プログラミングⅠ

Report#5

提出日：6 月 13 日（木）

所属：琉球大学工学部情報工学科

学籍番号：135713B

氏名：天願 寛之

目次

1. 次のプログラムは外部変数に定義され、初期化された 10 個の int 型データの平均を求め、各データと平均との差を表示するプログラムである。このプログラムを以下のような関数の翻訳単位にファイルを分け、同様な動作をするプログラムを作成せよ。(元となるプログラムは省略)

型	関数名	引数・パラメータ
float	get_ave	なし
void	print_data	配列の添字、平均値

型：機能	戻り値
float：平均値を求め表示	平均値
void：1 つのデータと平均値との差を求め表示	なし

1-a Makefile の作成

1-a-1.Makefile の中身

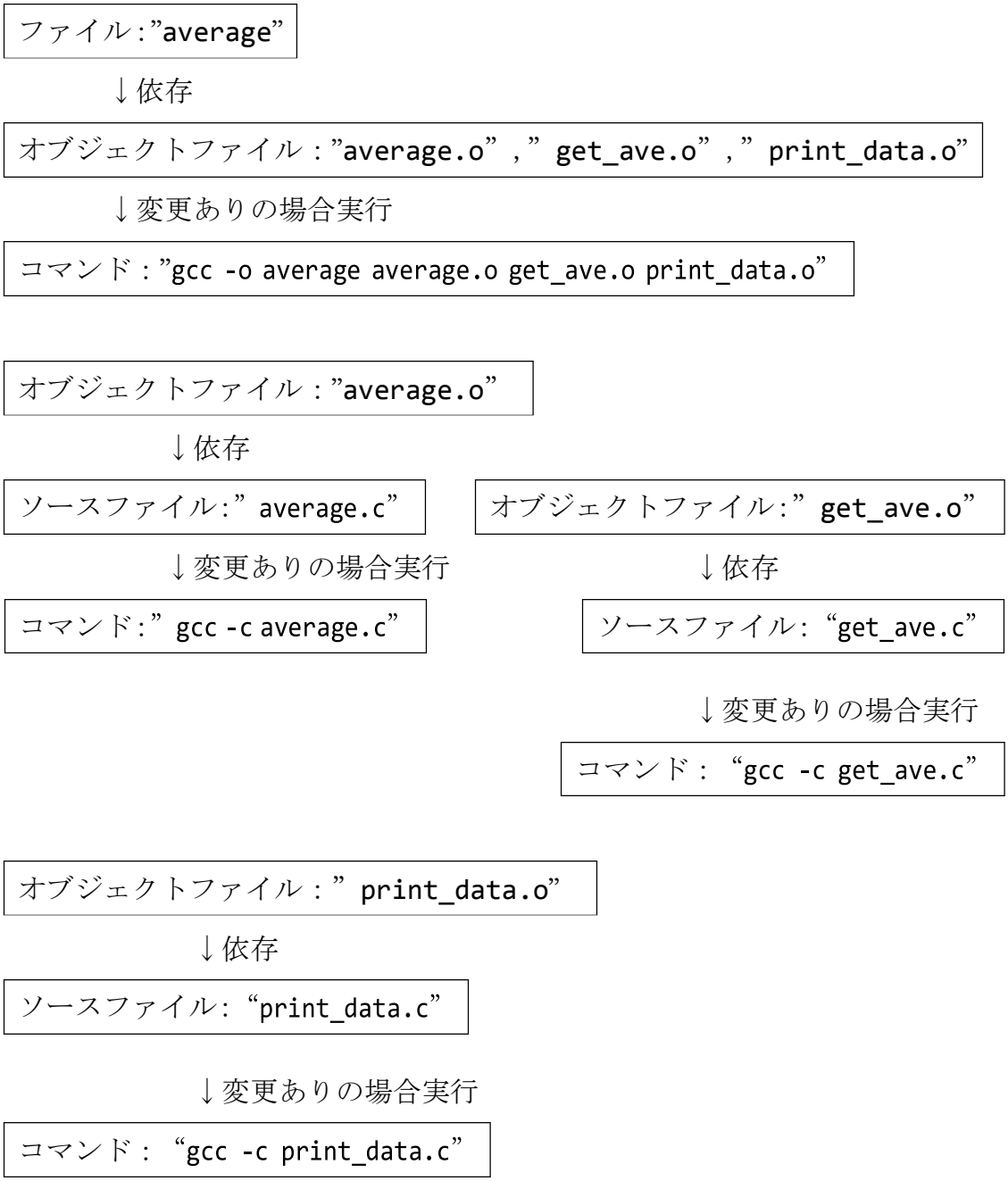
```
01 #
02 # average.c のコンパイル&実行のための makefile
03 #
04
05 average: average.o get_ave.o print_data.o
06         gcc -o average average.o get_ave.o print_data.o
07
08 average.o: average.c
09         gcc -c average.c
10
11 get_ave.o: get_ave.c
12         gcc -c get_ave.c
13
14 print_data.o: print_data.c
15         gcc -c print_data.c
```

