



# pe-6. 配列

(Pascal プログラミング入門)

URL: <https://www.kkaneko.jp/cc/pascal/index.html>

金子邦彦



# 内容



- 例題 1. 月の日数

配列とは. 配列の宣言. 配列の添字.

- 例題 2. ベクトルの内積
- 例題 3. 棒グラフを描く
- 例題 4. Horner 法による多項式の計算
- 例題 5. エラトステネスのふるい

配列と繰り返し計算の関係

# 目標



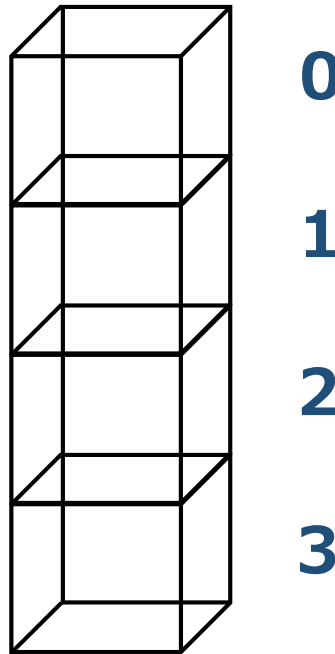
- **配列**とは何かを理解し, **integer, real の配列を使ったプログラム**を書けるようになる
- **配列と繰り返し文**を組み合わせて, **多量のデータ**を扱えるようになる

# 配列



- データの並びで，番号（添字）が付いている

添字





- プログラミングを行えるオンラインのサービス

<https://www.onlinegdb.com>

- ウェブブラウザを使う

- たくさんの言語を扱うことができる

Pascal, Python3, Java, C/C++, C#, JavaScript,  
R, アセンブリ言語, SQL など

- オンラインなので、「秘密にしたいプログラム」を扱うには十分な注意が必要

# Online GDB で Pascal を動かす手順



① ウェブブラウザを起動する

② 次の URL を開く

<https://www.onlinegdb.com>

A screenshot of a web browser's address bar. The address bar is a light gray rectangle with a magnifying glass icon on the left. Inside the bar, the text "https://www.onlinegdb.com" is displayed in a dark gray font. Below the address bar is a thin horizontal line, and below that is a larger, light gray rectangular area representing the rest of the browser window.



### ③ 「Language」 のところで、「Pascal」 を選ぶ

The screenshot shows the GDB Online interface. At the top, there is a navigation bar with buttons for Run, Debug, Stop, Share, Save, Beautify, and a Language dropdown menu. The Language dropdown menu is open, showing a list of programming languages. The 'Pascal' option is highlighted with a red box. The source code in the background is a C program that prints 'Hello World'.

```
1 - /*****  
2  
3 Welcome to GDB OnLine.  
4 GDB online is an online compiler and debugger tool for C, C++, Python  
5 C#, VB, Perl, Swift, Prolog, Javascript, Pascal, HTML, CSS, JS  
6 Code, Compile, Run and Debug online from anywhere in world.  
7  
8 *****/  
9 #include <stdio.h>  
10  
11 int main()  
12 {  
13     printf("Hello World");  
14  
15     return 0;  
16 }  
17
```



## 実行ボタン

The screenshot shows the GDB Online IDE interface. At the top, there is a toolbar with buttons for Run, Debug, Stop, Share, Save, Beautify, and a download icon. The 'Run' button is highlighted with a red box. Below the toolbar is the code editor, which contains the following Pascal code:

```
1 {  
2  
3 Welcome to GDB Online.  
4 GDB online is an online compiler and debugger tool for C, C++, Python, Java,  
5 C#, VB, Swift, Pascal, Fortran, Haskell, Objective-C, Assembly, HTML, CSS,  
6 Code, Compile, Run and Debug online from anywhere in world.  
7  
8 }  
9 program Hello;  
10 begin  
11     writeln ('Hello World')  
12 end.  
13
```

