スタックとキュー

山本昌志*

2005 年 12 月 12 日

1　本日の学習内容

1.1　前回の復習

前回は，リストというデータ構造を学習した．イメージは，図 1 のようなものであった．配列とは異なり，ランダムアクセスはできないが，要素の追加や削除が容易なデータ構造である．

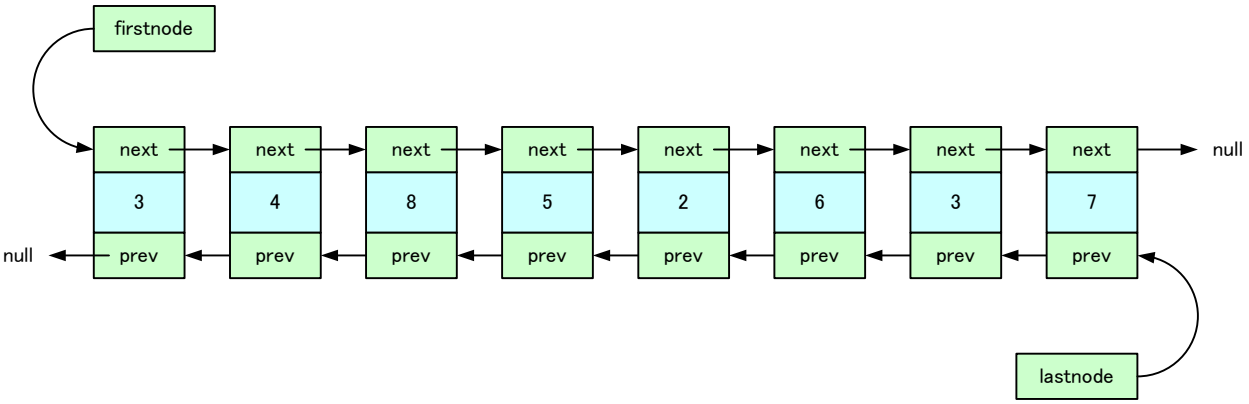

図 1: リスト

1.2　本日の学習内容

スタックとキュー呼ばれるデータ構造を学習する．本日の授業のゴールは，

- スタックとキューのデータ構造のイメージがつかめる．イメージの図が書ける．
- スタックとキューの操作の仕方が分かる．
- スタックとキューの実装方法が理解できる．

*独立行政法人　秋田工業高等専門学校　電気情報工学科

である．

2　スタック

2.1　イメージ

スタック (stack) を辞書で調べてみると，次のような意味が書かれている．

1. (干し草などの) 大きな山，積みわら (haystack);(物のきちんとした) 積み重ね
2. (図書館などの) 書棚の列，書架；((the 〜s)) (図書館の)(閉架) 書庫
3. (屋上の) 組み合わせ煙突 (chimney stack); 煙突，煙出し
4. 《コンピューター》スタック:最後に入れたデータを最初に取り出せるようにしたデータ構造

もちろん，情報科学の分野で使われるのは最後の意味で，図 2 のようなデータ構造である．データ構造であるから，データを蓄えることと，それを取り出すことができる．スタックの特徴は，最後に入れたデータが一番最初に取り出されることにある．取り出されるデータは，格納されている最新のデータで，最後に入れられたものが最初に取り出されることから，LIFO(last in first out, 後入れ先出し) と呼ばれる．スタックの途中のデータを取り出すことは許されないのである．

スタックにデータを積むことをプッシュ(push) と，スタックからデータを取り出すことをポップ (pup) と呼ぶ．これらの英語の意味は，

push ＜人・物を＞押す，突く

pop ポンという音を立てる，ひょいとやって来る [出て行く]，急にはいる [出る], ひょっこり現れる

である．

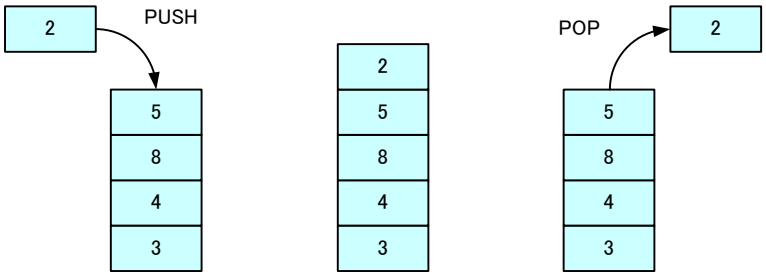

図 2: スタック

2.2　実際の応用

スタックは単純なデータ構造であるが，有用でいろいろな場面で使われる．例えば，次のような場合である．

- 関数を呼び出した場合，呼び出し元のデータをいったん保存するときのデータ構造として使われる．
- 算術式の評価 (教科書の List 4-3)