1-a-2.解説

Makefile は複数のファイルの再コンパイルと再リンクに必要とするプログラムの依存関係を示すものである。この場合生成ファイル(05 行目)”average”はオブジェクトファイル(05 行目)”average.o”,” get_ave.o”,” print_data.o”に依存しており、依存されているファイルに変更があったとき、コマンド(06 行目)”gcc -o average average.o get_ave.o print_data.o”を実行する。以下のことは次のページに”mekefile の依存関係図”として載せておく。

このように mekefile を記述しておくことにより、make コマンドを実行する事で分割コンパイルを一括してコンパイルする事が出来る。

Makefile の依存関係図



1-a-3.考察

イ. makefile、make コマンドを用いることで多くのモジュールで構成された大きなプログラムを容易にコンパイルすることが出来る。

ロ. 複数人で部分的に分けてプログラムを作成するときに非常に役にたつ方法である。

1-b 関数の翻訳単位にファイルを分けたプログラムの作成

1-b-1.ソース 1.average.c

```
01 /*
02  program   : average.c
03  comments  : 結合(linkage)を用いて、平均・差を求める。
04 */
05
06 #include <stdio.h>
07
08 float get_ave(void); //関数 get_ave のプロトタイプ宣言
09 void print_data(int i,float ave); //関数 print_data のプロトタイプ宣言
10 int score[10]={3,5,8,9,10,6,7,9,8,3}; //int 型変数配列 score を定義
11
12 int main(){
13   int   i; //int 型変数 i の宣言
14   float ave; //float 型変数 ave の宣言
15   ave = get_ave(); //関数 get_ave の呼び出し、かつその戻り値を変数 ave に代入
16   print_data(i,ave); //変数 i,ave を引数として、関数 print_data の呼び出し
17
18   return(0);
19 }
```

1-b-2.ソース 2.get_ave.c

```
01 //関数 get_ave の定義
02
03 float get_ave(void){
04   int i; //int 型変数 i を宣言
05   extern int score[10]; //外部変数の呼び出し
06   float ave = 0.0; //float 型変数 ave を宣言し、0.0 を初期値とする
07   float total = 0.0; //float 型変数 total を宣言し、0.0 を初期値とする
08
09   for(i=0; i<10; i++) //for 文でループ
10     total = total + (float)score[i]; //score に格納された値を一時的に float 型にし、全て
11 の値を加算
12   ave = total / 10.0; //加算された値の平均値を変数 ave に代入
13   printf("ave = %4.1f¥n",ave); //平均値を出力
14
15   return(ave); //戻り値 ave を返す
16 }
```

1-b-3.ソース 3.print_data.c

```
01 //関数 print_data の定義
02
03 void print_data(int i, float ave){ //main 関数の int 型変数 i と float 型変数 ave を引数と
04 する
05   extern int score[10]; //外部変数の呼び出し
06   float dif; // float 型変数 dif を宣言
07
08   for(i=0; i<10; i++){ //for 文でループ
09     dif = score[i] - ave; //score に格納された値と平均値との差を変数 dif に代入
10     printf ("score[%02d]=%2d  Difference from average = %4.1f¥n",i,score[i],dif);
11   //配列の番地、配列の値、変数 dif を出力
12       }
13 }
```

