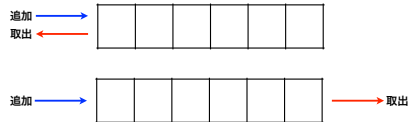


スタックと再帰関数

計算機構成の補足資料

1次元に並んだデータ構造に関する2つの操作

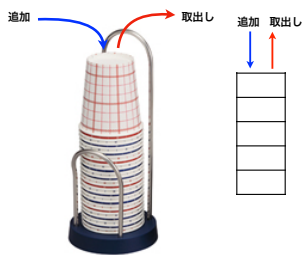
- 1次元に並んだデータ構造に対する追加と取出しの操作について、2つの考え方がある。
- 片側だけを使い、追加と取出しを行う → 「スタック」
- 両側を使い、片方が追加、もう片方が取出しを行う。 → 「キュー」



実社会でのスタックとキューの例

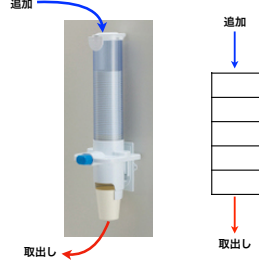
■ スタック

- ▶ 上から追加して、上から取り出す



■ キュー

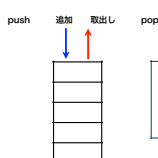
- ▶ 上から入れて、下から取り出す



データ構造としてのスタックとキュー

■ スタック

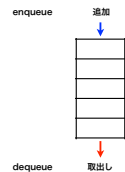
- ▶ 片側だけを使って追加・取り出し
- ▶ 上から追加して、上から取り出す
- ▶ 新しいもの（最後に追加したもの）が、最初に取り出される
- ▶ LIFO Last In First Out



	追加	取出
スタック	push	pop
キュー	enqueue	dequeue

■ キュー

- ▶ 両側を使って追加・取り出し
- ▶ 上から入れて、下から取り出す
- ▶ 古いもの（最初に追加したもの）が、最初に取り出される
- ▶ FIFO First In First Out



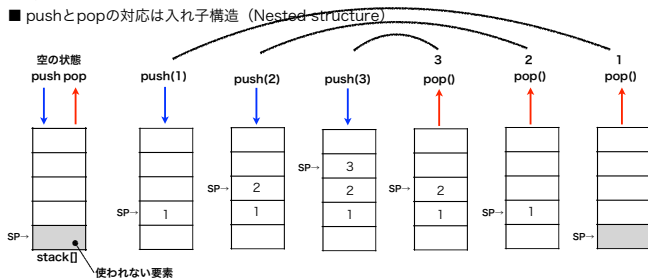
スタックとキュー

スタックとキューが混在している



スタックの実装

- `stack[]` とスタックポインタ (`sp`) を使って, `push` と `pop` 操作を実現する.
- `sp` は一番上の要素を指す.
- `push` と `pop` の対応は入れ子構造 (Nested structure)



push関数, pop関数, empty関数

■ push()関数

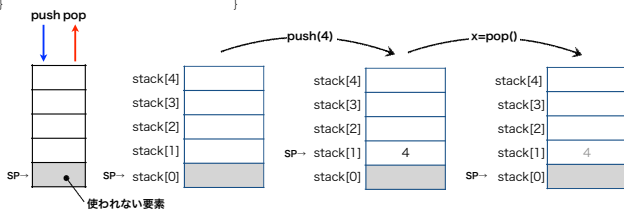
```
void push(x)
{
    if (full()) エラー処理;
    sp=sp+1;
    stack[sp]=x;
}
```

■ pop()関数

```
int pop()
{
    if (empty()) エラー処理;
    sp=sp-1;
    return stack[sp+1];
}
```

■ empty()関数

```
int empty()
{
    if (sp==0) return 1;
    return 0;
}
```



インクリメント/デクリメント演算子とpush関数/pop関数

■ push()関数

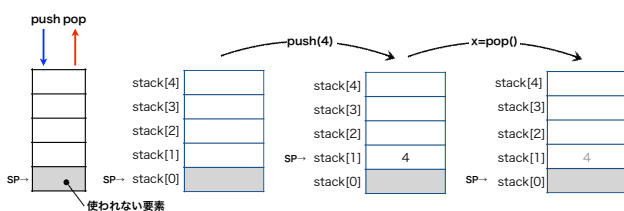
```
void push(x)
{
    if (full()) エラー処理;
    stack[++sp]=x;
}
```

