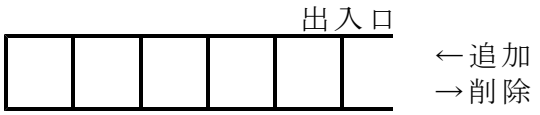
スタック・キューＩ

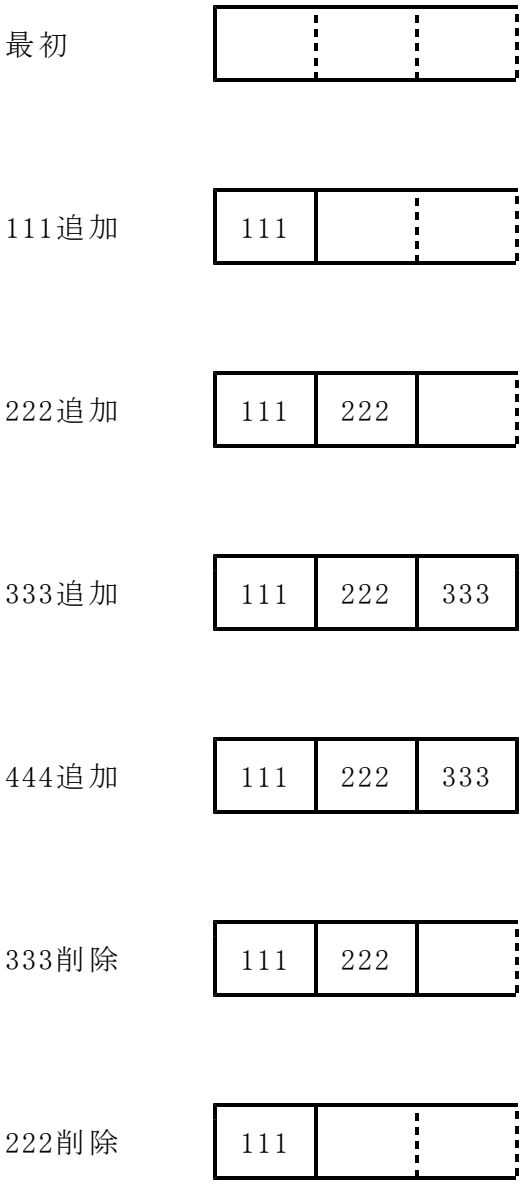
０．目次

１．スタック

スタックは、データの出し入れが一カ所で行われ、操作は追加と削除ができるデータ構造をいう。

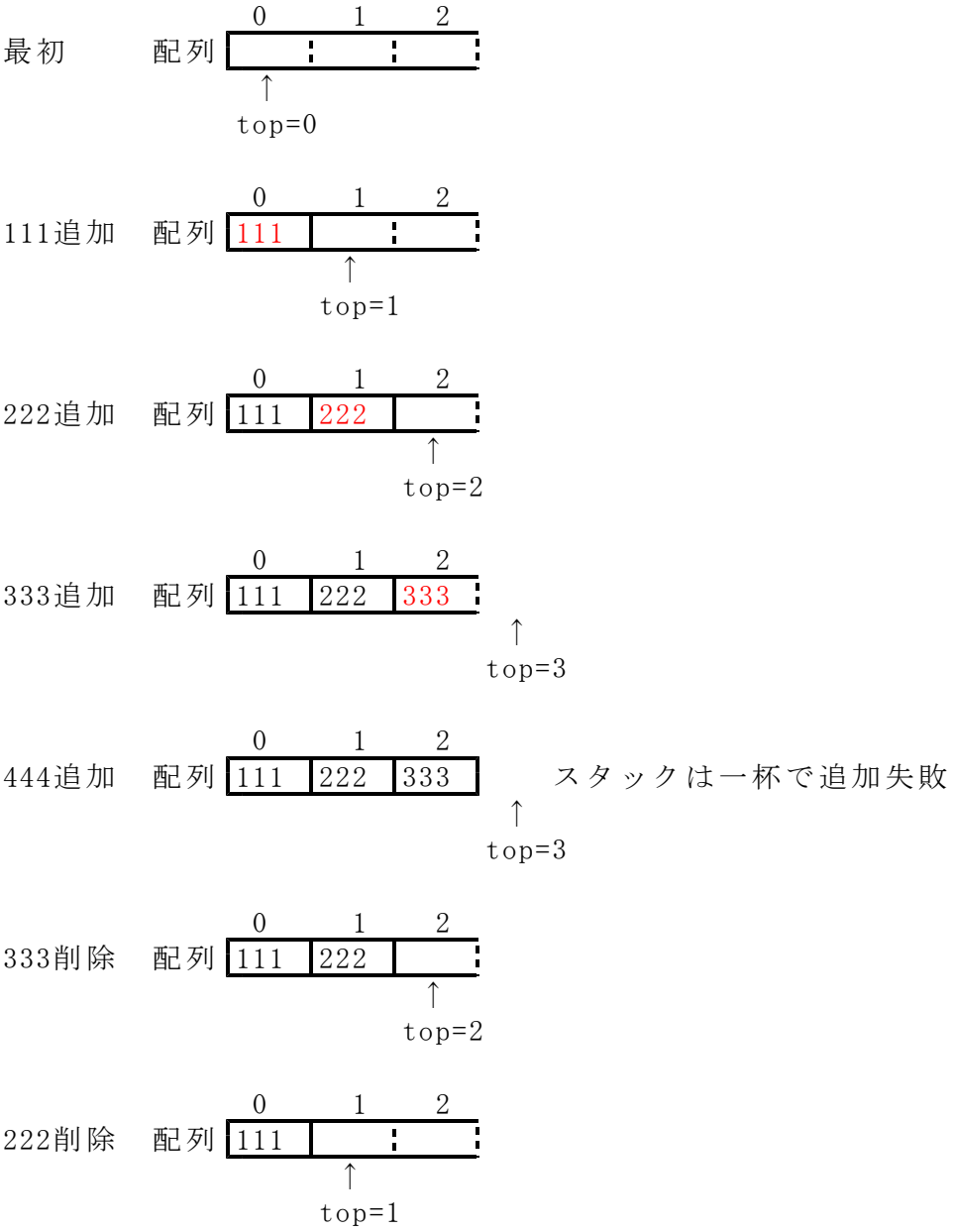


●スタック操作



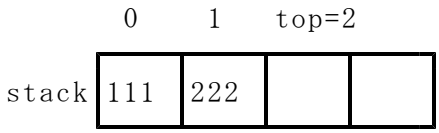
１．１　配列によるスタックの実現

配列での実現：変数topでスタックに保存されるデータの位置を示す。
変数topの初期値は、0とする。
データを保存する場合、変数topが指す位置にデータが保存され、変数topの値を1増やす。
データを取り出す場合、変数topの値を1減らし、変数topの指す位置からデータが取り出される。
変数topの値は、要素の数でもある。



