入門講習会 未分類

1. マクロ

マクロとは，あるテキストを別のテキストに置換する機能のことである。ソースコードで定数を表現したい場合や，何かを省略して書きたい場合に用いられる。
繰り返すが，マクロはテキストの置換機能である。この置換がどこで行われるのかというと，コンパイルの前である。テキスト置換という性質上，しばしばバグを引き起こす。使い方に注意が必要である。
マクロは#define とよばれる命令によって書かれる。#include の直後に書かれることがほとんどである。
マクロにはオブジェクト形式マクロと関数形式マクロの 2 種類がある。以下，それぞれ述べる。

オブジェクト形式マクロは，文字を値に置換するためのマクロである。定数を文字で表したい場合に使われることが多い。以下の形式で書く。

```
#define マクロ名 置き換えたい文字列
```

例を示す。PI を 3.14, INF を 1000000000, HELLO を printf, MAX_N を 1000 に置換するマクロは次のように書ける。

```
#define PI 3.14
#define INF 1000000000
#define HELLO printf("Hello!¥n")
#define MAX_N 1000
```

これらを利用したプログラムを以下に示す。

```
#include <stdio.h>

#define PI 3.14
#define INF 1000000000
#define HELLO printf("Hello!¥n")
#define MAX_N 1000

int main(void)
{
    int a[MAX_N];
    double d;
```

```
    d = PI;
    a[0] = INF;

    HELLO;
    printf("%d %f", a[0], d);

    return 0;
}
```

実行結果は以下のようになる。

```
Hello!
1000000000 3.140000
```

コンパイル前にテキストが置換されると述べた。実際, PI が 3.14, INF が 1000000000, HELLO が printf, MAX_N が 1000 に置換され, 以下のソースコードがコンパイルされる。

```
#include <stdio.h>

int main(void)
{
    int a[1000];
    double d;

    d = 3.14;
    a[0] = 1000000000;

    printf("Hello!¥n");
    printf("%d %f", a[0], d);

    return 0;
}
```

関数形式マクロは, 引数を取ってテキスト置換を行うマクロである。「関数として書くには単純すぎる」「関数呼び出し時間を削減したい」場合に使われることが多いと思われる。以下の書式で書く。

```
#define マクロ名(引数) 置き換えたいテキスト
```

例を示す。a と b の掛け算処理，文字列の出力処理に置換するマクロ MUL と SHOW は以下のように書ける。

```
#define MUL(a, b) ((a) * (b))
#define SHOW(str) printf(str)
```

MUL について，なぜ a*b と書かずに((a)*(b))と書くのか疑問に思うかもしれないが，これについては後に述べる。

マクロを利用したプログラムを以下に示す。

```
#include <stdio.h>

#define MUL(a, b) ((a) * (b))
#define SHOW(str) printf(str)

int main(void)
{
    printf("%d¥n", MUL(-3, 2));
    SHOW("Hello¥n");
    return 0;
}
```

実行結果は以下のようになる。

```
-6
Hello
```

コンパイル前にテキストが置換されると述べた。実際，MUL(-3, 2)は((-3) * (-2))，SHOW(“Hello¥n”)は printf(“Hello¥n”)に置換され，以下のソースコードとしてコンパイルされることになる。

```
#include <stdio.h>

int main(void)
{
    printf("%d¥n", ((-3) * (2)));
    printf("Hello¥n");
    return 0;
}
```

ここで，

```
#define MUL(a, b) ((a) * (b))
```

について詳しく見てみよう。こう定義しておけば，例えば「2 + 3 の結果と 4 + 2 の結果の積を求めたい」場合に，MUL(2 + 3, 4 + 2)と書ける。実際，

```
MUL(2 + 3, 4 + 2)
```
は
```
((2 + 3) * (4 + 2))
```
に置換される。これにより, ((2 + 3) * (4 + 2)) = 5 * 6 = 30 が得られる。
対して, MUL が次のように書かれていた場合はどうだろうか。
```
#define MUL(a, b) a * b
```
マクロはテキスト置換機能であることを思いだそう。すると,
```
MUL(2 + 3, 4 + 2)
```
は,
```
2 + 3 * 4 + 2
```
に置換される。計算の優先順位を考えて, 2 + 3 * 4 + 2 = 2 + 12 + 2 = 16 が得られる。
もし MUL(2 + 3, 4 + 2)を「2 + 3 の結果と 4 + 2 の結果の積を求めたい」という目的で用いたなら, この結果は誤りである。
このように, マクロは「テキスト置換」という性格を持つがために, 一見正しく書けていたとしても, 実際に使ってみると意図しない結果を招いてしまうようなことがある。できるだけ正しいマクロを書くための注意点がいくつかあるのだが, これについては長くなってしまうため説明しない。他人のマクロを色々眺めてみて, 便利そうだったら真似するのが最も安全かと思われる。

