ソース作成はMicrosoft Visual C++ 2008 Express Editionを使用し、実行はVisual Studio 2008 コマンド プロンプトをしようしているため、コンパイルは「cc」でなく「cl」で、
Makeはnmakeとなっている。

1. 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。

ソース
//ave1.c

```
1.  #include <stdio.h>
2.
3.  int score[10]={3,5,8,9,10,6,7,9,8,3};
4.
5.  float ave2();
6.  void ave3(int i, float ave);
7.
8.  int main(){
9.      int i;
10.     float ave;
11.     ave = ave2();
12.     ave3(i, ave);
13.     return(0);
14. }
```

ave1の考察
3行目:変数scoreの宣言。Scoreに10個の領域をとりscore[00]には3を、score[01]には、5をというように初期値を定める。なお初期値は定数でなくてはならない。
5行目:float型の関数ave2の宣言。
6行目:void型の関数ave3の宣言。()の中は引数。
9行目:変数iの宣言。
10行目:float(実数)型の変数ave の宣言。
11行目:aveの値は関数ave2に従って決まる。
12行目:関数ave3に11行目で決まったaveが入りave3が実行される。

//ave2.c

```
1.  #include <stdio.h>
2.
3.  extern int score[10];
4.  float ave2(){
5.      int i;
6.      float ave;
7.      for(i = 0; i < 10; i++)
8.          ave = ave + (float)score[i];
9.      ave = ave / 10.0;
10.     printf("ave = %3.1¥n",ave);
11.     return(ave);
12. }
```

ave2 の考察

3 行目：extern 複数ソースコードをまたいで変数やメソッドを使用するときに時に使う。
全ファイル中のどれかに定義されている宣言だけを行い定義は行わない宣言方法。
今回は ave1 で int score[10] 宣言している。
4 行目～
float 型の関数 ave2 の内容
7 行目: for 文によるループ。条件から 10 回目に 9 行目へ。
内容は score[00]～score[09]までを足すものである。
具体的な動作は、まず 1 回目に ave(初期値 0)と score[00](3)を足したものが、
ave の値となりループ。先ほどの ave の値(3)と score[01]を足したものが、ave となりループ。
これが score[09]まで続く。
9 行目:8 行目で求めた ave の値(68)を 10 で割り、平均を出す。
10 行目 ave の値の表示。%d でなく%f なのは、float 型であるため。
11 行目:ave の値を返す。

//ave3.c

```
1.  #include <stdio.h>
2.
3.  extern score[10];
4.
```

```
5.  void ave3(int i, float ave){
6.      float dif;
7.      for(i = 0; i < 10; i++){
8.          dif = score[i] - ave;
9.          printf("score[%02d]=%2d Difference from average = %4.1f¥n",i ,score[i], dif);
10.         }
11. }
```

ave3の考察

3行目:extern指定子の使用。

5行目～

void型の関数ave3の内容

6行目:float型の変数difの宣言

7～10行目:for文によるループ。

内容は、scoreからave2で求めたaveの値を引いたものの値がdifの値になる。

それを9行目で画面に出力する。

//makefile

```
1.  OBJECTS = ave1.obj ave2.obj ave3.obj
2.  average1.exe: $(OBJECTS)
3.  ＊  cl -o average1.exe $(OBJECTS)
4.  ave1.obj: ave1.c
5.  ＊  cl -c ave1.c
6.  ave2.obj: ave2.c
7.  ＊  cl -c ave2.c
8.  ave3.obj: ave3.c
9.  ＊  cl -c ave3.c
```

＊はTABキーの意

makefileの考察

1行目:OBJECTにマクロ定義をすることにより、以降の文章で楽ができる。

2行目: コロンの左側をターゲットと呼びnmakeで作成するもの(average1.exe)を指定する。

3行目average1.exeを作成するためのコマンドでTABを入れてから記述する。

OBJECTS(ave1.obj ave2.obj ave3.obj)とaverage1.exeの更新時刻を比べOBJECTSのほうが更新時刻が新しければコンパイルを実行するという意味。

