr)->nextPtr = newPtr;  // the queue isn't empty
        }
        *tailPtr = newPtr;
    }
    else{
        printf("%c not inserted. No memory available.\n", value);
    }
}
```

Implementation queue by linked lists

- 刪除: dequeue()



```
char dequeue(QueueNodePtr *headPtr, QueueNodePtr *tailPtr){
    char value;
    QueueNodePtr tempPtr;
    value = (*headPtr)->data;
    tempPtr = *headPtr;
    *headPtr = (*headPtr)->nextPtr;
    if(*headPtr == NULL){
        *tailPtr = NULL;
    }
    free(tempPtr);
    return value;
}
```

Polynomials

$$A(x) = a_{m-1}x^{e_{m-1}} + a_{m-2}x^{e_{m-2}} + \ldots + a_0 x^{e_0}$$

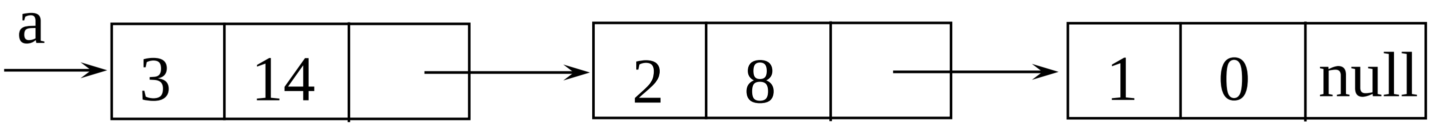
Representation

```
typedef struct poly_node *poly_pointer;
typedef struct poly_node {
    int coef;
    int expon;
    poly_pointer link;
};
poly_pointer a, b, c;
```

coef	expon	link

Examples

$$a = 3x^{14} + 2x^{8} + 1$$

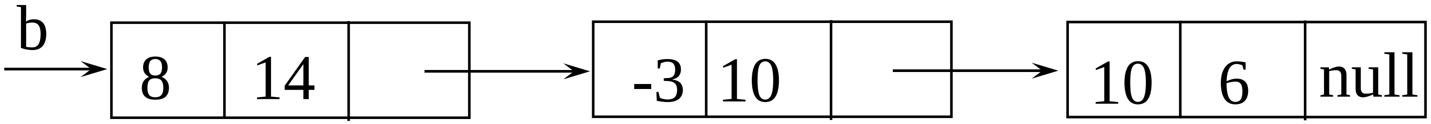

a

3 14

2 8

1 0 null

$$b = 8x^{14} - 3x^{10} + 10x^{6}$$

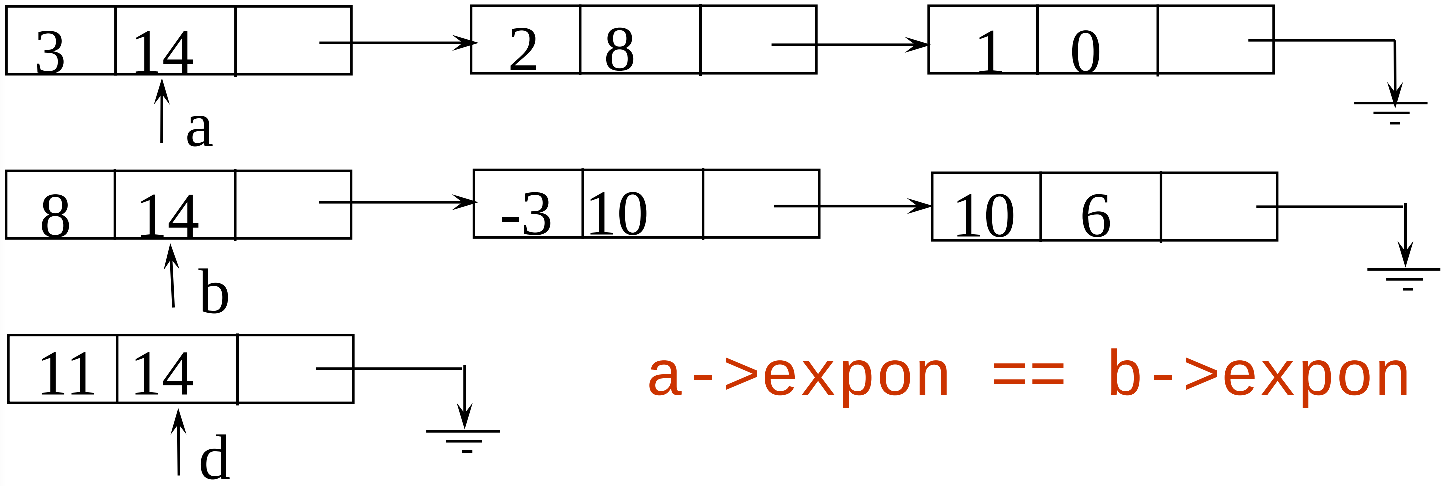

b

8 14

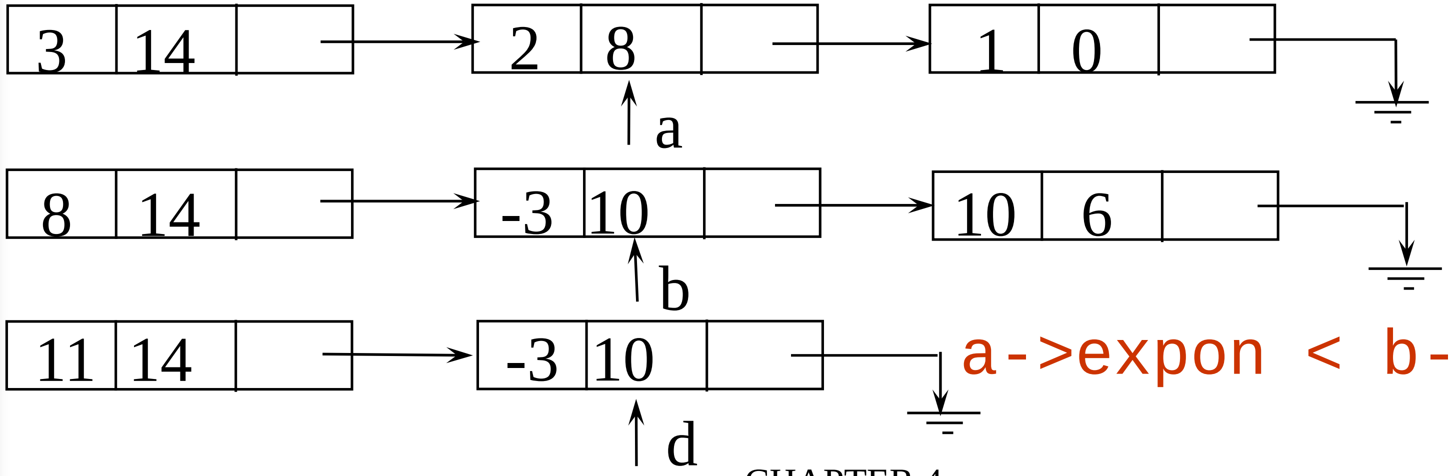
-3 10

10 6 null

Adding Polynomials

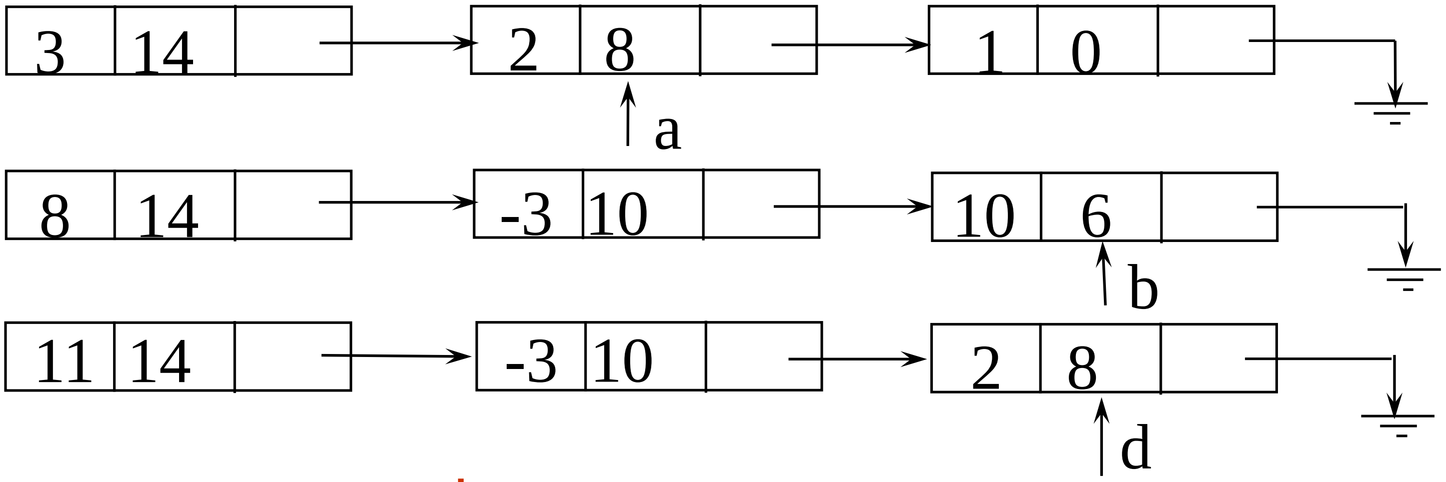


a->expon == b->expon



a->expon < b->expon

Adding Polynomials (*Continued*)



a->expon > b->expon

Alogrithm for Adding Polynomials

```
poly_pointer padd(poly_pointer a, poly_pointer b)
{
    poly_pointer c, rear, temp;
    int sum;
    rear =(poly_pointer)malloc(sizeof(poly_node));
    if (IS_FULL(rear)) {
        fprintf(stderr, “The memory is full\n”);
        exit(1);
    }
    front = rear;
    while (a && b) {
        switch (COMPARE(a->expon, b->expon)) {
            case -1: /* a->expon < b->expon */
                attach(b->coef, b->expon, &rear);
                b= b->link;
                break;
            case 0: /* a->expon == b->expon */
                sum = a->coef + b->coef;
                if (sum) attach(sum,a->expon,&rear);
                a = a->link;    b = b->link;
                break;
            case 1: /* a->expon > b->expon */
                attach(a->coef, a->expon, &rear);
                a = a->link;
        }
    }
    for (; a; a = a->link)
        attach(a->coef, a->expon, &rear);
    for (; b; b=b->link)
        attach(b->coef, b->expon, &rear);
    rear->link = NULL;
    temp = front;  front = front->link;  free(temp);
    return front;
}
```

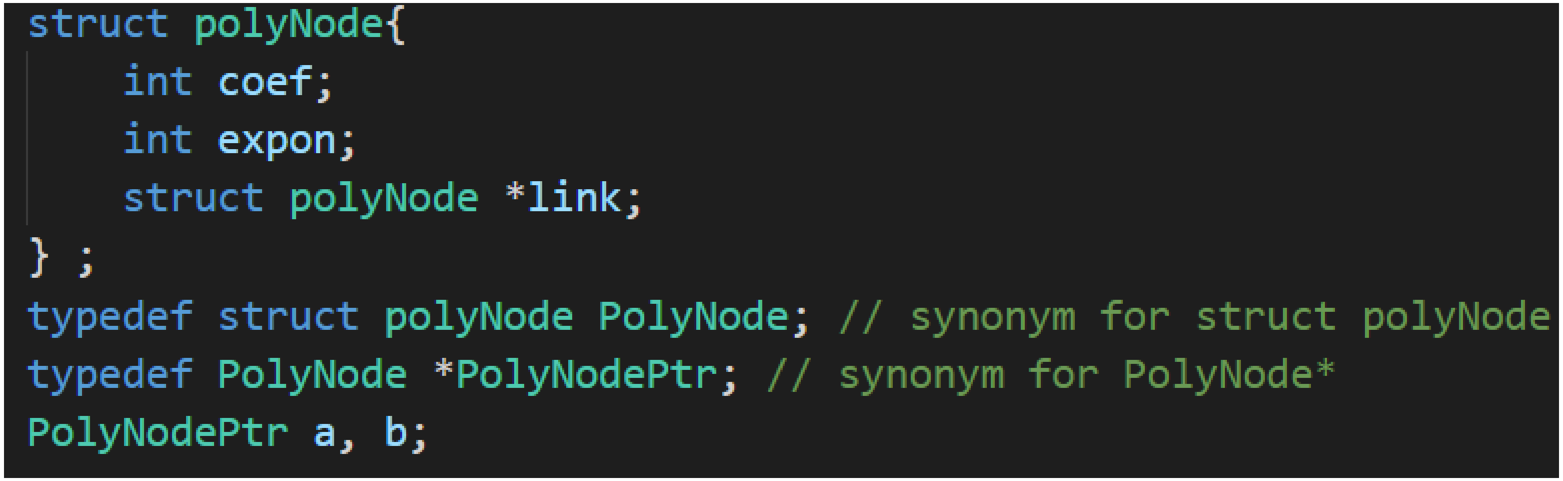
Delete extra initial node.

Attach a Term

```
void attach(float coefficient, int exponent,
            poly_pointer *ptr)
{
/* create a new node attaching to the node pointed to
   by ptr. ptr is updated to point to this new node. */
    poly_pointer temp;
    temp = (poly_pointer) malloc(sizeof(poly_node));
    if (IS_FULL(temp)) {
        fprintf(stderr, “The memory is full\n”);
        exit(1);
    }
    temp->coef = coefficient;
    temp->expon = exponent;
    ptr ->link = temp;
    *ptr = temp;
}
```

Algorithm for adding polynomials

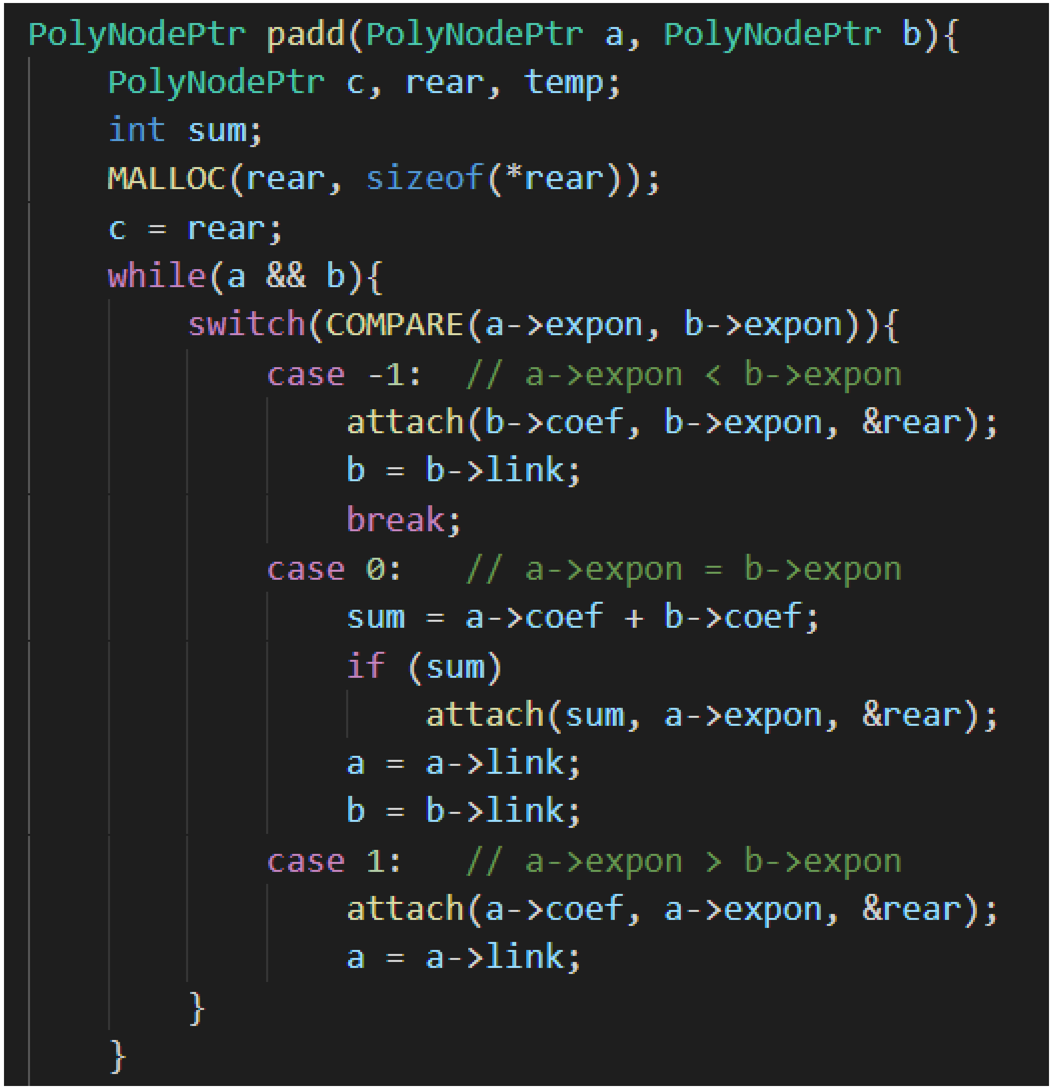
- 建立多項式node的結構(struct)



```
struct polyNode{
    int coef;
    int expon;
    struct polyNode *link;
} ;
typedef struct polyNode PolyNode; // synonym for struct polyNode
typedef PolyNode *PolyNodePtr; // synonym for PolyNode*
PolyNodePtr a, b;
```

Algorithm for adding polynomials

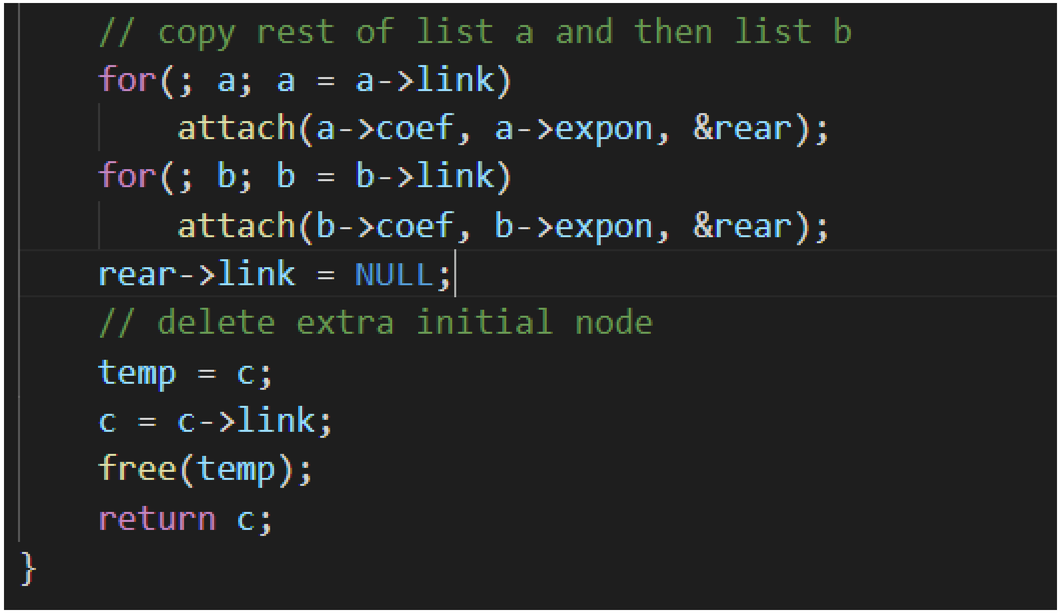
- padd()



```
PolyNodePtr padd(PolyNodePtr a, PolyNodePtr b){
    PolyNodePtr c, rear, temp;
    int sum;
    MALLOC(rear, sizeof(*rear));
    c = rear;
    while(a && b){
        switch(COMPARE(a->expon, b->expon)){
            case -1:  // a->expon < b->expon
                attach(b->coef, b->expon, &rear);
                b = b->link;
                break;
            case 0:   // a->expon = b->expon
                sum = a->coef + b->coef;
                if (sum)
                    attach(sum, a->expon, &rear);
                a = a->link;
                b = b->link;
            case 1:   // a->expon > b->expon
                attach(a->coef, a->expon, &rear);
                a = a->link;
        }
    }
```

Algorithm for adding polynomials

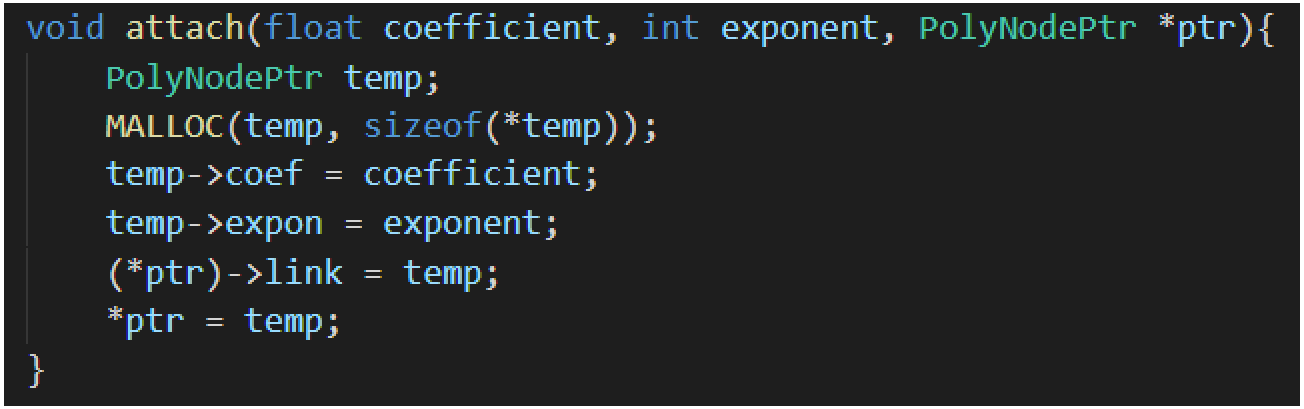
- padd()



```
    // copy rest of list a and then list b
    for(; a; a = a->link)
        attach(a->coef, a->expon, &rear);
    for(; b; b = b->link)
        attach(b->coef, b->expon, &rear);
    rear->link = NULL;
    // delete extra initial node
    temp = c;
    c = c->link;
    free(temp);
    return c;
}
```

Algorithm for adding polynomials

- attach()



```c
void attach(float coefficient, int exponent, PolyNodePtr *ptr){
    PolyNodePtr temp;
    MALLOC(temp, sizeof(*temp));
    temp->coef = coefficient;
    temp->expon = exponent;
    (*ptr)->link = temp;
    *ptr = temp;
}
```

Analysis

(1) coefficient additions

$0 \leq$ number of coefficient additions $\leq \min(m, n)$

where m (n) denotes the number of terms in A (B)

(2) exponent comparisons

extreme case

$e_{m-1} > f_{m-1} > e_{m-2} > f_{m-2} > \ldots > e_0 > f_0$

m+n-1 comparisons

(3) creation of new nodes

extreme case

m + n new nodes   O(m+n)

summary

A Suite for Polynomials

$e(x) = a(x) * b(x) + d(x)$

```
poly_pointer a, b, d, e;
...
a = read_poly();
b = read_poly();
d = read_poly();
temp = pmult(a, b);
e = padd(temp, d);
print_poly(e);
```

```
read_poly()
print_poly()
padd()
psub()
pmult()
```

temp is used to hold a partial result. By returning the nodes of temp, we may use it to hold other polynomials

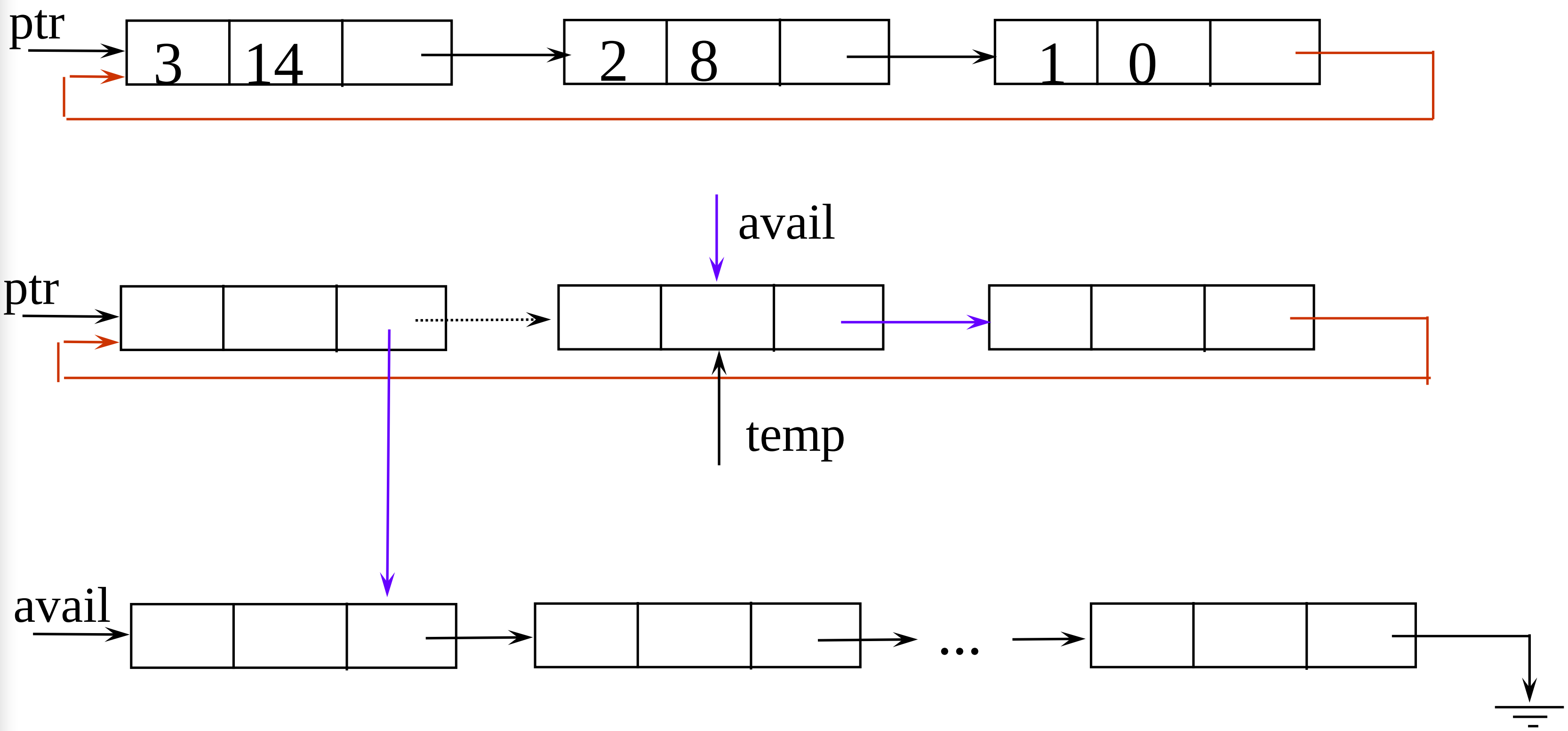
Erase Polynomials

```
void earse(poly_pointer *ptr)
{
/* erase the polynomial pointed to by ptr */
    poly_pointer temp;
    while (*ptr) {
        temp = *ptr;
        *ptr = ptr->link;
        free(temp);
    }
}
```

O(n)

Circularly Linked Lists

circular list vs. chain



Maintain an Available List

```
poly_pointer getnode(void)
{
  poly_pointer node;
  if (avail) {
      node = avail;
      avail = avail->link:
  }
  else {
      node = (poly_pointer)malloc(sizeof(poly_node));
      if (IS_FULL(node)) {
          printf(stderr, “The memory is full\n”);
          exit(1);
      }
  }
  return node;
}
```

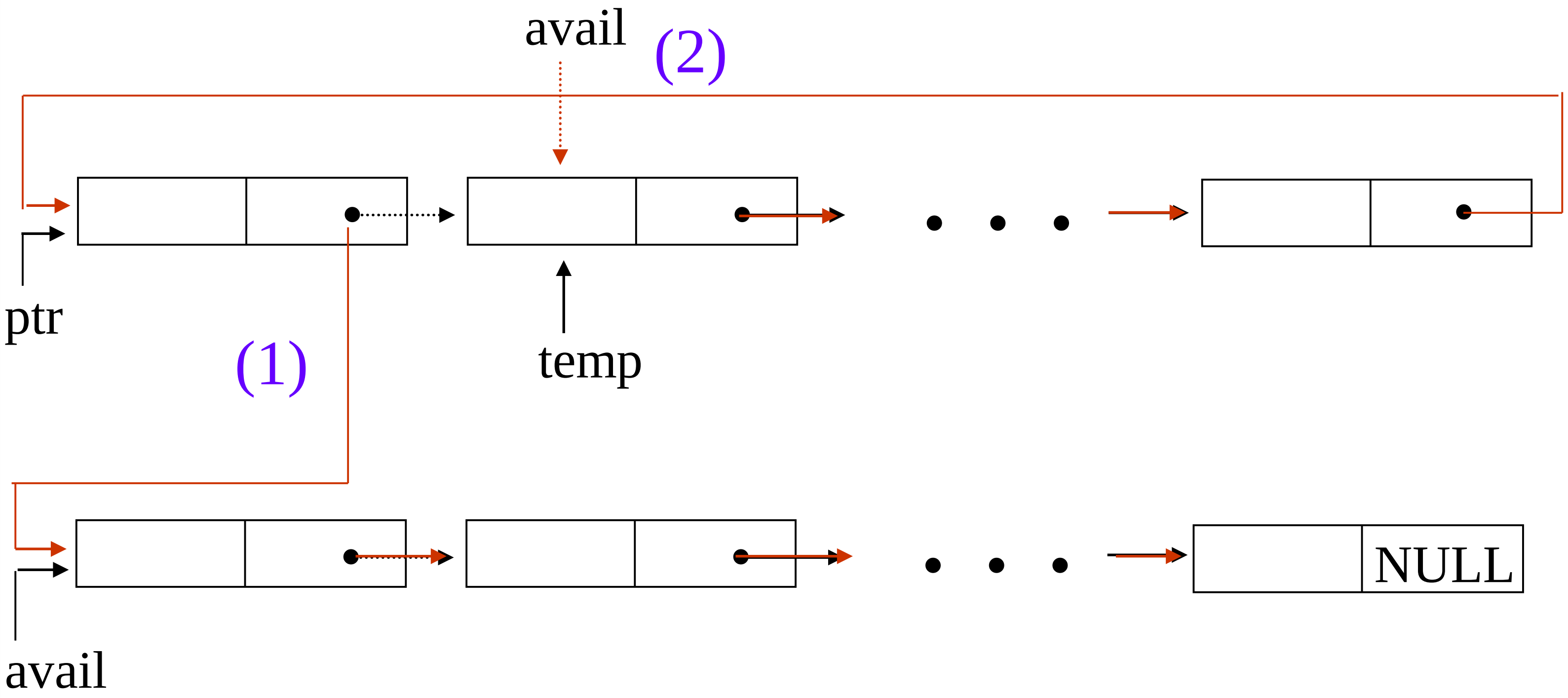
Maintain an Available List *(Continued)*

```
void retNode(poly_pointer ptr)
{
  ptr->link = avail;
    avail = ptr;
}

void cerase(poly_pointer *ptr)
{
    poly_pointer temp;
    if (*ptr) {
        temp = ptr->link;
        ptr->link = avail;     ←     (1)
        avail = temp;     ←     (2)
        *ptr = NULL;
    }
}
```

Erase a circular list (see next page)

Independent of # of nodes in a list O(1) constant time

Circular List Representing of Polynomials

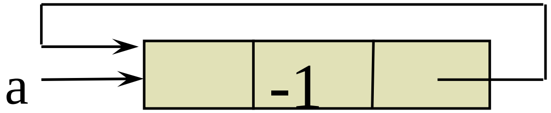


Returning a circular list to the avail list

Head Node

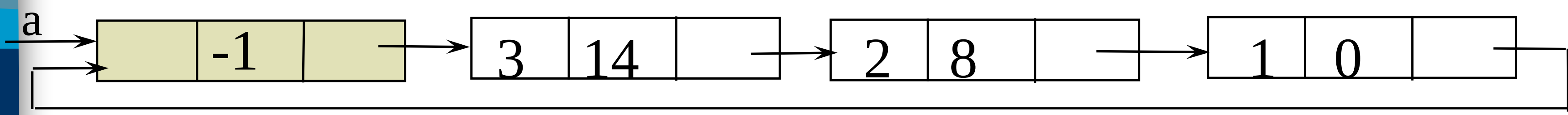
Represent polynomial as circular list.

(1) zero



Zero polynomial

(2) others



$$a = 3x^{14} + 2x^{8} + 1$$

Another Padd

```
poly_pointer cpadd(poly_pointer a, poly_pointer b)
{
  poly_pointer startA, c, lastC;
  int sum, done = FALSE;
  starta = a;
  a = a->link;
  b = b->link;
  c = getnode();
  c->expon = -1;    lastC = c;
  /* get a header node for a and b*/
  do {
    switch (COMPARE(a->expon, b->expon)) {
      case -1: attach(b->coef, b->expon, &lastC);
               b = b->link;
               break;
```

Set expon field of head node to -1.

Another Padd *(Continued)*

```
      case 0: if (startA == a) done = TRUE;
              else {
                sum = a->coef + b->coef;
                if (sum) attach(sum,a->expon,&lastC);
                a = a->link;   b = b->link;
              }
              break;
      case 1: attach(a->coef,a->expon,&lastC);
              a = a->link;
    }
  } while (!done);
  lastC->link = c;
  return c;
}
```

Link last node to first

Additional List Operations

```
typedef struct list_node *list_pointer;
typedef struct list_node {
    char data;
    list_pointer link;
};
```

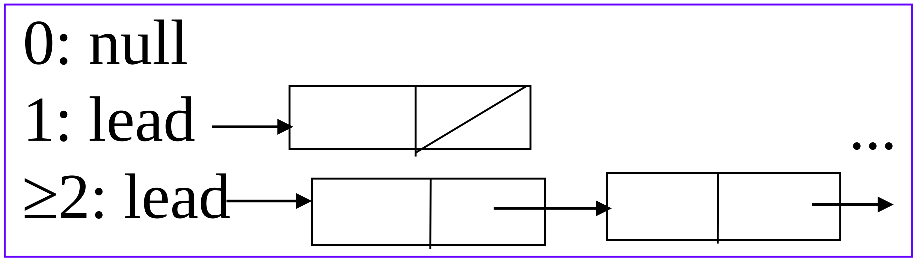
Invert single linked lists

Concatenate two linked lists

Invert Single Linked Lists

Use two extra pointers: middle and trail

```
list_pointer invert(list_pointer lead)
{
    list_pointer middle, trail;
    middle = NULL;
    while (lead) {
        trail = middle; /* NULL */
        middle = lead;
        lead = lead->link;
        middle->link = trail;
    }
    return middle;
}
```



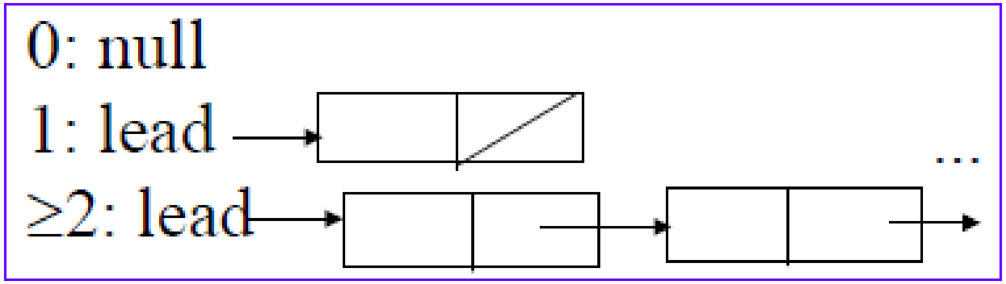
0: null

1: lead

≥2: lead

...

```
middle = NULL
While(lead){
    trail = middle
    middle = lead
    lead = lead -> link
    middle -> link = trail
}
```



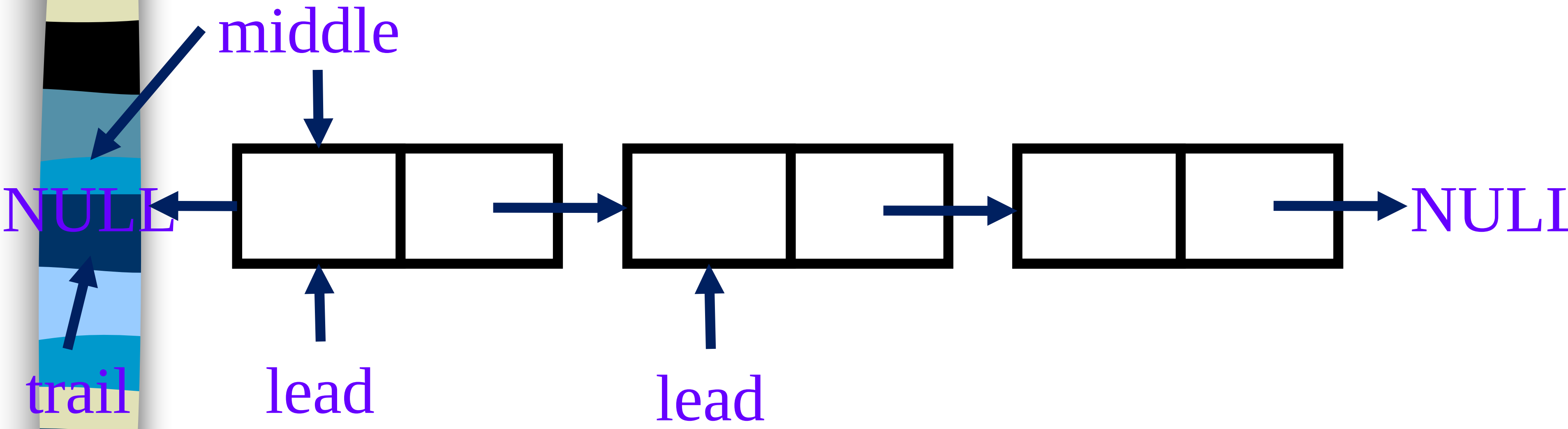
0: null

1: lead

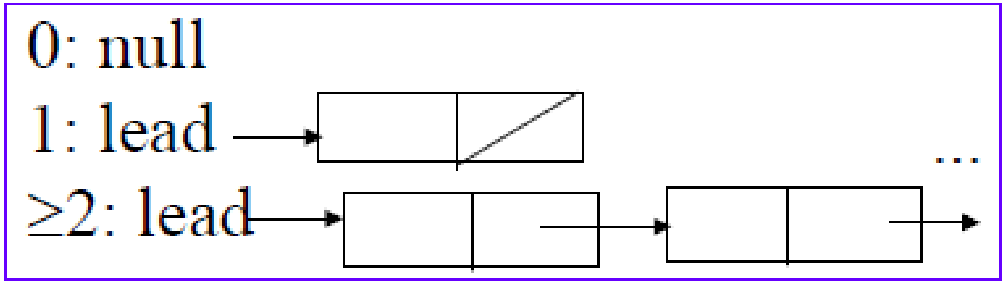
≥2: lead

...

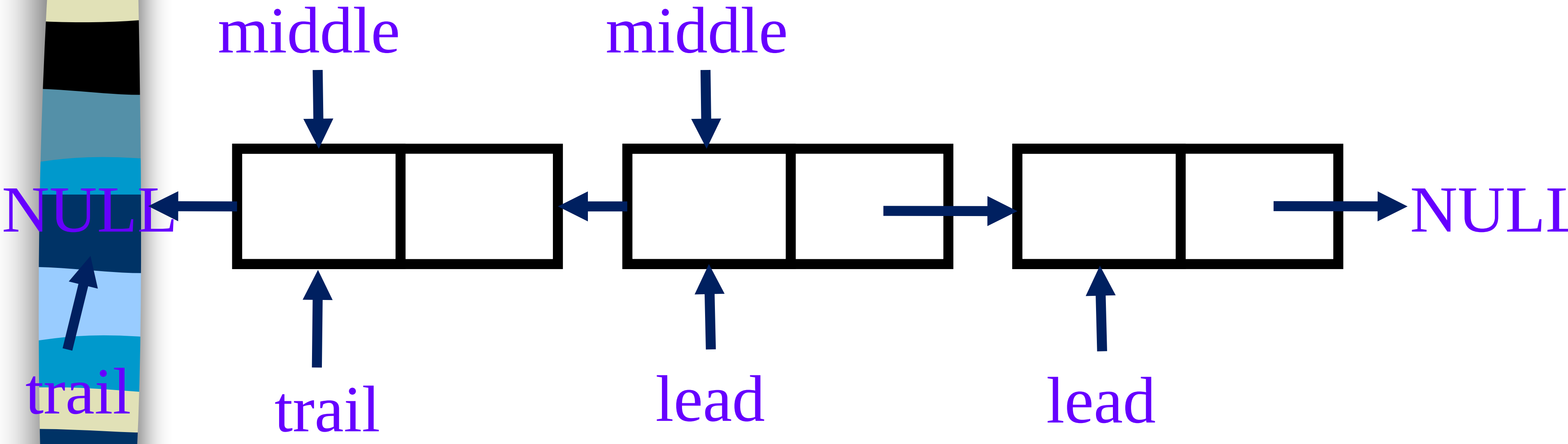
Round 1



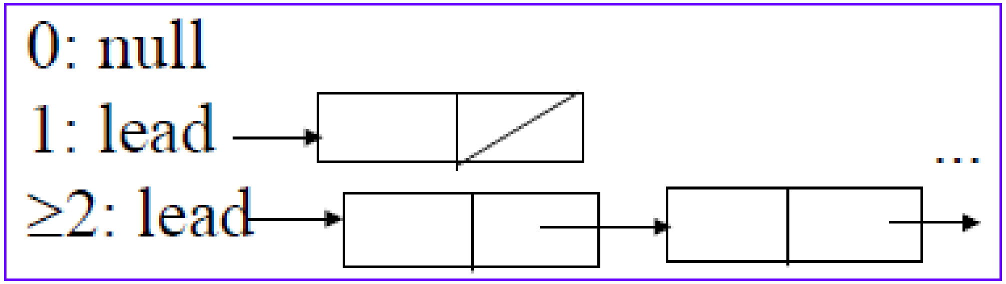
```
middle = NULL
While(lead){
    trail = middle
    middle = lead
    lead = lead -> link
    middle -> link = trail
}
```



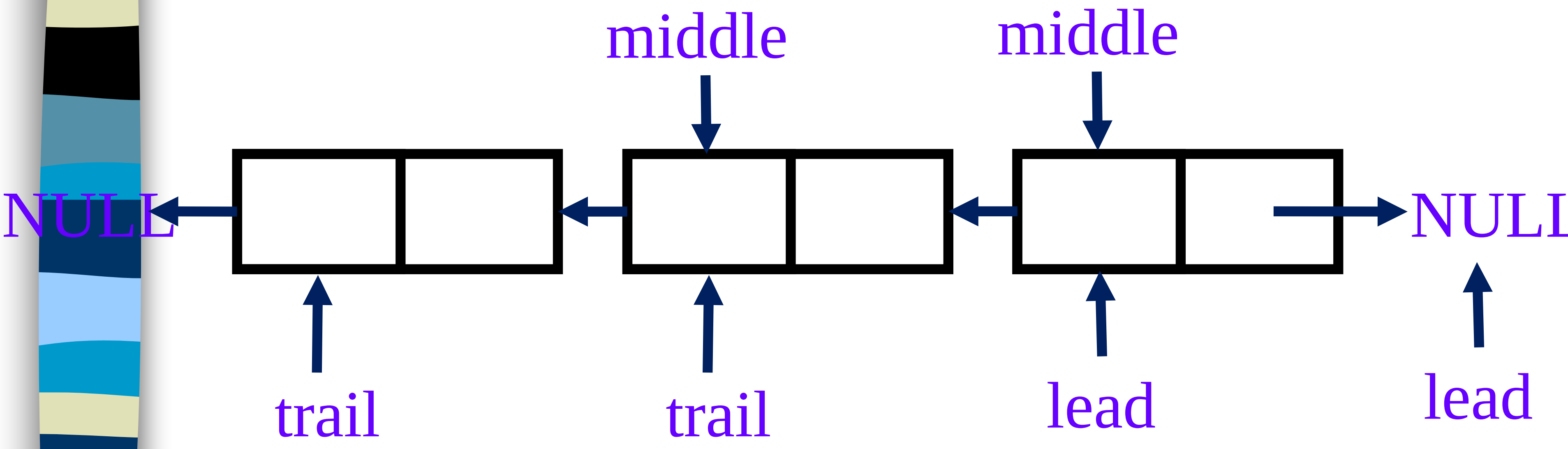
Round 2



```
middle = NULL
While(lead){
    trail = middle
    middle = lead
    lead = lead -> link
    middle -> link = trail
}
```



Round 3



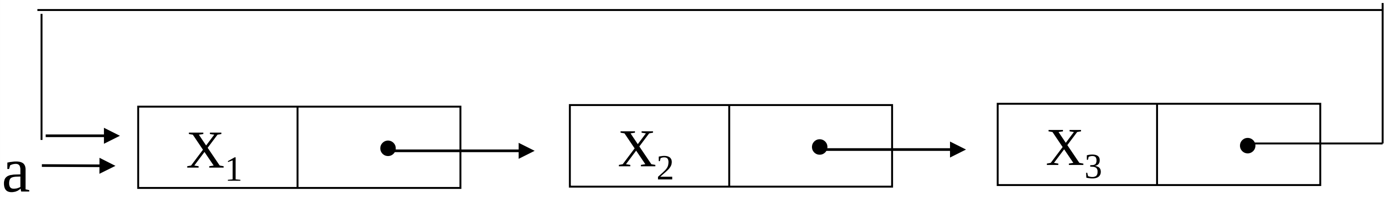
Concatenate Two Lists

```
list_pointer concatenate(list_pointer
             ptr1, list_pointer ptr2)
{
  list_pointer temp;
  if (IS_EMPTY(ptr1)) return ptr2;
  else {
    if (!IS_EMPTY(ptr2)) {
      for (temp=ptr1;temp->link;temp=temp->link);
 /*find end of first list*/
       temp->link = ptr2;
    }
    return ptr1;
  }
}
```

O(m) where m is # of elements in the first list

Operations for Circularly Linked List

What happens when **we insert a node to the front of a circular linked list**?

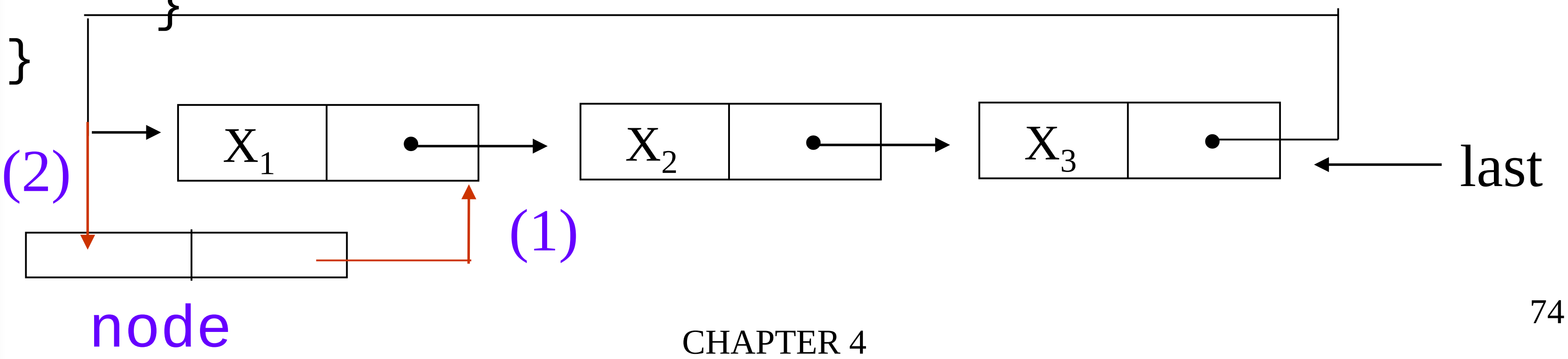


Problem: move down the whole list.

***Figure 4.16:** Example circular list

Operations for Circular Linked Lists

```
void insertFront(list_pointer *last, list_pointer
node)
{
    if (!(*last)) {
    /* list is empty, change last to point to new
entry*/
       *last= node;
       node->link = node;
    }
    else {
        node->link = (*last)->link;  (1)
        (*last)->link = node;        (2)
     }
}
```



Length of Linked List

```
int length(list_pointer last)
{
    list_pointer temp;
    int count = 0;
    if (last) {
        temp = last;
        do {
            count++;
            temp = temp->link;
        } while (temp!=last);
    }
    return count;
}
```

Equivalence Relations

A relation over a set, S, is said to be an *equivalence relation* over S *iff* it is symmertric, reflexive, and transitive over S.

reflexive, x=x

symmetric, if x=y, then y=x

transitive, if x=y and y=z, then x=z

Examples

$0 \equiv 4$, $3 \equiv 1$, $6 \equiv 10$, $8 \equiv 9$, $7 \equiv 4$,
$6 \equiv 8$, $3 \equiv 5$, $2 \equiv 11$, $11 \equiv 0$

three equivalent classes
{0,2,4,7,11}; {1,3,5}; {6,8,9,10}

A Rough Algorithm to Find Equivalence Classes

```
void equivalenec()
{
    initialize;
    while (there are more pairs) {
        read the next pair <i,j>;
        process this pair;
    }
    initialize the output;
    do {
        output a new equivalence class;
    } while (not done);
}
```

Phase 1

Phase 2

What kinds of data structures are adopted?

First Refinement

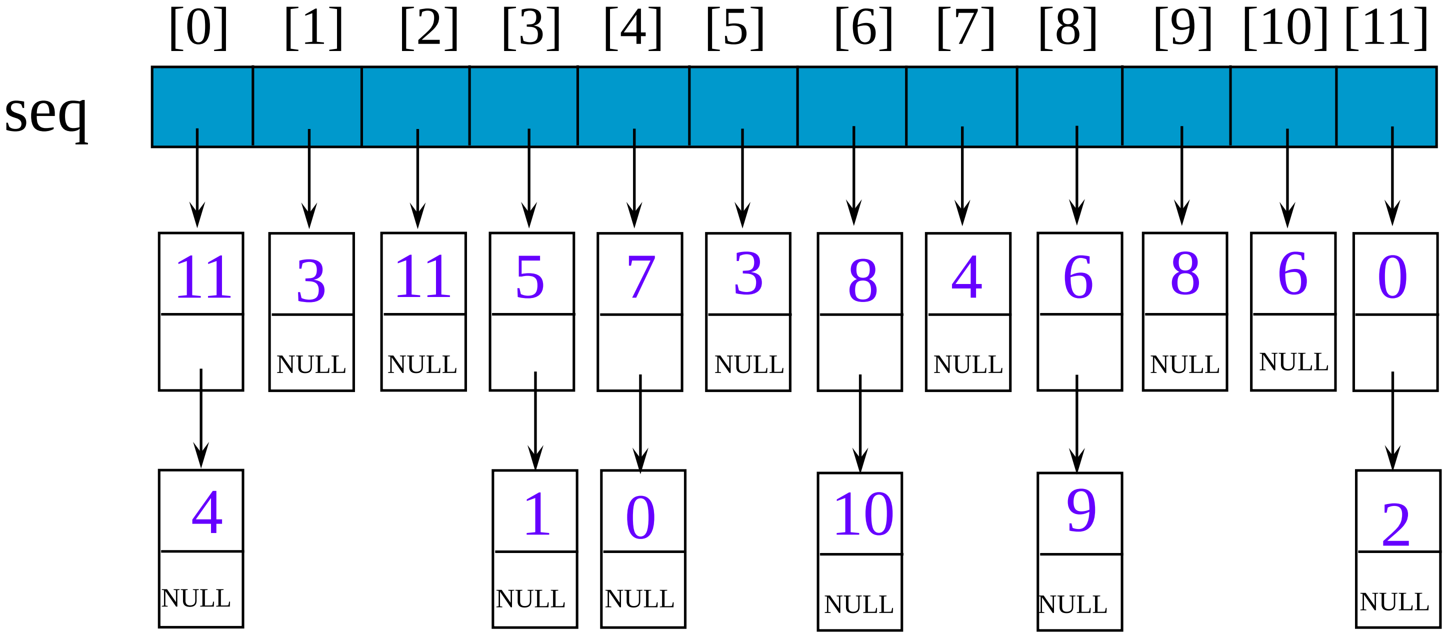
```
#include <stdio.h>
#include <alloc.h>
#define MAX_SIZE 24
#define IS_FULL(ptr)  (!(ptr))
#define FALSE  0
#define TRUE   1
void equivalence()
{
    initialize seq to NULL and out to TRUE
    while (there are more pairs) {
        read the next pair, <i,j>;
        put j on the seq[i] list;
        put i on the seq[j] list;
    }
    for (i=0; i<n; i++)
        if (out[i]) {
            out[i]= FALSE;
            output this equivalence class;
        }
}
```

direct equivalence

Compute indirect equivalence using transitivity

Lists After Pairs are input

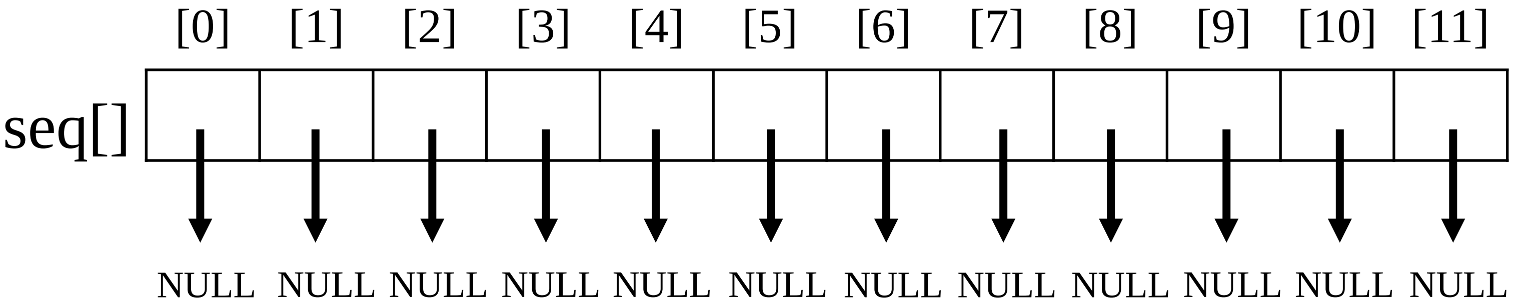
0 ≡ 4
3 ≡ 1
6 ≡ 10
8 ≡ 9
7 ≡ 4
6 ≡ 8
3 ≡ 5
2 ≡ 11
11 ≡ 0



```
typedef struct node *node_pointer ;
typedef struct node {
    int data;
    node_pointer link;
};
```

Equivalence relations

Phase 1: input the equivalence pairs



[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]

seq[]

NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL NULL

$0 \equiv 4$

$3 \equiv 1$

$6 \equiv 10$

$8 \equiv 9$

$7 \equiv 4$

$6 \equiv 8$

$3 \equiv 5$

$2 \equiv 11$

$11 \equiv 0$

Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

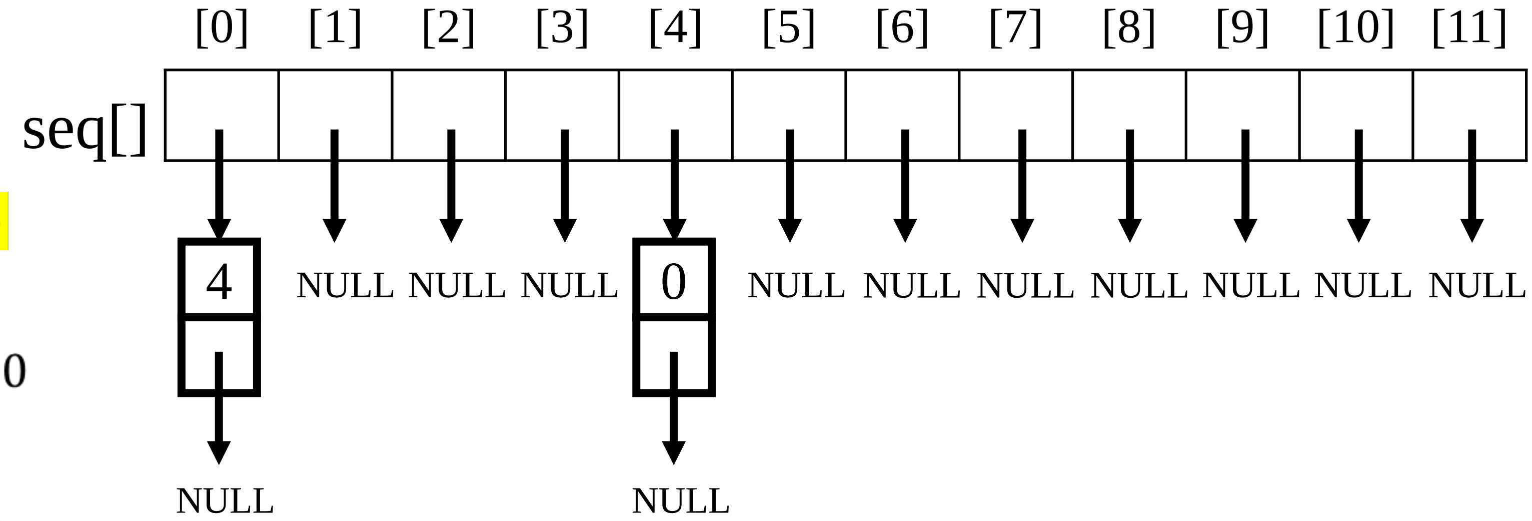
7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0



Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

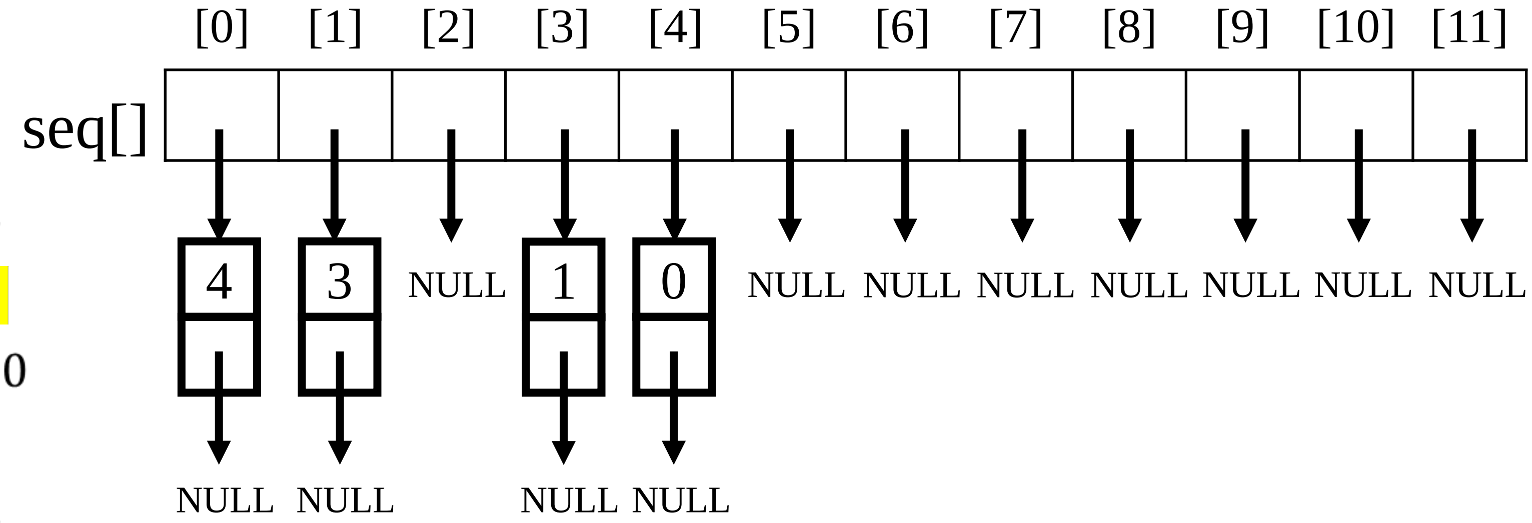
7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0



Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

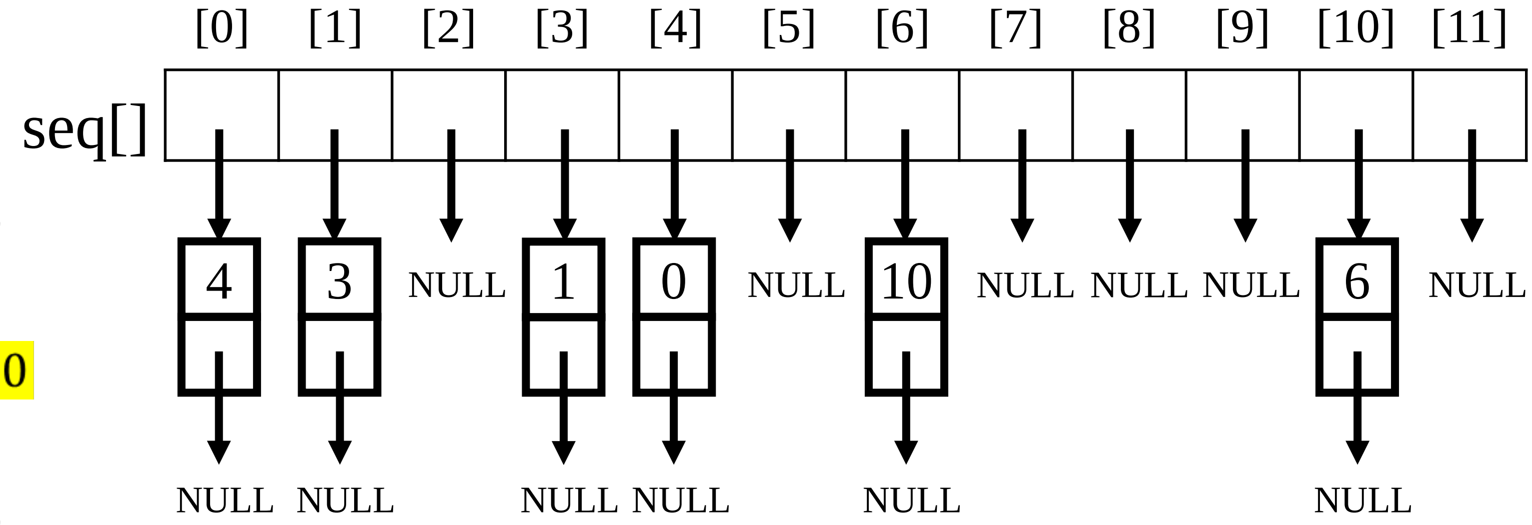
7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0



Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

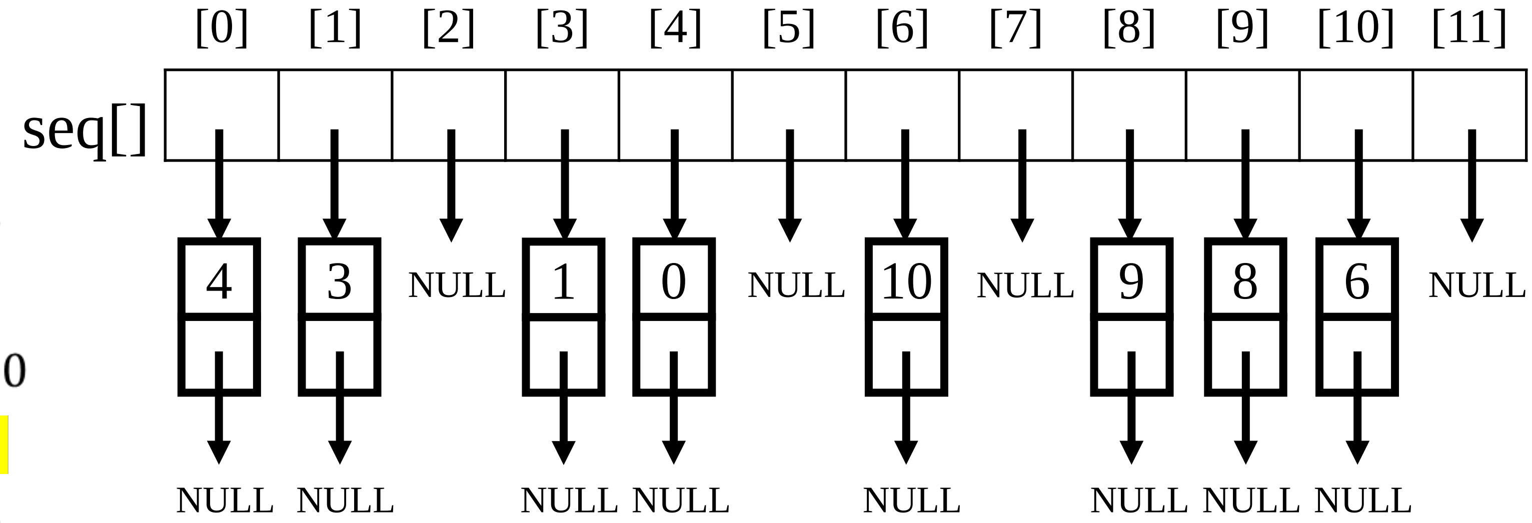
7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0



Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

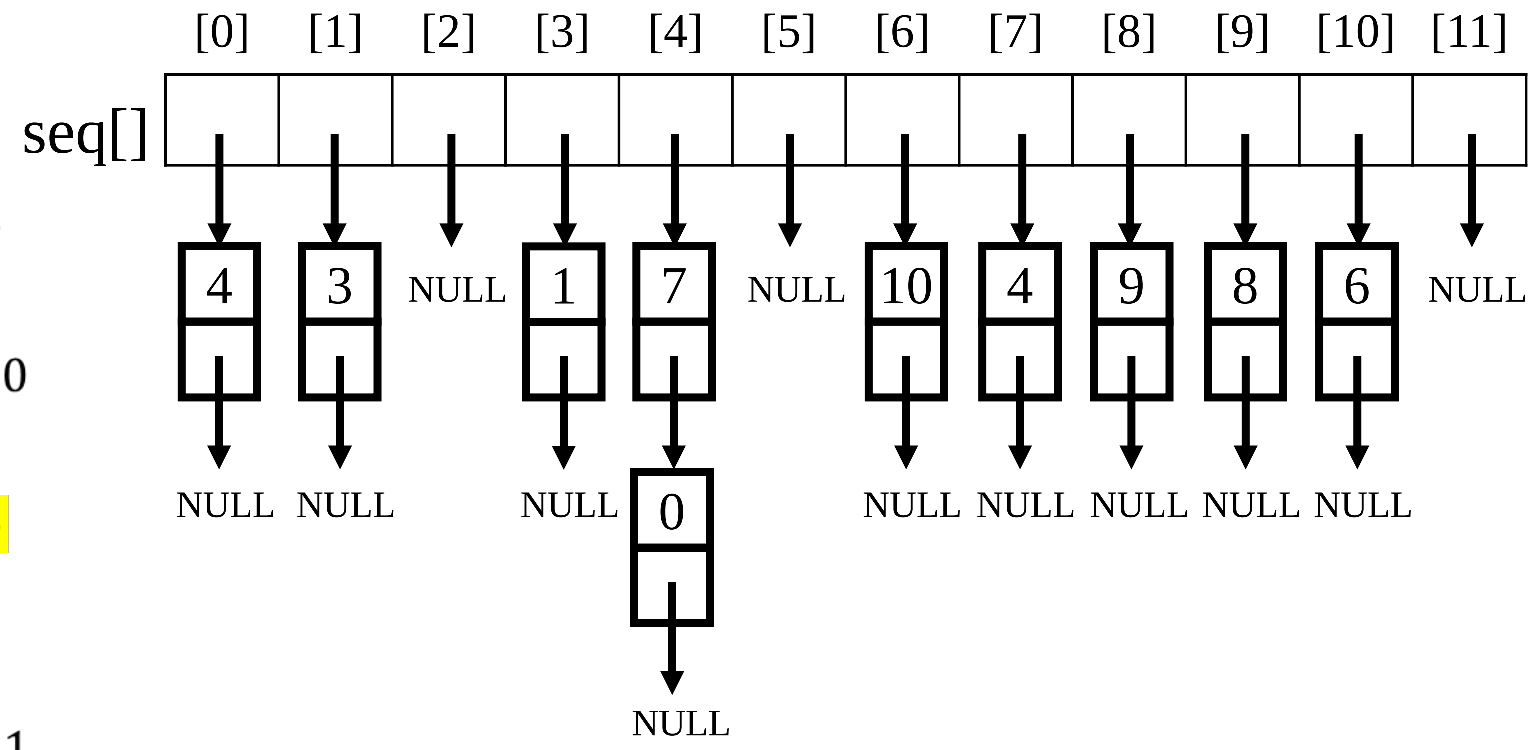
7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0



[0] [1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]

seq[]

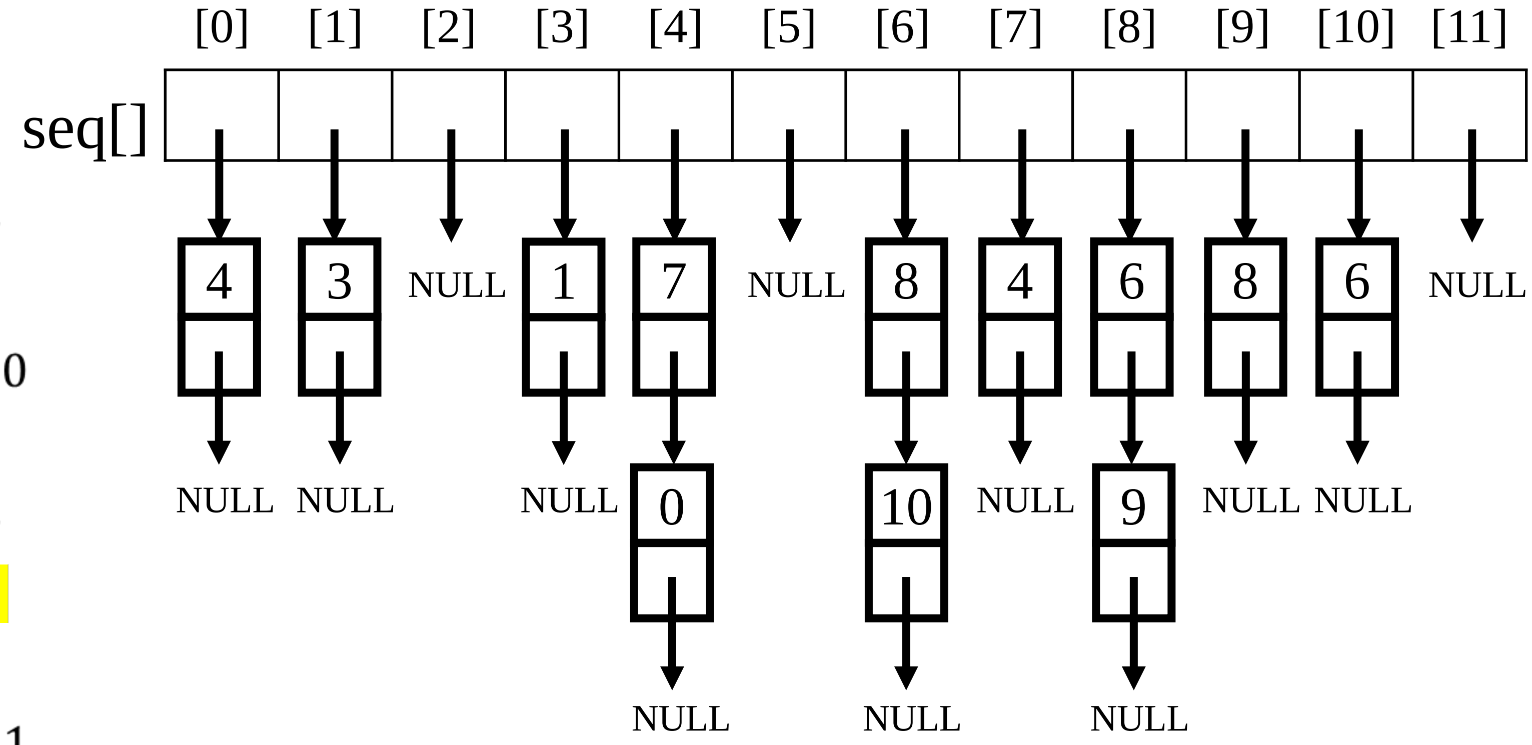
4 3 NULL 1 7 NULL 10 4 9 8 6 NULL

NULL NULL NULL 0 NULL NULL NULL NULL NULL

NULL

Equivalence relations

Phase 1: input the equivalence pairs



0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0

Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

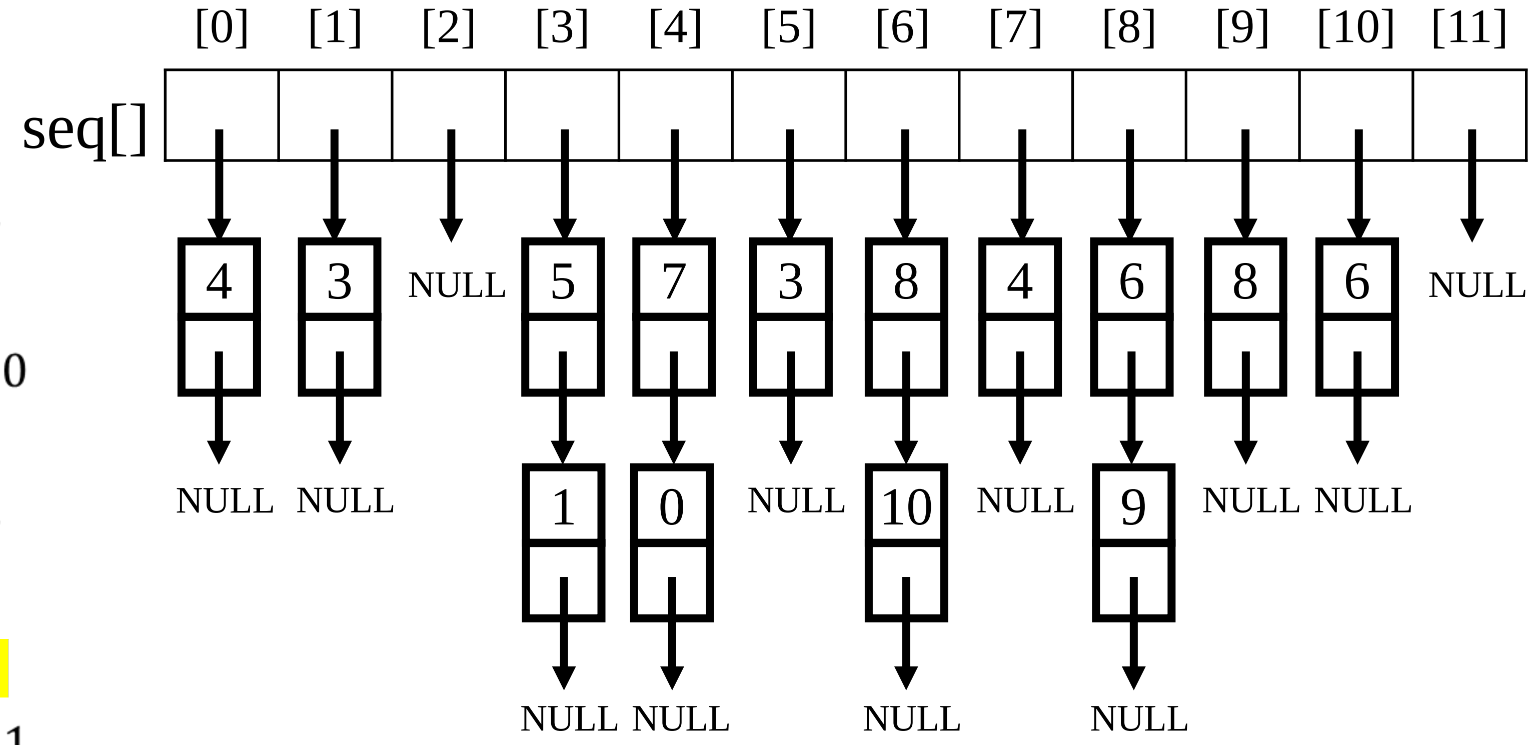
7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0



Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

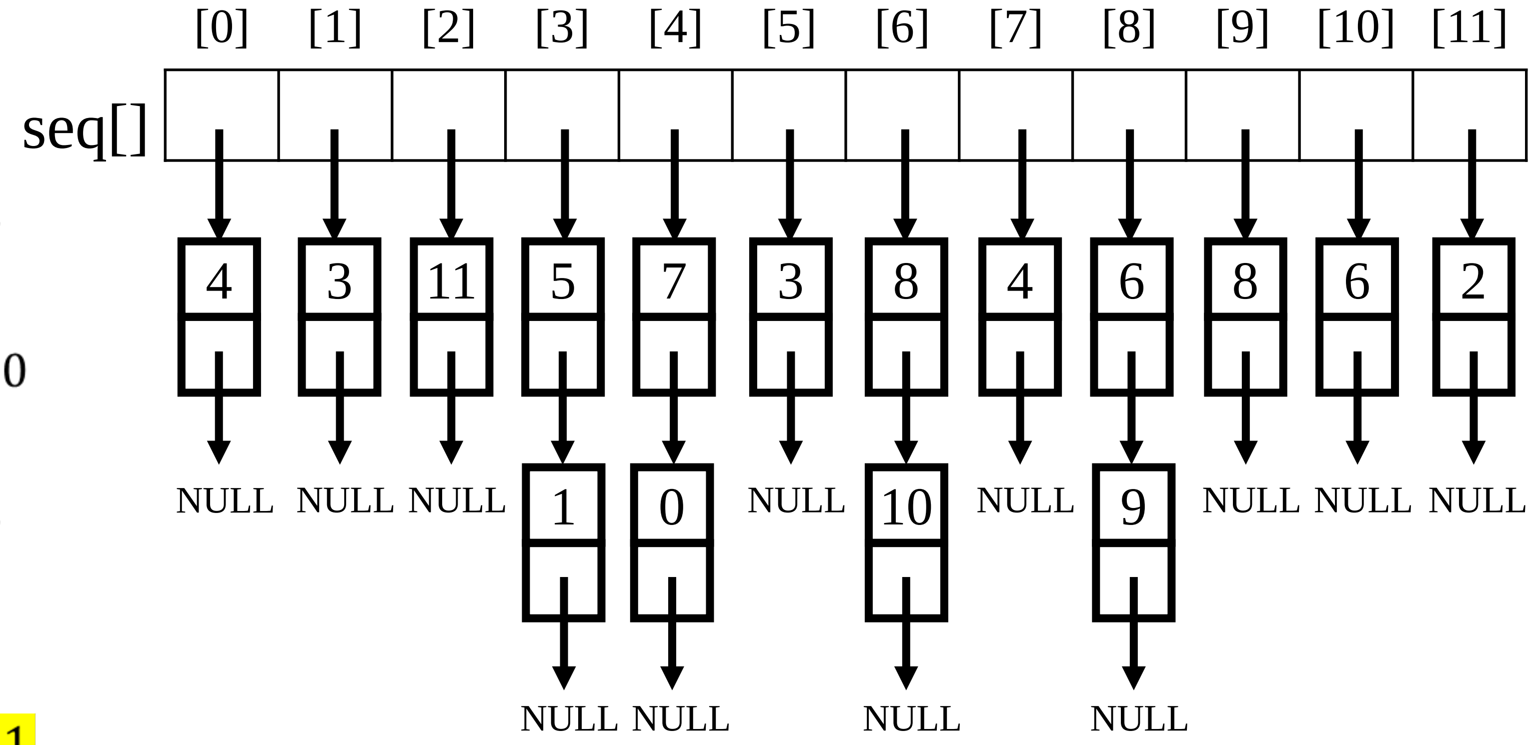
7 ≡ 4

6 ≡ 8

3 ≡ 5

2 ≡ 11

11 ≡ 0



Equivalence relations

Phase 1: input the equivalence pairs

0 ≡ 4

3 ≡ 1

6 ≡ 10

8 ≡ 9

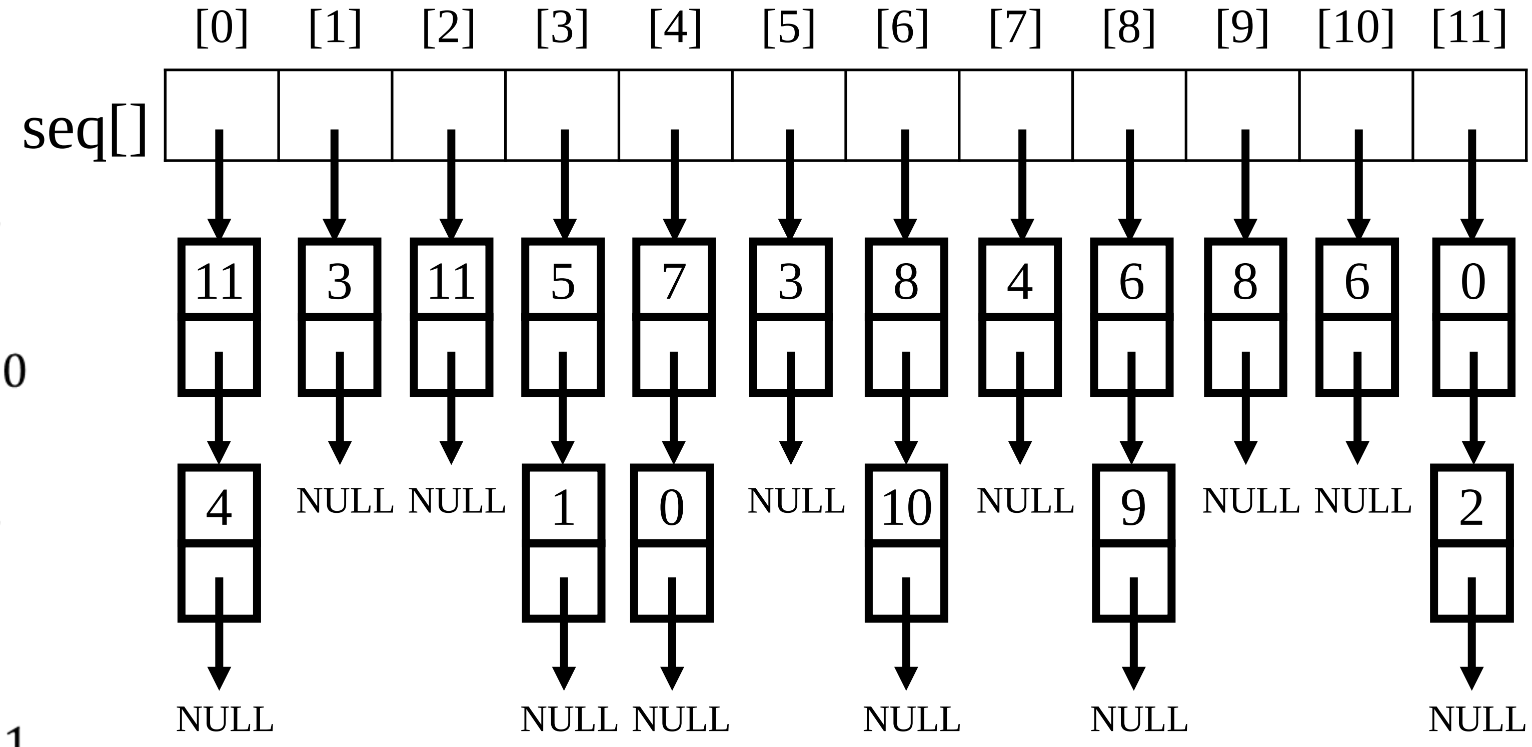
7 ≡ 4

6 ≡ 8

3 ≡ 5

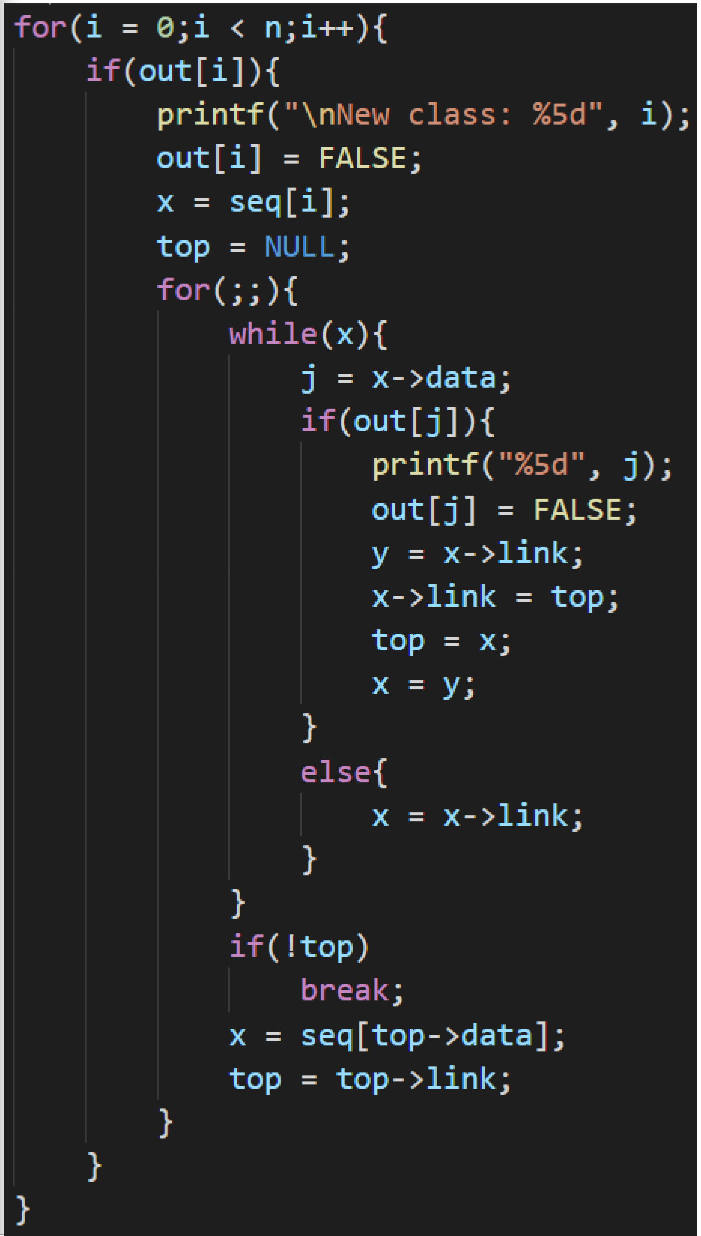
2 ≡ 11

11 ≡ 0



Equivalence relations

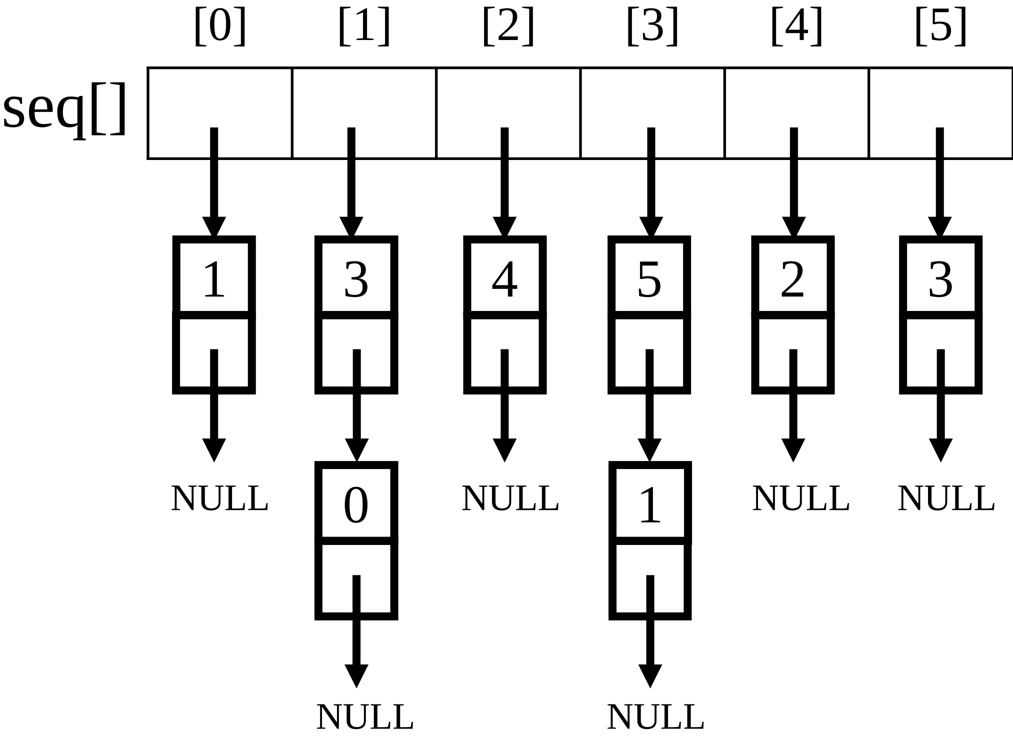
Phase 2: output the equivalence classes



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

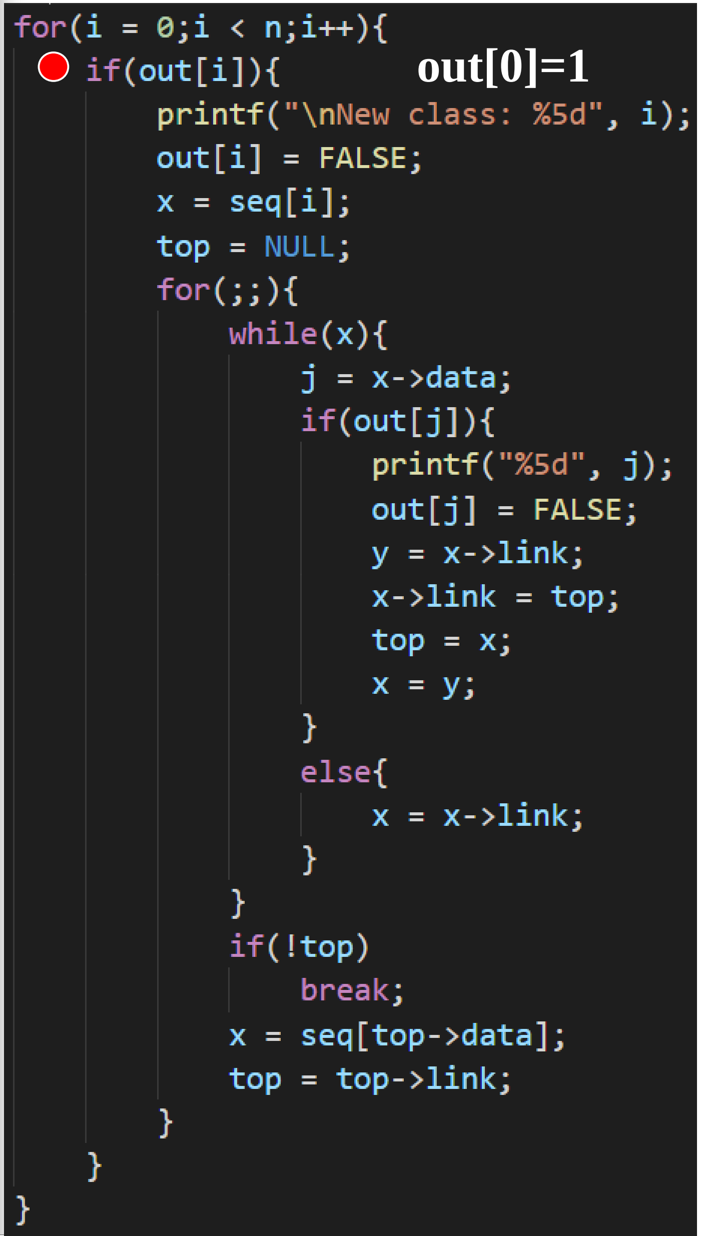
$i = 0$ $\quad 0 \equiv 1 \quad 1 \equiv 3 \quad 3 \equiv 5 \quad 2 \equiv 4$

	[0]	[1]	[2]	[3]	[4]	[5]
out[]	1	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes



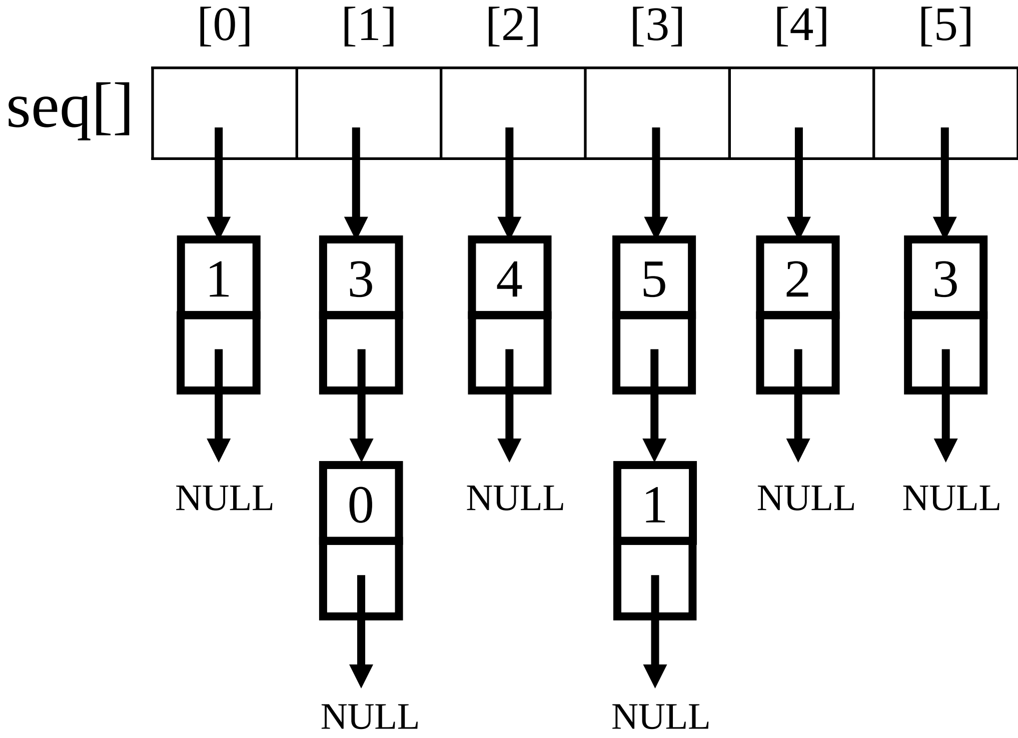
out[0]=1