[補足]
マクロ名を大文字で書いていたが, 別に小文字でも構わない。マクロであることが明確に分かるようにするために大文字で書くことがある。しかしこれは好みの問題で, 小文字でマクロを書く人もいる。

2. sizeof 演算子
sizeof 演算子を用いると, (厳密ではないが)型のバイト数や配列の要素数を取得できる。以下のような書式で書く。

```
sizeof(型・変数名・配列)
```

以下のようなプログラムを作成して, sizeof 演算子の結果を出力してみよう。

```
#include <stdio.h>

#define MAX_N 1000

int main(void)
{
```

```
    int a[MAX_N];

    printf("int: %d¥n", sizeof(int));
    printf("char: %d¥n", sizeof(char));
    printf("long long: %d¥n¥n", sizeof(long long));

    printf("a[0]: %d¥n", sizeof(a[0]));
    printf("a: %d¥n", sizeof(a));
    printf("a_size: %d¥n", sizeof(a) / sizeof(a[0]));

    return 0;
}
```

実行結果は以下のようになる。

```
int: 4
char: 1
long long: 8
a[0]: 4
a: 4000
a_size: 1000
```

sizeof(int)により int 型のバイト数である 4 が返ってくる。sizeof(char)により char 型のバイト数である 1 が返ってくる。sizeof(long long)により long long 型のバイト数である 8 が返ってくる。sizeof(a[0])により，a[0](=int 型)のバイト数である 4 が返ってくる。

ここで注意したいのは，sizeof(配列)と書くと，返ってくる値は(各要素のサイズ)×(要素数)であることだ。なので，sizeof(a)によって返ってくる値は 4×1000＝4000 である。要素数を出力したいなら，それを sizeof(a[0])，すなわち int 型のサイズで割ればよい。まとめると，配列 a の要素数を取得するためには以下のように書けばよいことが分かる。

```
sizeof(a) / sizeof(a[0])
```

となると，配列の要素数を返す関数を次のように書けそうである。

```
int getIntArraySize(int *a)
{
      return sizeof(a) / sizeof(a[0]);
}
```

ところが，これはうまくいかない。前回のポインタの話を思い出そう。関数が引数として受け取ったのは，配列ではなくあくまでポインタである。よって，geIntArraySize は

受け取ったものが配列であることが分からない。よって，配列のサイズを関数で取得することができない。コンパイルエラーは起きないが，(int 型へのポインタのサイズ)/(int 型のサイズ)の値が返ってくる。int 型へのポインタのサイズは多くの場合 4 であるため，結局 1 が返ってくることになる。つまり要素数は返ってこない。
配列 a の要素数が取得できるのは，あくまで a のスコープ内でのみであることを覚えておこう。

3. 評価

評価とは，式の値を調べることである。式を適当な値に読み替えることである。
足し算の場合はその計算結果が評価値として返ってくるし，関係演算子を評価すると 0 または 1 が返ってくる。変数 a を評価すると，変数 a の値が返ってくる。評価はプログラムの実行時に行われる。
(例)

- 1 + 2 を評価すると 3 が得られる
- a = 5 のとき，a を評価すると 5 が得られる
- 2 <= 5 を評価すると 1 が得られる

第一回の変数の部分を引用する。

```
val1 = 10;
val2 = val1 + 35;
```

で,val1 に 10 を,val2 に(val1+35)を代入している.val2 の代入の右辺については,

```
val1 + 35 = 10 + 35 = 45
```

と計算されて,最終的には val2 には 45 が代入される.