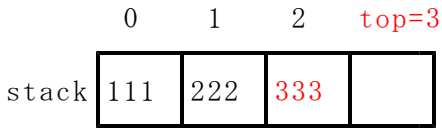
●スタック（配列表現）へ要素の追加

追加前



追加データ：333

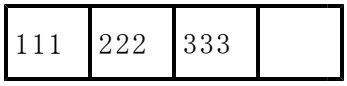
追加後




```
stack[top] = x; top++;
```

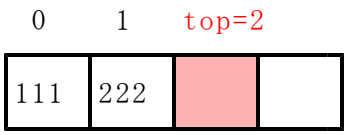
●スタック（配列表現）から要素の削除

削除前



削除データ：333

削除後




```
top--;
```

●プログラム
スタックの操作を行うプログラム
操作1：正整数を読み込みスタックに追加。
操作0：スタックからデータを取り出す。
操作-1：終了。

```
1  /* << d111.c >> */
2  #include <stdio.h>
3  #define SMAX 2 /* SMAXはスタックに保存できるデータ数。*/
4  main() {
5    int i,op,stack[SMAX],top,x;
6    /* スタックの初期化。*/
7    top = 0;
8    /* 操作。*/
9    while( 1 ) {
10     /* スタックの表示。*/
11     printf("スタック：");
12     for( i=0; i<top; i++ ) { printf("%d ",stack[i]); }
13     printf("¥n");
14     /* 操作入力。*/
15     printf("操作："); scanf("%d",&op);
16     switch( op ) {
17       case 1: /* 追加データを読み込む。*/
18           printf("データ:"); scanf("%d",&x);
19           /* スタックにデータを追加。*/
20           if( top < SMAX ) {
21             stack[top] = x; top++;
22           } else {
23             printf("スタックが一杯です¥n");
24           }
25           break;
26       case 0: /* スタックからデータを削除。*/
27           if( top > 0 ) {
28             top--;
29           } else {
30             printf("スタックが空です¥n");
31           }
32           break;
33       case -1: exit(0);
34     }
35   }
36 }
```

実行結果

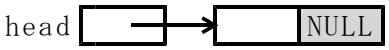
```
% cc d111.c
% a.out
スタック :
操作：1
データ:111
スタック：111
操作：1
データ:222
スタック：111 222
操作：1
データ:333
スタックが一杯です
スタック：111 222
操作：0
スタック：111
操作：0
スタック :
操作：0
スタックが空です
スタック :
操作：-1
```

１．２　再帰的なデータ構造によるスタックの実現

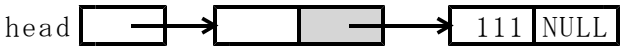
構造体を用いた再帰的なデータ構造（構造体のメンバに他の構造体を指すポインタを含む）で実現する。headは、スタックの先頭を指す。
NULLはリストの最後を意味する。

●スタック操作

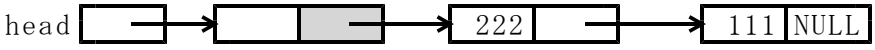
最初



111追加



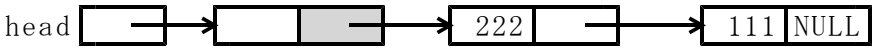
222追加



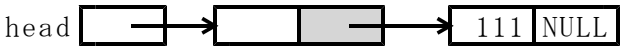
333追加

(注意)構造体が確保できる間、追加操作が可能。

333削除

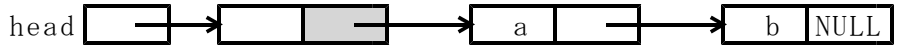


222削除

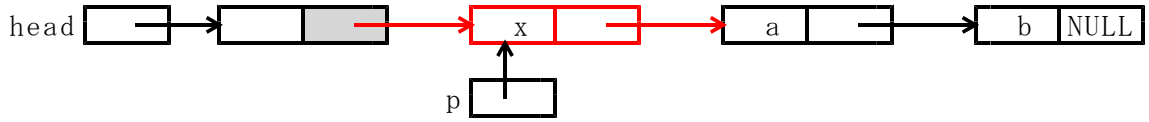


●スタック（再帰的表現）へ要素の追加

追加前



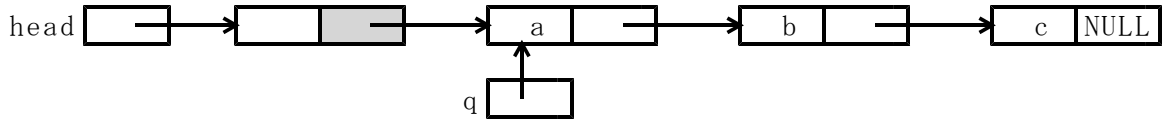
追加後　追加データ：x




```
p = (struct NODE *)malloc(sizeof(struct NODE));
p->info = x; p->next = head->next;
head->next = p;
```