教科書では，カッコの対応を調べるプログラム (p.104-108 List 4-2) や逆ポーランド記法を用いた数式の計算 (p.114-115 List 4-3) が示されている．授業では説明しないので，興味のある者は自分でソースコードを解析するのが良いだろう．

2.3 スタックの実装

スタックを実装する関数をリスト 1 に示す．これは，教科書の List 4-1(p.100) と同じである．

スタックを実装するために必要な記憶領域は，スタックそのものの記憶領域とスタックの頂上を表す記憶領域である．スタックのための記憶領域として配列 `stack[]` を使っている．また，スタックの頂上を表すために，変数 `stack_top` を使っている[1]．

スタックのために必要な記憶領域が確保されたならば，それを操作しなくてはならない．スタックに必要な操作は 2 つで，データを積む (push) ことと，データを取り出す (pop) ことである．リスト 1 では，次の 2 つの関数でそれを行っている．

`void stack_push(double val)` スタックへデータ `val` をプッシュする関数である．データを積み重ねたならば，スタックの頂上を表す変数 `stack_top` を 1 加算している．スタックがいっぱいで，さらにデータをプッシュしようとするとプログラムは終了するようになっている．

`double stack_pop(void)` スタックからデータを取り出す関数で，戻り値が取り出されたデータである．データを取り出したならば，スタックの頂上を表す変数 `stack_top` を 1 減算している．スタックが空の状態でデータをポップしようとするとプログラムは終了するようになっている．

ここで使われているプログラムのテクニックは，すでに学習済みであるが，分かりにくいものだけ述べておく．

- 15 行目 `exit(EXIT_FAILURE);`
  関数 `exit()` は，正常にプログラムを終了させるために使う．これが呼び出されると，諸々の処理を行いプログラムが止まる．ここで使われている引数 `EXIT_FAILURE` の定義は `stdlib.h` に書かれている．私のコンパイラーでは，
  ```
  #define EXIT_FAILURE 1 /* Failing exit status.  */
  ```
  となっていた．したがって，この行は，
  ```
  exit(1);
  ```
  と同じである．`exit(EXIT_FAILURE);` とすると処理系定義の不成功状態の形式が返される．

リスト 1: スタックの実装

```
1 #define STACK_MAX 10
2
```

[1] ここでは単なる変数を使ったが，実際のプログラムではポインターが使われることが多い．そのため，頂上を表す変数をスタックポインターと呼ぶ．

```
3  /* ※この例では，double型のデータを格納するスタックを作成 */
4  double stack[STACK_MAX];
5  /* スタック頂上の位置（最下部からのオフセット） */
6  int stack_top=0;
7
8  /* スタックへpush */
9  void stack_push(double val)
10 {
11     if(stack_top==STACK_MAX)
12     {
13         /* スタックが満杯になっている */
14         printf("エラー:スタックが満杯です(Stack overflow)\n");
15         exit(EXIT_FAILURE);
16     }
17     else
18     {
19         /* 渡された値をスタックに積む */
20         stack[stack_top]=val;
21         ++stack_top;
22     }
23 }
24
25 /* スタックからデータをpop */
26 double stack_pop(void)
27 {
28     if(stack_top==0)
29     {
30         /* スタックには何もない */
31         printf("エラー:スタックが空なのにpopが呼ばれました"
32                                         "(Stack underflow)\n");
33         exit(EXIT_FAILURE);
34         return 0;
35     }
36     else
37     {
38         /* いちばん上の値を返す */
39         --stack_top;
40         return stack[stack_top];
41     }
42 }
```