ここでの処理を評価を用いて説明しよう。

```
val2 = val1 + 35;
```

という部分では，まず代入処理や足し算が行われる前に，val1 が評価される。val1 には 10 が入っているのだった。よって，val1 の評価値は 10 である。上の式は以下の式に読み替えられる。

```
val2 = 10 + 35;
```

次に，足し算の式が評価される。評価値は 10 と 35 を足した 45 である。よって，上の式は以下の値に読み替えられる。

```
val2 = 45;
```

そして，val2 には 45 が代入されることになる。

実は代入演算子も評価ができる。評価値は代入後の値である。
たとえば a = 5 を評価すると 5 が得られるから，以下のように printf を書けば 5 が出力されることになる。

```
printf("%d", a = 5);
```

代入の評価をうまく利用すると，次のように，一行で複数の変数に値を代入するような処理が書ける。

```
a = b = 2;
```

少し詳しく見てみよう。まず b＝2 で b に 2 が代入処理が行われるが，同時に評価値として 2 が返ってくる。よって，以下のように読み替えができる。

```
a = 2;
```

そして, a には 2 が代入されることになる。

4. 前置インクリメント/デクリメント

今まで i++, i—のように, ++や—を変数の後ろにくっつけたものを「インクリメント演算子」と呼んでいたが，これは厳密には「後置インクリメント演算子」と呼ばれる。
対して，++i, --i のように，++や—が変数の前にくっついたものを「前置インクリメント演算子」という。
++i と i++の何が違うのかというと，それは評価の値である。++i を評価すると, i に 1 加えた後の値が得られるが, i++を評価すると, i に 1 加える前の値が得られる。これはデクリメント演算子の場合も同様である。

5. 条件式の短絡評価

論理演算子は，ある条件を見たすと評価がスキップされることがある。これは短絡評価と言われる。論理演算子&&と||についてみていこう。
論理演算子&&は，式を左から評価する。もし左の式の値が 0 なら，右の値は評価されず，問答無用で全体の式は 0 になる。例えば，以下の式があったとする。

```
A == B && C > 2 && D <= B
```

もし A == B を評価して 0 が得られると，以下の状態になる。

```
0 && C > 2 && D <= B
```

&&演算子の性質により，右の条件 C>2 や D<=B が吟味されることはなく，全体の式として評価値 0 が得られる。
論理演算子||も同様に，式を左から評価する。もし左の式が 1 なら，右の値は評価されず，問答無用で全体の式は 1 になる。例えば，以下の式があったとする。

```
A == B && C > 2 && D <= B
```

もし A == B を評価して 1 が得られると，以下の状態になる。

```
1 || C > 2 || D <= B
```

||演算子の性質により，右の条件 C>2 や D<=B が吟味されることはなく，全体の式として評価値 1 が得られる。
短絡評価を意識しておくと，以下のようなプログラムが書ける。

```
/* n は配列の要素数 */
/* check は配列に対して何かの走査をする関数 */
if(i < n && check(i) == 1 ) ...
```

条件式の順番が重要である。もしも順番が逆だったら， i >= n のときも check(i)が呼ばれてしまうため，配列外参照の危険がある。i＜n という条件が満たされなければ，そもそも check(i)がみられることはない。

6. カンマ演算子

カンマ演算子を用いると複数の式を一つの文にまとめて書ける。以下の書式で書く。

式，式，式，…，式;

式は左から順番に処理される。

例を示そう。代入やインクリメント演算子を組み合わせた次のように書ける。

```
x = 1, y = a + 2;
l++, r--;
```

カンマ演算子は処理が簡潔に書けることがあるが，あまり多用しすぎるとソースコードが読みづらくなってしまうため，注意が必要である。

7. CSV ファイルでグラフ描画

問題を解く際，何かの数列の規則性を掴むために値を出力してみたりグラフにしてみたりすることは大切である。

そこで，printf で出力した値をリダイレクト機能を用いて CSV に出力し，Excel の散布図で描画する例を示す。

以下の数式を描画してみよう

$$y = \begin{cases} 50x\ (0 \leq x < 50) \\ -(x-50)^2 + 2500\ (50 \leq x < 100) \end{cases}$$

以下のプログラムを書く。

```
#include <stdio.h>

int main(void)
{
    int i;

    for(int i = 0; i < 100; i++) {
        if(i <= 50) printf("%d¥n", 50 * i);
        else printf("%d¥n", -(i - 50)*(i - 50) + 2500);
```

```
    }

    return 0;
}
```

