スタックとキュー

スタックやキューとは，データを一時的に蓄える際に利用するデータ構造の一種である．今回はこれらデータ構造の概念とその操作方法について学ぶ．

スタック

スタックに対してのデータの出し入れは後入れ先出し（LIFO / Last In First Out）と呼ばれ，図１に表されるように，データを蓄える場合には積み上げるように行われ，蓄えられたデータを取り出す場合には，最後に蓄えたデータから行われる．

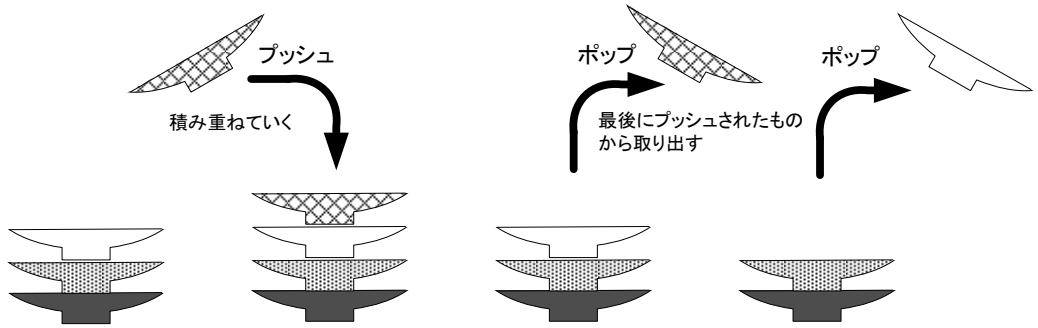


図１　スタックに対する操作

これらのスタックの操作には以下のような名称がついている．

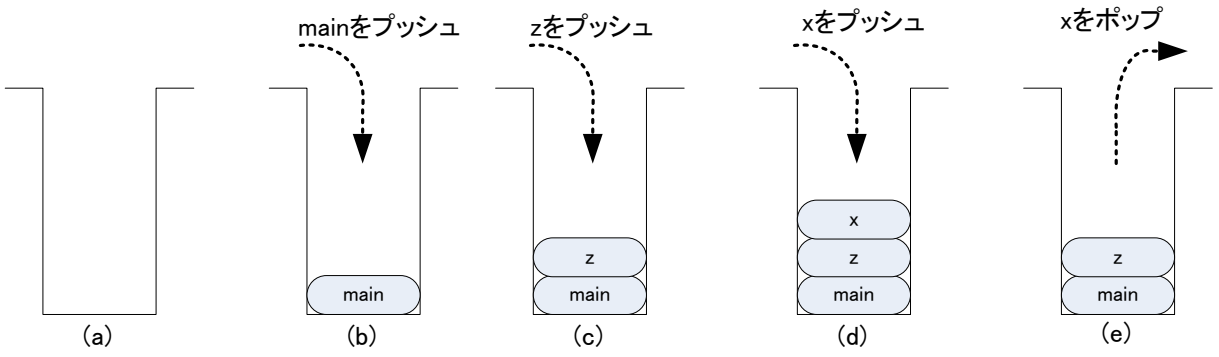
プッシュ：スタックにデータを追加する操作
ポップ　：スタックからデータを取り出す操作

