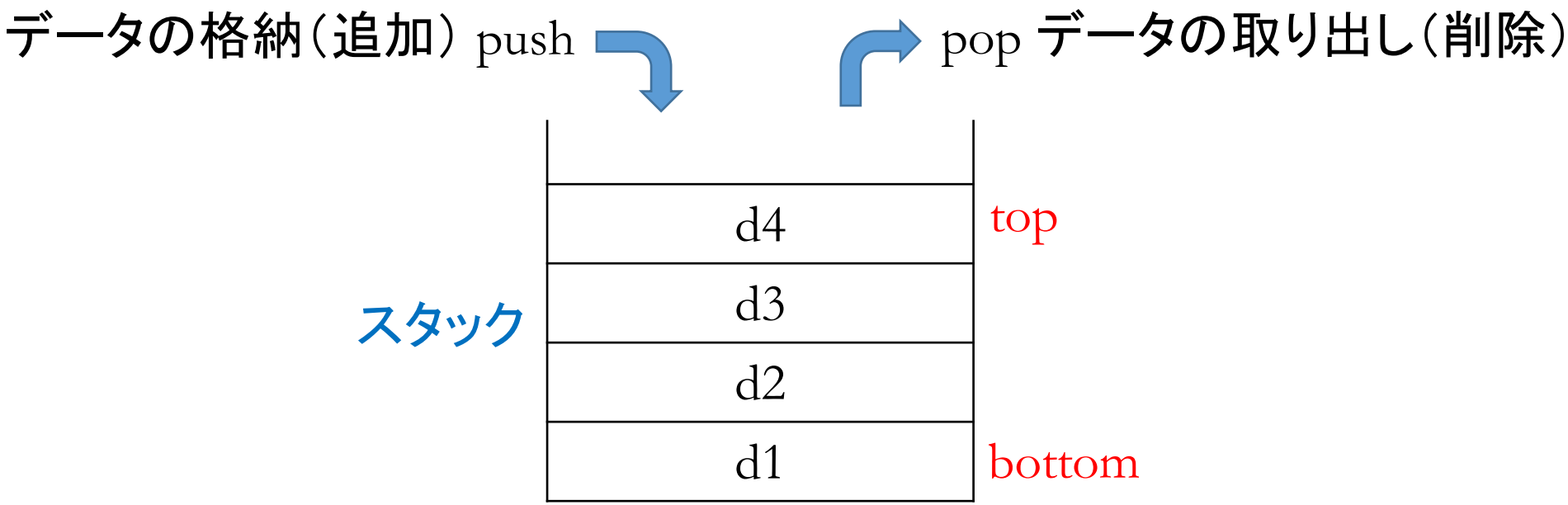
スタック

基礎から応用まで

スタック（Stack）とは

- データを一時的に保存しておくためのもの
- スタックは本来、（本などの）積み重ねの意味。積み重ねなので、一番上の（一番後に置いた）ものから取っていく
- すなわち、データは後入れ先出し（LIFO: Last In First Out）の構造で保持する