●スタック（再帰的表現）から要素の削除

削除前



削除後


```
q = head->next;
head->next = q->next;
```

●プログラム
スタックの操作を行うプログラム
操作1：正整数を読み込みスタックに追加。
操作0：スタックからデータを取り出す。
操作-1：終了。

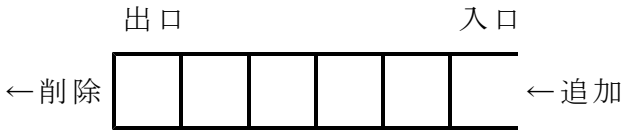
```
 1  /* << d121.c >> */
 2  #include <stdio.h>
 3  struct NODE {
 4    int info;
 5    struct NODE *next;
 6  };
 7  main() {
 8    int op,x;
 9    struct NODE *head,*p,*q;
10    /* リストヘッドの作成。*/
11    head = (struct NODE *)malloc(sizeof(struct NODE));
12    head->next = NULL;
13    while( 1 ) {
14      /* スタックの表示。*/
15      printf("スタック：");
16      p = head->next;
17      while( p != NULL ) { printf("%d ",p->info); p = p->next; }
18      printf("¥n");
19      /* 操作入力。*/
20      printf("操作："); scanf("%d",&op);
21      switch( op ) {
22        case 1: /* 追加データを読み込む。*/
23            printf("データ:"); scanf("%d",&x);
24            /* スタックにデータを追加。*/
25            p = (struct NODE *)malloc(sizeof(struct NODE));
26            p->info = x; p->next = head->next;
27            head->next = p;
28            break;
29         case 0: /* スタックからデータを削除。*/
30            if( head->next != NULL ) {
31              q = head->next;
32              head->next = q->next;
33            } else {
34              printf("スタックは空です¥n");
35            }
36            break;
37         case -1: exit(0);
38      }
39    }
40  }
```

実行結果

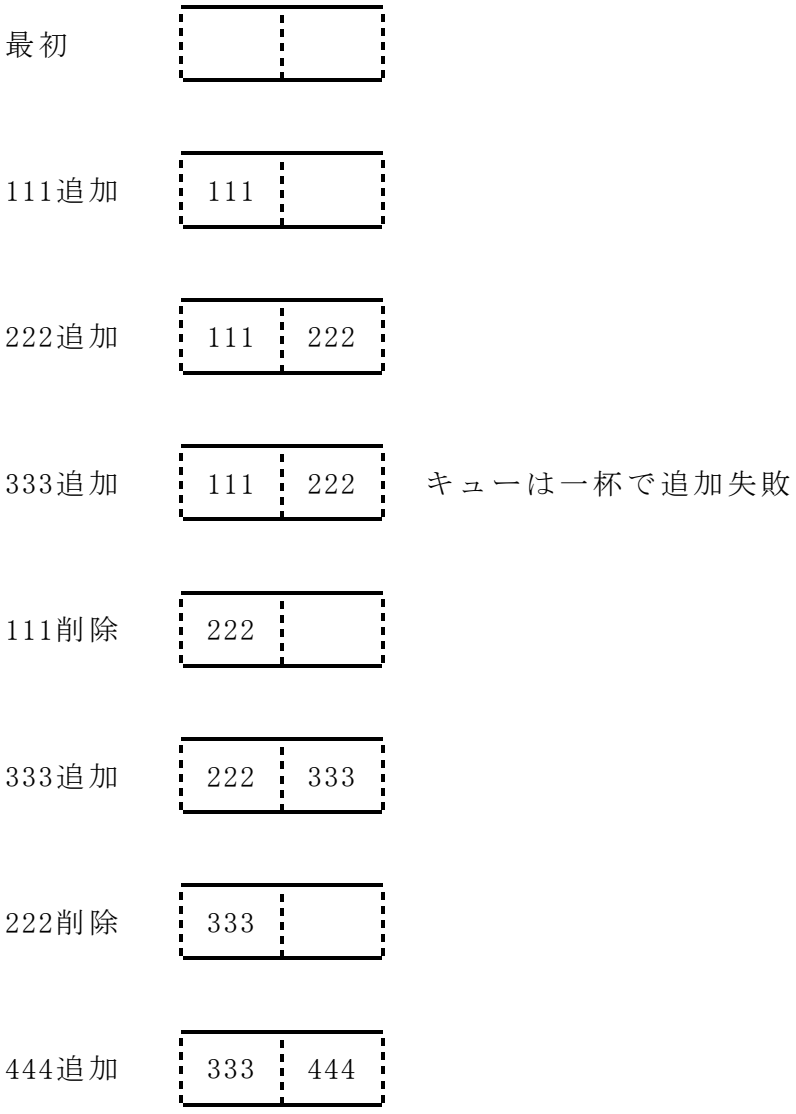
```
% cc d121.c
% a.out
スタック :
操作 : 1
データ:111
スタック : 111
操作 : 1
データ:222
スタック : 222 111
操作 : 1
データ:333
スタック : 333 222 111
操作 : 0
スタック : 222 111
操作 : 0
スタック : 111
操作 : 0
スタック :
操作 : 0
スタックは空です
スタック :
操作 : -1
```

