スタックと逆ポーランド記法

プログラミング言語C　I

(プッシュダウン）スタック(Stack)

- スタックは次の条件を満たすデータ構造

1. 新しい情報を追加できる
2. 古い情報の一つを取り出す。そのときには、最も新しいものから順に取り出す

(キューと同様) 追加と取り出し方法が制限された「リスト構造」とみることもできる

「最後に入った情報から先に出て行く」という性質から、LIFO(Last In First Out)とも呼ばれている

例：生協の食堂で積まれたお盆

スタックのデータ構造と固有な操作

- スタックの実現方法：配列と変数を用いる

配列array：　データの記憶用
変数top　：　データを記録する／取り出す位置を記憶。

配列の0から(top – 1)までが、スタックに記憶されているデータ
top の値は同時に、記録された要素数も表す

スタックに固有な操作

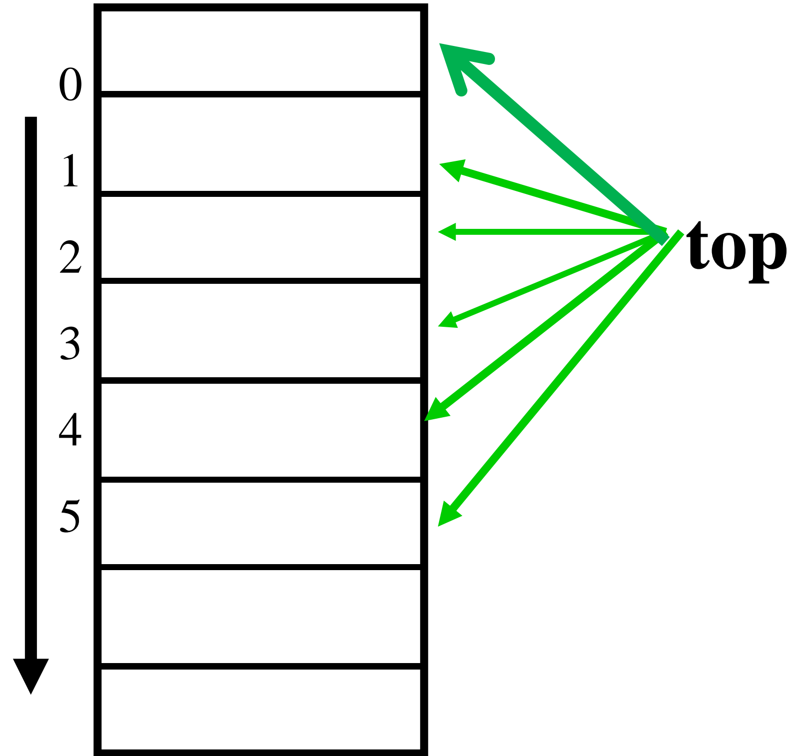
プッシュ(push)：スタックにおいてデータを記憶
ポップ(popup)：スタックにおいてデータを取出す

スタックの実現法

プッシュ

a
b
c
d
e

スタックの延びる方向



ポップ

配列array

配列によるスタックの実現

stack-1.c

```
#define NUM 100
int stack [NUM];      // データ記憶用の配列
int stack_top = 0;            // top変数(初期値は0)
```

```
void push(int st[ ], int *top, int
item) {
    int x = *top;  // top変数の値
    st[x] = item;  // データ記憶
   *top = x+1;   // top変数の更新
}
```

```
int pop(int st[], int *top) {
    int x = *top - 1,    // 最新要素の場所
        item = st[x];    // 最新データ
    *top = x;            // top変数の更新
    return item;      // データを返す
}
```

使用例

```
int main(void) {
    push(stack, &stack_top, 10);      //  スタックに 10 をpush
    push(stack, &stack_top, 20);      //  スタックに 20 をpush
    push(stack, &stack_top, 30);      //  スタックに 30 をpush
   printf(“popped --- %d -(%d)--¥n”, pop(stack, &stack_top),stack_top);   // popする
    push(stack, &stack_top, 40);      //  スタックに 40 をpush
    push(stack, &stack_top, 50);      //  スタックに 50 をpush
    while (stack_top > 0) {           // スタックに要素があるかぎりpopする
            printf("popped --- %d  (%d) ¥n", pop(stack, &stack_top),stack_top);
    }
    return 0;
}
```