スタックの操作・用語



スタックの使用例

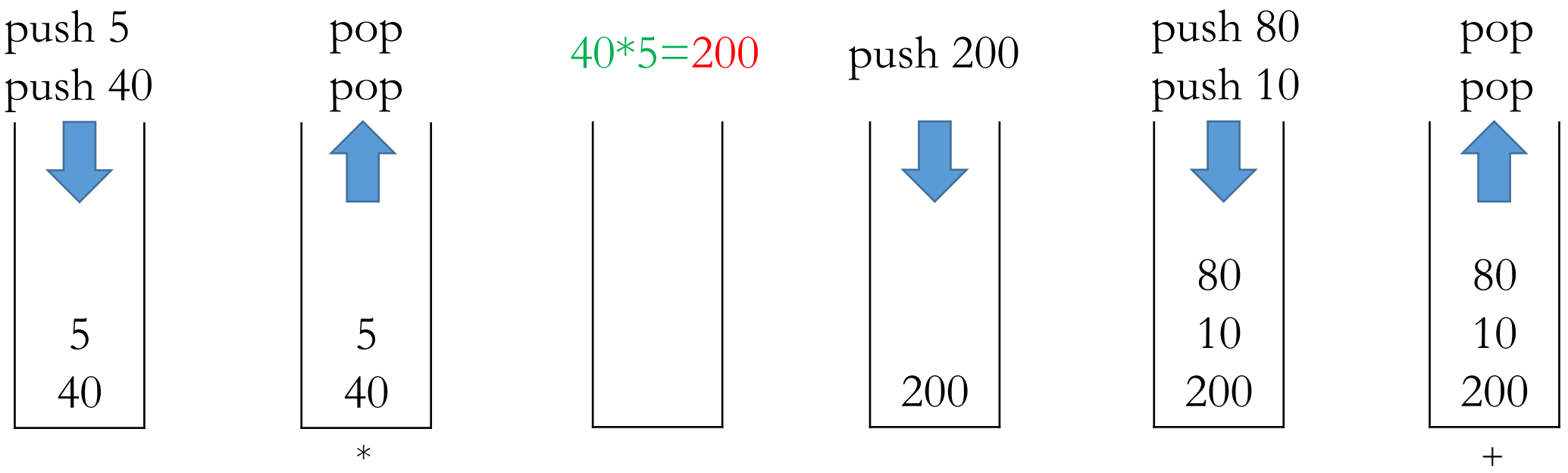
- 関数呼び出しが入れ子になっている場合の各関数の戻り先の管理
  - func1()がfunc2()を、func2()がfunc3()を…で呼び出すとき、戻り先の保存と取り出しは「後入れ先出し」の関係
  - 当然、再帰関数も入れ子になっていると考えてよい
- 実は、以下すべてがスタックを使っている
  - 関数リターン時の戻り先アドレス
  - 引き数
  - ローカル変数
  - 一時変数（計算の途中結果を保存するためにコンパイラが生成する変数）
- Undo機能
  - Wordやパワーポイントに「元に戻す」というボタンがある。これがUndo機能

スタック利用の具体例：逆ポーランド記法

- 一部の計算機や電卓は逆ポーランド記法を使用している
- 逆ポーランド記法は、計算機のスタックを活用することで、演算子の優先順位を明示せずに計算を行うことができる