```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0

$0 \equiv 1 \quad 1 \equiv 3 \quad 3 \equiv 5 \quad 2 \equiv 4$

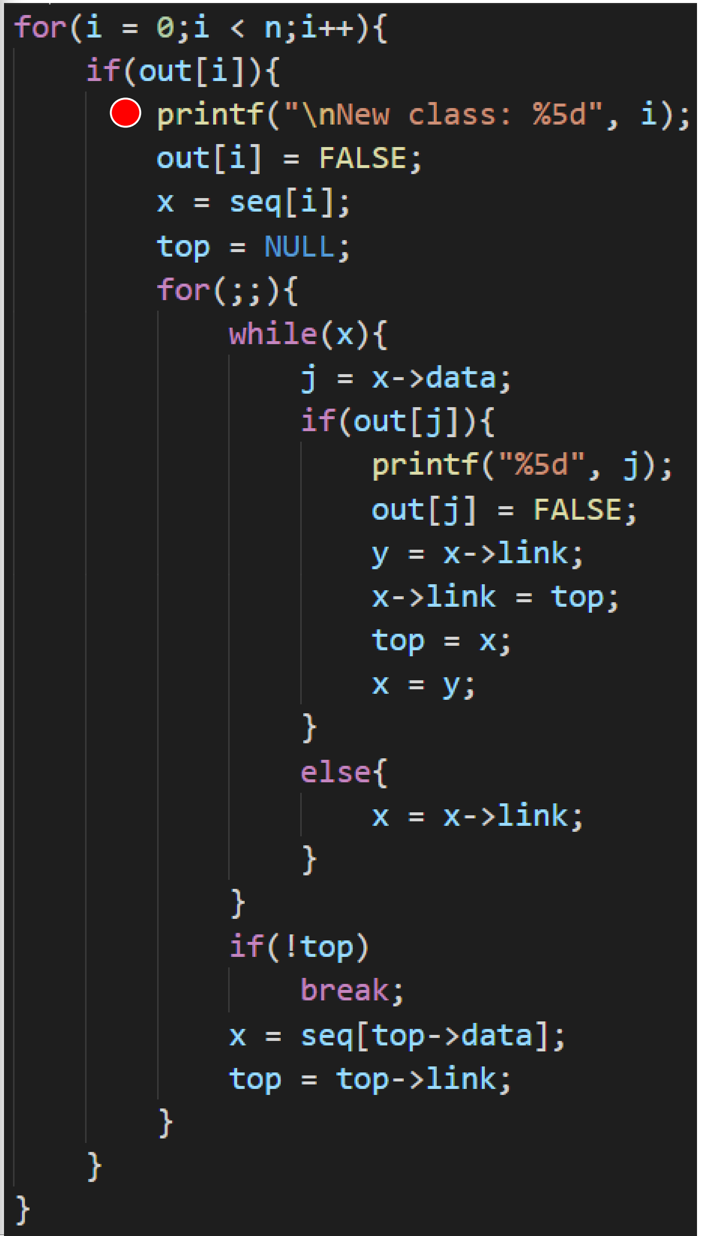
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	1	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

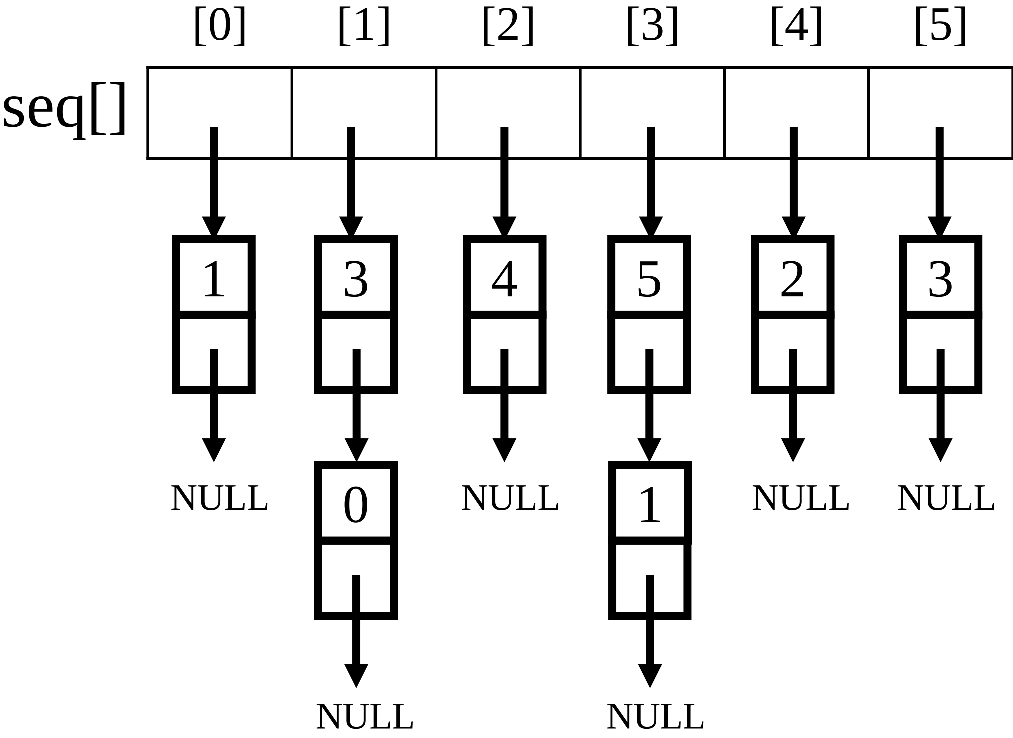
New class: 0



```c
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

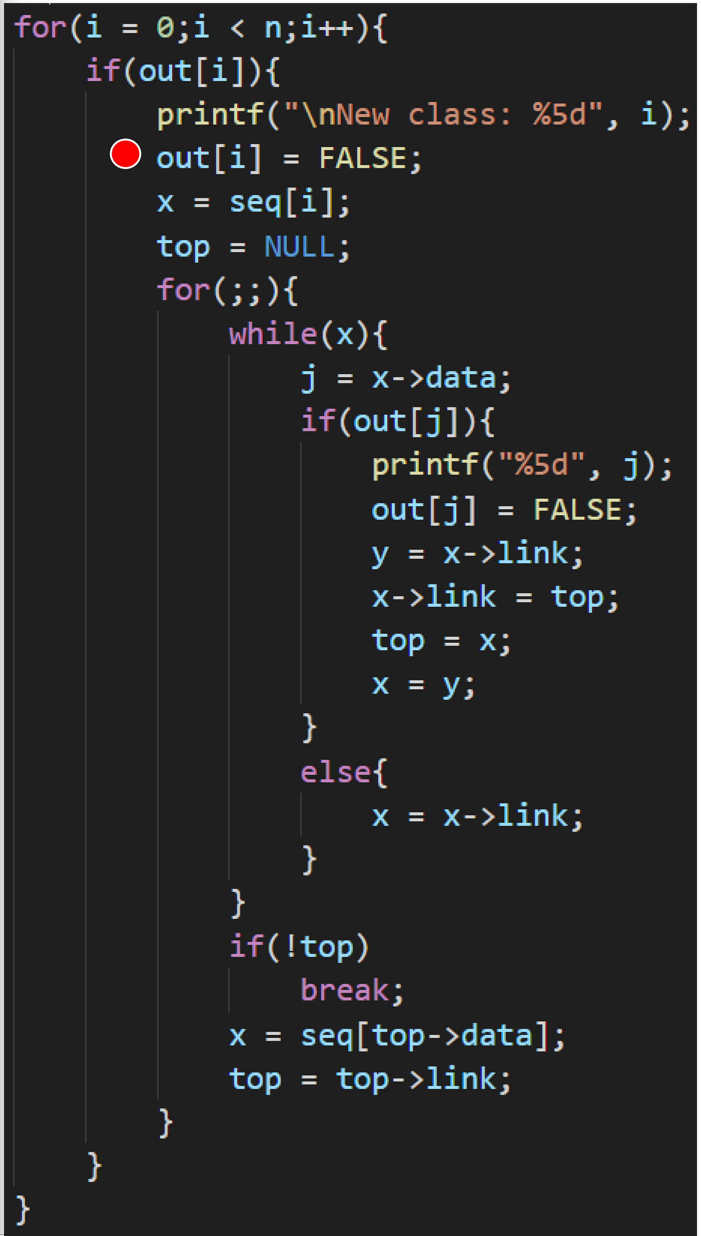
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	1	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

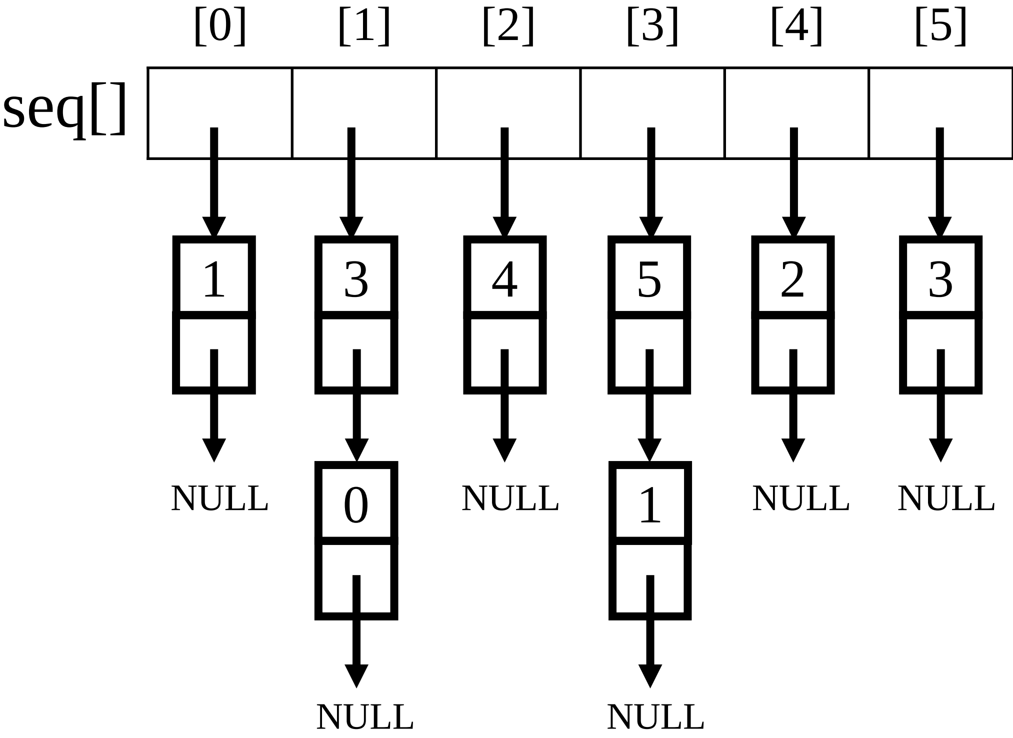
New class: 0



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

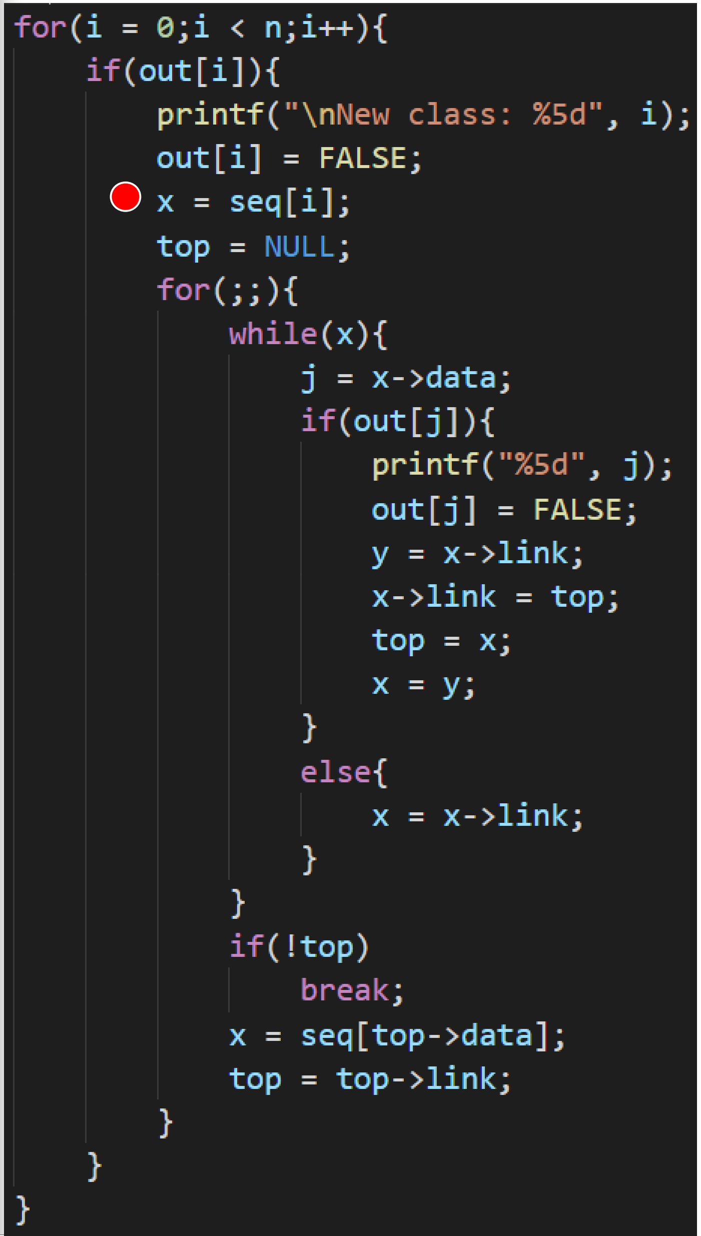
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

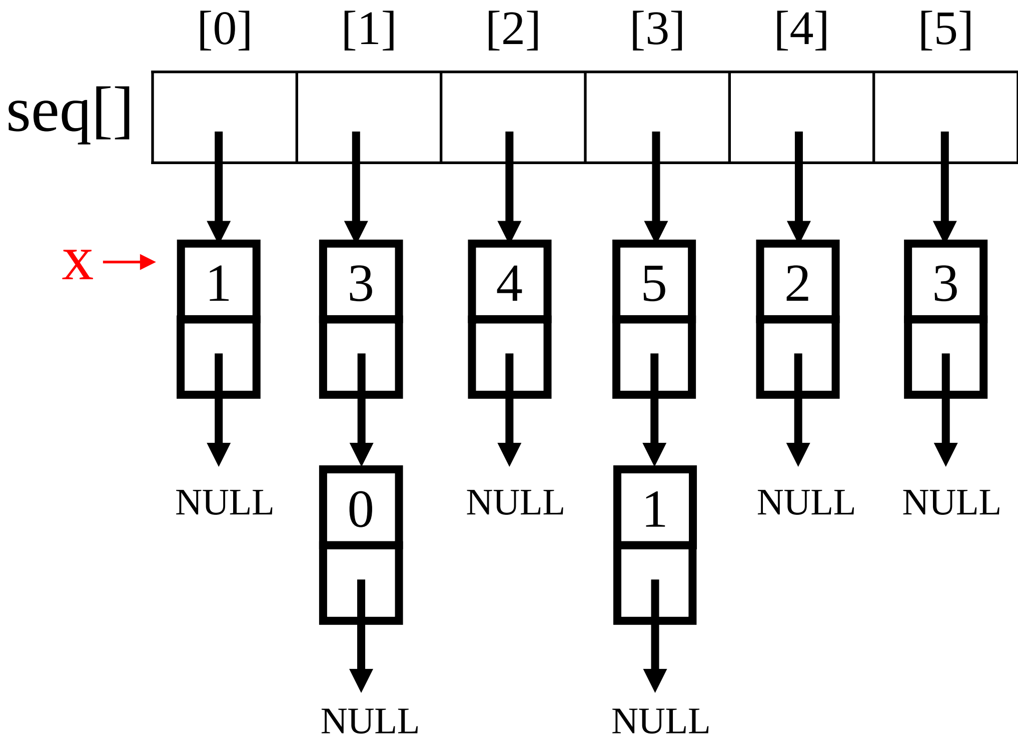
New class: 0



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

$i = 0$ $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

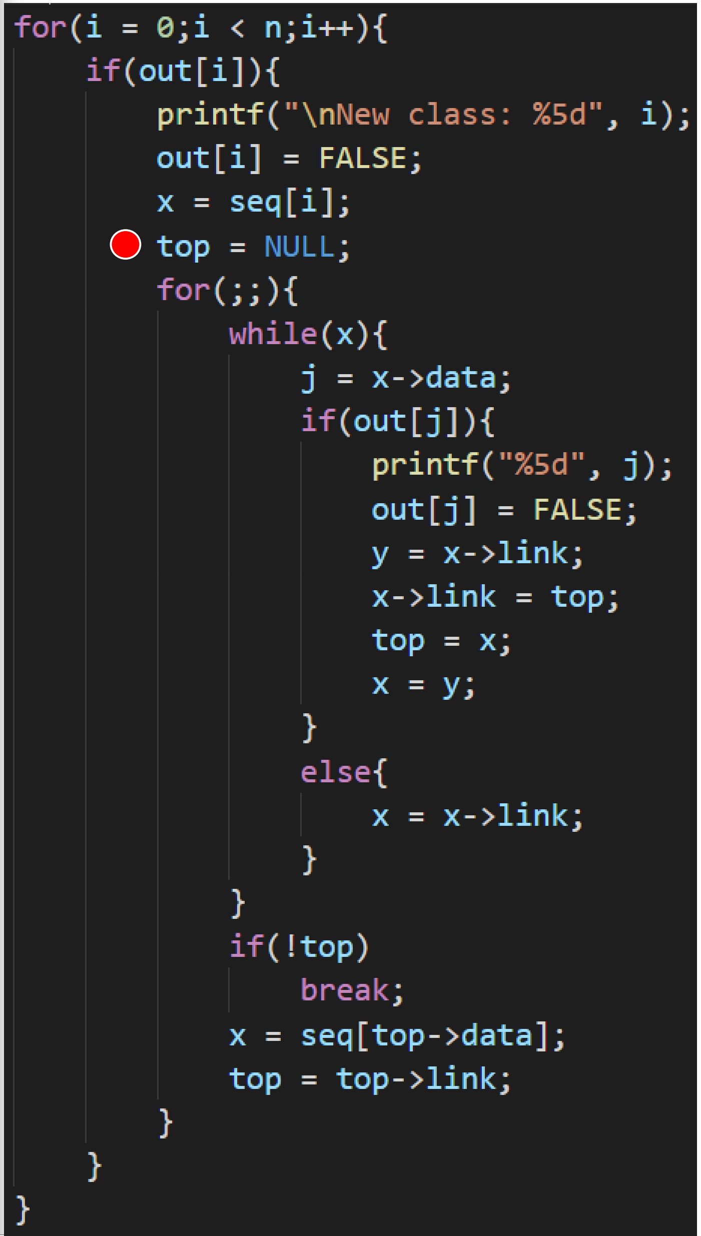
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

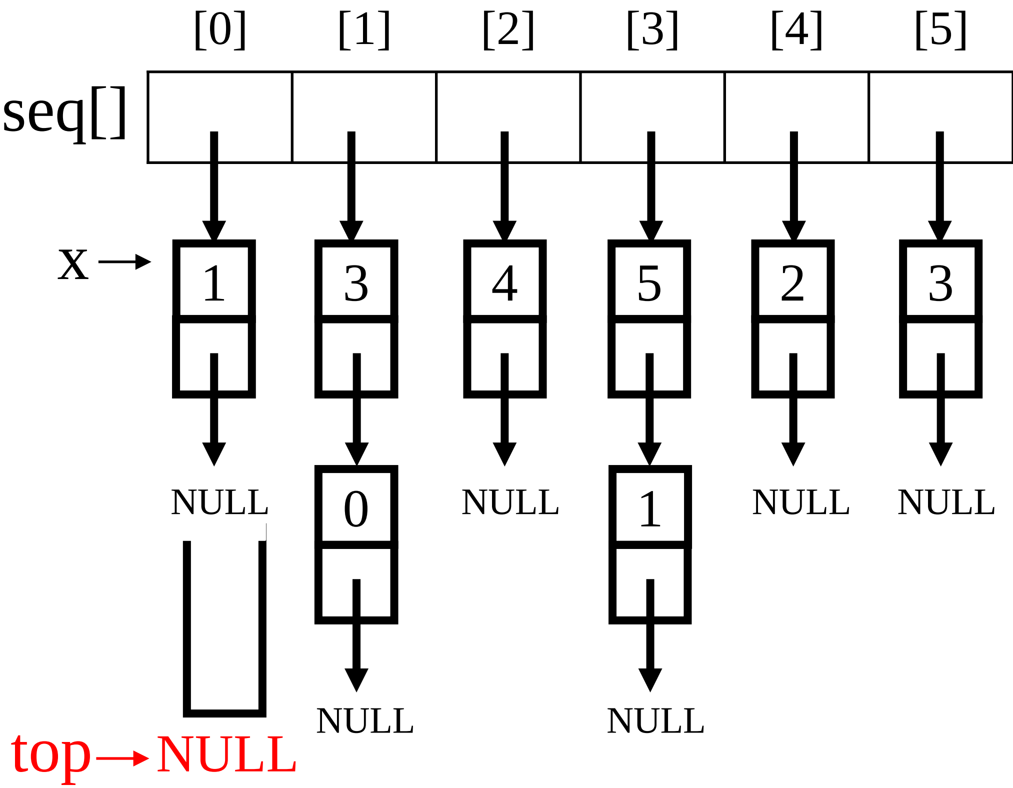
New class: 0



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

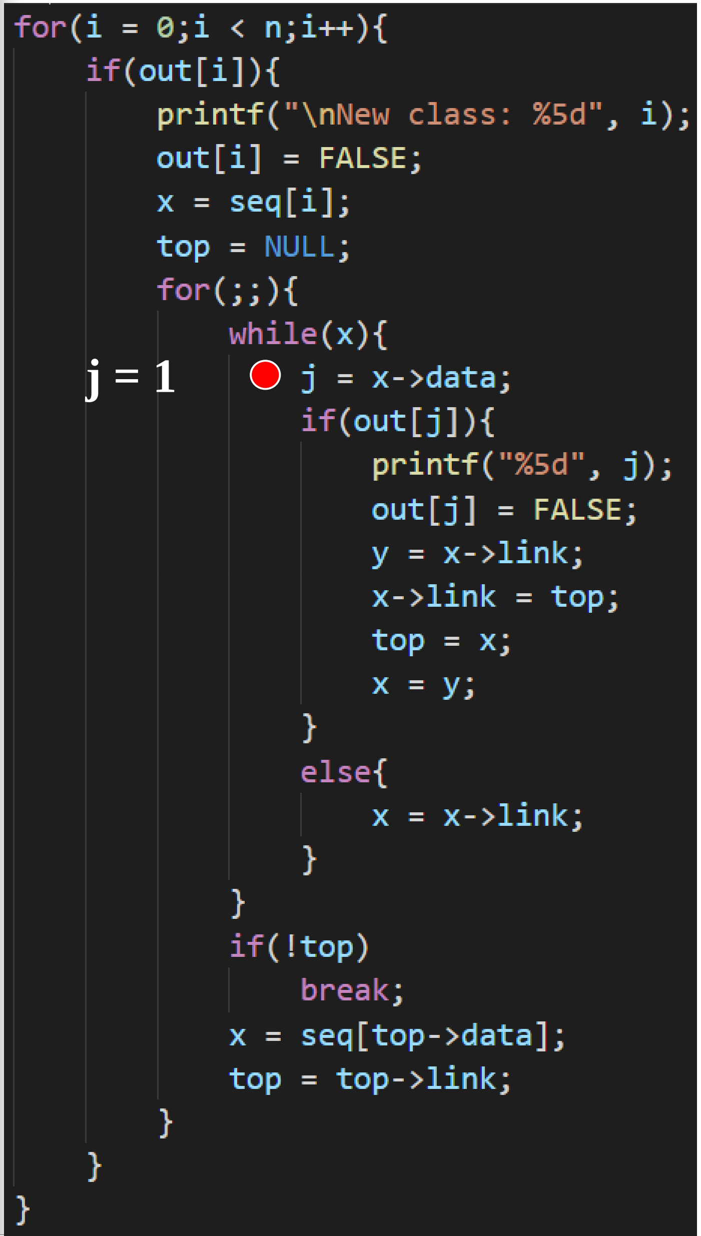
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0

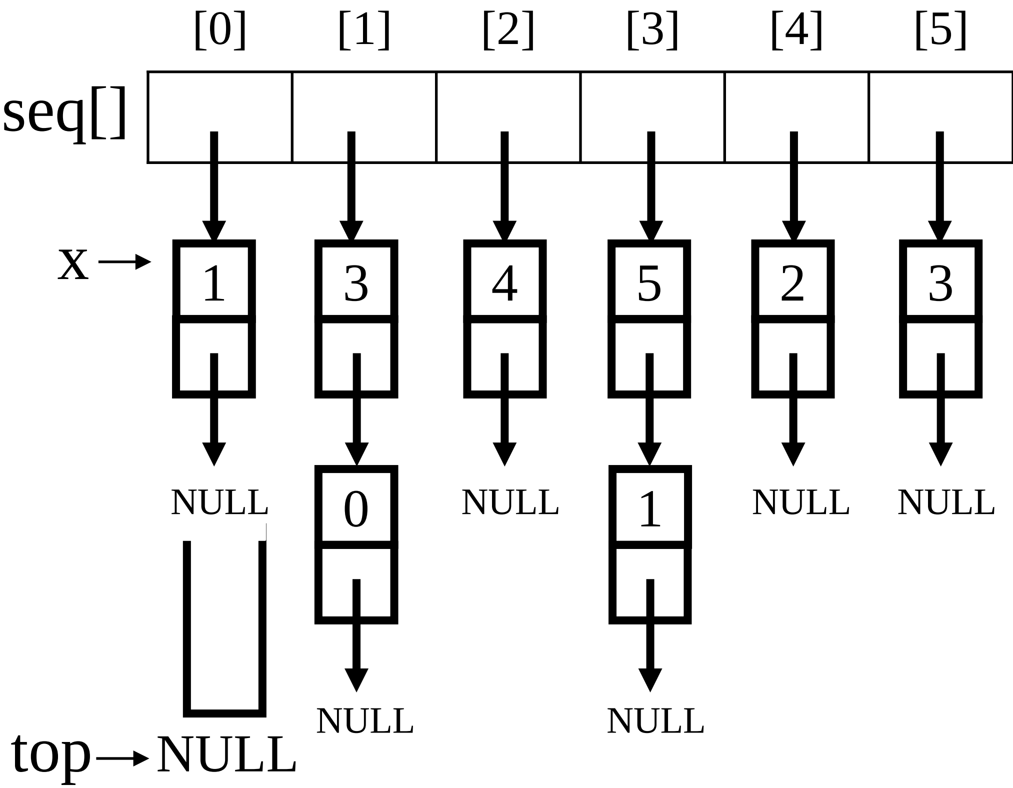


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

j = 1

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

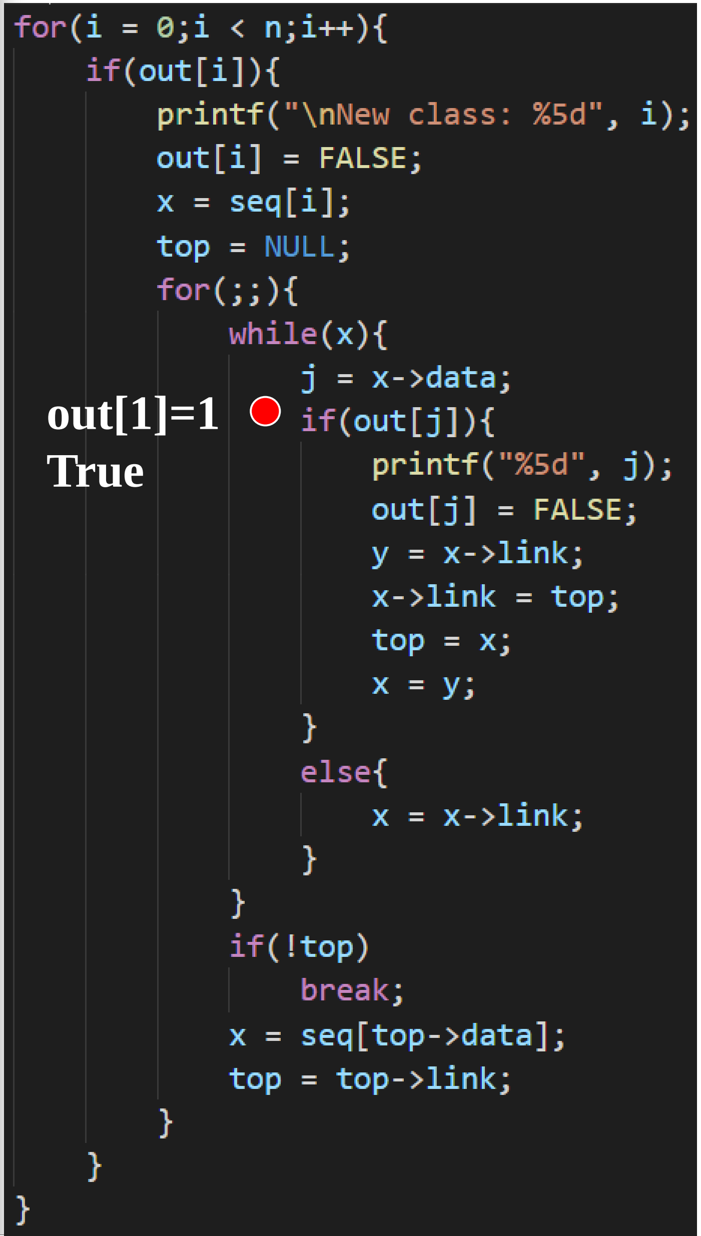
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0



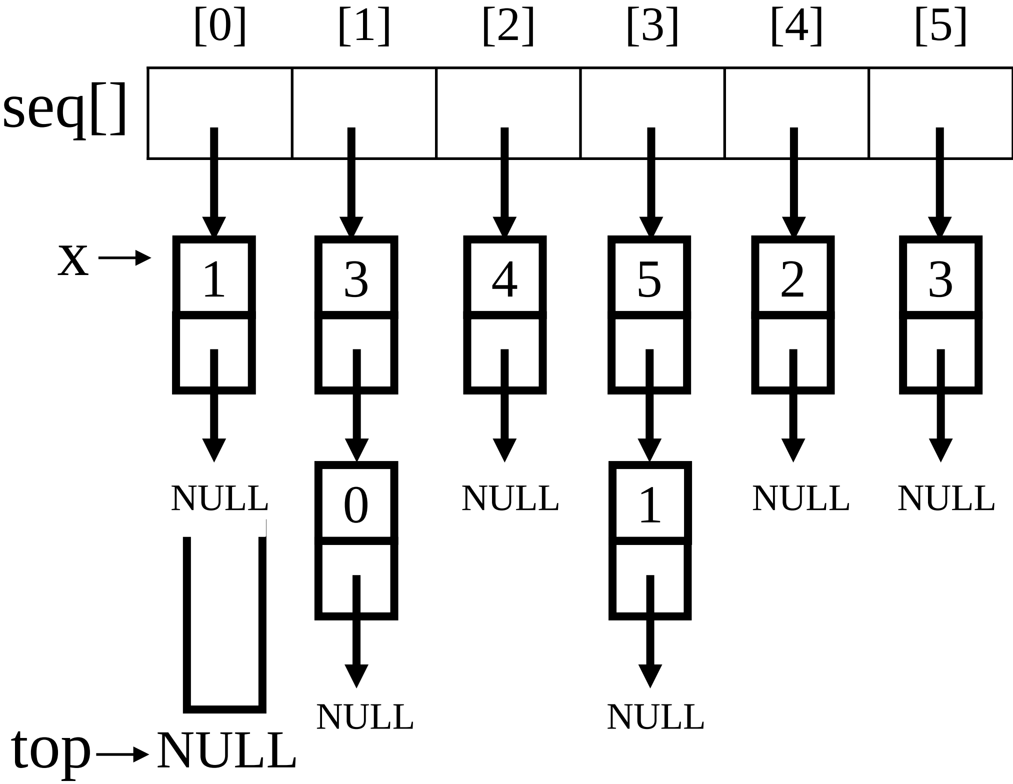
```c
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[1]=1

True

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

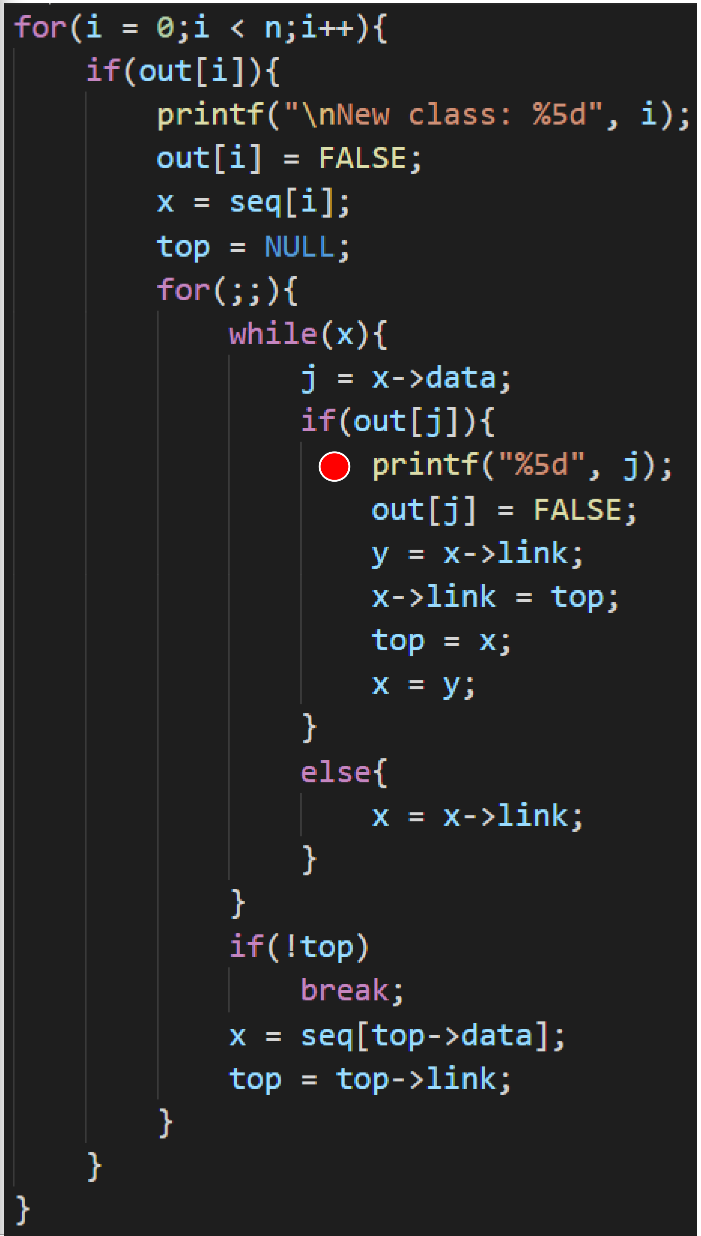
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

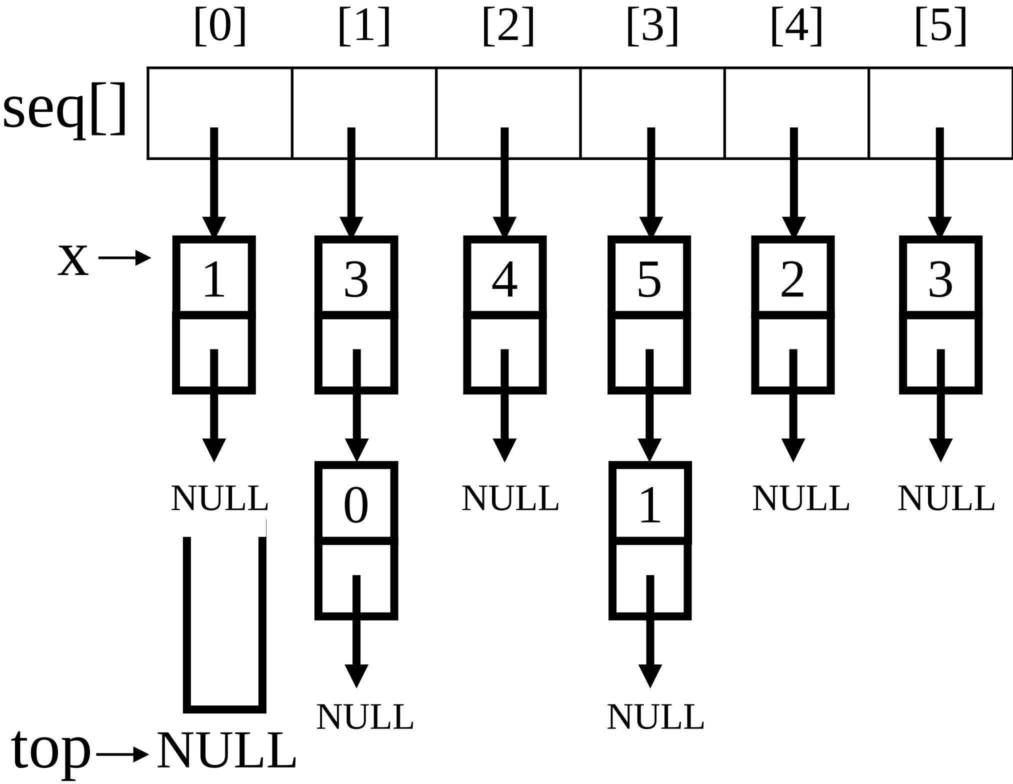
New class: 0 1



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0 0 ≡ 1 1 ≡ 3 3 ≡ 5 2 ≡ 4

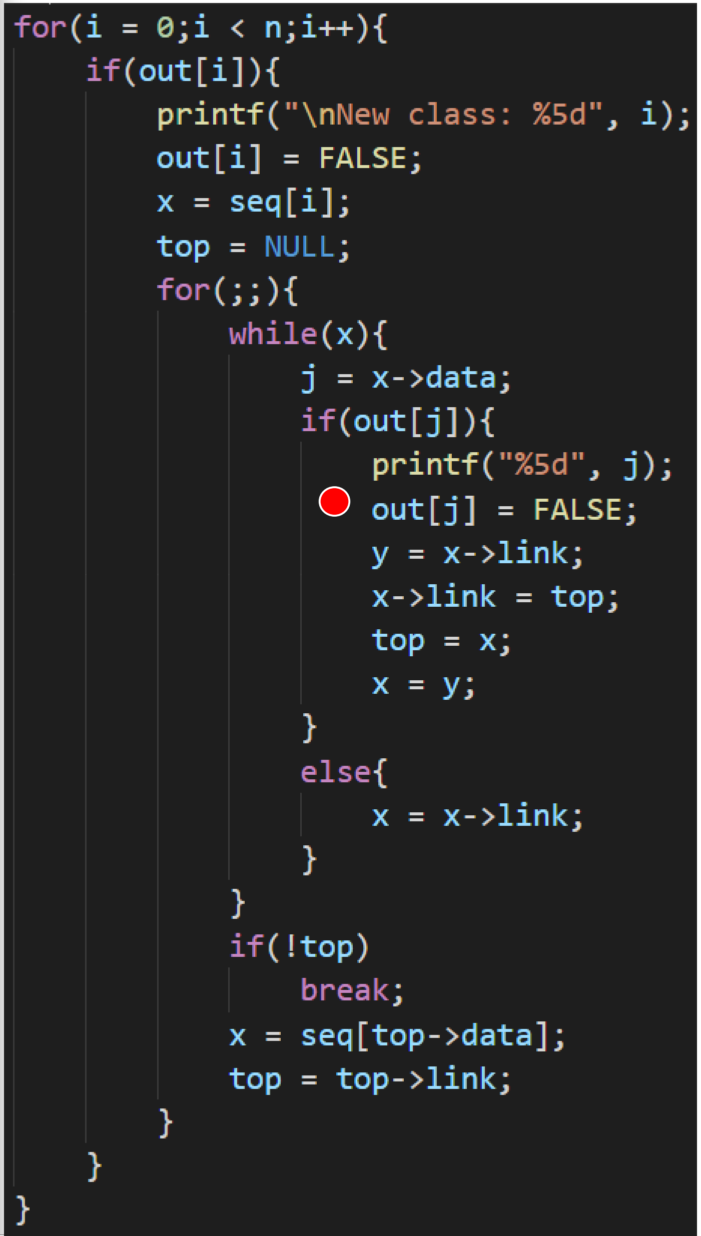
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	1	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

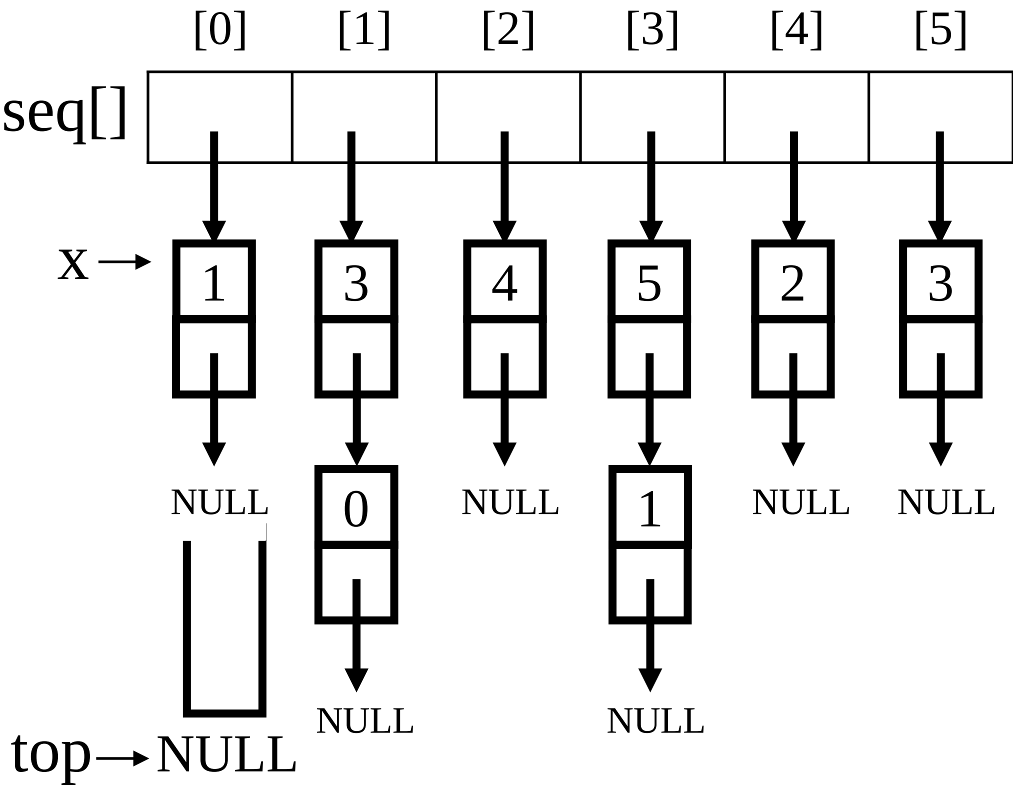
New class: 0 1



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

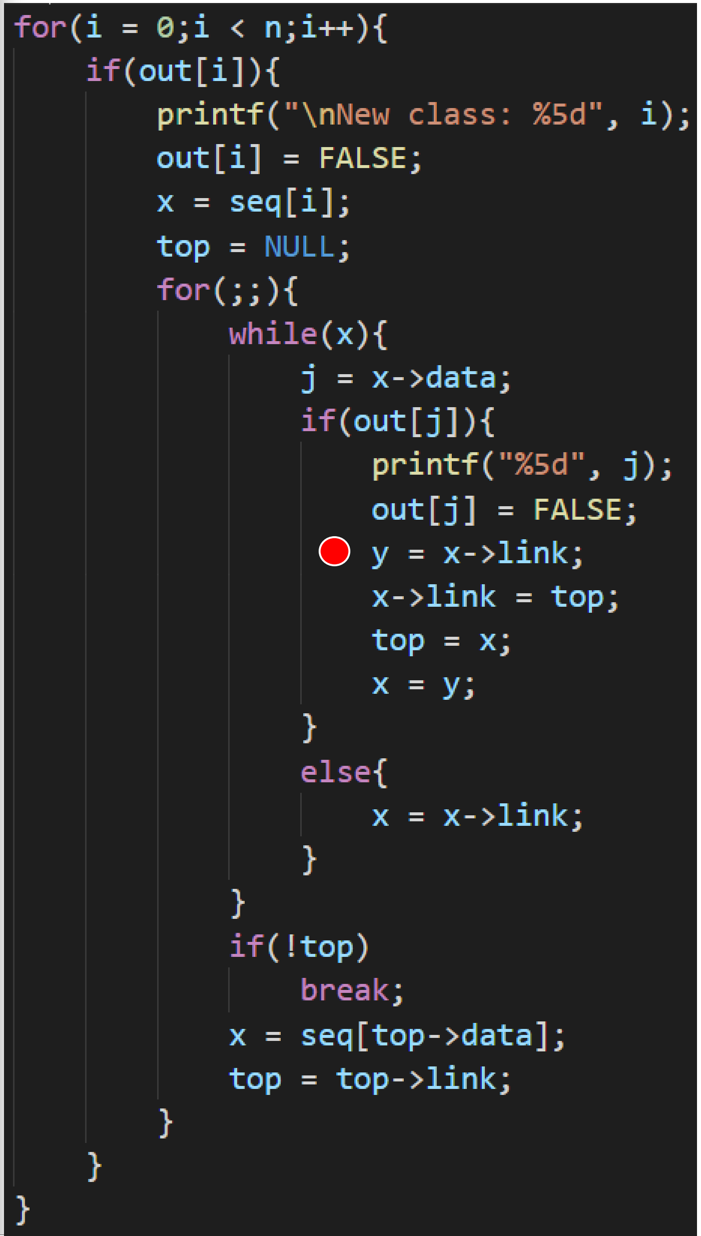
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

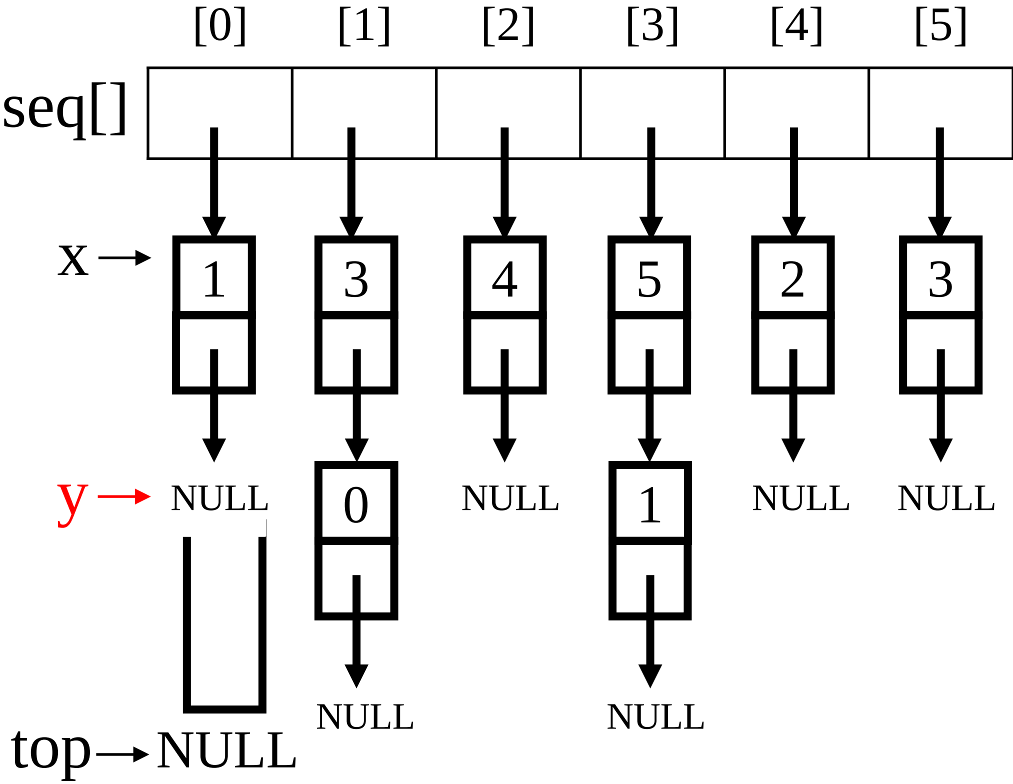
New class: 0 1



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

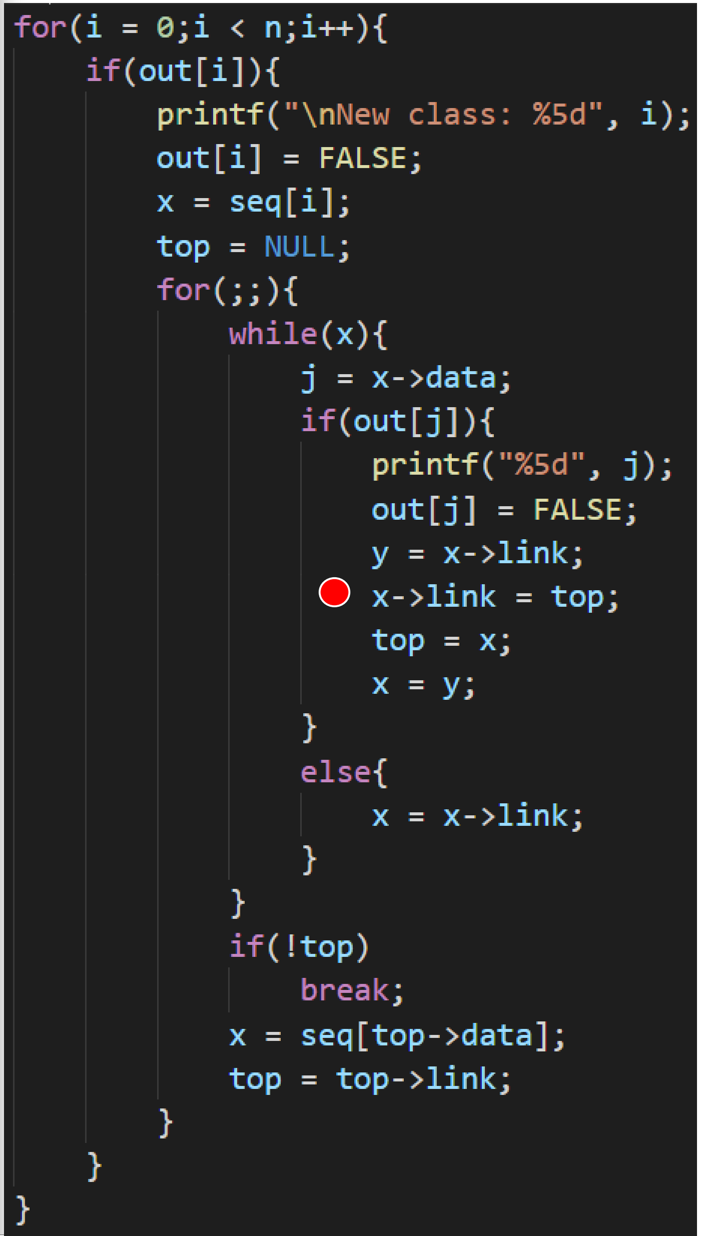
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

