

cp-9. 再帰関数

(C プログラミング入門)

URL: <https://www.kkaneko.jp/pro/adp/index.html>

金子邦彦





内容

例題 1 . スタック

局所変数, 大域変数

例題 2 . 再帰関数による総和

- 局所変数を使った関数を理解する
- 関数の再帰呼び出しを使って，簡単な漸化式の計算を行う

例題 1 . スタック

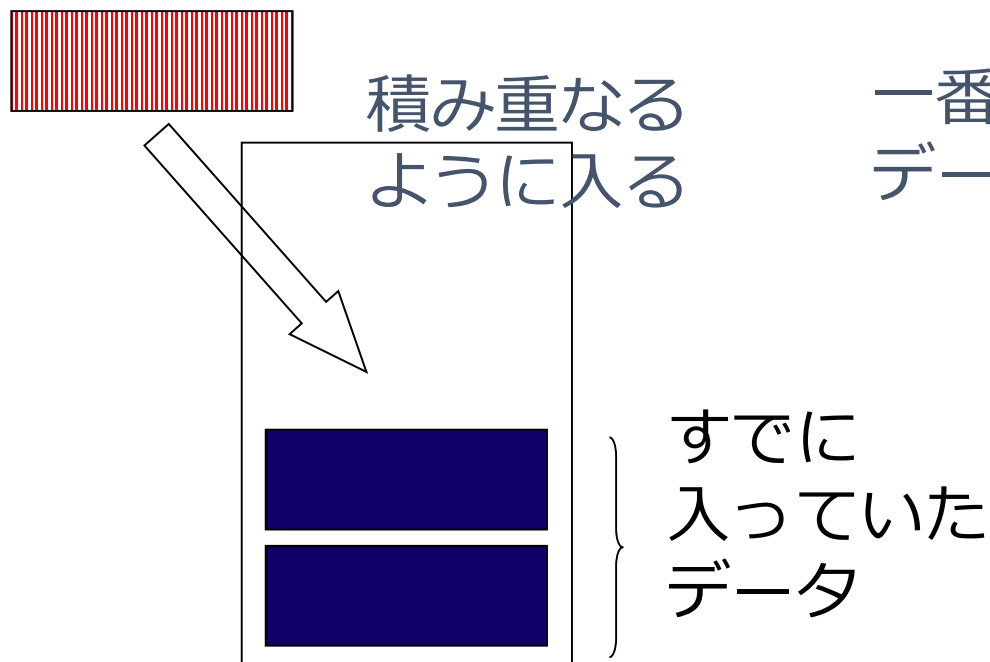


- スタックのプッシュ(push), ポップ(pop) を行う関数を作る
 - 次の2つの大域変数を使うこと
 1. 配列 **stack**
 2. スタックポインタ **top**