Below the code editor is the console output, which shows the compilation and execution process:

```
input  
Copyright (c) 1993-2017 by Florian Klaempfl and others  
Target OS: Linux for x86-64  
Compiling main.pas  
Linking a.out  
/usr/bin/ld.bfd: warning: link.res contains output sections; did you forget -T?  
12 lines compiled, 0.1 sec  
Hello World  
  
...Program finished with exit code 0  
Press ENTER to exit console.
```

## エディタ画面

プログラムを  
書き換えること  
ができる





# 例題 1 . 月の日数

- 年と月を読み込んで、日数を求めるプログラムを作る
  - うるう年の2月ならば 29
  - 日数を求めるために、サイズ12（1から12まで）の integer の 配列 を使う

例) 2021年11月 → 30



```
program sum;
var num : array [1..12] of integer;
var y, m : integer;
begin
  num[1] := 31; num[2] := 28; num[3] := 31; num[4] := 30;
  num[5] := 31; num[6] := 30; num[7] := 31; num[8] := 31;
  num[9] := 30; num[10] := 31; num[11] := 30; num[12] := 31;
  write('Please Enter y(year): ');
  readln(y);
  write('Please Enter m(month): ');
  readln(m);
  if (m=2) and (((y mod 400)=0) or (((y mod 100)<>0) and ((y mod 4)=0))) then begin
    writeln('number of days(', y, '/', m, ') is 29');
  end
  else begin
    writeln('number of days(', y, '/', m, ') is ', num[m]);
  end;
  readln
end.
```

配列への  
書き込み

配列からの  
読み出し

## 実行結果の例

```
Please Enter y(year) : 2022
Please Enter m(month) : 6
number of days(2022/6) is 30
```

## メモリ

num[m];

配列からの値の  
読み出し

num[1]	31
num[2]	28
num[3]	31
num[4]	30
num[5]	31
num[6]	30
num[7]	31
num[8]	31
num[9]	30
num[10]	31
num[11]	30
num[12]	31

# 配列の宣言



- **配列**には、**名前と型**（データの種類のこと）と**サイズ**がある
  - 整数データ                    **integer**
  - 浮動小数データ                **real**
- **配列**を使うには、**配列の使用をコンピュータに伝えること**（宣言）が必要

```
var num : array [1..12] of integer;
```

名前は  
num

配列の添字  
は1から12

整数  
データ



## array [ ] of の意味

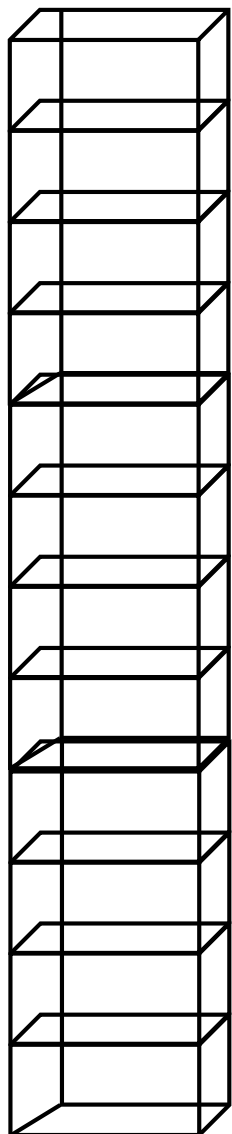
- [] の中に, **添字の範囲**を書く  
おのずと, 配列のサイズが決まる

**array [1..12] of integer のとき  
添字は, 1 から 1 2 まで**

# 配列の添字



## 添字



1

2

3

4

5

6

7

8

9

10

11

12

配列の中身を読み書きするときには、配列の名前と添字を書く

例) num[m]

添字

# 配列の読み書き



- 配列の名前と添字を書く

例)

```
num[1] := 31;
```

```
writeln('number of days(' y, '/', m, ') is', num[m] );
```



## 例題 2. ベクトルの内積



- ベクトル (1.9, 2.8, 3.7) と, ベクトル (4.6, 5.5, 6.4) の内積を表示するプログラムを作る
  - 2つのベクトルの内積の計算のために, サイズ 3 の配列を2つ使う