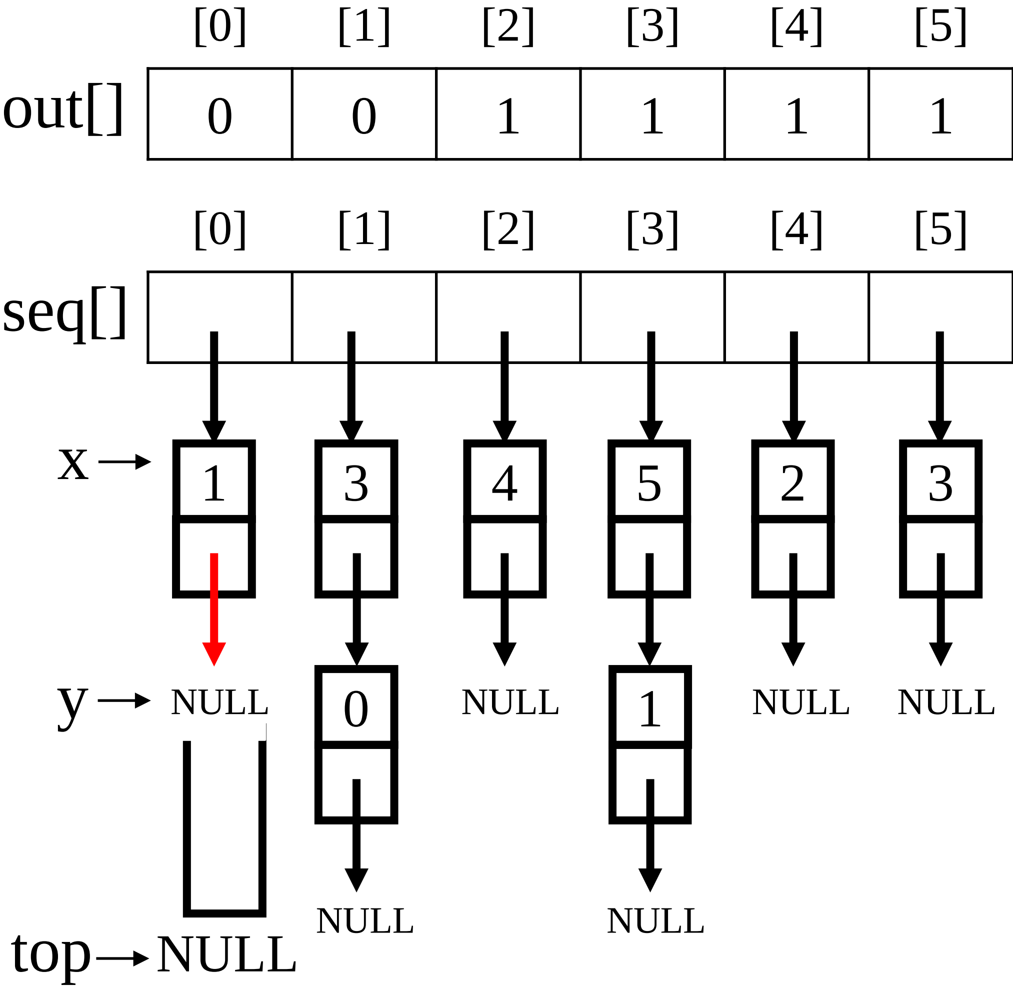
Phase 2: output the equivalence classes

New class: 0 1



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

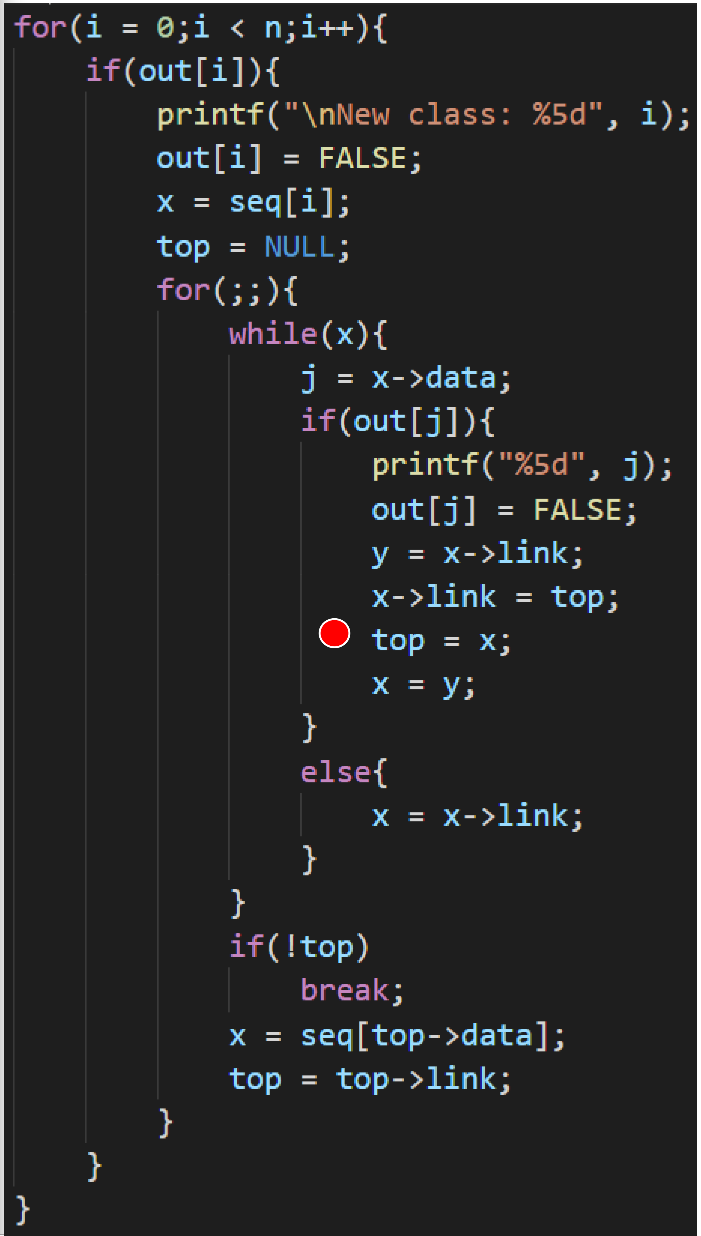


	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1

Equivalence relations

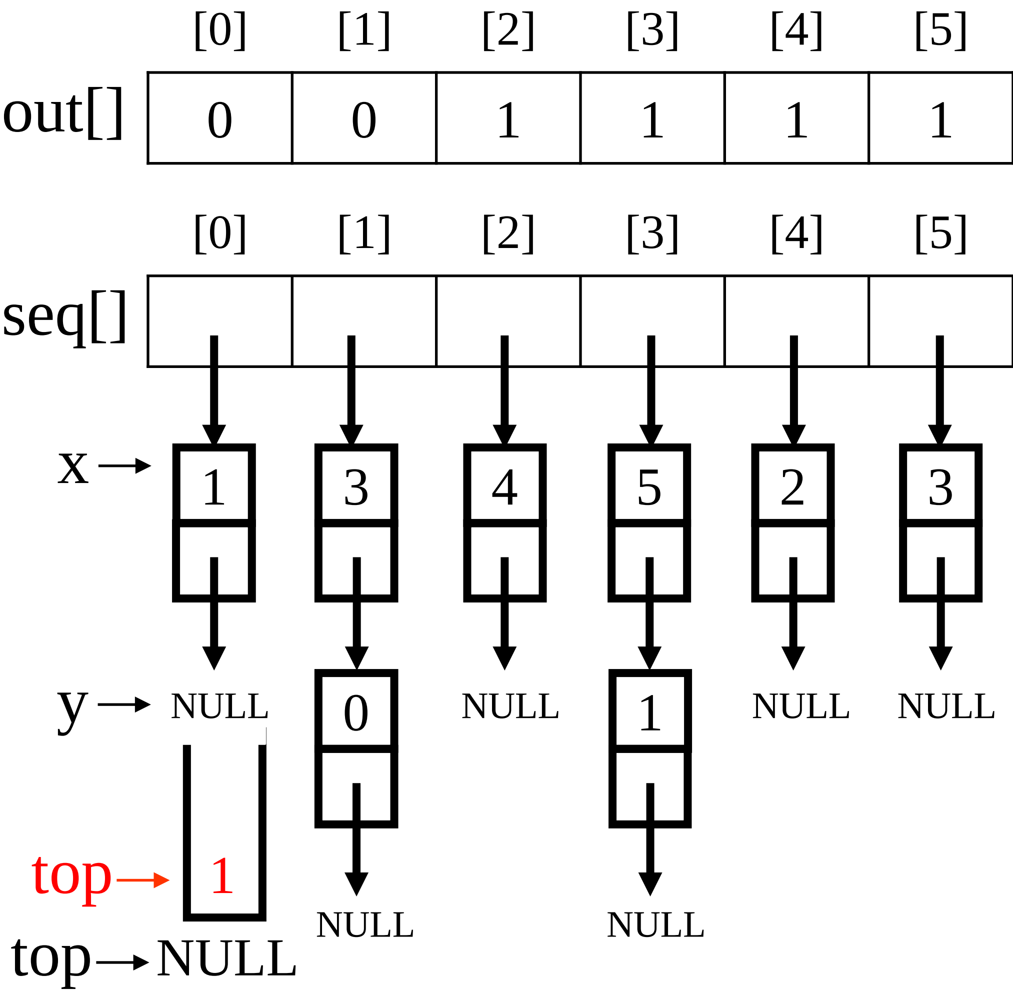
Phase 2: output the equivalence classes

```
New class:    0    1
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

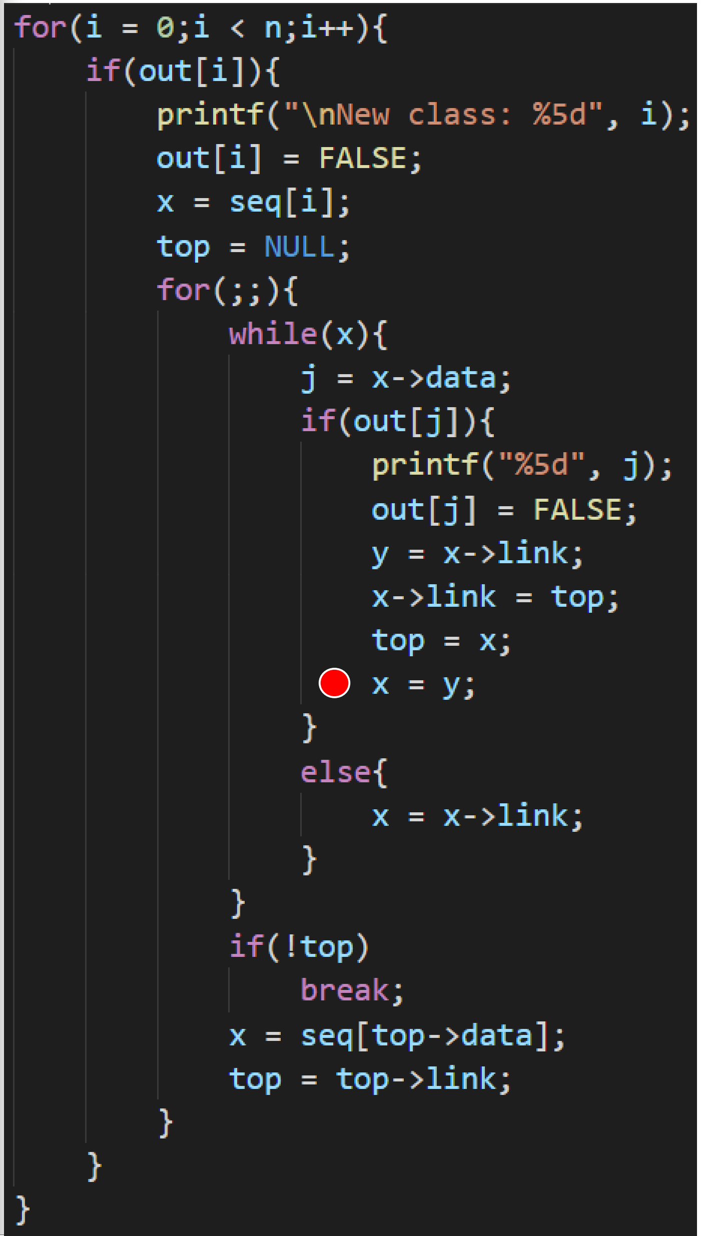


	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1

Equivalence relations

Phase 2: output the equivalence classes

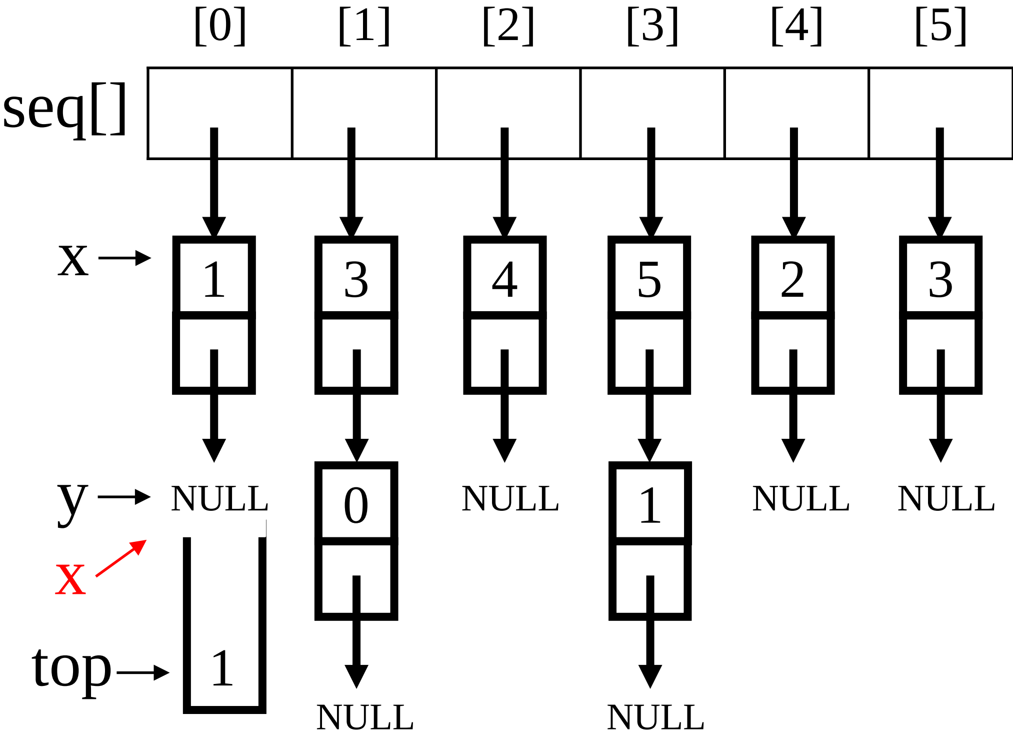
New class: 0 1



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

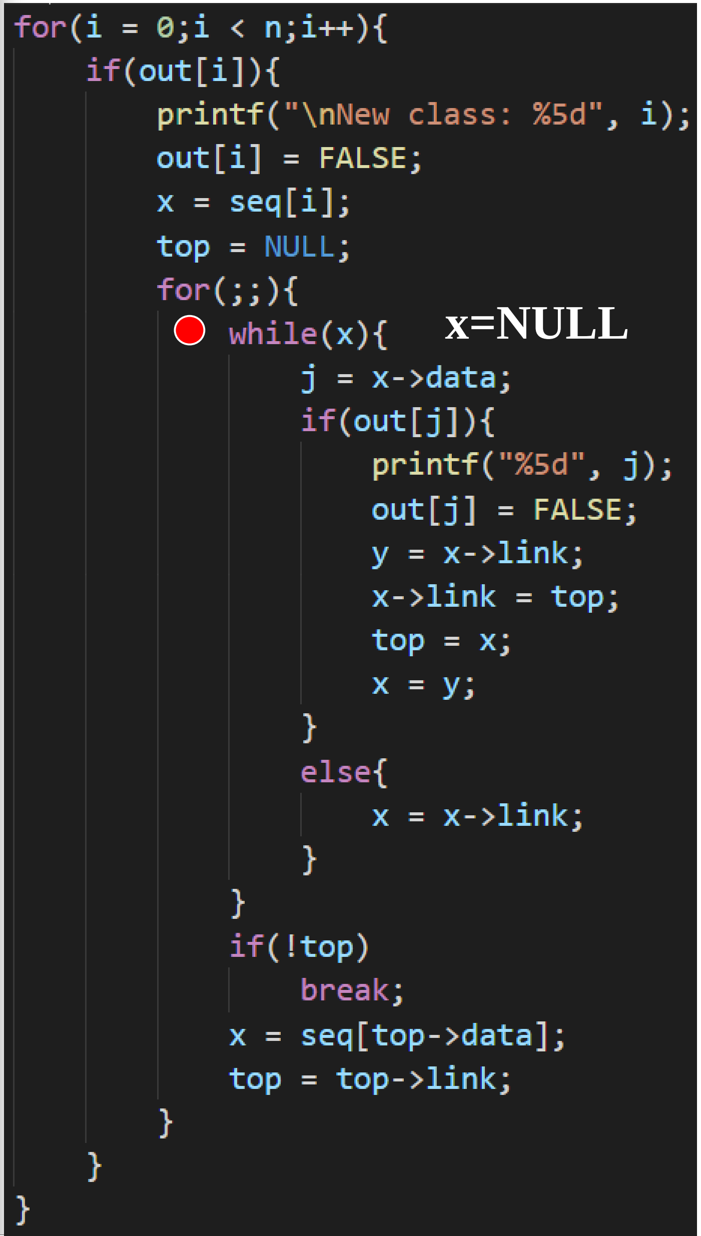
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1

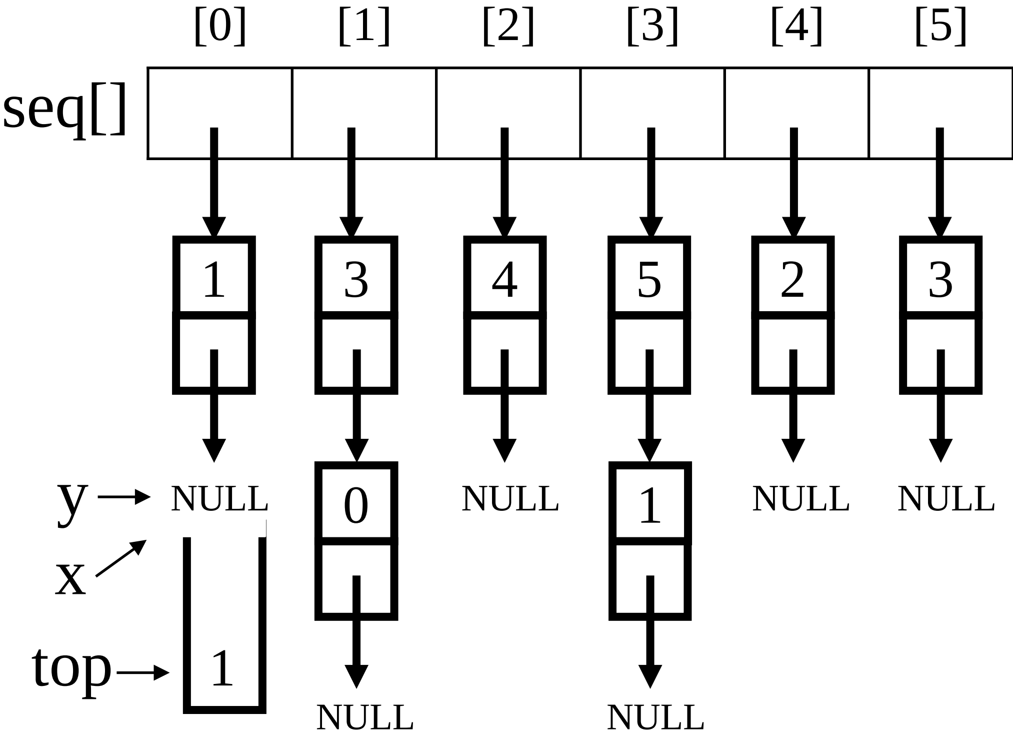


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

x=NULL

i = 0 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

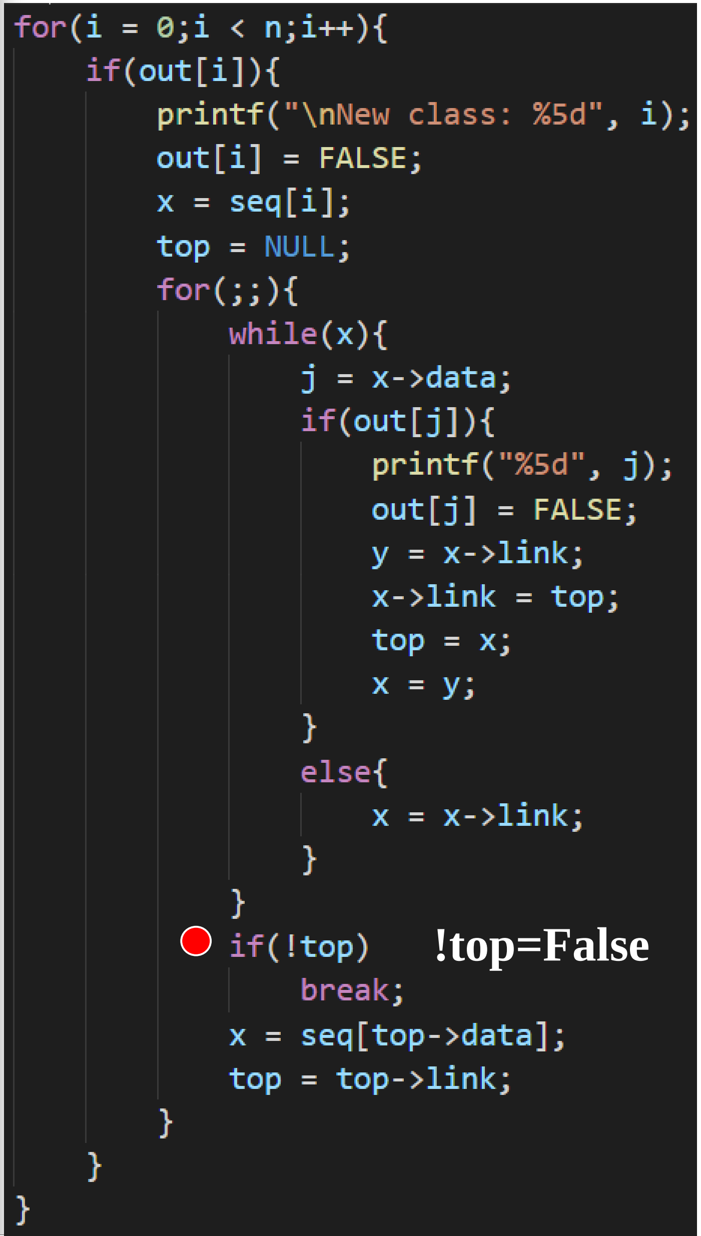
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1

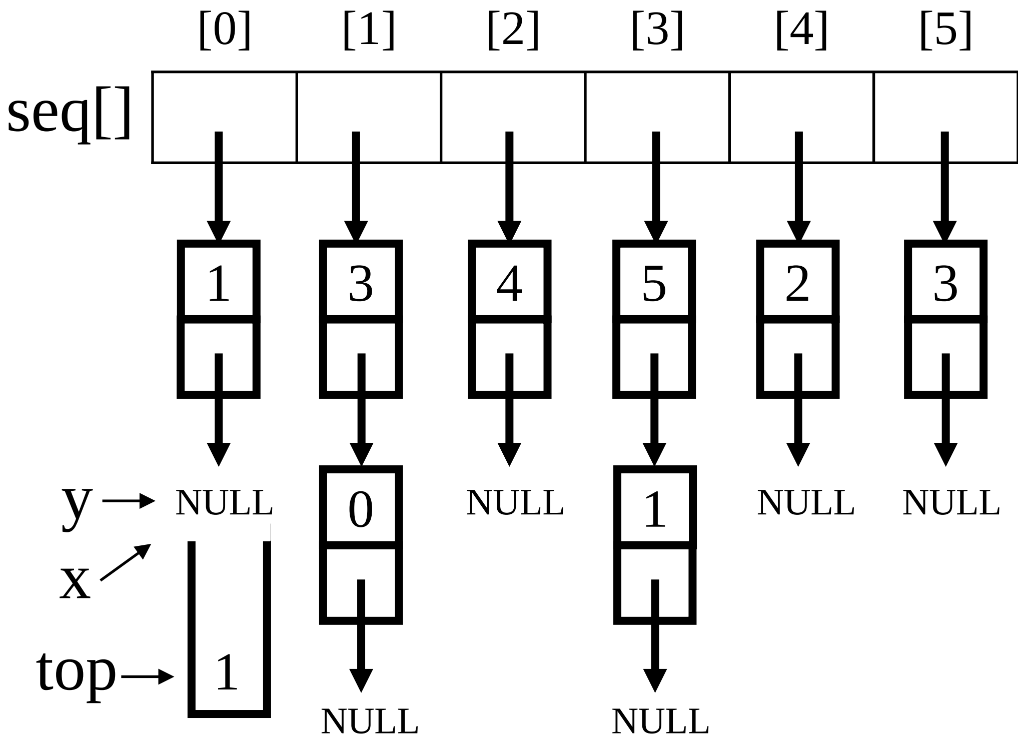


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

!top=False

i = 0 0 ≡ 1 1 ≡ 3 3 ≡ 5 2 ≡ 4

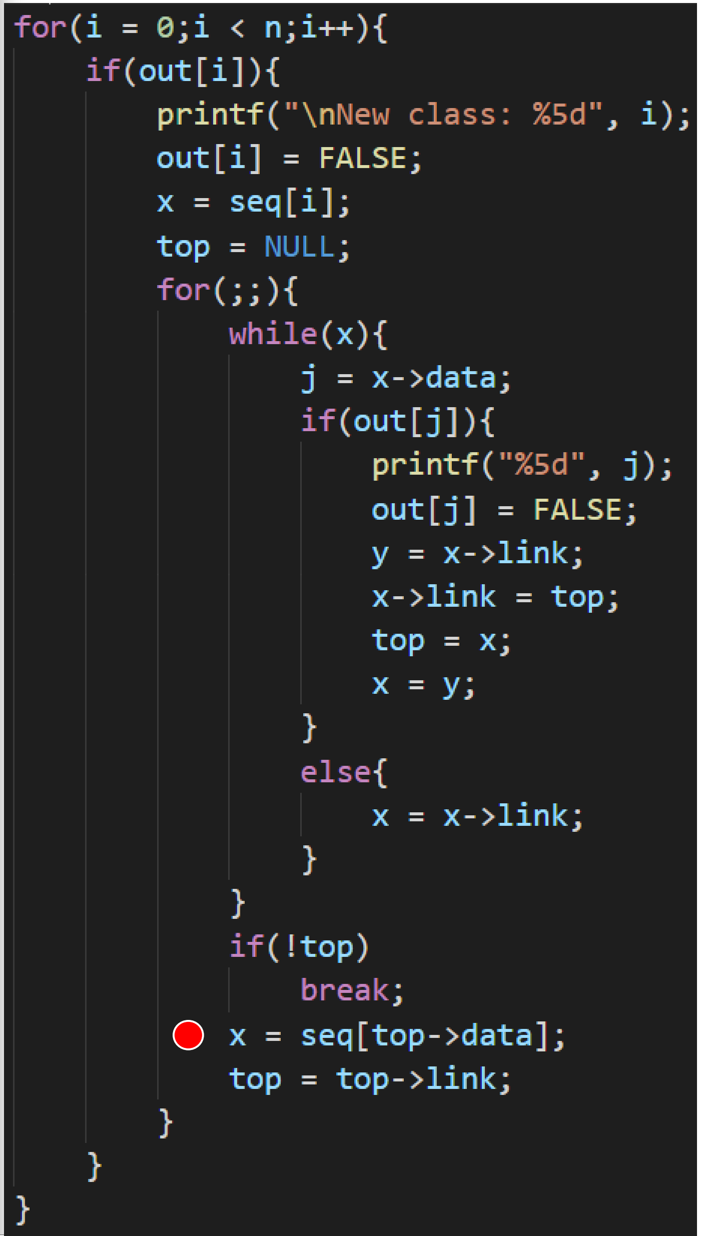
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

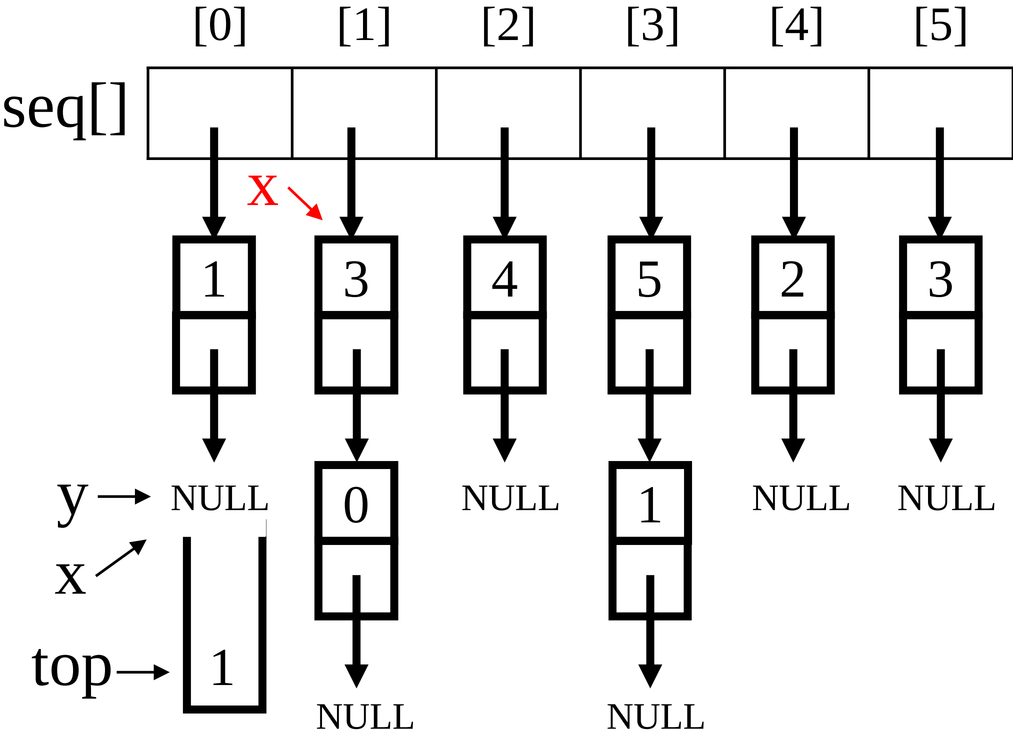
New class: 0 1



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
●           x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

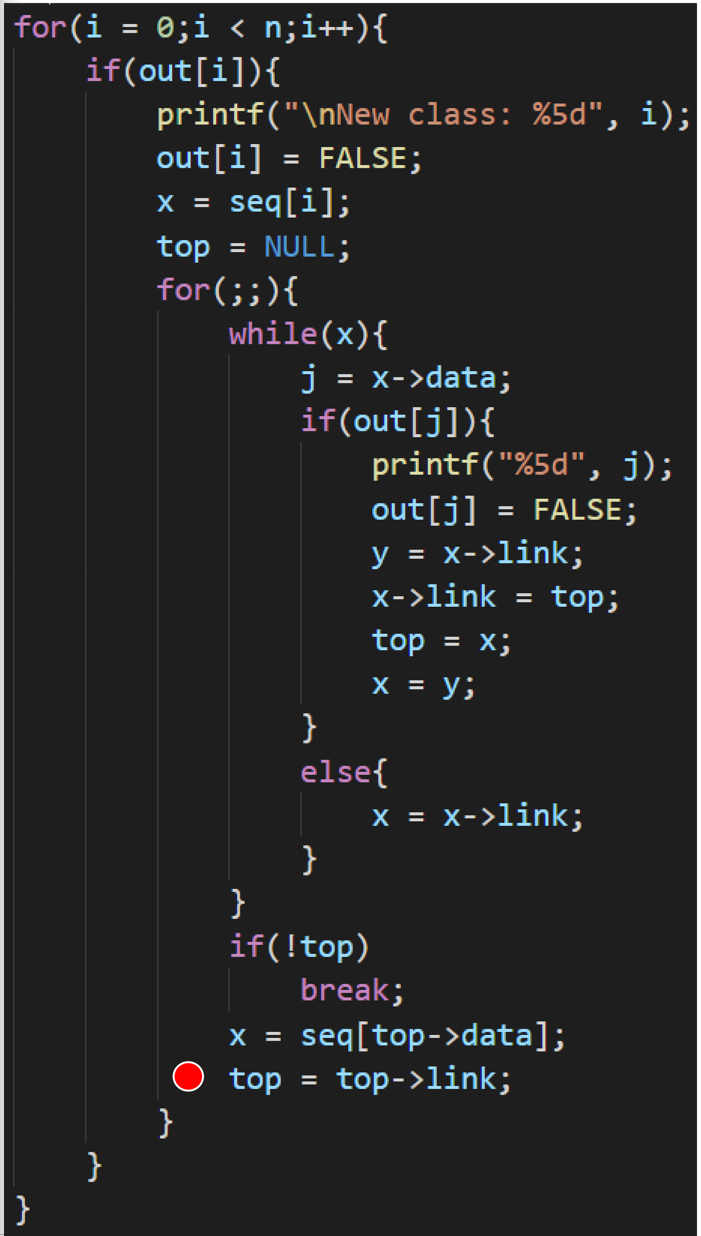
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



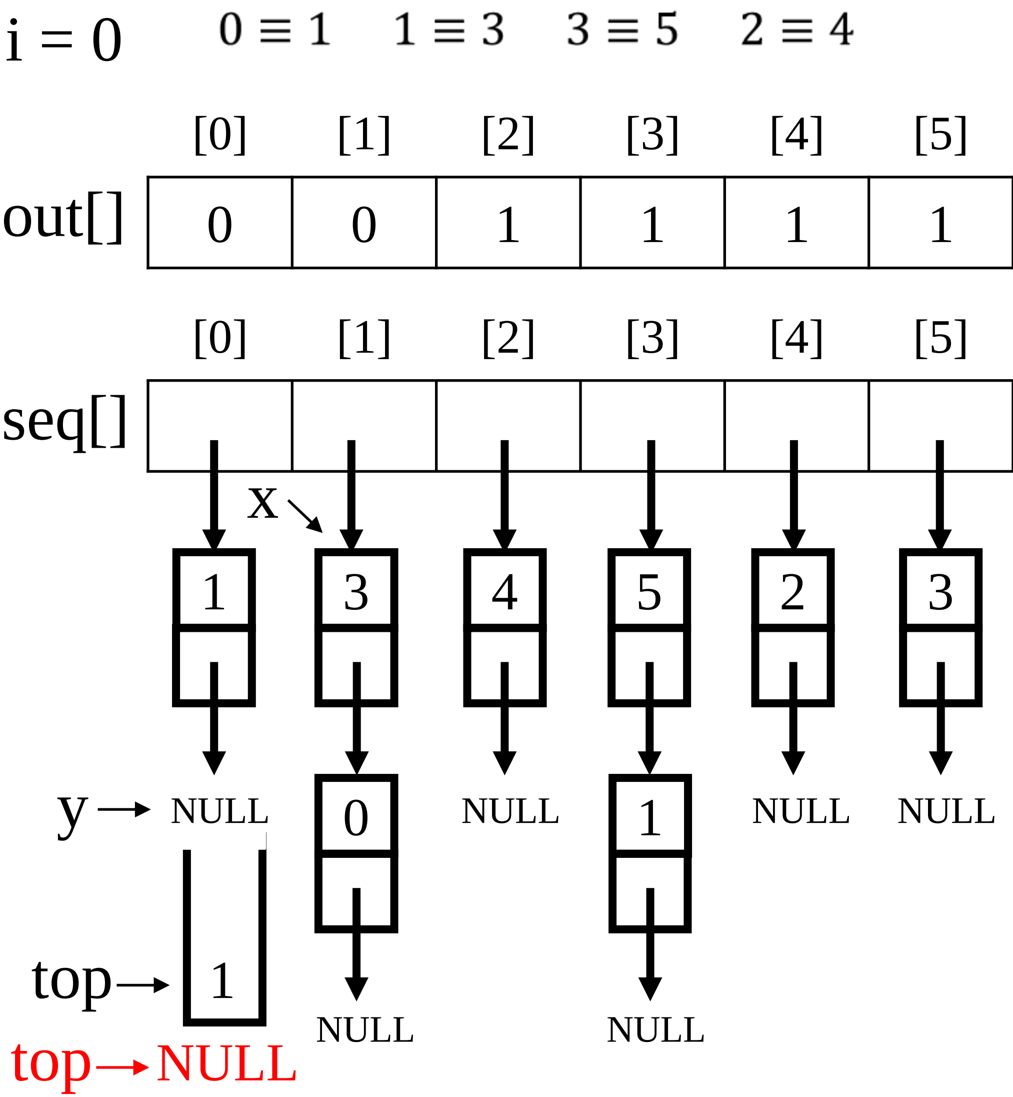
Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1



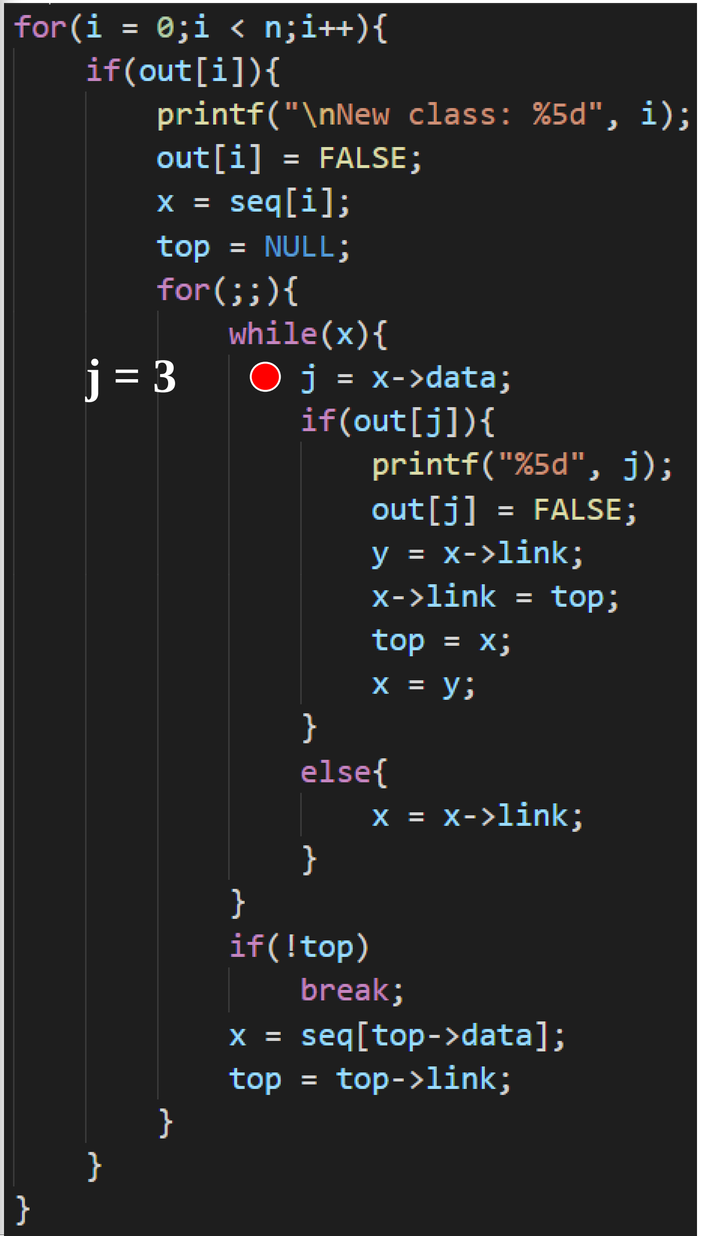
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```



Equivalence relations

Phase 2: output the equivalence classes

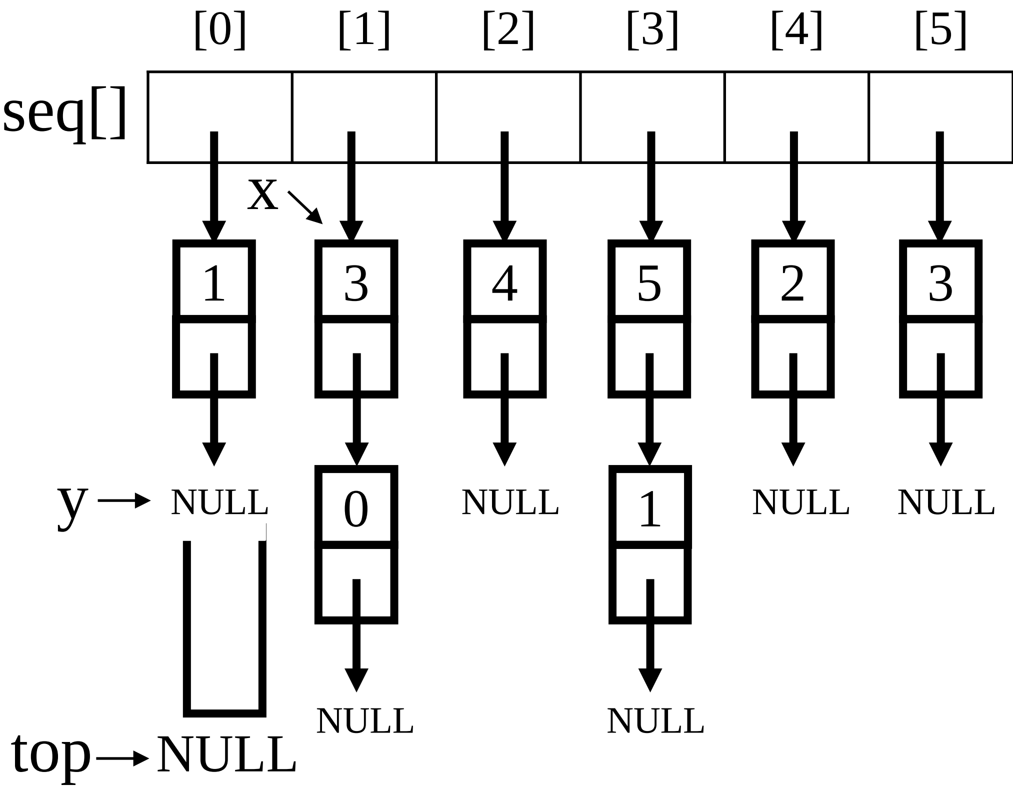
New class: 0 1



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

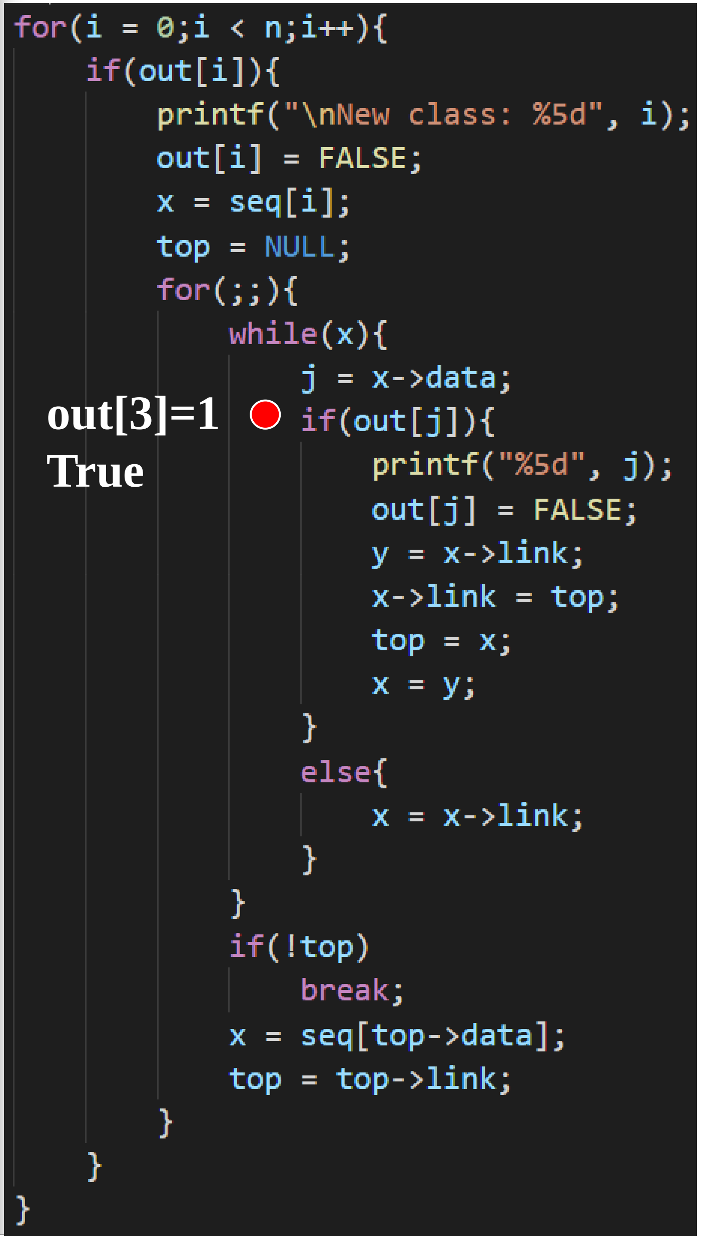
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1



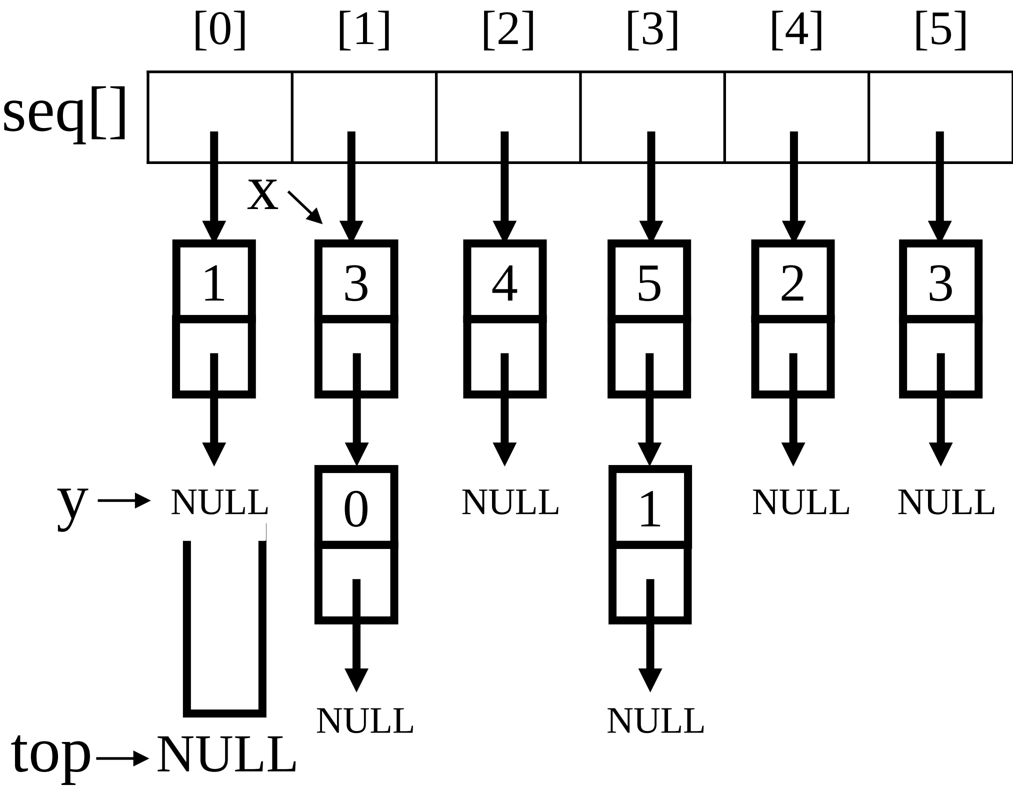
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[3]=1

True

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

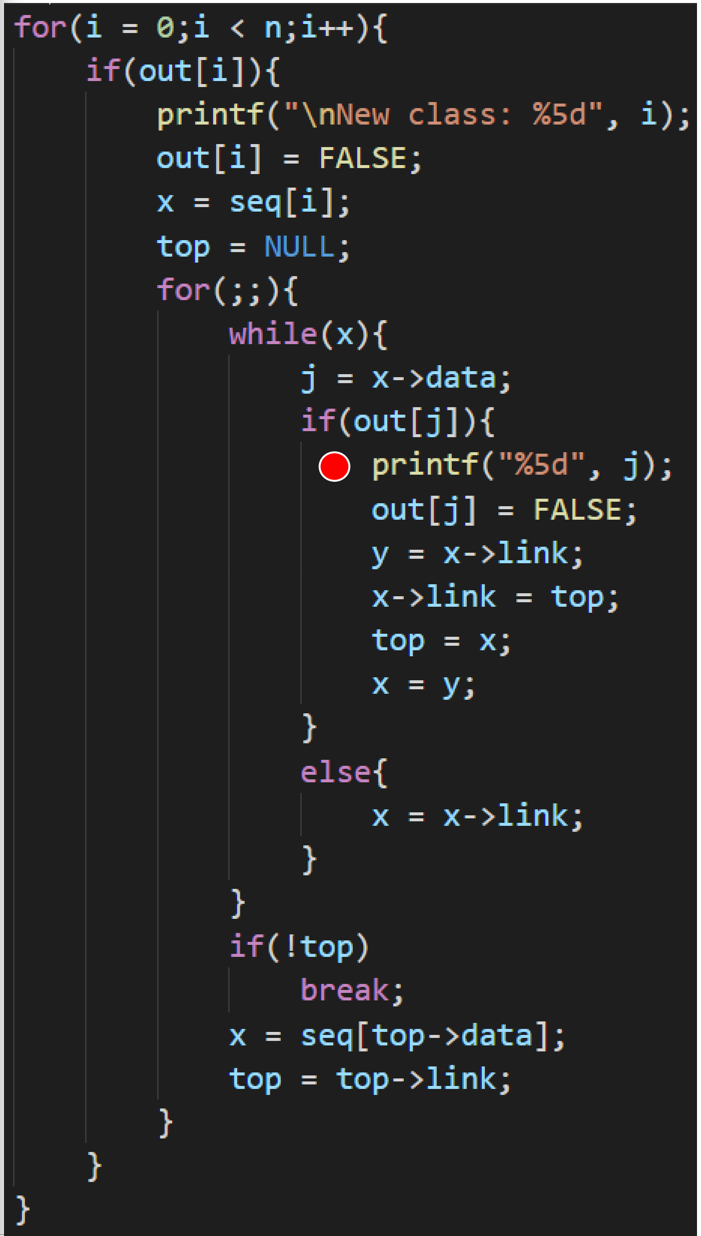
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

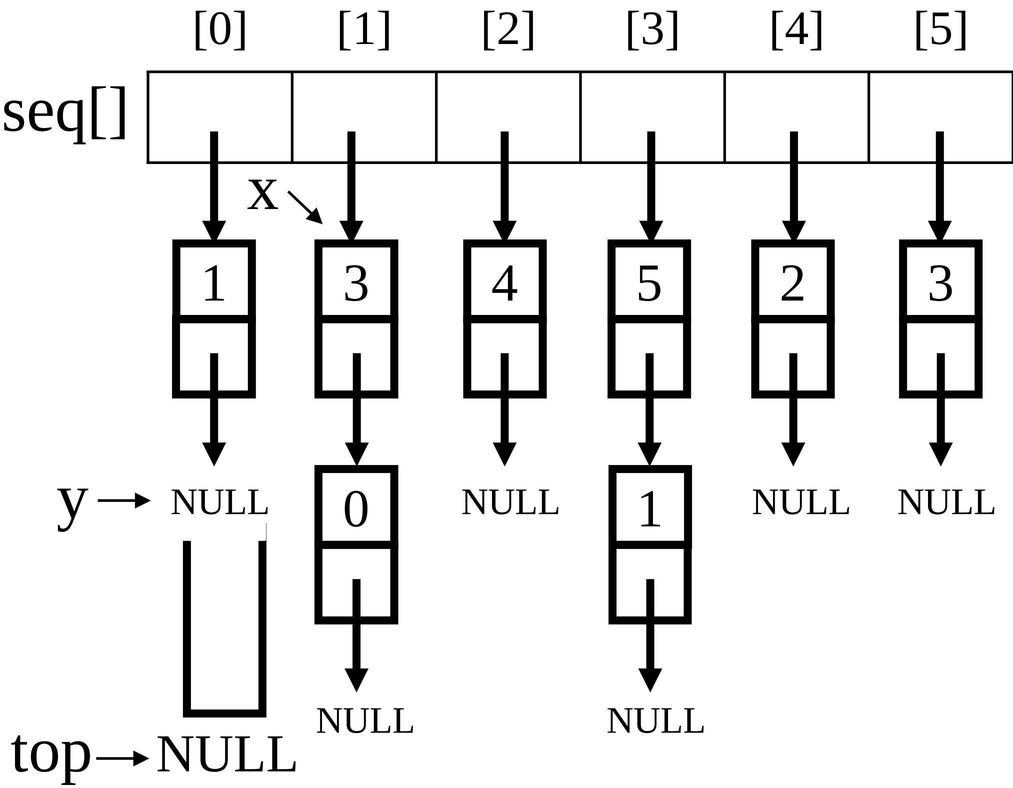
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

$i = 0$ $\quad 0 \equiv 1 \quad 1 \equiv 3 \quad 3 \equiv 5 \quad 2 \equiv 4$

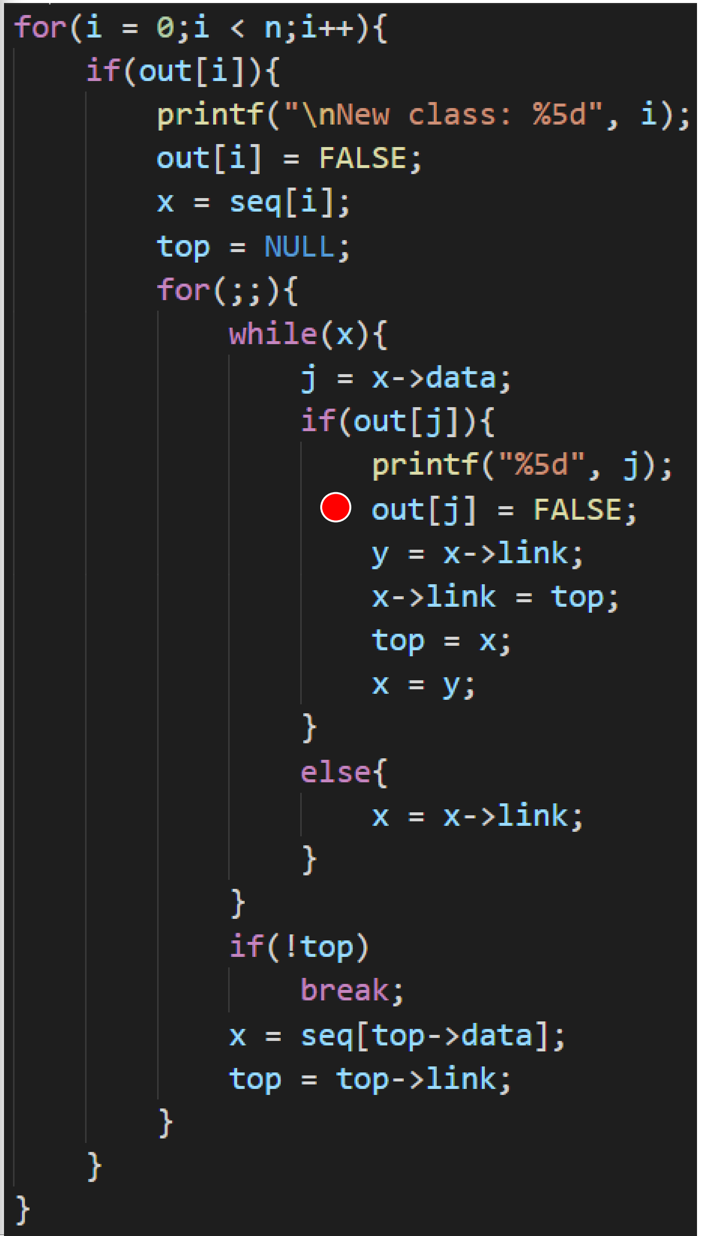
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	1	1	1



Equivalence relations

Phase 2: output the equivalence classes

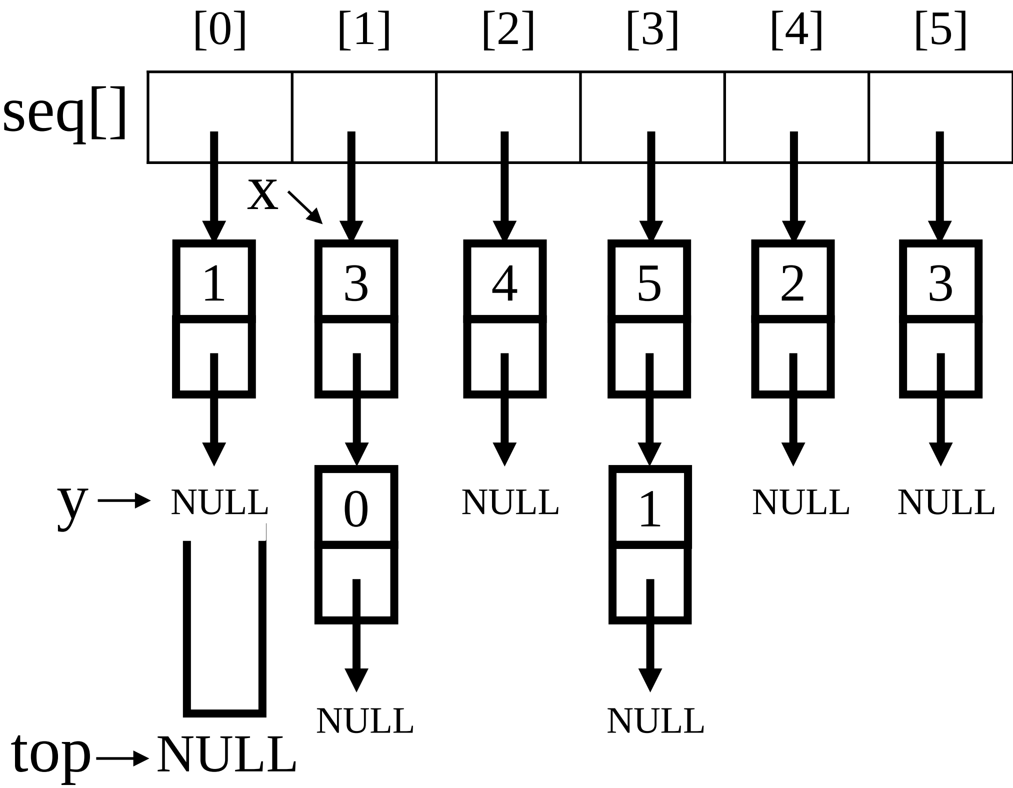
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

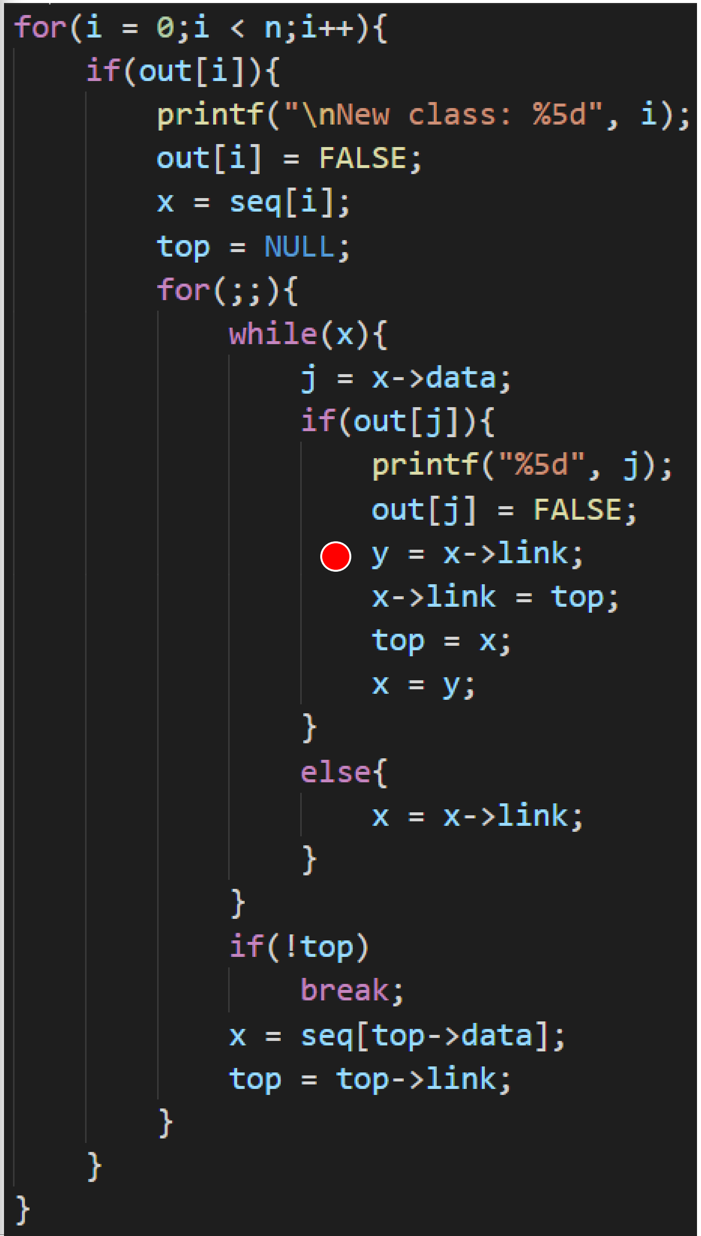
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

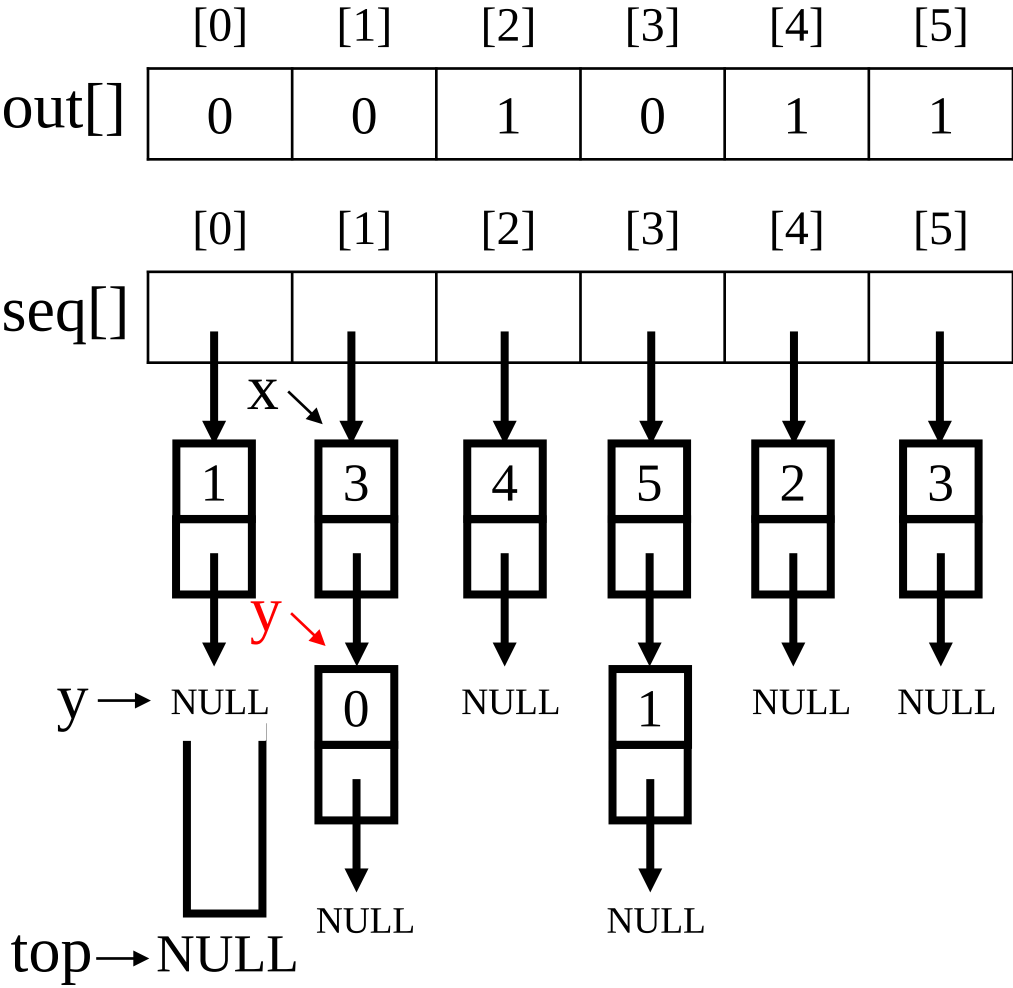
Phase 2: output the equivalence classes

New class: 0   1   3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

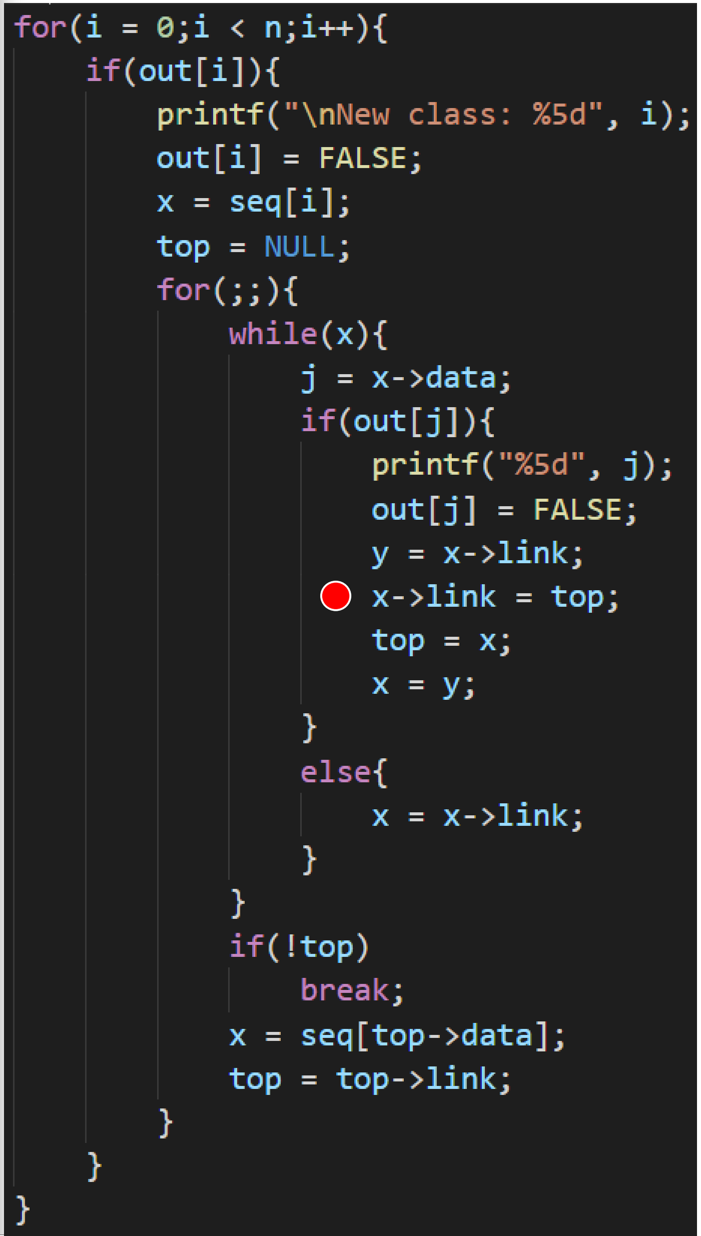


	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1

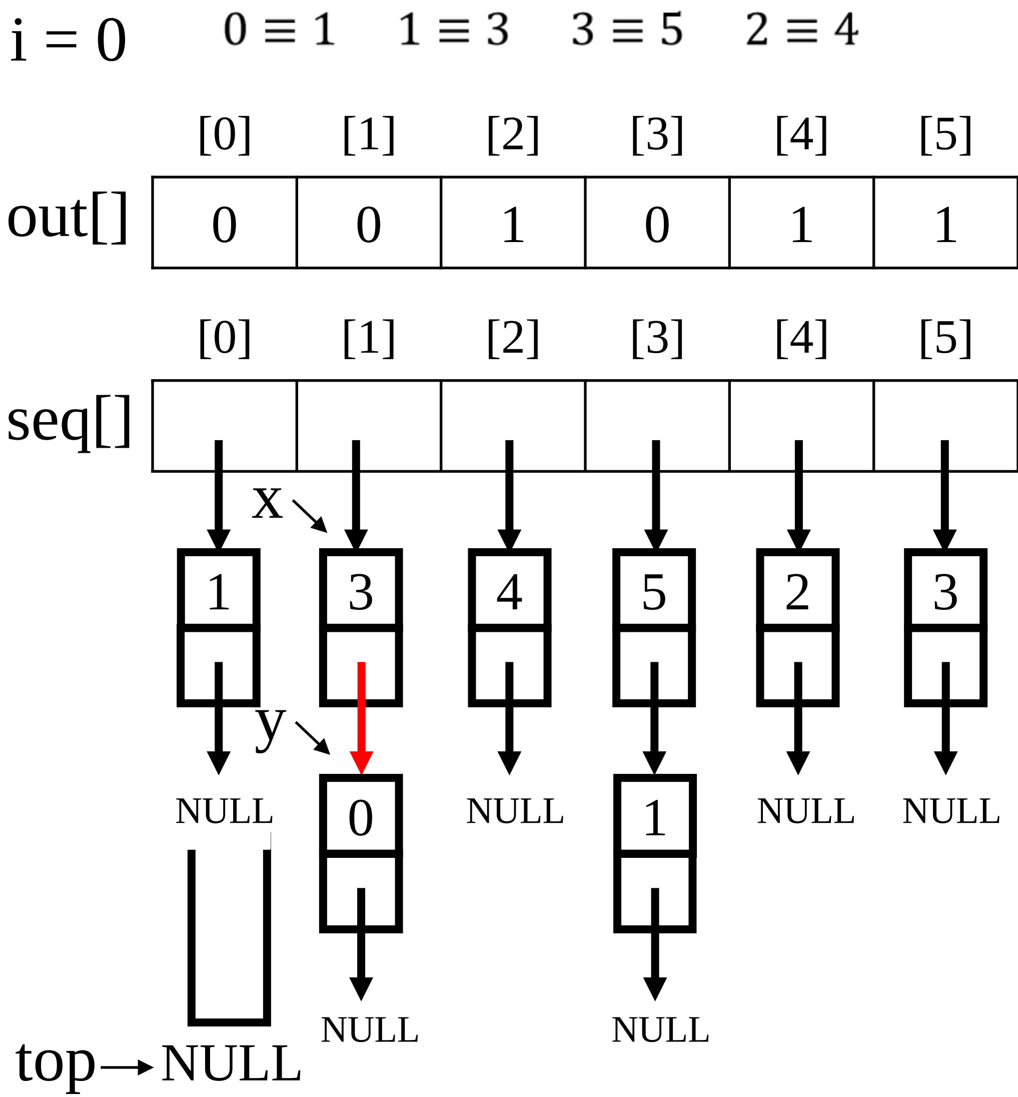
Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3



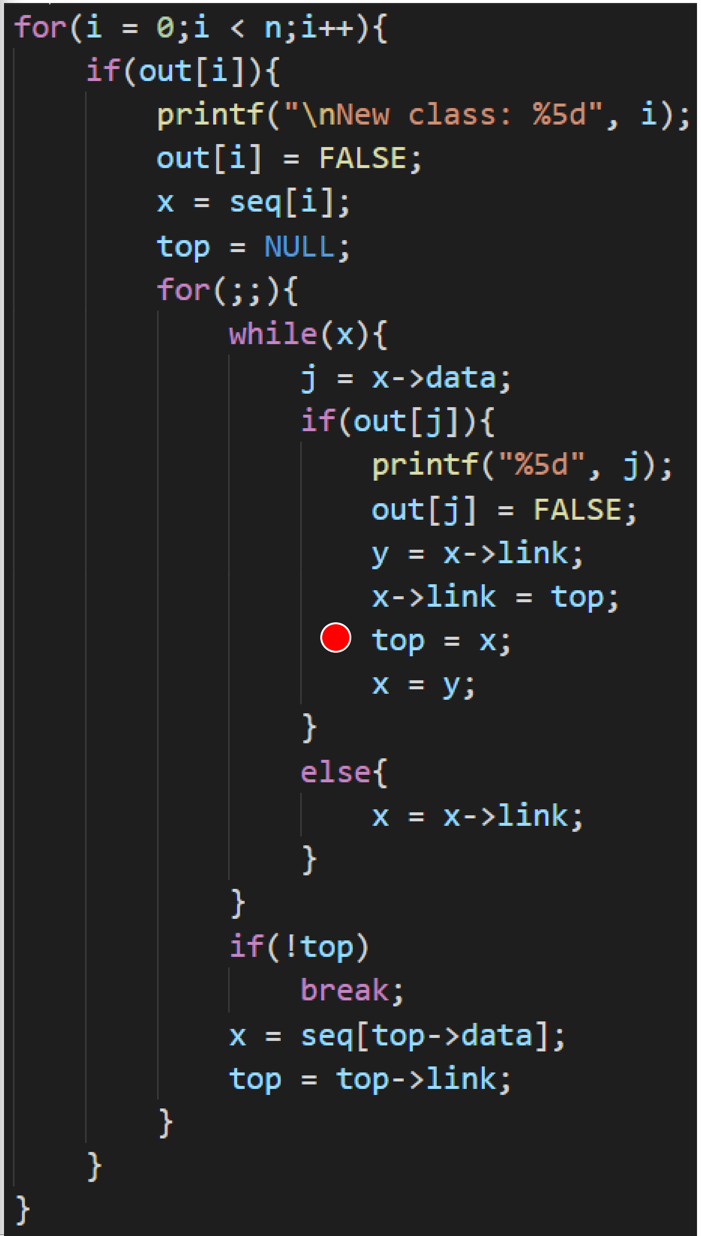
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```



Equivalence relations

Phase 2: output the equivalence classes

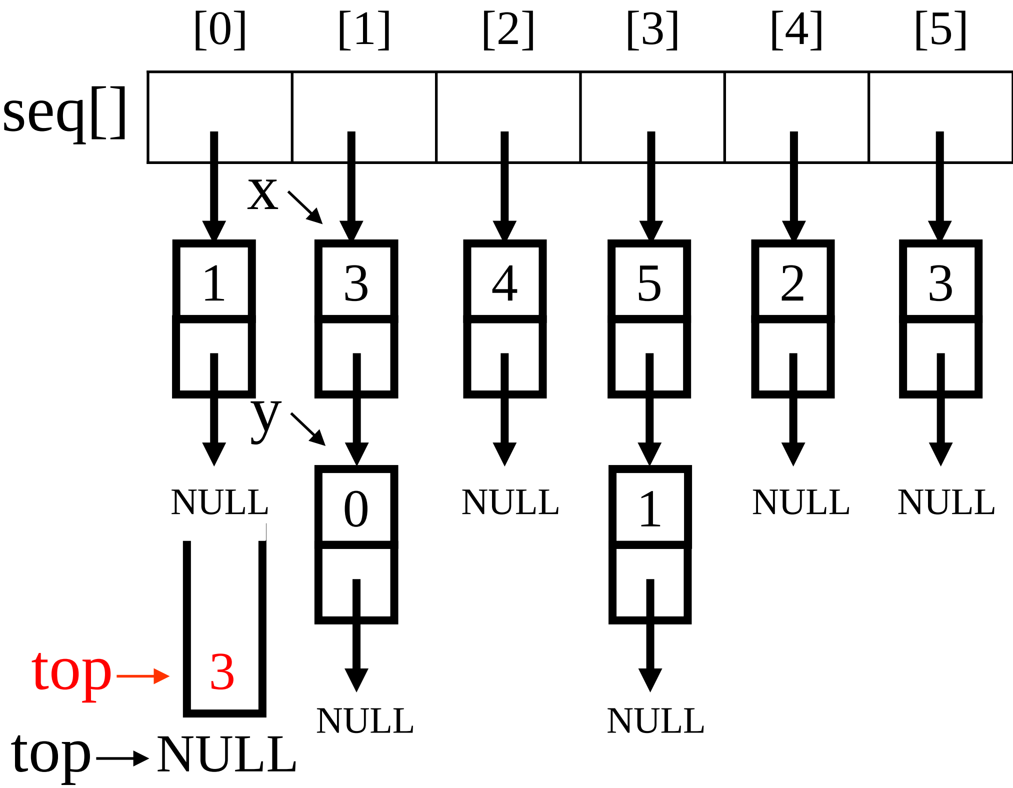
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

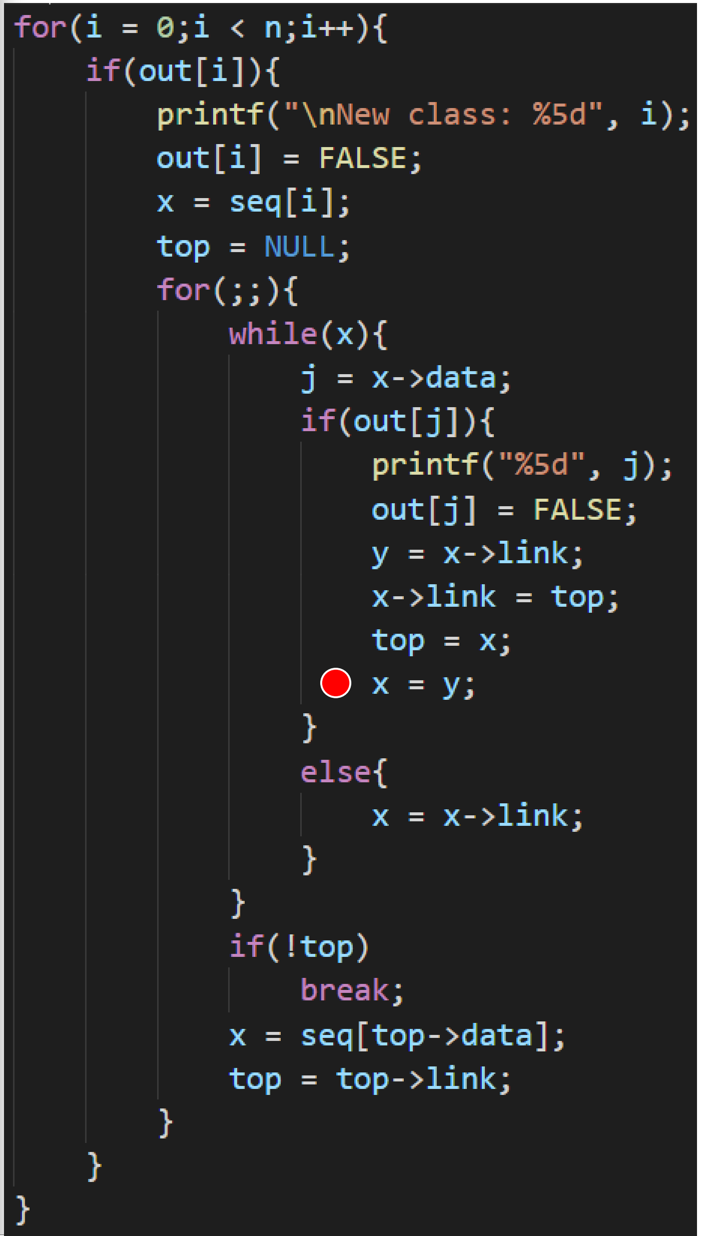
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

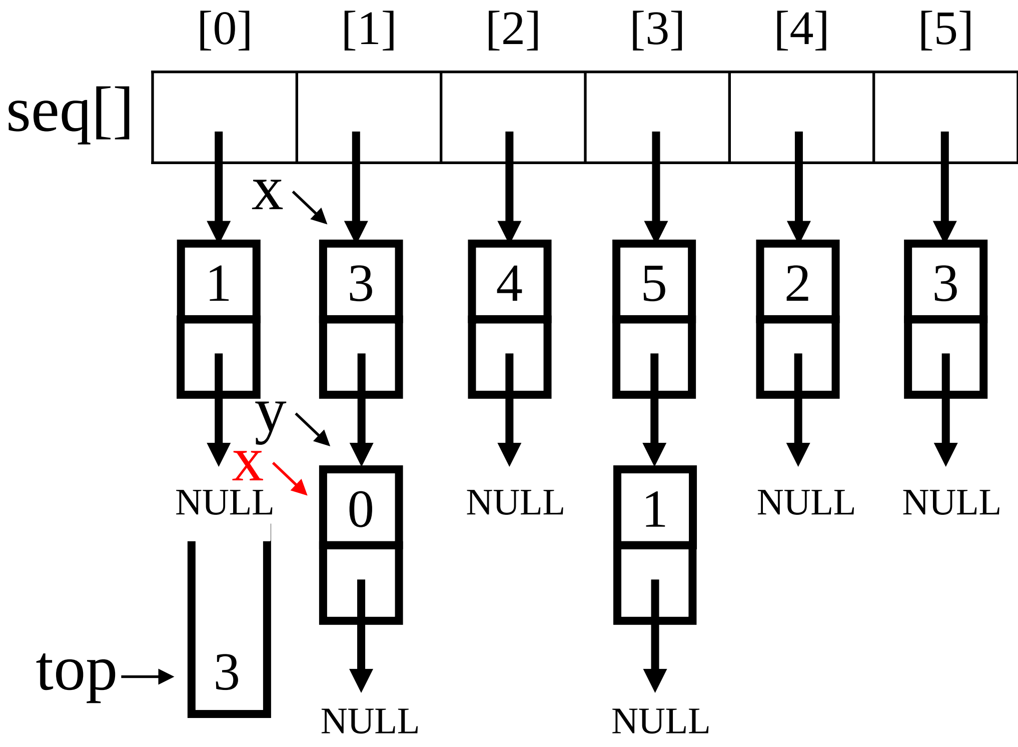
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

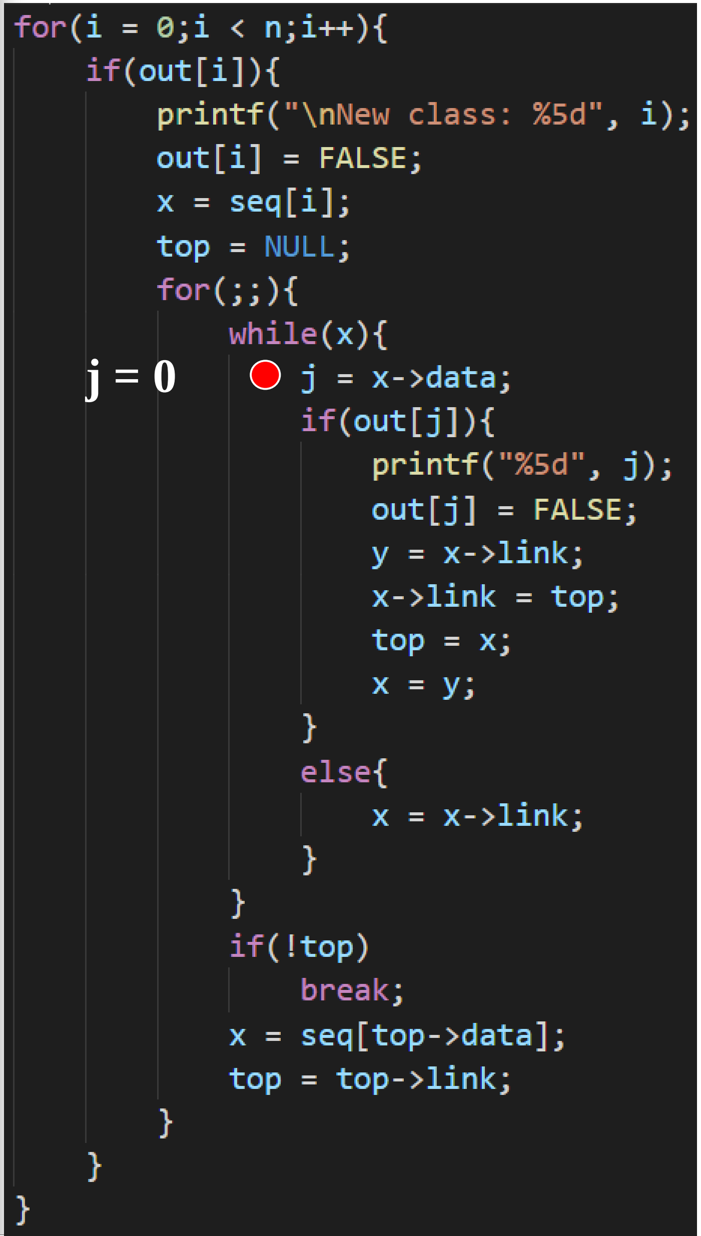
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

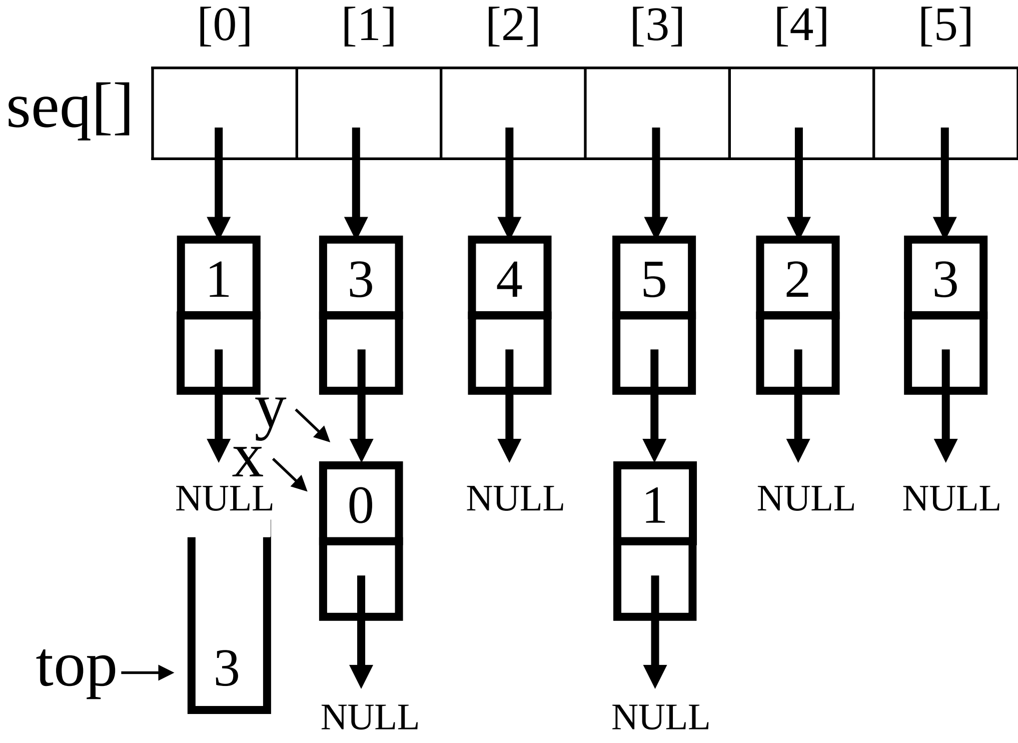
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
    j = 0       ● j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

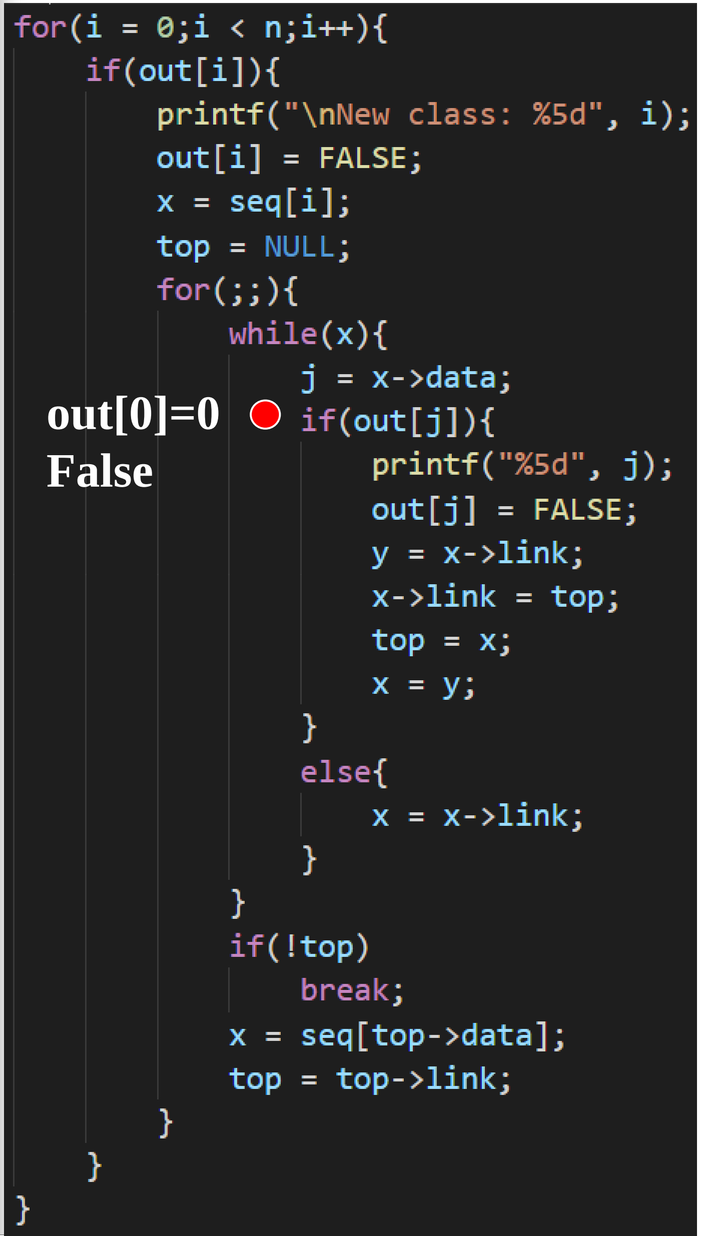
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0   1   3



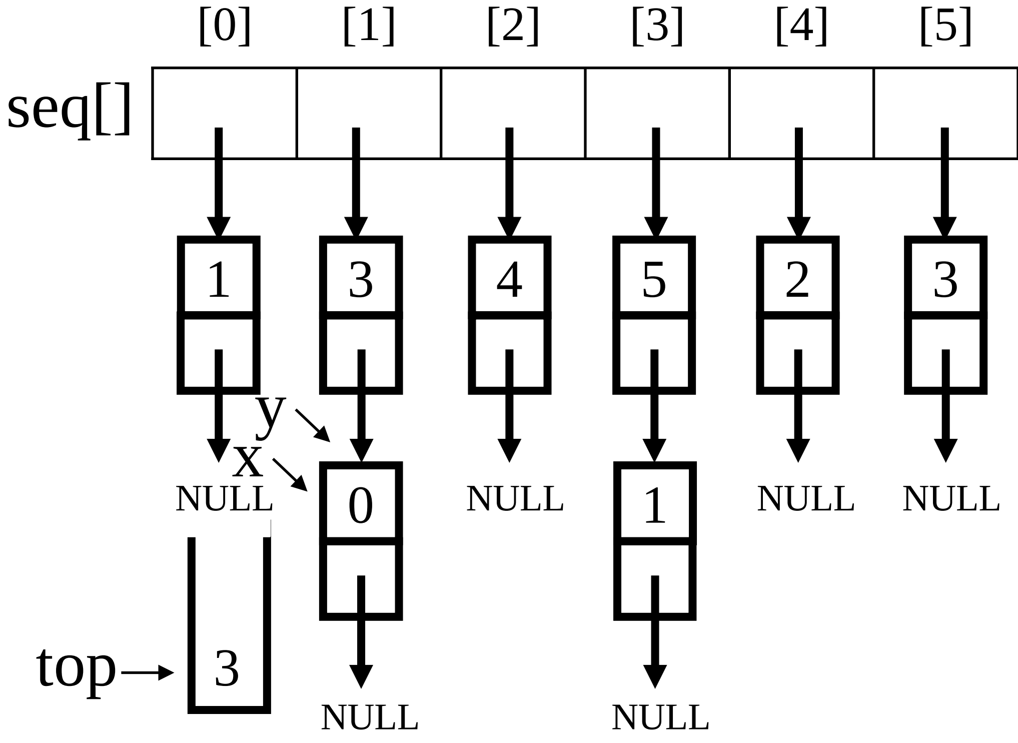
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[0]=0

False

i = 0 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

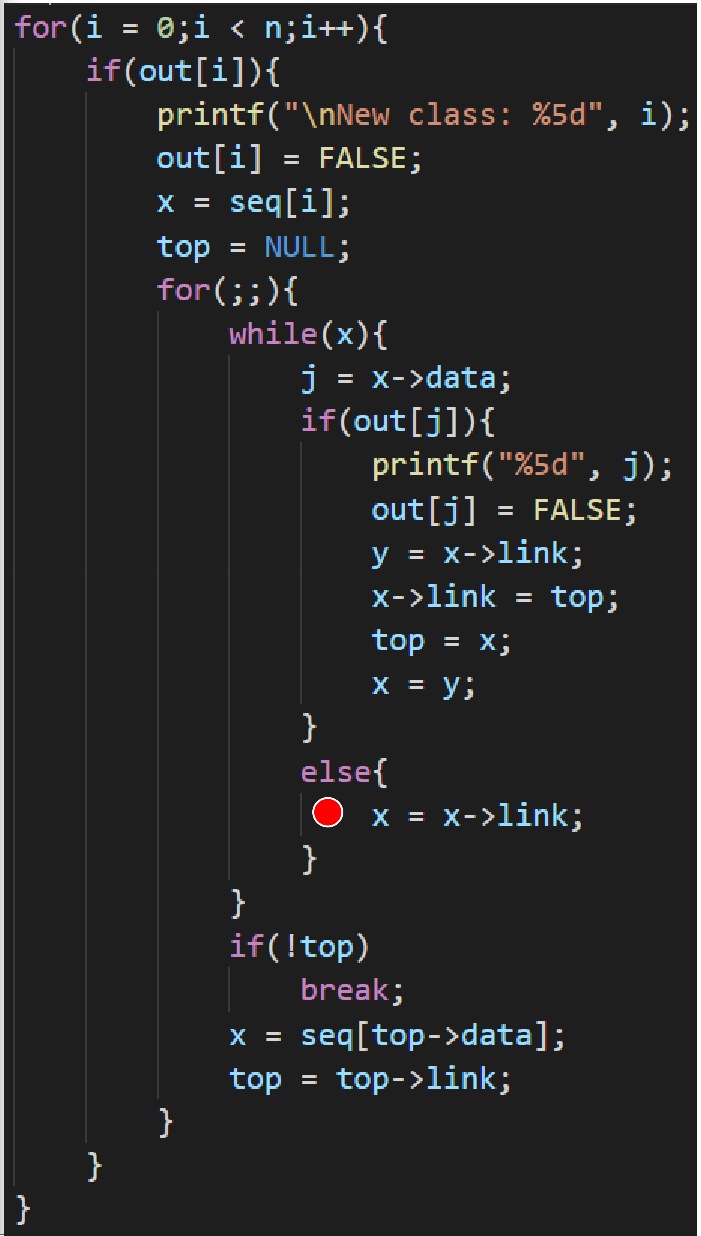
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

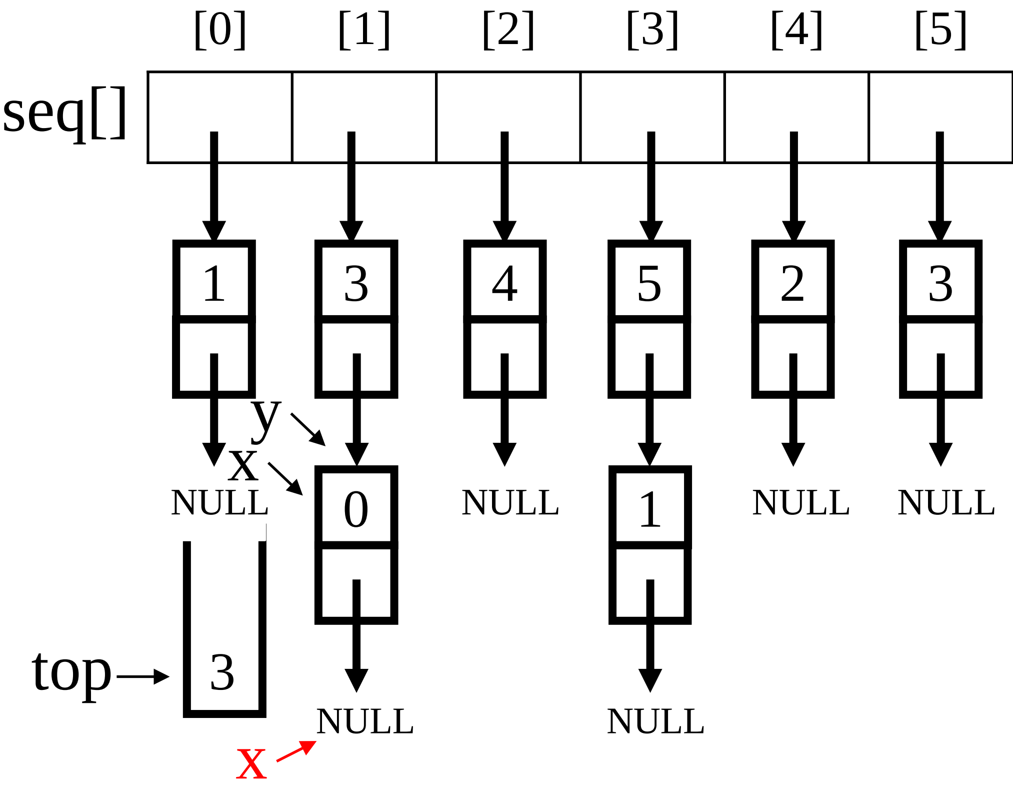
New class: 0   1   3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

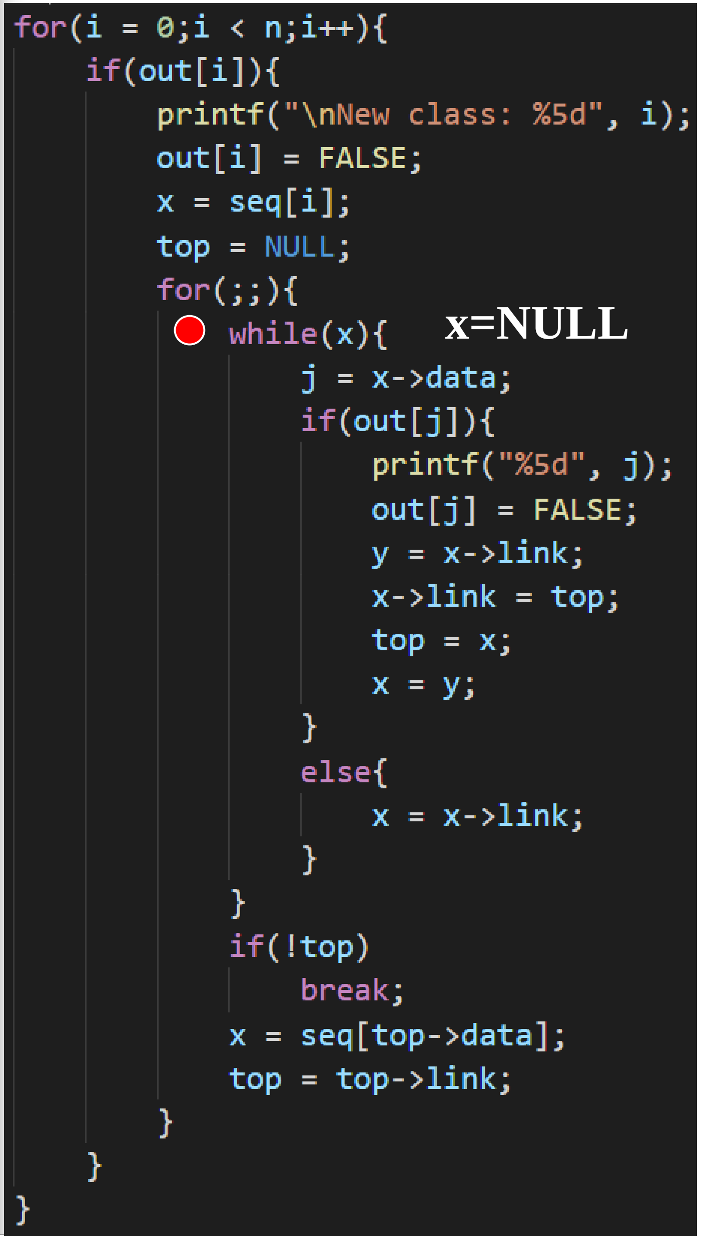
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3

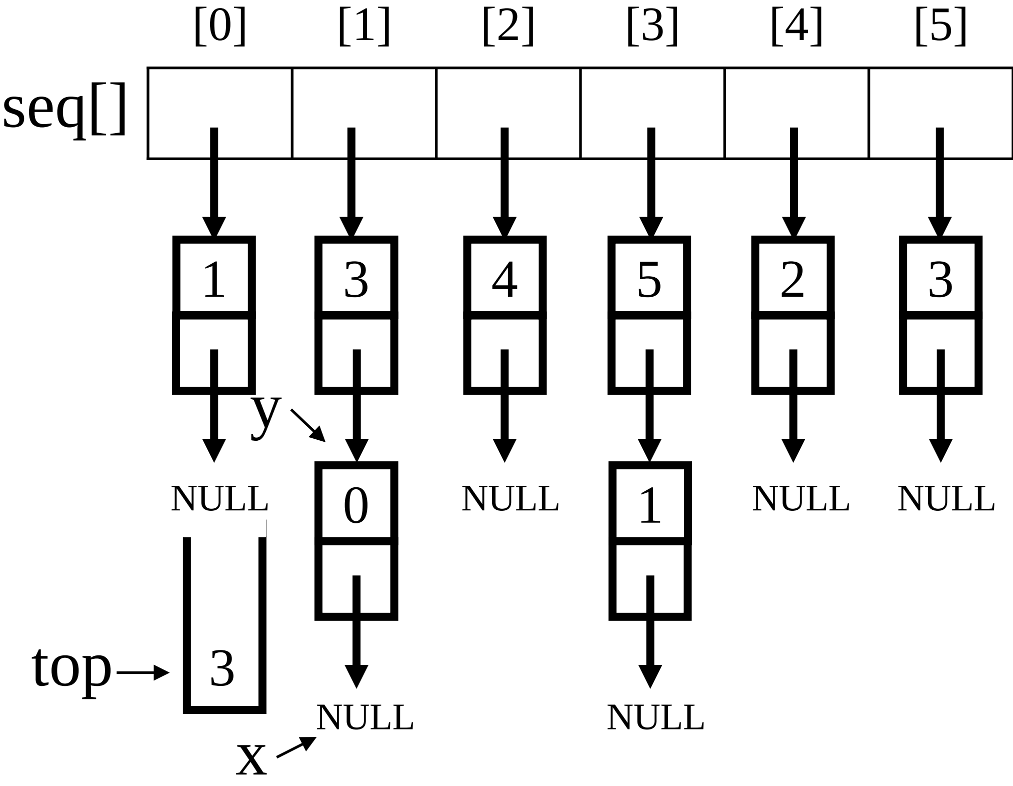


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

x=NULL

i = 0 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

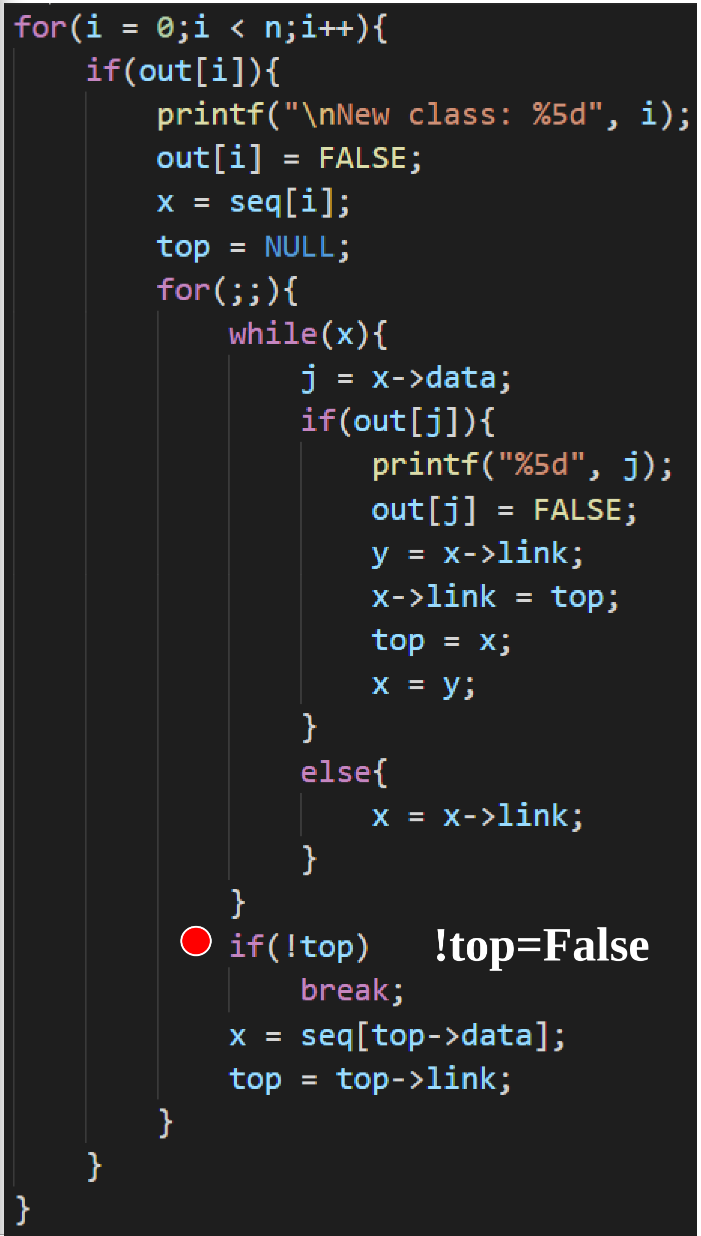
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