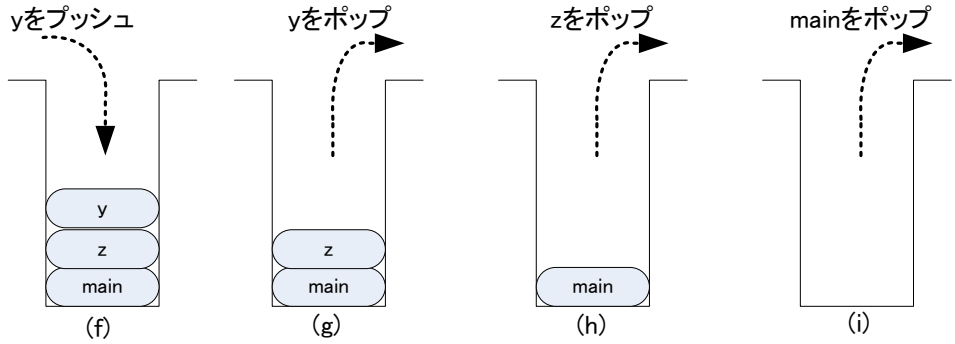
また，スタックのデータ構造においてプッシュ，ポップが行われる側を頂上（top），その反対側を底（bottom）と呼ぶ．図２にスタックの性質をよく表す事例として，関数の呼び出しと実行の手順に関する様子を示す．

```
void x( ) { /* */ }

void y( ) { /* */ }

void z( ) {
    x( );
    y( );
 }

int main( ) {
    z( );
}
```

(a) main関数の実行開始前
(b) main関数の呼び出し
(c) 関数zの呼び出し
(d) 関数xの呼び出し
(e) 関数xが終了して関数zに戻る
(f) 関数yの呼び出し
(g) 関数yが終了して関数zに戻る
(h) 関数zが終了してmain関数に戻る
(i) main関数の終了

(a) サンプルプログラムと実行の手順





(b)スタックの実行手順

図２　関数呼び出しとスタック

スタックの実現

格納するデータが int 型の整数値であるとして，スタックを実現してみる．ここでは図３に示す構造体を利用する．

```
/* スタックを実現する構造体 */
typedef struct{
    int max;        /* スタックのサイズ */
    int ptr;        /* スタックのポインタ */
    int *stk;       /* スタックの先頭要素へのポインタ */
} IntStack;
```

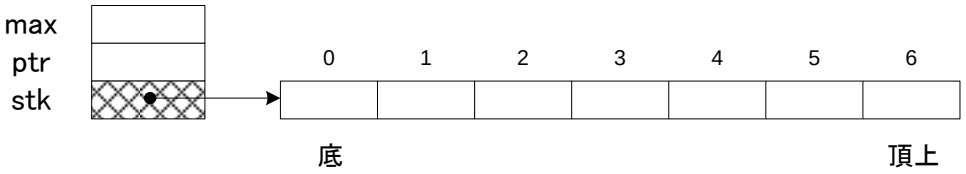


図３　スタックの実現例

スタックの本体は配列で実現する．プログラムにする場合には実行時に配列の要素数を決定し，スタックの配列領域を動的に確保すれば，柔軟に利用することが出来る．スタックの本体では配列の先頭要素 stk[0]を “底”として利用する．以下にスタックの構造体メンバの説明と操作方法を示す．

◆設定

```
max  : スタックに保持し得る最大のデータ数
ptr  : 現在スタック中に積まれているデータ数
       この値をスタックポインタという．
stk  : スタックの配列領域への先頭ポインタ
```

◆操作方法

・データがスタックにプッシュされた場合には ptr をインクリメント
・データがスタックからポップされた場合には ptr をデクリメント
・プッシュ・ポップに伴い，データの追加・削除を行う

