篩の効果（多重分岐）

ifのみ

```
if (point<0 || point>100)    printf("誤入力¥n");
if (point>=80 && point<=100) printf("優¥n");
if (point>=70 && point<80)   printf("良¥n");
if (point>=0 && point<70)    printf("可or不可¥n");
```

篩の効果　⇩ 多重分岐による単純な篩い分け

```
if (point<0 || point>100) printf("誤入力¥n");
else if (point>=80)       printf("優¥n");
else if (point>=70)       printf("良¥n");
else                      printf("可or不可¥n");
```

条件式の評価回数

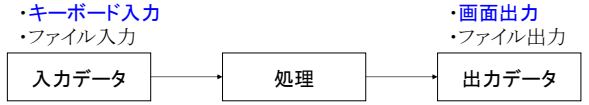
point	ifのみ	篩の効果
101	4	1
90	4	2
75	4	3
60	4	3

篩の効果により
- 条件式の評価回数を減らせる.
- 論理演算子の評価回数を減らせる

C言語を用いたプログラミング4 補足資料

- 反復
  - while ~
  - do ~ while
  - for ~
- 処理のスキップ
  - continue

プログラム



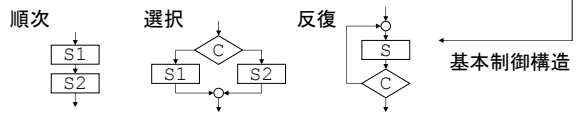
- データを記憶するための変数（データ構造）
  - 整数型
  - 実数型
  - 文字型

  → 複数のデータを記憶 → 配列（数値リスト，ベクトル，行列，...）
- 処理をするための制御構造（アルゴリズム）
  - 順次
  - 選択
  - 反復

  → モジュール化 → 関数

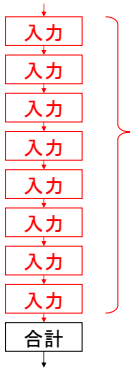
主な制御構造

制御構造	意味	分類
		順次
if ~ else ~	条件分岐	選択 selection
switch ~ case ~	多岐選択	
while ~	前判定	反復 repetition
do ~ while	後判定	
for ~	繰り返し	
break	処理の中断	ループ処理
continue	処理のスキップ	



反復

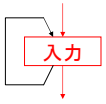
同じ処理を何度も繰り返したい



2+4+6+8+10+12+14+16+18+20 合計は？
5,4,3,2,1,0 Go!! カウントダウンせよ.
1.2 + 8.4 + 4.3 + ... 合計は？

8回繰り返せ

特徴：上流へ戻る



キーポイント
- 処理を繰り返す条件 → 継続条件
- 反復回数の記憶 → カウンタ

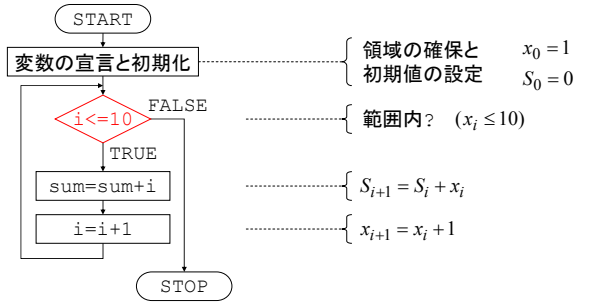
プログラム4.1

設計仕様　1から10までの和を求めるプログラムを作成しなさい

$$S = 1 + 2 + \cdots + 10$$

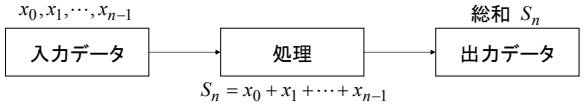
⇩

$$S_n = x_0 + x_1 + \cdots + x_{n-1}$$



総和のアルゴリズム

n個の数値$x_0, x_1, \cdots, x_{n-1}$の和$S_n$を求めなさい.



⇩ アルゴリズム

初期値
反復処理

$$\begin{cases} S_0 = 0 \\ S_{i+1} = S_i + x_i \quad (i = 0, \cdots, n-1) \end{cases}$$

繰返し範囲

（注）5章の配列の添字範囲を考慮し，
nn個の数値の添え字を0, . . , n-1としている.

総和のアルゴリズム（式の展開）

$$\begin{cases} S_0 = 0 \\ S_{i+1} = S_i + x_i \quad (i = 0, \cdots, n-1) \end{cases}$$

⇩ 展開

$$\begin{cases} S_0 = 0 \\ S_1 = S_0 + x_0 = x_0 \\ S_2 = S_1 + x_1 = x_0 + x_1 \\ S_3 = S_2 + x_2 = x_0 + x_1 + x_2 \\ \quad \vdots \\ S_n = S_{n-1} + x_{n-1} = x_0 + x_1 + x_2 + \cdots + x_{n-1} \end{cases}$$