※出力結果は元となったプログラムと同じなので省略

1-b-4.解説

基本的な流れと役割はコメントを参照

イ. 外部変数とは複数のソースファイルをリンクして1つのプログラムを作る際、複数のソースファイル内で共通に使う変数である。つまり呼び出すことにより複数のソースファイルで役割を引き継ぐことが出来る。また外部変数を扱うときには記憶域クラス指定子 exterm を用いる。

ロ. 関数 get_ave で返した値”ave”(平均値)を関数 main で宣言した変数 ave に代入し、この代入された”ave”と float 型変数 i を引数として関数 print_data で用いている。

ハ. ”%f”は浮動小数点数で出力するときに用い、”%4.1f”とは小数点第 1 位から小数点を含んだ 4 桁で出力する。

1-b-5.考察

イ. 最初は関数 get_ave で返した値”ave”(平均値)を関数 print_data で用いるにはどうしたら良いか分からなかった。

ロ. プロトタイプ宣言された関数内の int 型変数配列 score の外部変数 extern を除いて出力するとどうなるか試してみる。

1-b-5-1.出力結果

```
ave = 321293568.0
score[00]=1646208424  Difference from average = 1324914816.0
score[01]=32767  Difference from average = -321260800.0
score[02]=16  Difference from average = -321293568.0
score[03]=32767  Difference from average = -321260800.0
score[04]=1566628968  Difference from average = 1245335424.0
score[05]=32767  Difference from average = -321260800.0
score[06]= 0  Difference from average = -321293568.0
score[07]= 0  Difference from average = -321293568.0
score[08]= 0  Difference from average = -321293568.0
score[09]=10  Difference from average = -321293568.0

[プロセスが完了しました]
```

1-b-5-2.考察

イ. 出力結果は理解出来ないが、プロトタイプ宣言された関数内の変数 score は外部変数とは異なる変数であり、引き継いでいないことは明白である。

ロ. 関数外で定義された変数よりもファイル内で宣言された同名の変数が優先されているということも分かる。

1-c. オリジナルプログラムの作成

1-c-1. オリジナルソース 1.average2.c

```
#include <stdio.h>

float get_ave(void); //関数 get_ave のプロトタイプ宣言
void print_data(int i,float ave); //関数 print_data のプロトタイプ宣言
int score[7]; //int 型変数配列 score を定義

int main(){
  int   i; //int 型変数 i の宣言
  int count; //int 型変数 count の宣言
  float ave; //float 型変数 ave の宣言

  for(count=0; count<7; count++){
    printf("%d 科目目のテストの点数 ＝ ",count+1);
    scanf("%03d",&score[count]);
 }
  ave = get_ave(); //関数 get_ave の呼び出し、かつ、その戻り値を変数 ave に代入
  print_data(i,ave); //変数 i,ave を引数として、関数 print_data の呼び出し
  return(0);
}
```

1-c-2. オリジナルソース 2.get_ave2.c

```
//関数 get_ave の定義

float get_ave(void){
  int i; //int 型変数 i を宣言
  extern int score[7]; //外部変数の呼び出し
  float ave = 0.0; //float 型変数 ave を宣言し、0.0 を初期値とする
  float total = 0.0; //float 型変数 total を宣言し、0.0 を初期値とする

  for(i=0; i<7; i++) //for 文でループ
    total = total + (float)score[i]; //score に格納された値を一時的に float 型にし,全ての値
を加算
  ave = total / 7.0; //加算された値の平均値を変数 ave に代入
  printf("ave = %4.1f¥n",ave); //平均値を出力
  return(ave); //変数 ave を返す
}
```