3 キュー

3.1 イメージ

待ち行列と呼ばれるキュー (queue) も辞書で調べてみた．それには，次のような意味がある．

1. 順番を待つ人・車などの）列 (line)；(待つ）一群の人々
2. 《コンピューター》待ち行列

これは，窓口に並ぶ順番待ちの行列の意味で，図 3 のようなデータ構造となっている．スタックではデータの挿入と取り出しが列の一方からのみであったの対して，キューは列の両端から行う．一方がデータの追加で一方がデータの取り出しとして使われる．キューでは，最初に入れたデータが一番最初に取り出されることにある．取り出されるデータは格納されている最古のデータで，最初に入れられたものが最初に取り

出されることから，FIFO(first in first out, 先入れ先出し) と呼ばれる．スタック同様，スタックの途中のデータを取り出すことは許されないのである．

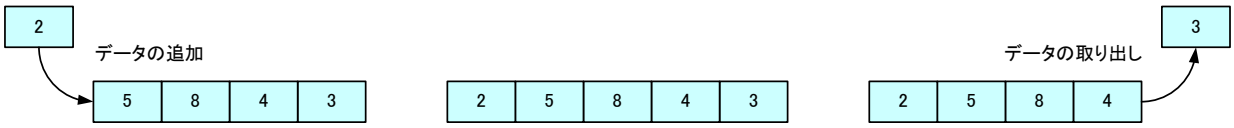

図 3: キュー

キューはスタックに比べて少しばかり，構造が複雑になっている．実際，それを直線的なイメージのメモリーにデータを追加しようとすると，以下のような問題が生じる．

- FIFO を続けると，いずれはメモリーの端に到達して，データの追加が出来なくなる．
- データを追加したり取り出したりする毎に，データの列を移動させることも考えられる．こうすると計算量が増加して不経済である．

これを防ぐために，図 4 のようなリングバッファと言うものが考えられた．

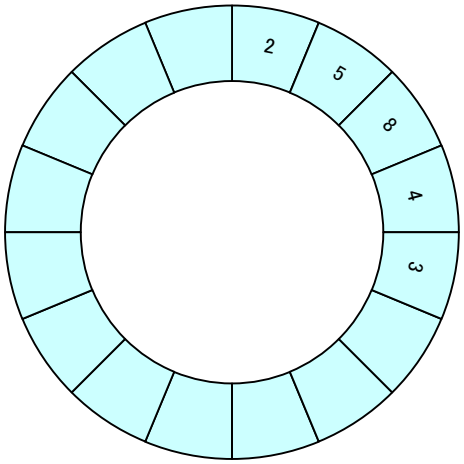

図 4: リングバッファー

3.2 実際の応用

これは，入れられた順序通りに処理すべきデータに使われる．たとえば，次のような応用がある．

- データをバッファーにためて，処理を行う場合．プリンターの処理などである．プリント待ちのデータをプリントキューと言うことがある．

- オンライントランザクション処理[2]の管理に用いれれる．この処理では，原則的には到着順に処理しなくてはならない．

教科書では，配列を用いたプリントキューのプログラム (p.123-125 List 4-5) を載せている．これも講義では説明しないので，各自，プログラムを解析せよ．

3.3 キューの実装

キューを実装する関数をリスト 2 に示す．これは，教科書の List 4-4(p.120-121) と同じである．

このプログラムでは，配列 queue[] を使ってキューを実現している．配列 queue[] とキューの先頭を表す変数 queue_first と末尾を表す queue_last はグローバル変数と宣言されているので，以下のキューを操作する関数で直にアクセスすることができる．

キューを実装するための記憶領域が確保されたならば，それを操作しなくてはならない．キューの操作は 2 つで，データを追加することと，データを取り出すことである．リスト 2 では，次の 2 つの関数でそれを行っている．

`void enqueue(int val)` キューへデータ val を追加する関数である．データを追加したならば，キューの末尾を表す変数 queue_last を 1 移動 (加算) している．キューがいっぱいで，さらにデータを追加しようとするとプログラムはメッセージを出すようになっている．

`int dequeue(void)` キューからデータを取り出す関数で，戻り値が取り出したデータである．データを取り出したならば，キューの先頭を表す変数 queue_first を 1 移動 (加算) している．キューが空の状態でデータを取り出そうとするとプログラムは QUEUE_EMPTY を返すようになっている．