# ベクトルの内積

ベクトルの成分から内積を求める

$\vec{a} = (a_0, a_1, a_2)$   $\vec{b} = (b_0, b_1, b_2)$  のとき

$$\vec{a} \cdot \vec{b} = a_0 b_0 + a_1 b_1 + a_2 b_2$$



```
program sum;
var u, v : array [0..2] of real;
var i : integer;
var ip : real;
begin
  u[0] := 1.9; u[1] := 2.8; u[2] := 3.7;
  v[0] := 4.6; v[1] := 5.5; v[2] := 6.4;
  ip := 0;
  for i := 0 to 2 do begin
    ip := ip + u[i] * v[i];
  end;
  writeln('product =', ip:8:3 );
  readln
end.
```

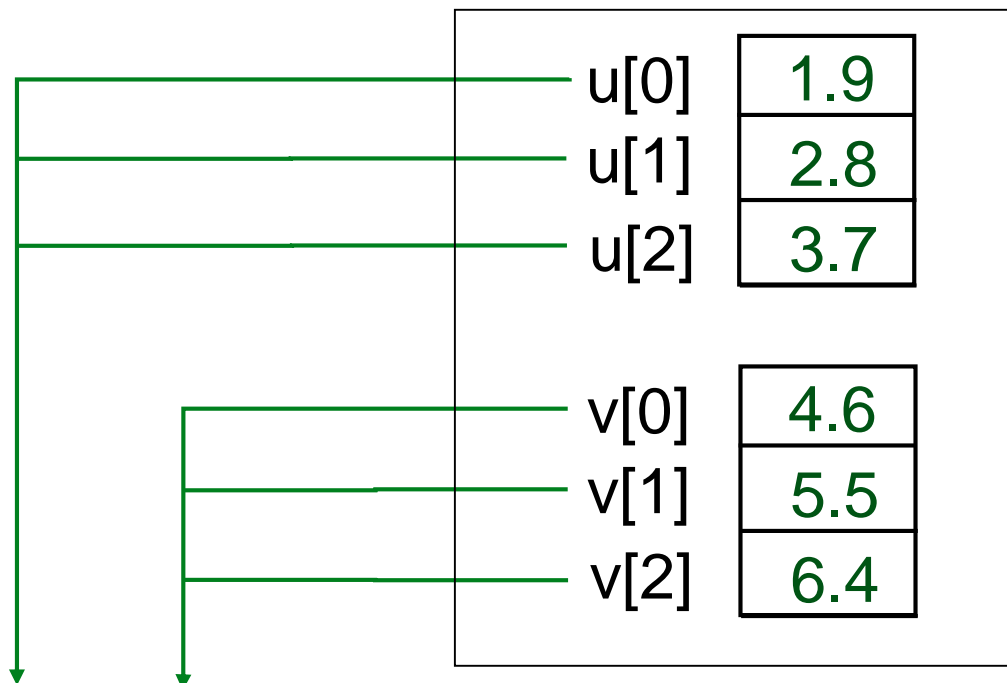
配列への  
書き込み

配列からの  
読み出し

## 実行結果の例

```
product = 47.820
```