1-c-3. オリジナルソース 3.print_data2.c

```
//関数 print_data の定義

void print_data(int i, float ave){ //main 関数の int 型変数 i と float 型変数 ave を引数¥
とする
extern int score[7]; //外部変数の呼び出し

 float dif; // float 型変数 dif を宣言
 for(i=0; i<7; i++){ //for 文でループ
   dif = score[i] - ave; //score に格納された値と平均値との差を変数 dif に代入
   printf("score[%02d]=%2d  Difference from average = %4.1f¥n",i,score[i],dif)
; //配列の番地、配列の値、変数 dif を出力
   if(dif <= -10){
     printf("苦手を無くそう¥n");
   }else if(-10 <= dif && dif <= 0){
     printf("もう少し頑張ろう¥n");
   }else if(0 <= dif && dif <= 10){
     printf("そのまま頑張ろう¥n");
   }else printf("得点源として活用しよう¥n");
 }
 }
```

1-c-4. 出力結果

```
1科目目のテストの点数 ＝ 90
2科目目のテストの点数 ＝ 80
3科目目のテストの点数 ＝ 70
4科目目のテストの点数 ＝ 60
5科目目のテストの点数 ＝ 50
6科目目のテストの点数 ＝ 46
7科目目のテストの点数 ＝ 30
ave = 60.9
score[00]=90  Difference from average = 29.1
得点源として活用しよう
score[01]=80  Difference from average = 19.1
得点源として活用しよう
score[02]=70  Difference from average =  9.1
そのまま頑張ろう
score[03]=60  Difference from average = -0.9
もう少し頑張ろう
score[04]=50  Difference from average = -10.9
苦手を無くそう
score[05]=46  Difference from average = -14.9
苦手を無くそう
score[06]=30  Difference from average = -30.9
苦手を無くそう

[プロセスが完了しました]
```

1-c-5. 解析

イ. テストの点数を入力したらそれぞれの科目の点数に対する評価を出すプログラムを作成した。

ロ. main 関数内に for 文の scanf 関数で 1 人 1 人の得点を入力している。

ハ. print_data 関数で入力した点数と平均との差をだし、if-else 文を用いて、それぞれに評価を出すようにした。

1-c-6. 考察

関数ごとにファイルを分けることで、このような改良の際に変更が容易に行えることが分かった。これが make コマンドを使う利点と言える。

2. 変数のスコープと記憶域クラスについて考察せよ。
2-a. 変数のスコープとは変数が有効であるプログラムの範囲のことである。

2-a-1. 考察
変数の関数原型スコープは関数宣言の内部でのみ有効。例えば”1-b”の”1 ソース average.c”でプロトタイプ宣言した関数 print_data の仮引数としていれた int 型変数 i、float 型変数 ave(9 行目)の有効範囲はプロトタイプ宣言内だけということである。ここで宣言しても変数 i,ave は使えず、使う為には再度宣言しなければならない。

以下 2-a-2-1 のプログラムを参考にして考える。
2-a-2-1. プログラム I (関数スコープ、関数有効範囲)

```
#include <stdio.h>

#define FALSE 0
#define TRUE  !FALSE

int main(){
  int  count; //関数スコープ

  count = 0;
  while(TRUE){
    count++;
    if(count > 5) break;
    printf("While-Count=%2d¥n",count);
  }
  return(0);
}
```

2-a-2-2. プログラム I 出力結果

```
While-Count= 1
While-Count= 2
While-Count= 3
While-Count= 4
While-Count= 5
```

2-a-2-3. 考察
イ.変数が宣言された関数内全域の有効な範囲が関数スコープである。
ロ.網掛けの部分の範囲が関数スコープである。

2-a-3-1. プログラムⅡ(ファイル・スコープ、ファイル有効範囲)※出力結果省略

```
#include <stdio.h>

#define FALSE 0
#define TRUE  !FALSE
int count; //グローバル変数・ファイルスコープ

int main(){

  count = 0;
  while(TRUE){
    count++;
    if(count > 5) break;
    printf("While-Count=%2d¥n",count);
  }
  return(0);
}
```