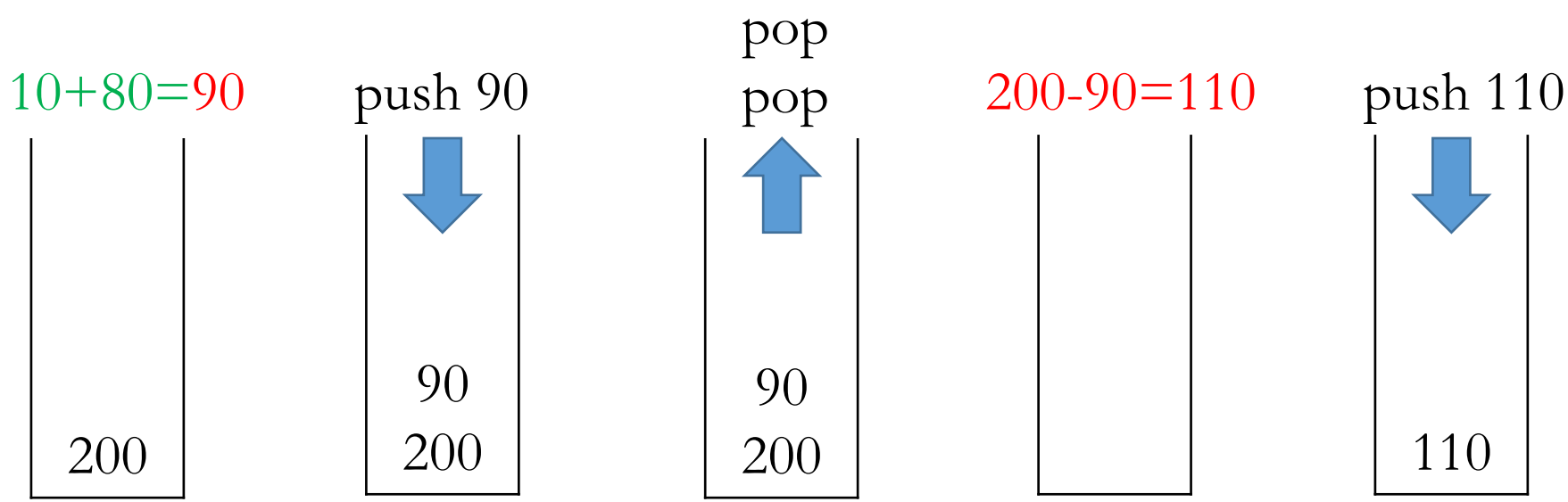
スタック利用の具体例：逆ポーランド記法

- 数式　　40*5-(10+80)
- 逆ポーランド記法　　40 5 * 10 80 + -
- 計算過程：①数であればpush。②演算子であれば、2回pop⇒それらを演算⇒結果をpush。①②を最後の演算子まで繰り返す



スタック利用の具体例：逆ポーランド記法

- 数式　　40*5-(10+80)
- 逆ポーランド記法　　40 5 * 10 80 + -
- 計算過程：①数であればpush。②演算子であれば、2回pop⇒それらを演算⇒結果をpush。①②を最後の演算子まで繰り返す



スタックの実装（スタックの関数⇒stack.h）

```
#include <stdlib.h>

typedef  int data_t;
#define STSIZE 1000
data_t stack[STSIZE];
int top;

void InitStack(void){top = -1;}

int StackEmpty(void)
{
  if(top==-1) return 1;
  else return 0;
}
```

```
void push(data_t x){
  if(top==STSIZE-1){
    puts(“stack is full”);
    exit(-1);}
  stack[++top]=x;  // top++;stack[top]=x;
}

data_t pop(void){
  if(top==-1){
    puts(“stack is empty");
    exit(-1);}
  return stack[top--];  // stack[top]; top--
}
```

manaba小テスト：05-1

- 8分
- 8点

スタックによる四則演算

応用：括弧のない四則演算式

何が問題？

- 括弧のある複雑な数式は、数式について構文解析を行い、例えば逆ポーランド記法に直すことができれば例で示したようにスタックを用いて計算ができる
- 実際、括弧のない四則演算もそんなに簡単ではない
  - たとえば、$5 + 2 * 10 + 16 - 4 / 2$
- 順に演算を行うと、結果が41になってしまう
- スタックを利用することにより、正しく演算できる