## メモリ



```
ip := ip + u[i]*v[i];
```

配列からの  
読み出し

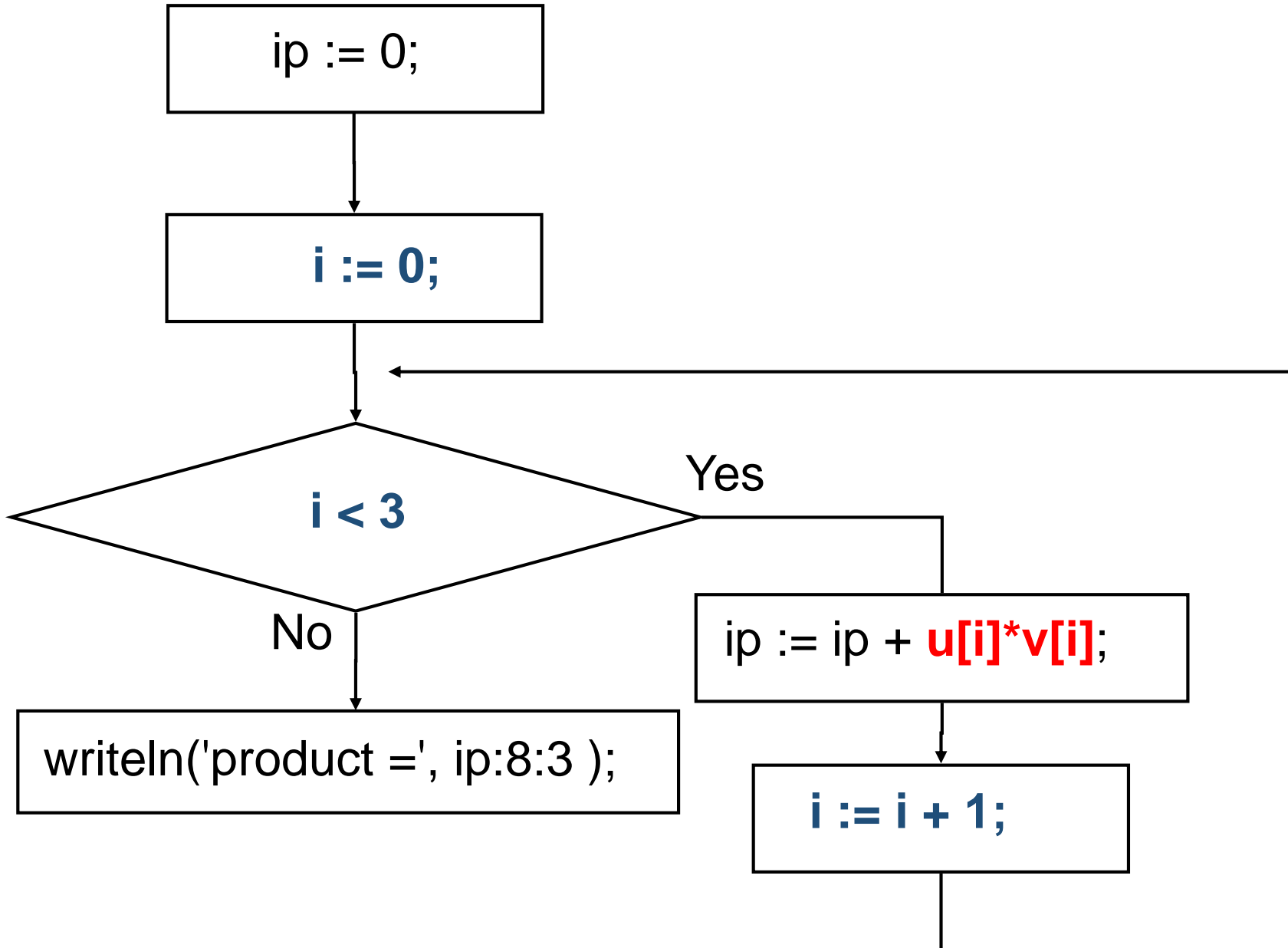
```
var u, v : array [0..2] of real;
```

名前は  
u と v

添字は  
0 から 2 まで

浮動小数  
データ

# プログラム実行順



# ベクトルの内積



	i の値	繰り返し条件式 が成り立つか	ip の値
繰り返し 1 回目	$i = 0$	$i < 3$ が成り立つ	$ip := ip + u[0] * v[0];$ つまり ip の値は $u[0]*v[0]$
繰り返し 2 回目	$i = 1$	$i < 3$ が成り立つ	$ip := ip + u[1] * v[1];$ つまり ip の値は $u[0]*v[0] + u[1]*v[1]$
繰り返し 3 回目	$i = 2$	$i < 3$ が成り立つ	$ip := ip + u[2] * v[2];$ つまり ip の値は $u[0]*v[0] + u[1]*v[1]$ $+u[2]*v[2]$
繰り返し 4 回目	$i = 3$	$i < 3$ が成り立たない	





