

関数とポインタ

三池 克明

ポインタを引数とする関数について解説します。

— 目 次 —

1.	ポインタと関数.....	1
1.1.	int 型変数の値を交換する関数.....	1
1.2.	関数 twicearray () の引数をポインタにしてみる	4
2.	ポインタを引数にした関数.....	8
2.1.	関数 intsaidai () の引数をポインタにしてみる	8
2.2.	関数 goukei () の引数をポインタにしてみる	11
2.3.	関数 mojinagasa () の引数をポインタにしてみる	14
2.4.	メッセージ付の整数入力.....	16
3.	演習問題.....	19

1. ポインタと関数

関数の引数としてポインタを使うことで、引数そのものを操作できるようになります。

1.1. int 型変数の値を交換する関数

以下のソースプログラムは2つの int 型変数の値を交換する関数 `intswap()` とその動作を確認するプログラムです。

まずは以下のソースプログラムを作成し、コンパイル・実行させてみましょう。

`intswap1.cpp`

```
1: #include <stdio.h>
2:
3: void intswap(int, int);
4:
5: main()
6: {
7:     int    a, b;
8:
9:     printf("a = ");
10:    scanf("%d", &a);
11:    printf("b = ");
12:    scanf("%d", &b);
13:
14:    intswap(a, b);
15:
16:    printf("a = %d, b = %d\n", a, b);
17: }
18:
19: void intswap(int x, int y)
20: {
21:     int    work;
22:
23:     work = x;
24:     x = y;
```

```
25:         y = work;
26:     }
```

実行例

```
a = 10
b = 5
a = 10, b = 5
```

おや？ 交換されませんね。
理由は分かりますよね。

続いて以下のソースプログラムを入力し、コンパイル、実行させてみましょう。

intswap2.cpp

```
1: #include <stdio.h>
2:
3: void intswap(int *, int *);
4:
5: main()
6: {
7:     int    a, b;
8:
9:     printf("a = ");
10:    scanf("%d", &a);
11:    printf("b = ");
12:    scanf("%d", &b);
13:
14:    intswap(&a, &b);
15:
16:    printf("a = %d, b = %d\n", a, b);
17: }
18:
19: void intswap(int *x, int *y)
20: {
21:     int    work;
22:
23:     work = *x;
24:     *x = *y;
```

```
25:     *y = work;
26: }
```

実行例

```
a = -4
b = 5
a = 5, b = -4
```

ポインタを引数にすることで値の交換が実現できました。
このように引数そのものを操作したい場合は、

- 呼び出す際は引数の変数名に&を付ける。
- 関数の引数はポインタにする

としましょう。

1.2. 関数 `twicearray()` の引数をポインタにしてみる

まずは以前に作った関数 `twicearray()` の動作を確認してみましょう。

twicearray.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: void twicearray(double [], int);
6:
7: main()
8: {
9:     double  a[SIZE];
10:    int      i;
11:
12:    for(i = 0; i < SIZE; i++){
13:        printf("a[%d] = ", i);
14:        scanf("%lf", &a[i]);
15:    }
16:
17:    twicearray(a, SIZE);
18:
19:    for(i = 0; i < SIZE; i++){
20:        printf("a[%d] = %f¥n", i, a[i]);
21:    }
22: }
23:
24: void twicearray(double a[], int  n)
25: {
26:     int      i;
27:
28:     for(i = 0; i < n; i++){
29:         a[i] *= 2;
30:     }
31: }
```

実行例

```
a[0] = 0.9
a[1] = 1
a[2] = 1.1
a[3] = 1.2
a[4] = 1.3
a[0] = 1.800000
a[1] = 2.000000
a[2] = 2.200000
a[3] = 2.400000
a[4] = 2.600000
```

関数 `twicearray()` の引数をポインタに修正したのが以下のソースプログラムです。

入力し、コンパイル・実行させてみましょう。

twicearray2.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: void twicearray(double *, int);
6:
7: main()
8: {
9:     double  a[SIZE];
10:    int      i;
11:
12:    for(i = 0; i < SIZE; i++) {
13:        printf("a[%d] = ", i);
14:        scanf("%lf", &a[i]);
15:    }
16:
17:    twicearray(a, SIZE);
18:
19:    for(i = 0; i < SIZE; i++) {
```

```
20:             printf("a[%d] = %f\n", i, a[i]);
21:         }
22:     }
23:
24: void twicearray(double *a, int n)
25: {
26:     int i;
27:
28:     for(i = 0; i < n; i++){
29:         *(a + i) *= 2;
30:     }
31: }
```