順に演算する場合の手順

- 結果を格納用変数r、数式（文字列）を格納する1次元配列fとその数式の個々の数または演算子（複数文字列）を格納する2次元配列sを用意

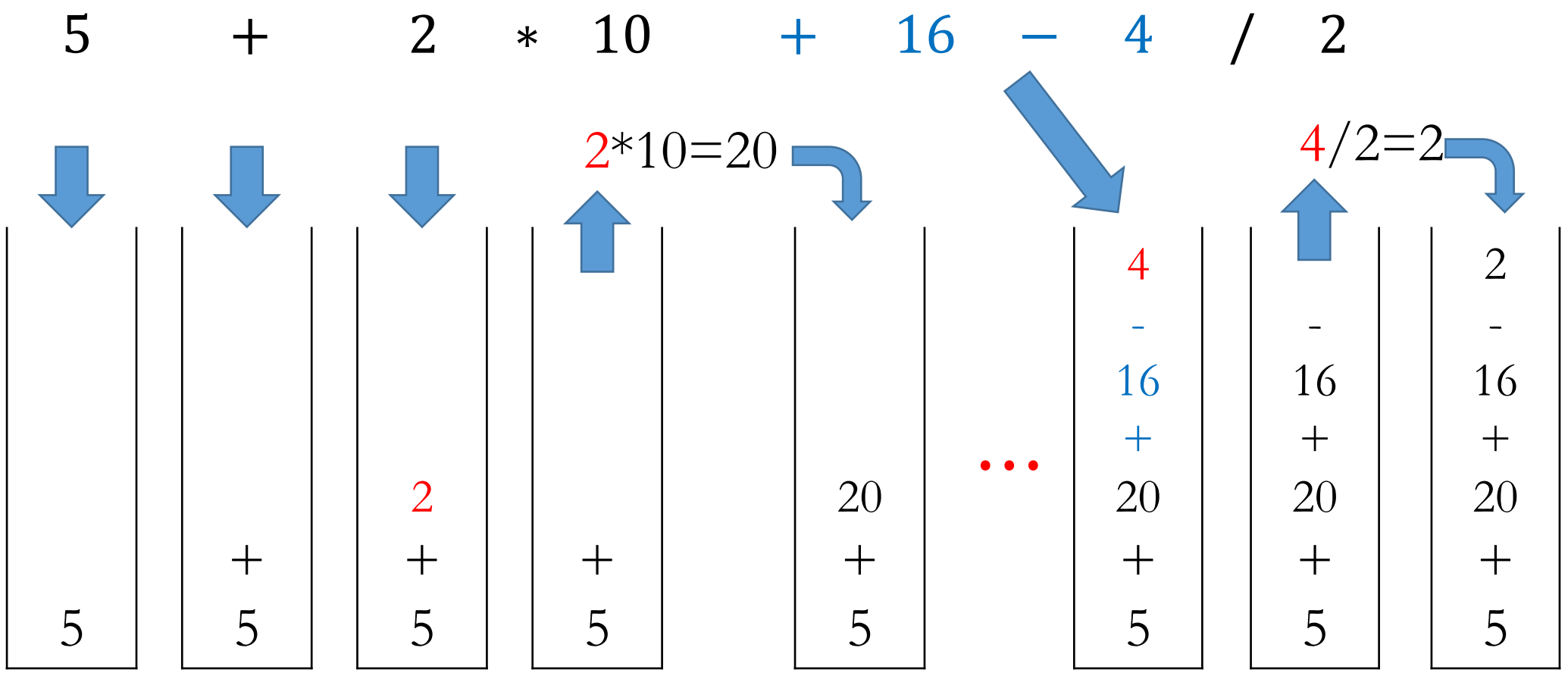
1. 入力された数式（文字列）の個々の数・演算子（文字列）を順にs[i]に入れる。その個数numも求める（i=0, 1, ..., num-1）。r=s[0]で初期化する
2. i=1からnum-1まで、以下を繰り返す