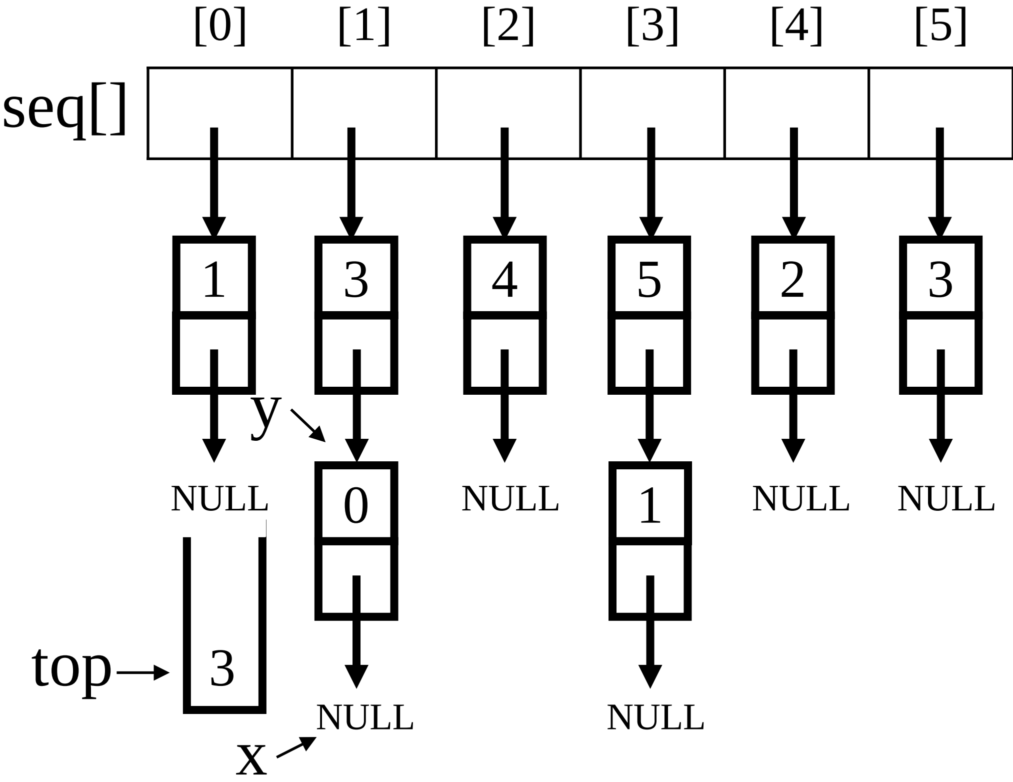
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
●           if(!top)      !top=False
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

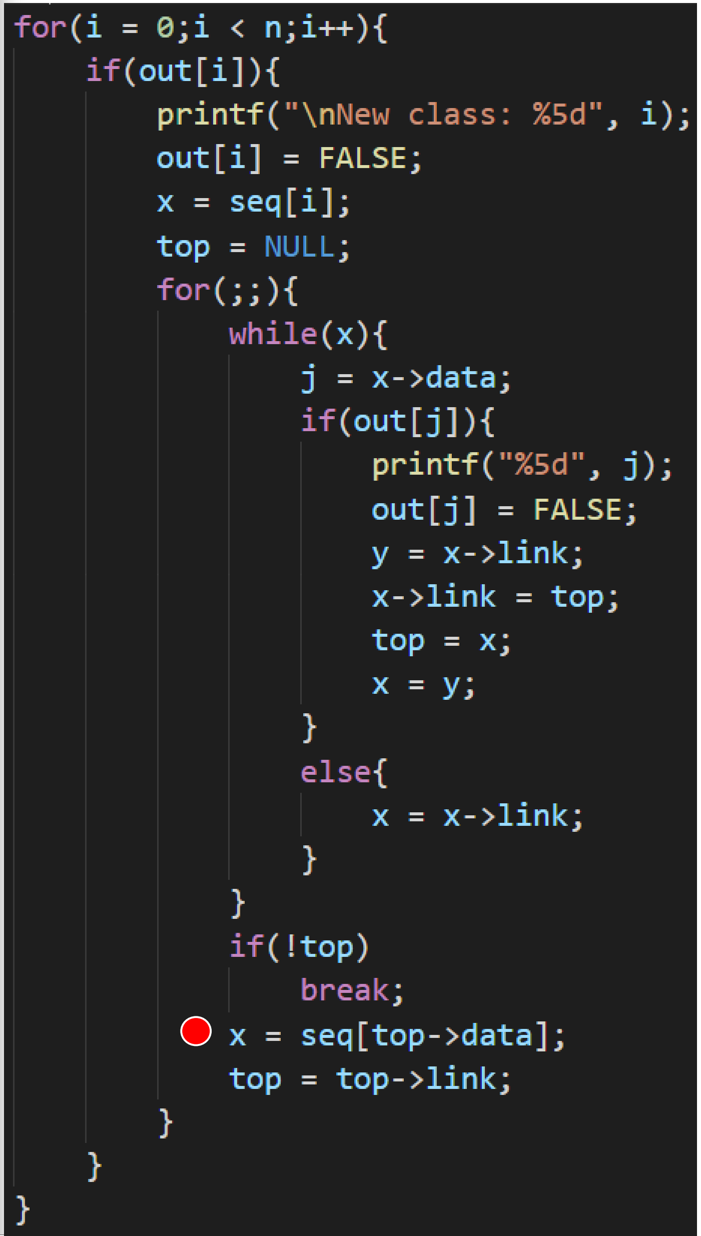
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

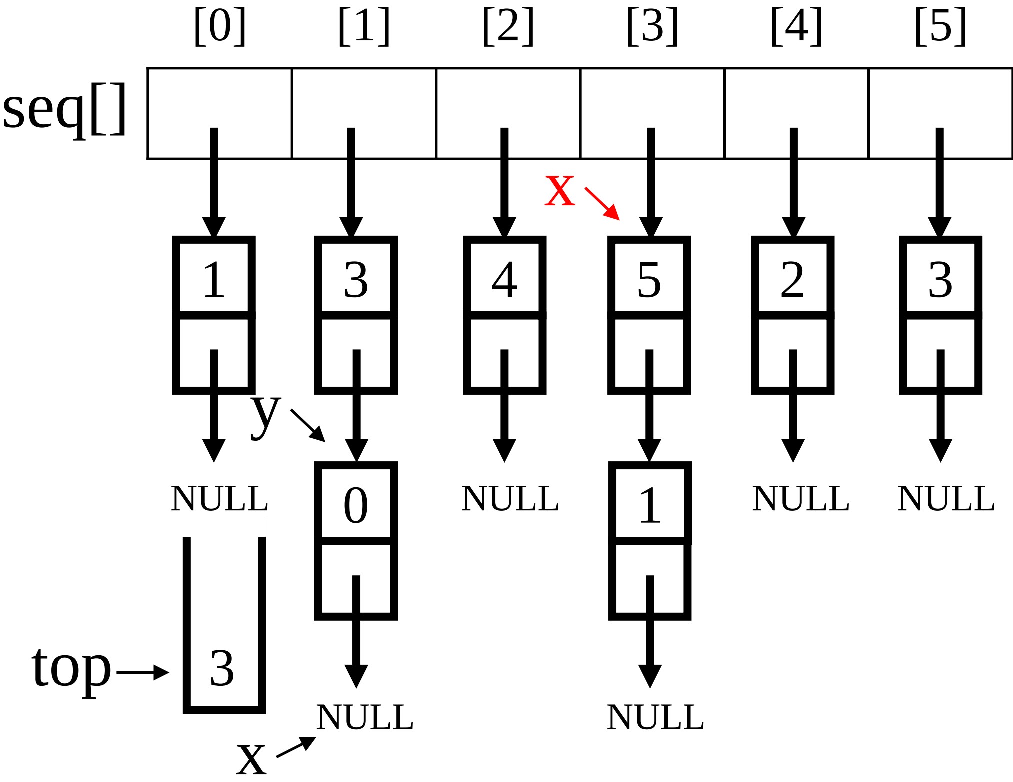
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

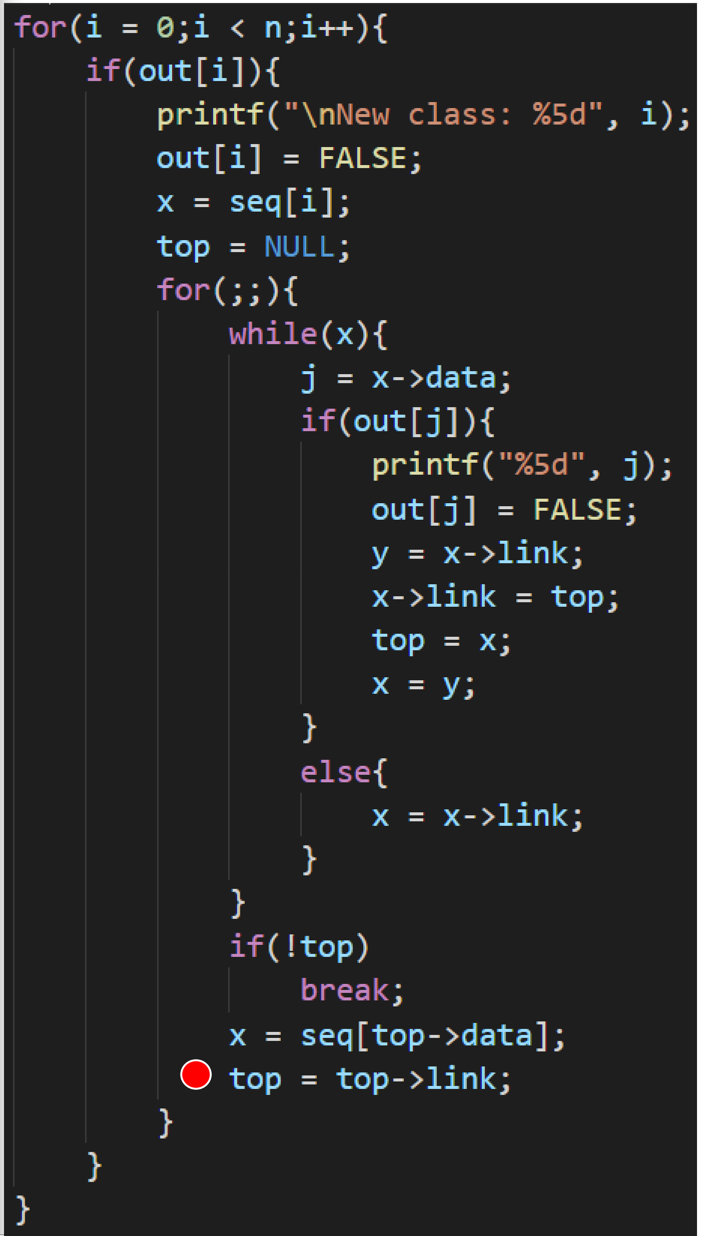
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

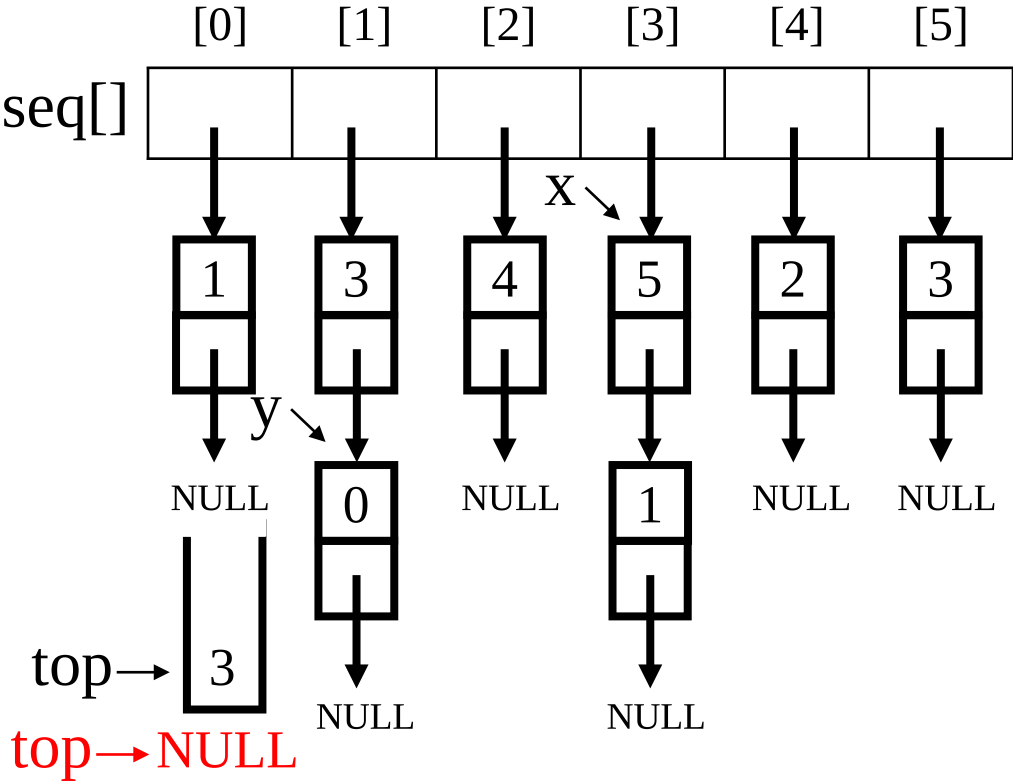
New class: 0 1 3



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
        ●   top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

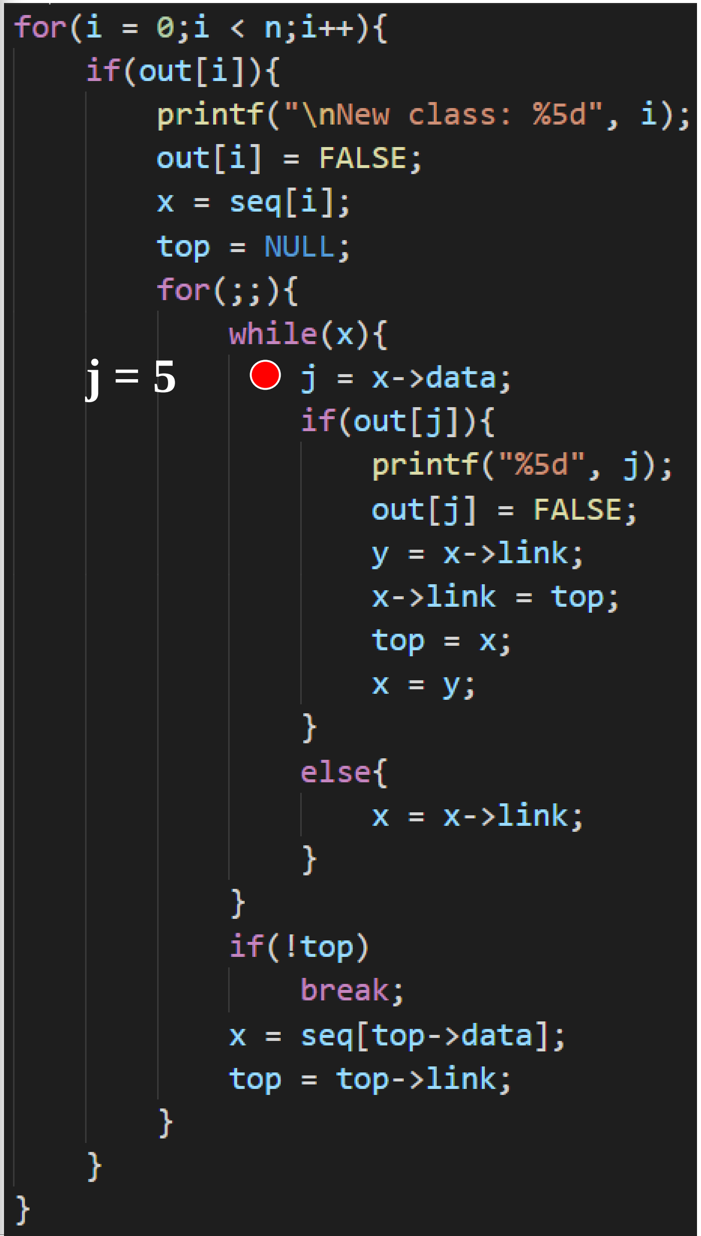
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3

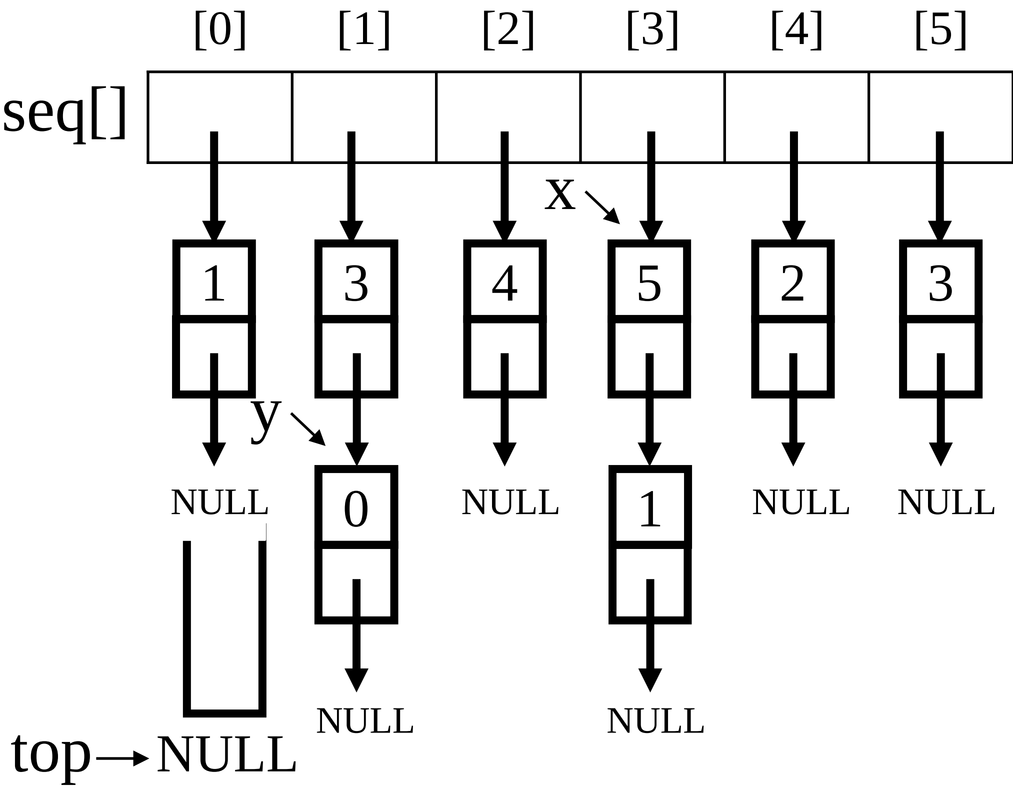


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

j = 5

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

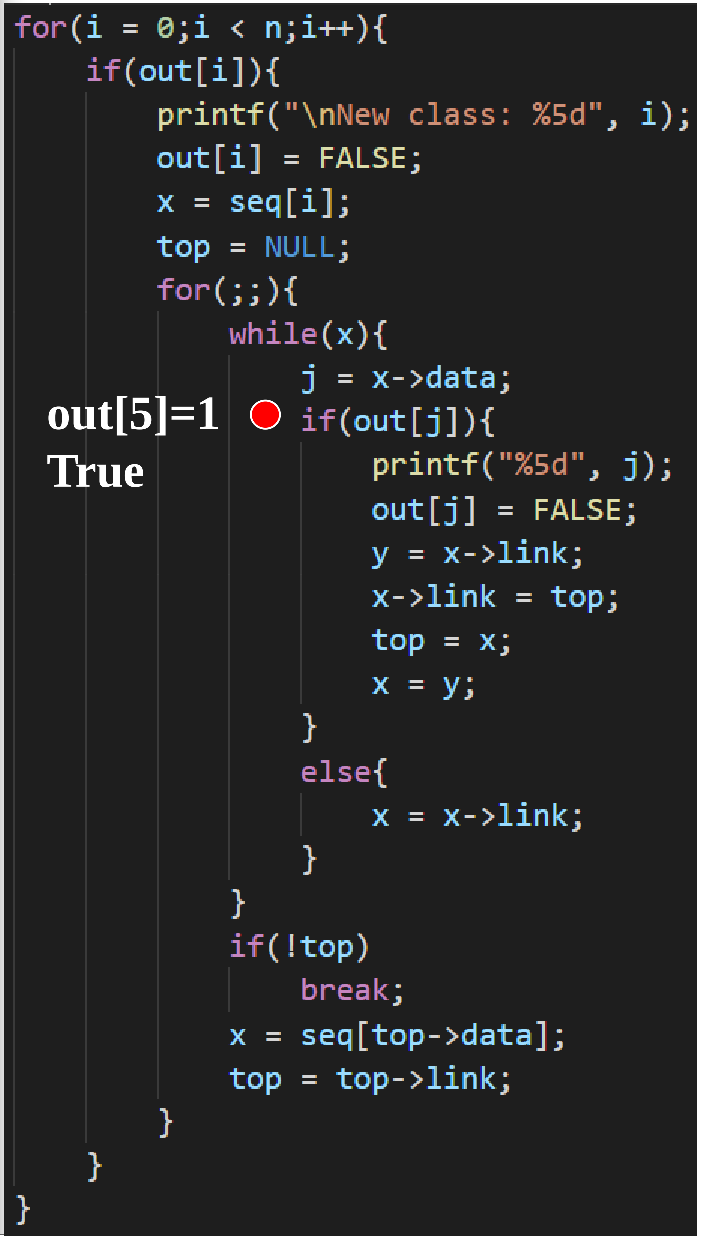
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3



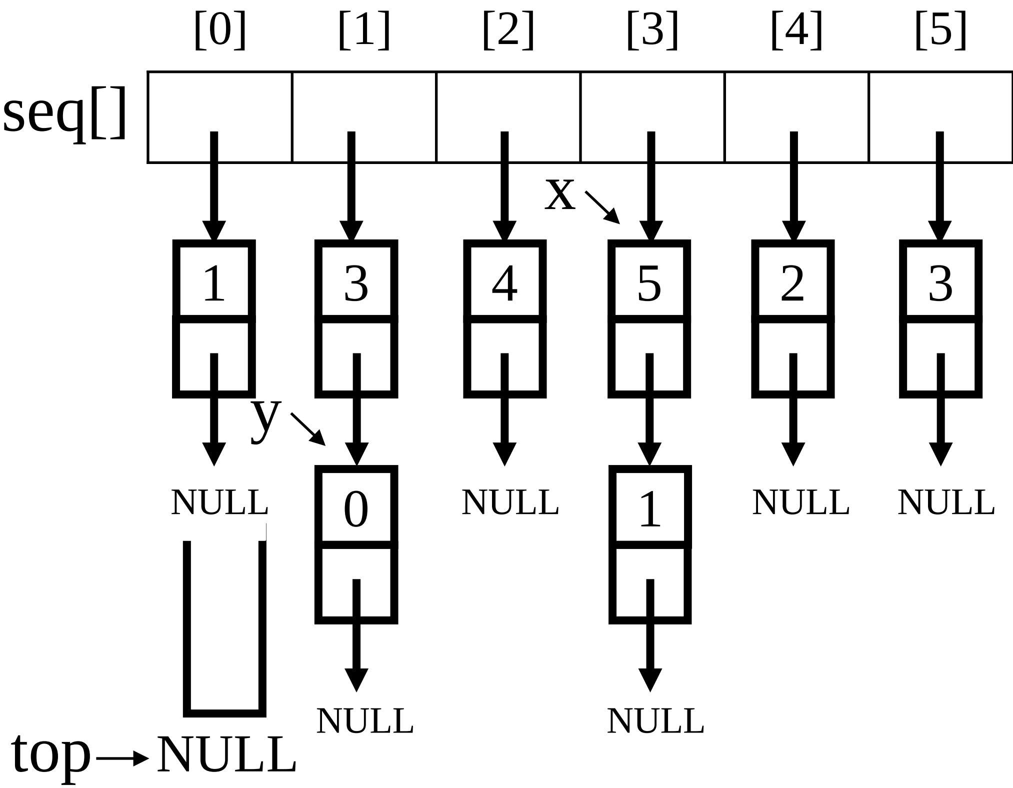
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[5]=1 ●

True

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

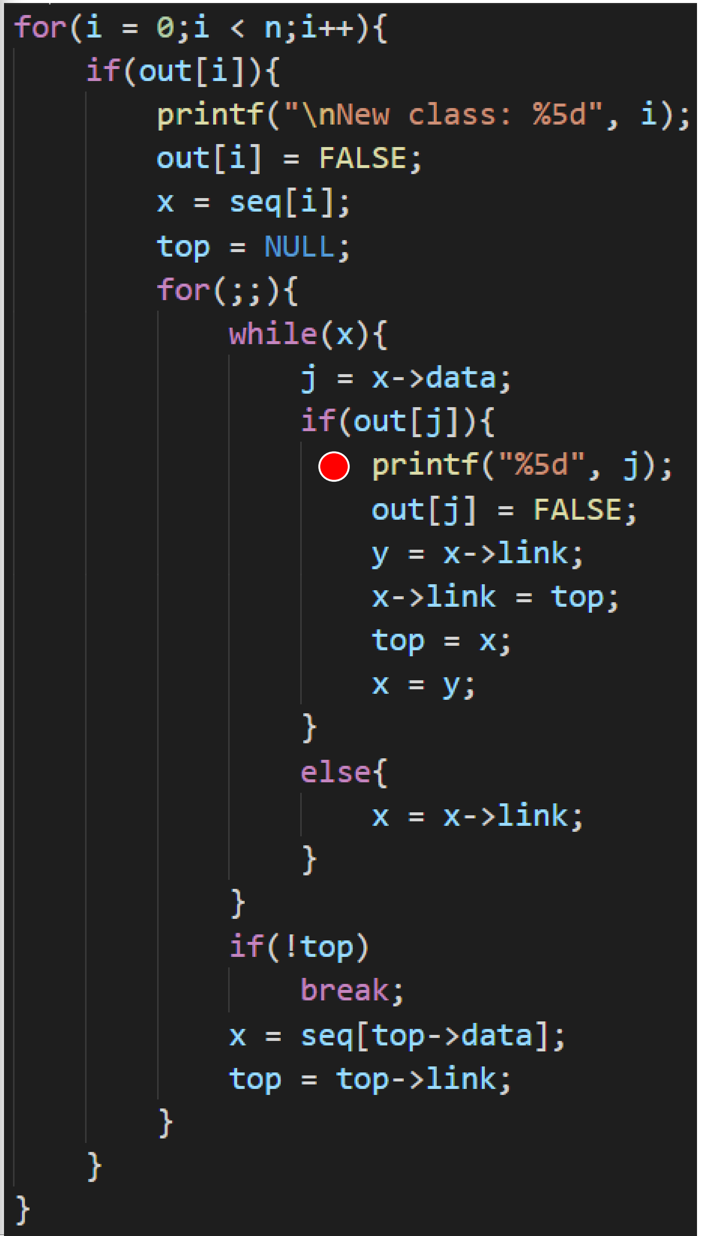
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

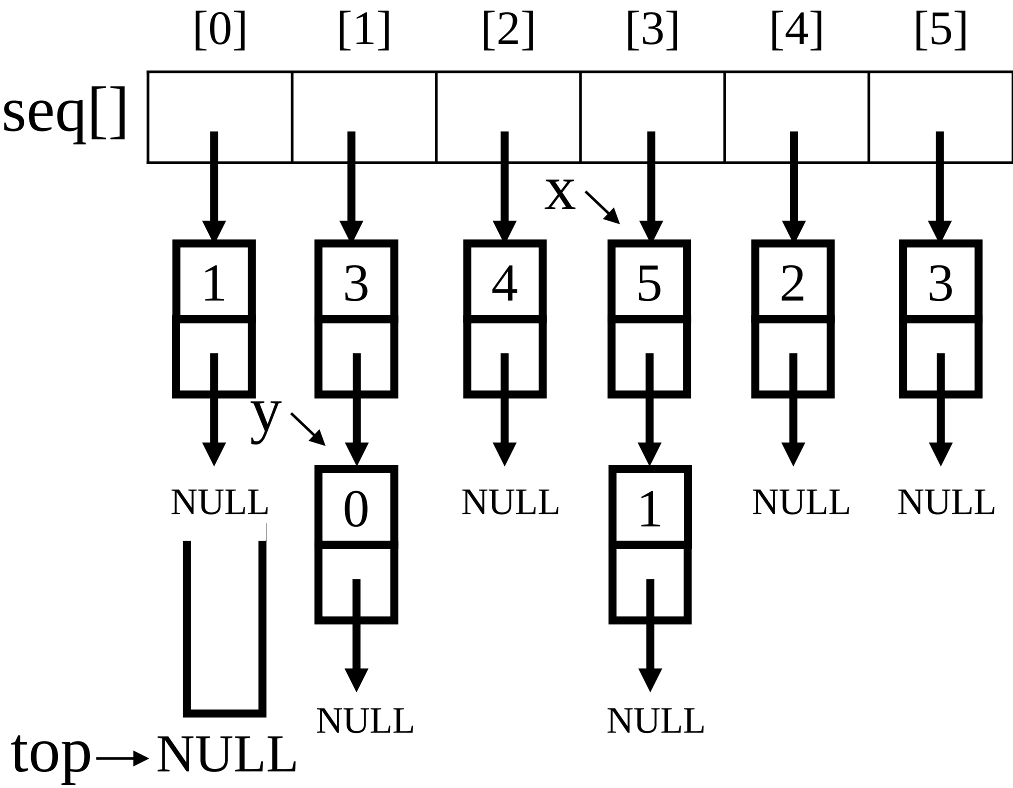
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

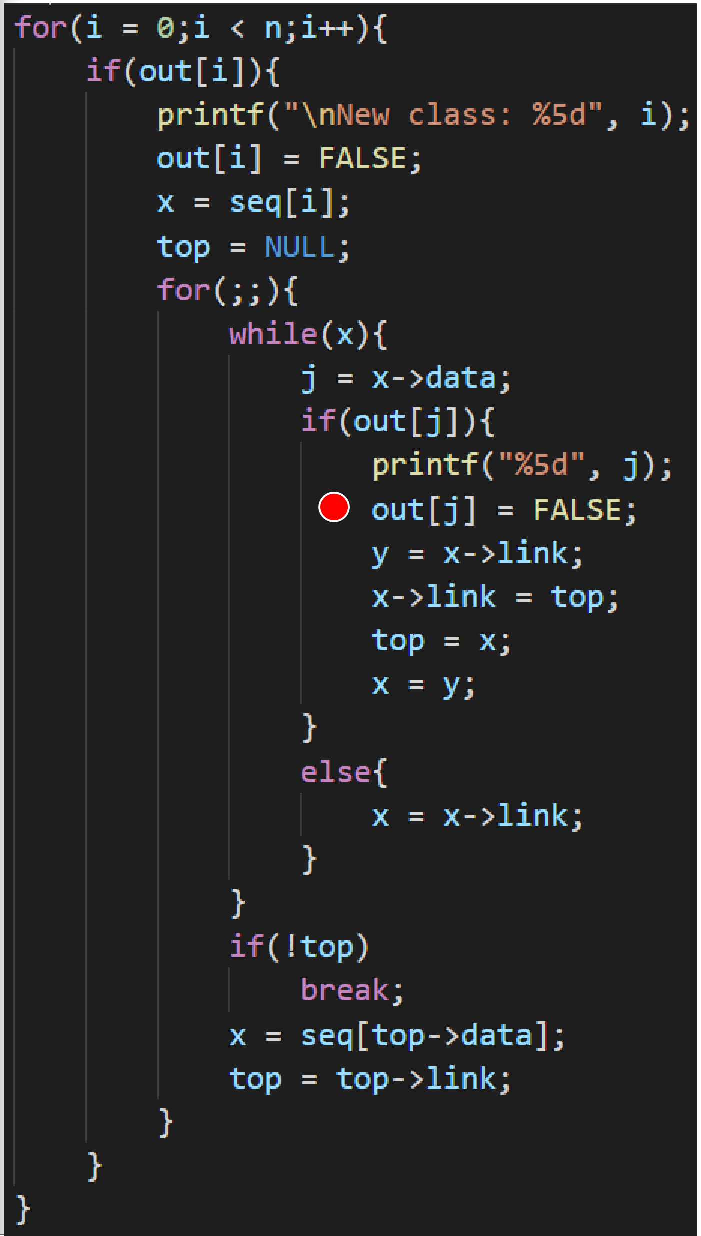
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	1



Equivalence relations

Phase 2: output the equivalence classes

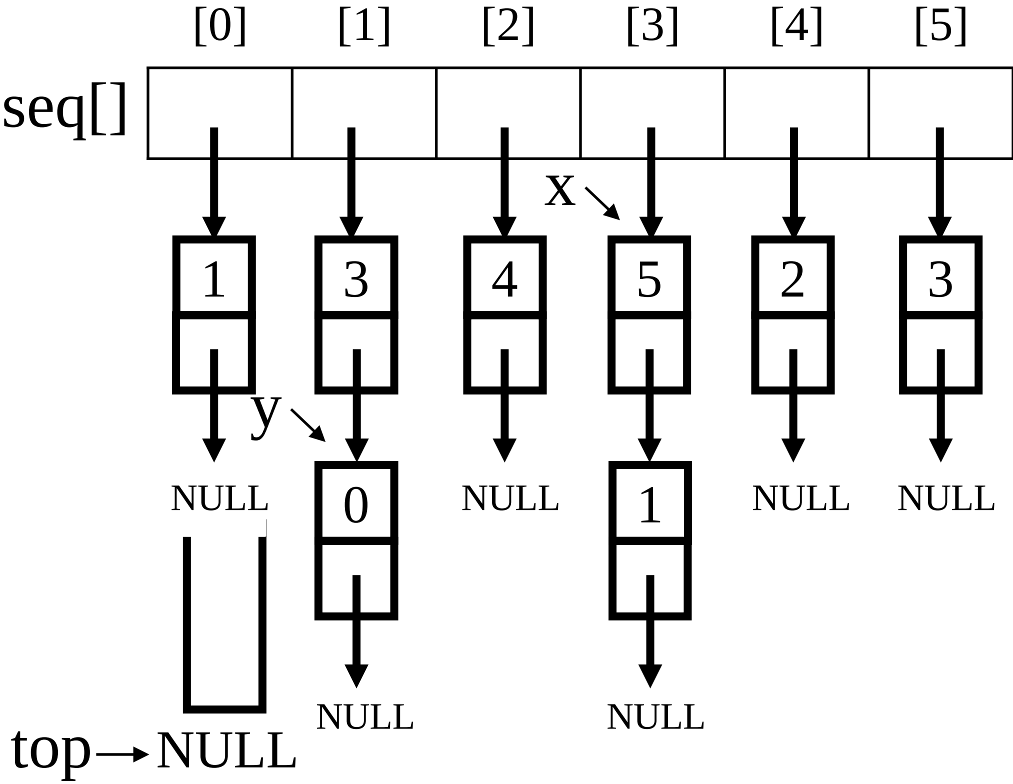
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

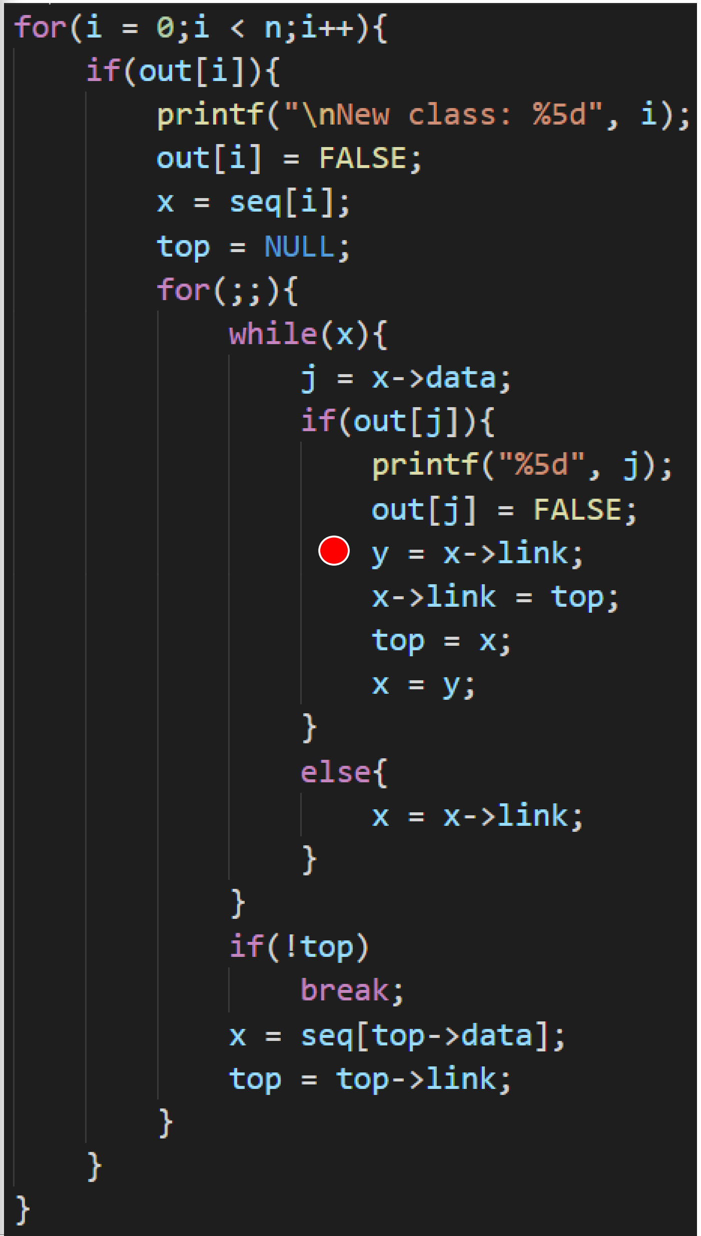
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

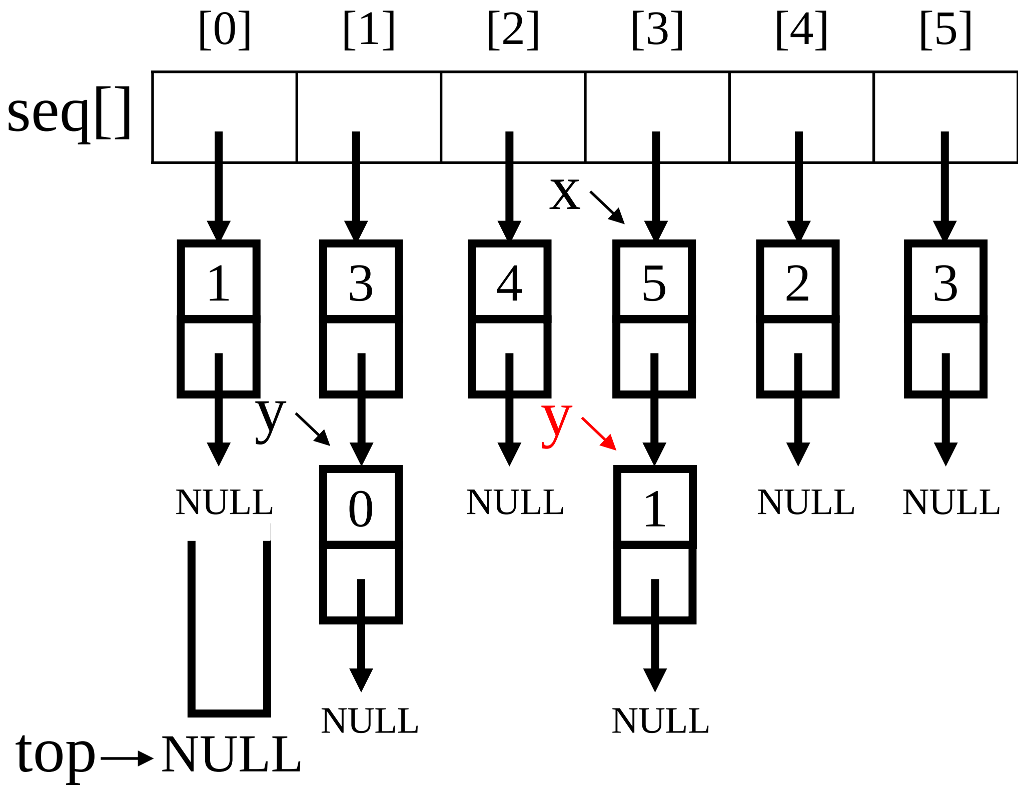
New class: 0   1   3   5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

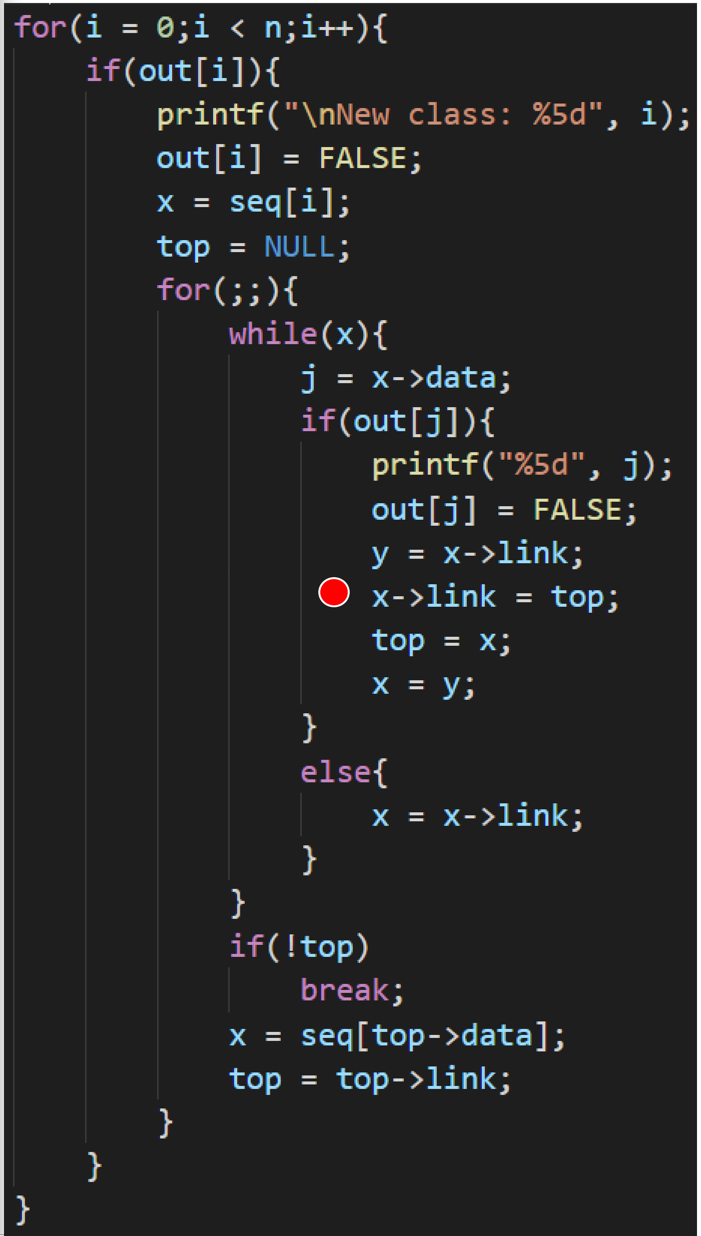
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

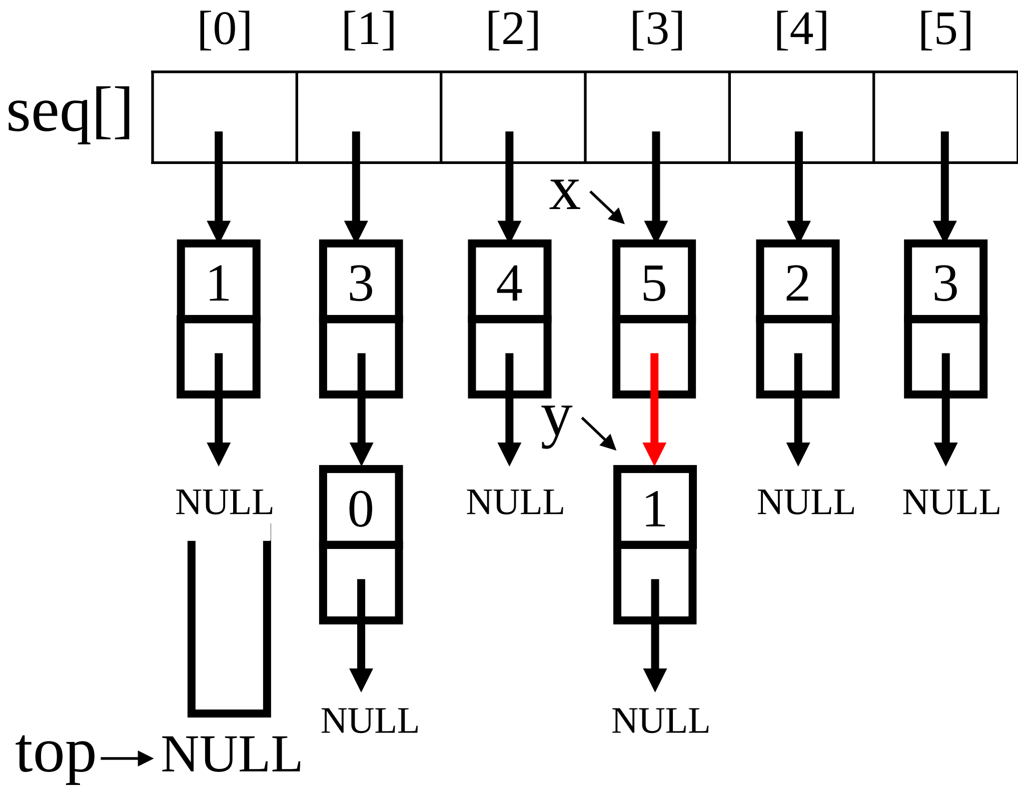
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

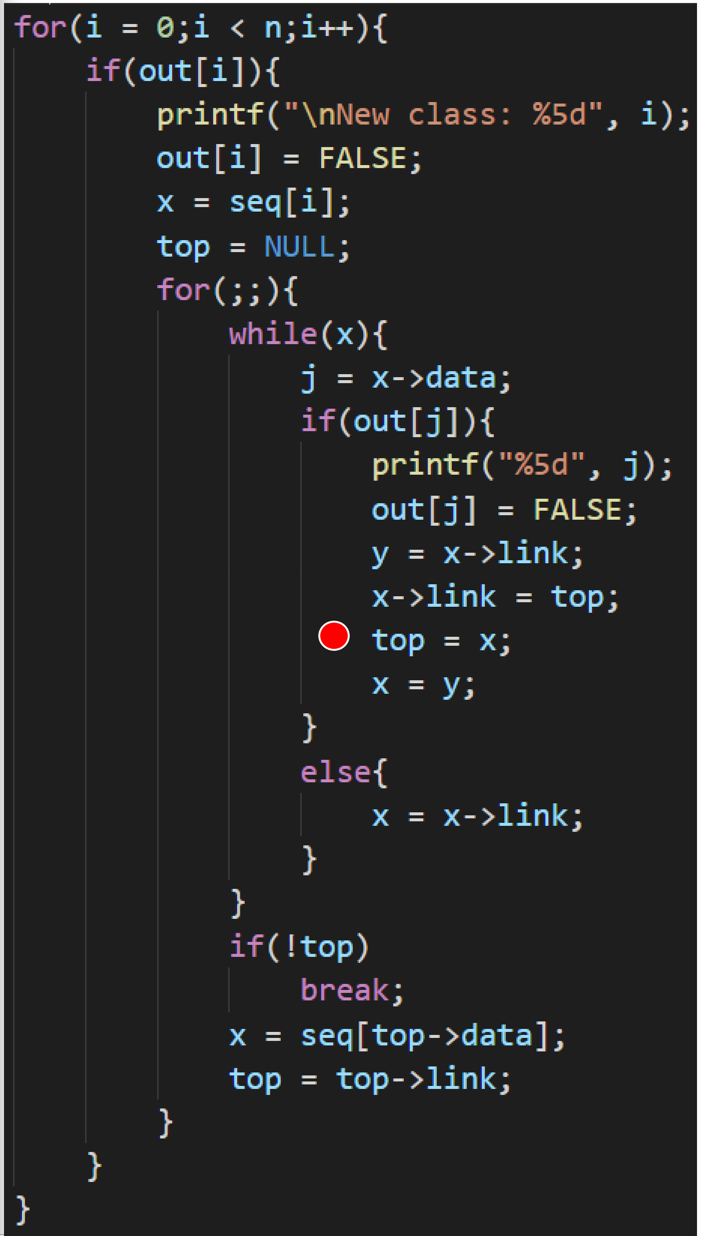
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5

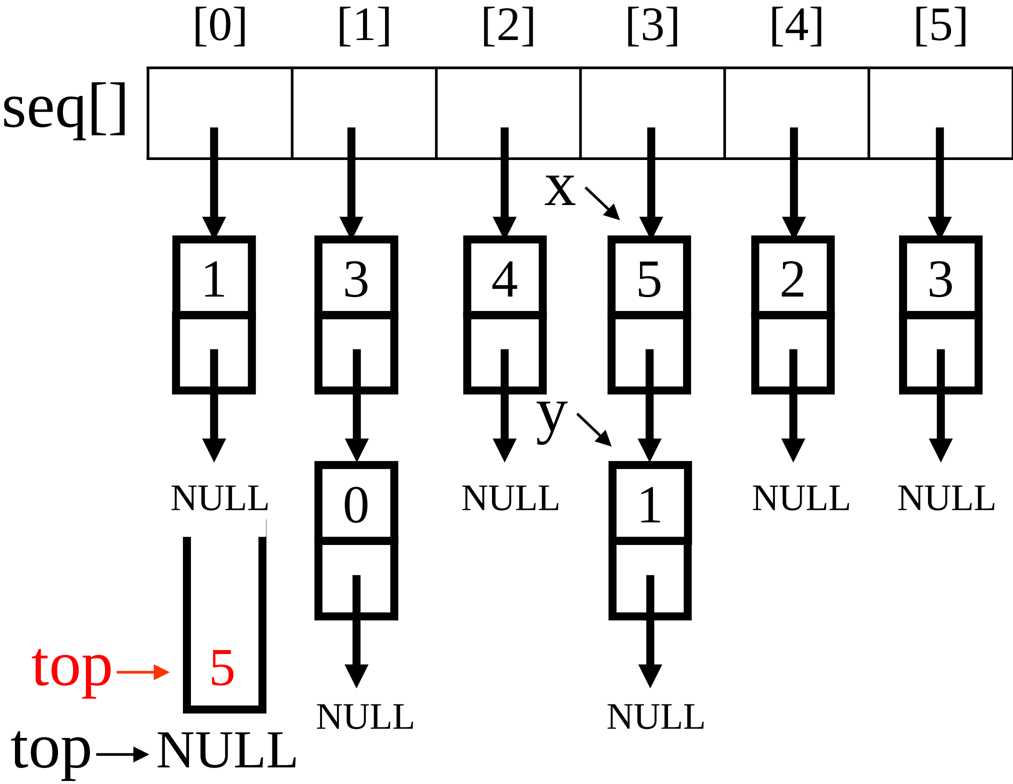


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

out[]

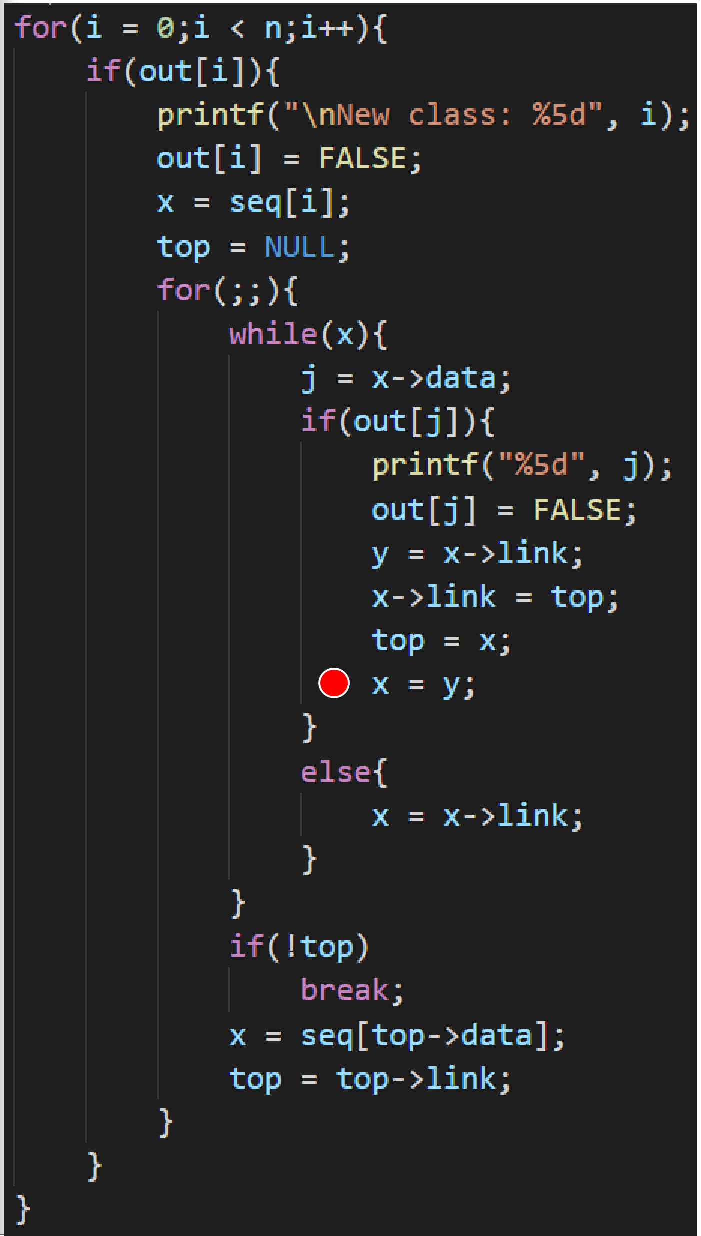
[0]	[1]	[2]	[3]	[4]	[5]
0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

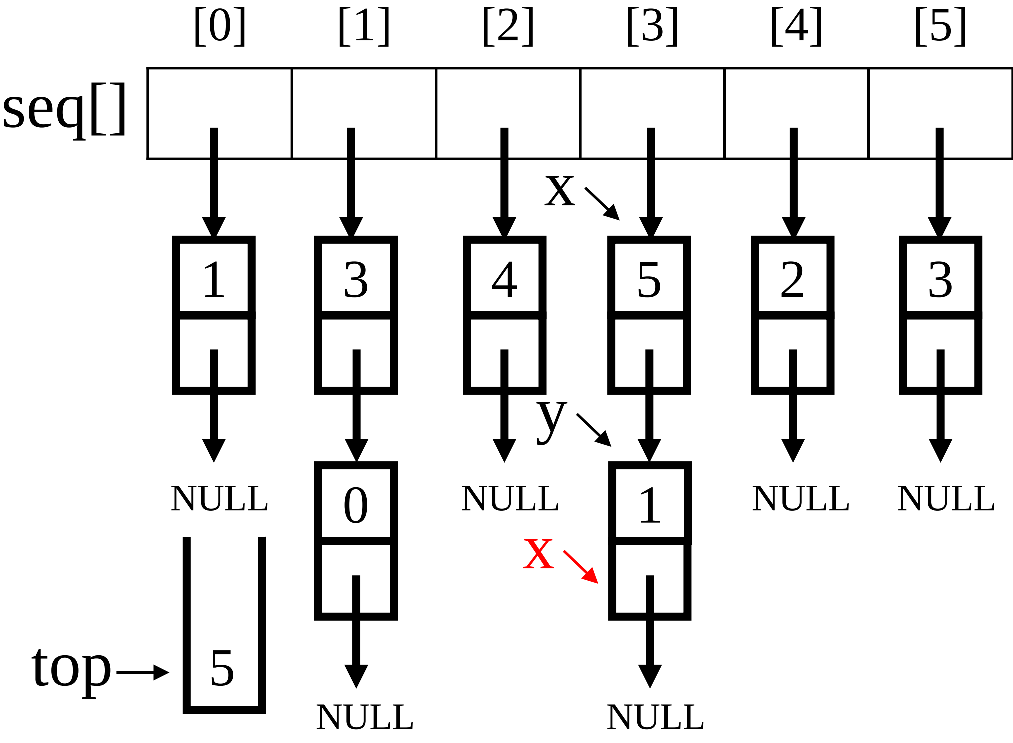
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

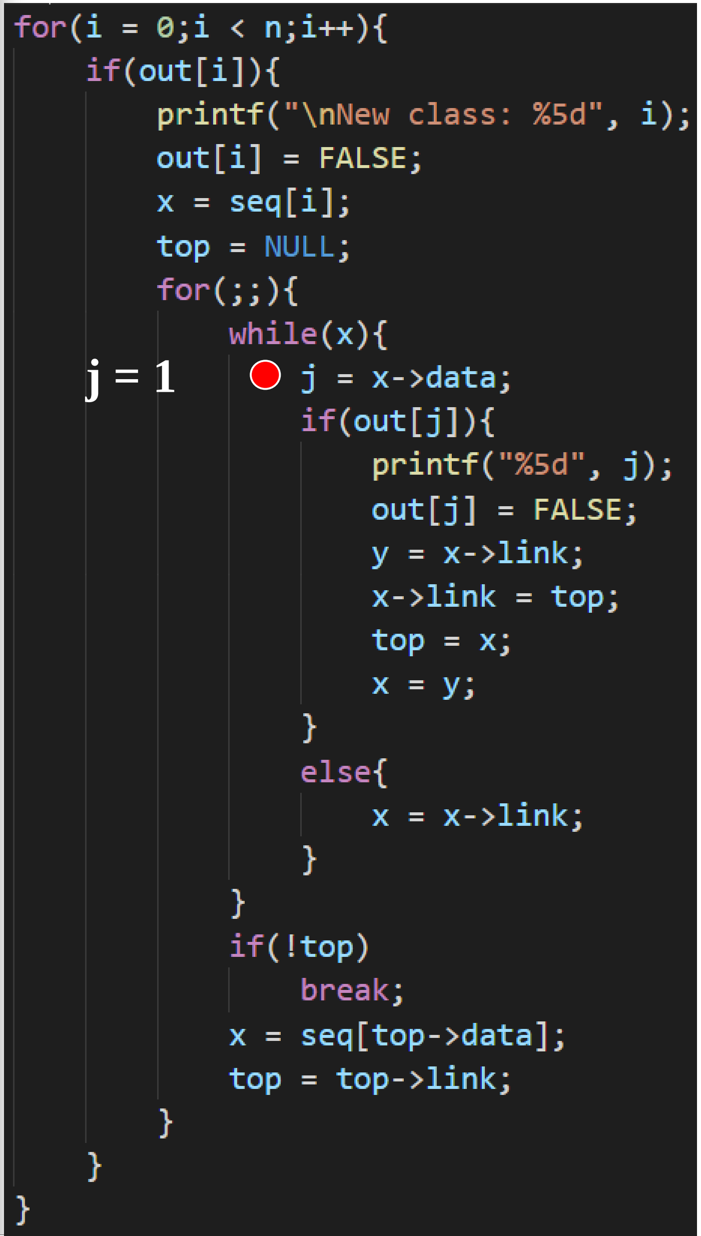
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

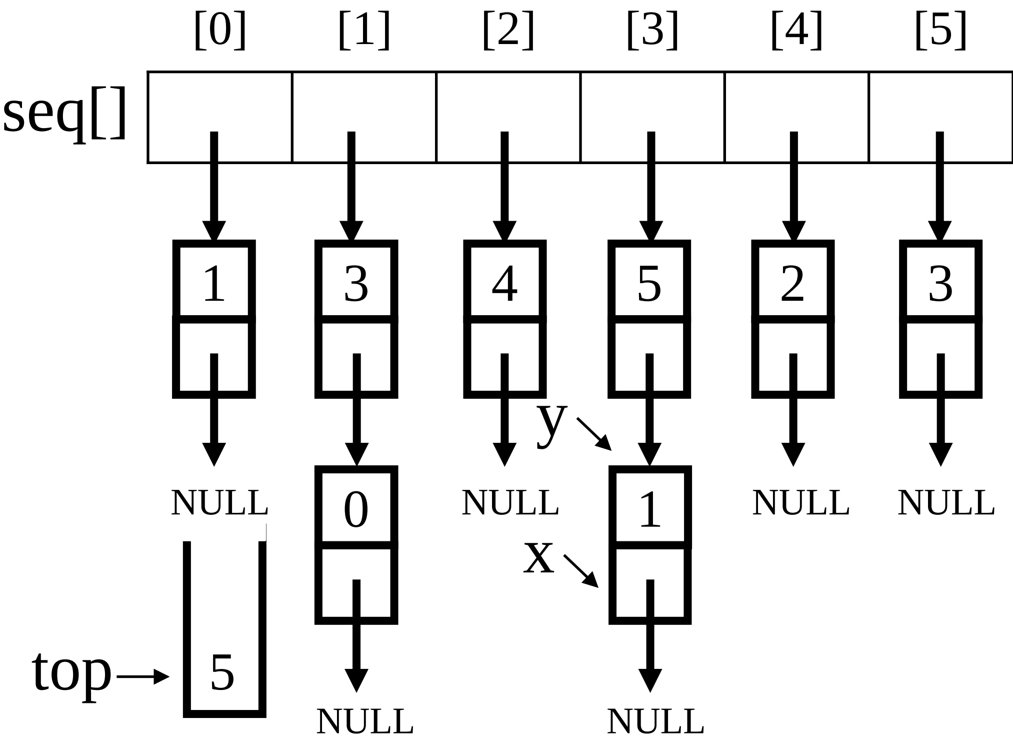
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

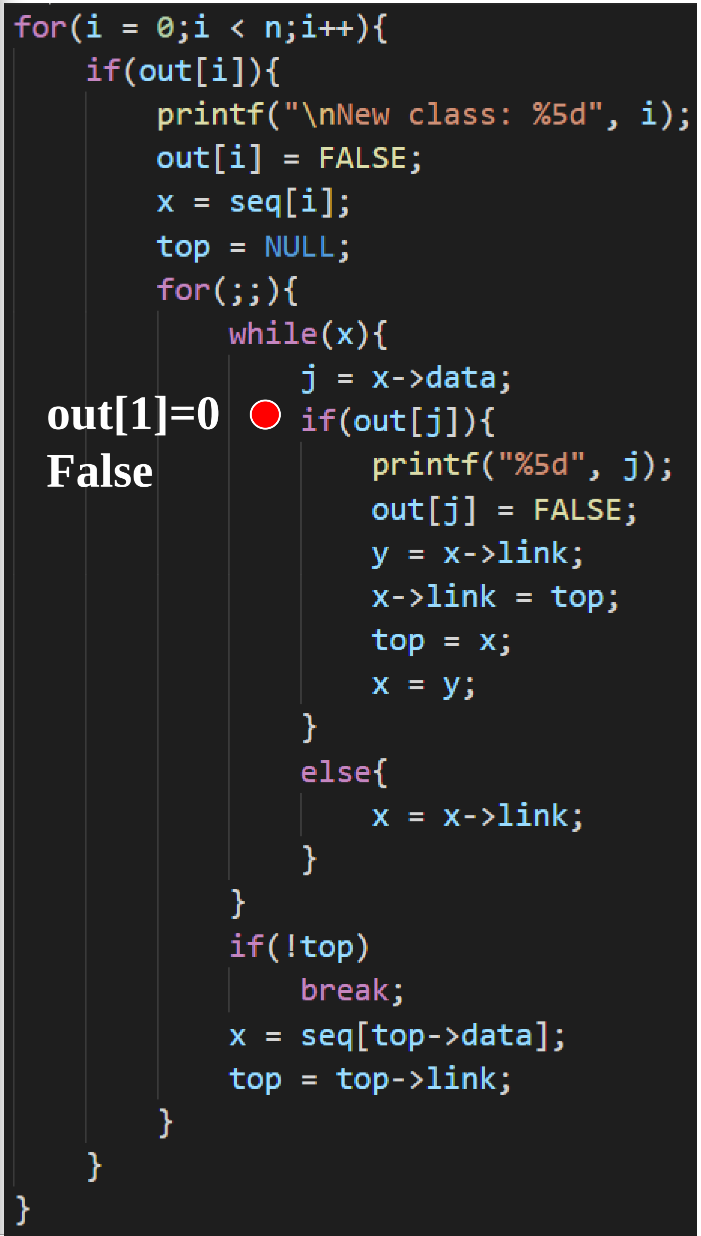
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5



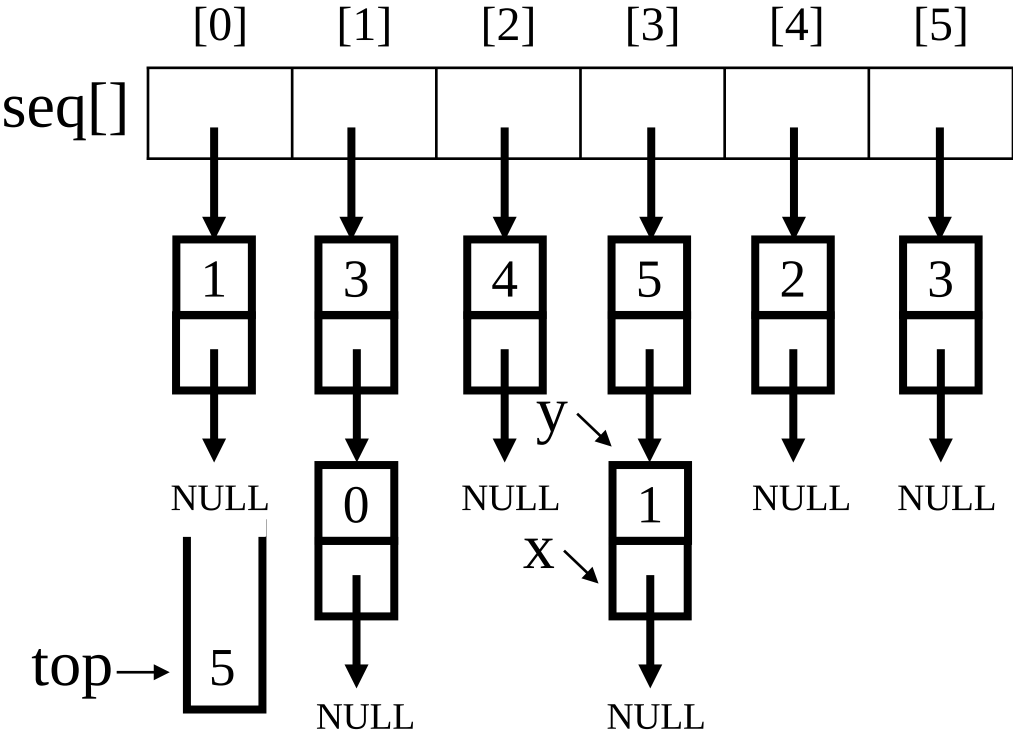
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[1]=0

False

i = 0 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

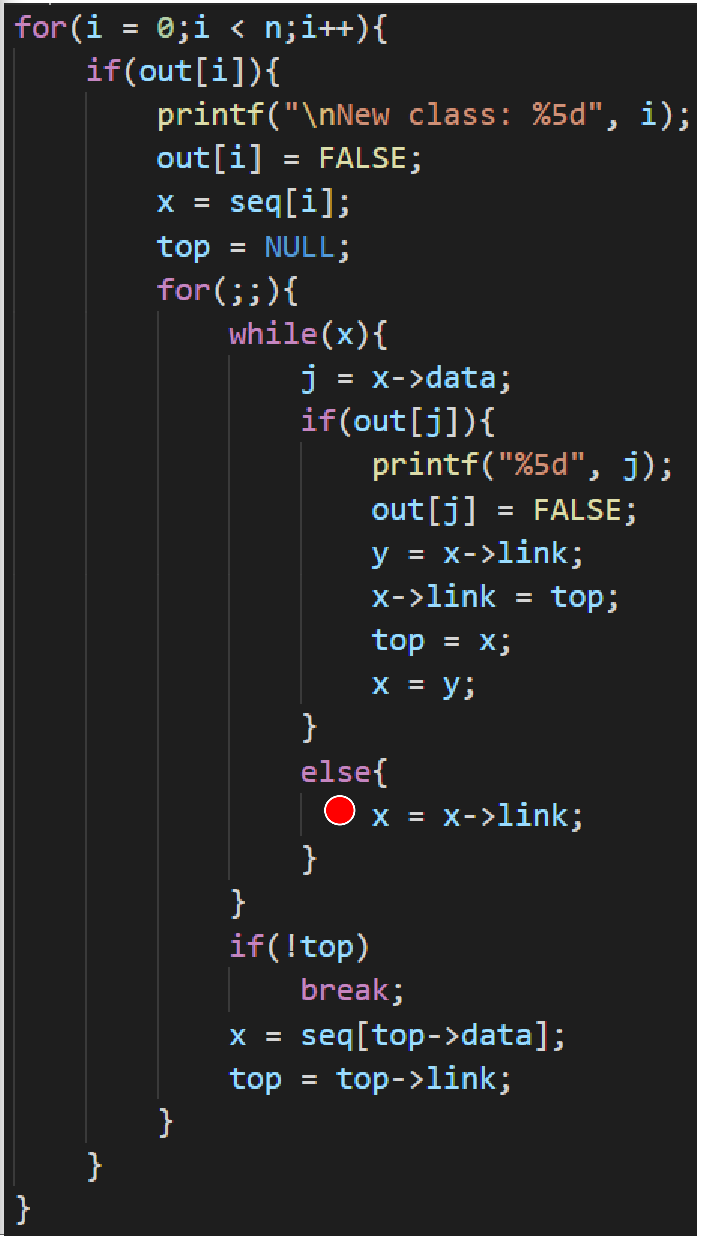
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

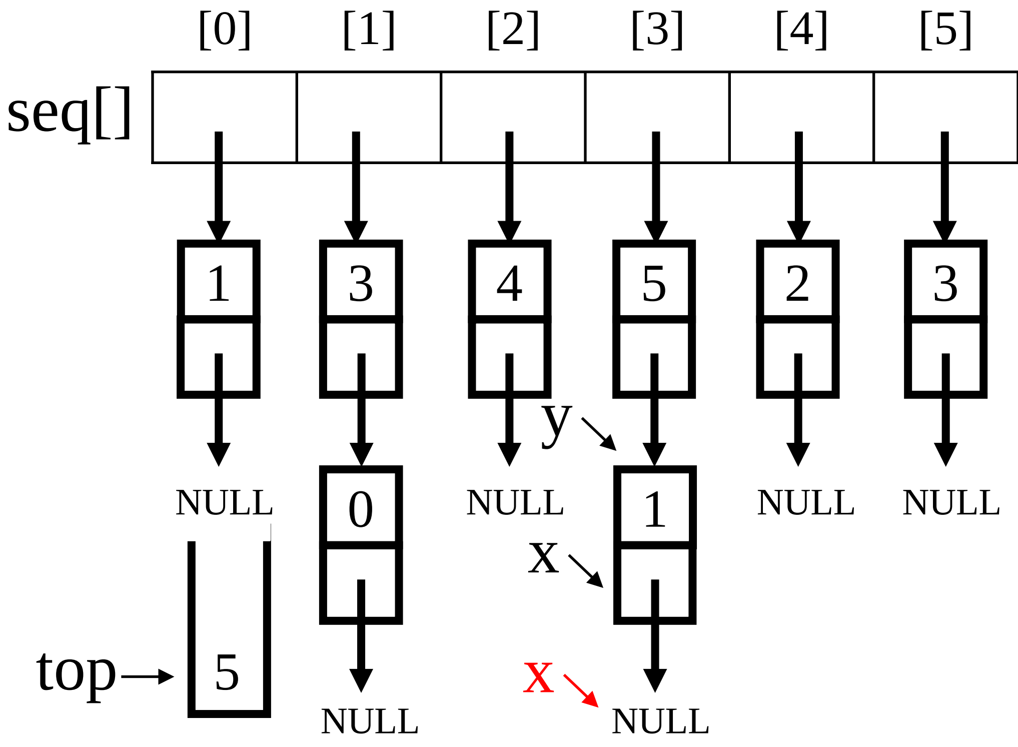
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

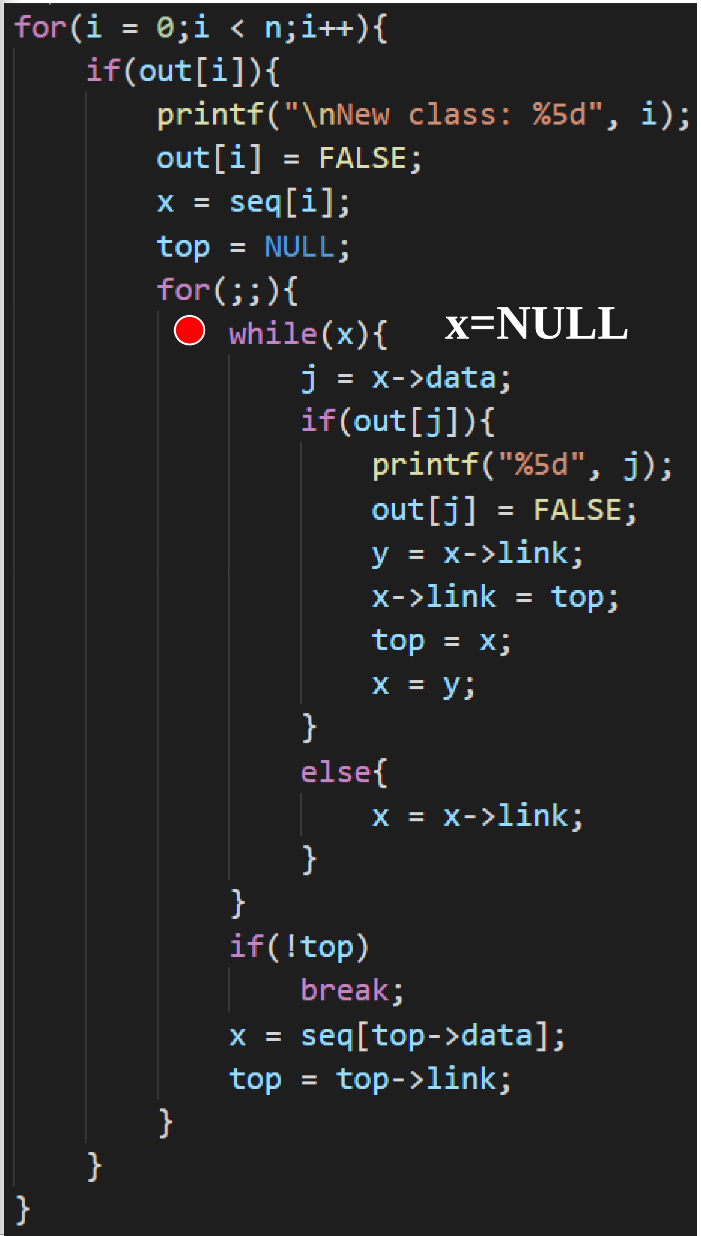
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0   1   3   5

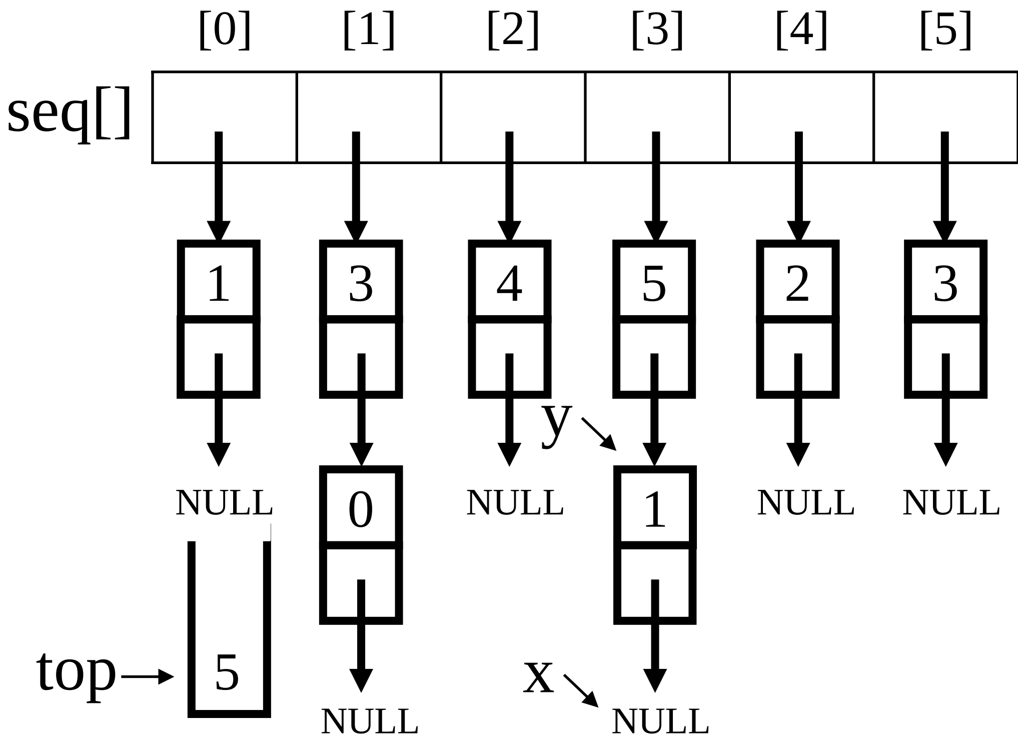


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

x=NULL

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

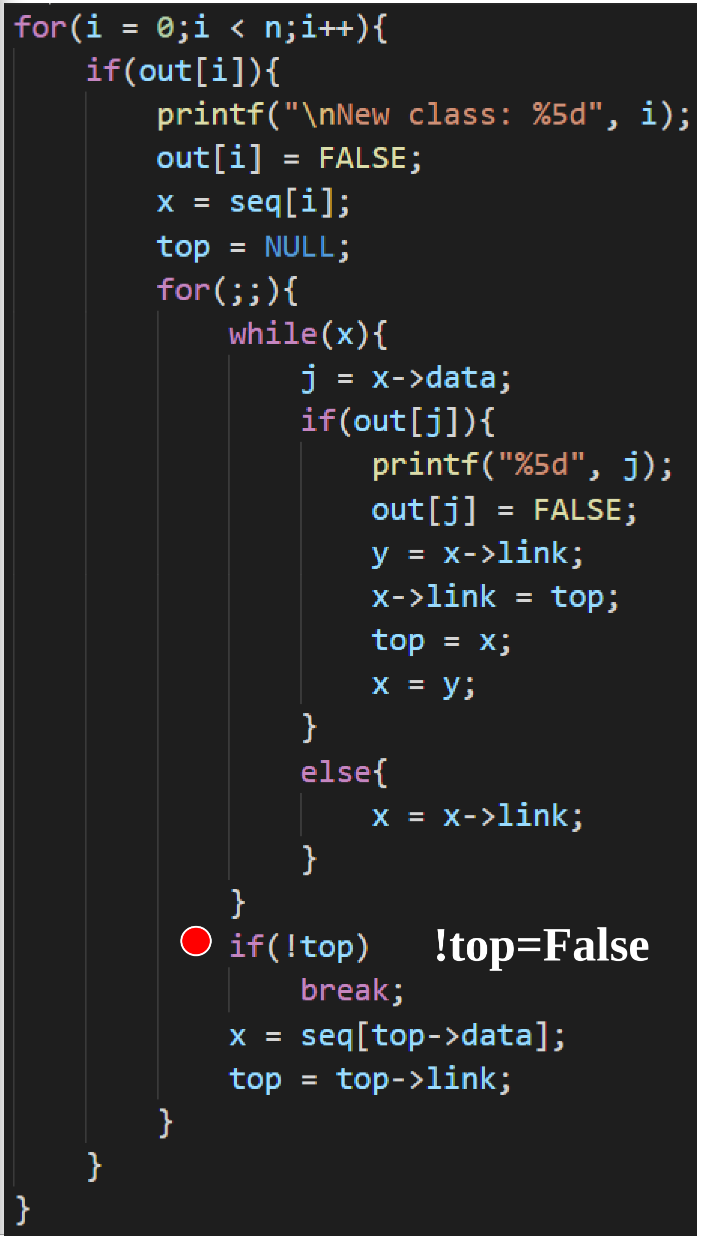
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0   1   3   5

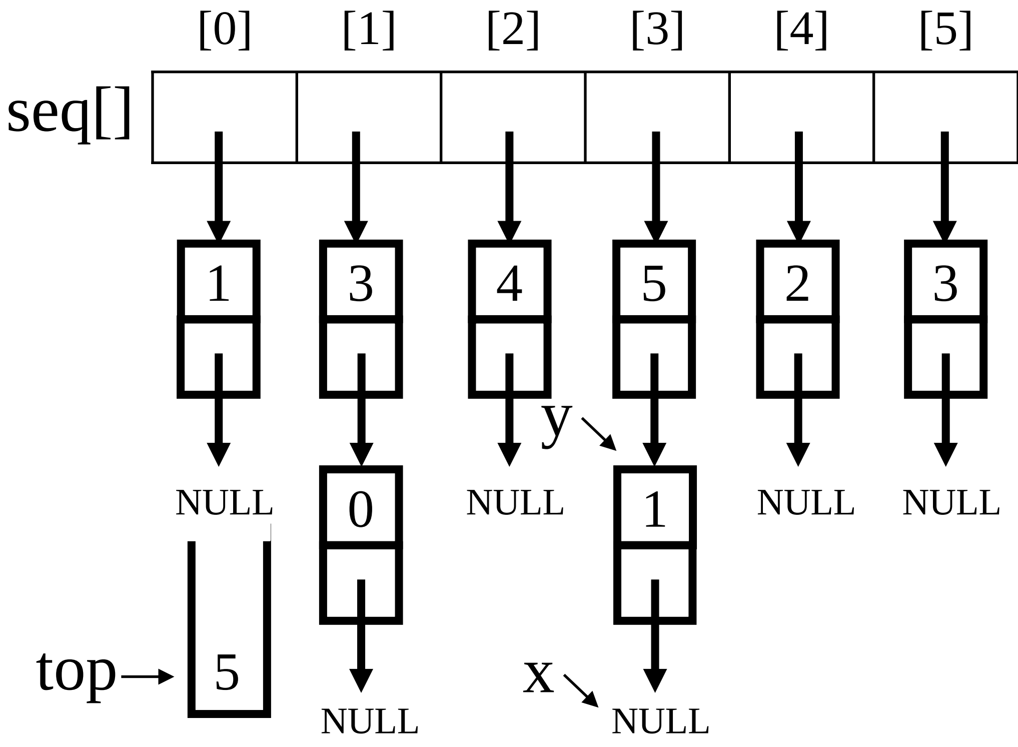


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

!top=False

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

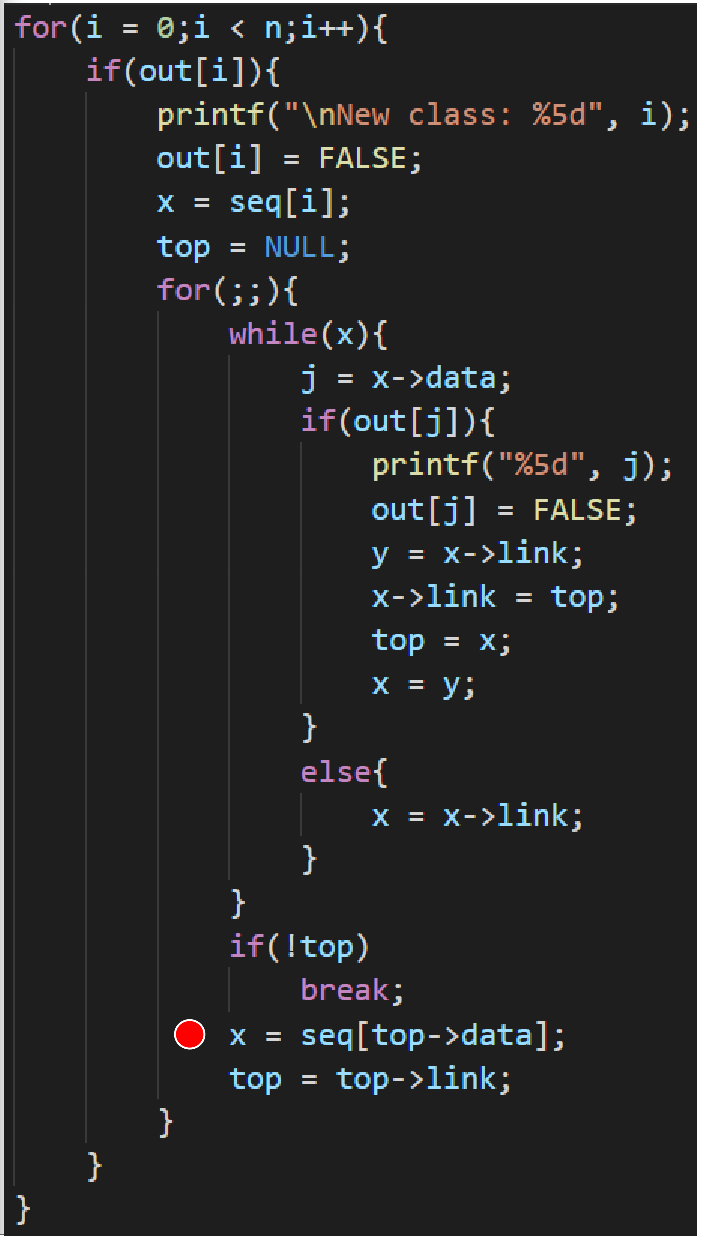
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

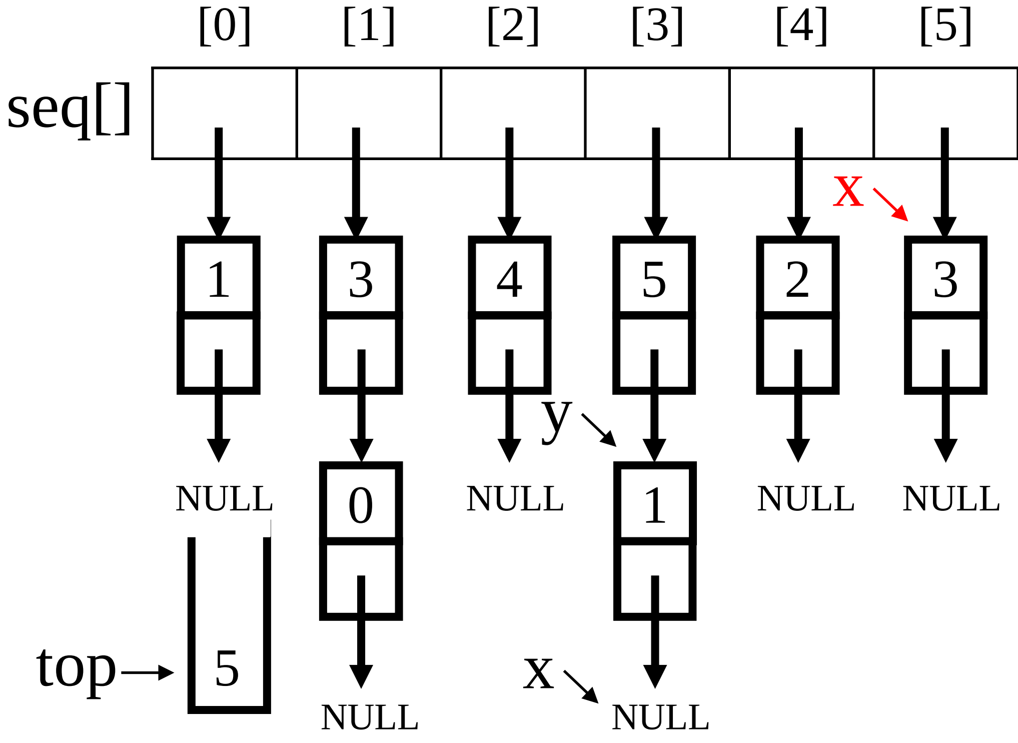
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

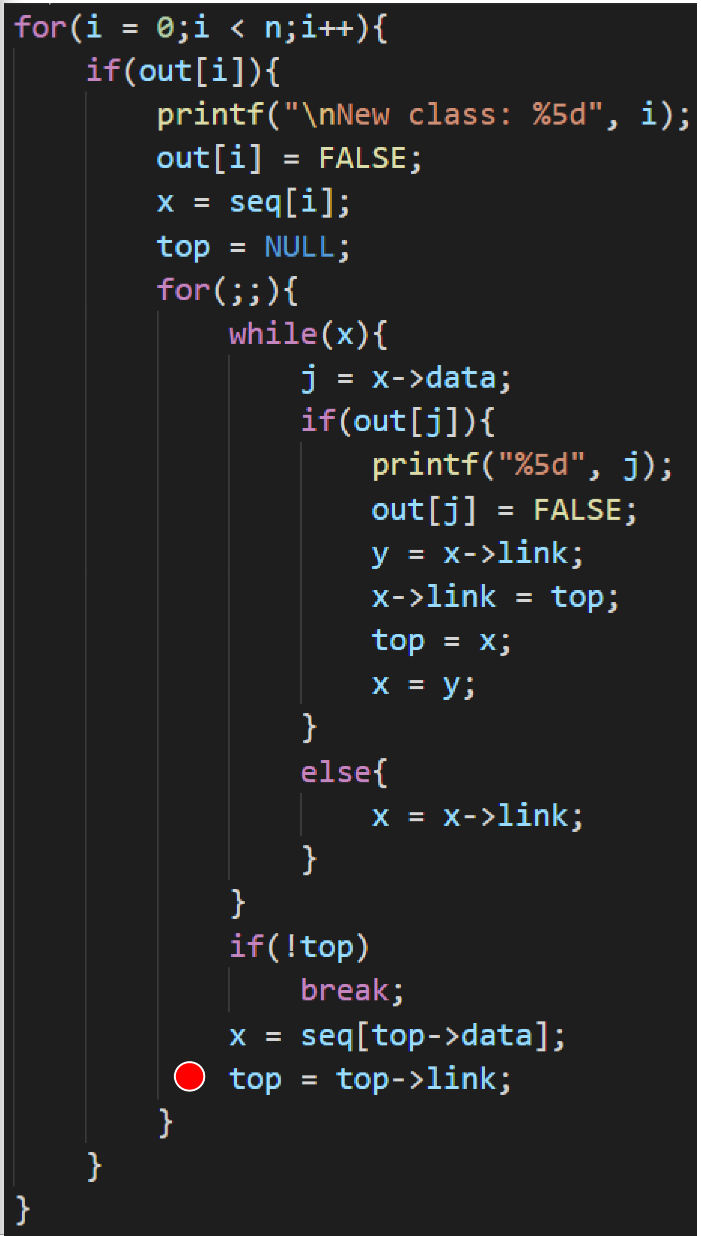
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

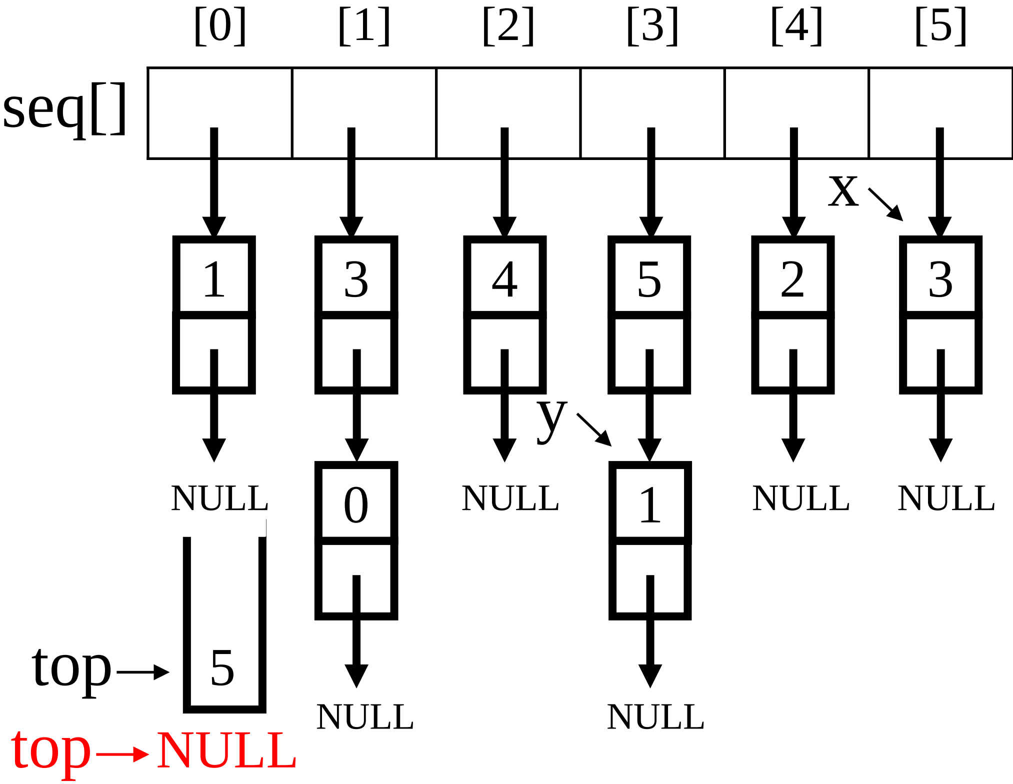
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0 0 ≡ 1 1 ≡ 3 3 ≡ 5 2 ≡ 4

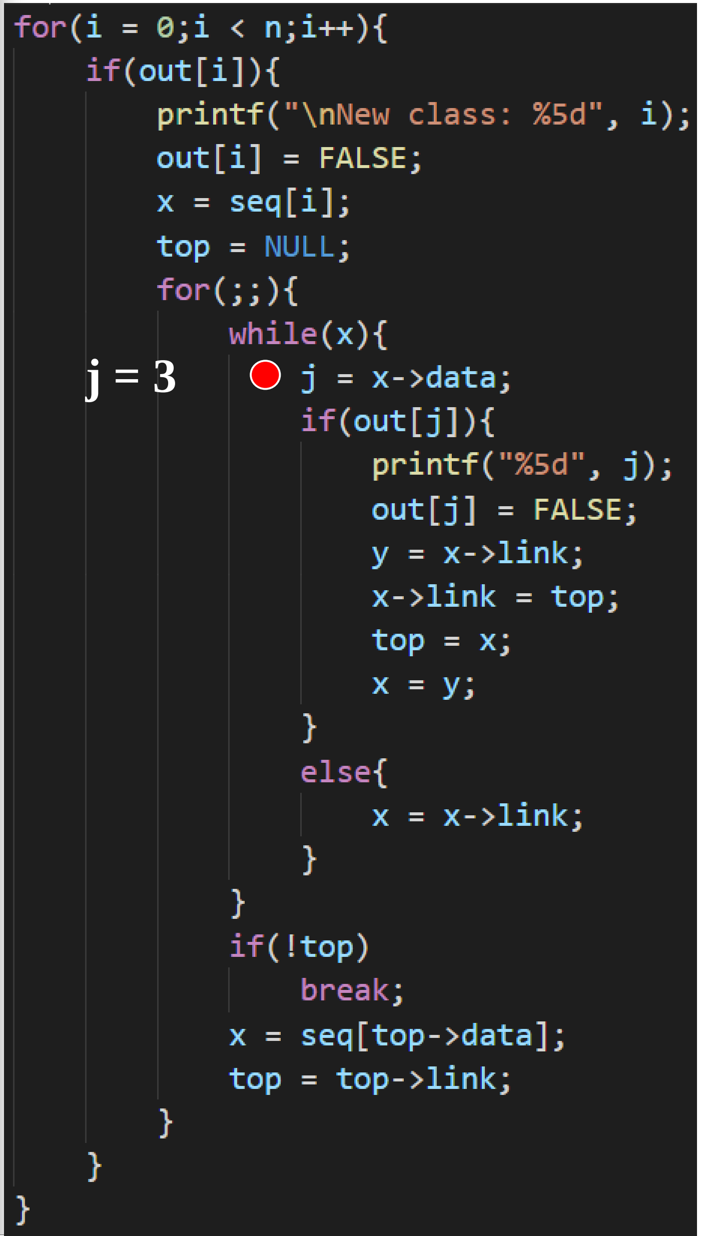
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5

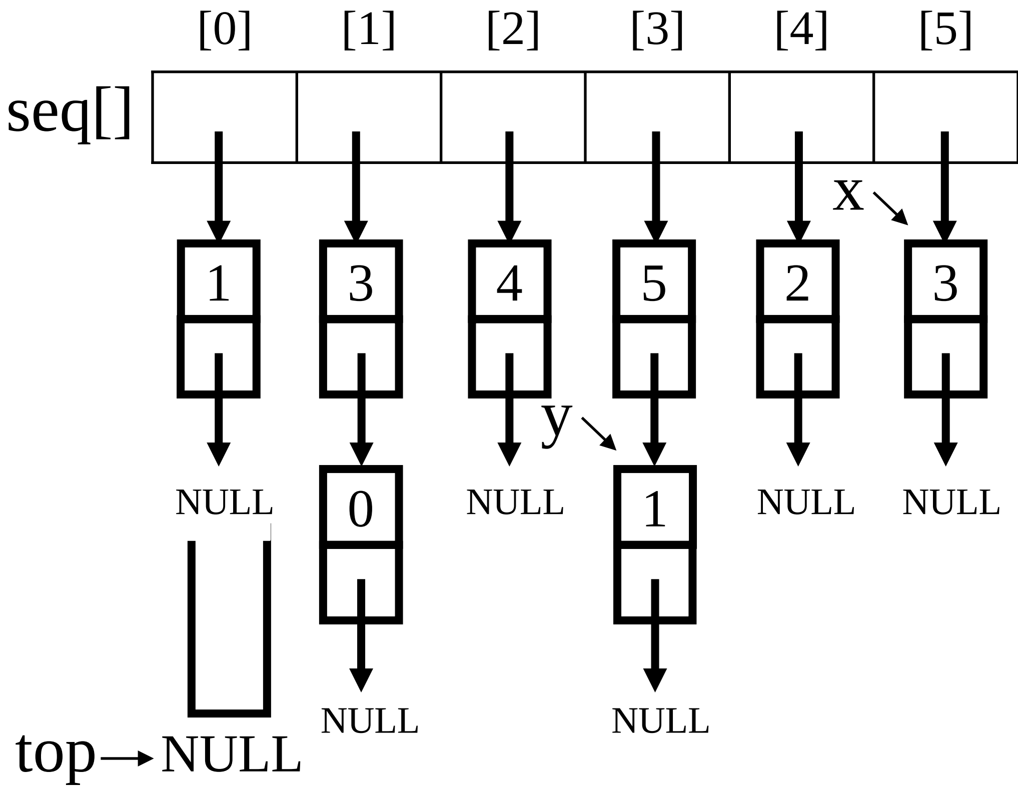


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

j = 3

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

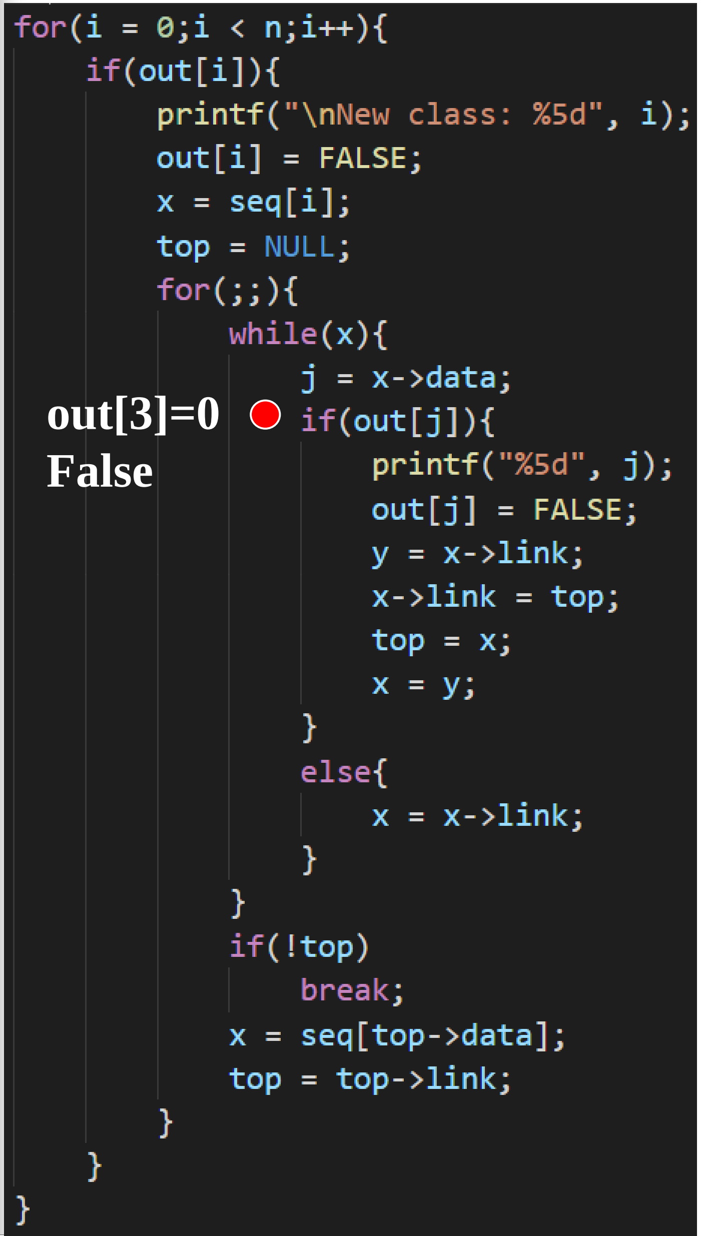
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5



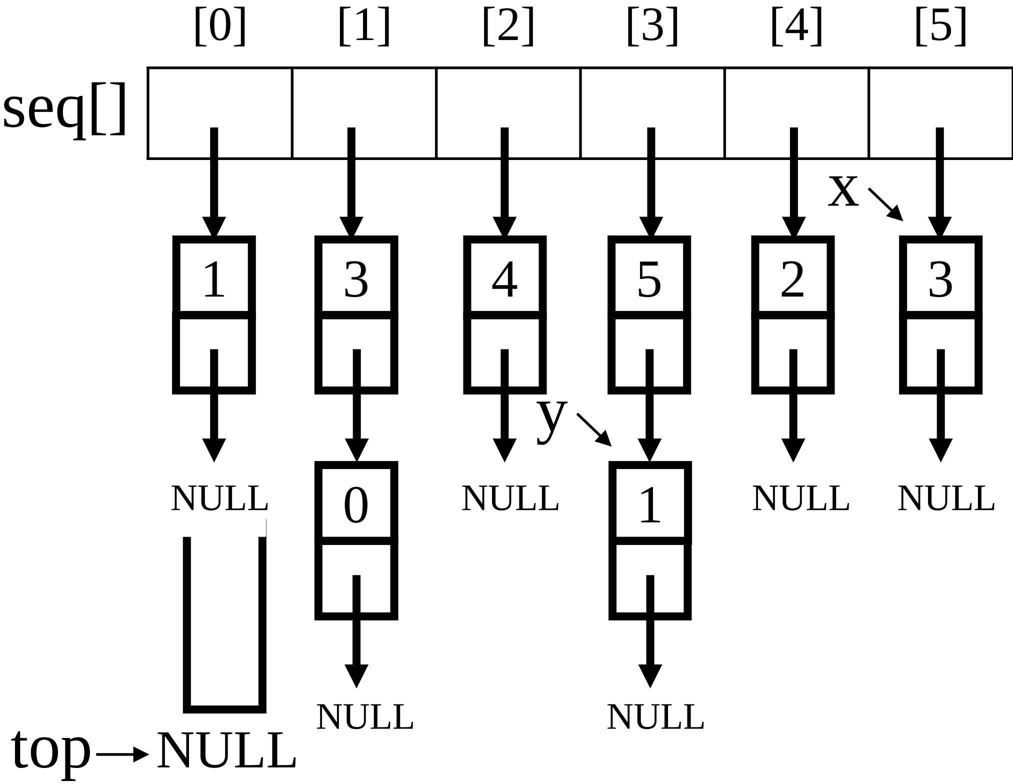
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[3]=0

False

i = 0 0 ≡ 1 1 ≡ 3 3 ≡ 5 2 ≡ 4

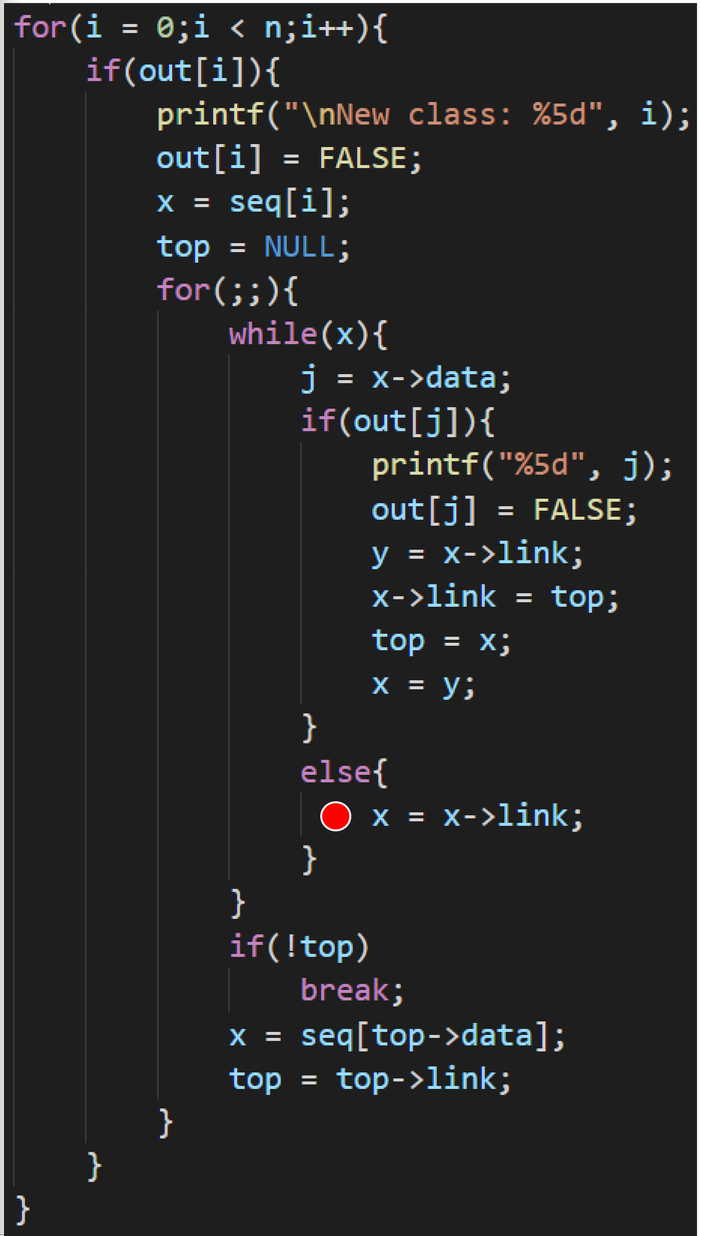
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

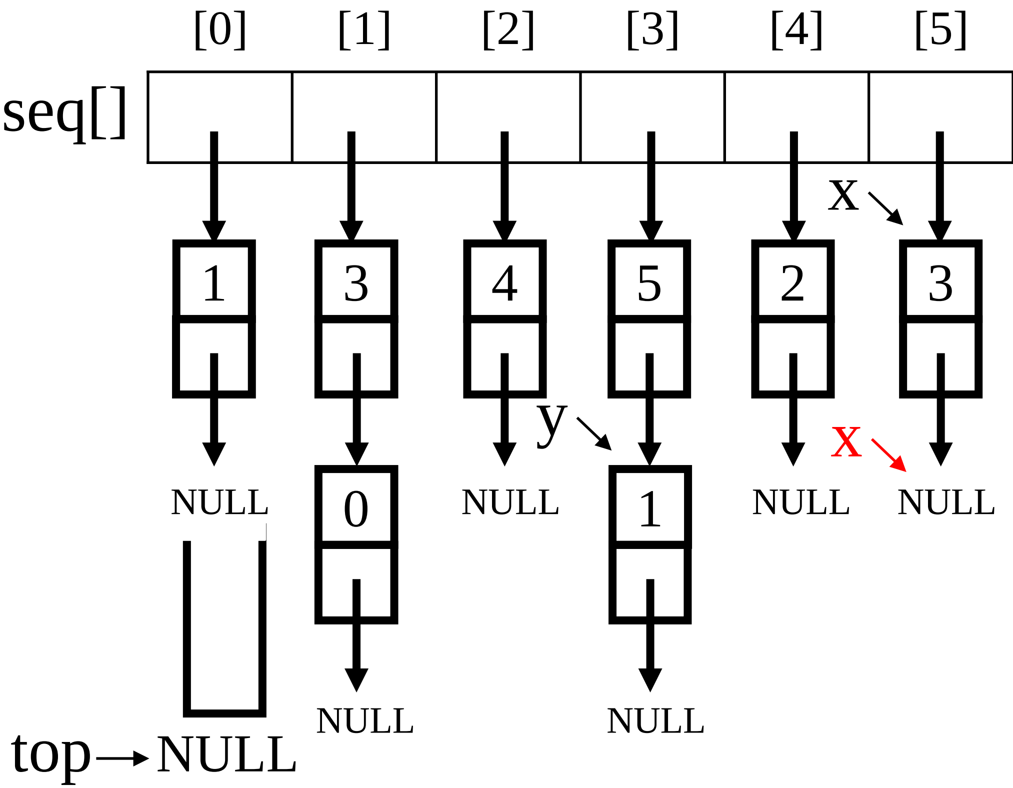
```
New class:    0    1    3    5
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