操作に伴うスタックポインタの変化を図４に示す．この例ではスタックに保持し得る最大のデータ数 max を 12

としている．

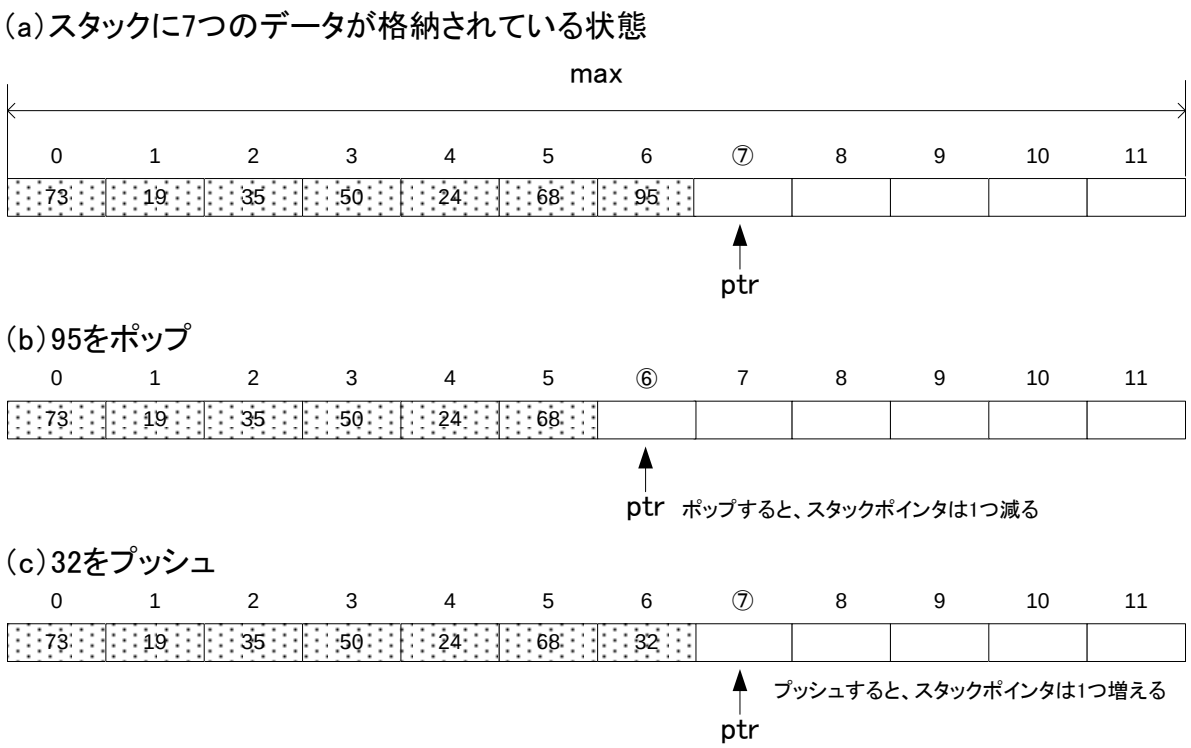


図４　スタックポインタの変化

キュー

キューに対してのデータの出し入れは先入れ先出し（FIFO / First In First Out）と呼ばれる．図５に表されるように，データを追加する場合にはデータの最後側に追加され，データを取り出す場合には，最初に入れられたデータから行われる．

