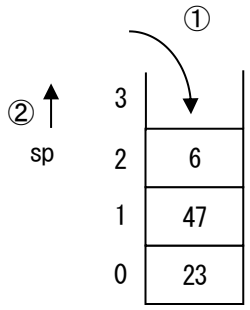
１．スタック（Stack）

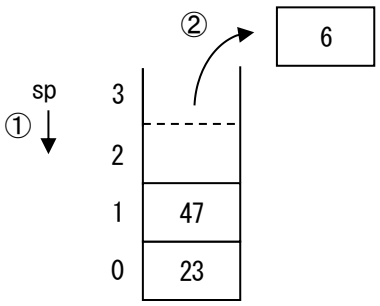
スタックは、最初に格納したデータが最後に取り出されるというデータ構造です（Last In, First Out、略して LIFO）。データの一時退避などで使われます。例えば Java であるメソッドを呼び出すとそのメソッドの処理が終わったら呼び出し元へ戻ってきますが、この戻ってくるアドレスを覚えておくような場合にスタックが使われます。

データ格納（プッシュ）



データ格納時は
①スタックポインタ（sp）の指す場所にデータを格納し、
②sp を+1 する。

データ取り出し（ポップ）

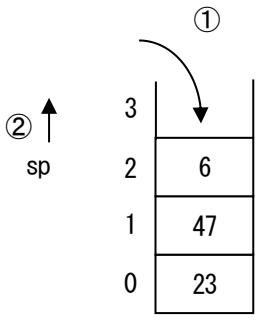


データ取り出し時は
①sp を−1 してから
②データを取り出す。

２．キュー（Queue、データシフト方式）

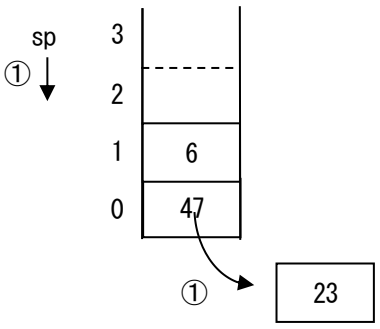
キューは、最初に格納したデータが最初に取り出されるというデータ構造です（First In, First Out、略して FIFO）。非同期処理をするプログラム間でデータを受け渡す場合などで使われます。代表的なものに Windows のメッセージキューがあります。ゲームではゲームプログラムとサウンドプログラムでデータのやり取りをする場合などに使われます。待ち行列やリングバッファとも呼ばれます。

データ格納（プッシュ）



データ格納時は
①sp の指す場所にデータを格納し、
②sp を+1 する。

データ読み出し（ポップ）



データ読み出し時は
①0 番からデータを取り出し、
②sp を−1 すると同時に残っているデータをずらす。

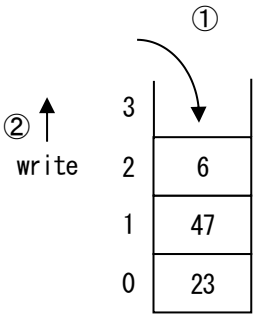
３．キュー（リングバッファ）

配列の先頭データを取り出して残りのデータをひとつずつシフトさせると、かなりのオーバーヘッドが発生します。配列の最後と先頭がつながっているように扱うリングバッファを使えば、この問題が解決します。

リングバッファの実装①

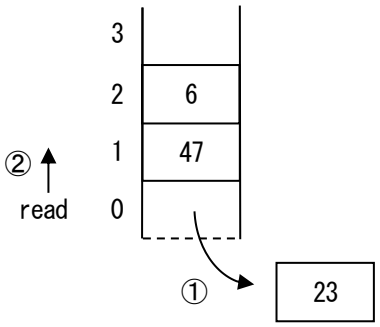
データ書き込み場所を表す write とデータ読み出し場所を表す read を使って（read が write を追いかけるように）作成します。データの書き込みや読み出しを行うと 1 進み、配列の最後まで進むと先頭へ戻ります。また、write が read を追い越さないように間をひとつ空けるため、格納するデータ数+1 のバッファサイズが必要となります。

データ格納



データ格納時は
①書き込み位置（write）の指す場所に格納し、
②write を+1 する。write がバッファサイズを越えたときは 0 にする。

データ読み出し



データ読み出し時は
①読み出し位置（read）の指す場所から読み出し、
②read を+1 する。read がバッファサイズを越えたときは 0 にする。

リングバッファの実装②

データの先頭位置を表す head とデータ数を表す count を使って作成します（head は**実装①**の read に該当）。データの書き込みは head＋count の場所に行います。書き込みを行うと count を 1 増やします。読み出しは head の場所を読み出し、head を 1 進め count を 1 減らします。先頭のデータ（head）と最後のデータ（head＋count）の間に空席を設けなくても動作します。