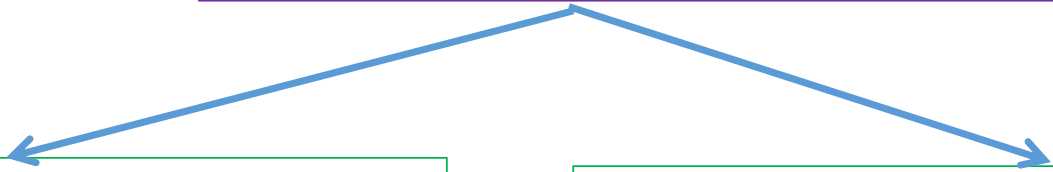
Cプログラミング演習資料

ファイルからの読み込み

ファイルから整数のデータを読み込む方法に二通り：比較しよう

共通部分
NUMの値は
1000などとする

```
int n,i,ar[NUM];
FILE *fp;
fp= fopen(FName,"r");
```



```
for(i=0; i<NUM; i++) {
  fscanf(fp,"%d",&ar[i]);
 }
```

ファイルの行数が既知（NUM行)の場合にしか使えない
少なかったり多かったりすると問題が起きる

```
n =0;
while (fscanf(fp,”%d”,&ar[n]) != EOF) {
        n++;
  }
```

ファイルの行数が未知でも大丈夫
読み込んだ個数がnの値となっている　∴お勧め！

ファイルのデータを使う場合のmain関数

ファイルからデータを読み込むのはmain関数で行い、平均や分散を求める関数を呼び出す、という形式にする。具体例は以下:

```
#define NUM 100
int main(void)
{
        int i=0,ar[NUM];
        FILE *fp = fopen("data.txt","r");
        while ( fscanf(fp,"%d",&ar[i]) != EOF ) {
                printf("%d ", ar[i]);
                i++;
        }
        fclose(fp);
        printf("¥n平均=%f, 分散=%f¥n",mean(ar,i),var(ar,i));
        return 0;
}
```

ちゃんと読み込めているかどうかのチェック
読み込めていたらコメントアウトする

平均を求める関数meanの呼び出し

分散を求める関数varの呼び出し

データを記憶した配列(ar)と要素数(n)を引数として、それぞれの関数を呼び出す

関数…

main関数の中で処理をするのをやめ、関数の形で書くことを議論してきた。その大きな理由は、関数の形にしておくと「再利用できる」ことにある。

次の例をみてみよう:

```
int main(void) {
    int ar[NUM1], br[NUM2], cr[NUM3];
            // 途中省略
    // arの和を求める
    sumA = 0;
    for (i=0; i < NUM1; i++) {
            sumA += ar[i];
    }
    // brの和を求める
    sumB = 0;
    for (i=0; i < NUM2; i++) {
            sumB += br[i];
    }
     // crの和を求める
    sumC = 0;
    for (i=0; i < NUM3; i++) {
            sumC += cr[i];
    }
    ...
}
```

同じことを何回も書かなければならない。
プログラムがわかりにくく、間違いを起こしやすい

関数…

ひとつにまとまっている

```
int sum(int x[ ], int n) { // xは配列、nは要素数
     // xの和を求める
    int i, ans= 0;
    for (i=0; i < n; i++) {
            ans += x[i];
    }
    return ans;
}

int main(void) {
    int ar[NUM1], br[NUM2], cr[NUM3], as, bs, cs;
    as = sum(ar, NUM1);
    bs = sum(br, NUM2);
    cs = sum(cr, NUM3);
            ...
}
```

関数sumを適切な引数で呼び出せば良い
main関数にかぎらず、いろいろなところで利用可能

```
int main(void) {
    int ar[NUM1], br[NUM2], cr[NUM3];
            // 途中省略
    // arの和を求める
    sumA = 0;
    for (i=0; i < NUM1; i++) {
            sumA += ar[i];
    }
    // brの和を求める
    sumB = 0;
    for (i=0; i < NUM2; i++) {
            sumB += br[i];
    }
     // crの和を求める
    sumC = 0;
    for (i=0; i < NUM3; i++) {
            sumC += cr[i];
    }
    ...
}
```

同じことを何回も書かなければならない。
プログラムがわかりにくく、間違いを起こしやすい

ファイルに書出す

```
#define NUM 100
int main(void)
{
        int i=0,ar[NUM];
        FILE *fp = fopen("data.txt","r");
        while ( fscanf(fp,"%d",&ar[i]) != EOF ) {
                printf("%d ", ar[i]);
                i++;
        }
        fclose(fp);
        printf("¥n平均=%f, 分散=%f¥n",mean(ar,i),var(ar,i));

       return 0;
}
```

```
fp = fopen(“result.txt”,”w”)
 // “result.txt”ファイルを書き出し用に開く
fprintf(fp, "¥n平均=%f, 分散=%f¥n",
        mean(ar,i),var(ar,i));
   // printf とよく似ている
fclose(fp);
```

注意点： 関数が配列を引数とするときにどう書くか？

更に進んで:ファイルから読み込んだ数を配列で記憶するのも「関数」にする

これが template0501.cでやっていたこと

```
#define NUM 100
int main(void)
{
        int i=0,ar[NUM];

        i=getData(ar, “data.txt”)

        printf("¥n平均=%f, 分散=%f¥n",mean(ar,i),var(ar,i));
        return 0;
}
```

```
int getData(int ar[], char fname[]) {
  FILE *fp;   // ファイルポインタ
  int n=0 ;   // 読み込んだ整数の個数
  fp = fopen(fname,”r”)
  // while文を用いてファイルから数を読み込む
  while (fscanf(fp, “%d”, &ar[n])!=EOF) {
     n++;
  }
  return n;// 読み込んだ整数の個数を返す
}
```

注意点： 関数が配列を引数とするときにどう書くか？

これまでみてきた処理のパタン

（1）数式の指定通りに計算

例： 階乗計算　　nを自然数として n! = 1×2×... × n

（2）配列を使うもの　⇒ 繰り返し

(a) 答えの変数の初期値 0

例：要素の和を求める

例： 分散　　要素と平均との差の自乗の和を求め、要素数で割る

(b) 答えの変数の初期値は、配列の先頭要素

例：最大／最小要素を求める

答えの候補と、それぞれの要素とを比較し、答えの候補の値を更新する

例：2番めに大きい・小さい要素

最大（最小）と2番目の2つを記憶する必要がある