## 例題 3. 棒グラフを描く

- 整数の配列から, その棒グラフを表示するプログラムを作る.
  - ループの入れ子で, 棒グラフの表示を行う



```
program sum;  
var a : array [1..7] of integer;  
var i, j : integer;  
begin  
  a[1] := 6; a[2] := 4; a[3] := 7; a[4] := 1;  
  a[5] := 5; a[6] := 3; a[7] := 2;  
  for i := 1 to 7 do begin  
    for j := 1 to a[i] do begin  
      write('*');  
    end;  
    writeln;  
  end;  
  readln  
end.
```

配列への  
書き込み

配列からの  
読み出し

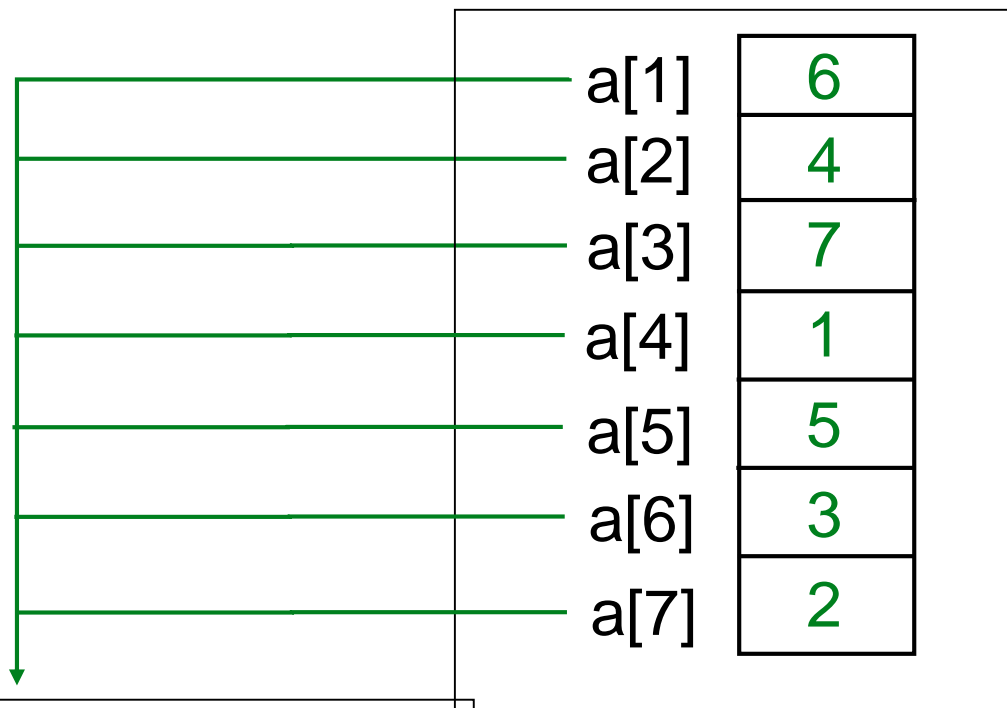
# 棒グラフを描く



## 実行結果の例

```
*****  
****  
*****  
*  