2-a-3-2 考察

イ. このソースファイル内の全てのブロック文の外側で宣言された変数の有効範囲がファイル・スコープである。

ロ. プログラムⅠと同じ出力結果になったことから有効範囲は網掛けの部分である。

ハ.このようにプログラムのあらゆる箇所で有効な変数をグローバル変数という。

2-a-4-1. プログラムⅢ(ブロック・スコープ、ブロック有効範囲)

```
#include <stdio.h>

#define FALSE 0
#define TRUE  !FALSE
int count; //グローバル変数・ファイルスコープ

int main(){
  count = 0;
  {
    int count; //ローカル変数・ブロックスコープ
    count = 1;
    while(TRUE){
    count++;
    if(count > 6) break;
    printf("While-Count=%2d¥n",count);
  }
  }
  count++;
    printf("Count=%2d¥n",count);
  return(0);
}
```

2-a-4-2 出力結果

```
While-Count= 2
While-Count= 3
While-Count= 4
While-Count= 5
While-Count= 6
Count= 1
```

2-a-4-3. 考察

イ. ブロック文の中で宣言された変数の有効範囲がブロック・スコープのことである。

ロ. 出力結果から分かるようにグローバル変数以外の同名の変数がブロック文にあるとブロック文での変数が優先されるが、それ以外には影響を及ぼさない。つまりブロック文の変数はブロック文の中だけで役割を果たし、外部からはアクセス出来ないということだ。

ハ. このようにブロック文の中だけの変数をローカル変数という。

ニ. ローカル変数は網掛けの部分が有効範囲である。

ホ. 外部側で宣言された変数は内部側でも有効だが，内部側にも同一名で変数宣言がある場合には，内部側では内部で宣言した方が有効となり，外部側で宣言した方は内部側では隠れた状態になる。

2-b. 記憶域クラスについて

記憶域クラスとは、どのようにメモリに記憶され、どのように開放されるかを示しているものである。

※ extern(外部変数)については 1-b-4~5. 解説・考察、1-b-5-1~2. 考察で終えているので省略。

2-b-1. 静的記憶と動的記憶のプログラム

```
01 #include <stdio.h>
02
03 void  inc_n(void); //関数 inc_n のプロトタイプ宣言
04
05 int main(){
06   inc_n(); //関数 inc_n の呼び出し 1 回目
07   inc_n(); //関数 inc.n の呼び出し 2 回目
08   inc_n(); //関数 inc.n の呼び出し 3 回目
09 }
10 //関数 inc_n の定義
11 void inc_n(void){
12   static  int sn = 0; //静的変数 sn を宣言、中身は固定的に記憶され続ける
13   auto    int an = 0; //動的変数 an を宣言、関数を呼び出すたびに初期化される
14
15   printf("befor n++  sn=%2d  an=%2d¥n",sn,an);
16   sn++;
17   an++;
18   printf("after n++  sn=%2d  an=%2d¥n¥n",sn,an);
19 }
```

2-b-2. 出力結果

```
befor n++  sn= 0  an= 0
after n++  sn= 1  an= 1

befor n++  sn= 1  an= 0
after n++  sn= 2  an= 1

befor n++  sn= 2  an= 0
after n++  sn= 3  an= 1
```