実行例

```
a[0] = 0.9
a[1] = 1
a[2] = 1.1
a[3] = 1.2
a[4] = 1.3
a[0] = 1.800000
a[1] = 2.000000
a[2] = 2.200000
a[3] = 2.400000
a[4] = 2.600000
```

どうやら問題ないようです。

— a[i] と *(a + i) —

C コンパイラはソースプログラムをコンパイルするときに配列である a[i] という書式をポインタである *(a + i) に置き換えています。

よってどちらの形式で記述しても問題ありません。

ですが配列であれば a[i] 形式、ポインタであれば *(a + i) 形式で記述するのが通例のようです。

本書でもこのような通例に従います。

2. ポインタを引数にした関数

記述の仕方が違うだけで、実際は配列を引数にした関数とほぼ同じです。

2.1. 関数 `intsaidai()` の引数をポインタにしてみる

`int` 型配列の最大値を返す関数 `intsaidai()` を以前作りました。
ここで再度確認してみましょう。

intsaidai.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: int intsaidai(int [], int);
6:
7: main()
8: {
9:     int    a[SIZE];
10:    int    i;
11:
12:    for(i = 0; i < SIZE; i++) {
13:        printf("a[%d] = ", i);
14:        scanf("%d", &a[i]);
15:    }
16:
17:    printf("最大値は%d です。¥n", intsaidai(a, SIZE));
18: }
19:
20: int intsaidai(int array[], int n)
21: {
22:     int    i, saidai;
23:
24:     saidai = array[0];
25:     for(i = 1; i < n; i++) {
26:         if(saidai < array[i]) {
27:             saidai = array[i];
```

```
28:         }
29:     }
30:
31:     return saidai;
32: }
```

実行例

```
a[0] = -5
a[1] = -8
a[2] = 20
a[3] = 45
a[4] = 9
最大値は 45 です。
```

そして関数 `intsaidai()` の引数をポインタに修正したのが以下のソースプログラムです。

入力し、コンパイル・実行させてみましょう。

intsaidai2.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: int intsaidai(int *, int);
6:
7: main()
8: {
9:     int    a[SIZE];
10:    int    i;
11:
12:    for(i = 0; i < SIZE; i++) {
13:        printf("a[%d] = ", i);
14:        scanf("%d", &a[i]);
15:    }
16:
17:    printf("最大値は%d です。¥n", intsaidai(a, SIZE));
18: }
```

```
19:
20: int intsaidai(int *array, int n)
21: {
22:     int    i, saidai;
23:
24:     saidai = *array;
25:     for(i = 1; i < n; i++){
26:         if(saidai < *(array + i)){
27:             saidai = *(array + i);
28:         }
29:     }
30:
31:     return  saidai;
32: }
```

実行例

a[0] = -5

a[1] = -8

a[2] = 20

a[3] = 45

a[4] = 9

最大値は 45 です。

特に問題は無いようです。

2.2. 関数 goukei () の引数をポインタにしてみる

次は関数 goukei () の引数をポインタにしてみましょう。

まずは以前に作った関数 goukei () とその動作確認プログラムの確認です。

goukei.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: double goukei(double [], int);
6:
7: main()
8: {
9:     double  data[SIZE];
10:    int      i;
11:
12:    for(i = 0; i < SIZE; i++){
13:        printf("data[%d] = ", i);
14:        scanf("%lf", &data[i]);
15:    }
16:
17:    printf("合計 = %g\n", goukei(data, SIZE));
18: }
19:
20: double goukei(double array[], int n)
21: {
22:     double  sum;
23:     int      i;
24:
25:     sum = array[0];
26:     for(i = 1; i < n; i++){
27:         sum += array[i];
28:     }
29:
30:     return  sum;
31: }
```

実行例