*****  
***  
**
```

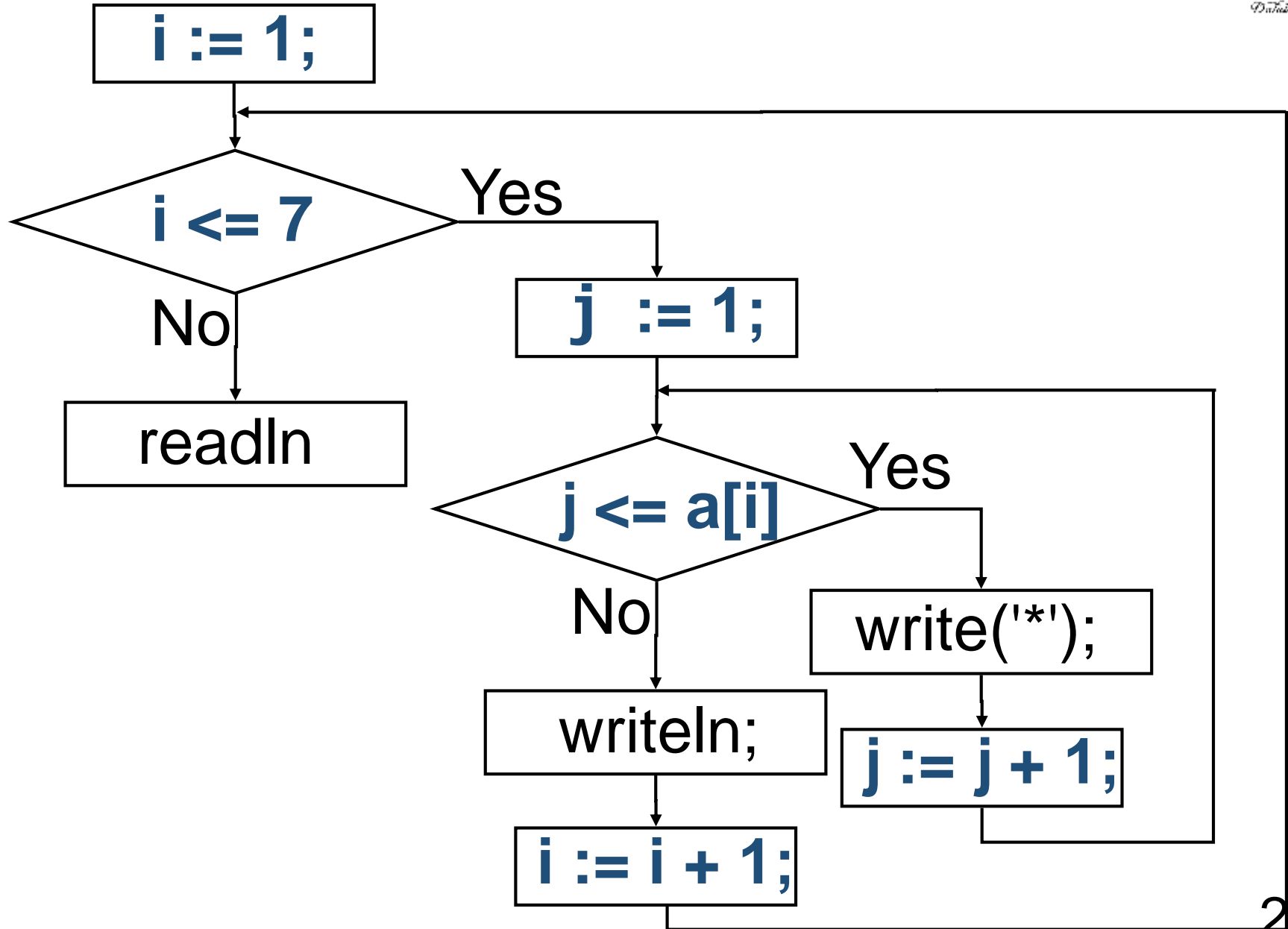
## メモリ



```
for j := 1 to a[i] do begin
```

配列からの  
読み出し

# プログラム実行順



# 例題 4 . Horner 法による多項式の計算



## • n次の多項式

$$f(x) = a_0 + a_1 \cdot x + a_2 \cdot x^2 + \cdot \cdot \cdot + a_n \cdot x^n$$

について, **次数 n** と, **係数 a<sub>0</sub> から a<sub>n</sub>** を読み込んで, **f(x)** を計算するプログラムを作る

- まず n を読み込んで, その後に a<sub>0</sub> から a<sub>n</sub> を読み込む . 最後に x を読み込む.
- 次ページで説明する Horner法を使う
- 読み込んだ係数は, いったん配列に格納する. n は高々 20 とする.