4～5行目:ave1.objはave1.cに依存するという意味。またave1.objとave1,cとの更新時刻を比べave1.cのほうが新しければコンパイルを実行するという意味。
6～7、8～9行目:は4～5行目と同意。

結果

```
ave = 6.8
score[00]= 3 Difference from average = -3.8
score[01]= 5 Difference from average = -1.8
score[02]= 8 Difference from average =  1.2
score[03]= 9 Difference from average =  2.2
score[04]=10 Difference from average =  3.2
score[05]= 6 Difference from average = -0.8
score[06]= 7 Difference from average =  0.2
score[07]= 9 Difference from average =  2.2
score[08]= 8 Difference from average =  1.2
score[09]= 3 Difference from average = -3.8
```

ソース分割の考察

//Test.c

```
1.  #include <stdio.h>
2.
3.  int score[10]={3,5,8,9,10,6,7,9,8,3};
4.
5.  float ave2();
6.  void ave3(int i, float ave);
7.
8.  int main(){
9.      int i;
10.     float ave;
11.     ave = ave2();
12.     ave3(i, ave);
13.     return(0);
14. }
15. float ave2(){
16.     int i;
```

```
17.         float ave;
18.         for(i = 0; i < 10; i++)
19.                     ave = ave + (float)score[i];
20.         ave = ave / 10.0;
21.         printf("ave = %3.1f¥n",ave);
22.         return(ave);
23.   }
24.   void ave3(int i, float ave){
25.         float dif;
26.         for(i = 0; i < 10; i++){
27.                     dif = score[i] - ave;
28.                     printf("score[%02d]=%2d Difference from average = %4.1f¥n",i ,score[i], dif);
29.                     }
30.
31.   }
```

上記のソースはave1.c、ave2.c、ave3.cをまとめて少し編集したものである。
上記のソースでも分割したものと同じ結果が出る。
＊ただし、コンパイルするときに、
test.c(13)：warning C4700: 初期化されていないローカル変数 'i' が使用されます
test.c(20)：warning C4700: 初期化されていないローカル変数 'ave' が使用されます
とでた。
よってソースを分割する手順として、サブルーチンを使い分割したいソースを役割ごとにまとめ、
Main関数部分、サブルーチン部分を軸に分割しヘッダーなどをつけextern指定子を使い、
手直しして分割すればよいと思われる。

2．同様な動作をするオリジナルのプログラムを作成

//Random_ave.c

```
1.    #include <stdio.h>
2.    #include <stdlib.h>
3.    #include <time.h>
4.
5.    int score[10];
```

```
6.
7.    float get_ave();
8.    void print_ave(int i, float ave);
9.
10.   int main(){
11.         int i;
12.         float ave;
13.         srand((unsigned int)time(NULL));
14.         for(i=0; i<10; i++){
15.                   score[i]=rand()%101;
16.         }
17.
18.         ave = get_ave();
19.         print_ave(i, ave);
20.         return(0);
21.   }
```

Random_ave.c の考察

//ave1.c を変更、追加しているので、変更、追加したところのみ考察をする。

2 行目::乱数を発生させる rand()と srand を使うために必要なヘッダー

3 行目:time を使うためになヘッダー

13 行目:実行時の時間を元にして、発生させる乱数の元を決める

いつもこう書くので、とりあえずこの形で覚えて使えばよい。

14、15 行目:score[00]～score[09]の値をループさせることにより、決める。

//Get_ave.c

```
1.    #include <stdio.h>
2.
3.    extern int score[10];
4.
5.    float get_ave(){
6.          int i;
7.          float ave;
8.          for(i = 0; i < 10; i++)
9.                    ave = ave + (float)score[i];
```

```
10.         ave = ave / 10.0;
11.         printf("平均点%3.1f点¥n",ave);
12.         return(ave);
13.   }
```