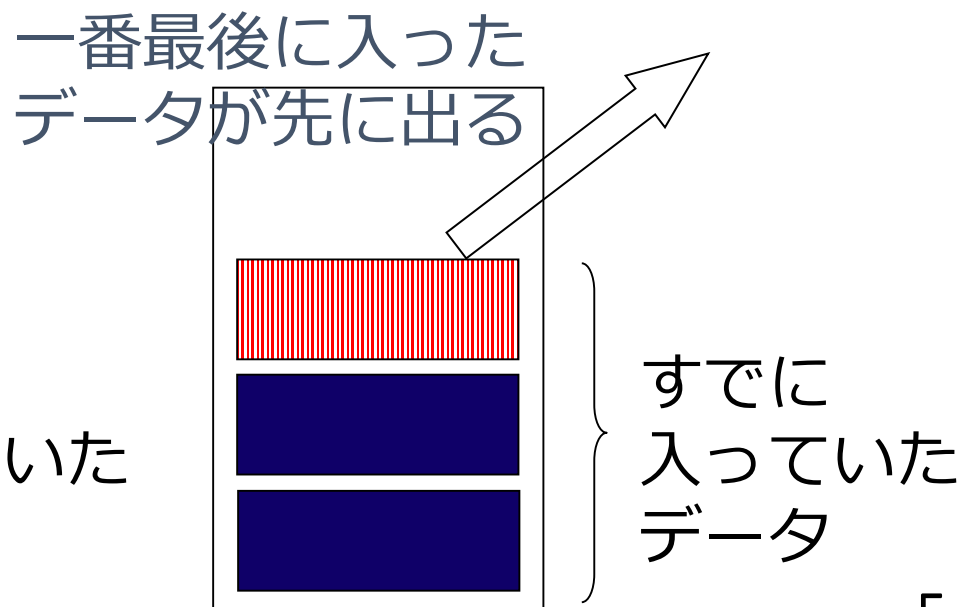
スタックのプッシュとポップ



- プッシュ
(入れる操作)



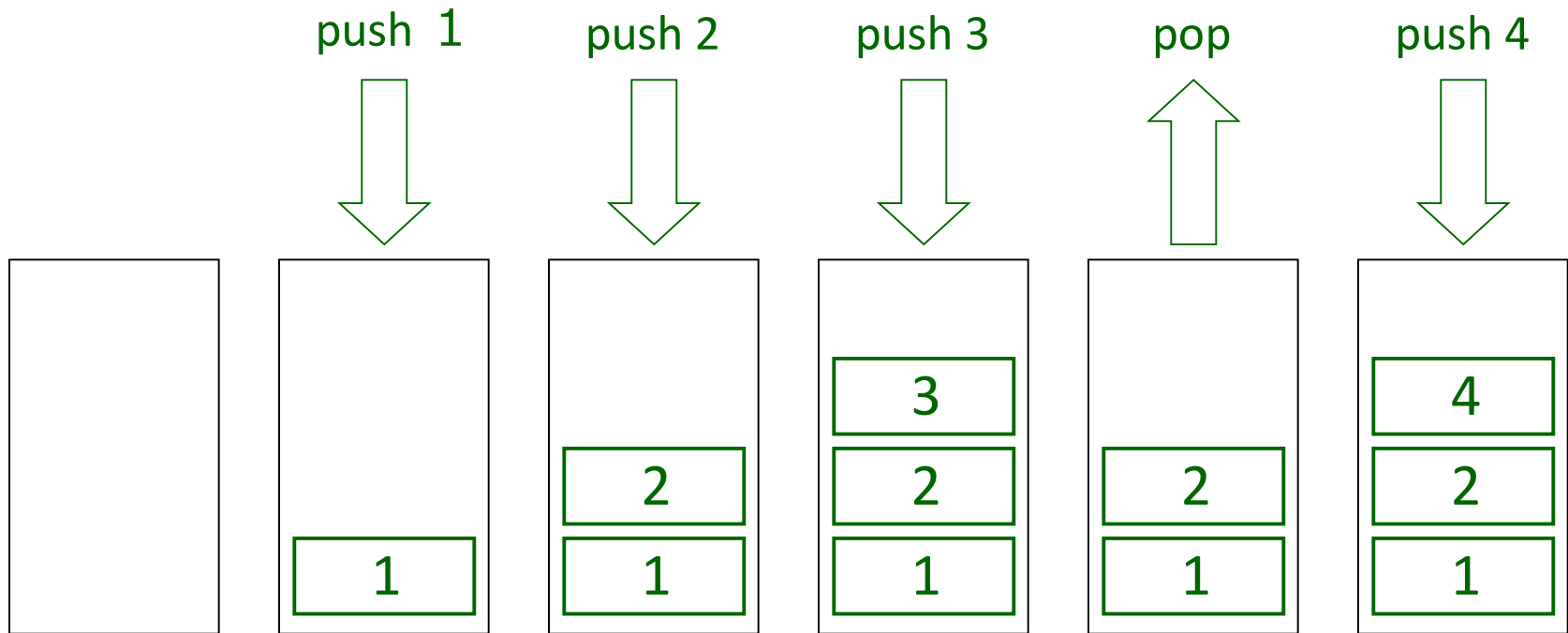
- ポップ
(出す操作)