２．キュー

キューは、データの入口と出口が一カ所（入口と出口は異なる）あり、操作は入口からの追加と出口からの削除ができるデータ構造をいう。



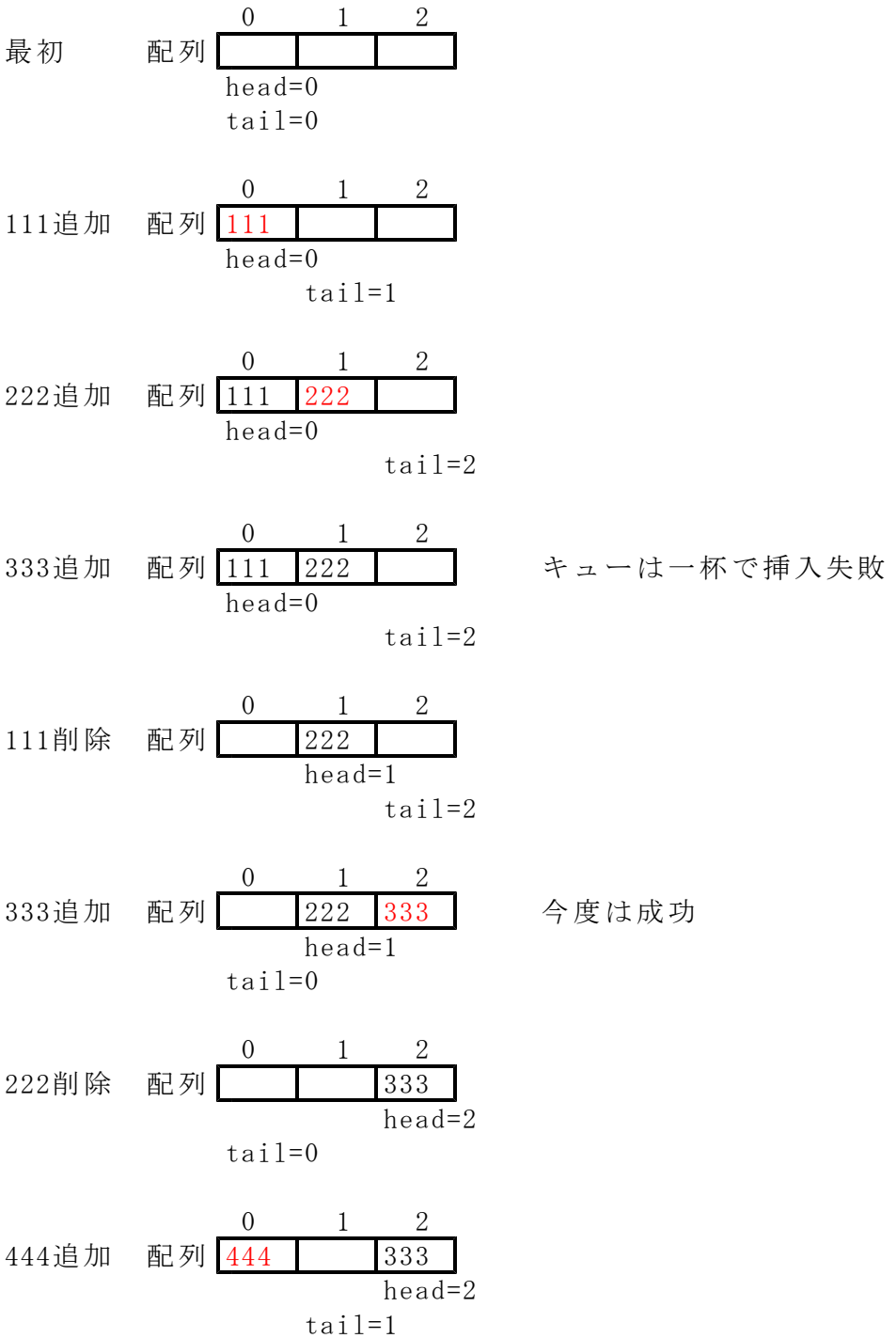
●キューの操作



２．１　配列によるキューの実現

配列で実現：変数headでキューの先頭（出口に対応）、変数tailで
キューの末尾（入口に対応）を示す。

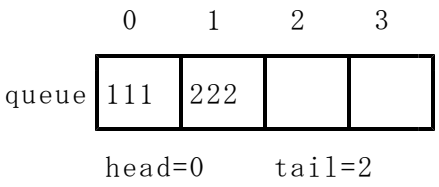
head = tail で**空のキュー**を意味し、tailを一つ後ろに更新したとき
headと一致する場合、**キューは一杯**になっているとする。



（注意）保存できるデータ数は配列の大きさ－１となる。

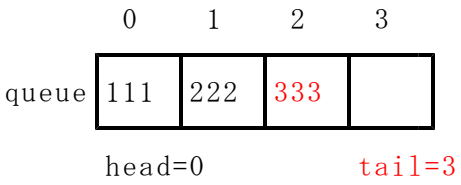
●キュー（配列）へ要素の追加

追加前



追加データ：333

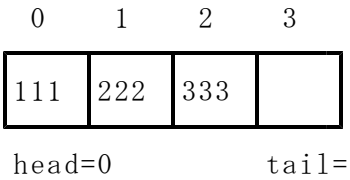
追加後




```
queue[tail] = x; tail++;
```

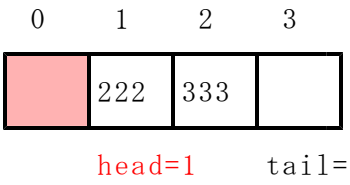
●キュー（配列）から要素の削除

削除前



削除データ：111

削除後




```
head--;
```

●プログラム
キューの操作を行うプログラム
操作1：正整数を読み込みキューに追加。
操作0：キューからデータを取り出す。
操作-1：終了。

```
1  /* << d211.c >> */
2  #include <stdio.h>
3  #define QMAX 3 /* QMAXはキューに保存できるデータ数。*/
4  main() {
5    int head,op,queue[QMAX],tail,x,w;
6    /* キューの初期化。*/
7    head = 0; tail = 0;
8    /* 操作。*/
9    while( 1 ) {
10     /* キューの表示。*/
11     printf("キュー：");
12     w = head;
13     while( w != tail ) {
14       printf("%d ",queue[w]);
15       w++; if( w >= QMAX ) { w = 0; }
16     }
17     printf("¥n");
18     /* 操作入力。*/
19     printf("操作："); scanf("%d",&op);
20     switch( op ) {
21       case 1: /* 追加データを読み込む。*/
22           printf("データ:"); scanf("%d",&x);
23           /* キューにデータを追加。*/
24           w = tail + 1; if( w >= QMAX ) { w = 0; }
25           if( w == head ) {
26             printf("キューは一杯です¥n");
27           } else {
28             queue[tail] = x; tail = w;
29           }
30           break;
31       case 0: /* キューからデータを削除。*/
32           if( head == tail ) {
33             printf("キューは空です¥n");
34           } else {
35             head++; if( head >= QMAX ) { head = 0; }
36           }
37           break;
38       case -1: exit(0);
39     }
40   }
41 }
```