```
data[0] = 10.5
data[1] = -222
data[2] = 55.235
data[3] = 33.3
data[4] = 83.2
合計 = -39.765
```

そして関数 `goukei()` の引数をポインタに修正したのが以下のソースプログラムです。

入力し、コンパイル・実行させてみましょう。

goukei2.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    5
4:
5: double goukei(double *, int);
6:
7: main()
8: {
9:     double  data[SIZE];
10:    int      i;
11:
12:    for(i = 0; i < SIZE; i++) {
13:        printf("data[%d] = ", i);
14:        scanf("%lf", &data[i]);
15:    }
16:
17:    printf("合計 = %g\n", goukei(data, SIZE));
18: }
19:
20: double goukei(double *array, int n)
21: {
22:     double  sum;
23:     int      i;
24:
25:     sum = *array;
```

```
26:         for(i = 1; i < n; i++){
27:             sum += *(array + i);
28:         }
29:
30:         return sum;
31:     }
```

実行例

```
data[0] = 10.5
data[1] = -222
data[2] = 55.235
data[3] = 33.3
data[4] = 83.2
合計 = -39.765
```

特に問題は無いようです。

2.3. 関数 `mojinagasa()` の引数をポインタにしてみる

そして今度は関数 `mojinagasa()` の引数をポインタにしてみましょう。

まずは以前に作ったプログラムの確認です。

どこを修正すれば良いのかはもうわかりますよね？

mojinagasa.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    128
4:
5: int mojinagasa(char []);
6:
7: main()
8: {
9:     char    s[SIZE];
10:
11:     printf("文字列を入力して下さい\n");
12:     gets(s);
13:
14:     printf("入力した文字列は%d 文字です。 \n", mojinagasa(s));
15: }
16:
17: int mojinagasa(char str[])
18: {
19:     int    i;
20:
21:     for(i = 0; str[i] != '\0'; i++) {
22:     }
23:
24:     return i;
25: }
```

実行例

文字列を入力して下さい

moji

入力した文字列は 4 文字です。

そして関数 `mojinagasa()` の引数をポインタに修正したのが以下のソースプログラムです。

入力し、コンパイル・実行してみましよう。

mojinagasa2.cpp

```
1: #include <stdio.h>
2:
3: #define SIZE    128
4:
5: int mojinagasa(char *);
6:
7: main()
8: {
9:     char    s[SIZE];
10:
11:     printf("文字列を入力して下さい\n");
12:     gets(s);
13:
14:     printf("入力した文字列は%d 文字です。 \n", mojinagasa(s));
15: }
16:
17: int mojinagasa(char *str)
18: {
19:     int    i;
20:
21:     for(i = 0; *(str + i) != '\0'; i++) {
22:     }
23:
24:     return i;
25: }
```

実行例

文字列を入力して下さい

moji

入力した文字列は 4 文字です。

特に問題は無いようです。

2.4. メッセージ付の整数入力

本書ではキーボードで入力を促す処理を記述するとき、ほとんどの場合は

```
printf("a = ");  
scanf("%d", &a);
```

としていました。

今後も頻繁に使うことがあると思いますので、これを関数化してみましょう。
仕様は以下のようにします。

<関数 `intinput()` の仕様>

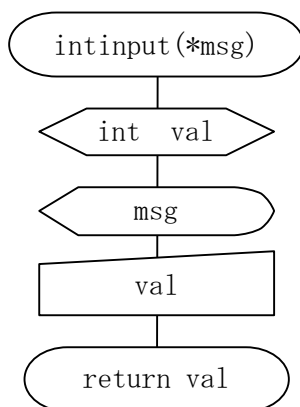
1. 外部向けの仕様

プロトタイプ	<code>int intinput(char *msg)</code>	
引数	<code>char *msg</code>	入力を促すメッセージ
返値	<code>int</code>	入力された値
【解説】 入力したメッセージを表示し、入力した整数を返す。		
【例】 <pre>a = intinput("a = ");</pre> 「a = 」とメッセージを表示し、入力された整数を a に代入します。		

2. 関数内で使用する変数

変数名	型	概要
<code>val</code>	<code>int</code>	入力された整数

3. フローチャート



4. ソースコード