スタックとは

- 入れた順に積みあがっていく





```
#include <stdio.h>
int stack[100];
int top=0;
```

```
void push(int n)
{
    stack[top]=n;
    top++;
    return;
}
```

push関数

```
int pop()
{
    int n;
    top--;
    n = stack[top];
    return n;
}
```

pop関数

スタック

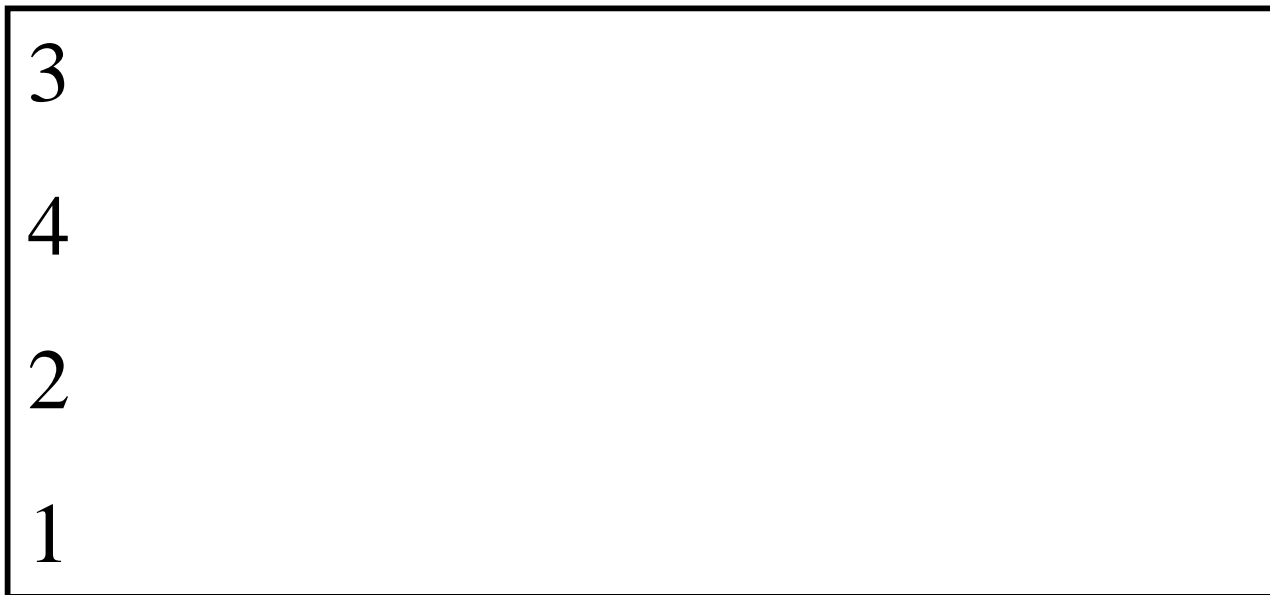


main 関数の例

```
int main()
{
    push(1);
    push(2);
    push(3);
    printf( "%d¥n", pop() );
    push(4);
    printf( "%d¥n", pop() );
    printf( "%d¥n", pop() );
    printf( "%d¥n", pop() );
    return 0;
}
```