この方法の問題点

1. 二つの配列が関係していることが明示されない:

```
int stack [NUM];     // データ記憶用の配列
int stack_top = 0;   // top変数(初期値は0)
```

2. Push関数とpop関数が「スタック」に固有な操作であることが明示されない

```
void push(int st[ ], int *top, int item) {
    int x = *top;  // top変数の値
    st[x] = item;  // データ記憶
   *top = x+1;   // top変数の更新
}
```

```
int pop(int st[], int *top) {
   int x = *top - 1,    // 最新要素の場所
        item = st[x];   // 最新データ
   *top = x;                    // top変数の更新
   return item;       // データを返す
}
```

構造体

「配列」のような構造（データ構造）を定義できる

配列と同様に、

複数の変数をひとまとめにできる

（その「ひとまとめにした」ものに名前を付ける）

配列と異なり

変数の型は、いろいろであってよい

構造体によるスタックの実現

stack-2.c

```
#define NUM 100
typedef struct stack {
    int array [NUM];     // データ記憶用の配列
    int top;             // top変数(初期値は後で与える)
} STACK;        // 上の構造体をSTACK型として宣言
```

```
void push(STACK *st, int item)
{
 // 記憶位置，top変数の更新
   int top = (*st).top++;
// 記憶
   (*st).array[top]=item ;
}
```

```
int pop(STACK *st) {
   //top変数の更新，データの位置
   int top=--(*st).top;
   return (*st).array[top];
  // データを返す
}
```

構造体によるスタックの実現(続)

stack-2a.c

(*st).top は st->top

(*st).array は st->array と書けることを使うと...

```
void push(STACK *st, int item)
{
 // 記憶位置, top変数の更新
  int top = st->top++;
// 記憶
  st->array[top]=item ;
}
```

参考: push関数の中身を一行で書くと

```
st->array[st->top++]=item;
```

参考: pop関数の中身を一行で書くと

```
return st->array[--st->top];
```

```
int pop(STACK *st) {
  //top変数の更新, データの位置
  int top=--st->top;
  return st->array[top];
 // データを返す
}
```

使用例

```
int main(void) {
    STACK st ;        st.top = 0;        //  スタック変数stの宣言と、st.topの初期化
    push(&st, 10);       //  スタックに 10 をpush
    push(&st, 20);
    push(&st, 30);
    printf(“popped ---
    push(&st, 40);
    push(&st, 50);
    while (st.top > 0)
            printf("pop
    }
    return 0;
}
```

配列によるスタックの実現と異なり、
スタックの中身(配列とtop変数)を
ひとまとめで扱う
大域変数ではなく局所変数として扱える
スタックを関数に渡すときは「アドレス」
を渡す
参考: stack-3.c

逆ポーランド記法

日本語表記
に合う

数式において演算子を最後に置く記法
(先頭に置く記法がポーランド記法)