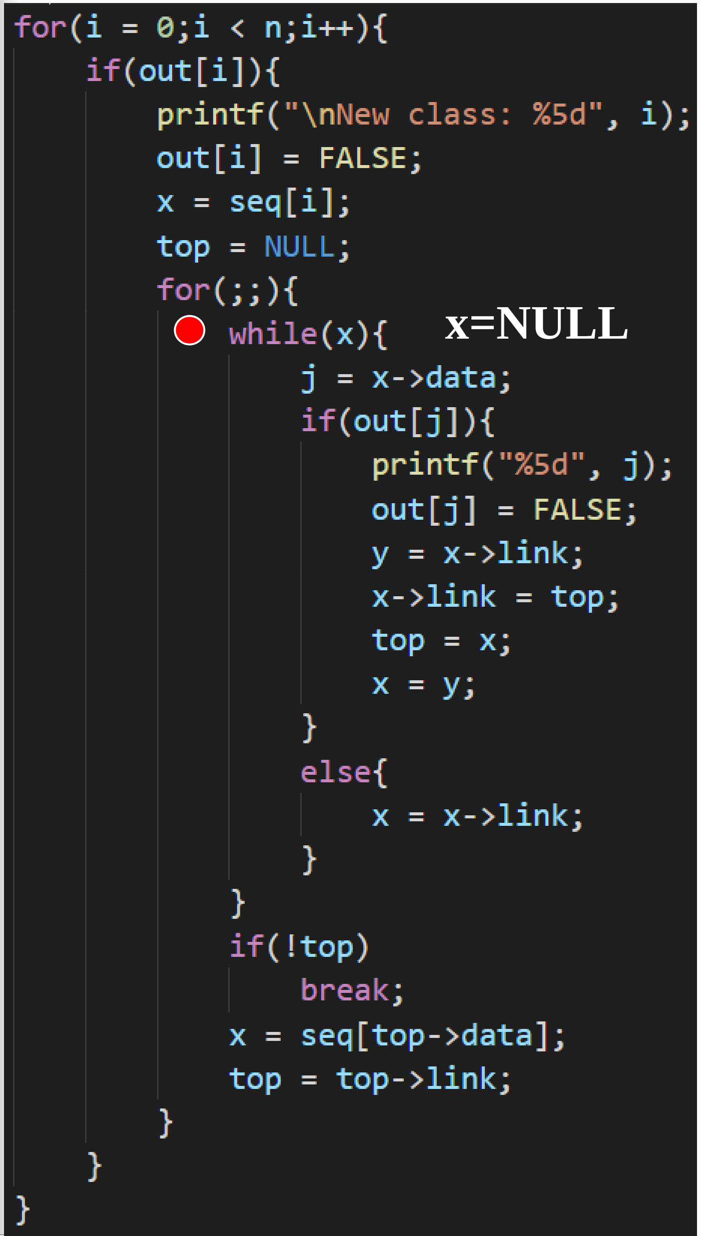
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5

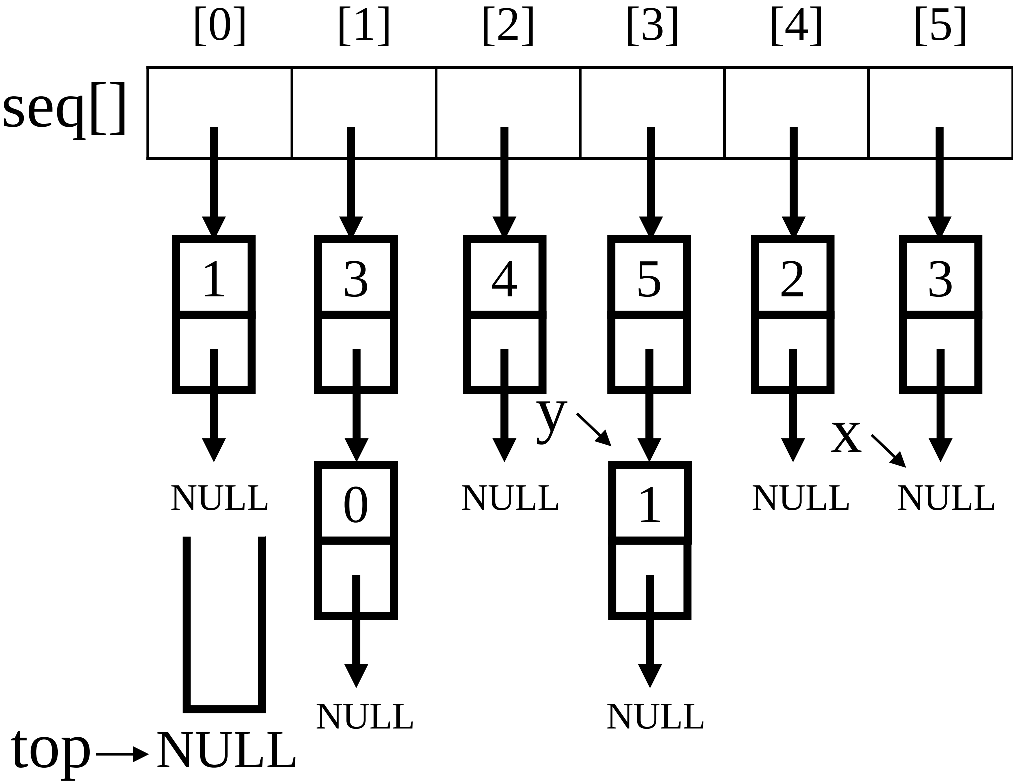


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

x=NULL

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

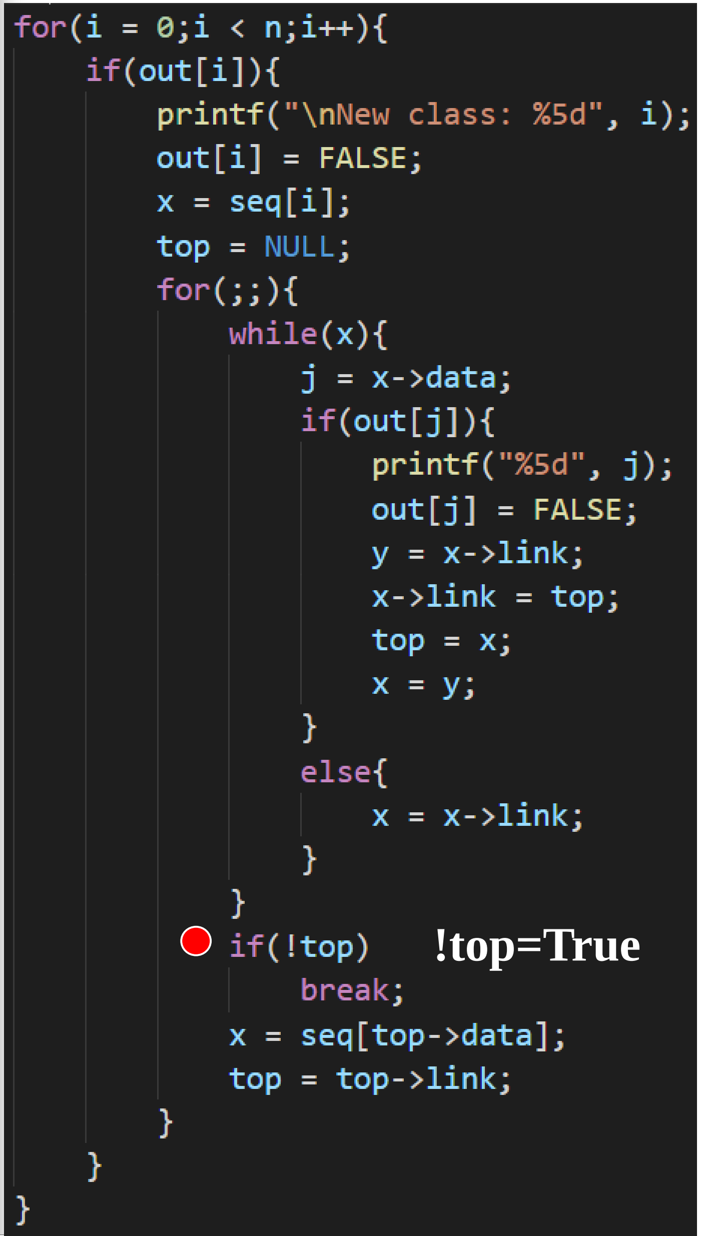
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

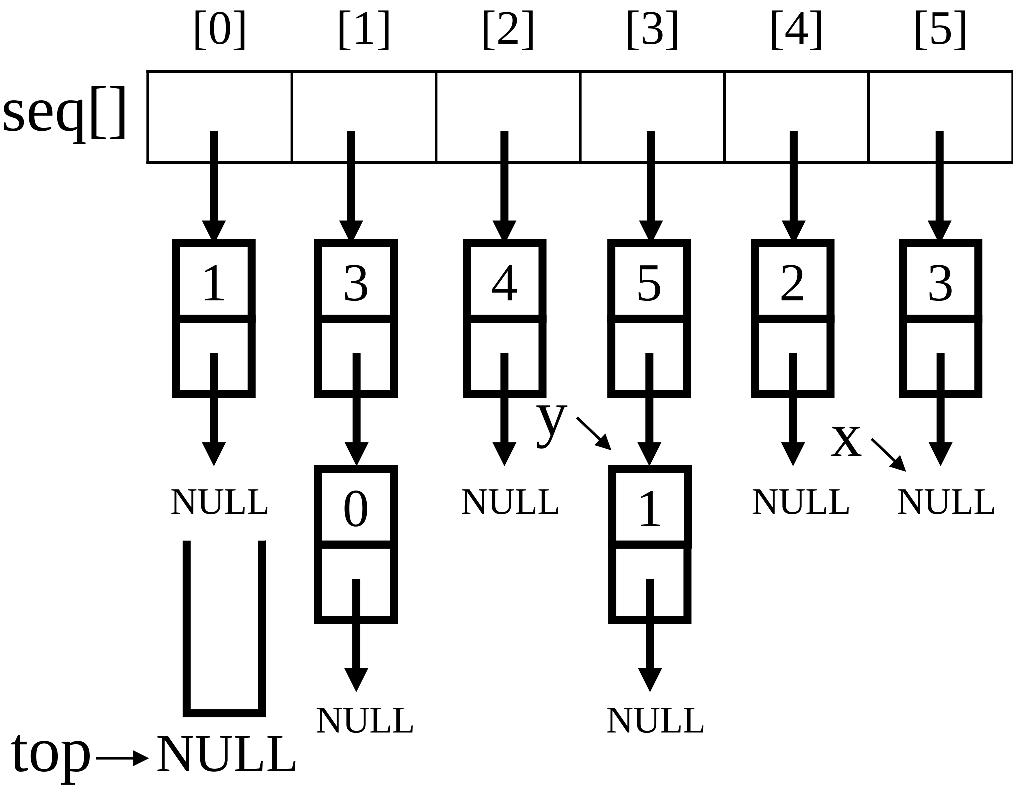
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
●           if(!top)     !top=True
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

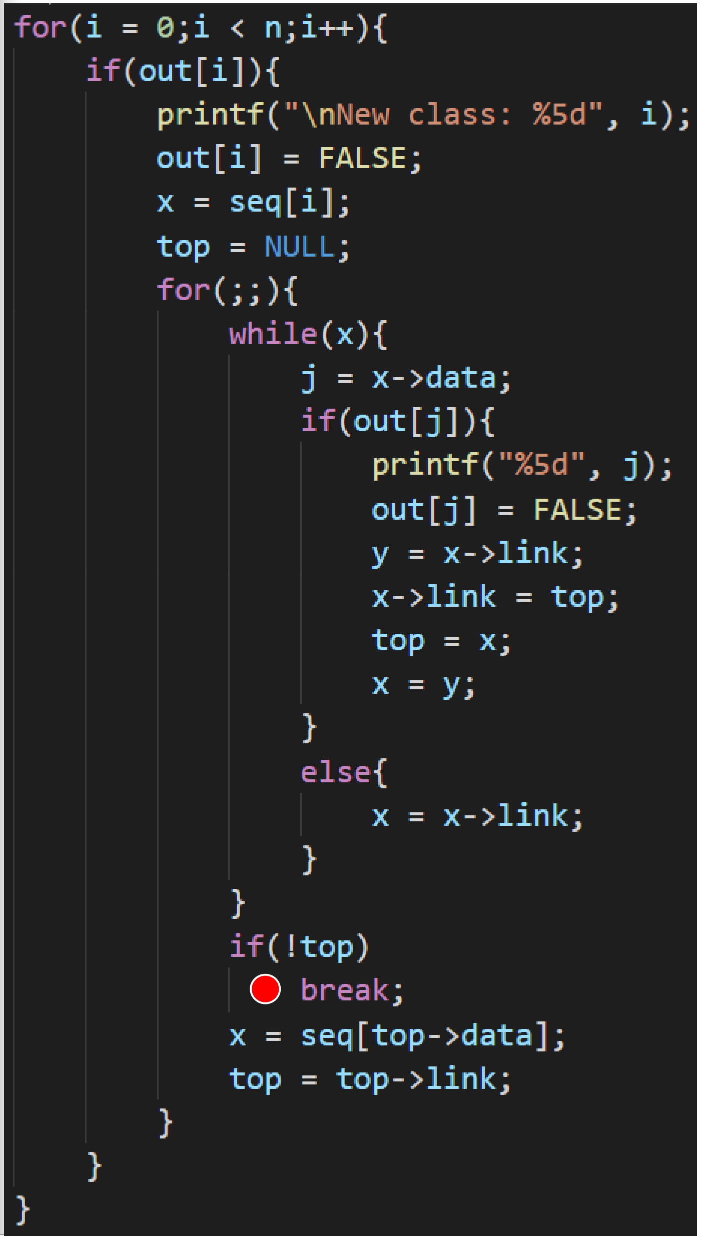
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