実行結果

```
% cc d211.c
% a.out
キュー：
操作：1
データ:111
キュー：111
操作：1
データ:222
キュー：111 222
操作：1
データ:333
キューは一杯です
キュー：111 222
操作：0
キュー：222
操作：0
キュー：
操作：0
キューは空です
キュー：
操作：-1
```

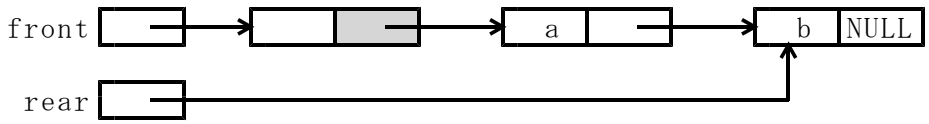
２．２　再帰的なデータ構造によるキューの実現

構造体を用いた再帰的なデータ構造（構造体のメンバに他の構造体を指すポインタを含む）で実現する。

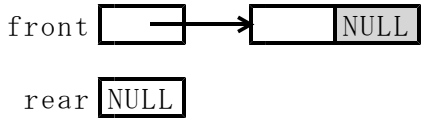
frontは、キューの出口を指す。

rearは、キューの入口を指す。

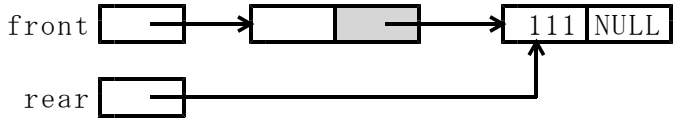
NULLはデータの最後を意味する。



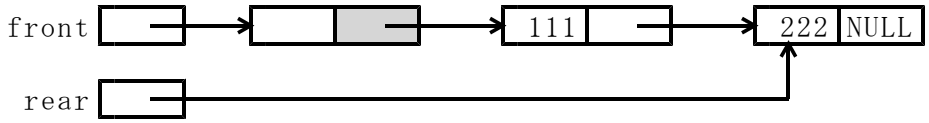
最初



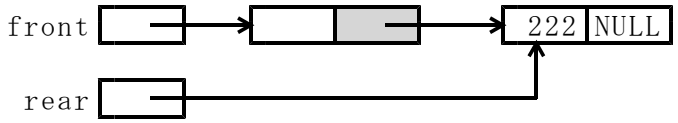
111追加



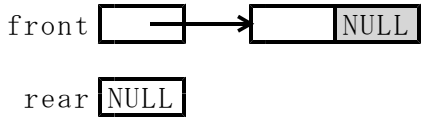
222追加



111削除

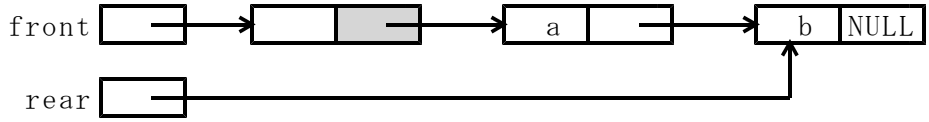


222削除

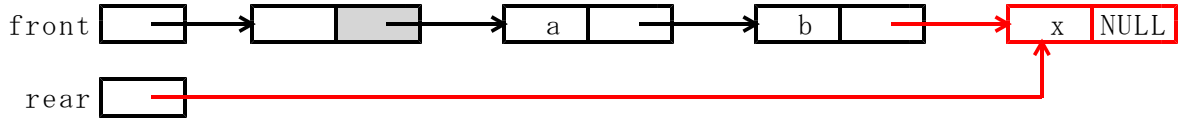


●キュー（再帰的表現）へ要素の追加

追加前



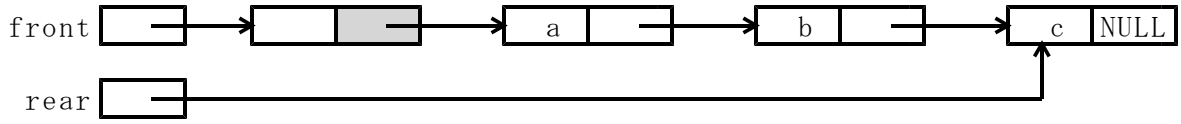
追加後　　追加データ：x



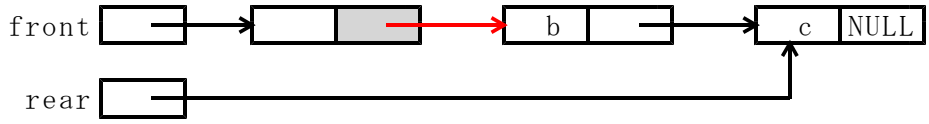

```
p = (struct NODE *)malloc(sizeof(struct NODE));
p->info = x; p->next = NULL;
if( front->next == NULL ) {
  front->next = p;
} else {
  rear->next = p;
}
rear = p;
```

●キュー（再帰的表現）から要素の削除

削除前



削除後




```
front = front->next;
```

●プログラム
キューの操作を行うプログラム
操作1：正整数を読み込みキューに追加。
操作0：キューからデータを取り出す。
操作-1：終了。

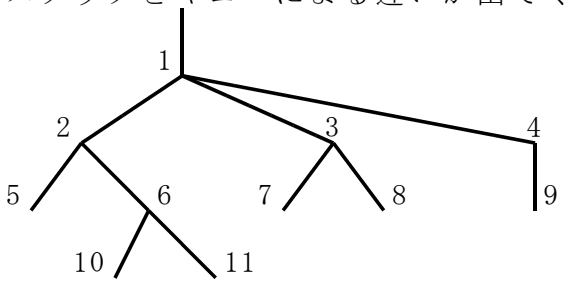
```
 1  /* << d221.c >> */
 2  #include <stdio.h>
 3  struct NODE {
 4    int info;
 5    struct NODE *next;
 6  };
 7  main() {
 8    int op,x;
 9    struct NODE *front,*rear,*p;
10    /* キューの初期化。*/
11    front = (struct NODE *)malloc(sizeof(struct NODE));
12    front->next = NULL;
13    rear = NULL;
14    /* 追加操作。*/
15    while( 1 ) {