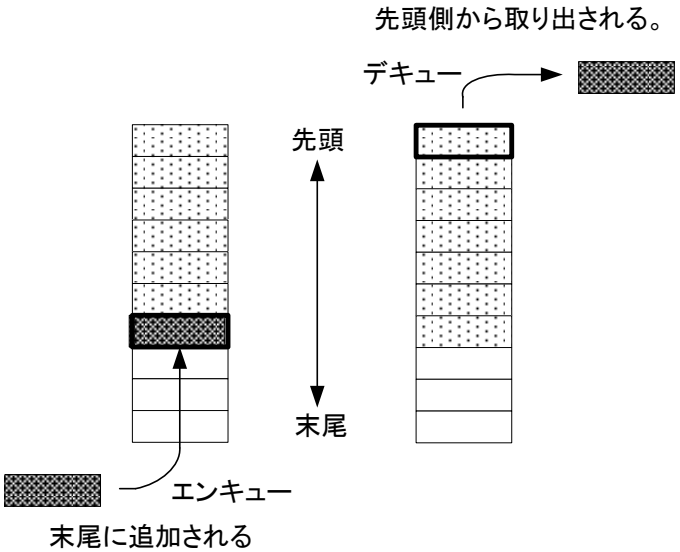


図５　キュー

これらのキューの操作には以下のような名称がついている．

エンキュー：キューにデータを追加する操作
デキュー　：キューからデータを取り出す操作

また，スタックのデータ構造においてデータの取り出しが行われる側を先頭（front），データの追加が行われる側を末尾（rear）と呼ぶ．図６に配列によって実現されるキューに対する操作についての例とその問題点を示す．図の（a）では，配列に先頭側から{ 73，19，35，50，24，68，95 }というデータが格納されている．ここに 82 を新規のデータとして，エンキューする．これは単純な操作で済む．さて，このキューのデータからデキューすることを考える．この場合，図の（b）に示されるような状態から，73 を取り出すわけだが，取り出したデータはキューから削除しなくてはならず，削除後はすべての要素を先頭側に一つずつずらさなくてはならない．このように，配列でキュー構造を実現しようとした場合，デキューを行うたびに，多くの計算コストを支払わなくてはならない．

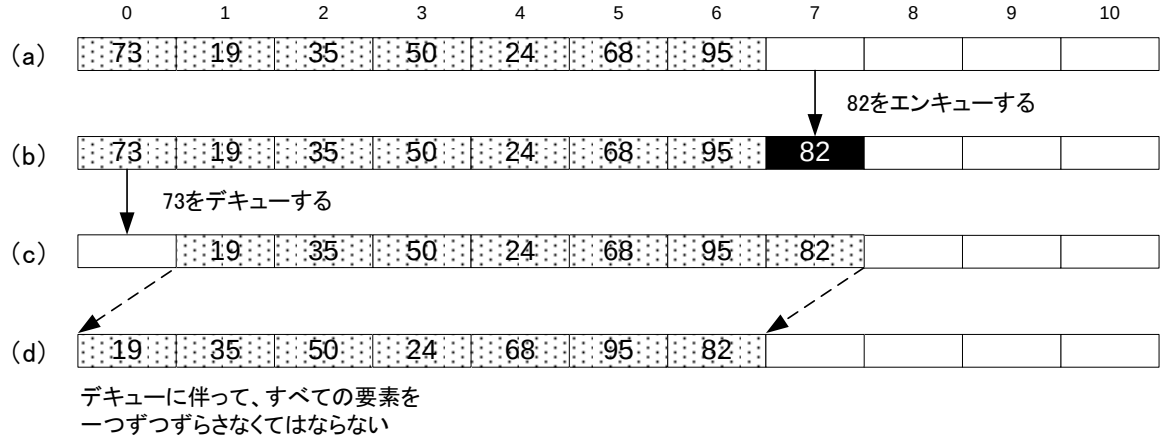


図６　配列によるキューの実現例

リングバッファによるキューの実現

配列によるキューの実現では配列内のデータをずらす操作が含まれてしまう．そこでこの問題を解決するために用いられるリングバッファ（ring buffer）と呼ばれる手法について学んでいく．格納するデータが int 型の整数値であるとして，リングバッファによるキューを実現してみる．ここでは図７に示す構造体を利用するものとする．

```
/* キューを実現する構造体 */
typedef struct{
    int max;        /* キューのサイズ */
    int num;        /* 現在の要素数 */
    int front;      /* 先頭要素のカーソル */
    int rear;       /* 末尾要素のカーソル */
    int *que;       /* キューの先頭要素へのポインタ */
} IntQueue;
```