■ pop()関数

```
int pop()
{
    if (empty()) エラー処理;
    return stack[sp--];
}
```

■ empty()関数

```
int empty()
{
    if (sp==0) return 1;
    return 0;
}
```



push関数とpop関数の他の実装

- spは一番上の要素を指している
 - ▶ 教科書的にはこちらが多いかな？

■ push()関数

```
void push(x)
{
    stack[++sp]=x;
}
```

■ pop()関数

```
int pop()
{
    return stack[sp--];
}
```

- spは次に格納する要素を指している
 - ▶ 配列の要素番号が0から開始するC言語向け

■ push()関数

```
void push(x)
{
    stack[sp++]=x;
}
```

■ pop()関数

```
int pop()
{
    return stack[--sp] ;
}
```

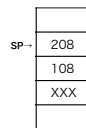
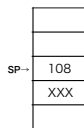
入れ子のjal命令は正しく戻れない

```
100 jal 200 $ra=104 → 200 jal 300 $ra=204 → 300 jal 400 $ra=304 → 400
104                                204                                304                                404
                                220 jr $ra                                330 jr $ra                                440 jr $ra
```

jal命令は\$raを書き換えてしまうので、jr \$raでは戻れない。
入れ子にjal命令を実行すると、\$raの値は最後に実行したjal命令が設定した\$raになる。

入れ子のjal命令とスタック

```
100 push($ra)      200 push($ra)      300 push($ra)      400 リーフ関数
104 jal 200 $ra=108 → 204 jal 300 $ra=208 → 304 jal 400 $ra=308 → 404 400番地以降、他の関数は呼び出さない
108                                208                                308                                404
                                330 $ra=pop()                                440 jr $ra
180 $ra=pop()      220 $ra=pop()      334 jr $ra $ra=208
184 jr $ra $ra=XXX 224 jr $ra $ra=108
```



スタックとスコープ

- 変数の値は、そのスコープ（有効範囲）に応じた値でなければならない。
- 変数nの値は、呼び出すと3、2、1、0と変わり、リターンすると1、2、3となる。
 - ▶ 変数の値は、スタックを使って退避・復元すれば良い。

```
int fact(int 3)      int fact(int 2)      int fact(int 1)
{
    n=3
    if( n==0 ) return 1;
    return n * f( n-1 );
}
3      2      1

int fact(int 0)
{
    n=0
    if( n==0 ) return 1;
    return n * f( n-1 );
}
```

これは、\$raと同じ問題である。
nは\$a0に割り当てられるので、\$a0もスタックを使って退避・復元するようにすれば良い。

再帰関数fact()の考え方(1)

■ Cコード

```
int fact(int n)
{
    if(n<1) return 1;
    return n * fact(n-1);
}
```

■ 次のif文でも動く。

```
if(n==1) return 1;
if(n==0) return 1;
```

■ バグあり①

```
fact:
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    beq $0,$0,fact2
fact1:
    addi $a0,$a0,-1
    jal fact
    mul $v0,$a0,$v0
fact2:
    jr $ra
```

n=n-1

■ バグあり②

```
fact:
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    beq $0,$0,fact2
fact1:
    add $s0,$0,$a0
    addi $a0,$a0,-1
    jal fact
    mul $v0,$s0,$v0
fact2:
    jr $ra
```

どんな実行結果が得られるか？
factはリープ関数ではない。

再帰関数fact()の考え方(2)

■ バグあり②

```
fact:
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    beq $0,$0,fact2
fact1:
    add $s0,$0,$a0
    addi $a0,$a0,-1
    jal fact
    mul $v0,$s0,$v0
fact2:
    jr $ra
```

値が変わると困る変数はスタックで退避・復元する。

■ 正解（教科書とは違う）

```
fact:
    addi $sp,$sp,-12
    sw $ra,8($sp)
    sw $a0,4($sp)
    sw $s0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    beq $0,$0,fact2
fact1:
    add $s0,$0,$a0
    addi $a0,$a0,-1
    jal fact
    mul $v0,$s0,$v0
fact2:
    jr $ra
```

```
fact2:
    lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    jr $ra
```

スタックにレジスタを退避する順番は、コンパイラによって決められている。
この講義では、正しく動作すれば良い。

教科書の解答に変形する (1)

■ 退避と復元の対応

```
fact:
    addi $sp,$sp,-12
    sw $ra,8($sp)
    sw $a0,4($sp)
    sw $s0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    beq $0,$0,fact2
fact1:
    add $s0,$0,$a0
    addi $a0,$a0,-1
    jal fact
    mul $v0,$s0,$v0
fact2:
    lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    jr $ra
```