# Horner法



$$\begin{aligned} f(x) &= a_0 + a_1 \cdot x + a_2 \cdot x^2 + \dots + a_n \cdot x^n \\ &= a_0 + (a_1 + (a_2 + \dots + (a_{n-1} + a_n \cdot x) x \cdot \\ &\quad \dots) x) x \end{aligned}$$

$$\begin{aligned} \text{例えば, } & 5 + 6x + 3x^2 \\ &= 5 + (6 + 3x) x \end{aligned}$$

計算手順

- ①  $a_n$
- ②  $a_{n-1} + a_n \cdot x$
- ③  $a_{n-2} + (a_{n-1} + a_n \cdot x) x$
- ... (a<sub>0</sub> まで続ける)



```
program sum;
var a : array [0..20] of real;
var i, n : integer;
var x, y : real;
begin
  write('Please Enter n: ');
  readln(n);
  for i := 0 to n do begin
    write('Please Enter a[' , i, ' ] : ');
    readln(a[i]);
  end;
  write('Please Enter x: ');
  readln(x);
  y := a[n];
  i := n - 1;
  while i >= 0 do begin
    y := y * x + a[i];
    i := i - 1;
  end;
  writeln('y =', y:8:3 );
  readln
end.
```

配列への  
書き込み

配列からの  
読み出し

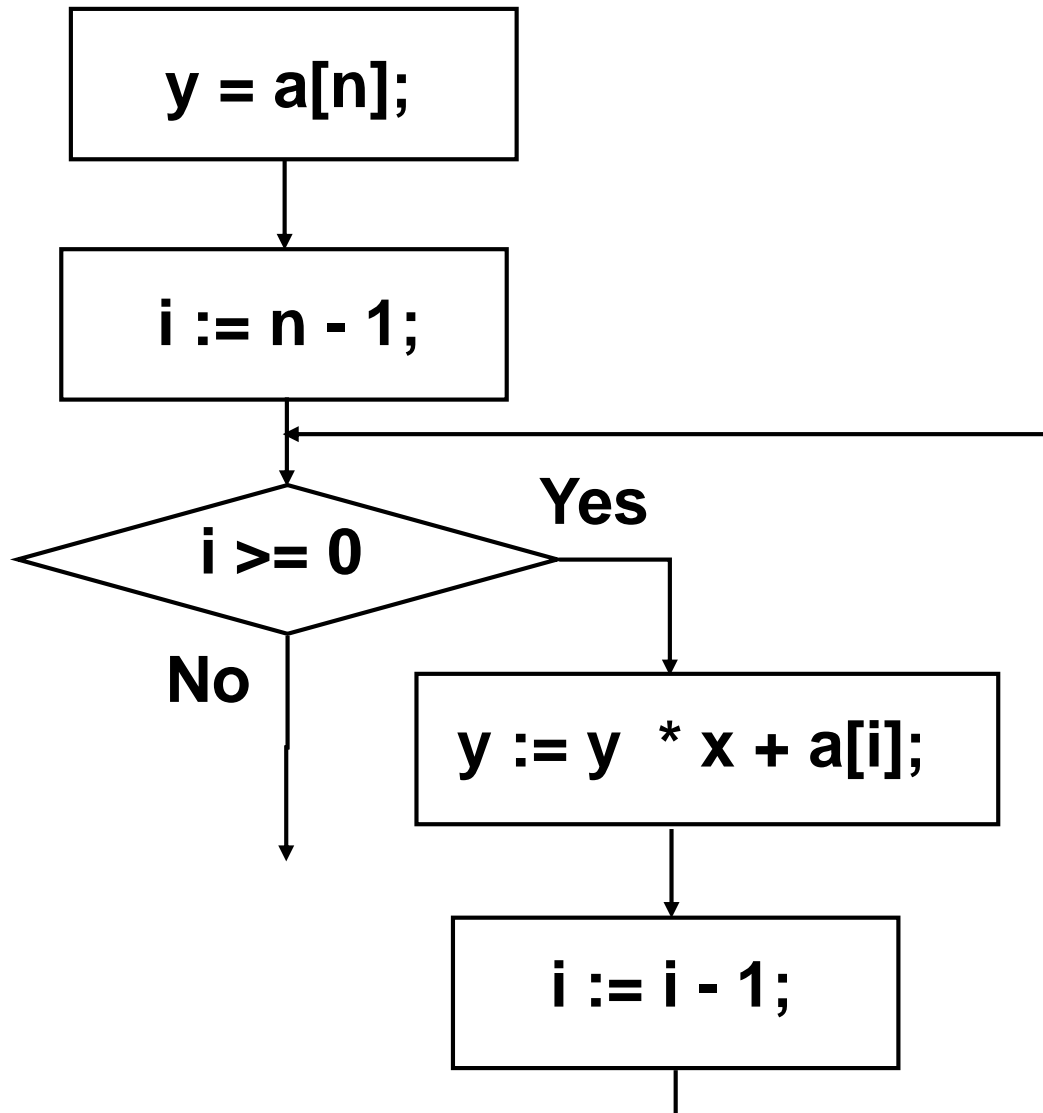


# 実行結果の例



```
Please Enter n: 2
Please Enter a[0] : 5
Please Enter a[1] : 6
Please Enter a[2] : 3
Please Enter x: 3
y = 50.000
□
```

# Horner 法



## 例題 5. エラトステネスのふるい



- 「エラトステネスのふるい」の原理に基づいて**100以下の素数**を求め、結果を表示するプログラムを作成する
  - 100以下の素数を求めるために、2から100までの配列を使う
  - 配列には、添字が素数なら**1**、そうでなければ**0**をセットする

# エラトステネスのふるい (1/4)



2   3   4   5   6   7   8   9   10   11   . . .

          ○          ○          ○          ○