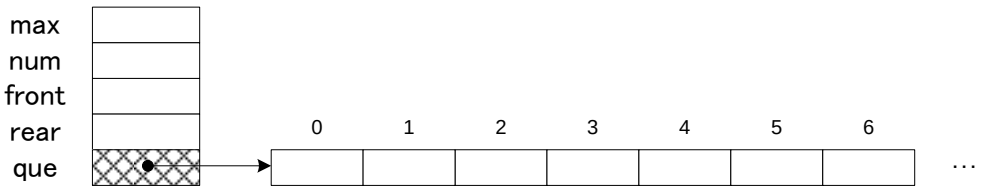


図７　キューの実現例

以下にスタックの構造体メンバの説明と操作方法を示す．

◆設定

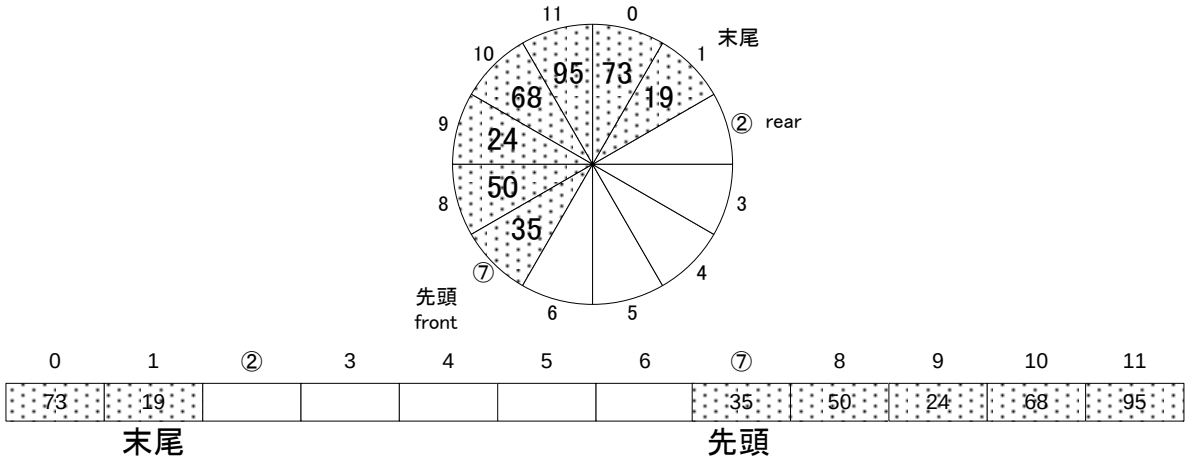
max　：キューに保持し得る最大のデータ数 num　：現在キューに格納されているデータ数 front ：キューの先頭要素の添字 rear　：キューの末尾要素の１つ後ろの添字 　　（次にエンキューされるデータが格納される要素の添え字） que　：キューの配列領域への先頭ポインタ 注）front や rear の変数はエンキューやデキューが行おうとする添字を記憶していると考えると良い

◆操作方法

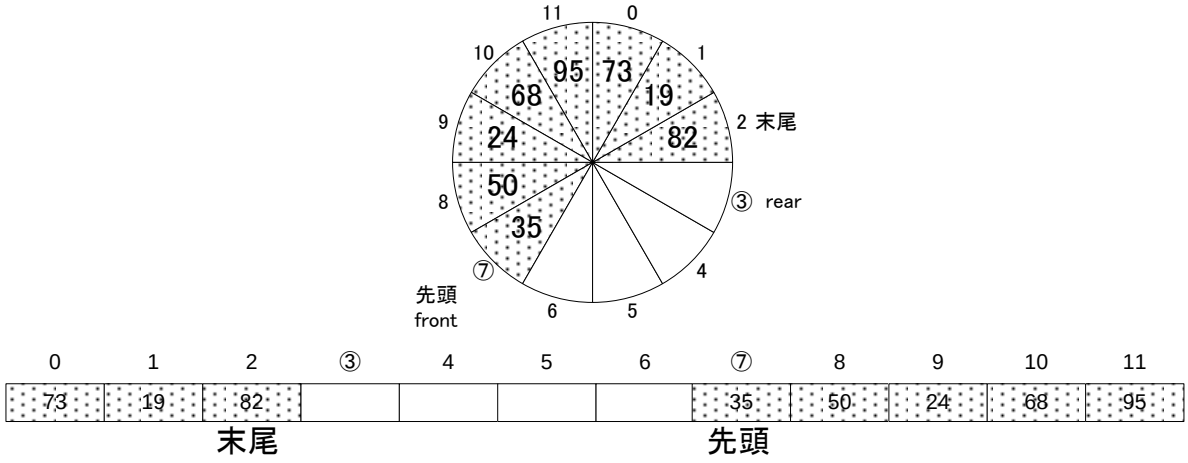
- 初期化は num と添字 front，rear の初期値は 0 に設定し，num に値を設定，キュー本体を calloc などで確保
- データがキューにエンキューされた場合には rear をインクリメント
- データがキューからデキューされた場合には front をインクリメント
- 添字 front 又は rear の値が max になった場合には 0 に設定（リングの実現）
- エンキュー・デキューに伴い，データの追加・削除を行う
- データ数に応じて num を変更