```
int intinput(char *msg)
{
    int    val;

    printf(msg);
    scanf("%d", &val);

    return val;
}
```

<以上>

それでは intinput() の動作確認プログラムを作りましょう。
以下のプログラムを入力し、コンパイル・実行します。

```
1: #include <stdio.h>
2:
3: int intinput(char *);
4:
5: main()
6: {
7:     int    x, y;
8:
9:     x = intinput("x の値は? : ");
10:    y = intinput("y の値は? : ");
11:
12:    printf("x + y = %d+%d = %d¥n", x, y, x + y);
13: }
14:
15: int intinput(char *msg)
16: {
17:     int    val;
18:
19:     printf(msg);
20:     scanf("%d", &val);
21:
22:     return val;
23: }
```

実行例

```
x の値は? : 30
y の値は? : -20
x + y = 30-20 = 10
```

どうやら問題は無いようです。

3. 演習問題

13-1. 「intsaidai2.cpp」を参考に以下の仕様を満たす関数 `intsaisyo()` を完成させ、その動作を確認するプログラム「ensyu13-1.cpp」を作りなさい。

プロトタイプ	<code>int intsaisyo(int *array, int n)</code>	
引数	<code>int *array</code>	配列の先頭を指すポインタ
	<code>int n</code>	ポインタ <code>array</code> が指す配列の大きさ
返値	<code>int</code>	配列の最小値
【解説】 配列の最小値を求める。		
【例】 <pre>min = intsaisyo(a, 100);</pre>		
配列 <code>a[]</code> の最小値を求め、 <code>max</code> に代入します。		

13-2. 「goukei2.cpp」を参考に以下の仕様を満たす関数 `heikin()` を完成させ、その動作を確認するプログラム「ensyu13-2.cpp」を作りなさい。

プロトタイプ	<code>double heikin(double *array, int n)</code>	
引数	<code>double *array</code>	配列の先頭を指すポインタ
	<code>int n</code>	ポインタ <code>array</code> が指す配列の大きさ
返値	<code>double</code>	配列の平均
<p>【解説】 配列の平均を求める。</p> <p>【例】</p> <pre style="text-align: center;">average = heikin(a, SIZE);</pre> <p>配列 <code>a[]</code> の平均を求め、<code>average</code> に代入します。</p>		

13-3. 「mojinagasa2.cpp」を参考に以下の仕様を満たす関数 `sujinagasa()` を完成させ、その動作を確認するプログラム「ensyu13-3.cpp」を作りなさい。

プロトタイプ	<code>int sujinagasa(char *str)</code>	
引数	<code>double *str</code>	文字列の先頭を指すポインタ
返値	<code>int</code>	ポインタ <code>str</code> が指す文字列に含まれる半角数字の文字数
【解説】 引数の文字列に含まれる半角数字の文字数を求める。		
【例】 <pre>n = alphanagasa("1234abcd56ef");</pre> 文字列"1234abcd56ef"に含まれる半角数字の文字数(本例では6)を、nに代入します。		

13-4. 「intinput.cpp」を参考に以下の仕様を満たす関数 `doubleinput()` を完成させ、その動作を確認するプログラム「ensyu13-4.cpp」を作りなさい。

プロトタイプ	<code>double doubleinput(char *msg)</code>	
引数	<code>char *msg</code>	入力を促すメッセージ
返値	<code>double</code>	入力された値
【解説】 入力したメッセージを表示し、入力した実数を返す。		
【例】 <pre>b = doubleinput("b = ");</pre> 「b = 」とメッセージを表示し、入力された実数を b に代入します。		