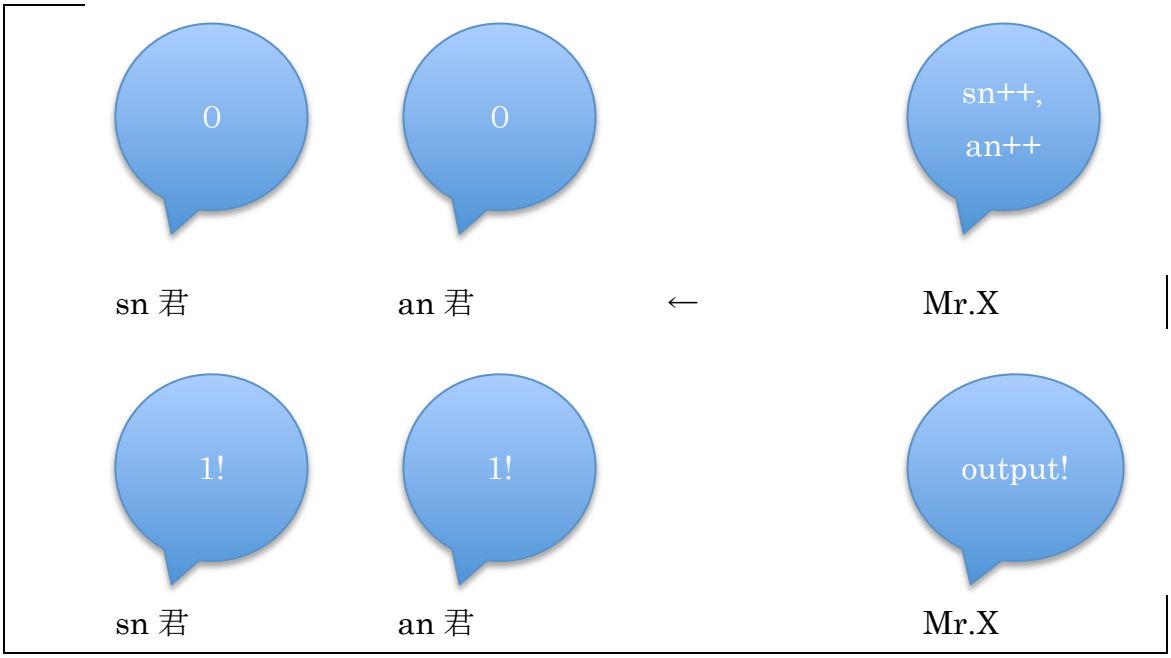
2-b-3. 考察

イ. 12 行目の記憶域クラス指定子”static(静的変数)”はコンパイル時にその変数値記憶用のメモリ領域を、固定的に確保してしまう。つまり出力結果に見えるように 1 を加算した後、再び関数 inc_n を呼び出すときには変数 sn に値 1 が記憶されており、これに 1 を加算する流れになる。

ロ. 13 行目の記憶域クラス指定子”auto(動的変数)”は変数を必要とするときだけメモリ上にある番地(スタック領域という)に変数値記憶用の領域を確保し、不要になったらそのメモリ領域を開放する。つまり出力結果に見えるように 1 を加算して、関数 inc_n が値を返した直後、メモリを空にしている。よって再び関数 inc_n が呼び出されたときには値が初期化している。