スタック

実行結果の例





関数呼び出しの流れ

main 関数
`int main()`

push 関数

`void push(int n)`

関数呼び出し

`push(1);`

`push(2);`

`push(3);`

`printf("%d\n", pop());`

`push(4);`

`printf("%d\n", pop());`

`printf("%d\n", pop());`

`printf("%d\n", pop());`

戻り

`return;`

pop 関数

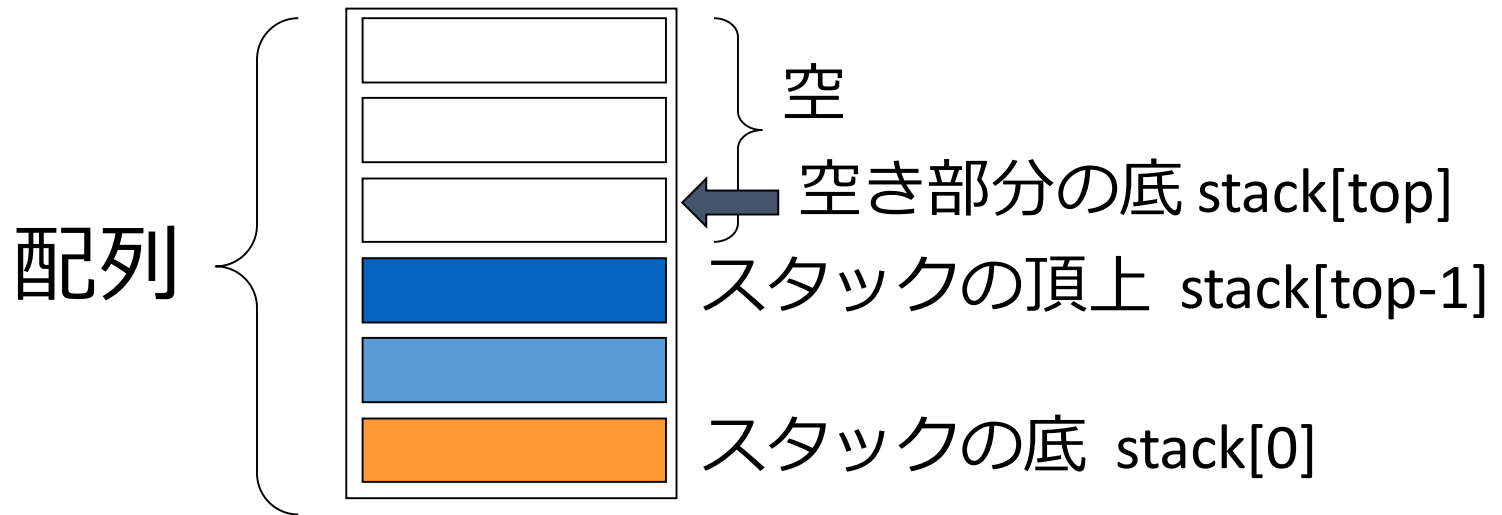
`int pop()`

戻り

`return n;`



配列によるスタックの実現



• スタック

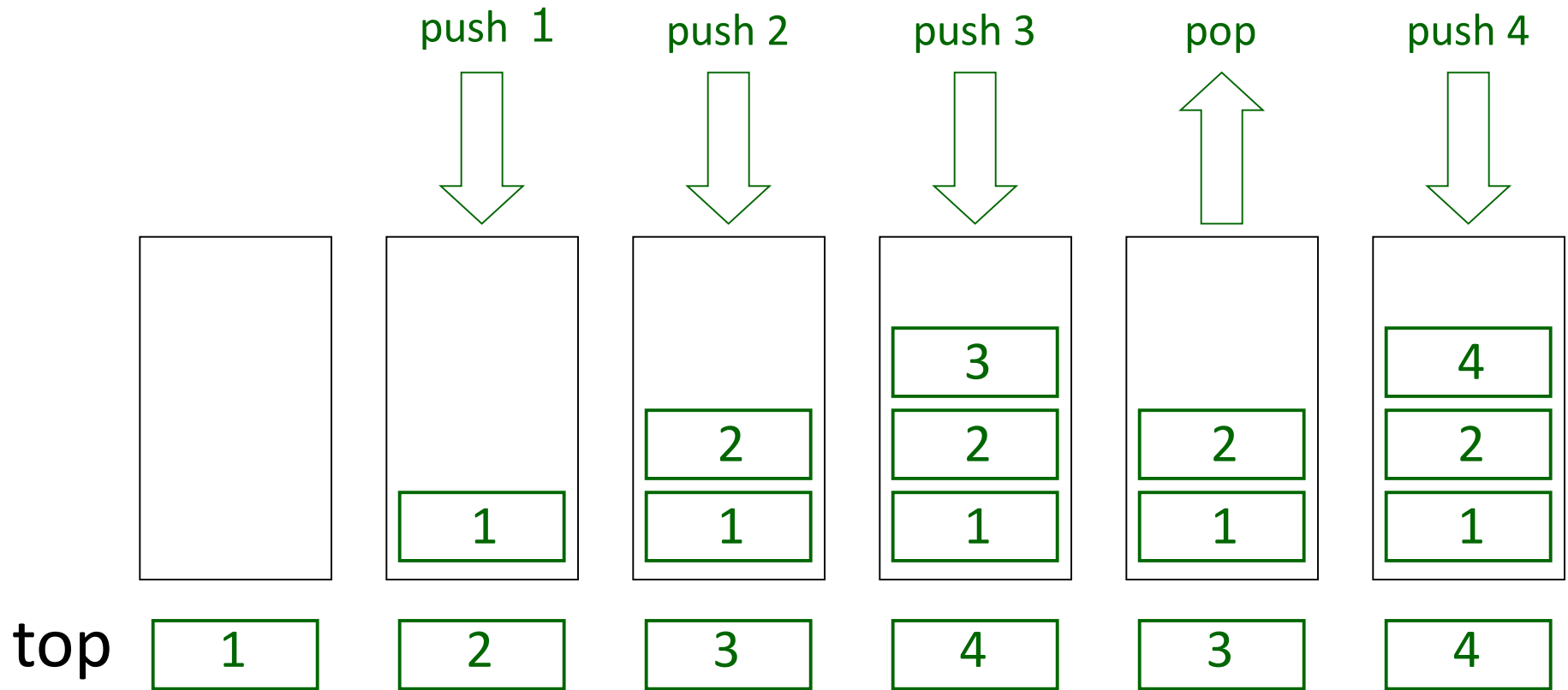
- 十分に多くのデータを格納できる配列**stack**を用意.

• スタックポインタ

- 現在の「空き部分の底」を示す値（スタックポインタ）を格納させる変数**top**を用意.
- スタックの底は **stack[0]**, スタックの頂上は **stack[top-1]**



配列によるスタックの実現



- push 時 : pushの後に, topの値を 1 足す
- pop 時 : popの前に, topの値を 1 引く



局所変数

- 関数の仮引数と、関数本体内で宣言された変数のこと
 - 関数の内部でだけ有効
 - ほかの関数（**main**関数など）からは直接アクセスできない
- **ある**関数の局所変数と、呼び出した関数中の局所変数と、例え同じ名前であっても、別の実体である

大域変数



- 関数の外側で宣言された変数のこと

例) 配列`stack`とスタックポインタ`top`は, `push`関数と`pop`関数のいずれからも参照できる.

```
#include <stdio.h>
int stack[100];
int top=0;
int push(int n)
{
    stack[top]=n;
    top++;
    return n;
}
int pop()
{
    int n;
    top--;
    n = stack[top];
    return n;
}
```

これが大域変数
`push`関数と
`pop`関数で
共有される



局所変数と大域変数

	局所変数	大域変数
宣言の場所	関数の仮引数 または関数内	関数の外側
有効範囲	関数の内部だけで有効	複数の関数で共有される

スタックの使用例



- 正整数を読み込んだら, **push** する.
- 負整数を読み込んだら, **pop** して, 取り出された値を表示する.
- **0** が読み込まれるまで上の操作を繰り返す.



```
#include <stdio.h>
#pragma warning(disable:4996)
int main()
{
    int item;
    top= -1;
    do {
        printf("Enter a number ");
        scanf("%d",&item);
        if (item>0) {
            printf("pushed: %d¥n",item);
            push(item);
        } else if (item<0) {
            printf("popped: %d¥n",pop());
        }
    } while (item!=0);
    return 0;
}
```



課題 1 . スタック操作の例外処理

スタック操作のプログラムにおいて,
push 関数と pop 関数を書き換えて,
次に挙げた, 2つの例外事象の対処を考慮しなさい.