   2.1 r = r op s[i]　op: 演算子+, -, *, /のいずれ

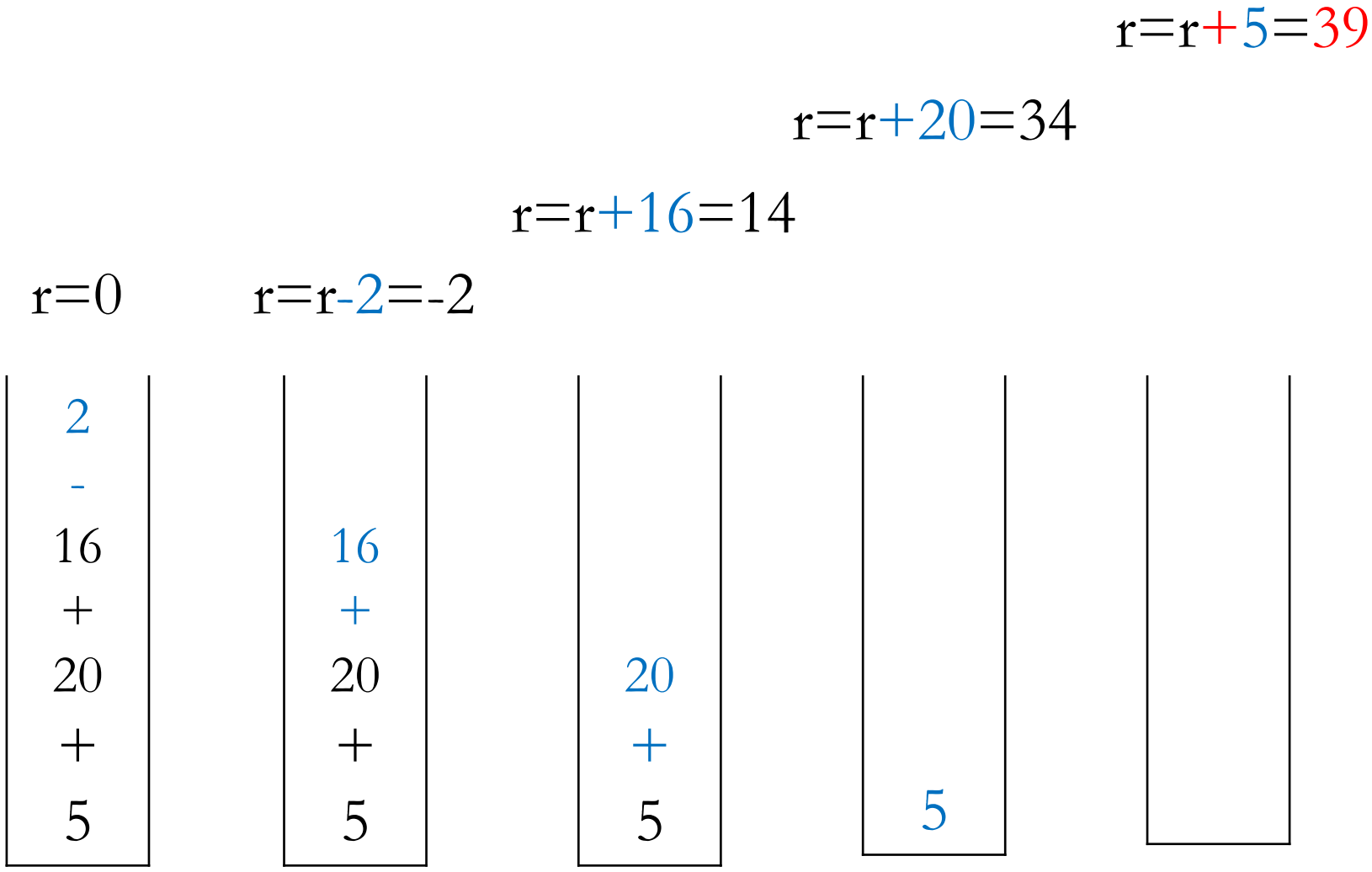
   // プログラムを作成する場合、文字列を整数に変換するとかいろいろな処理が必要

しかし、これだと、前の例の答えが41になってしまう！

スタックによる四則演算の優先順位の制御



スタックによる優先順位付き四則演算



スタックによる四則演算のまとめ

1. 数式を順にみて行き、数と演算子+, -であればスタックにpush
2. 演算子*, /であれば、一回popを行いその数と演算子直後の数を演算し、その結果をにスタックにpush
   - 1,2の処理を最後の数まで行うと、スタックには「数」と乗算除算の「中間結果」と「演算子+,-」が処理すべき順に残る
   - 5 + 2 * 10 + 16 – 4 / 2の場合、5 + 20 + 16 – 2がスタックに残る
   - 5+2*10*5の場合、5+100がスタックに残る
3. スタックが空になるまで数と演算子をペアでpopして順に計算していく
   - ただし、数が演算子より1つ多いので、最後の数については特例処理が必要

スタックによる四則演算の手順

- 結果を格納用変数r、数式（文字列）を格納する1次元配列fとその数式の個々の数または演算子（複数文字列）を格納する2次元配列sを用意

1. 入力された数式の個々の数・演算子を順にs[i]に入れる。その数numも求める（i=0, 1, ..., num-1）。r=0で初期化する
2. i=0からnum-1まで以下を繰り返す