リングバッファを用いたキューの実現における操作の様子を図８に示す．この例ではキューに保持し得る最大のデータ数 max を 12 としている．図の（a）では，7 つのデータ{ 35，50，24，68，95，73，19 }がすでに格納されている．配列の状態を見ると，que[0]側と que[max-1]側でデータが分かれて格納されている．ここに 82 を新規のデータとして，エンキューする．データは末尾に追加されるので，図の（b）に示されるように que[2]にデータが格納され，rear の値はインクリメントされる．次に図の（b）の状態から，デキューを行い 35 を取り出す．図の（c）に示されるように，que[7]から 35 を取り出し，front の値はインクリメントされる．このように，先頭と末尾の 1 つ後ろを示す添字を変数として記憶し，インクリメントすることにより，全体の要素をずらすことなくキューを配列で実現することが可能となる．

(a)ある状態



(b)82をエンキュー



(c)35をデキュー

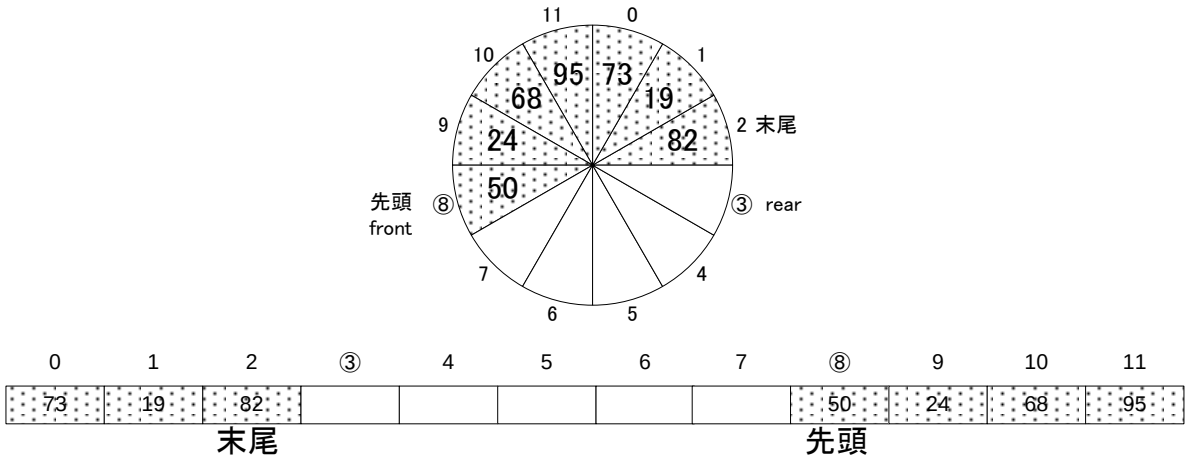


図８　リングバッファによるキューの実現

課題２

（１） 教科書のスタック，もしくはキューのプログラムを参考にオリジナルのプログラムを作成しなさい.

（２） スタックとキューについてまとめなさい.