1. スタックが空のときにポップ (pop) しない.

スタックが空のときに pop 関数が呼び出されると, 「Stack empty」のエラーメッセージを表示すること

2. スタックが満杯のときにプッシュ (push) しない.

スタックが満杯のときに push 関数が呼びだされると,
「Stack full」のエラーメッセージを表示すること

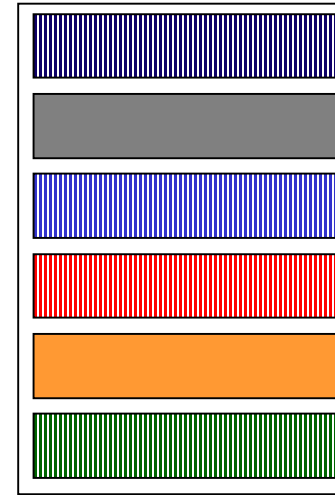
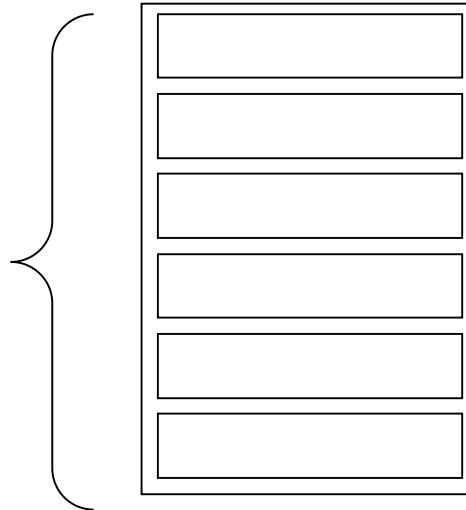


課題 1 のヒント

空のとき

満杯のとき

配列



スタックポインタのスタックポインタの
値は「0」

値は、「配列のサイズ」

- スタックポインタは、現在の「空き部分の底」の位置を示す値 → 「スタック内のデータ数」に等しい



例題 2 . 再帰関数による総和

- 整数 N から, 1 から N まで総和を求める再帰関数を作る

例) $5 \rightarrow 15$

$$\sum_{i=1}^n i = \begin{cases} 1 & \text{if } n = 1 \\ n + \sum_{i=1}^{n-1} i & \text{otherwise} \end{cases}$$

- 再帰関数とは, 関数 f の本体に f の呼出しを含むような関数のこと

再帰関数とは



- 自分自身を呼び出すような関数のこと
- 必ず終了条件が成立するように気をつけること



```
#include <stdio.h>
#pragma warning(disable:4996)
```

```
int sum(int n)
{
    if (n < 1) {
        return 0;
    }
    if (n == 1) {
        return 1;
    }
    else {
        return n+sum(n-1);
    }
}
```

sum関数

```
int main()
{
    int n;
    int s;
    printf("Enter a number: ");
    scanf("%d",&n);
    s = sum(n);
    printf("sum(%d)=%d¥n",n,s);
    return 0;
}
```

main関数



再帰関数による総和

実行結果の例

Enter a number: 2

sum(2)=3



関数呼び出しの流れ (main 関数で $n = 2$ のとき)

main 関数

int main()

関数呼び出し

sum(n);

sum 関数

int sum(int n)

関数呼び出しと戻

り
return n + sum(n-1);

sum 関数

int sum(int n)

戻り

return 1;

n の値の変化 (main 関数で $n = 2$ のとき)



main 関数

int main()

関数呼び出し

sum(n);

sum 関数

int sum(int n)

$n = 2$

関数呼び出しと戻

り
return n + sum(n-1);

3 を返す

sum 関数

int sum(int n)