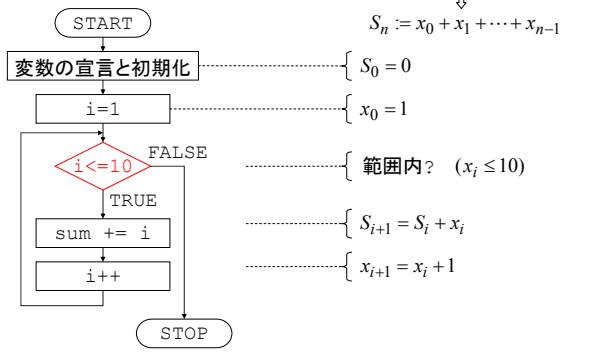
プログラム4.5

設計仕様　1から10までの和を求めるプログラムを作成しなさい

$$S = 1 + 2 + \cdots + 10$$

⇩

$$S_n := x_0 + x_1 + \cdots + x_{n-1}$$



増減算演算子，代入演算子

インクリメント演算子，デクリメント演算子（p.48-49）

増減算演算子	例	意味	別表現
++	i++	iの値を1増やす(increment)	i=i+1
--	i--	iの値を1減らす(decrement)	i=i-1

代入演算子	例	意味	別表現
=	a=b	aにbの値を代入する	
+=	a+=b	aの値をb増やす	a=a+b
-=	a-=b	aの値をb減らす	a=a-b
*=	a*=b	aの値をb倍する	a=a*b
/=	a/=b	aの値を1/b倍にする	a=a/b
%=	a%=b	aの値をa÷bの剰余にする	a=a%b

(p.50)

整数型のみ

（注） a+=b と a=+b とを混乱しないこと．

加算　代入

FAQ（増減算演算子，代入演算子）

Q 代入演算子，増減算演算子の必要性がわかりません．

A sum=sum+i を sum+=i と書くと，次の利点があります．

(1) 打ち込む文字が少なくて済む．

(2) その結果，タイプミスが少なくなる．

(3) 慣れると単純明快（sumにiを足す）．処理の意味が明確に限定される．

=と+は処理の対応範囲が広い．

（例えば，sum1=sum2+iの場合にも使える．）

計算式の集合

```
a=b+c
  a+=c （aとbが同じ変数の場合）
    a++ （cが1の場合）
```

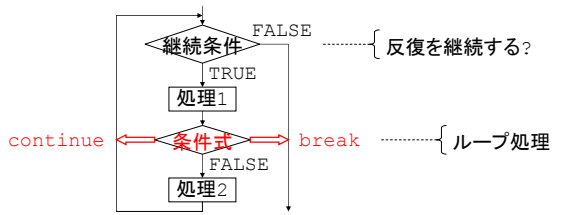
continueとbreak（4.6節参照）

continue（処理のスキップ）

```
while (····) {
  処理1
  if (····) continue;
  処理2
}
```

break（処理の中断）

```
while (····) {
  処理1
  if (····) break;
  処理2
}
```



多重反復の中断

多重反復内でbreakを用いた場合，最も内側の反復のみを中断する．

```
for (...){
  do {
    while (...) {
      処理1
      if (...) break;
      処理2
    }
  } while (...);
}
```

無限反復

無限反復

```
while (真) {
  処理;
}
```

⇒

```
while (1) {
  処理;
}
```

関係演算子	意味	使用例	演算結果	
			真の場合	偽の場合
==	等しい	a==b	1 (!0)	0
!=	等しくない	a!=b		
>=	以上	a>=b		
<=	以下	a<=b		
>	より大	a>b		
<	より小	a<b		

反復の相違点

制御構造	意味	反復回数	本体{}の処理回数
for ~	繰り返し	既知	0回以上（実行しない場合がある）
while ~	前判定	未知	
do ~ while	後判定		1回以上（必ず1回は実行する）

制御構造	用途の具体例
for ~	・配列の要素番号
while ~ do ~ while	・指定キーが押されるまで繰り返す ・指定値が入力されるまで繰り返す ・ある状態になるまで繰り返す ・それらの繰り返し回数を求める

同構造の置き換え


```
for （カウンタの初期化；継続条件；カウンタ増減式） {
    処理;
}
```

カウンタの範囲

反復回数制御が1ヶ所に集中 → 管理が容易

同構造

カウンタ以外の条件

```
カウンタの初期化;
while （継続条件） {
    処理;
    カウンタ増減式;
}
```

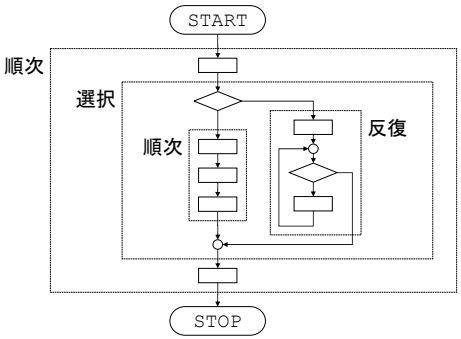
反復回数制御が分散 → 管理が面倒

反復回数の算出用

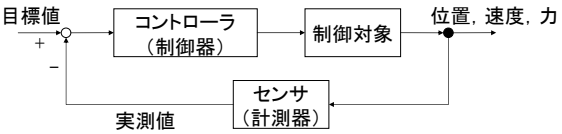
（注）

「while ~」に「初期化」と「増減式」を追加すると，「for ~」と同構造になる．しかし，whileは反復回数が未知であることを暗黙に訴えており，混用は読み手を混乱させる．

基本制御構造を用いた処理の分解



システム制御における反復処理



・1[ms]間隔で計測と制御

・反復回数は未知