Get_ave.c の考察

1 の ave2.c を参照。

//print_ave.c

```
1.    #include <stdio.h>
2.
3.    extern score[10];
4.
5.    void print_ave(int i, float ave){
6.          int name;
7.          float dif;
8.          name='A';
9.          for(i = 0; i < 10; i++){
10.                 dif = score[i] - ave;
11.                 printf("score[%c君] = %2d点平均点との差= %4.1f　評価…",name ,score[i], dif);
12.                 name++;
13.                 if(score[i]==100){
14.                       printf("Perfect.You are GOD.¥n");
15.                       }
16.                 else if(score[i]==0){
17.                       printf("You must die….¥n");
18.                       }
19.                 else if(dif<-25){
20.                       printf("F…不可ですね。¥n");
21.                       }
22.                 else if(-25.0 <=dif && dif <=-10.0){
23.                       printf("E¥n");
24.                       }
25.                 else if(-10 <dif && dif <= 0.0){
26.                       printf("Ｄ（Dope）¥n");
```

```
27.                                 }
28.                         else if(0 < dif && dif <= 10){
29.                                 printf("Ｃ（Cool）¥n");
30.                                 }
31.                         else if(10 < dif && dif <= 20){
32.                                 printf("Ｂ（Blast）¥n");
33.                                 }
34.                         else if(20 < dif && dif <= 30){
35.                                 printf("Ａ（Alright）¥n");
36.                                 }
37.                         else if(30 < dif && dif <= 40){
38.                                 printf("Ｓ（Sweet）¥n");
39.                                 }
40.                         else if(40 < dif && dif <= 50.0){
41.                                 printf("ＳＳ（Showtime）¥n");
42.                                 }
43.                         else
44.                                 printf("ＳＳＳ（Stylish）¥n");
45.
46.         }
47. }
```

Print_ave.c の考察

9～37 行目:score[i]の値、または dif の値により表示させる文字を変えている

Makefile

```
1.  OBJECTS = random_ave.obj get_ave.obj print_ave1.obj
2.  prog.exe: $(OBJECTS)
3.          cl -o prog.exe $(OBJECTS)
4.  random.obj: random.c
5.          cl -c random.c
6.  get_ave.obj: get_ave.c
7.          cl -c get_ave.c
8.  print_ave1.obj: print_ave1.c
9.          cl -c print_ave1.c
```

結果

Rand()を使ったから 2 つ載せます。

1 回目

平均点 73.3 点

score[A 君] = 80 点 平均点との差 = 6.7 評価… C（Cool）

score[B 君] = 54 点 平均点との差 = -19.3 評価… E

score[C 君] = 60 点 平均点との差 = -13.3 評価… E

score[D 君] = 99 点 平均点との差 = 25.7 評価… A（Alright）

score[E 君] = 99 点 平均点との差 = 25.7 評価… A（Alright）

score[F 君] = 100 点 平均点との差 = 26.7 評価… Perfect.You are GOD.

score[G 君] = 59 点 平均点との差 = -14.3 評価… E

score[H 君] = 61 点 平均点との差 = -12.3 評価… E

score[I 君] = 43 点 平均点との差 = -30.3 評価… F…不可ですね。

score[J 君] = 78 点 平均点との差 = 4.7 評価… C（Cool）

2 回目

平均点 36.3 点

score[A 君] = 15 点 平均点との差 = -21.3 評価… E

score[B 君] = 7 点 平均点との差 = -29.3 評価… F…不可ですね。

score[C 君] = 24 点 平均点との差 = -12.3 評価… E

score[D 君] = 0 点 平均点との差 = -36.3 評価… You must die….