   2.1 s[i]が*, /でなければpush(s[i])を行う

   2.2 s[i]が*, /であれば、x=pop(), s[i+1]=x op s[i+1]を行い、2.に戻る。ただし、opは*か/を表す

   // 演算子の後ろに必ず数があるので、上記s[i+1]のi+1はnum-1を超えない

スタックによる四則演算の手順

3. 以下を繰り返す

3.1 x=pop()

3.2 スタックが空か判定。空であれば（特例処理）r=r+xとし、処理終了

3.3 op=pop()

3.4 r = r op x

//実際のプログラミングにおいては、文字列の整数化など、いろいろが必要

計算の実行結果例

./a.out
計算式を入力(Enterで終了)
5+ 2*10 + 16 -4 /2
r: 39

計算式を入力(Enterで終了)
5+2*10*5
r: 105

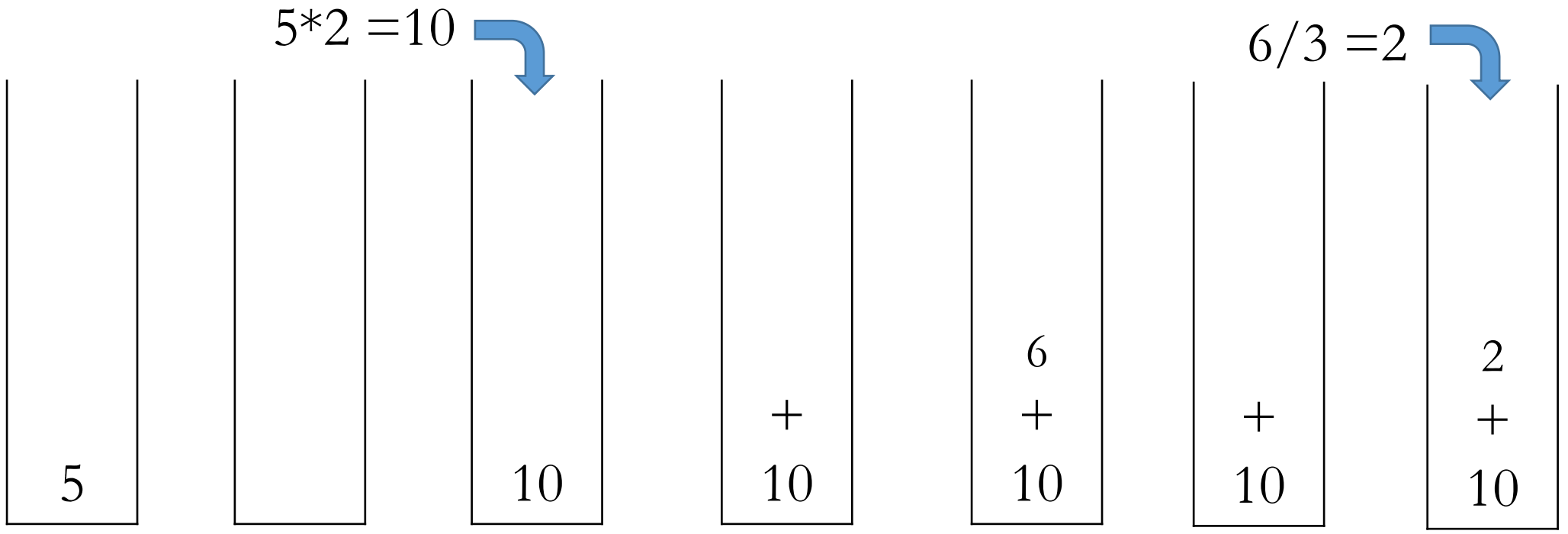
計算式を入力(Enterで終了)

manaba小テスト：05-2

- 12分
- 10点

manaba小テスト：05-2

- 数式が5*2+6/3の場合の10+2を得るまでのスタックの中身を時間順に示しなさい。



スタックによるクイックソート

スタックによる深さ優先探索

Nクイーンの配置