$2 \times 2$            $2 \times 3$            $2 \times 4$            $2 \times 5$

まず、2の倍数を消す

# エラトステネスのふるい (2/4)



2 (3) 4 5 (6) 7 8 (9) 10 11 . . .

$3 \times 2$   $3 \times 3$

次に, 3の倍数を消す

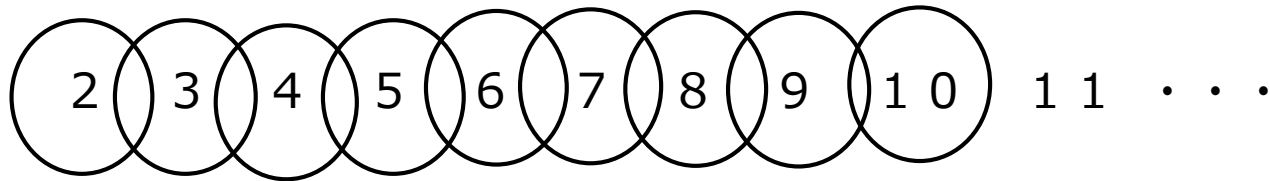
# エラトステネスのふるい (3/4)



2 3 4 5 6 7 8 9 10 11 . . .  
5 × 2

次に, 5 の倍数を消す  
(「4 の倍数」は考えない.  
それは, 「4」がすでに消えているから)

# エラトステネスのふるい (4/4)



以上のように, 2, 3, 5, 7 の倍数を消す.

10 (これは100の平方根) を超えたら, この操作を止める  
(100以下で, 11, 13...の倍数はすでに消えている)



```
program sum;  
var p : array [2..100] of integer;  
var n, i : integer;  
begin
```

```
  for i := 2 to 100 do begin  
    p[i] := 1;  
  end;
```

a[2] から a[100] までを  
「1」にセット  
まず「2」の倍数から

```
  n := 2;
```

```
  while n <= sqrt( 100 ) do begin  
    i := 2;
```

```
    while ( n * i ) <= 100 do begin  
      p[ n * i ] := 0;  
      i := i + 1;  
    end;
```

n が 100 を超えたら終わり  
n の倍数を「消す」  
(n が 100 以上になったら終わり)

```
    n := n + 1;  
    while ( p[n] = 0 ) and ( n <= sqrt(100) ) do begin  
      n := n + 1;  
    end;
```

n の「次」の  
素数を探す

```
  end;
```

```
  for i := 2 to 100 do begin  
    if p[i] = 1 then begin  
      write( i, ',' );  
    end;  
  end;
```

求めた素数の表示

```
  readln
```

```
end.
```



# 実行結果の例



2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97,

# 演習 1. 1000 までの素数



- エラトステネスのふるいを使って1000までの数の素数を求めるプログラムを作りなさい  
例題 5 のプログラムを書き換えなさい