# アルゴリズムとデータ構造　授業中練習問題7

次のプログラムは「キューの実現例」である．このプログラムに関して，以下の問いに答えなさい．さらに，このプログラムを入力し，自分のパソコンでコンパイル，実行できることを確認してください． なお,プログラムの日本語部分は,英語,ローマ字に変更してかまいません.

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

typedef struct {/*--- キューを実現する構造体 ---*/
  int max;  /* キューの容量 */
  int num;  /* 現在の要素数 */
  int front;/* 先頭要素カーソル */
  int rear; /* 末尾要素カーソル */
  int *que; /* キュー本体（の先頭要素へのポインタ） */
} IntQueue;

/*--- キューの初期化 ---*/
int Initialize(IntQueue *q, int max){
  q->num = q->front = q->rear = 0;
  if ((q->que = calloc(max, sizeof(int))) == NULL) {
    q->max = 0; /* 配列の確保に失敗 */
    return -1;
  }
  q->max = max;
  return 0;
}

/*--- キューの後始末 ---*/
void Terminate(IntQueue *q){
  if (q->que != NULL) {
    free(q->que);/* 配列を解放 */
    q->max = q->num = q->front = q->rear = 0;
  }
}

/*--- キューにデータをエンキュー ---*/
int Enque(IntQueue *q, int x){
  if (q->num >= q->max)
    return -1; /* キューは満杯 */
  else {
    q->num++;
    q->que[q->rear++] = x;
    if (q->rear == q->max) q->rear = 0;
    return 0;
  }
}

/*--- キューからデータをデキュー ---*/
int Deque(IntQueue *q, int *x){
```

```
  if (q->num <= 0)/* キューは空 */
    return -1;
  else {
    q->num--;
    *x = q->que[q->front++];
    if (q->front == q->max) q->front = 0;
    return 0;
  }
}
/*--- キューからデータをピーク ---*/
int Peek(const IntQueue *q, int *x)
{
  if (q->num <= 0)
    return -1;
  *x = q->que[q->front];
  return 0;
}

/*--- キューの容量 ---*/
int Capacity(const IntQueue *q){
  return (q->max);
}

/*--- キューに蓄えられているデータ数 ---*/
int Size(const IntQueue *q){
  return (q->num);
}

/*--- 全データの表示 ---*/
void Print(const IntQueue *q){
  int i;

  for(i = 0; i < q->num; i++)
    printf("%d ", q->que[(i + q->front) % q->max]);
  putchar('¥n');
}

int main(void){
  IntQueue  que;

  if (Initialize(&que, 7) == -1) {
    puts("キューの生成に失敗しました。");
    return 1;
  }
  while (1) {
    int  m, x;

    printf("現在のデータ数：%d/%d¥n", Size(&que), Capacity(&que));
    printf("(1) エンキュー (2) デキュー (3) ピーク (4) 表示 (0) 終了：");
    scanf("%d", &m);
```

①

```
    if (m == 0) break;

    switch(m){
    case 1: printf("データ：");   scanf("%d", &x);
      if (Enque(&que, x) == -1)
        puts("¥a エラー:データのエンキューに失敗しました。");
      break;
    case 2:
      if (Deque(&que, &x) == -1)
        puts("¥a エラー:デキューに失敗しました。");
      else
        printf("デキューしたデータは%d です。¥n", x);
      break;
    case 3: /* ピーク */
      if (Peek(&que, &x) == -1)
        puts("¥a エラー：ピークに失敗しました。");
      else
        printf("ピークしたデータは%d です。¥n", x);
      break;
    case 4: /* 表示 */
      Print(&que);
      break;
    }
  }
  Terminate(&que);
  return 0;
}
```

1）このプログラムを動作させ，キューに「47」,「8」,「11」,「24」の順で値をエンキューした後，下線部①の部分で入力待ちとなりました．このとき，次の問に答えなさい．

(ア) この状態で，キューから連続して何回デキューできますか．

4 回

(イ) この状態で，キューから連続して何回エンキューできますか．

3 回

(ウ) この状態での que.max, que.front, que.rear, que.que[2]の値を書きなさい．

que.max： 7

que.front：0

que.rear：4

que.que[2]：11

２）スタックに保存するデータ(プッシュ, ポップするデータ)が文字列となっている授業中練習問題６プログラムを参考に, このプログラムのキューに保存するデータ(エンキュー, デキューするデータ)を文字列に変更しなさい. ただし, 授業中練習問題6のプログラムで文字列のスタックとして StringsStack 型を定義したように, このプログラムの変更でも, 文字列のキューとして StringsQueue 型を定義して利用すること. なお, キューに保存できる文字列は, 動的な文字列として実現すること. ただし, main で入力できる文字列の長さは 80 文字以内とするここでは, 変更したプログラム中の StringsQueue 型の定義, Enque 関数と Deque 関数を答えなさい.