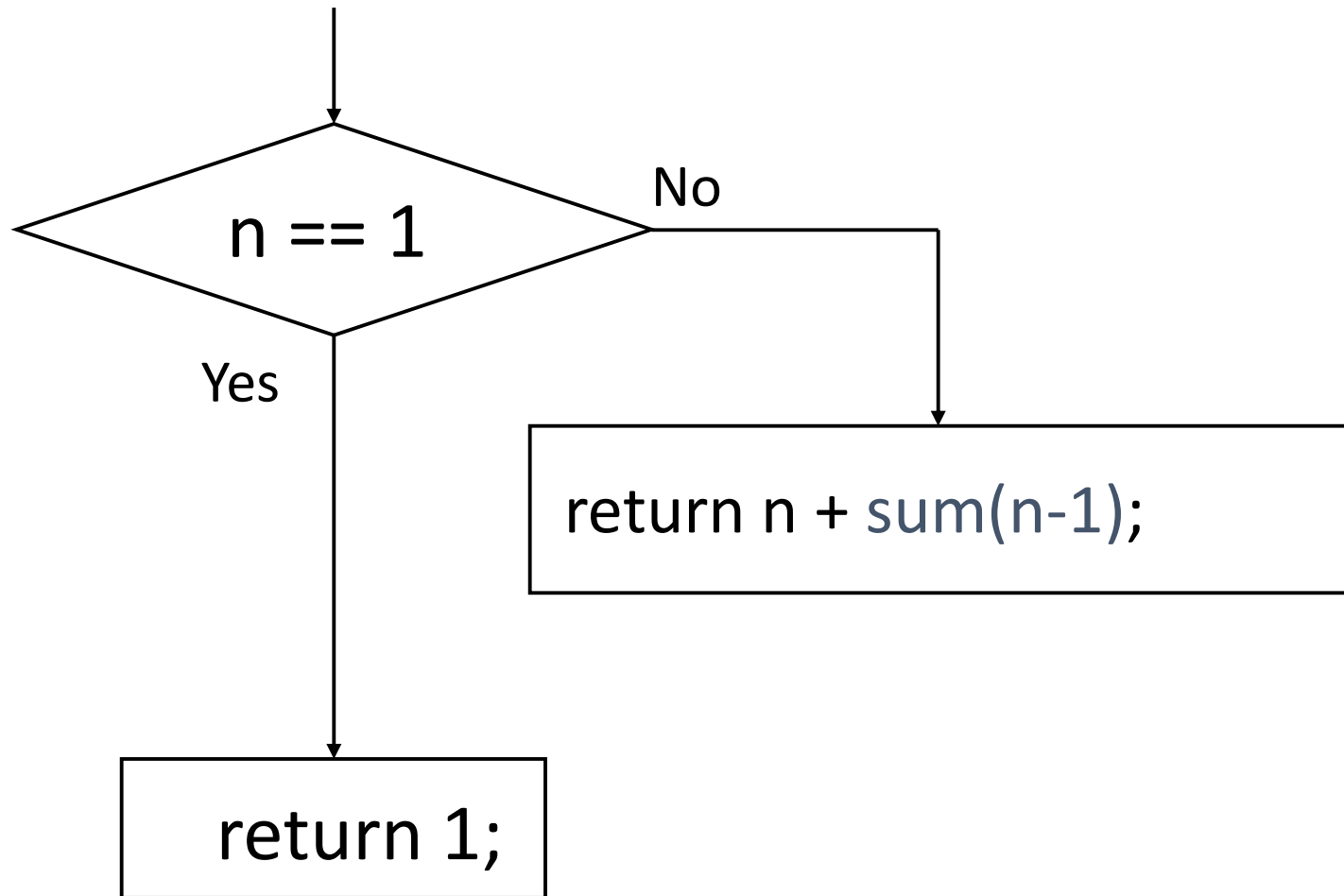
$n = 1$

戻り

return

1 を返す

再帰関数による総和





```
int main()
```

```
int sum( int n )
```

```
int sum( int n )
```