■ beq \$0,\$0,fact2の削除

```
fact:
    addi $sp,$sp,-12
    sw $ra,8($sp)
    sw $a0,4($sp)
    sw $s0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    jr $ra
fact1:
    add $s0,$0,$a0
    addi $a0,$a0,-1
    jal fact
    mul $v0,$s0,$v0
```

```
fact2:
    lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    jr $ra
```

・ beq \$0,\$0,fact2を使う分だけに、次のコードを挿入する。

```
lw $s0,0($sp)
lw $a0,4($sp)
lw $ra,8($sp)
addi $sp,$sp,12
jr $ra
```

教科書の解答に変形する (2)

■ fact2で\$a0の値は復元されるので、復元後の位置に、次の命令を移動すると\$s0が不要になる。

```
► mul $v0,$s0,$v0
    ↓
► mul $v0,$a0,$v0
```

■ mul命令の移動

```
fact:
    addi $sp,$sp,-12
    sw $ra,8($sp)
    sw $a0,4($sp)
    sw $s0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    jr $ra
fact1:
    add $s0,$0,$a0
    addi $a0,$a0,-1
    jal fact
    mul $v0,$s0,$v0
```

```
fact2:
    lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    mul $v0,$a0,$v0
    jr $ra
```

教科書の解答に変形する（3）

■ \$s0は不要になるので、\$s0に関する命令は削除できる。

- ▶ add \$s0,\$0,\$a0
- ▶ \$s0の退避・復元も削除

■ \$s0に関する命令の削除

```
fact:
    addi $sp,$sp,-12
    sw $ra,8($sp)
    sw $a0,4($sp)
sw $s0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
    jr $ra
fact1:
add $s0,$0,$a0
    addi $a0,$a0,-1
    jal fact
```

```
fact2:
lw $s0,0($sp)
    lw $a0,4($sp)
    lw $ra,8($sp)
    addi $sp,$sp,12
mul $v0,$a0,$v0
    jr $ra
```

教科書の解答に変形する（4）

■ \$s0に関する命令の削除に対応して、スタックへの退避・復元のコードを修正する。

■ 退避・復元の修正

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
sw $s0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
lw $s0,0($sp)
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    jr $ra
fact1:
    addi $a0,$a0,-1
    jal fact
```

```
fact2:
lw $s0,0($sp)
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
mul $v0,$a0,$v0
    jr $ra
```

教科書の解答に変形する（5）

■ 次のCコードに対応する部分で、\$a0と\$raの値は変更されていないので復元は不要である。

- ▶ if (n<1) return 1;
- ▶ addi \$v0,\$0,1
- ▶ lw \$a0,0(\$sp)
- ▶ lw \$ra,4(\$sp)
- ▶ addi \$sp,\$sp,8
- ▶ jr \$ra

■ 無駄な命令の削除

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
lw $s0,0($sp)
lw $ra,4($sp)
    addi $sp,$sp,8
    jr $ra
fact1:
    addi $a0,$a0,-1
    jal fact
```

```
fact2:
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

教科書の解答に変形する（6）

■ 見た目の違い

- ▶ L1とfact1
- ▶ fact2が残ってる

■ mul \$v0,\$a0,\$v0は存在しない命令

- ▶ \$v0=\$a0×\$v0
- ▶ 32ビット×32ビットは64ビットになる

■ 正しい命令列は次の通り。

```
mul $a0,$v0
mflo $v0
```

■ 教科書の解答

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,L1
    addi $v0,$0,1
    addi $sp,$sp,8
    jr $ra
L1: addi $a0,$a0,-1
    jal fact
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

■ 講義で示した解答

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    addi $sp,$sp,8
    jr $ra
fact1:
    addi $a0,$a0,-1
    jal fact
fact2:
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

教科書の解答の最適化

■ 教科書の解答

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,L1
    addi $v0,$0,1
    addi $sp,$sp,8
    jr $ra
L1: addi $a0,$a0,-1
    jal fact
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

■ 講義で示した解答

```
fact:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    slti $t0,$a0,1
    beq $t0,$0,fact1
    addi $v0,$0,1
    addi $sp,$sp,8
    jr $ra
fact1:
    addi $a0,$a0,-1
    jal fact
    jr $ra
fact2:
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```

■ 講義で示した解答の最適化

```
fact:
    slti $t0,$a0,1
    beq $t0,$0,fact1
    ori $v0,$0,1
    jr $ra
fact1:
    addi $sp,$sp,-8
    sw $ra,4($sp)
    sw $a0,0($sp)
    addi $a0,$a0,-1
    jal fact
fact2:
    lw $a0,0($sp)
    lw $ra,4($sp)
    addi $sp,$sp,8
    mul $v0,$a0,$v0
    jr $ra
```