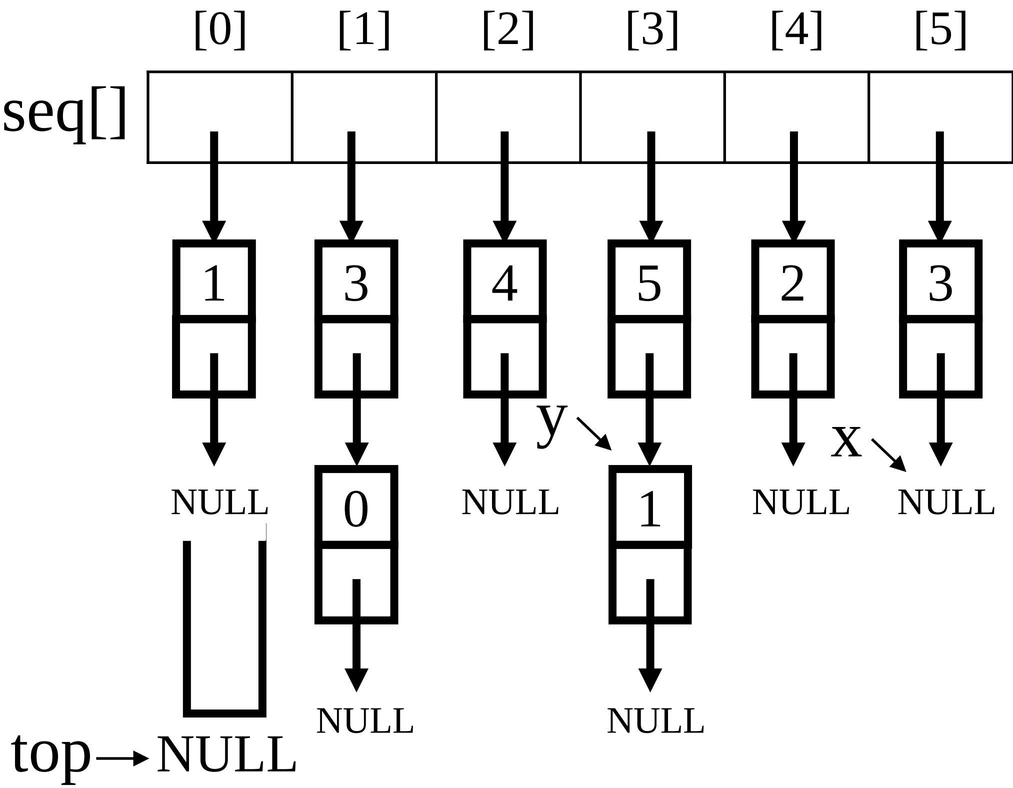
```
New class:    0    1    3    5
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 0   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

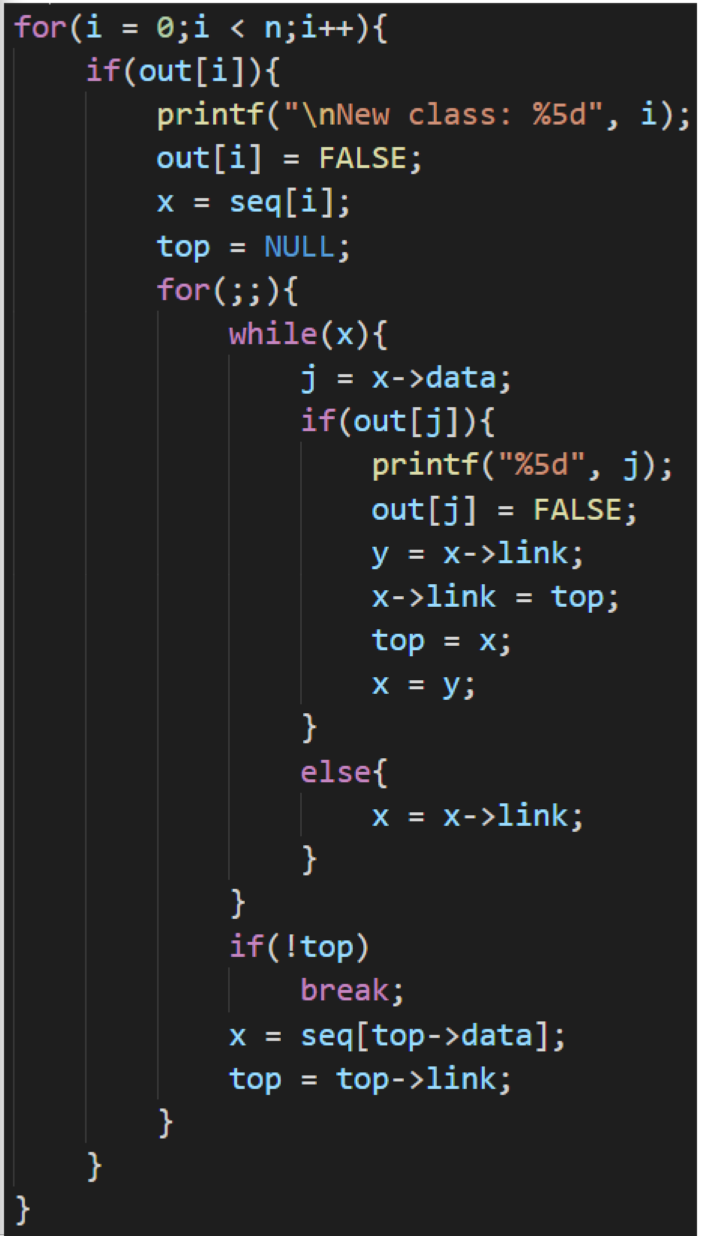
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

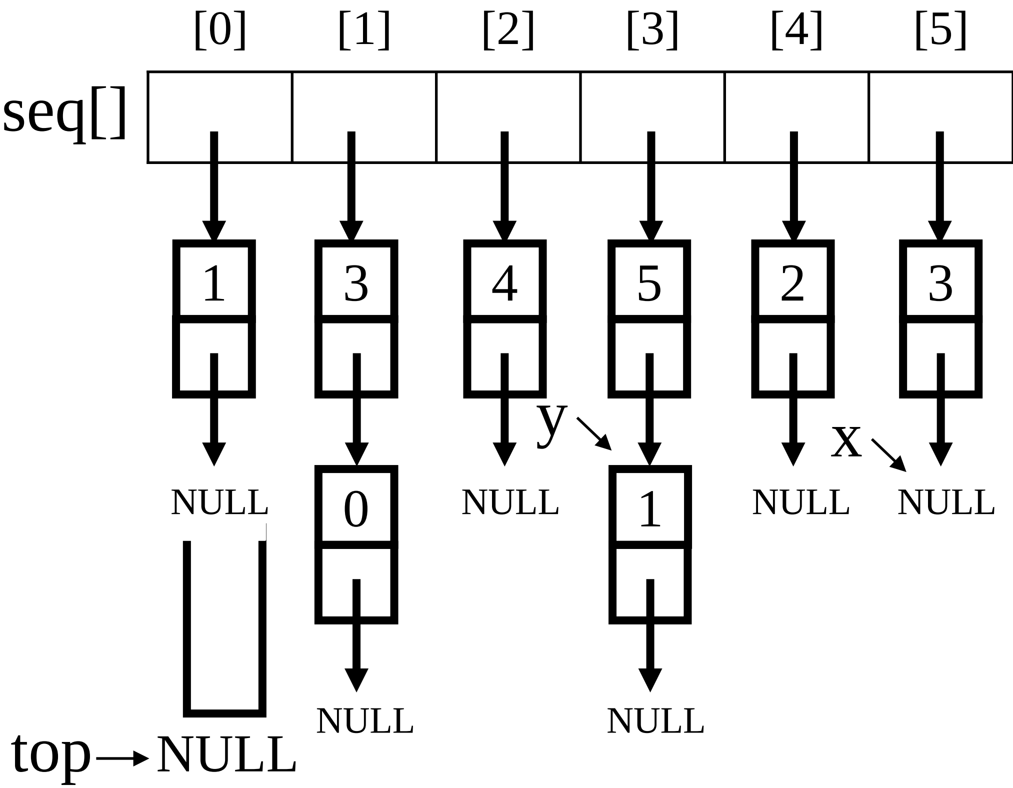
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 1   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

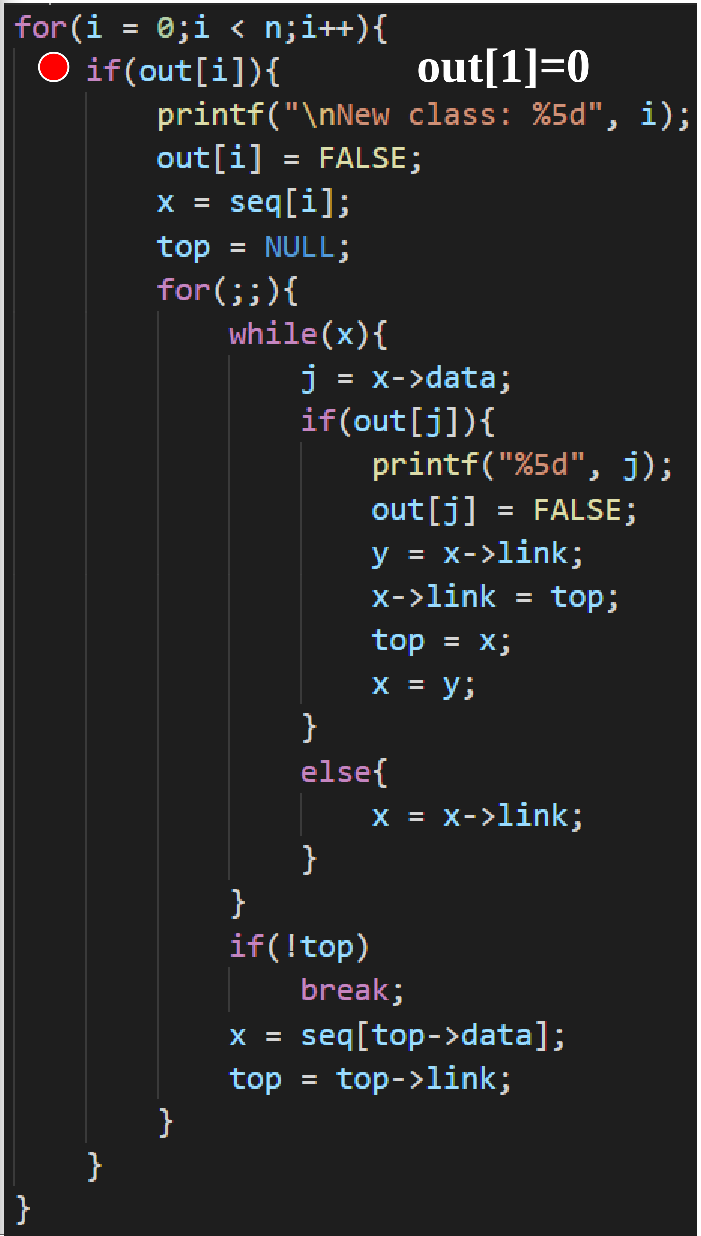
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5

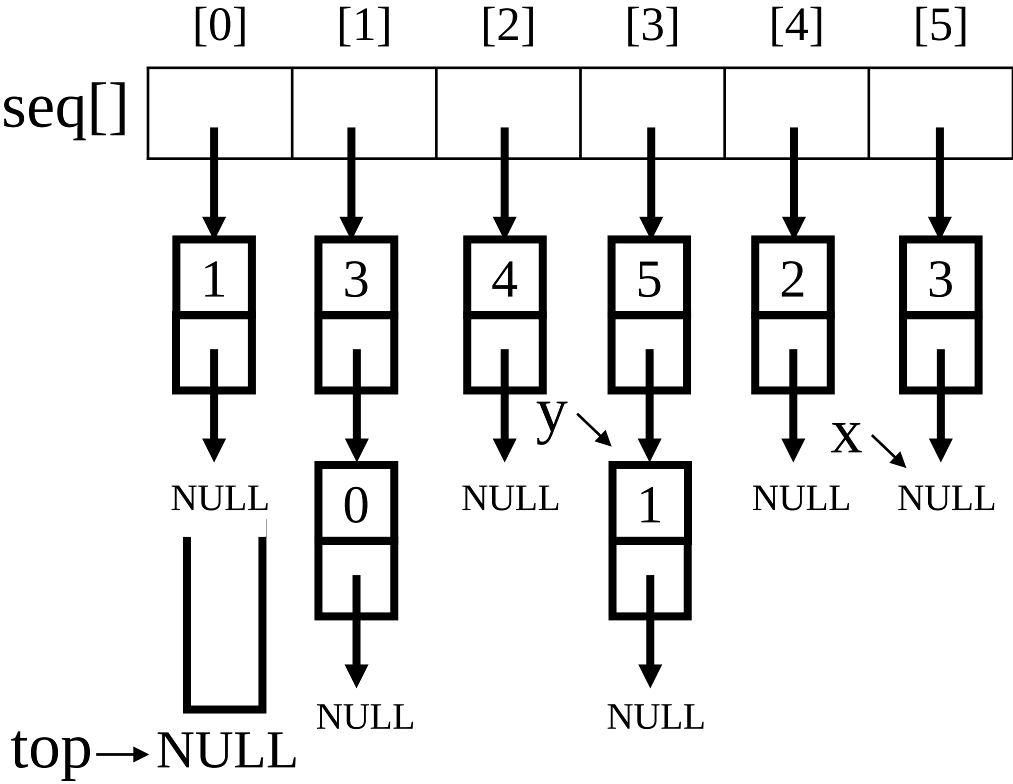


```
for(i = 0;i < n;i++){
●   if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[1]=0

i = 1   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

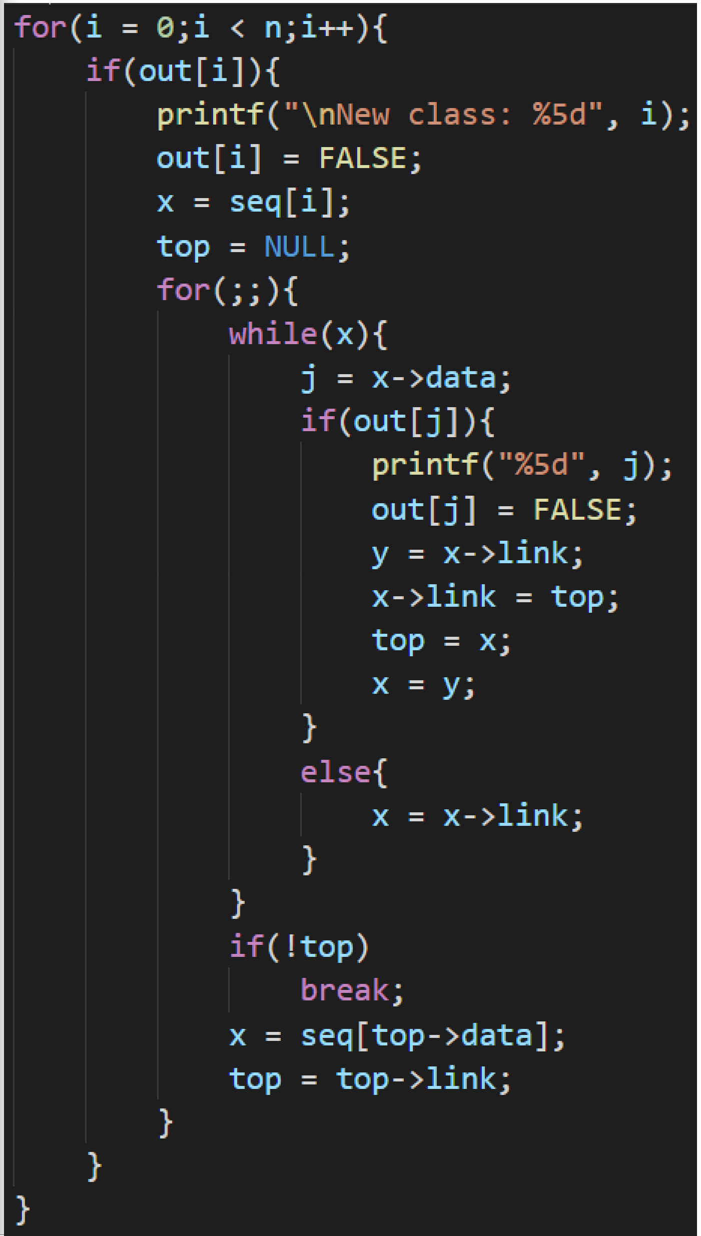
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

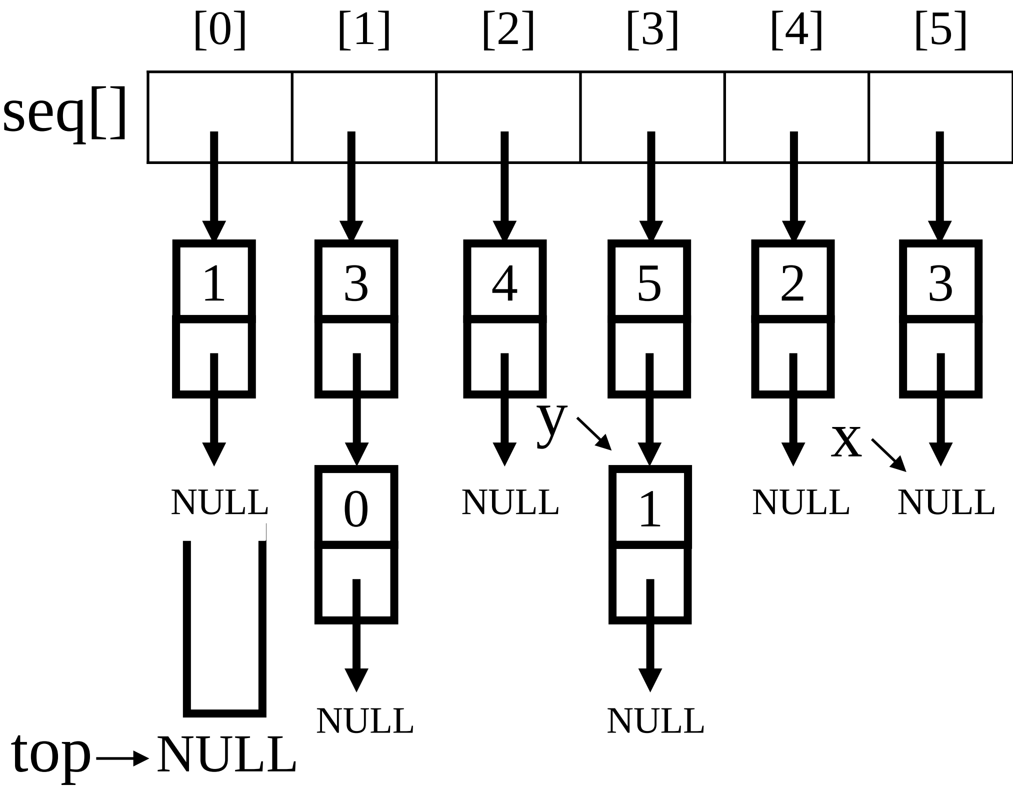
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

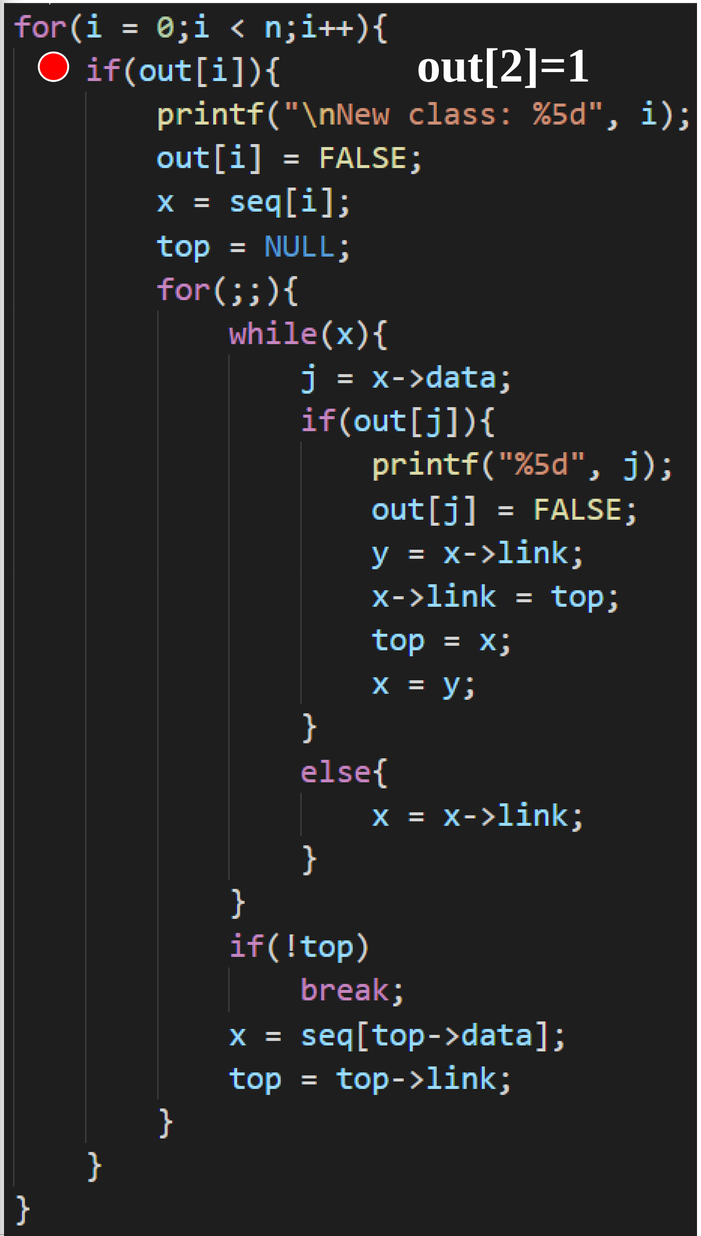
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

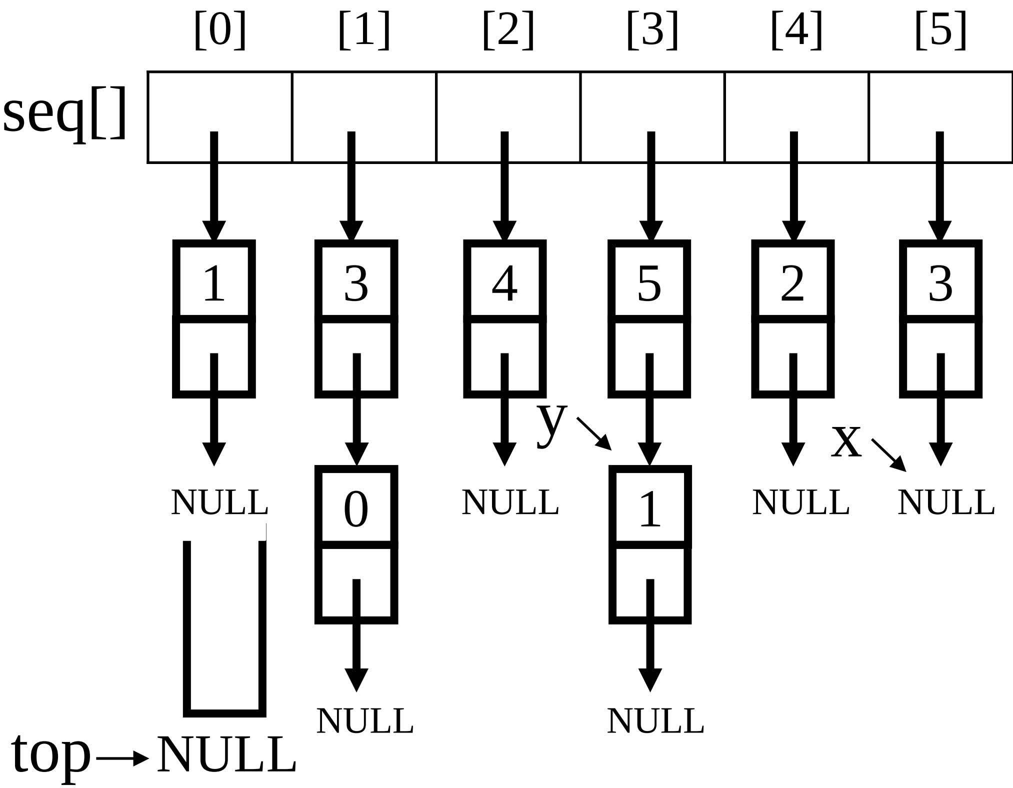
New class: 0 1 3 5



```
for(i = 0;i < n;i++){
    if(out[i]){          out[2]=1
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

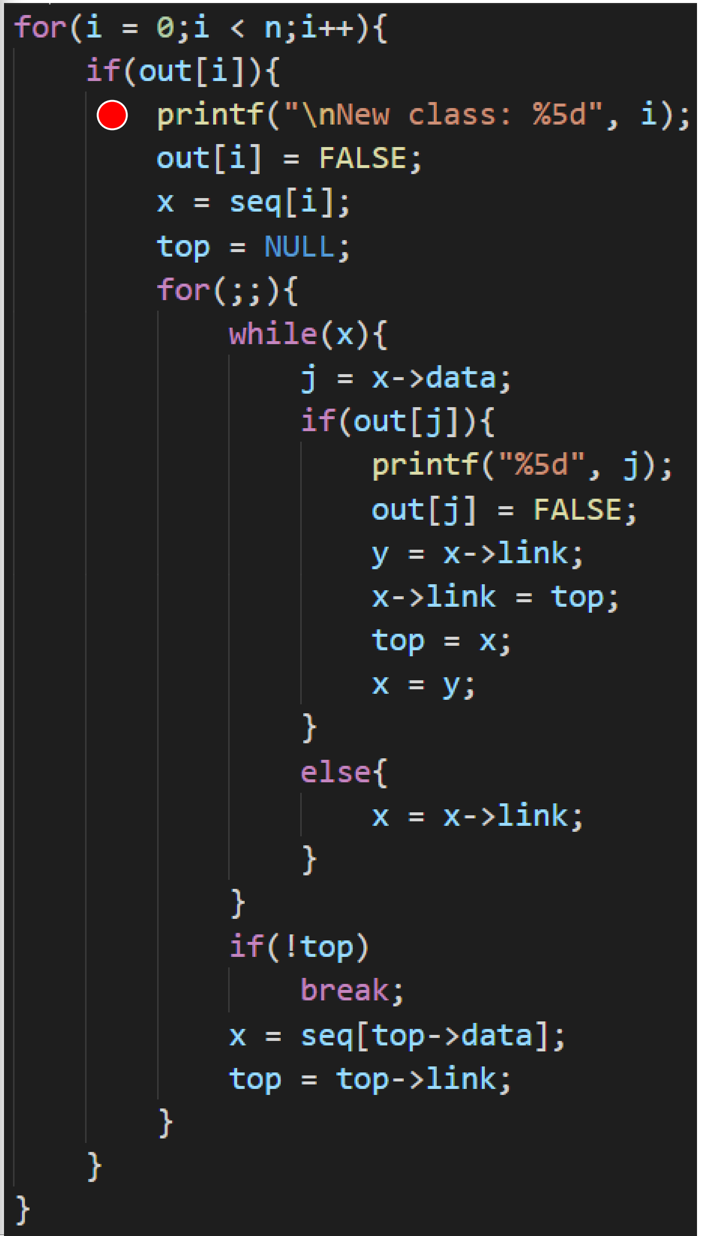
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

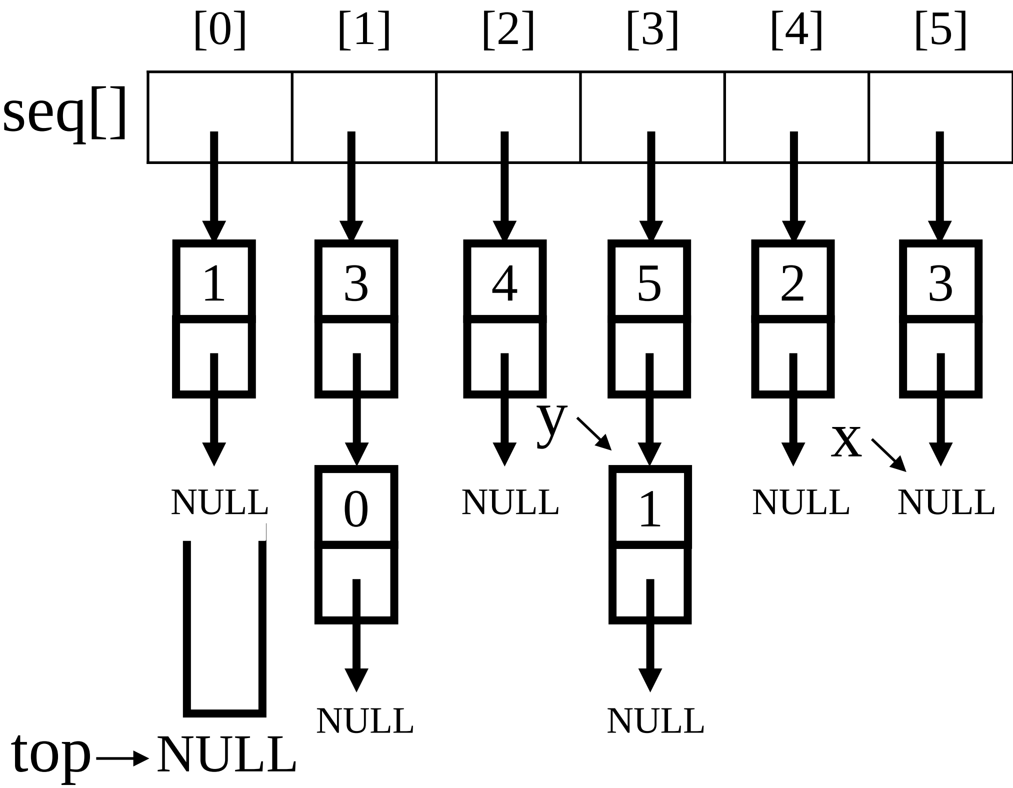
New class: 0 1 3 5
New class: 2



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

$i = 2$ $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

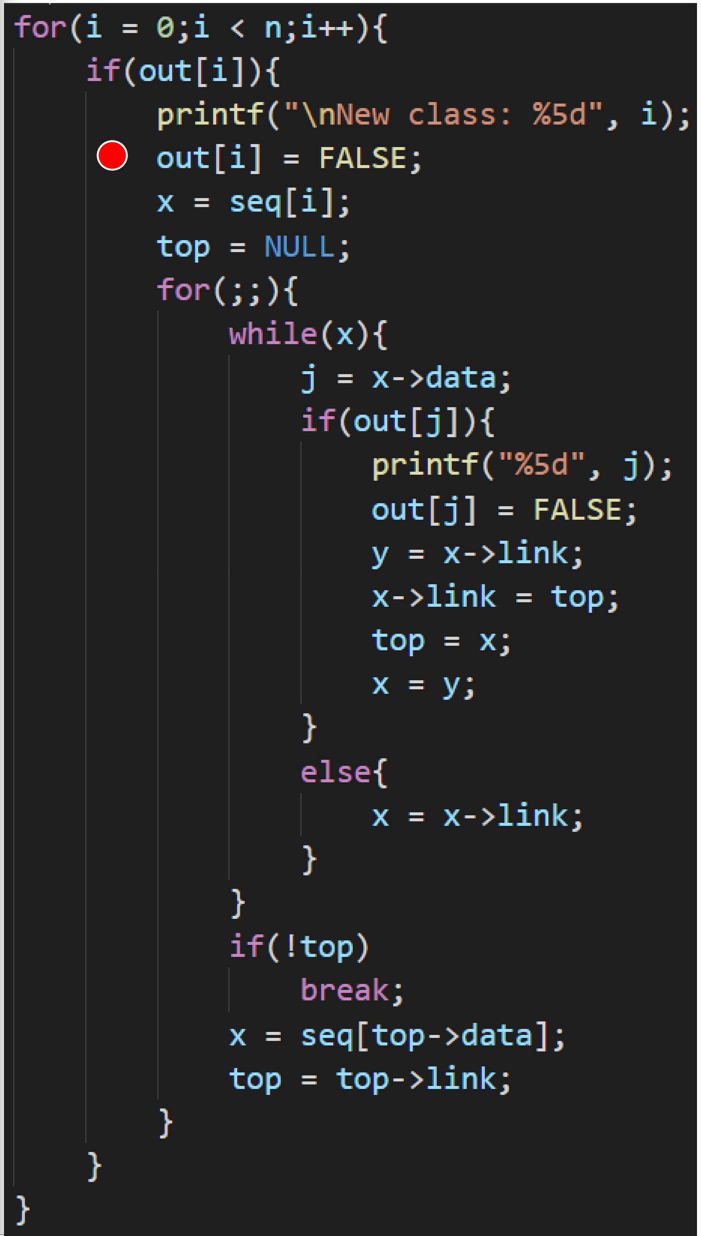
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	1	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

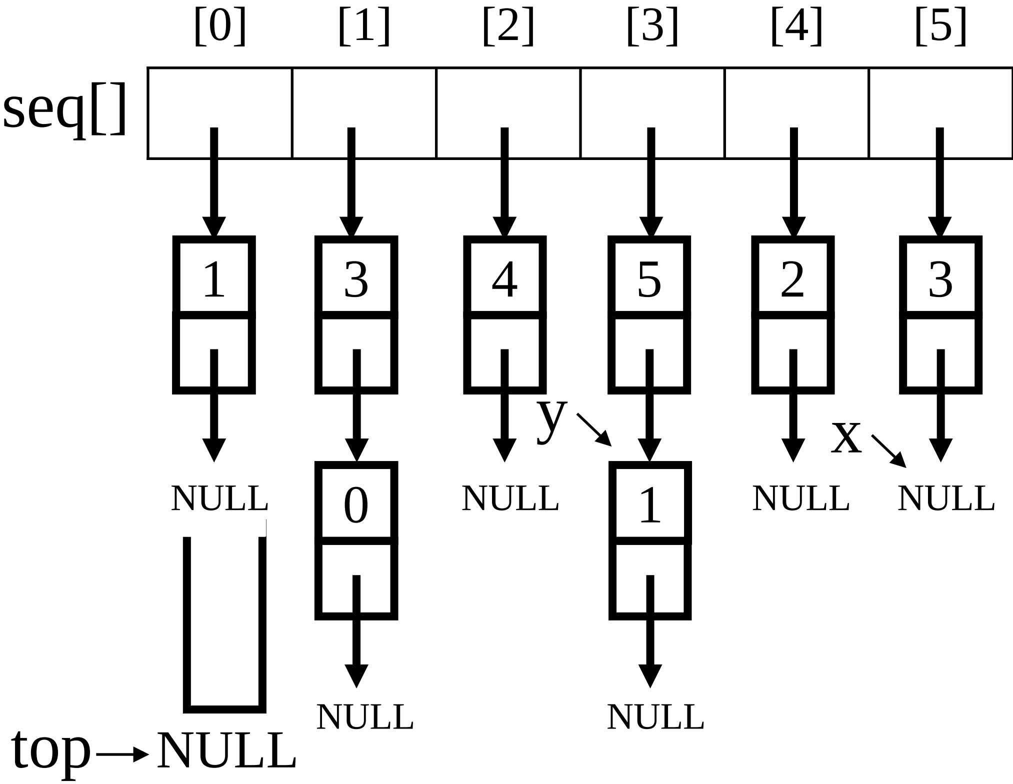
```
New class:    0    1    3    5
New class:    2
```



```c
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

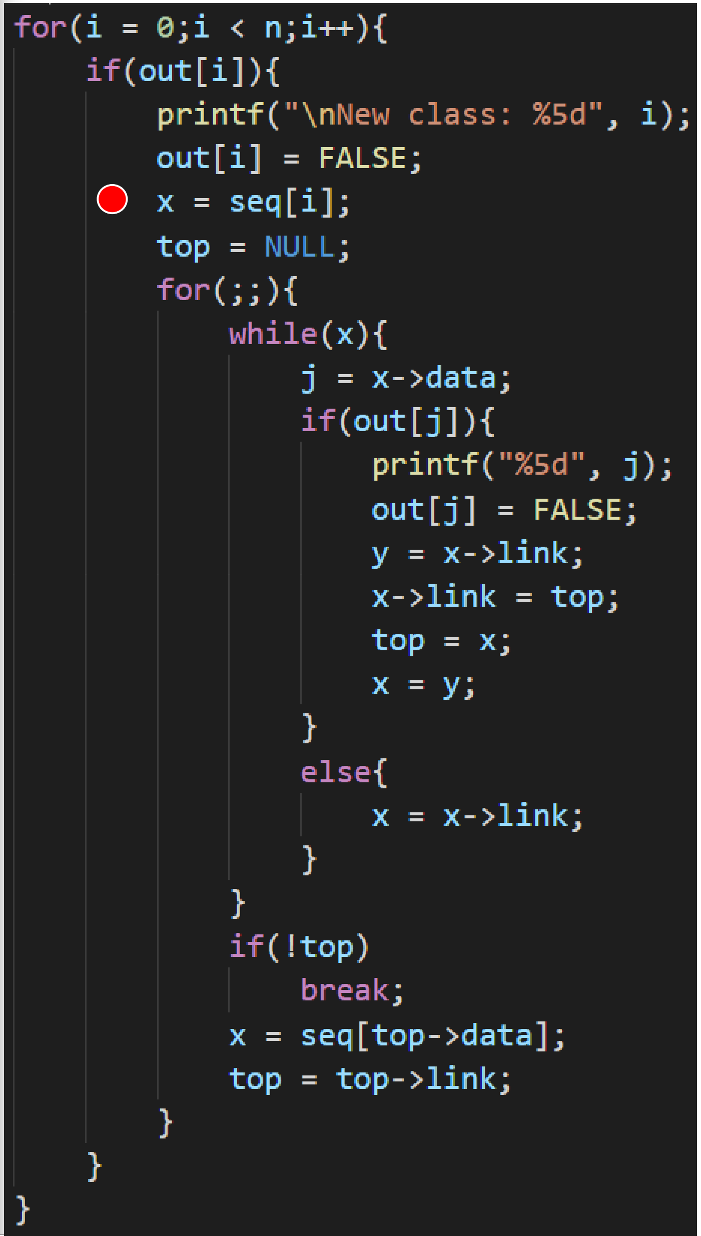
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

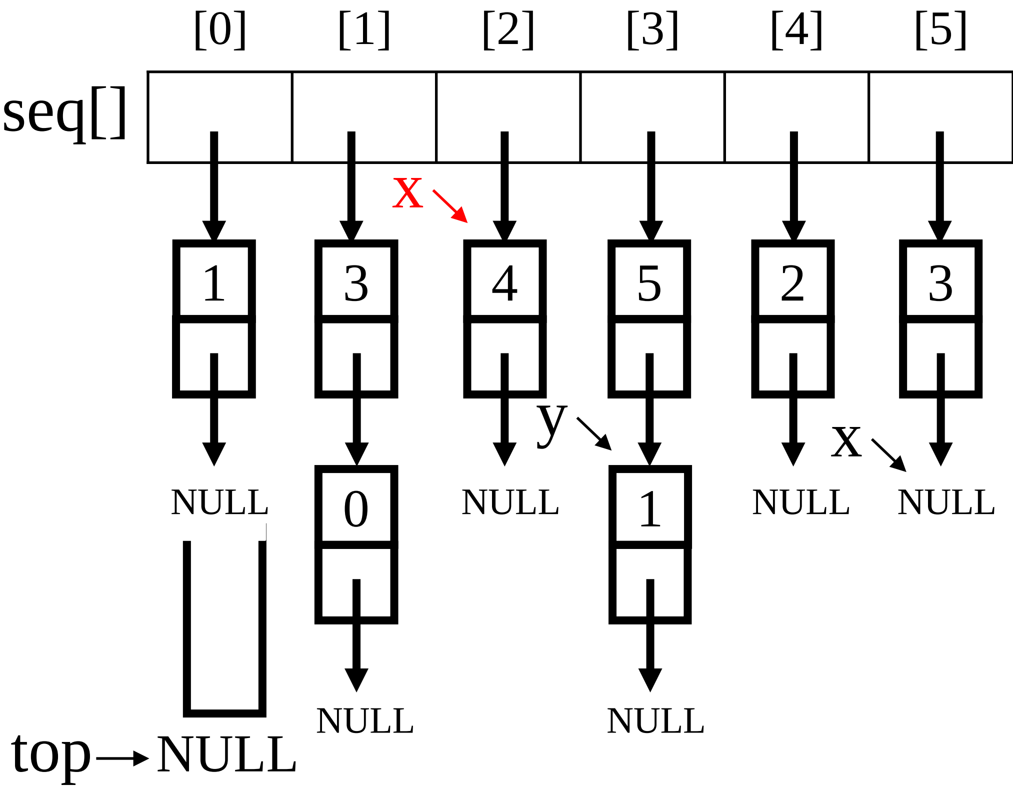
New class: 0 1 3 5
New class: 2



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

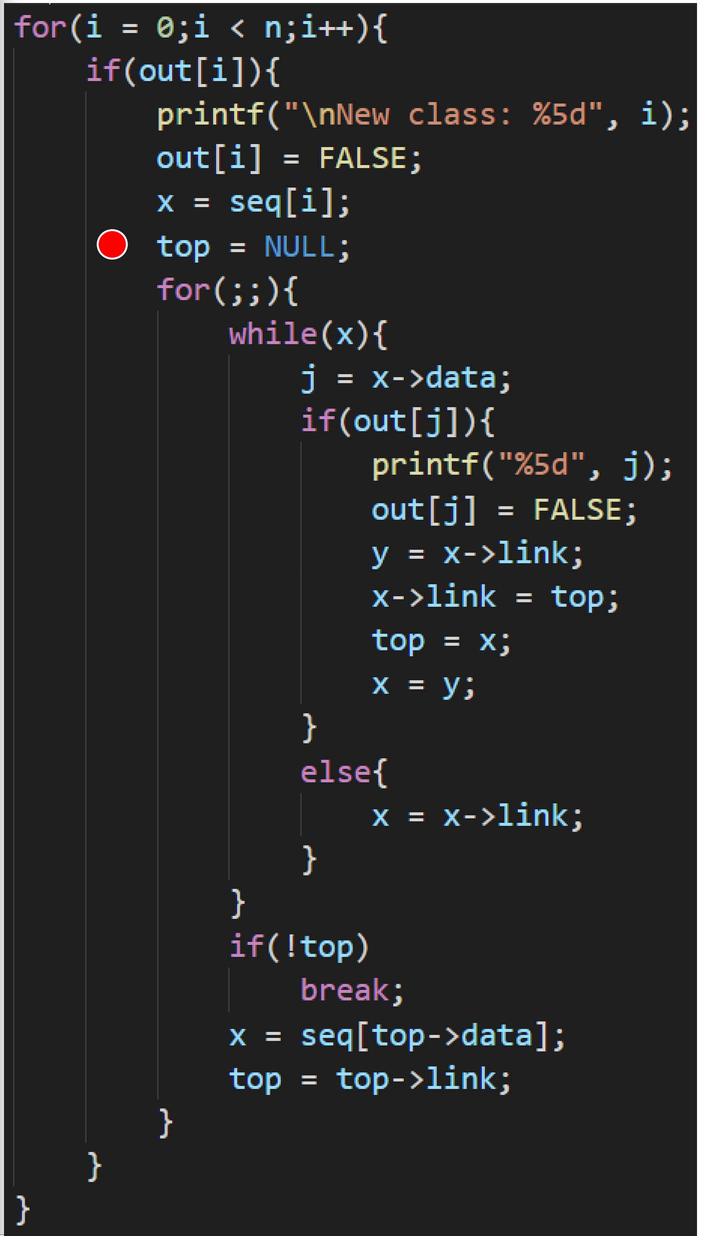
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

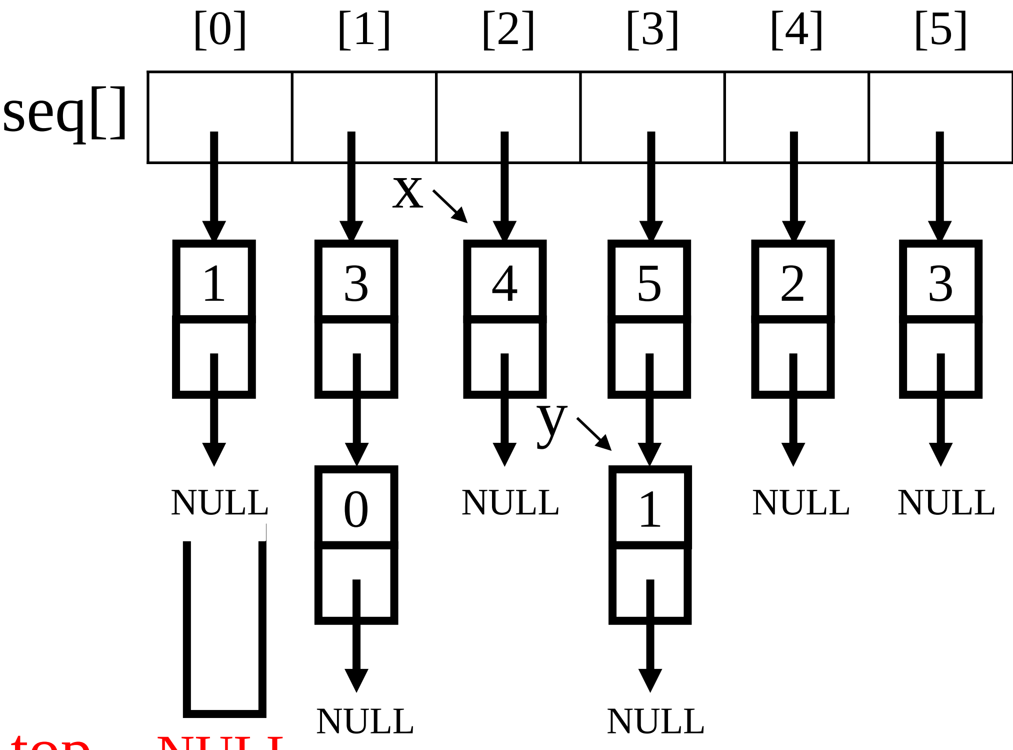
```
New class:    0    1    3    5
New class:    2
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	1	0

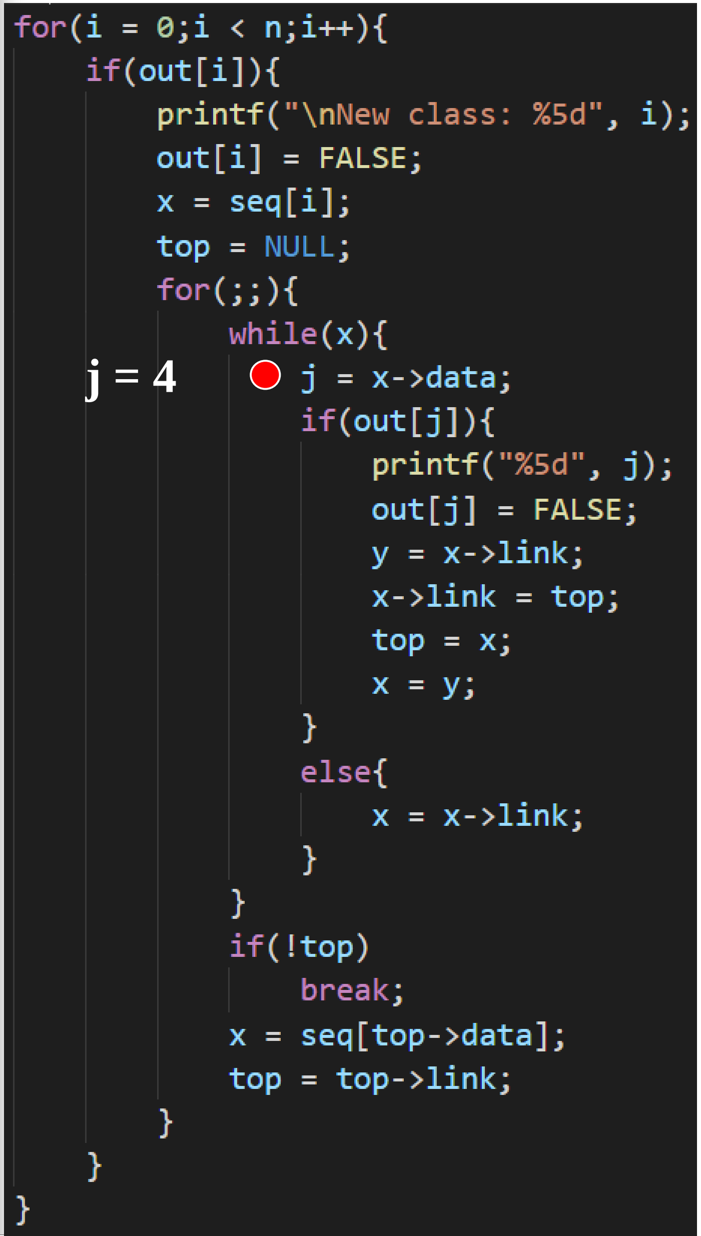


top→NULL

Equivalence relations

Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2
```

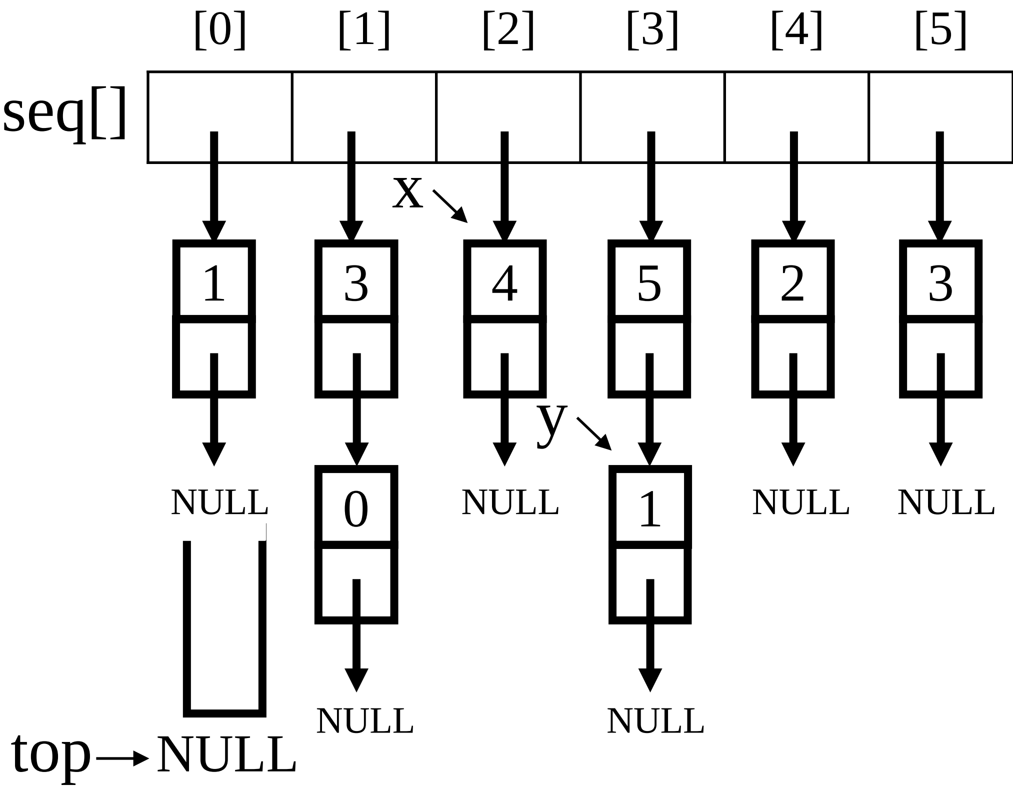


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

j = 4

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

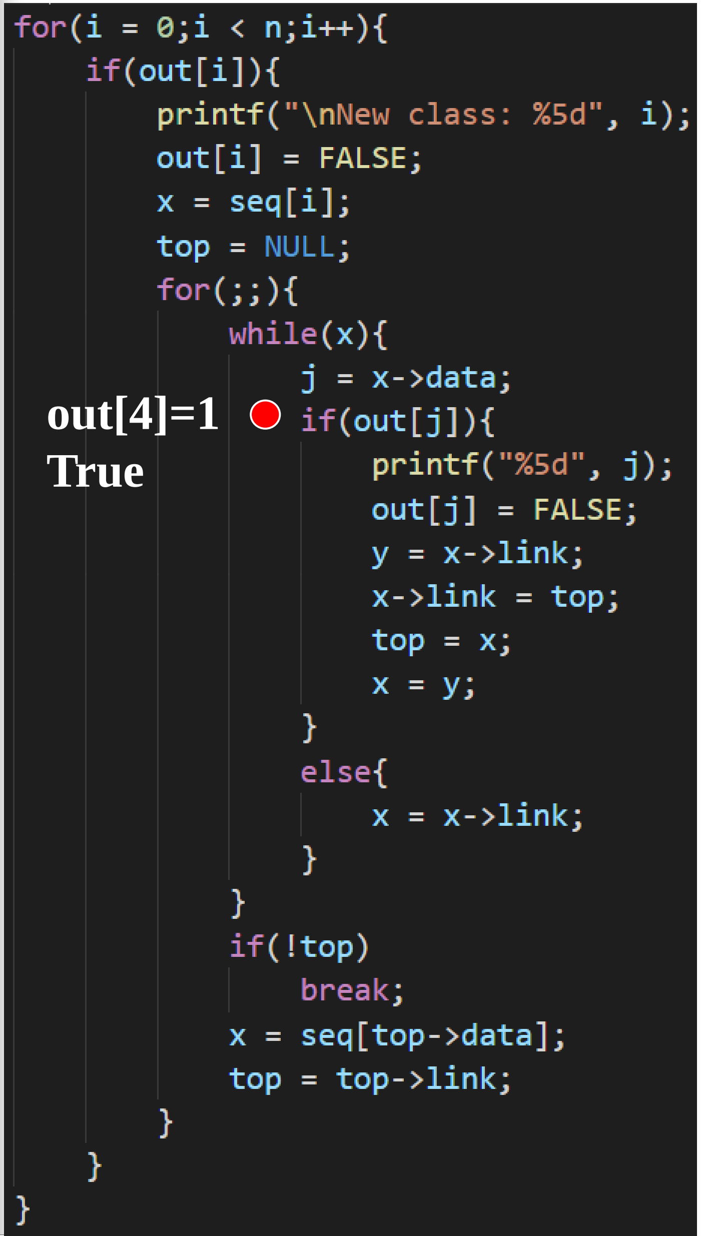
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	1	0



Equivalence relations

Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2
```



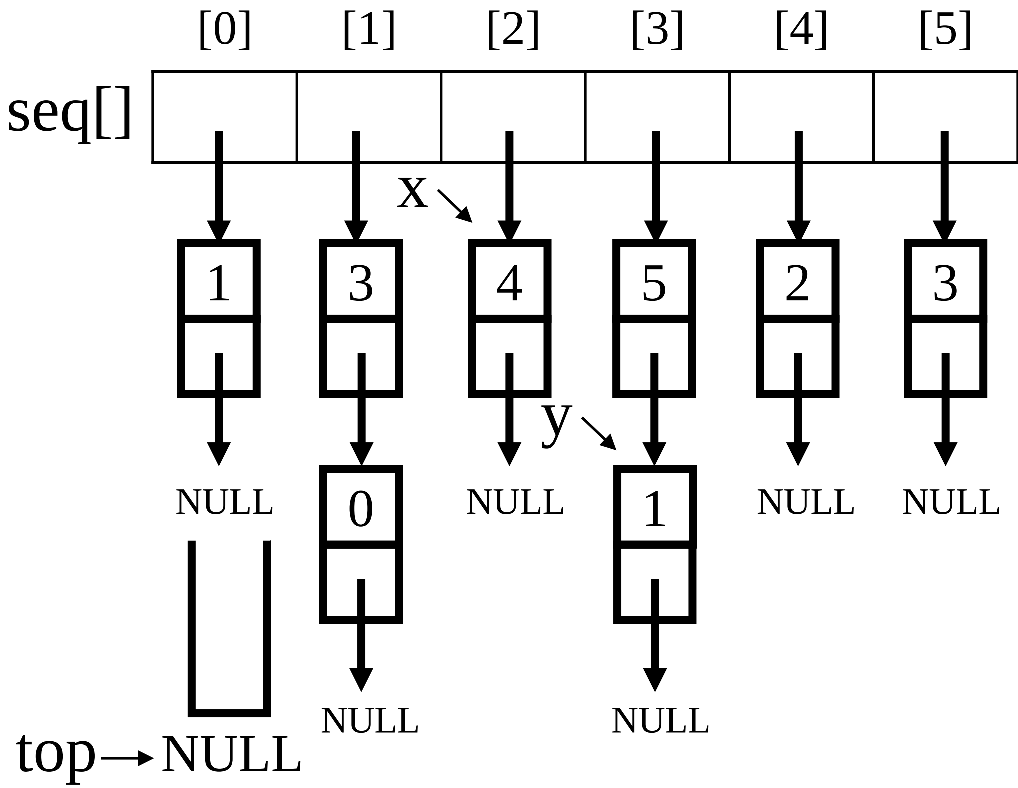
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[4]=1

True

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

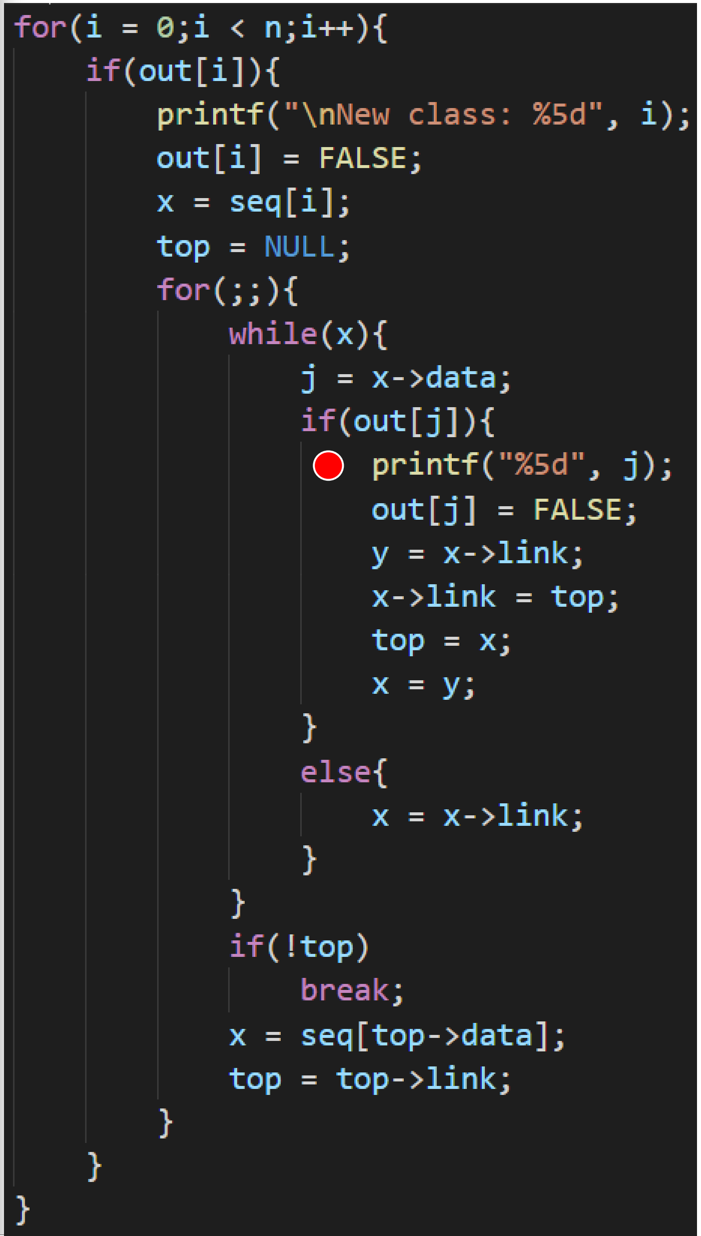
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	1	0



Equivalence relations

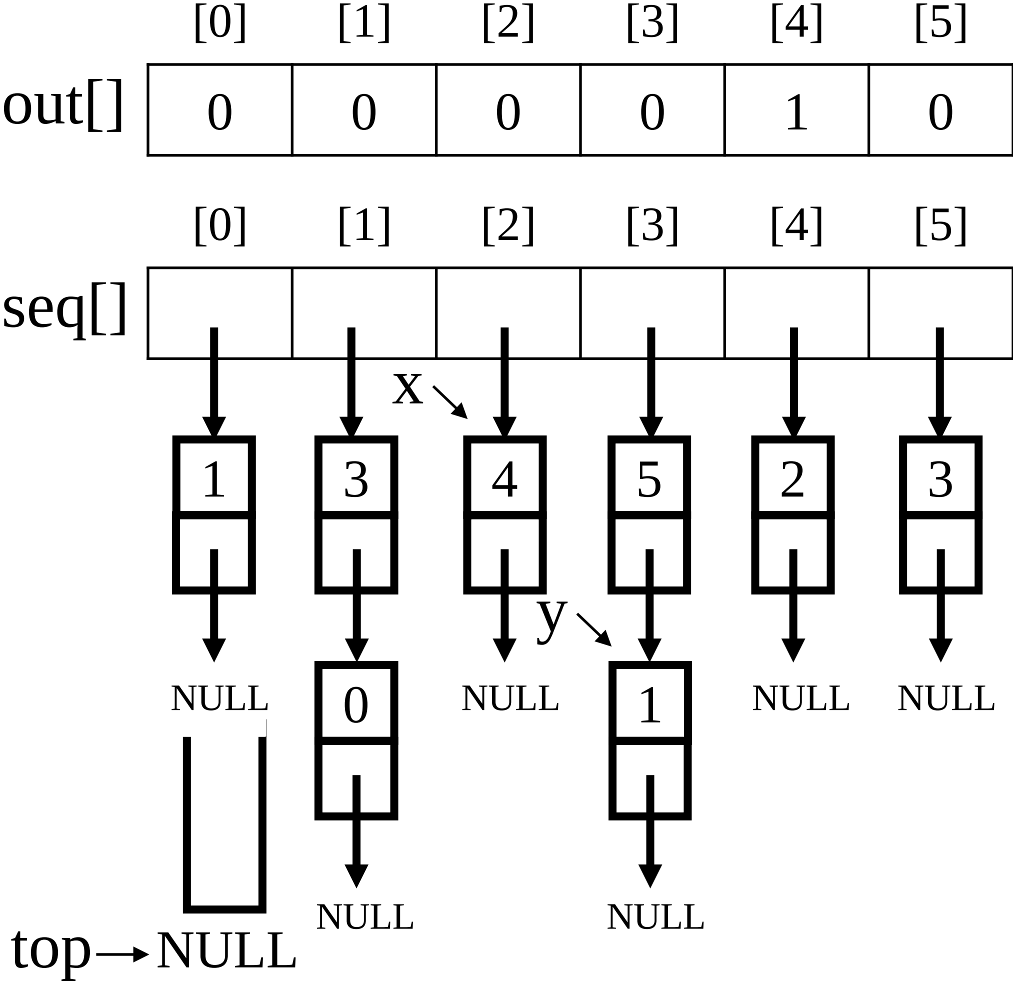
Phase 2: output the equivalence classes

New class: 0 1 3 5
New class: 2 4



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

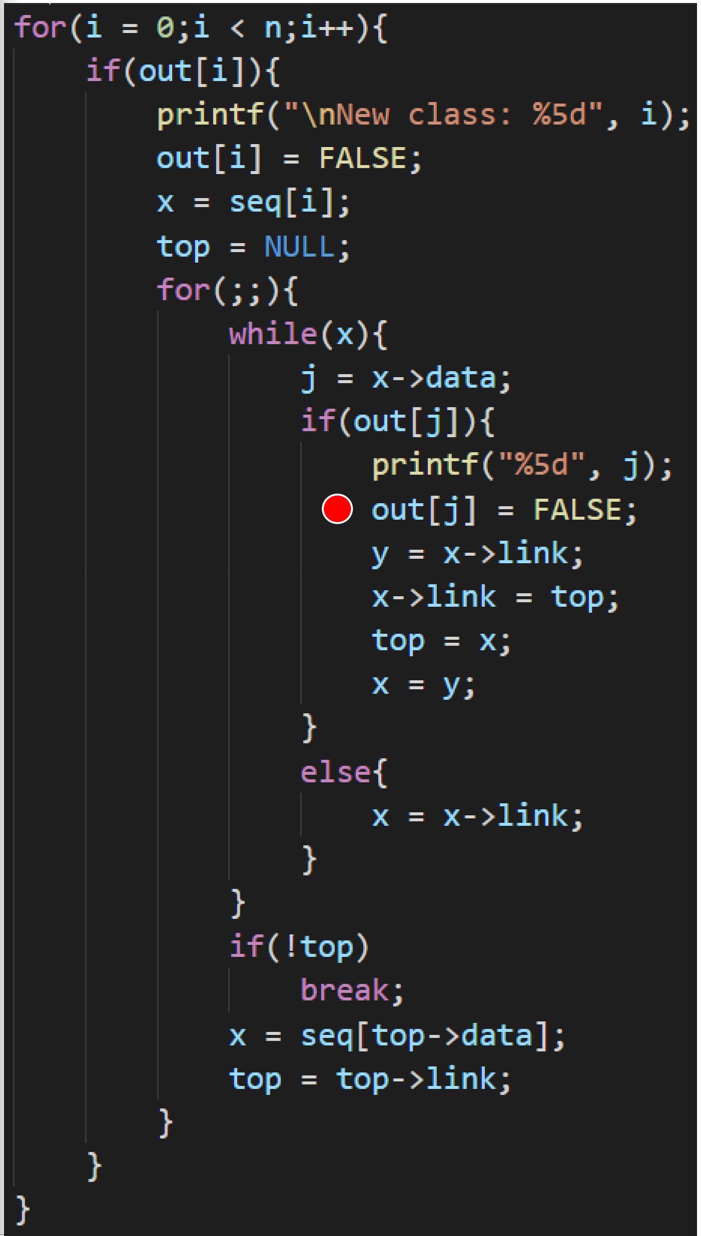
i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4



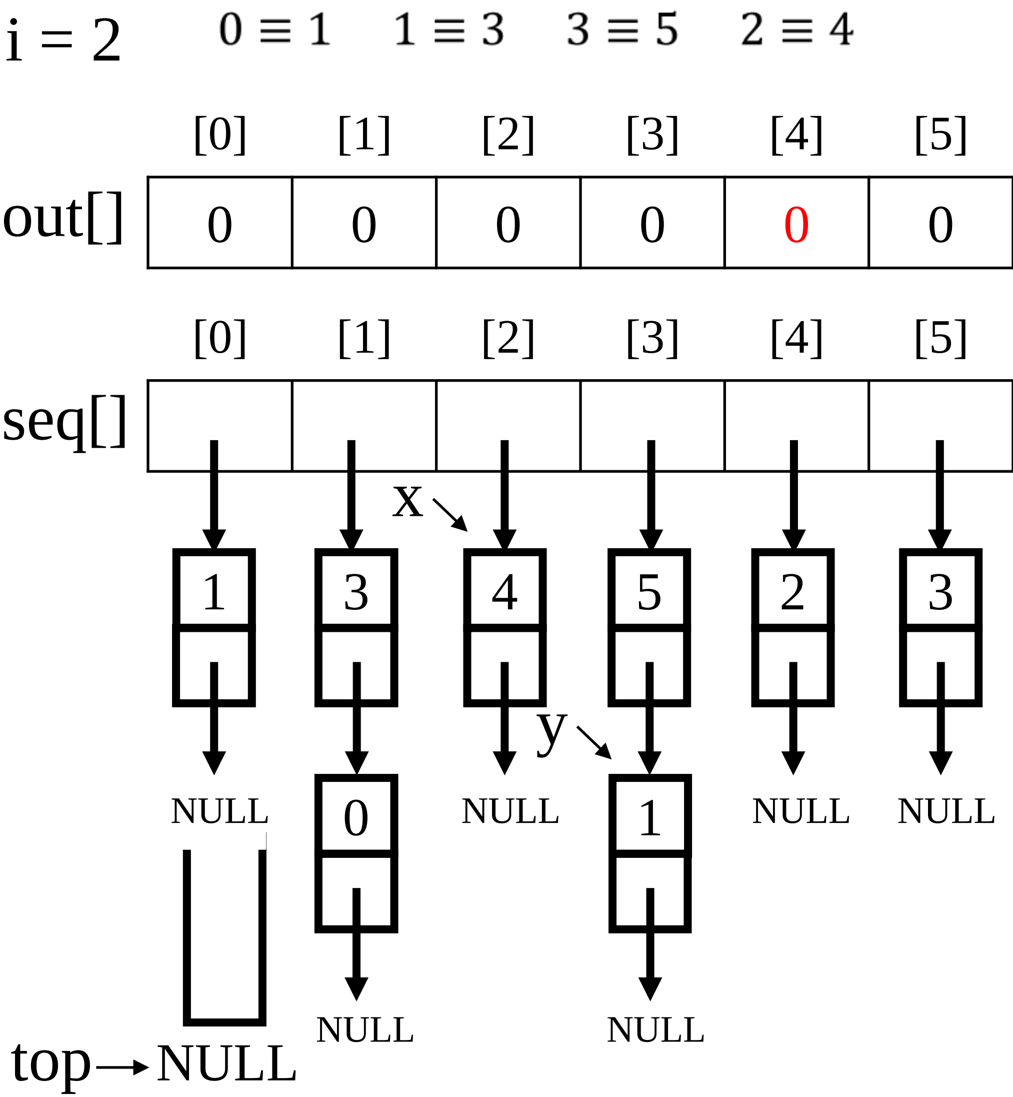
Equivalence relations

Phase 2: output the equivalence classes

```
New class:   0   1   3   5
New class:   2   4
```



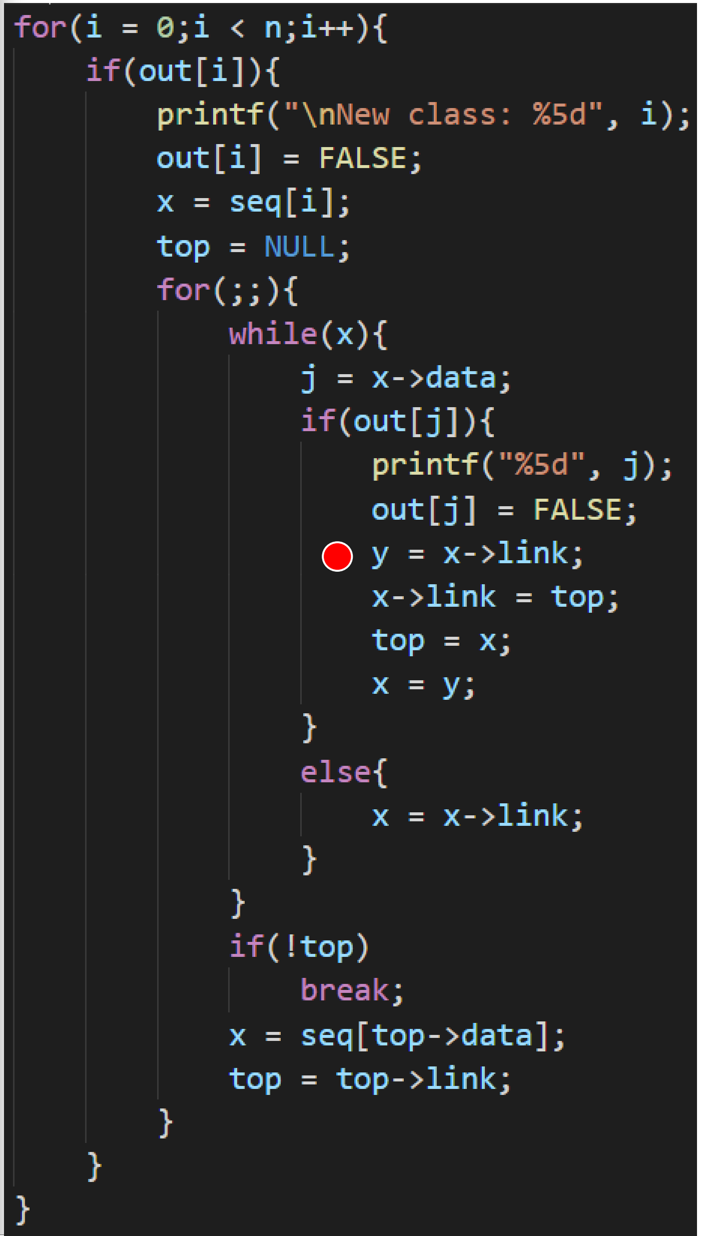
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                  ● out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```



Equivalence relations

Phase 2: output the equivalence classes

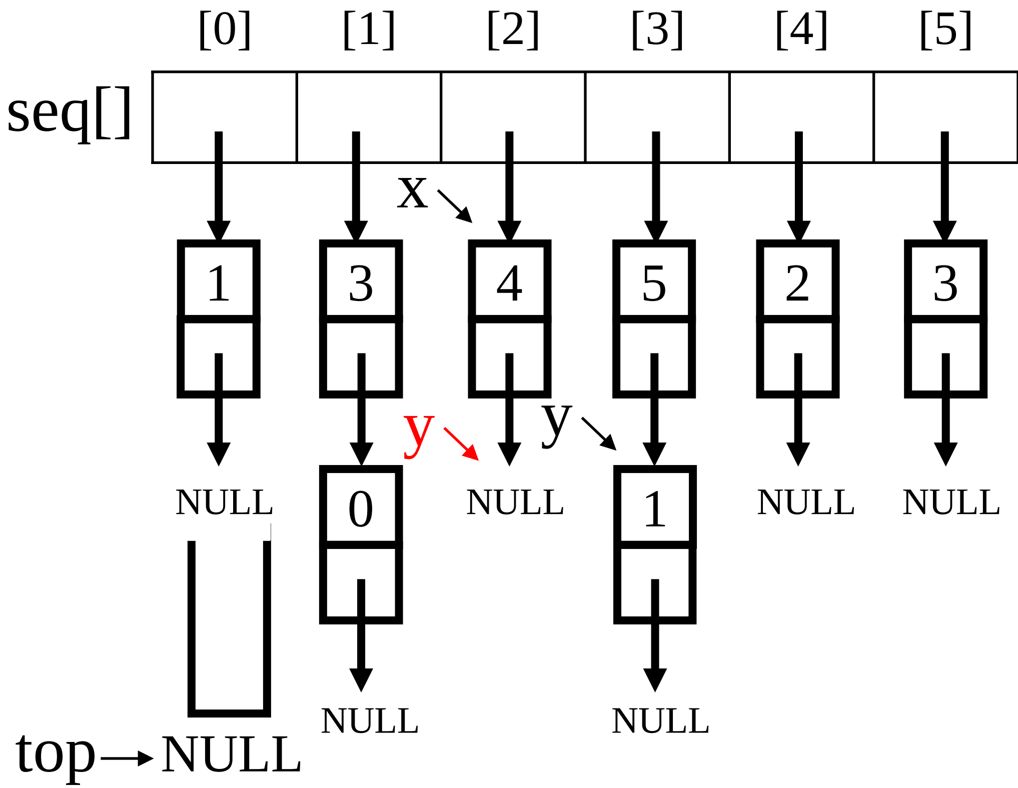
```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