データの流れ

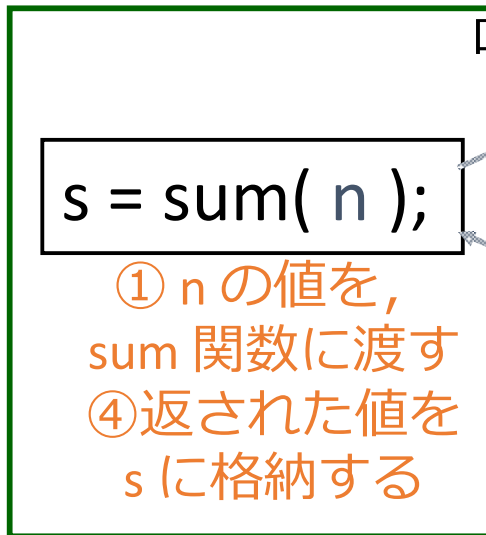


main 関数

int main()
データ



プログラム

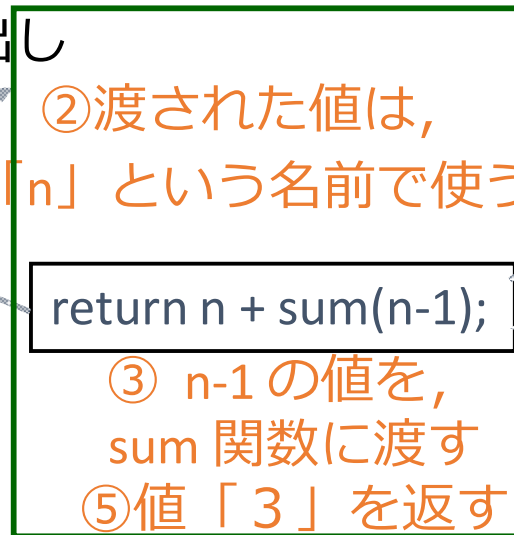


bar関数

int sum(int n)
データ

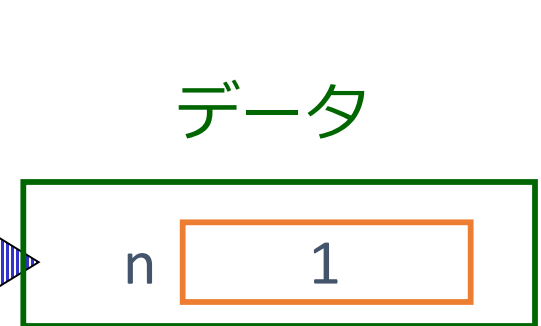


プログラム

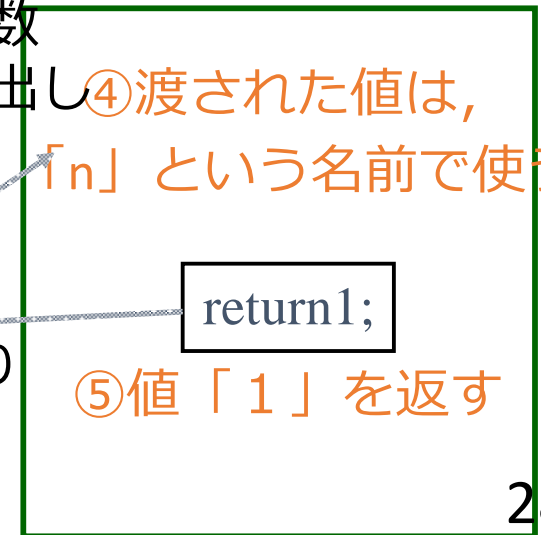


bar関数

int sum(int n)
データ



プログラム



関数
呼び出し

戻り

関数
呼び出し

戻り



呼び手

```
return n+sum(n-1);
```

引数を仮引数にセットする

関数
本体

```
int sum(int n)
{
    if (n < 1) {
        return 0;
    }
    if (n==1) {
        return 1;
    }
    else {
        return n+sum(n-1);
    }
}
```

戻り値の受け取り