ハ. 以下、イメージ図

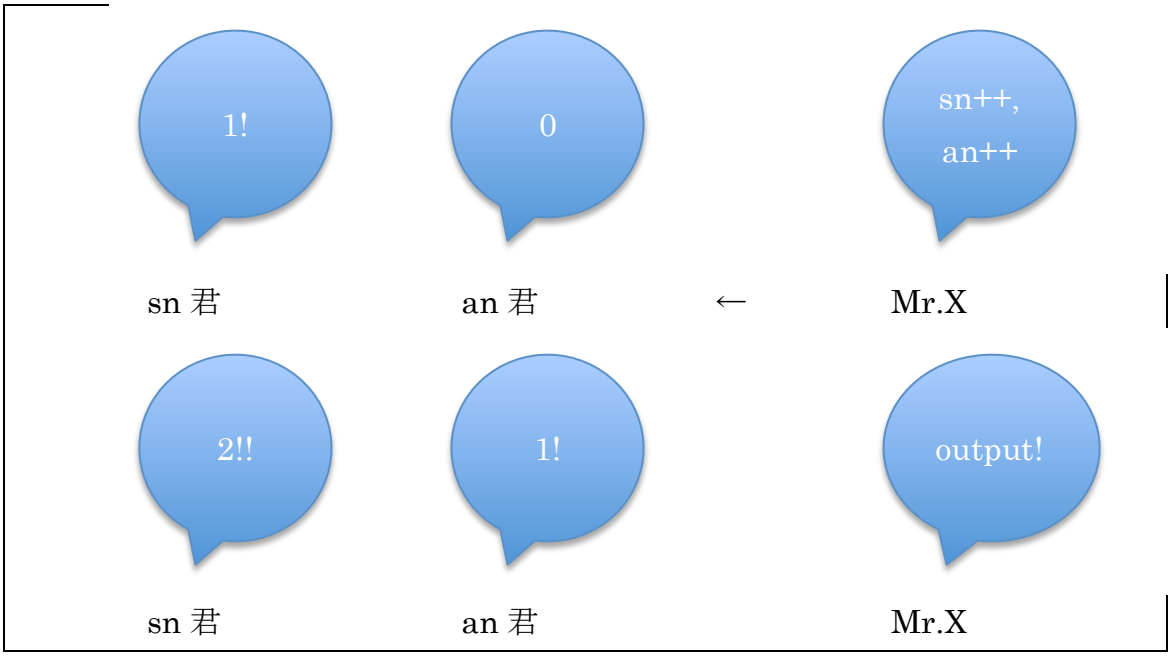
1 回目の呼び出し



an 君「開放ぉぉぉぉお!!!」----------

an 君「ハッ!?私はいったい今まで何を…?」 <10>

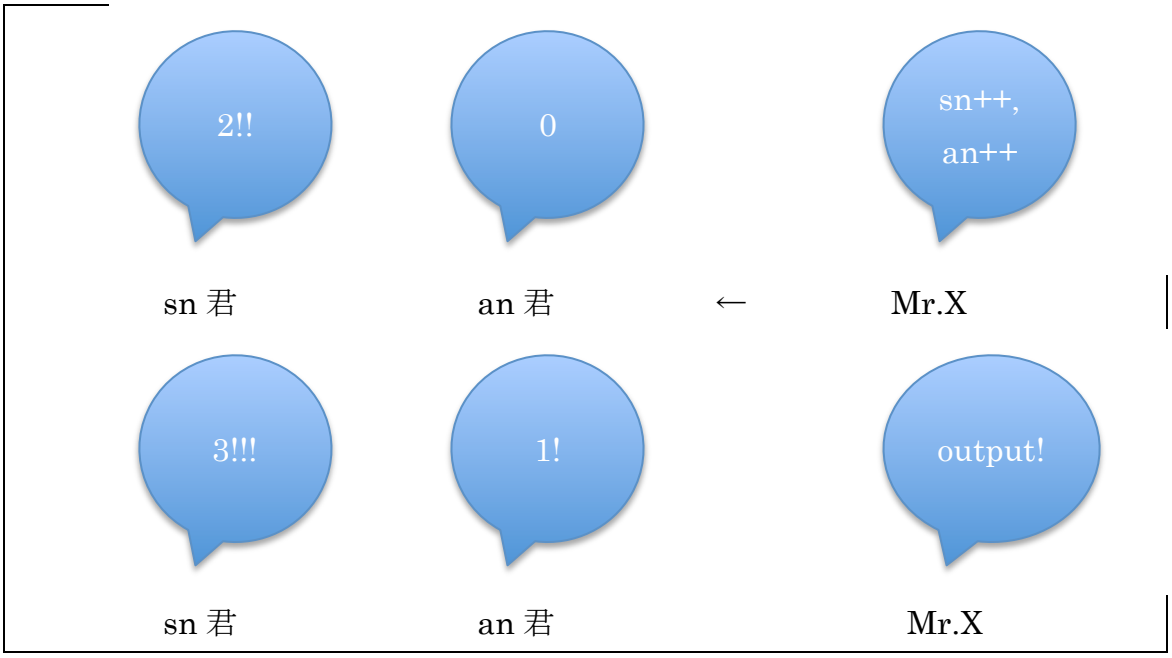
2 回目の呼び出し



an 君「開放ぉぉぉぉお!!!」----------

an 君「ハッ!?私はいったい今まで何を…?」

3 回目の呼び出し



3. 反省・感想

いろいろと図を用いて説明しようと頑張ったのだけれども、うまくいきませんでした。最後の図での説明などはむしゃくしゃしてやった気がします。プログラミングの前に word の使い方をマスターすべきかもしれません。しかし、今回の複数のプログラムファイルを一括してコンパイルする作業は楽しかったです。記憶域や変数スコープなどもしっかりと理解し生かせるようにしたいです。