score[E 君] = 55 点 平均点との差 = 18.7 評価… B（Blast）

score[F 君] = 72 点 平均点との差 = 35.7 評価… S（Sweet）

score[G 君] = 13 点 平均点との差 = -23.3 評価… E

score[H 君] = 64 点 平均点との差 = 27.7 評価… A（Alright）

score[I 君] = 49 点 平均点との差 = 12.7 評価… B（Blast）

score[J 君] = 64 点 平均点との差 = 27.7 評価… A（Alright）

3. 変数のスコープと記憶域クラスについて考察せよ。

【記憶域クラス指定子】

Auto（自動変数）：省略可

関数内部で宣言され、宣言された関数の中でのみ使用可能。（= ローカル変数）

関数実行中のみメモリ上のスタックに確保され、関数の実行が終了すると、メモリ上から削除さ

れる。（⇒ メモリの有効利用）
関数が呼ばれるたびに初期化を行う。
auto int a; のように宣言する。
ただし、「auto」は通常省略され、単に「int a;」のように宣言する。
明示的に初期化を行わないと、初期値は不定値になってしまう。
いつも変数を宣言しているのはこの形である。

Extern (外部変数)
関数外で定義され、定義以降のどの関数からでも使用可能。(= グローバル変数)
プログラム開始処理の前に一度だけ初期化を行う。
プログラム実行中に常に同じ場所に配置され値を保持。
明示的に初期化を行わない場合は、初期値は 0 になる。
今回、ソースの分割をする際に使った。

static（静的変数）
プログラム実行中に常に同じ場所に配置され値を保持 → 値を保持したいときに使用。(外部変数の機能)
関数内部で宣言され、宣言された関数の中でのみ使用可能。（自動変数の機能）
プログラム開始処理の前に一度だけ初期化を行う。
static int a; のように宣言する。
明示的に初期化を行わない場合は、初期値は 0 になる。

register（レジスター変数）
レジスター変数は文字通り CPU 内部に用意されたレジスターと呼ばれる記憶場所を直接利用するために用意されたもので、もしレジスターが利用できる場合にはレジスターに直接割り当てられ、メモリは利用されない(レジスターが使えない場合には次の自動変数と同じになる)。その用途は CPU の内部にあることによる高速性である。つまり、プログラマーが明示的にレジスターを利用する事で、より最適なプログラムが書けるようにしているものである。但し、現実には、レジスター変数は使う必要はない。その理由は現在のコンパイラーは最適化が進んでいるので、自動的にレジスターへの割り当てを行うからである。

typedef（型定義）
typedef（タイプデフ）は、プログラミング言語の C と C++の予約語である。これはデータ型に新しい名前をつけるために使用される。プログラマが容易にソースコードを理解できるようにすることが目的である。

例

Typedef　int　Windows

上記のように記述することで windows は int と同じように変数名を宣言できる。

ERROR 報告

乱数をとる際に、最初

int score[10]={a,b,c,d,e,f.…と変数名を当てたところ「error C2099: 初期化子が定数ではありません。」と怒られました。

```
int main(){
  int i;
  i=0;
  float ave = 0.0, dif;
```

上記のようにすると、下記のように言われる。

error C2065: 'ave'： 定義されていない識別子です。

error C2065: 'dif'： 定義されていない識別子です。

これから変数は先に宣言しておいてからのほうがよいとよそうされる。

感想

今回、初めてパソコンに殺意がわきました。エラーかどうかわからないが、

ソースをコンパイルし実行したところおもった通りにならず変な値が出ました。

コンパイルし直さずもう一回実行すると値が変わりました。また実行し続けると思った通りに結果を返してきました。それに気づかないので何回もいろいろ変えたりしたのに一向に変わらないのでものすごく頭にきました。具体的な場所は ave の値を出すところなのですが、値が 10 桁どころではありませんでした。最終的には思い通りのオリジナル？プログラムができてよかったです。

参考資料

「c 実践プログラミング」

「新 The UNIX Super Text(上)」

「初心者のためのポイント学習Ｃ言語」http://www9.plala.or.jp/sgwr-t/index.html

「Makefile の書き方」

http://www.c.csce.kyushu-u.ac.jp/~seiichirou/wiki/index.php?Makefile%A4%CE%BD%F1%A4%AD%CA%FD