16      /* キューの表示。*/
17      printf("キュー：");
18      p = front->next;
19      while( p != NULL ) { printf("%d ",p->info); p = p->next; }
20      printf("¥n");
21      /* 操作入力。*/
22      printf("操作："); scanf("%d",&op);
23      switch( op ) {
24        case 1: /* 追加データを読み込む。*/
25            printf("データ:"); scanf("%d",&x);
26            /* キューにデータを追加。*/
27            p = (struct NODE *)malloc(sizeof(struct NODE));
28            p->info = x; p->next = NULL;
29            if( front->next == NULL ) {
30              front->next = p;
31            } else {
32              rear->next = p;
33            }
34            rear = p;
35            break;
36        case 0: /* キューからデータを削除。*/
37            if( front->next != NULL ) {
38              front = front->next;
39            } else {
40              printf("キューは空です¥n");
41            }
42            break;
43        case -1: exit(0);
44      }
45    }
46  }
```

実行結果

```
% cc d221.c
% a.out
キュー：
操作：1
データ:111
キュー：111
操作：1
データ:222
キュー：111 222
操作：1
データ:333
キュー：111 222 333
操作：0
キュー：222 333
操作：0
キュー：333
操作：0
キュー：
操作：0
キューは空です
キュー：
操作：-1
```

３．スタックとキューの違い

迷路をたどることを考える。枝分かれしている場所（1,2,…,10,11の番号で示す）で道を選択する場合、左から右へ道を選び、その先の番号をスタック（またはキュー）に保存する。その後、スタック（またはキュー）から番号を取りだし進む道を決める。スタックとキューによる違いが出てくる。



選択された番号	スタック	選択された番号	キュー
	1		1
1	2 3 4	1	2 3 4
4	2 3 9	2	3 4 5 6
9	2 3	3	4 5 6 7 8
3	2 7 8	4	5 6 7 8 9
8	2 7	5	6 7 8 9
7	2	6	7 8 9 10 11
2	5 6	7	8 9 10 11
6	5 10 11	8	9 10 11
11	5 10	9	10 11
10	5	10	11
5		11	