これをコンパイルして実行すると，以下のように 100 個の値が改行区切りで出力される。

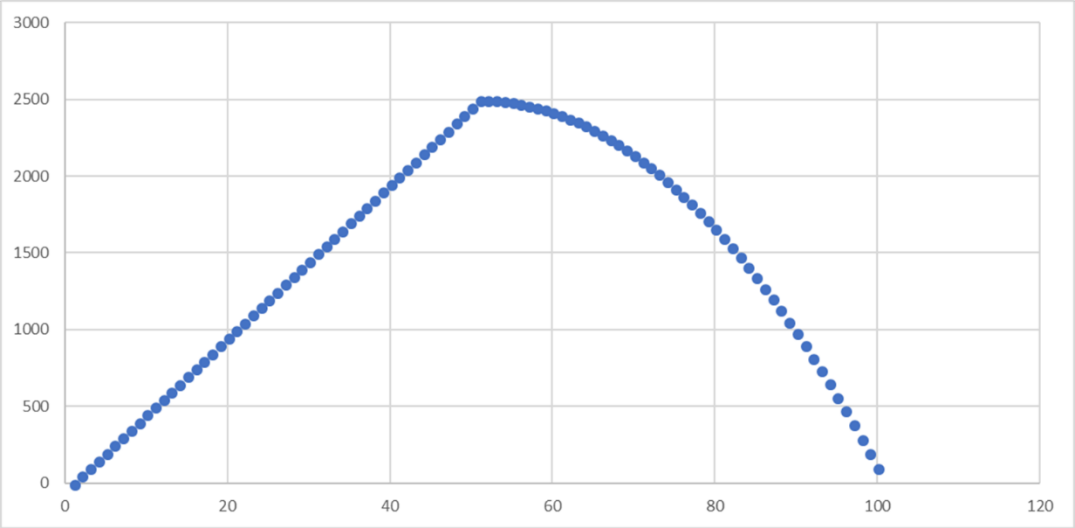
```
¥maximum>a.exe
0
50
100
150
200
250
300
350
400
450
500
550
600
650
700
750
800
850
…
```

この出力結果を，リダイレクトを用いて out.csv に出力する。

```
¥maximum>a.exe > out.csv

¥maximum>
```

ここでは,maximum フォルダ内に out.csv を作成した。out.csv を Excel で開くと，出力した値がセルに入っていることが分かる。これを用いて散布図を作成すると，次のようになる。



Excel の散布図の作り方やリダイレクトなどについて，かなり説明を省略している。分からない場合は各自調べるなり質問するなりしてほしい。

また，今回は点の x 軸の間隔を特に設定していないが，幅 1 として描画されている。幅をもう少し細かく設定したいなら，カンマ付きで出力すると良い。csv ファイルのフォーマット上，カンマを利用して列を区切ることができる。

例えば，$y=\sqrt{x}$ のグラフを 0.1 の幅で出力するには，次のように書ける。

```c
#include <stdio.h>
#include <math.h>

int main(void)
{
    int i;

    for(i = 0; i < 1000; i++) {
        printf("%f,%f¥n", i / 10.0, sqrt(i / 10.0));
    }

    return 0;
}
```

i の値は整数値なのだが，それを 10 で割ることで 0.1 刻みを実現している。ループカウンタを実数にして，

```c
double d;
for(d = 0; d < 100; d += 0.1) {
    ...
}
```

と書きたいところだが，実数の誤差の関係でやるべきではない。0.1 程度なら大丈夫だが、もっと刻みを小さくすると、正常に動作しなくなる。ループカウンタには整数値を使うべきである。

8. ストリーム

ストリームとは，入出力とプログラムをつなぐための経路である。ストリームという名前の通り，川の流れをイメージすると良い。ストリームが無いと，プログラムはどこから入力し，どこへ出力すればよいかわからない。ストリームを通じて，プログラムで指定した文字や値を出力に送る。ストリームを通じて，キーボードなどからの入力をプログラムに送る。それぞれ，出力ストリーム・入力ストリームと呼ばれている。

C 言語ではストリームがあらかじめ作られている。実は，printf は出力ストリームに文字列や数値を送る関数であり，scanf は入力ストリームから文字列や数値を受け取るための関数である。