15. 便利な方法

(1) 領域サイズが固定的でないとき

たとえば，以下のように配列宣言されているものとしましょう。

```
int intArea[100];
double dblArea[100];
```

いま，これらの配列が同じサイズの領域だとします。これらのサイズを 100 から 200 に変更するとき，「100」と書かれた 2 箇所を「200」に変更しなければなりません。さらに，プログラム中の関連する繰返し回数や判定部分もすべて変更しなければなりません。

一方，マクロ定義を使って，

```
#define MAXAREA 100

int intArea[MAXAREA];
double dblArea[MAXAREA];
```

としておくと，マクロ定義部分だけを変更すれば構いません。

(2) マクロの定義

ソースプログラムのある範囲が同じ形をとるとき，マクロを使うと便利です。たとえば，以下のように定義して，

```
#define swap(type,x,y)
  do{type t;t=x;x=y;y=t;}while(0)
```

次のようにソースプログラム中に記述します。

```
swap(int A, B);
```

すると，次のように記述されたものとみなされます。

```
do {int t; t=A; A=B; B=t;} while(0)
```

関数の呼び出しに似ていますが，ソースプログラム中に展開されます。関数呼出しのオーバヘッドがないので，高速になります。一方，

```
#define swap(type, x,y)
      {type t; t=x; x=y; y=t;}
```

と定義すると，

```
if(C1) swap(int, A, B);
else   swap(int, A, C);
```

を展開したとき，

```
if(C1) { int t; t=A; A=B; B=t; };
else   { int t; t=A; A=C; C=t; };
```

となり，コンパイルエラーになります。このエラーをなくすためには，

```
if(C1) swap(int, A, B)
else   swap(int, A, C)
```

と記述せざるをえませんが，C のスタイルではありません。最初の定義だと，

```
if(C1) do{ int t; t=A; A=B; B=t;} while(0);
else   do{ int t; t=A; A=C; C=t;} while(0);
```

となりエラーとなりません。便宜的に do_While の構文を使ってエラーを避けるわけです。

【発展】

ちょっと C らしからぬ記述になってしまいますが，こんなマクロも定義できます。

```
#define And &&
#define Or  ||
       ⋮
if (a>0 And b>0) printf("両方共に正");
if (a>0  Or b>0) printf("どちらかが正");
```