例: 1 + 2 ⇒ 1 2 +

「1に2を+(たす)」

1 + 2 * 3 ⇒ 1 2 3 * +

「1に、2に3を*(掛けた)のを+(たす)」

(1 + 2) / (3 – 4) ⇒ 1 2 + 3 4 - /

「1に2を+(足した)ものを、3から4を-(引いた)もので/(割る)」

逆ポーランド記法の数式の計算

手続き：

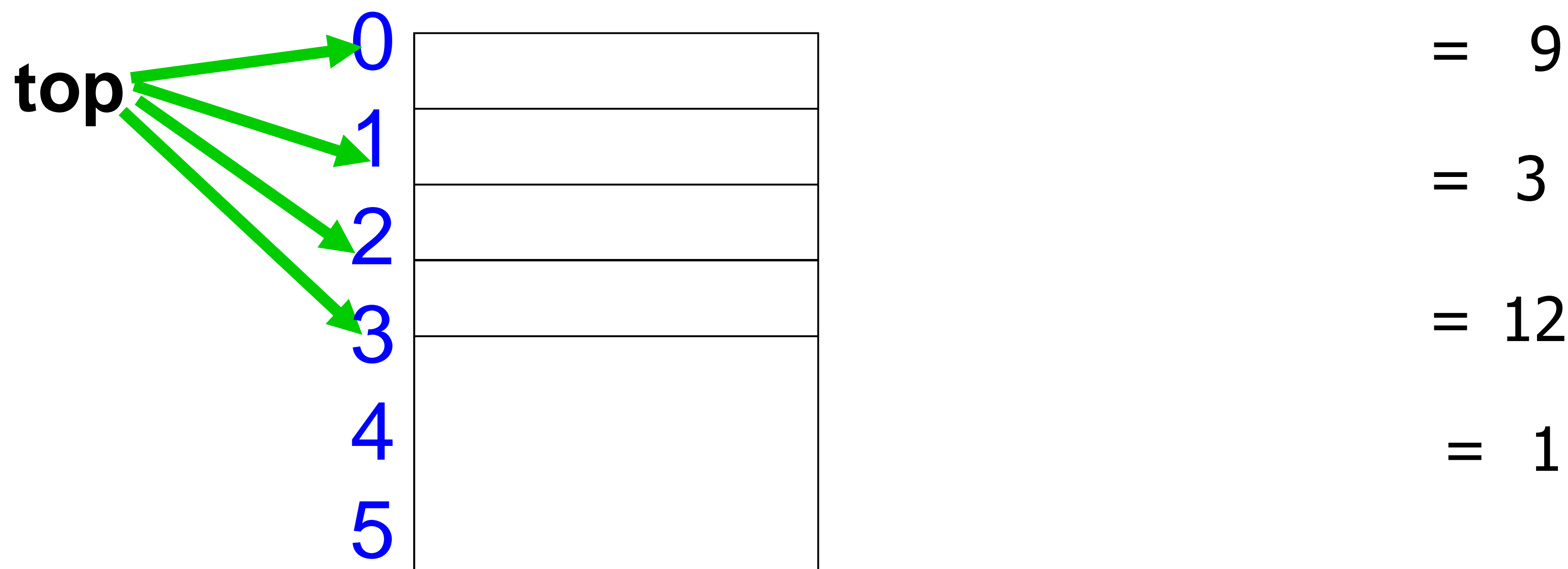
1. 数を読み込んだらスタックに積む
2. 演算子(＋－＊／　これらを@で表す）を読み込んだら、スタックから二つの数をpopし（取り出した順にa,bとする）、
   b @ aを計算してスタックに積む
3. ＝を読み込むか、もう読み込むものがなければ
   スタックをpopして終了

(4. p を読み込んだら、スタックの状態を表示する)

例：13 5 4 + 3 / 4 ＊ －

例：「13 5 4 + 3 / 4 ＊ －」をスタックで処理

入力： 13 5 4 ＋ 3 ／ 4 ＊ －



1. 数を読み込んだらスタックに積む

2. 演算子(＋－＊／、＠で表す)を読み込んだらスタックから二つの数をpopし(取り出した順にa,bとする)、b ＠ aを計算してスタックに積む

逆ポーランド記法の数式の計算

- 演習問題

  次の逆ポーランド記法で書かれた数式の処理の過程を記述せよ。また結果はいくらか？

  (1) 1 2 3 + + 5 * 10 /

  (2) 1 2 3 4 5 6 7 8 9 - + - + - + - +

逆ポーランド記法の計算プログラム

porland.c は配列を用いた逆ポーランド記法の計算プログラムである。

課題

1. 例題を入力し、計算結果を照合せよ
2. 構造体を用いたプログラムに改変せよ
3. 「逆ポーランド記法の数式の計算」の課題を入力し、計算結果と自分の計算結果とを照合せよ