ここで使われているプログラムのテクニックは，すでに学習済みであるが，分かりにくいものだけ述べておく．

- 15 行目 `(queue_last+1)%QUEUE_MAX`

  2 項演算子%は，割り算の結果の余りを返す．これをりようして，このプログラムでは，queue_last や queue_first の値を，

  $$\cdots \to 0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 0 \to 1 \to 2 \to 3 \to 4 \to 5 \to 0 \to 1 \to \cdots$$

  とサイクリック (cyclic:周期の) に変化させている．5 の次の値を 0 にしているのである．これは，サイクリックに値を変化させて対時の常套手段である．

リスト 2: キューの実装

```
1 #define QUEUE_MAX 5+1    /* キューのサイズ+1 */
2 #define QUEUE_EMPTY -1
3
4 /* 配列によるキュー構造 */
5 int queue[QUEUE_MAX];
```

[2]ネットワークに接続された複数のパソコンがホストコンピュータに処理要求を行い、ホストコンピュータがその要求にもとづいてデータを処理し、処理結果を即座にパソコンに送り返す処理方式。データベースの処理などに多く使われる．

```
6  /* キューの先頭位置（配列先頭からのオフセット） */
7  int queue_first=0;
8  /* キューの末尾位置（配列先頭からのオフセット） */
9  int queue_last=0;
10
11 /* キューにデータを追加する */
12 void enqueue(int val)
13 {
14     /* lastの次がfirstならば */
15     if((queue_last+1)%QUEUE_MAX ==queue_first)
16     {
17         /* 現在配列の中身は，すべてキューの要素で埋まっている */
18         printf("ジョブが満杯です\n");
19     }
20     else
21     {
22         /* キューに新しい値を入れる */
23         queue[queue_last]=val;
24         /* queue_lastを1つ後ろにずらす。
25            もし，いちばん後ろの場合は，先頭にもってくる */
26         queue_last=(queue_last+1)%QUEUE_MAX;
27     }
28 }
29
30 /* キューからデータを取り出す */
31 int dequeue(void)
32 {
33     int ret;
34
35     if(queue_first==queue_last)
36     {
37         /* 現在キューは1つもない */
38         return QUEUE_EMPTY;
39     }
40     else
41     {
42         /* いちばん先頭のキューを返す準備 */
43         ret=queue[queue_first];
44         /* キューの先頭を次に移動する */
45         queue_first=(queue_first+1)%QUEUE_MAX;
46         return ret;
47     }
48 }
```

4 練習問題

スタックとキューについて，以下の操作を定義する．

PUSH(n) :スタックにデータ n をプッシュする．戻り値は無し．
POP() :スタックからデータをポップする．戻り値は，取り出した値．
ENQ(n) :キューにデータ n を追加する．戻り値はなし．
DEQ() :キューからデータを取り出す．戻り値は取り出した値．

空のスタックやキューに対して，以下の操作を行ったとき，データ構造の様子を図で示せ．

[問 1] PUSH(3)→PUSH(5)→POP()→PUSH(2)→PUSH(1)→POP()→POP()→PUSH(1)→POP()→PUSH(7)

[問 2] PUSH(1)→POP()→PUSH(9)→PUSH(5)→POP()→PUSH(2)→POP()→PUSH(4)→PUSH(8)→POP()

[問 3] ENQ(6)→ENQ(2)→DEQ()→ENQ(7)→DEQ()→ENQ(3)→ENQ(1)→ENQ(2)→DEQ()→DEQ()

[問 4] ENQ(1)→DEQ()→ENQ(2)→DEQ()→ENQ(3)→DEQ()→ENQ(4)→ENQ(5)→DEQ()

[問 5] PUSH(3)→ENQ(POP())→PUSH(8)→PUSH(5)→ENQ(POP())→PUSH(DEQ())→ENQ(2)

4.1 レポート 提出要領

提出方法は、次の通りとする。

期限	12 月 19 日 (月) AM 10:40
用紙	A4
提出場所	山本研究室の入口のポスト
表紙	表紙を 1 枚つけて、以下の項目を分かりやすく記述すること。 授業科目名「情報工学」 課題名「課題　スタックとキュー」 2E　学籍番号　氏名 提出日
内容	2 ページ以降に問いに対する答えを分かりやすく記述すること.