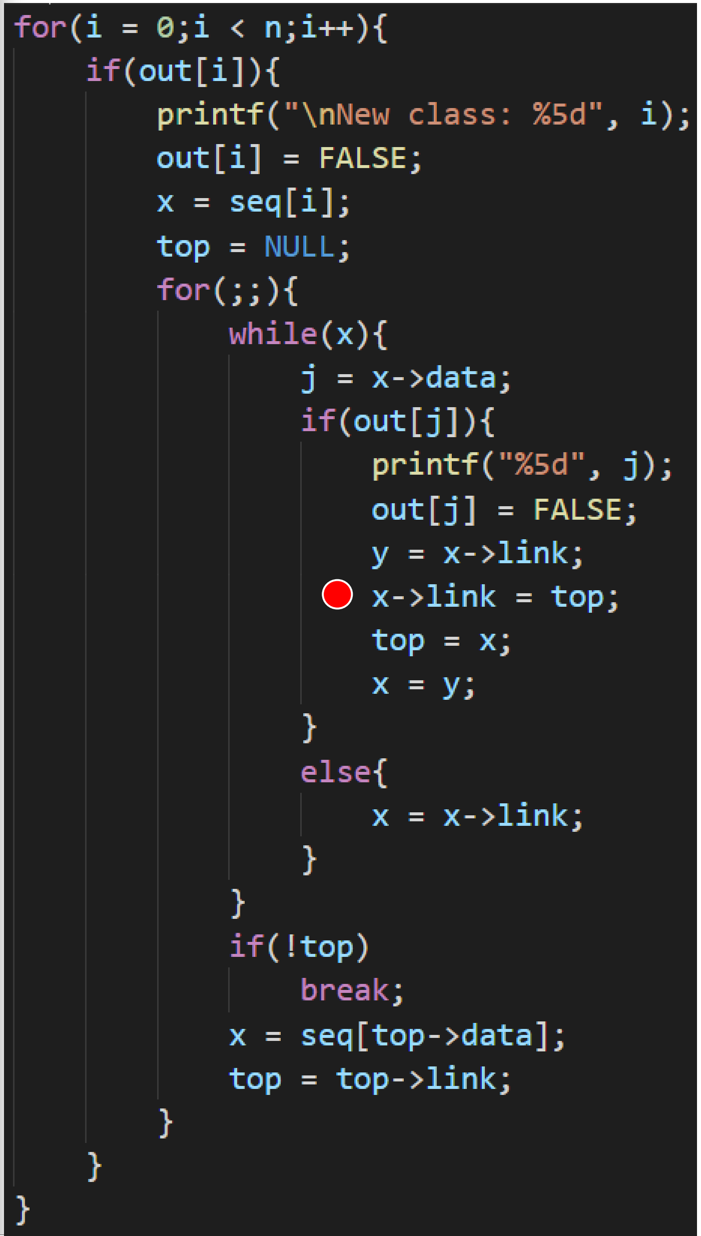
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

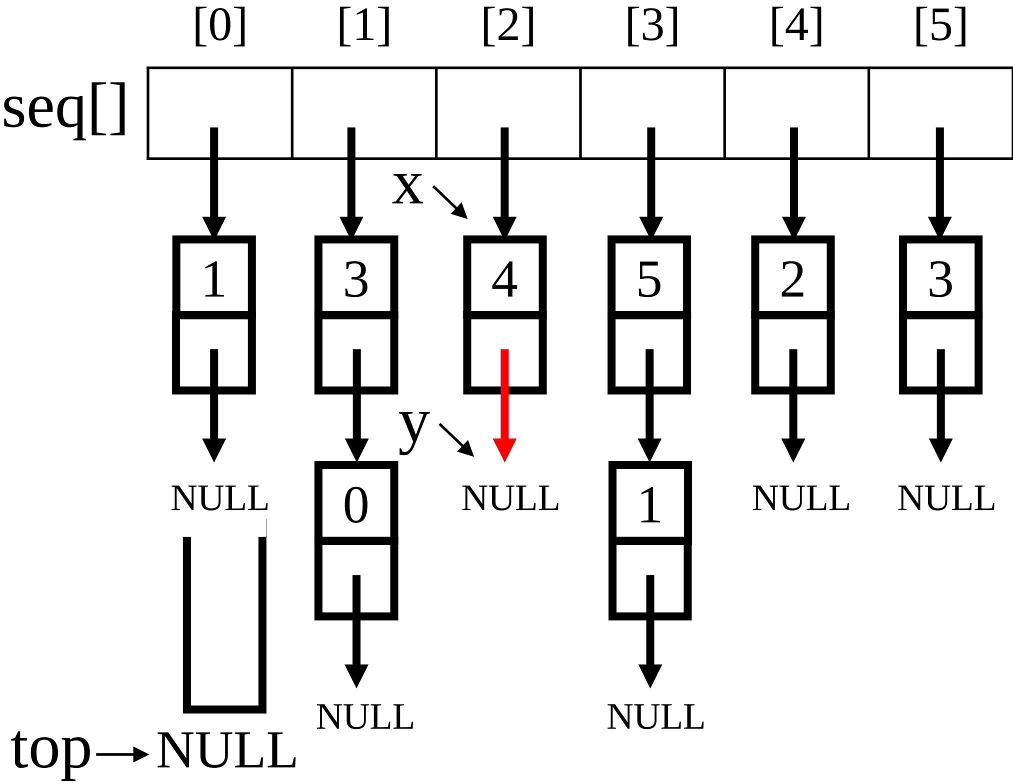
New class: 0 1 3 5
New class: 2 4



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

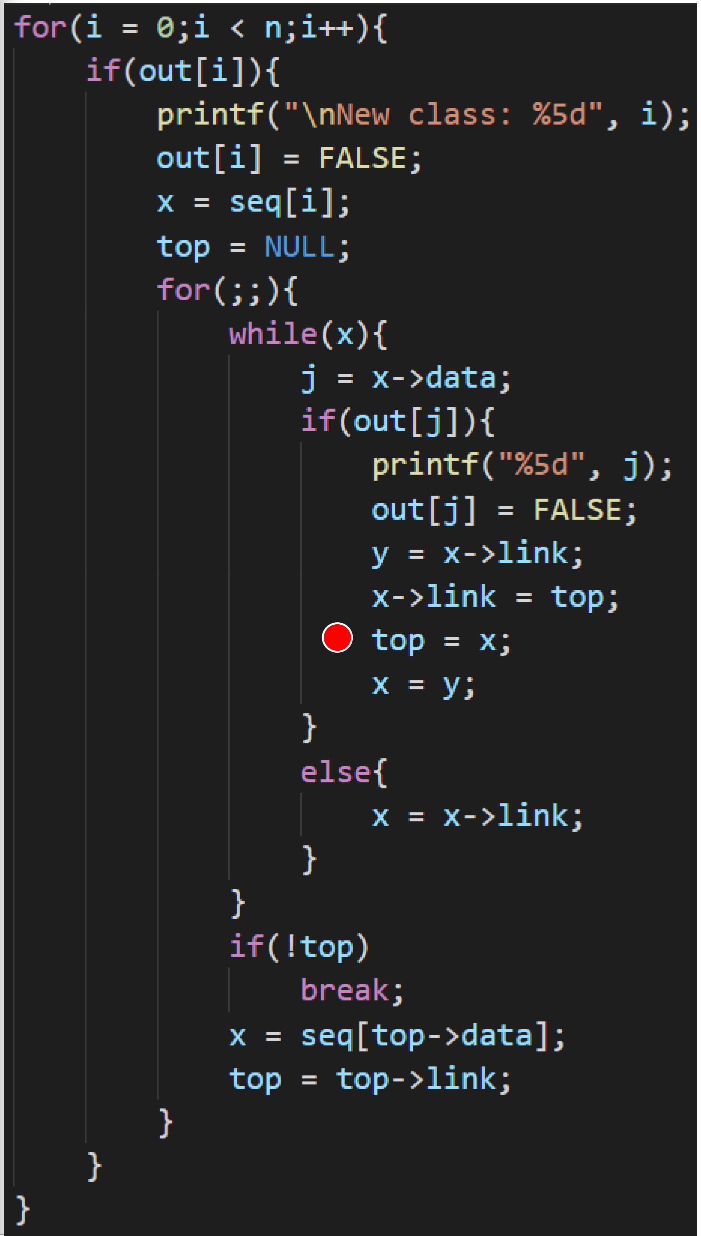
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

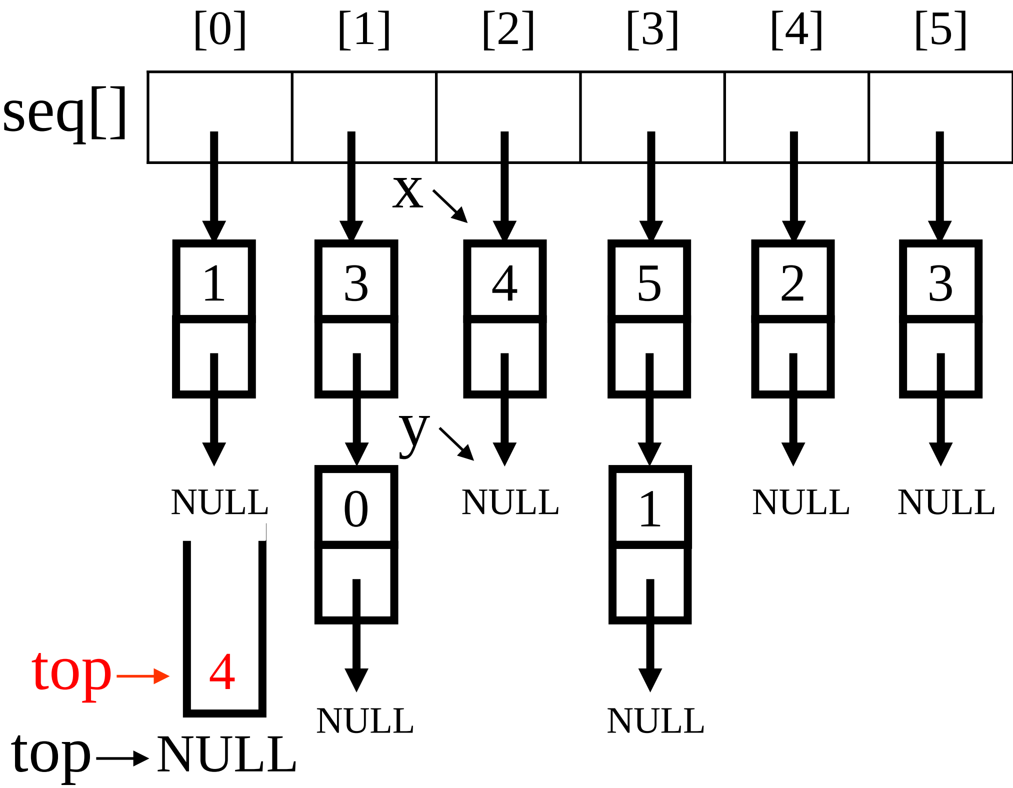
```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

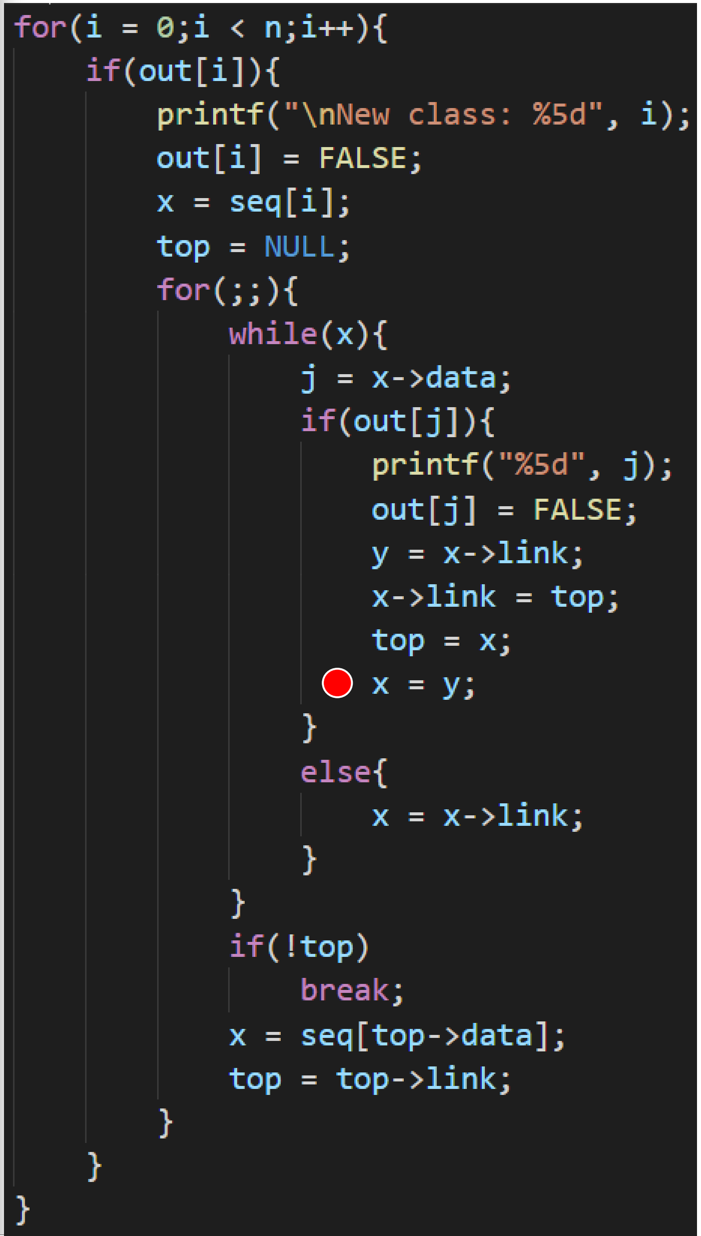
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



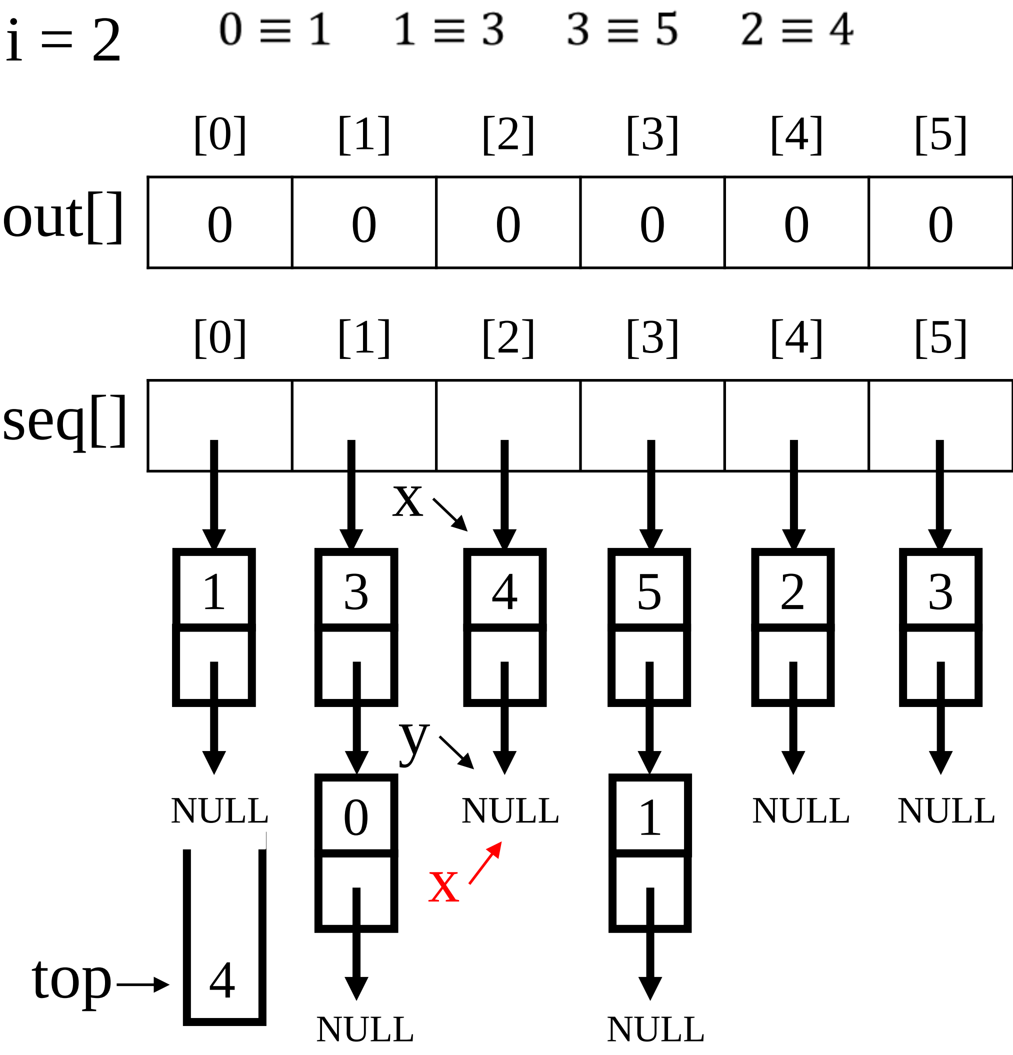
Equivalence relations

Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2    4
```



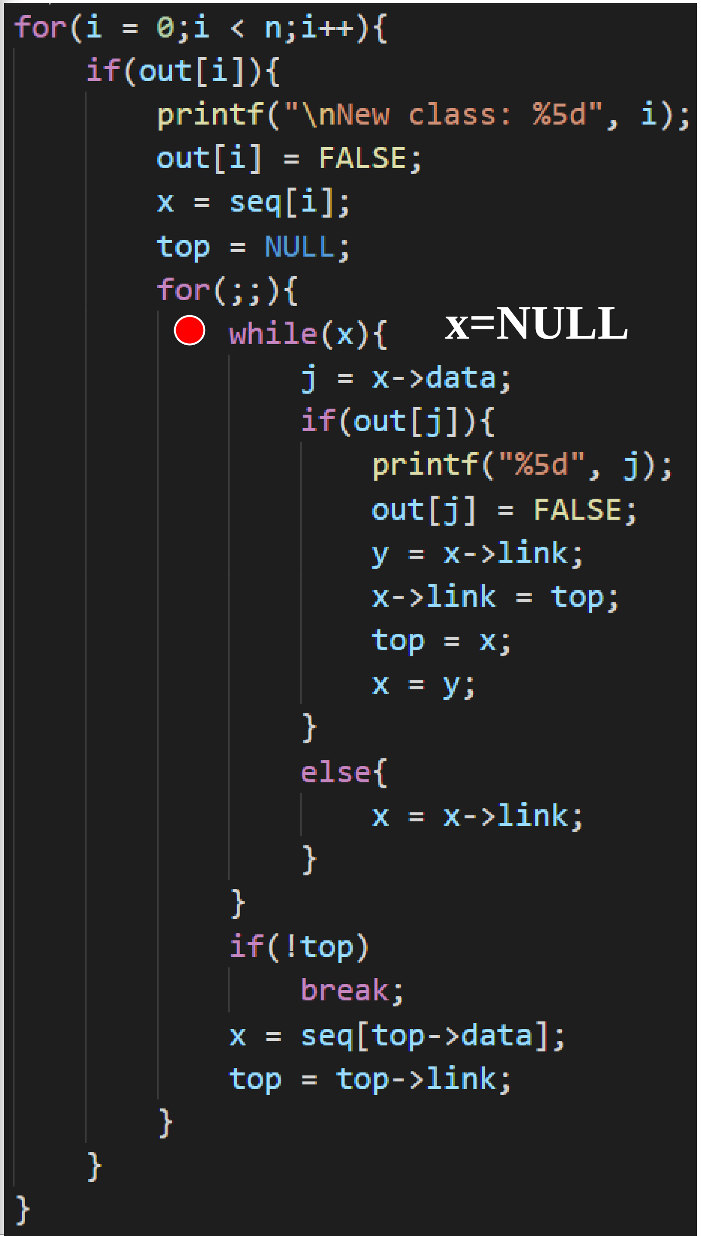
```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5
New class: 2 4

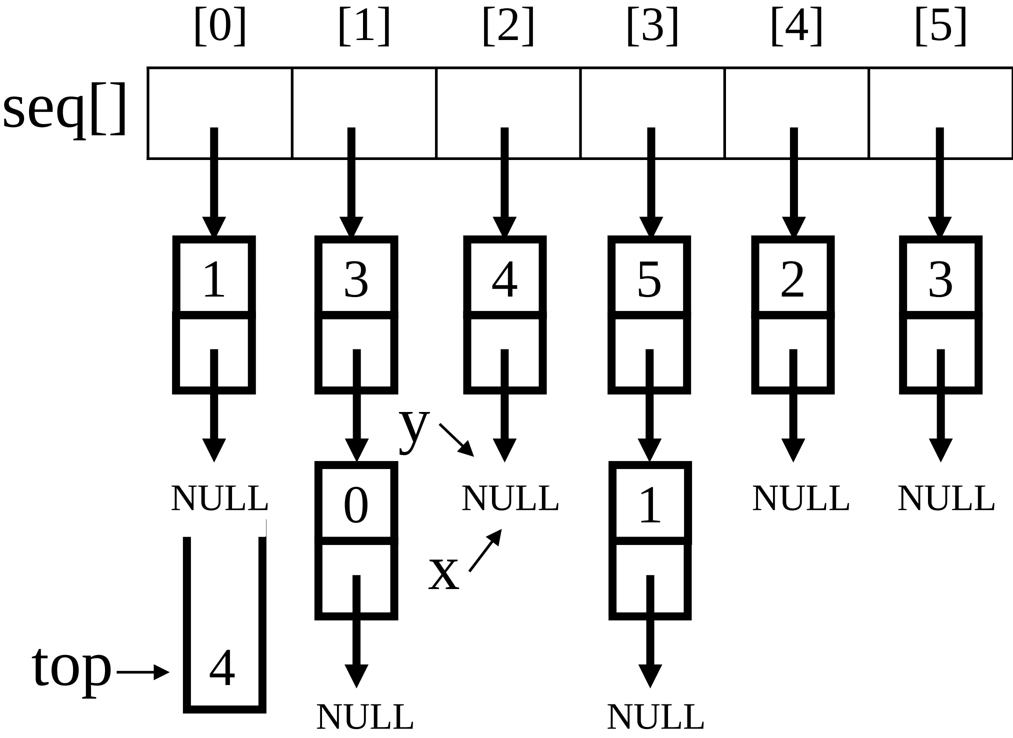


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

x=NULL

i = 2 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

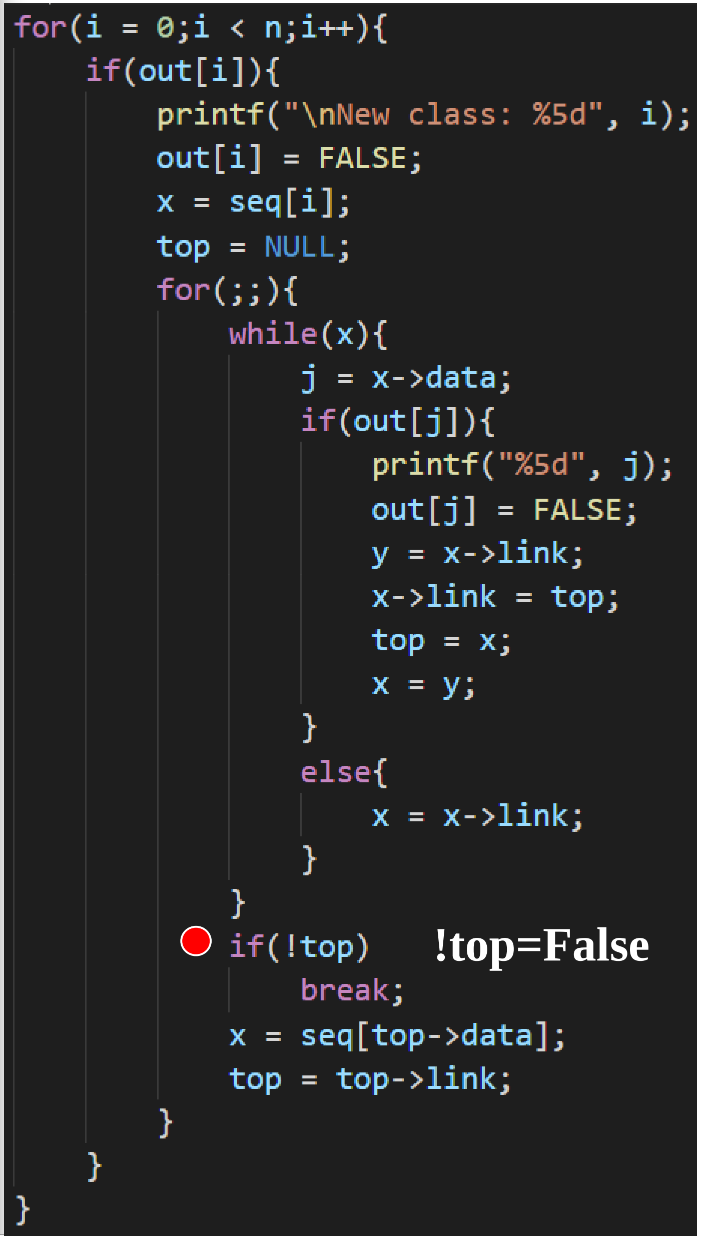
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

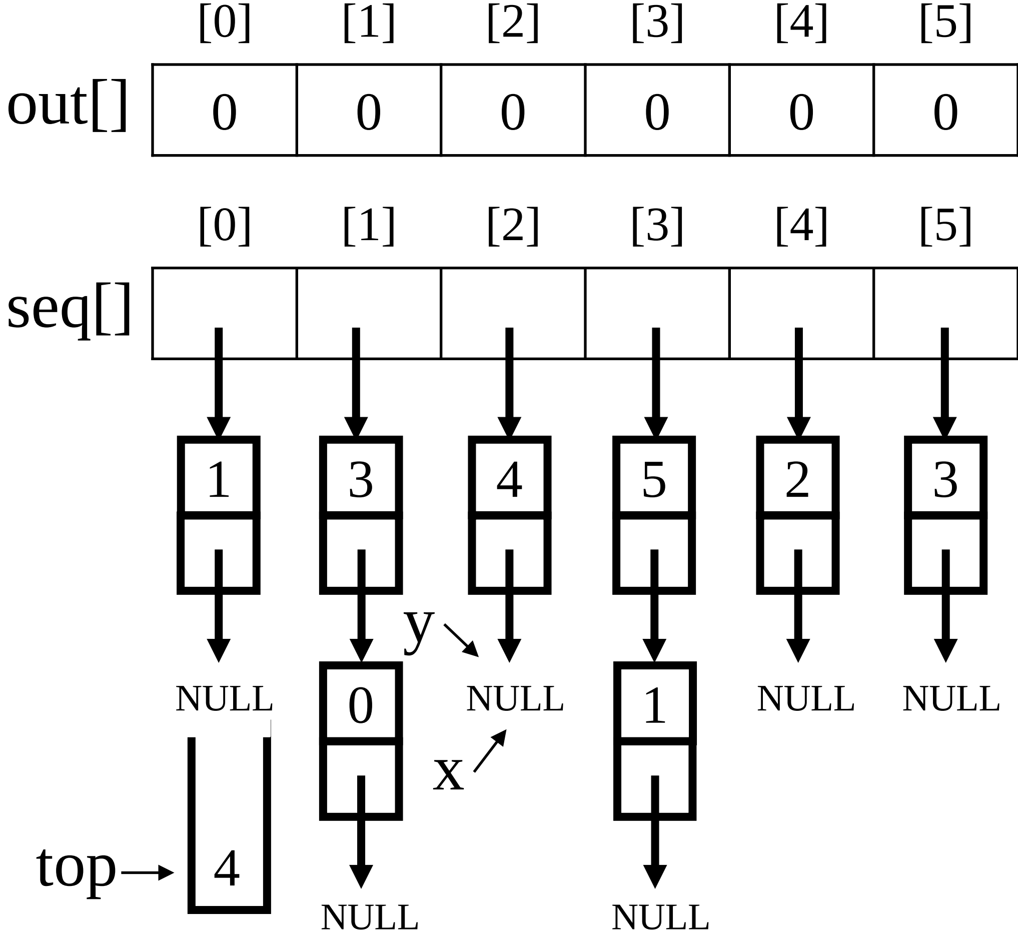
Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
●           if(!top)      !top=False
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

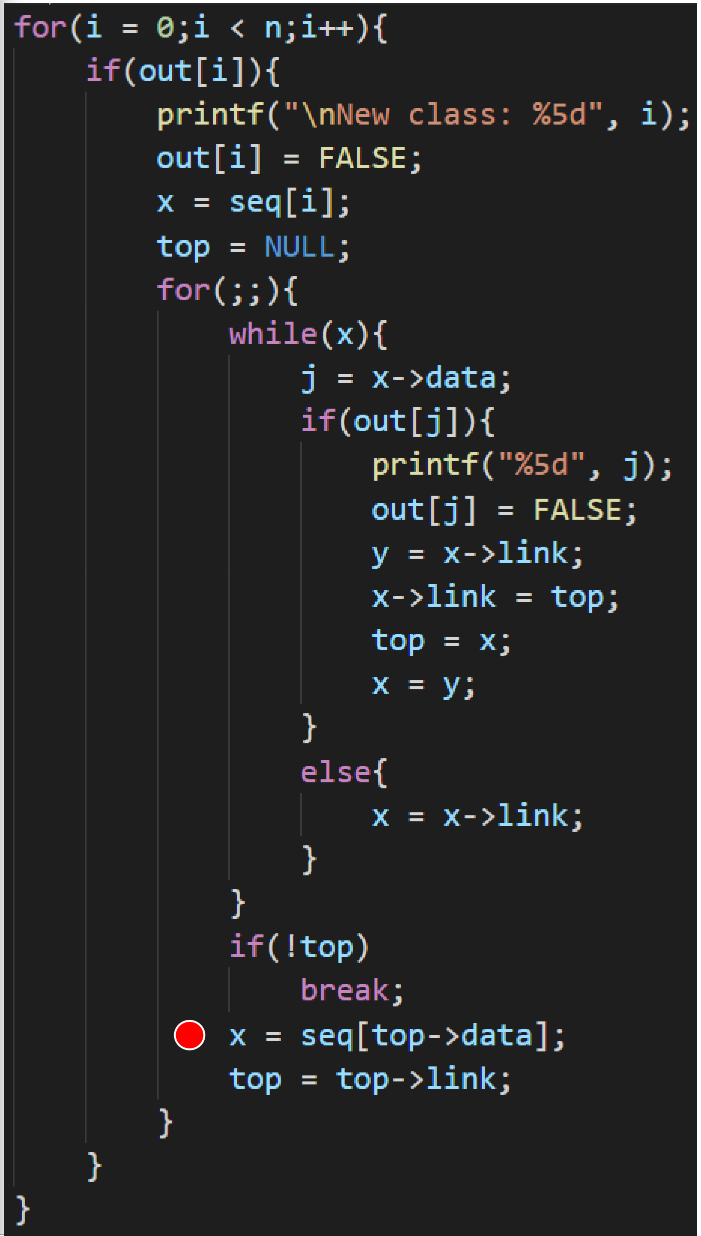
i = 2 0 ≡ 1 1 ≡ 3 3 ≡ 5 2 ≡ 4



Equivalence relations

Phase 2: output the equivalence classes

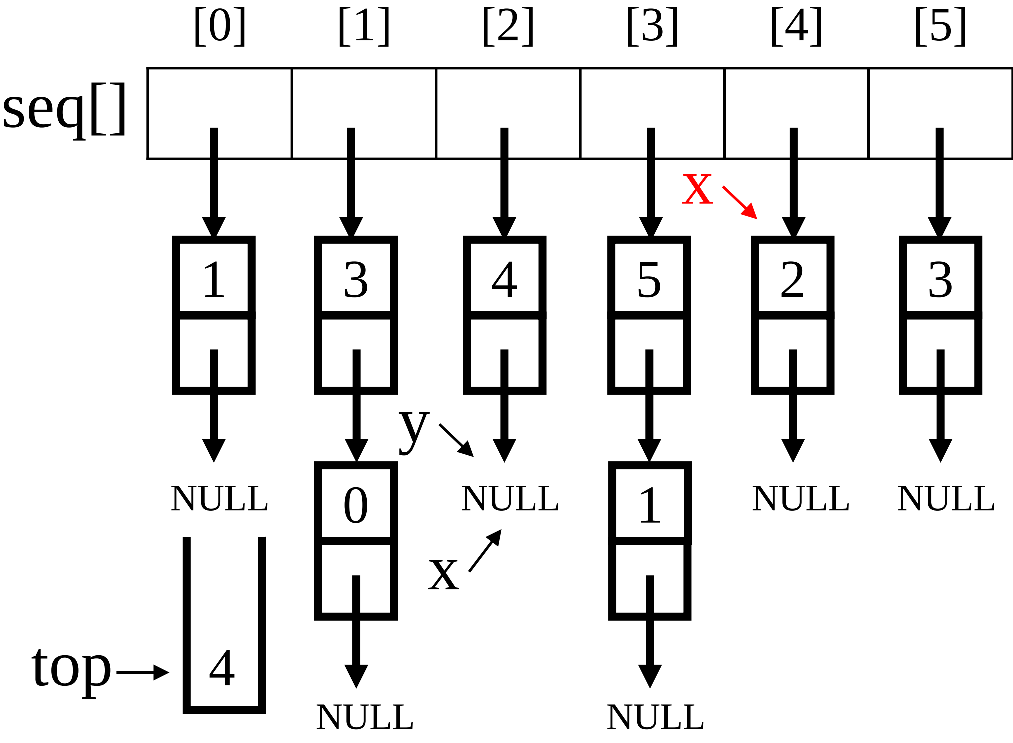
New class: 0 1 3 5
New class: 2 4



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

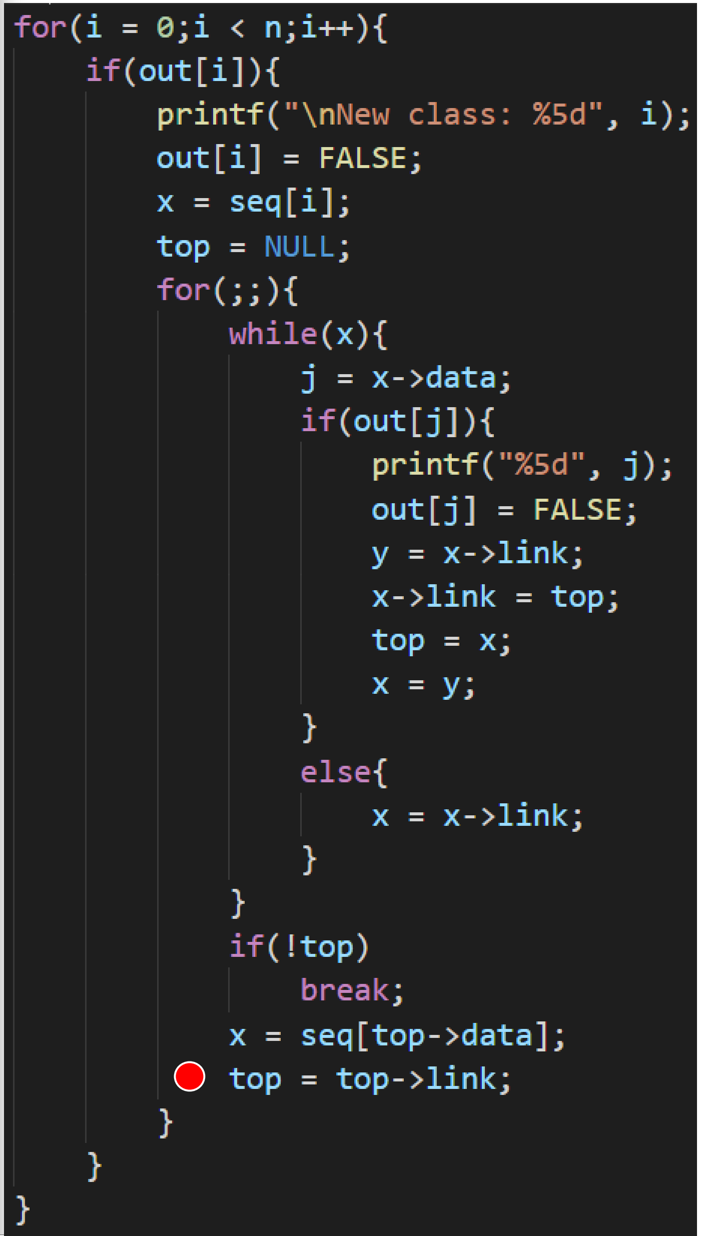
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

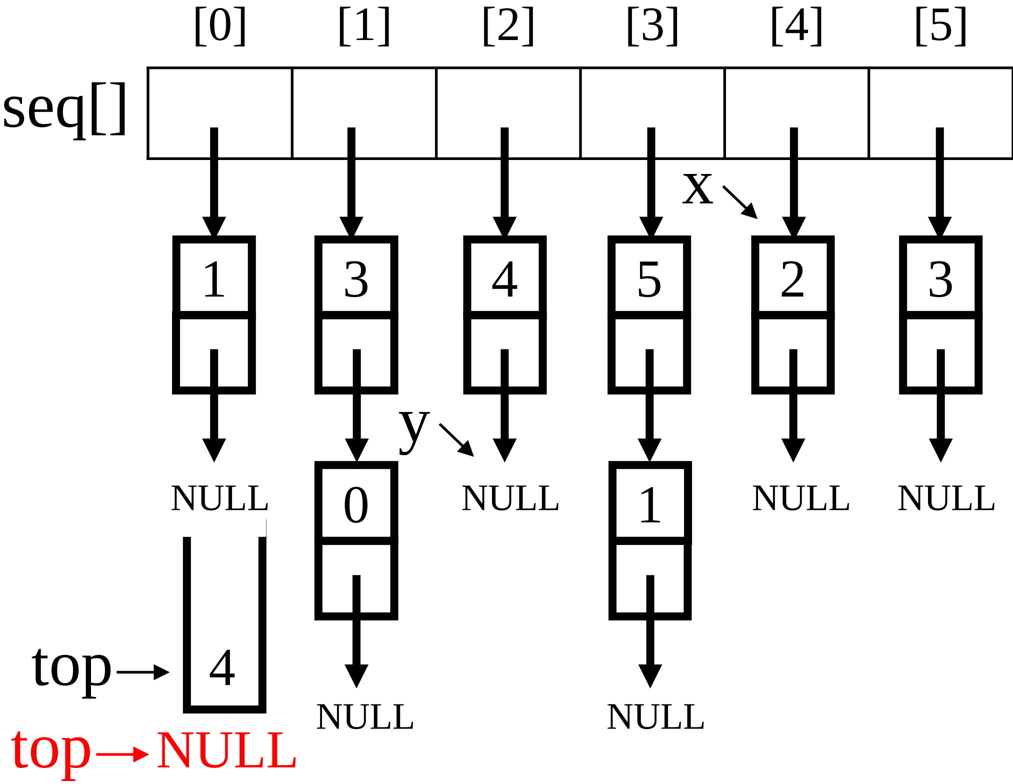
```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2 $\quad 0 \equiv 1 \quad 1 \equiv 3 \quad 3 \equiv 5 \quad 2 \equiv 4$

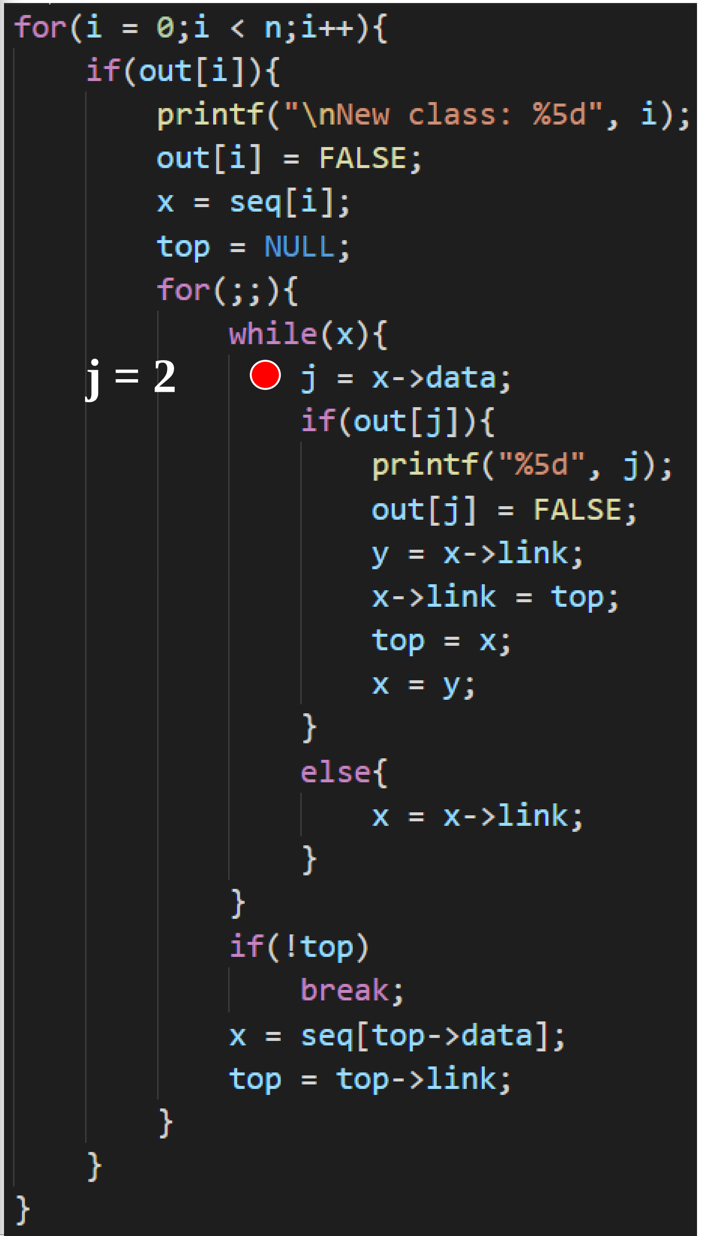
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

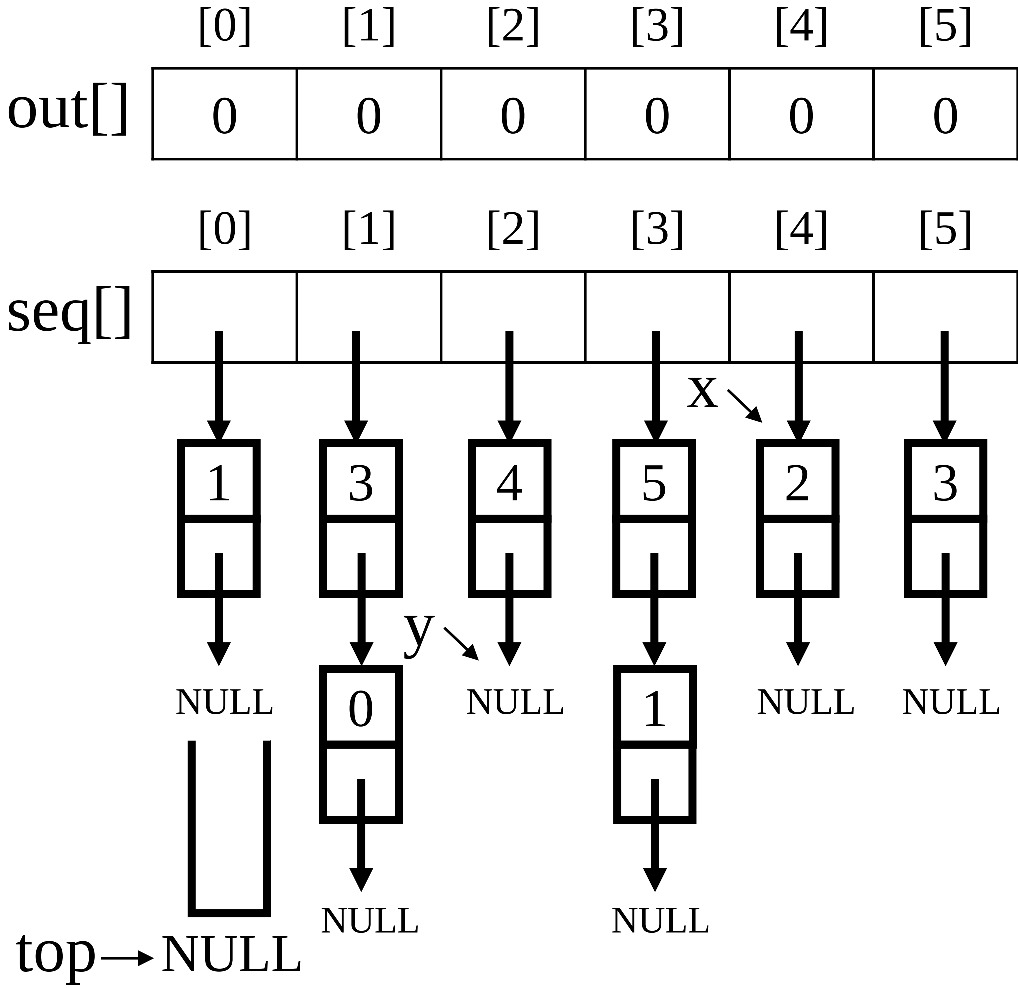
Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

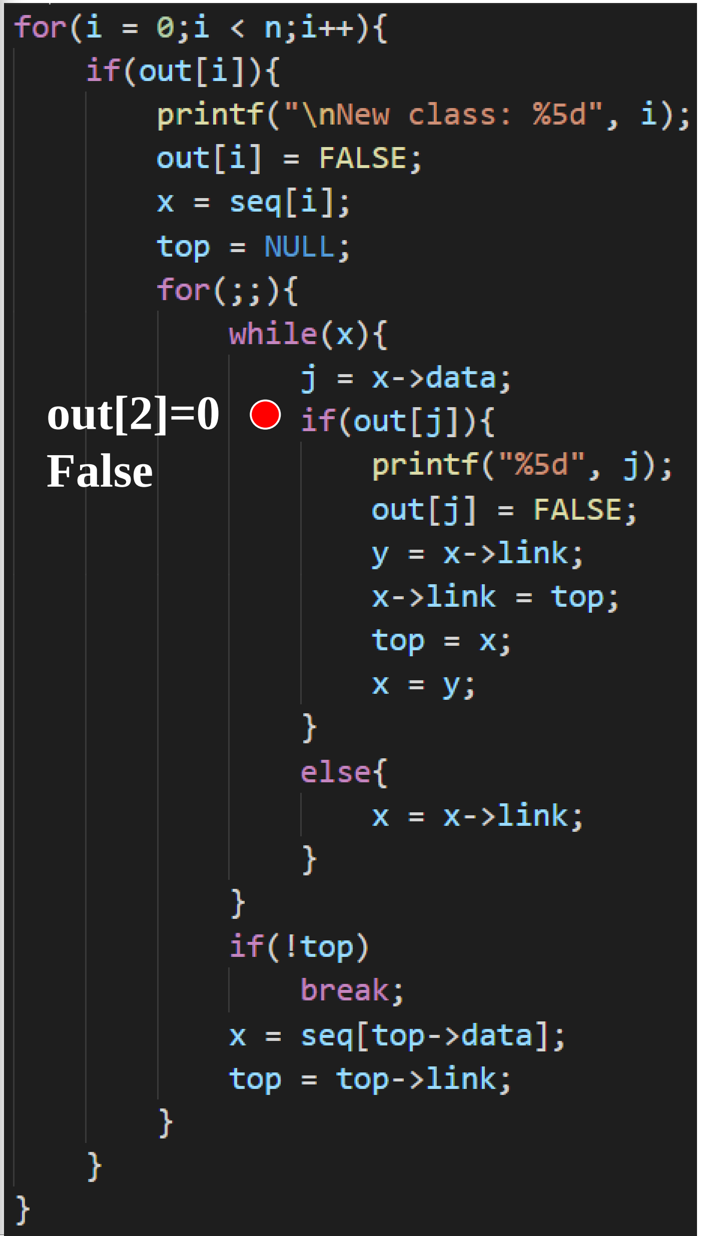
i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4



Equivalence relations

Phase 2: output the equivalence classes

New class: 0 1 3 5
New class: 2 4

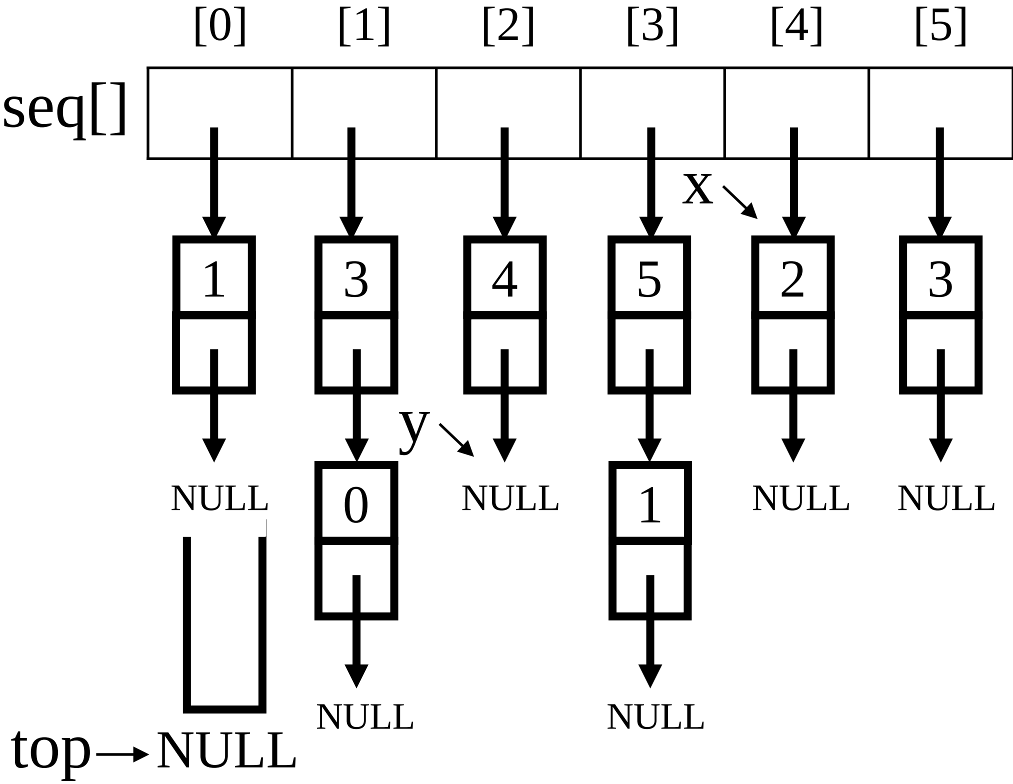


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[2]=0
False

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

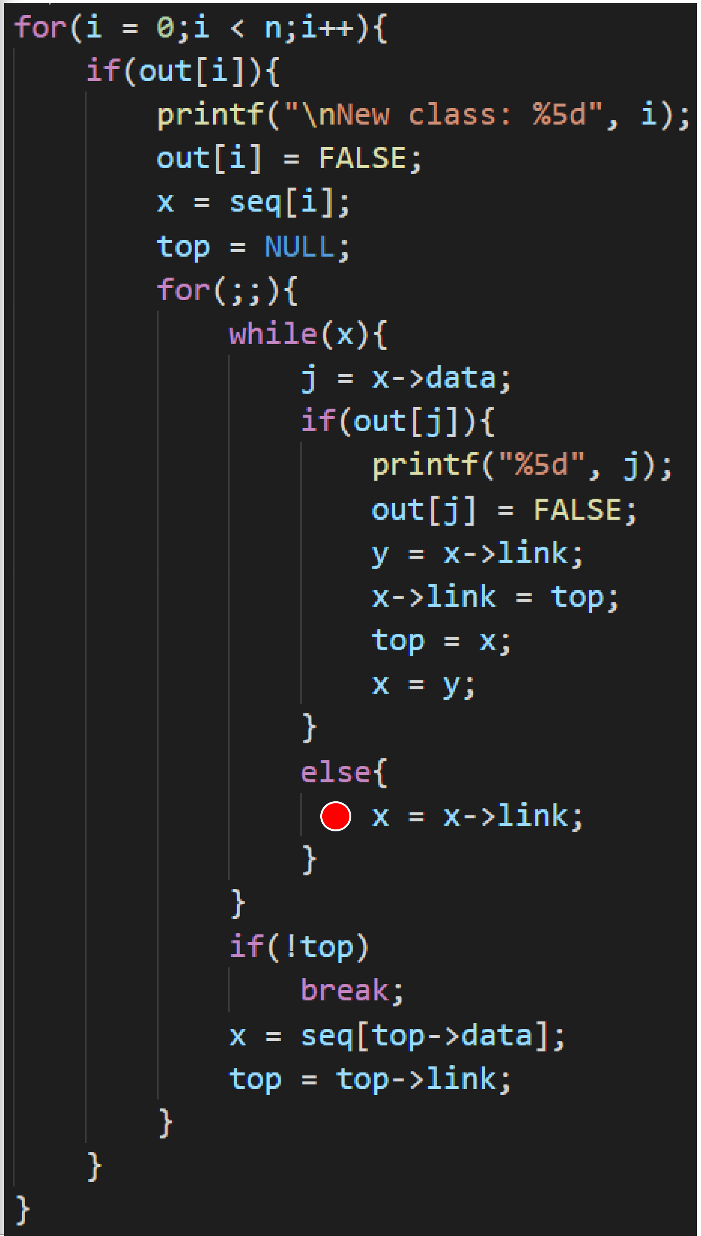
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

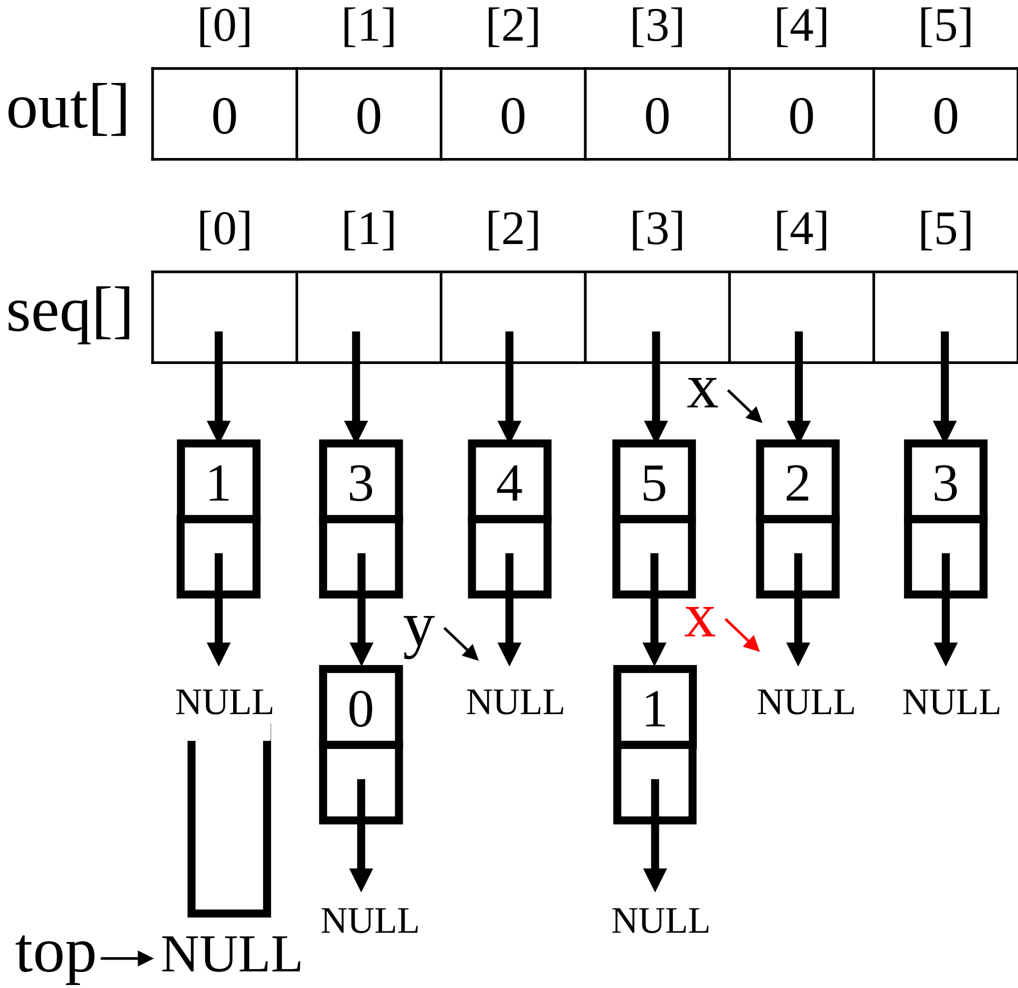
Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

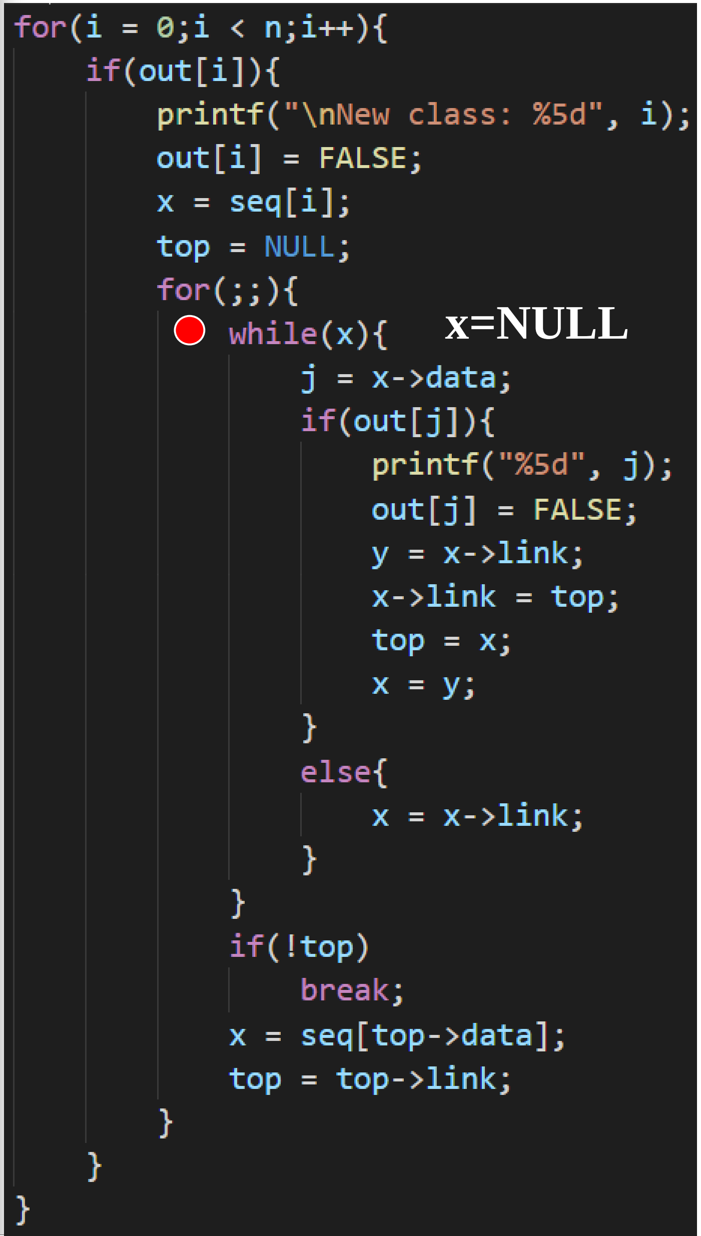
i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4



Equivalence relations

Phase 2: output the equivalence classes

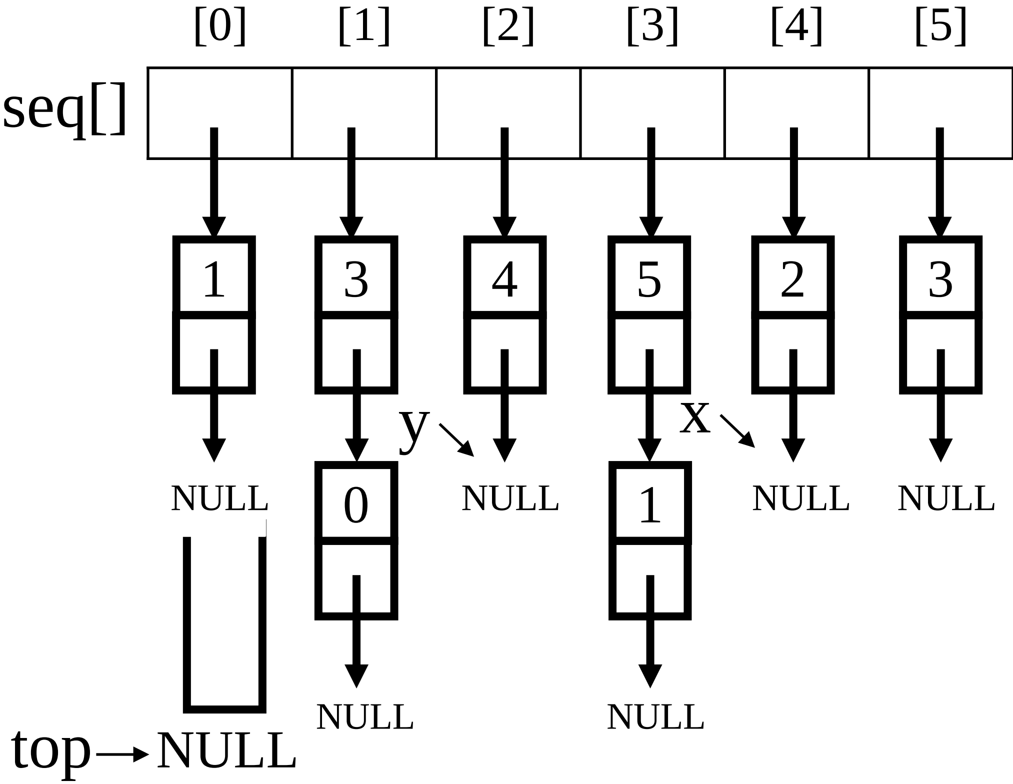
```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){    x=NULL
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

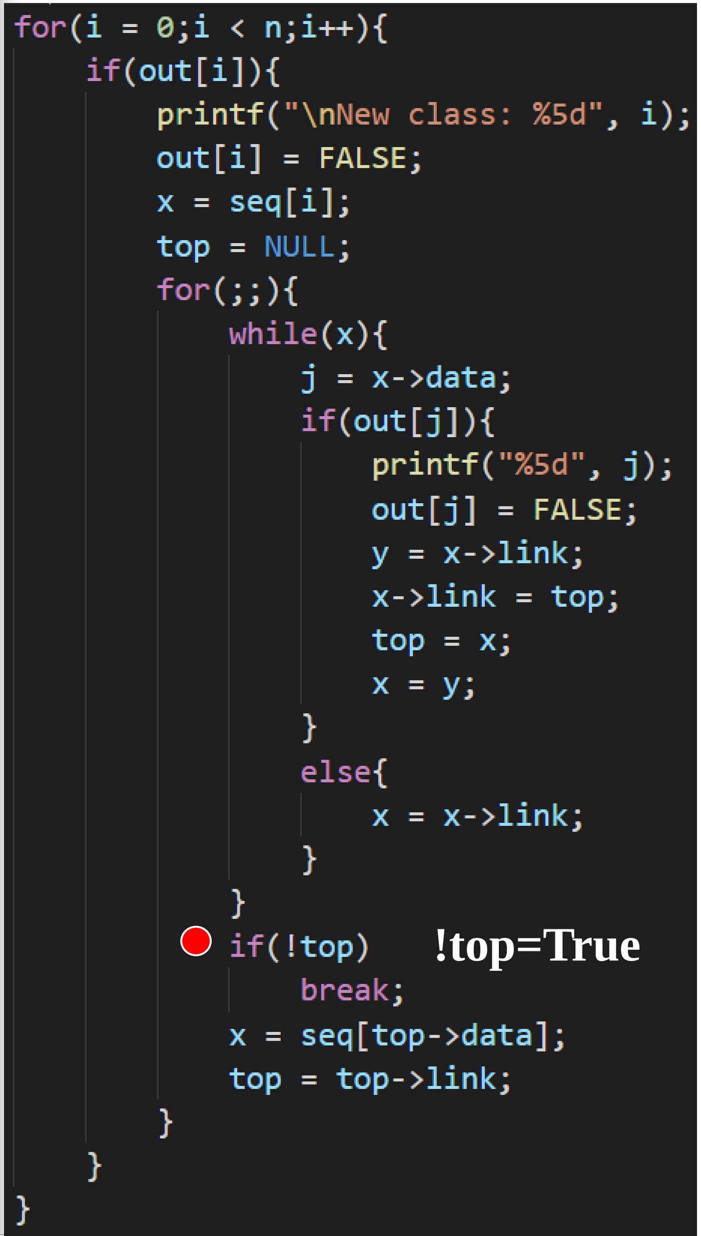
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2    4
```

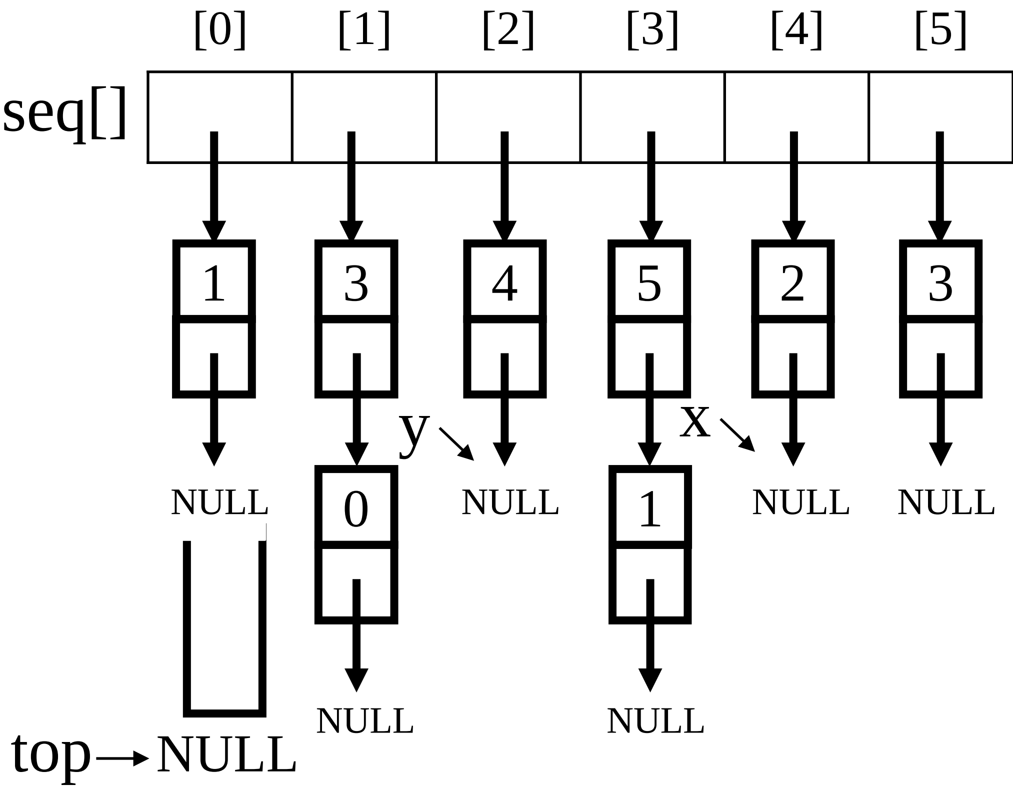


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

!top=True

i = 2   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

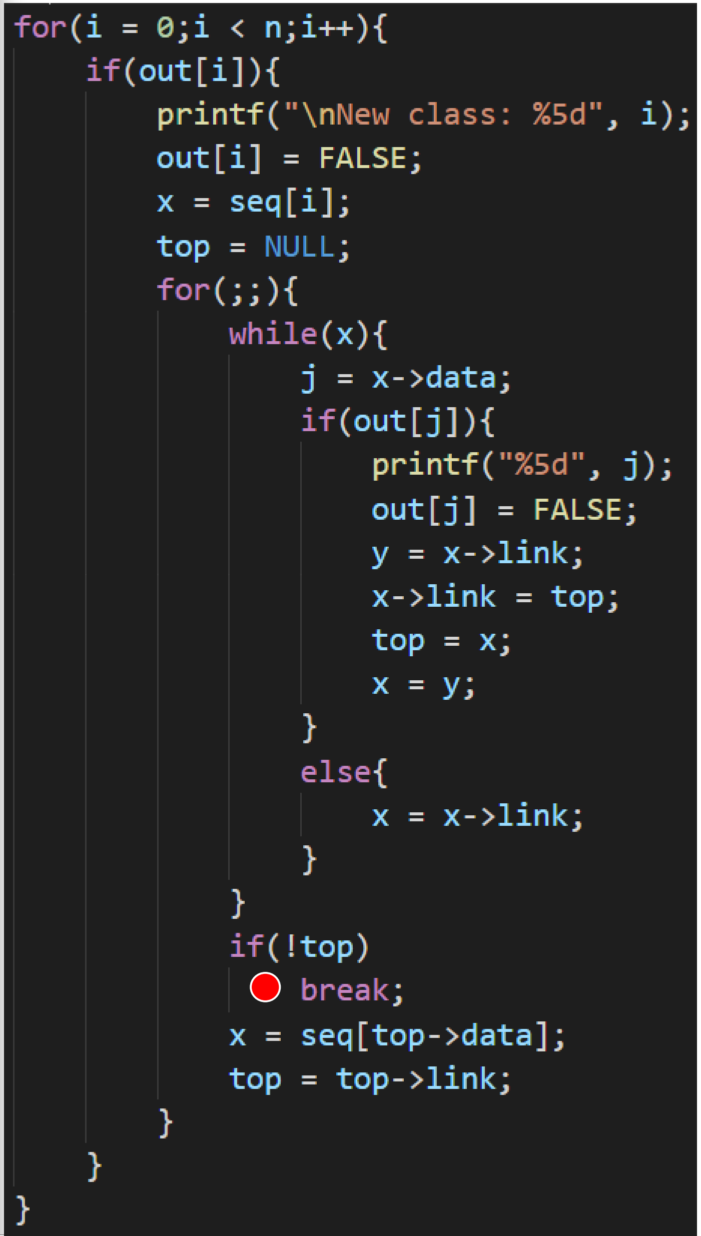
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

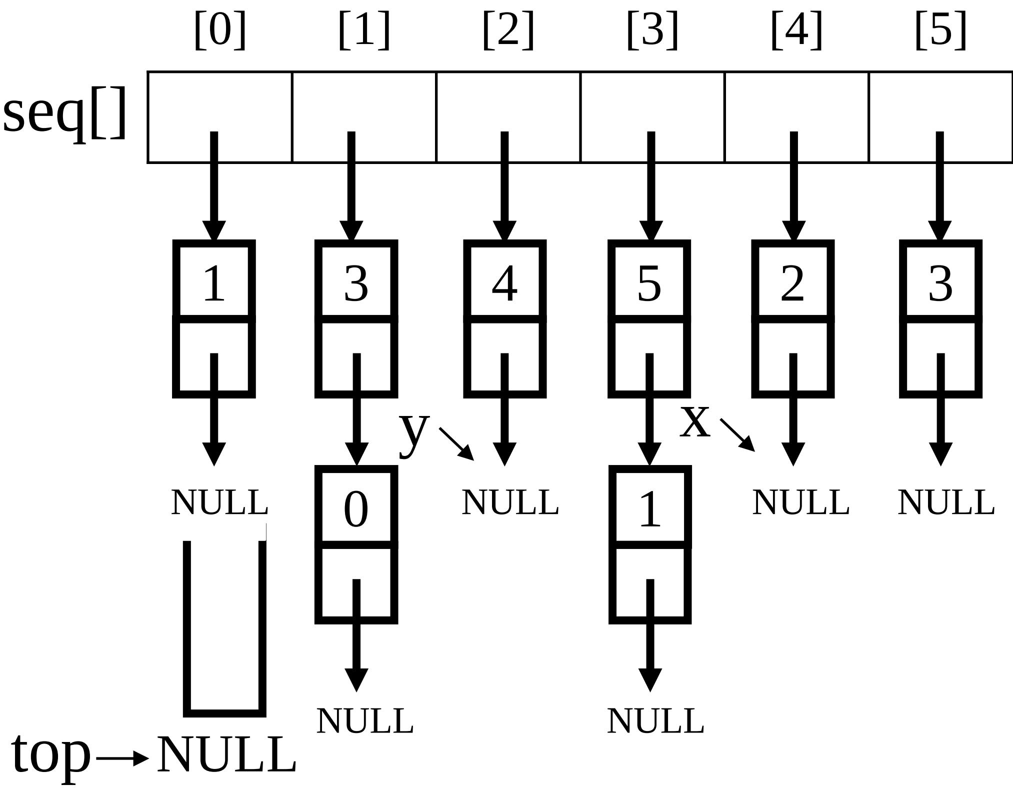
```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

$i = 2$ $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

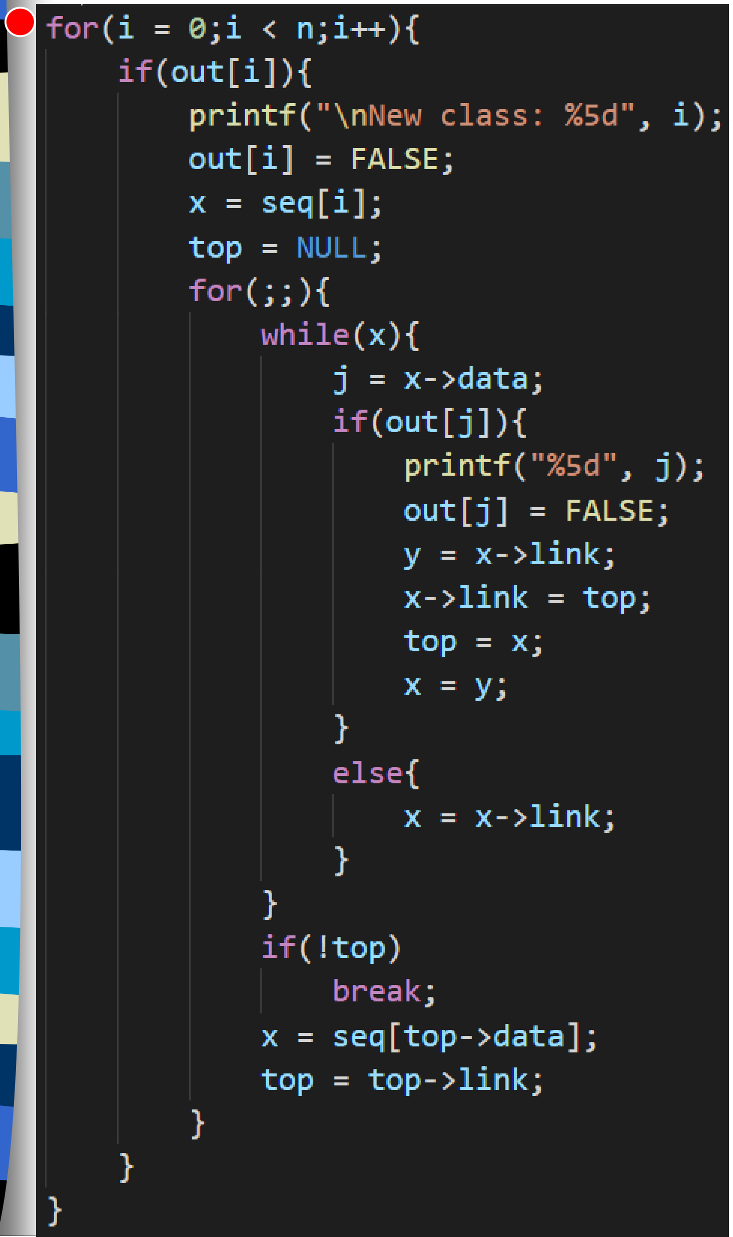
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

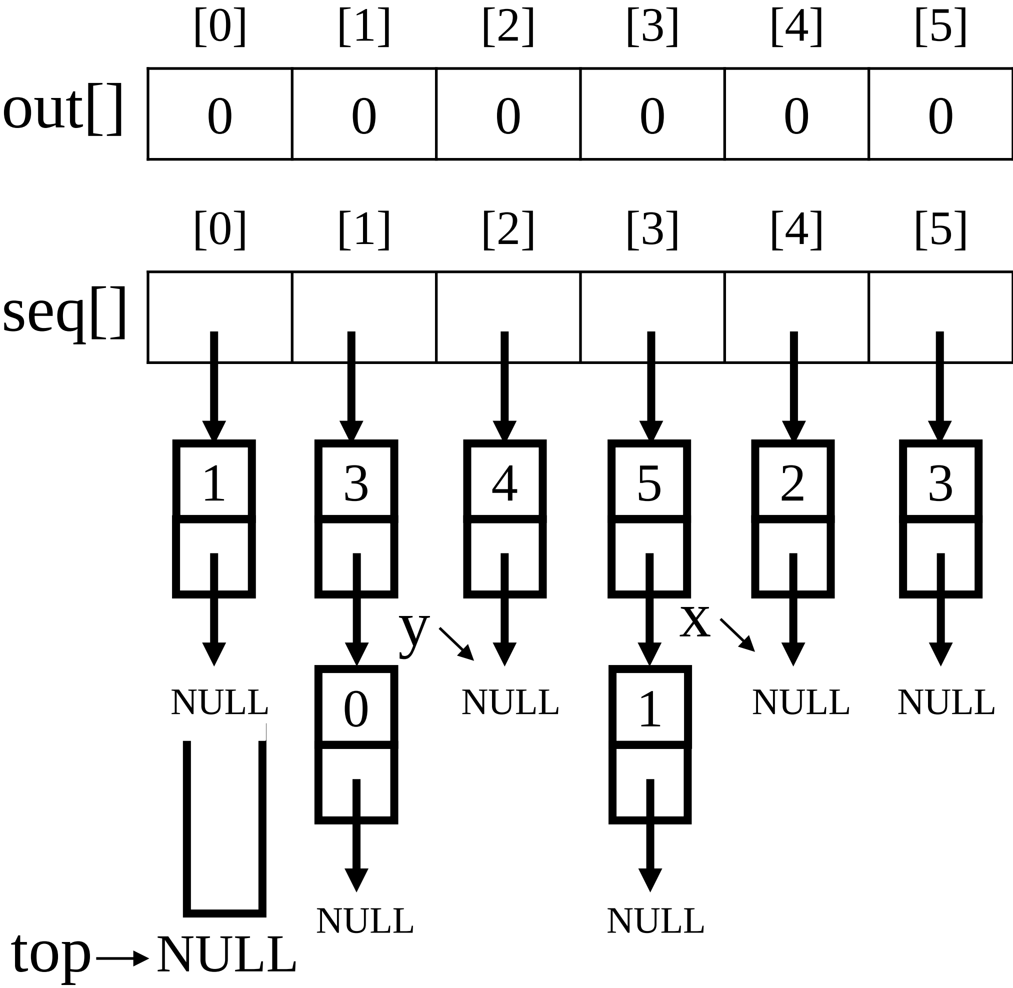
Phase 2: output the equivalence classes