```
#include  <stdio.h>
#include  <string.h>
#include  <stdlib.h>
#define   String_Max  80

typedef struct {/*--- 文字列型キューを実現する構造体 ---*/
  int max;  /* キューの容量 */
  int num;  /* 現在の要素数 */
  int front;/* 先頭要素カーソル */
  int rear; /* 末尾要素カーソル */
  char **que; /* キュー本体(char* へのポインタ配列 )*/
} StringsQueue;

/*--- キューの初期化 ---*/
int Initialize(StringsQueue *q, int max){
  q->num = q->front = q->rear = 0;
  if ((q->que = calloc(max, sizeof(char *))) == NULL) {
    q->max = 0; /* 配列の確保に失敗 */
    return -1;
  }
  q->max = max;
  return 0;
}

/*--- キューの後始末 ---*/
void Terminate(StringsQueue *q){
  if (q->que != NULL) {
    while( q->num > 0){
      q->num--;
      free(q->que[q->front++]); /* 動的な文字列保存用配列を解放 */
      if (q->front == q->max) q->front = 0;
    }
    free(q->que);
  }
  q->max = q->num = q->front = q->rear = 0;
}

/*--- キューにデータをエンキュー ---*/
int Enque(StringsQueue *q, char *x){
  if (q->num >= q->max) /* キューは満杯 */
    return -1;
  if ((q->que[q->rear] = calloc(strlen(x)+1, sizeof(char))) == NULL)
```

```
    /* データをコピーするための動的な文字列保存用配列を確保することに失敗 */
    return -1;
  q->num++;
  strcpy(q->que[q->rear++], x);
  if (q->rear == q->max) q->rear = 0;
  return 0;
}

/*--- キューからデータをデキュー ---*/
int Deque(StringsQueue *q, char *x){
  if (q->num <= 0)/* キューは空 */
    return -1;
  q->num--;
  strcpy(x, q->que[q->front]);
  free(q->que[q->front++]);
  if (q->front == q->max) q->front = 0;
  return 0;
}
/*--- キューからデータをピーク ---*/
int Peek(const StringsQueue *q, char *x)
{
  if (q->num <= 0)
    return -1;
  strcpy(x, q->que[q->front]);
  return 0;
}

/*--- キューの容量 ---*/
int Capacity(const StringsQueue *q){
  return (q->max);
}

/*--- キューに蓄えられているデータ数 ---*/
int Size(const StringsQueue *q){
  return (q->num);
}

/*--- 全データの表示 ---*/
void Print(const StringsQueue *q){
  int i;

  for(i = 0; i < q->num; i++)
    printf("%s¥n",q->que[((i + q->front)% q->max)]);
}

int main(void){
  StringsQueue  que;

  if (Initialize(&que, 8) == -1) {
    puts("キューの生成に失敗しました。");
    return 1;
```

```
  }

  while (1) {
    int  m;
    char x[String_Max];

    printf("現在のデータ数：%d/%d¥n", Size(&que), Capacity(&que));
    printf("(1) エンキュー (2) デキュー (3) ピーク (4) 表示 (0) 終了：");
    scanf("%d", &m);

    if (m == 0) break;

    switch(m){
    case 1: printf("データ：");
      scanf("%s", x);
      if (Enque(&que, x) == -1)
        puts("¥a エラー:データのエンキューに失敗しました。");
      break;
    case 2:
      if (Deque(&que, x) == -1)
        puts("¥a エラー:デキューに失敗しました。");
      else
        printf("デキューしたデータは%s¥n", x);
      break;
    case 3: /* ピーク */
      if (Peek(&que, x) == -1)
        puts("¥a エラー：ピークに失敗しました。");
      else
        printf("ピークしたデータは%s¥n", x);
      break;
    case 4: /* 表示 */
      Print(&que);
      break;
    }
  }
  Terminate(&que);
  return (0);
}
```