```
New class:   0   1   3   5
New class:   2   4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

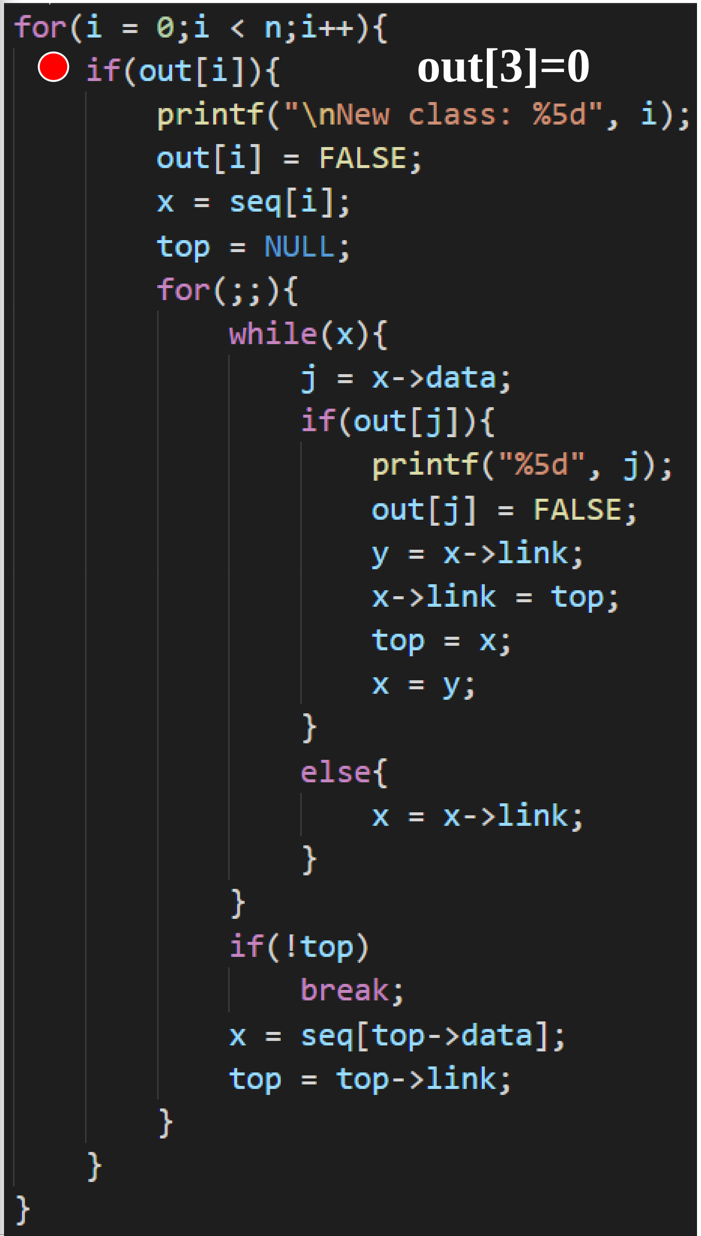
i = 3   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4



Equivalence relations

Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2    4
```

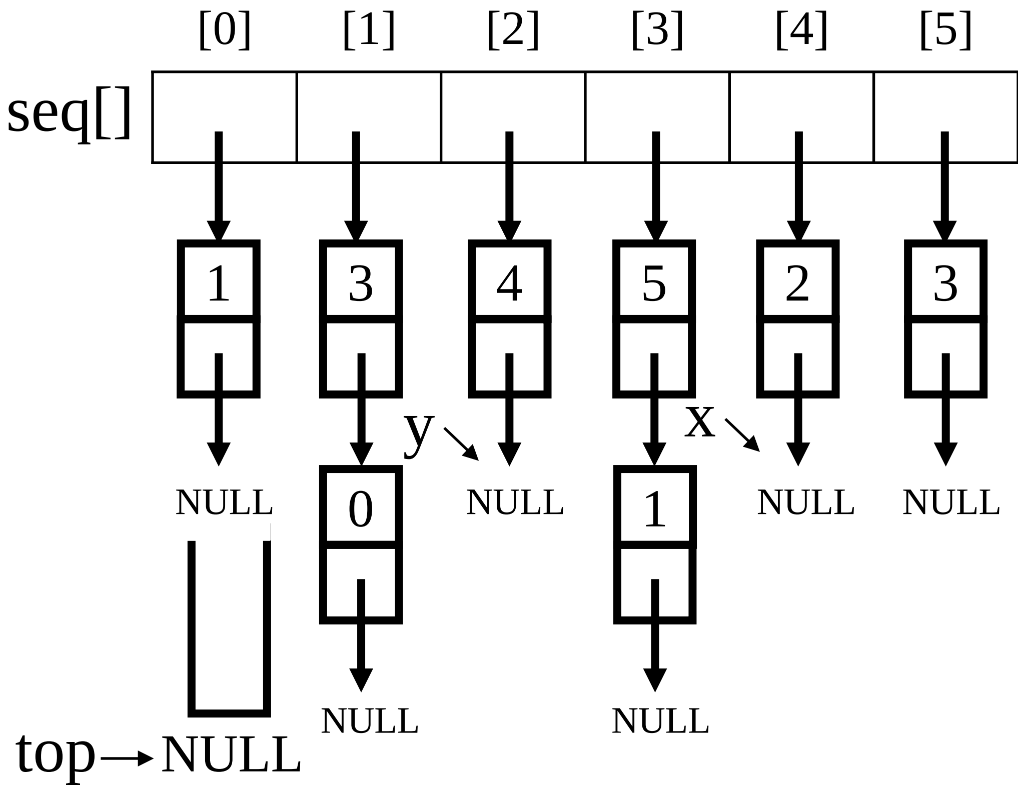


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[3]=0

i = 3 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

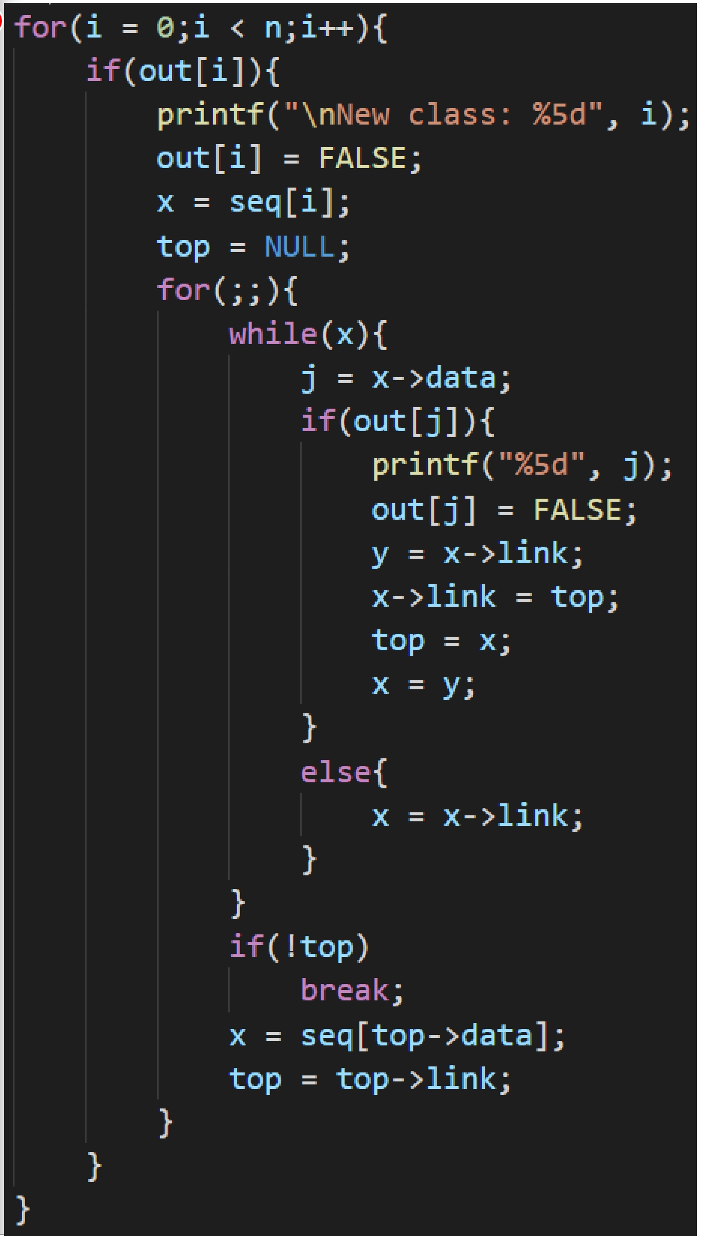
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

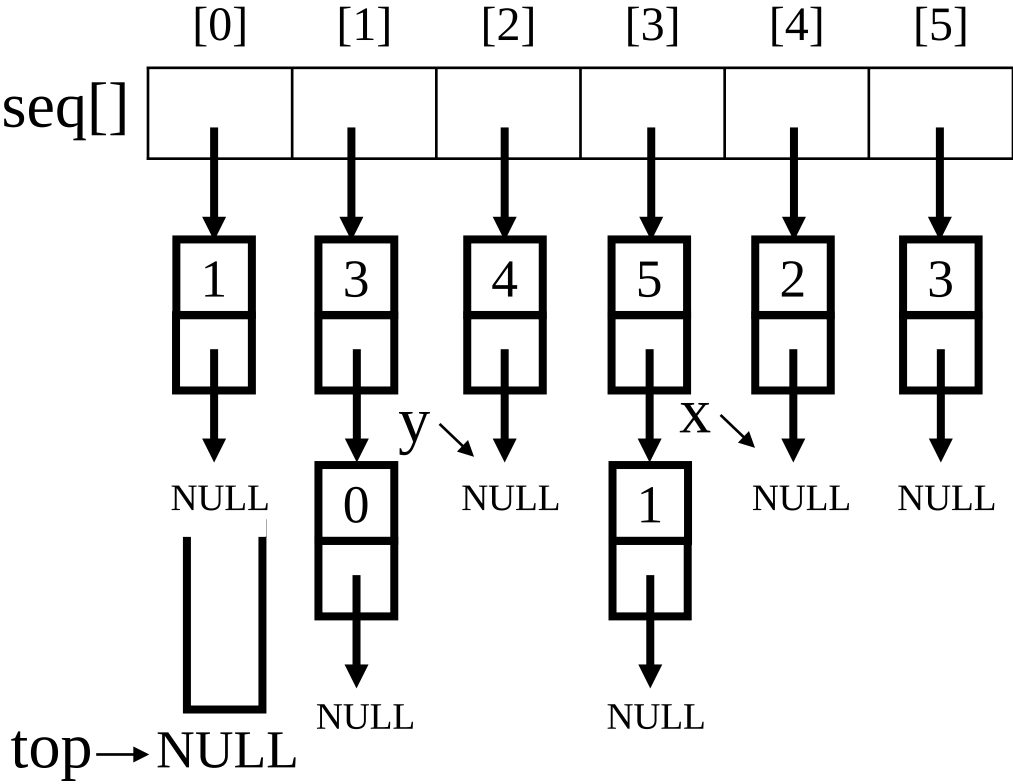
```
New class:   0   1   3   5
New class:   2   4
```



```c
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 4   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

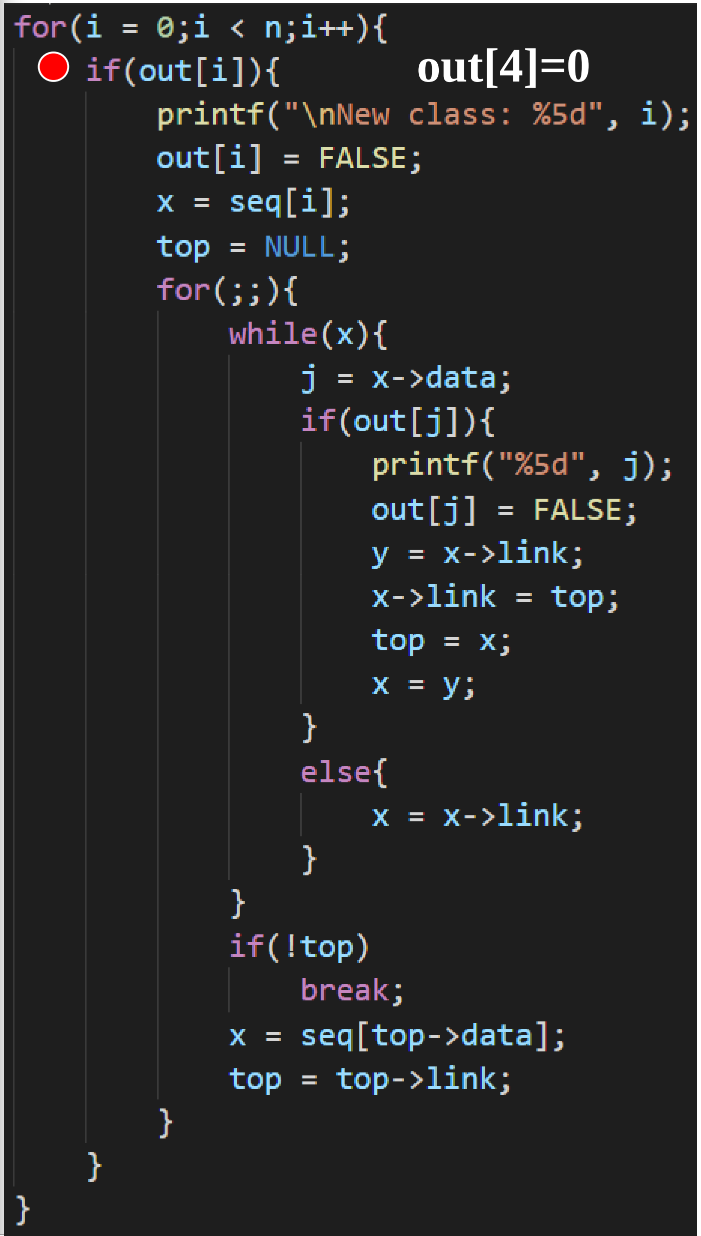
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

```
New class:    0    1    3    5
New class:    2    4
```

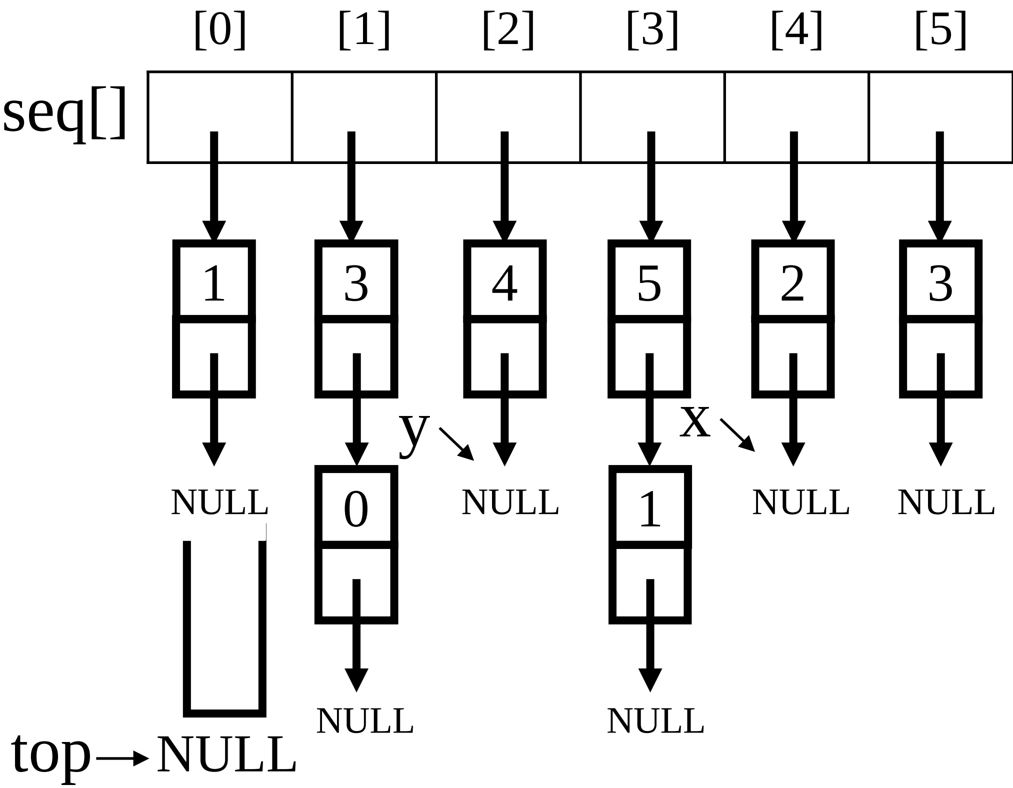


```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[4]=0

i = 4   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

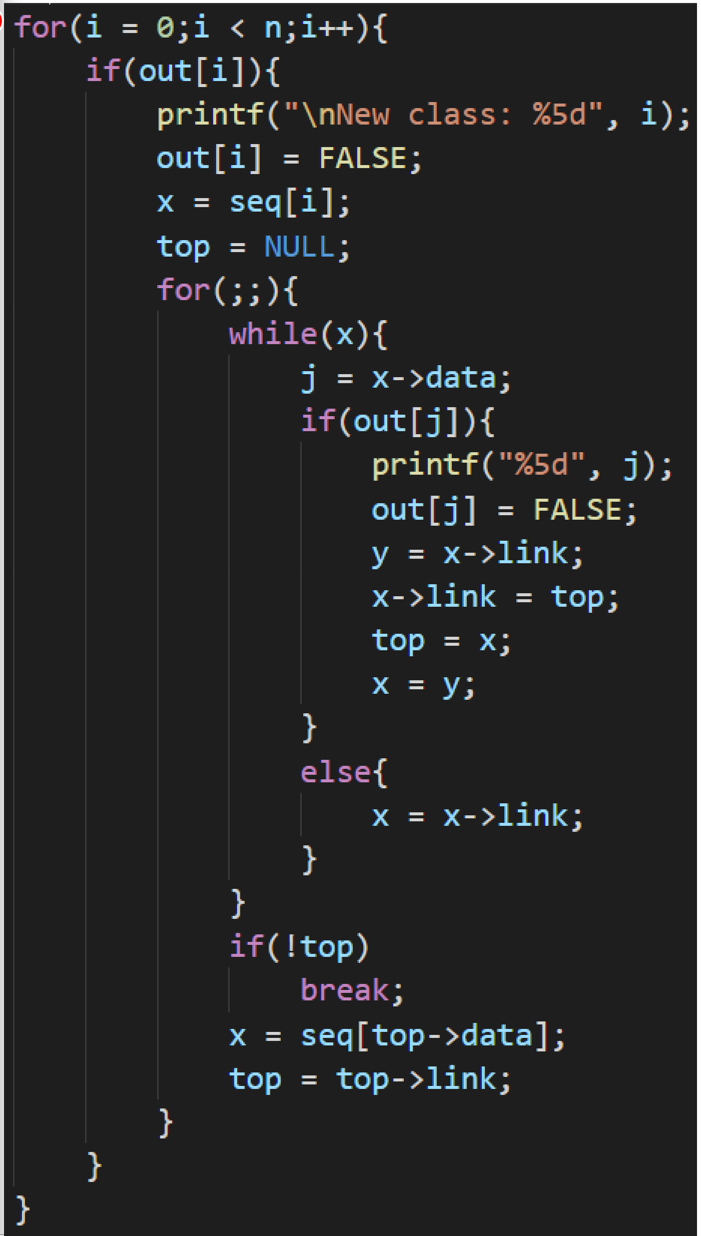
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

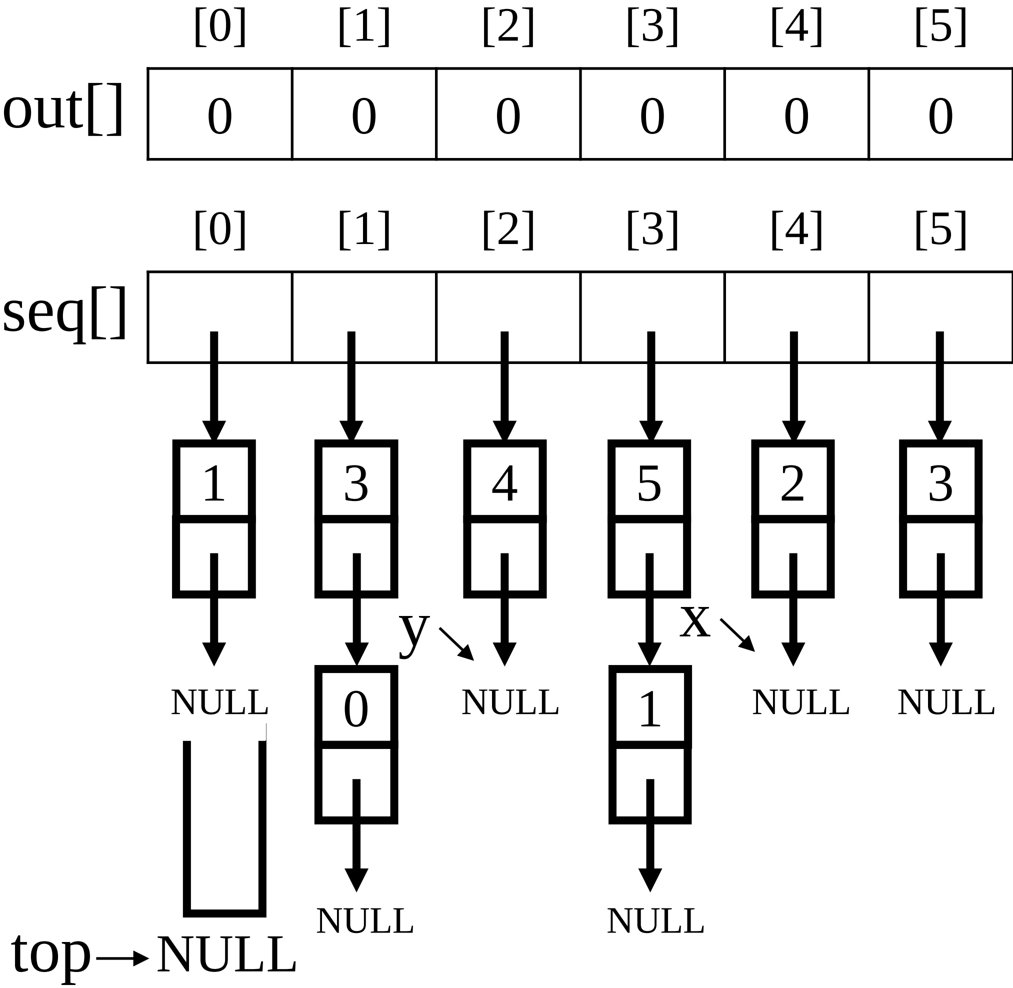
Phase 2: output the equivalence classes

```
New class:     0    1    3    5
New class:     2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

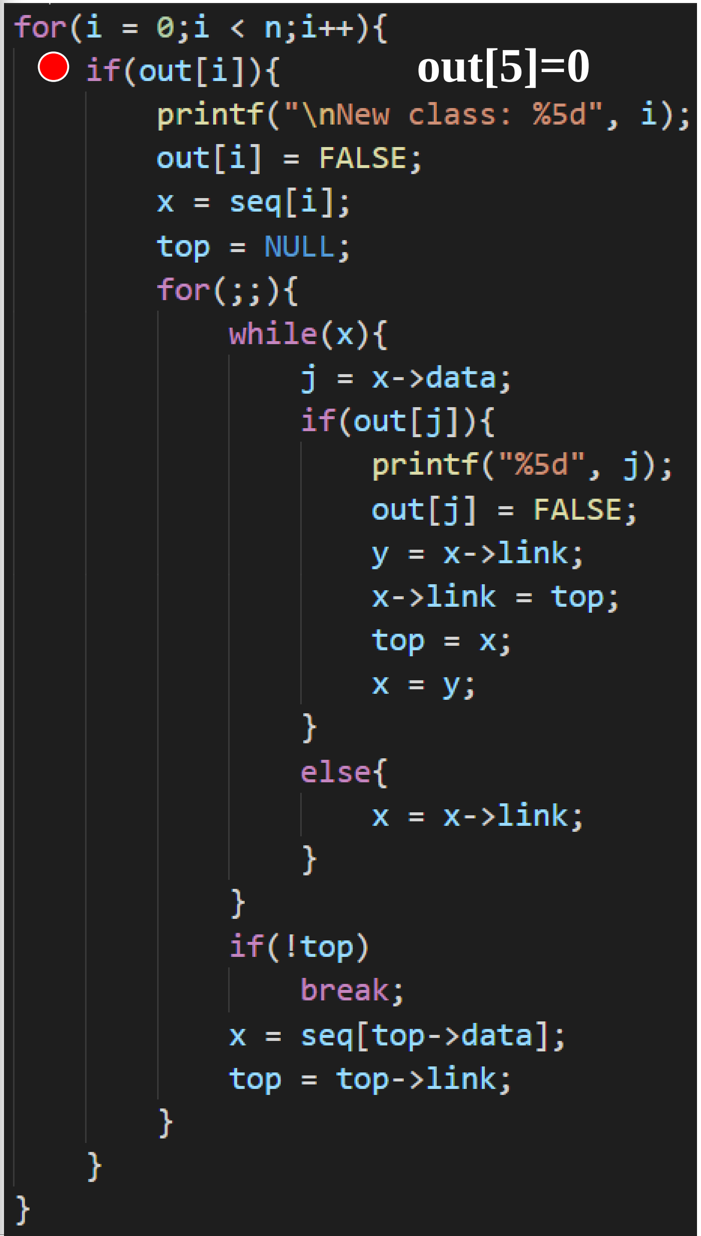
i = 5   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4



Equivalence relations

Phase 2: output the equivalence classes

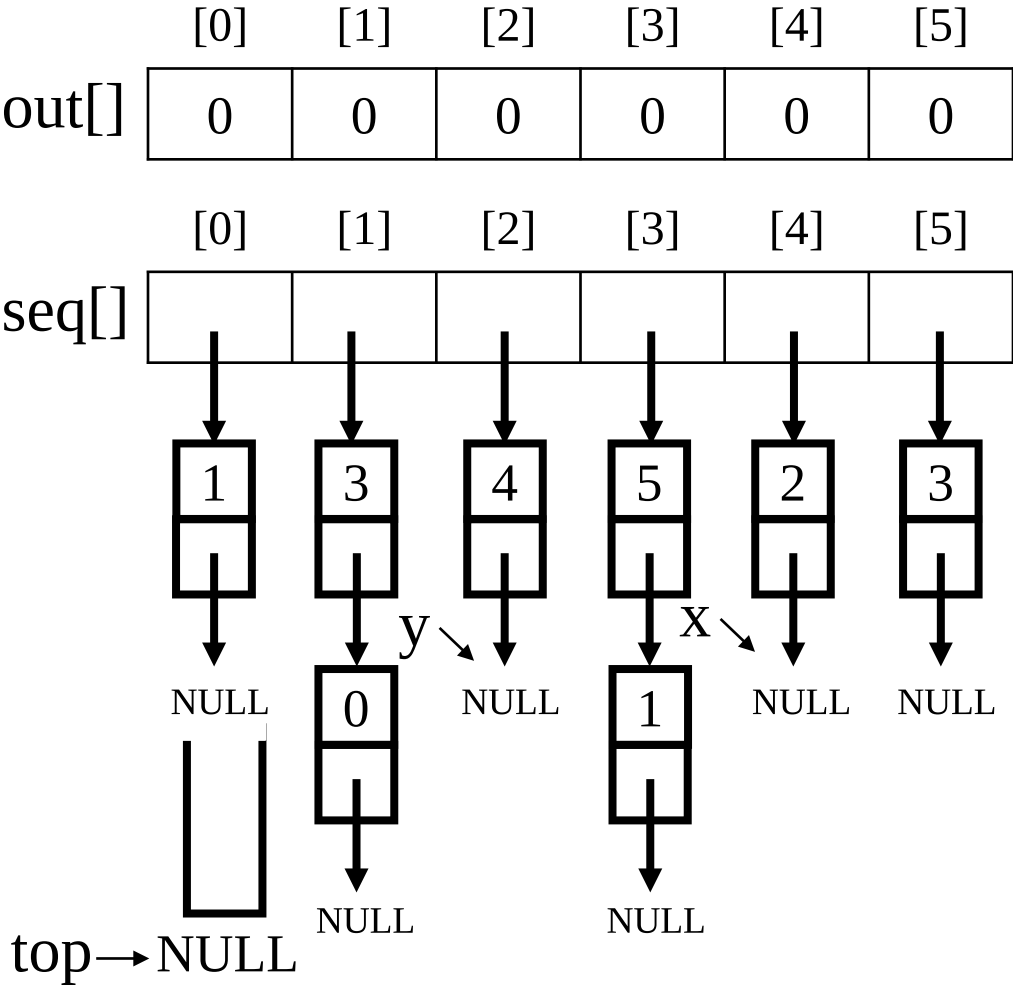
```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

out[5]=0

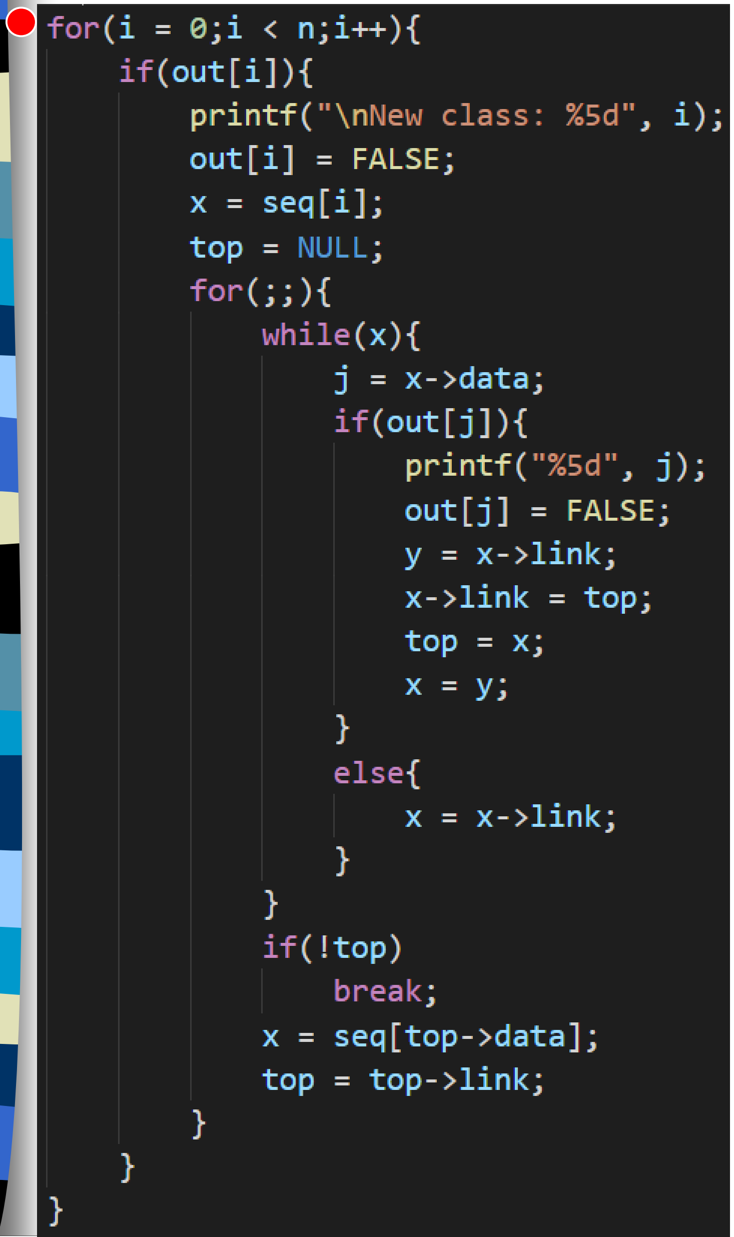
i = 5   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4



Equivalence relations

Phase 2: output the equivalence classes

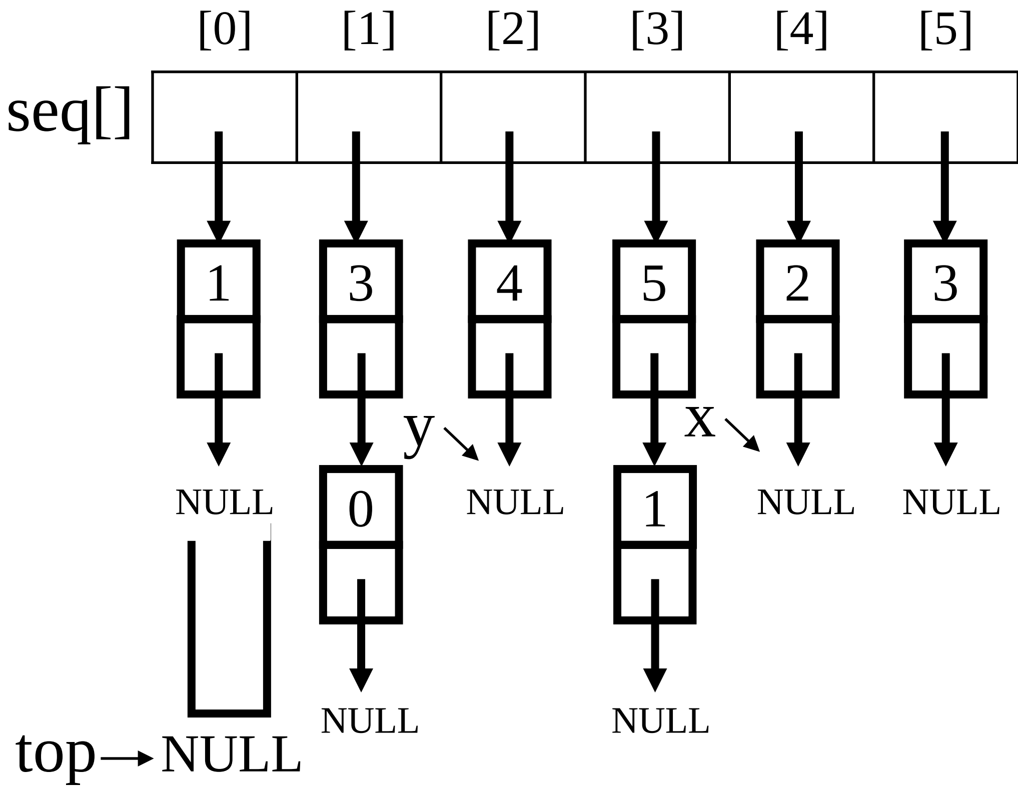
```
New class:    0    1    3    5
New class:    2    4
```



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 6 $0 \equiv 1$ $1 \equiv 3$ $3 \equiv 5$ $2 \equiv 4$

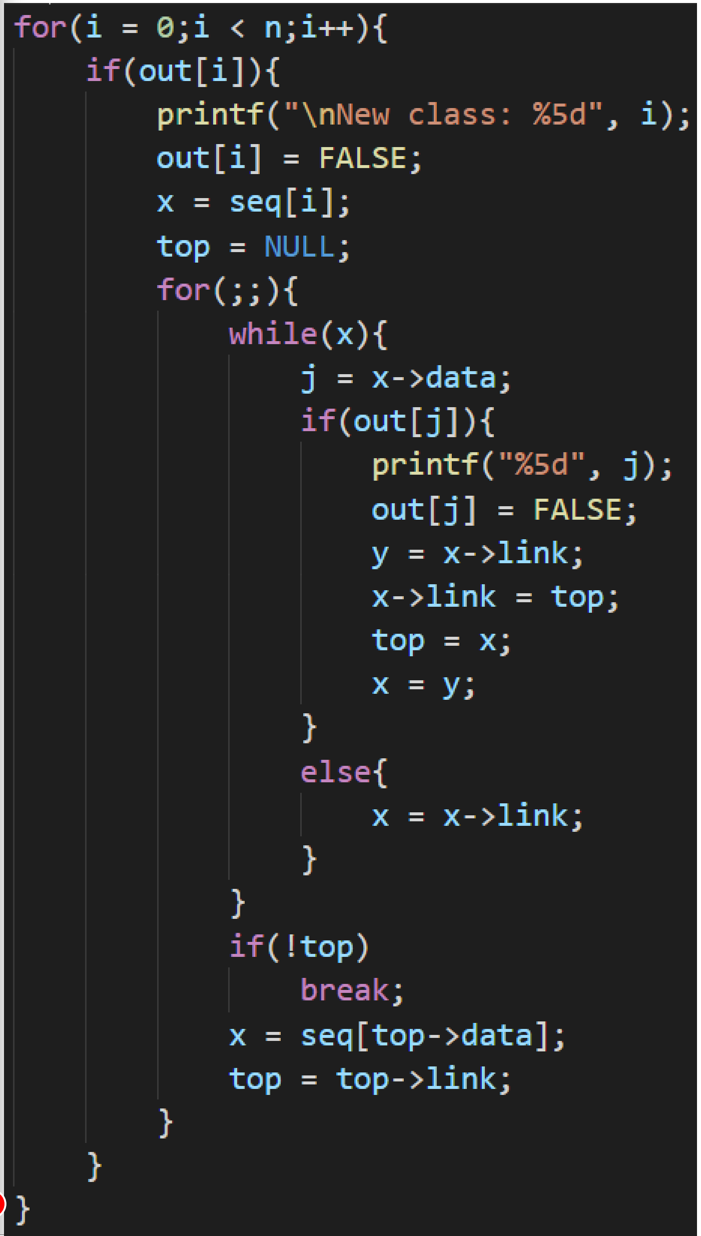
	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Equivalence relations

Phase 2: output the equivalence classes

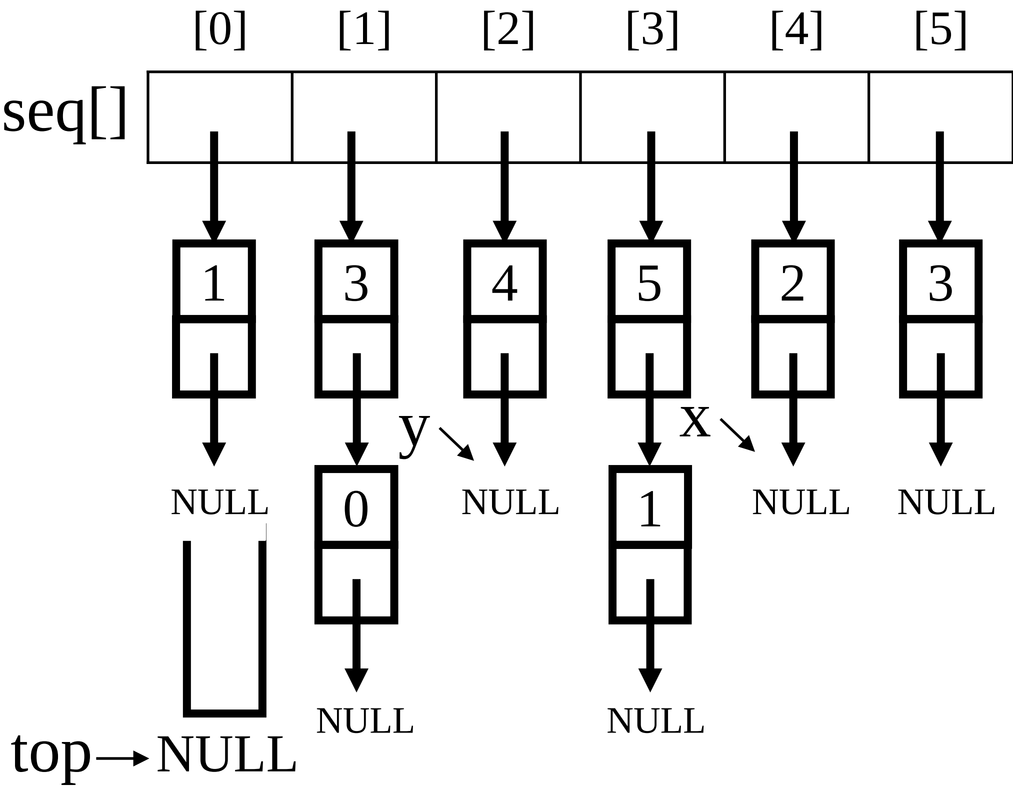
New class: 0 1 3 5
New class: 2 4



```
for(i = 0;i < n;i++){
    if(out[i]){
        printf("\nNew class: %5d", i);
        out[i] = FALSE;
        x = seq[i];
        top = NULL;
        for(;;){
            while(x){
                j = x->data;
                if(out[j]){
                    printf("%5d", j);
                    out[j] = FALSE;
                    y = x->link;
                    x->link = top;
                    top = x;
                    x = y;
                }
                else{
                    x = x->link;
                }
            }
            if(!top)
                break;
            x = seq[top->data];
            top = top->link;
        }
    }
}
```

i = 6   0 ≡ 1   1 ≡ 3   3 ≡ 5   2 ≡ 4

	[0]	[1]	[2]	[3]	[4]	[5]
out[]	0	0	0	0	0	0



Final Version for Finding Equivalence Classes

```
void main(void)
{
  short int out[MAX_SIZE];
  node_pointer seq[MAX_SIZE];
  node_pointer x, y, top;
  int i, j, n;
  printf(“Enter the size (<= %d)“, MAX_SIZE);
  scanf(“%d”, &n);
  for (i=0; i<n; i++) {
      out[i]= TRUE;    seq[i]= NULL;
  }
  printf(“Enter a pair of numbers (-1 -1 to quit): “);
  scanf(“%d%d”, &i, &j);
```

Phase 1: input the equivalence pairs:

```
while (i>=0) {
    x = (node_pointer) malloc(sizeof(node));
    if (IS_FULL(x))
      fprintf(stderr, “memory is full\n”);
        exit(1);
    }   Insert x to the top of lists seq[i]
    x->data= j;  x->link= seq[i];  seq[i]= x;
    if (IS_FULL(x))
      fprintf(stderr, “memory is full\n”);
        exit(1);
    }   Insert x to the top of lists seq[j]
    x->data= i;  x->link= seq[j];  seq[j]= x;
    printf(“Enter a pair of numbers (-1 -1 to \
          quit): “);
    scanf(“%d%d”, &i, &j);
}
```

Phase 2: output the equivalence classes

```
for (i=0; i<n; i++)
 {
    if (out[i]) {
        printf(“\nNew class: %5d”, i);
        out[i]= FALSE;
        x = seq[i];    top = NULL;
        for (;;)
        {
           while (x)
           {
                j = x->data;
                if (out[j]) {      Move down
                    printf(“%5d”, j);
                    out[j] = FALSE;
                    y = x->link;  x->link = top;
                    top = x;  x = y;
                }
                else x = x->link; Next x
            }
            if (!top) break;
            x = seq[top->data];  top = top->link;
        }
     }
} // main
```

4.7 Sparse Matrices

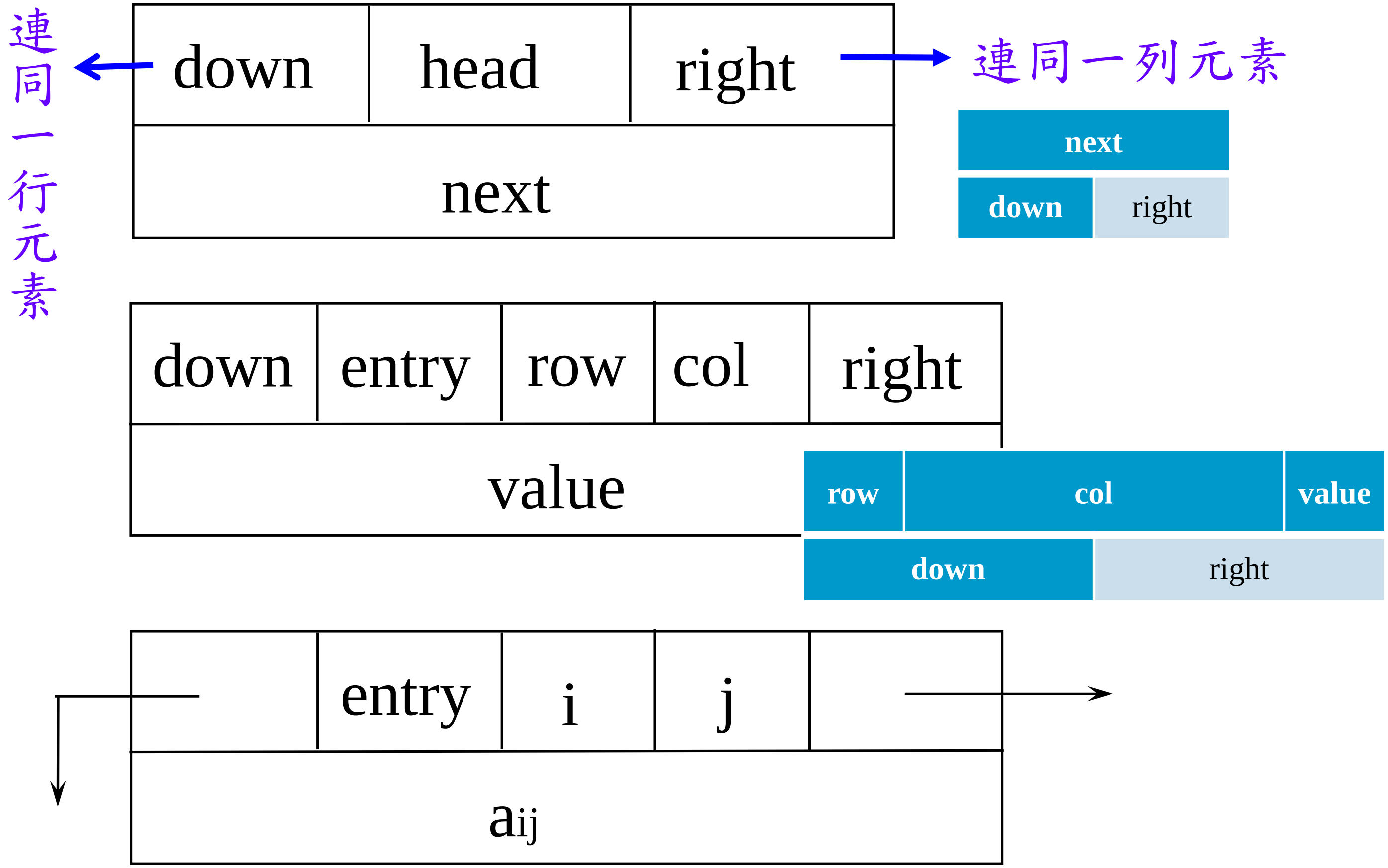
$$\begin{bmatrix} 0 & 0 & 11 & 0 \\ 12 & 5 & 0 & 0 \\ 0 & -4 & 0 & 0 \\ 0 & 0 & 0 & -15 \end{bmatrix}$$

inadequates of sequential schemes
(1) # of nonzero terms will vary after some matrix computation
(2) matrix just represents intermediate results

New scheme
Each column (row): a circular linked list with a head node

Revisit Sparse Matrices

# of head nodes = max{# of rows, # of columns}



head node

連同一行元素

down | head | right

next

連同一列元素

next

down | right

entry node

down | entry | row | col | right

value

row | col | value

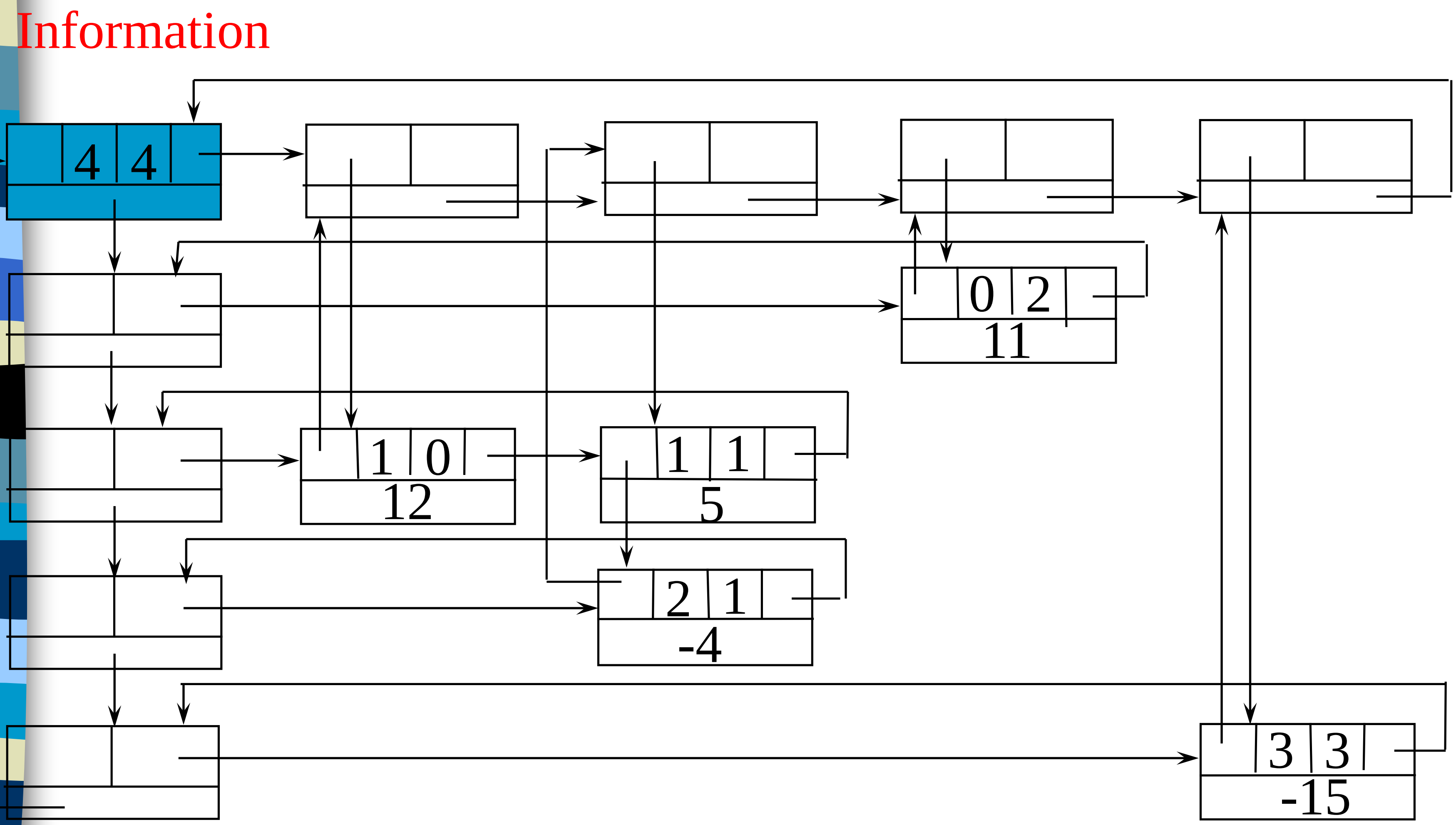
down | right

aij

entry | i | j

a_{ij}

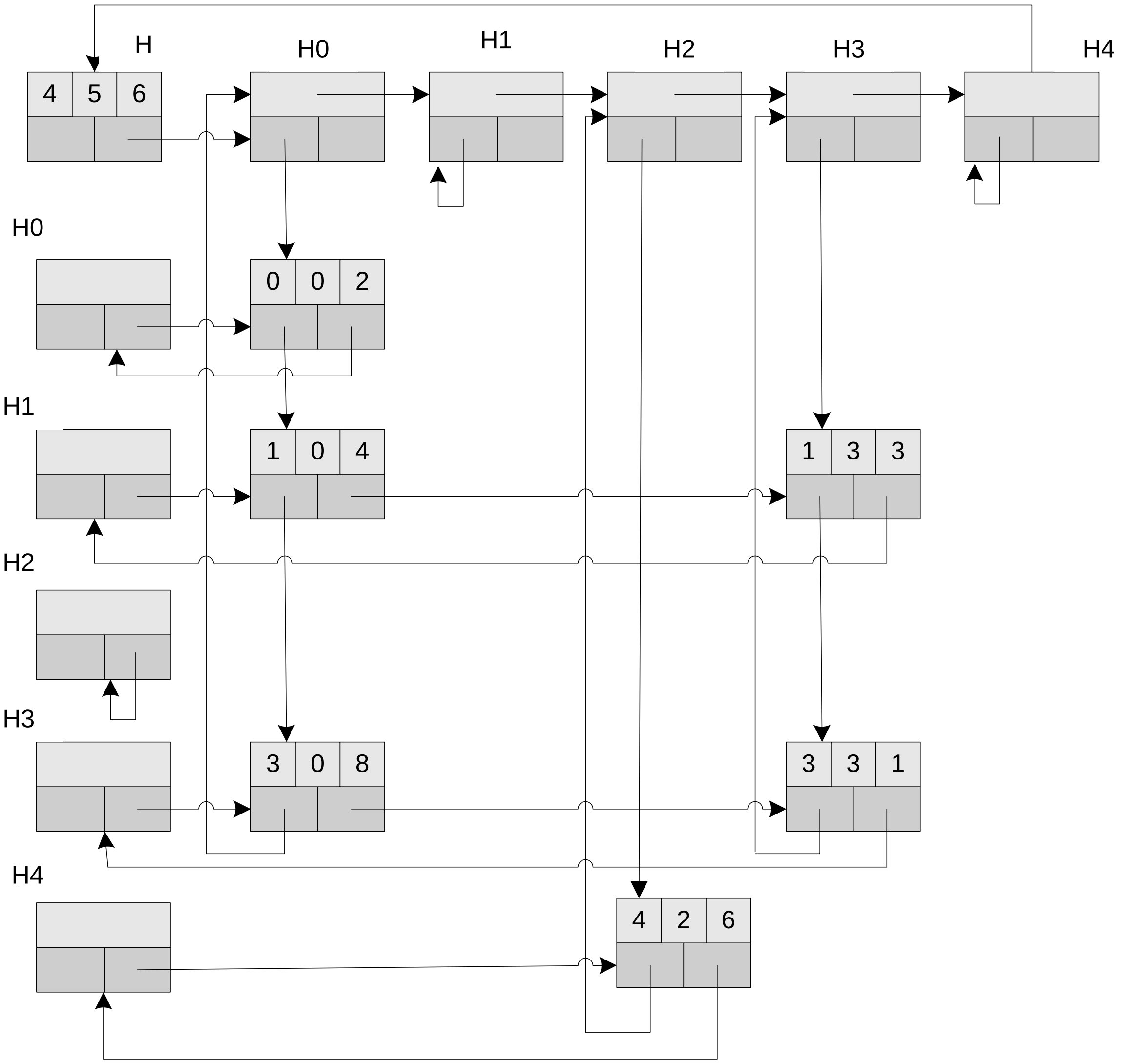
Linked Representation for Matrix



Circular linked list

```
#define MAX_SIZE 50 /* size of largest matrix */
typedef enum {head, entry} tagfield;
typedef struct matrixNode *matrixPointer;
typedef struct entryNode {
          int row;
          int col;
          int value;
          };
typedef struct matrixNode {
          matrixPointer down;
          matrixPointer right;
          tagfield tag;     → head or entry
          union {
                   matrixPointer next;
                   entryNode entry;
                  } u;
           };
matrixPointer hdnode[MAX_SIZE];
```

Information



$$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 4 & 0 & 0 & 3 \\ 0 & 0 & 0 & 0 \\ 8 & 0 & 0 & 1 \\ 0 & 0 & 6 & 0 \end{bmatrix}$$

Read in a Matrix

```
matrix_pointer mread(void)
{
/* read in a matrix and set up its linked
 list. An global array hdnode is used */
   int num_rows, num_cols, num_terms;
   int num_heads, i;
   int row, col, value, current_row;
   matrixPointer temp, last, node;

   printf(“Enter the number of rows, columns
         and number of nonzero terms: “);
```

```
scanf(“%d%d%d”, &num_rows, &num_cols,
      &num_terms);
 num_heads =
 (num_cols > num_rows)? num_cols : num_rows;
 /* set up head node for the list of head
    nodes; upper left corner (左上角)*/
 node = new_node();    node->tag = entry;
 node->u.entry.row = num_rows;
 node->u.entry.col = num_cols;

 if (!num_heads) node->right = node;
 else { /* initialize the head nodes */
   for (i=0; i<num_heads; i++) {
     temp= new_node();
     hdnode[i] = temp;
     hdnode[i]->tag = head;
     hdnode[i]->right = temp;
     hdnode[i]->u.next = temp;   O(max(n,m))
   }
```

```
current_row= 0;    last= hdnode[0];
/*last node in current row*/
for (i=0; i<num_terms; i++)
{
  printf(“Enter row, column and value:”);
  scanf(“%d%d”, &row, &col, &value);
  if (row>current_row) { /*close current row*/
    last->right= hdnode[current_row];
    current_row= row;   last=hdnode[row];
  }
  MALLOC(temp, sizeof(*temp));
  temp->tag=entry;
  temp->u.entry.row=row;
  temp->u.entry.col = col;
  temp->u.entry.value = value;
  last->right = temp; /*link to row list */
  last= temp;
  /* link to column list */
  hdnode[col]->u.next->down = temp;
  hdnode[col]->u.next = temp;
}
```

利用next field 存放column的last node

```
    /*close last row */
    last->right = hdnode[current_row];
    //current_row =row;
    /* close all column lists */
   /*將此列每一個行串列最後一節點指向同一串列的第一個節點*/
    for (i=0; i<num_cols; i++)
      hdnode[i]->u.next->down = hdnode[i];
    /* link all head nodes together */
    /*將所有標頭節點串連起來*/
    for (i=0; i<num_heads-1; i++)
      hdnode[i]->u.next = hdnode[i+1];
    hdnode[num_heads-1]->u.next= node;
    node->right = hdnode[0];
  }
  return node;
}
```

O(max{#_rows, #_cols}+#_terms)

Write out a Matrix

```
void mwrite(matrix_pointer node)
{ /* print out the matrix in row major form */
  int i;
  matrix_pointer temp, head = node->right;
  printf("\n num_rows = %d, num_cols= %d\n",
         node->u.entry.row,node->u.entry.col);
  printf("The matrix by row, column, and
         value:\n\n");
  for (i=0; i<node->u.entry.row; i++) {
    for (temp=head->right;temp!=head;temp=temp->right)
      printf("%5d%5d%5d\n", temp->u.entry.row,
           temp->u.entry.col, temp->u.entry.value);
    head= head->u.next; /* next row */
  }
}
```

#_rows

#_terms

O(#_rows+#_terms)

Free the entry and head nodes by row.

Erase a Matrix

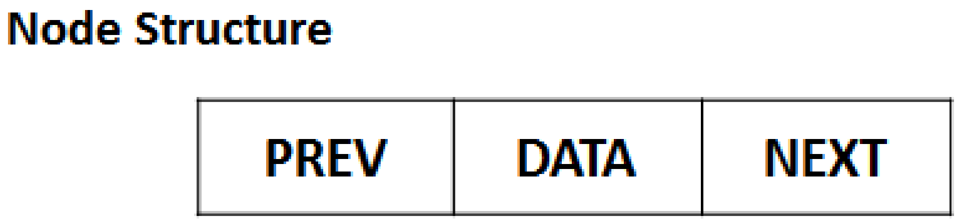
```
void merase(matrix_pointer *node)
{
  int i, num_heads;
  matrix_pointer x, y, head = (*node)->right;

  /*free the entry and header nodes by row*/

  for (i=0; i<(*node)->u.entry.row; i++) {
    y=head->right;
    while (y!=head) {
      x = y;  y = y->right;  free(x);
    }
    x= head;  head= head->u.next; /* next row */
    free(x);
  }
  /*free remaining header nodes*/
  y = head;
  while (y!=*node) {
    x = y;  y = y->u.next;  free(x);
  }
  free(*node);  *node = NULL;
}
```

Doubly Linked List

Move in forward and backward direction.

Singly linked list (in one direction only)
How to get the preceding node during deletion or insertion?

Using 2 pointers



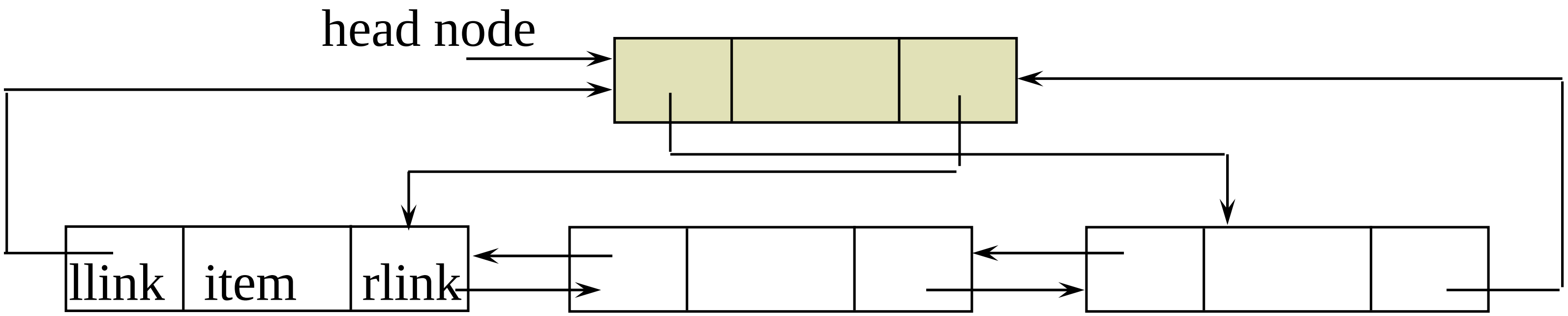
Node Structure

PREV	DATA	NEXT

Doubly Linked Lists

```
typedef struct node *node_pointer;
typedef struct node {
    node_pointer llink;
    element item;
    node_pointer rlink;
}
```

ptr
= ptr->rlink->llink
= ptr->llink->rlink



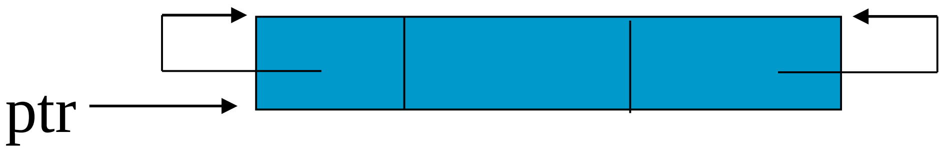


***Figure 4.22:**Empty doubly linked circular list with header node

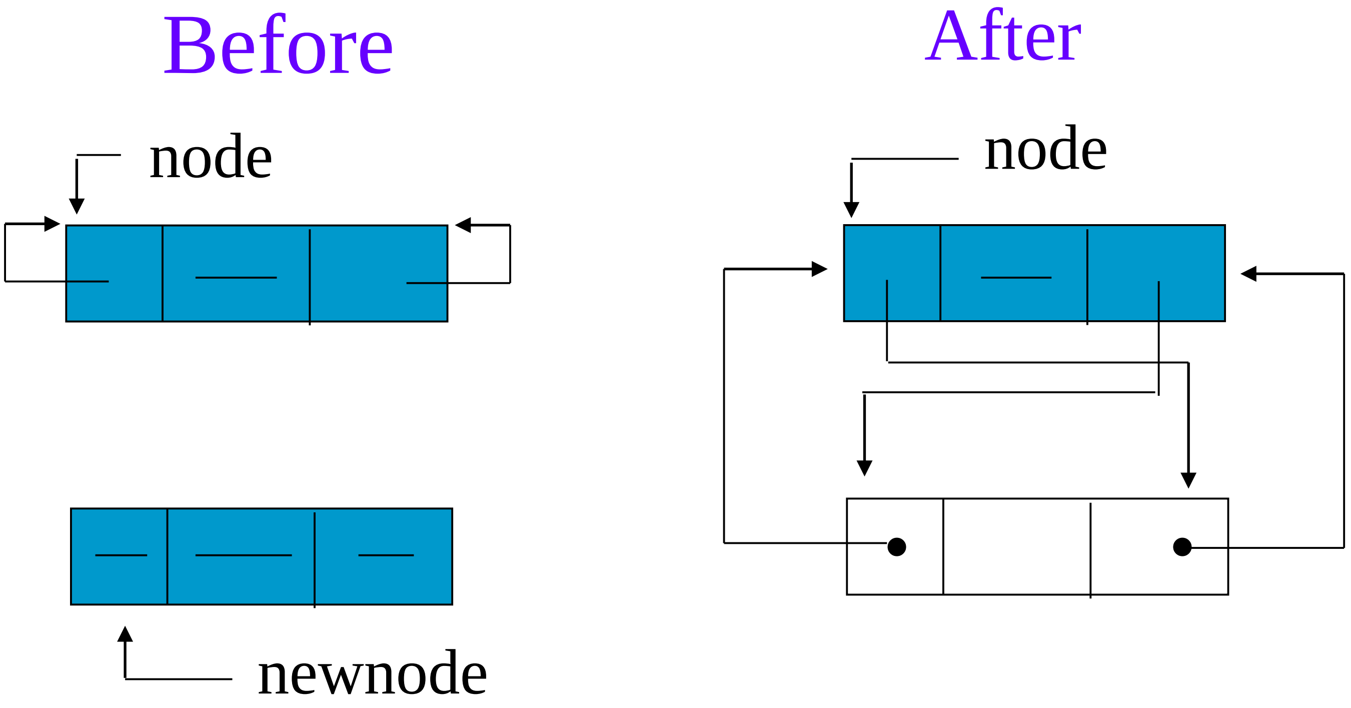
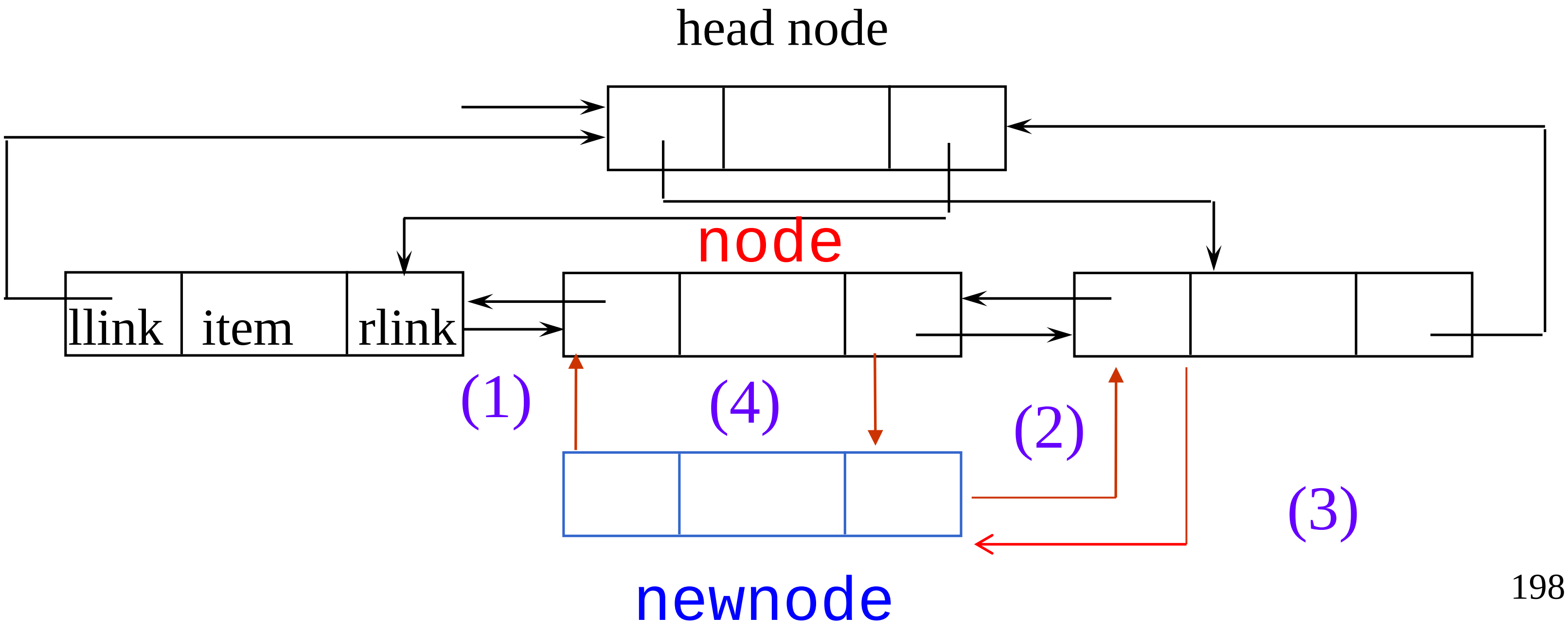


***Figure 4.25:** Insertion into an empty doubly linked circular list

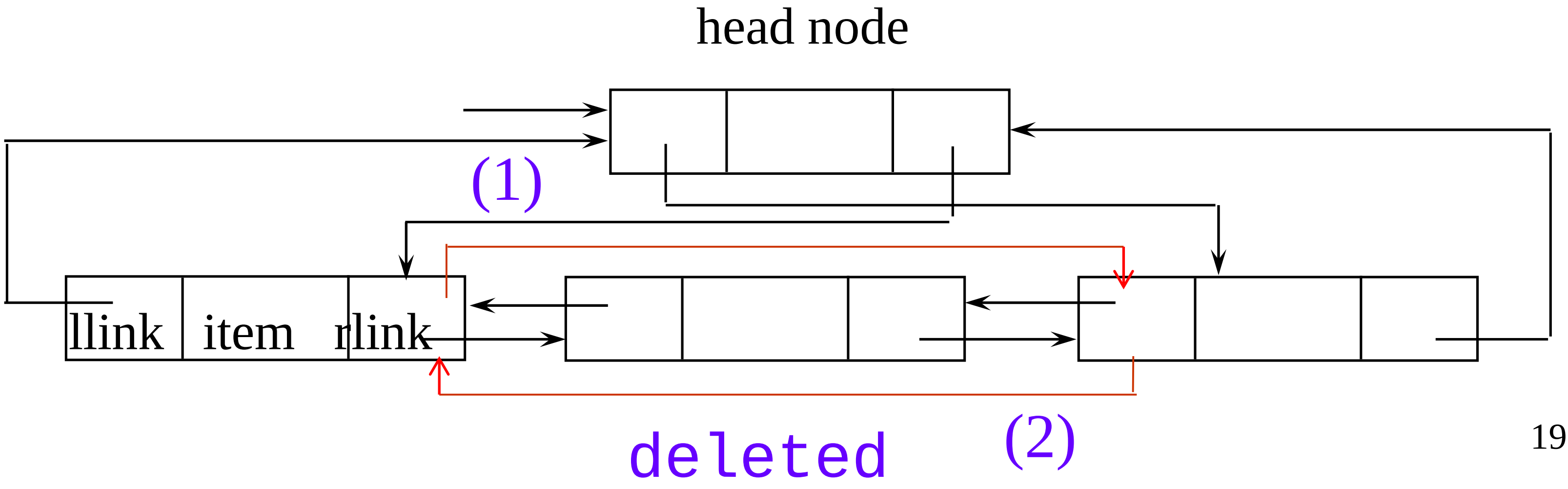
Insert

```
void dinsert(node_pointer node, node_pointer newnode)
{
    (1) newnode->llink = node;
    (2) newnode->rlink = node->rlink;
    (3) node->rlink->llink = newnode;
    (4) node->rlink = newnode;
}
```



Delete

```
void ddelete(node_pointer node, node_pointer deleted)
{
    if (node==deleted) printf("Deletion of head node
                             not permitted.\n");
    else {
        (1) deleted->llink->rlink= deleted->rlink;
        (2) deleted->rlink->llink= deleted->llink;
            free(deleted);
    }
}
```


head node
(1)
llink
item
rlink
deleted
(2)