＊ サンプルプログラム ： average.c

```
 1  /*
 2    program   : average.c
 3    comments  : 結合(linkage)を用いて、平均・差を求める。
 4  */
 5
 6  #include <stdio.h>
 7
 8  int score[10]={3,5,8,9,10,6,7,9,8,3};
 9
10  int main(){
11    int   i;
12    float ave = 0.0, dif;
13
14    for(i=0; i<10; i++) ave = ave + (float)score[i];
15    ave = ave / 10.0;
16    printf("ave = %3.1f\n",ave);
17
18    for(i=0; i<10; i++){
19      dif = score[i] - ave;
20      printf("score[%02d]=%2d  Difference from average = %4.1f\n",i,score[i],dif);
21    }
22
23    return(0);
24  }
```

＊ 実行結果 ： average.c

```
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
```

＊ 解　析

8行目
- int型の配列scoreを宣言、それぞれを初期化している。

14行目
- float型変数aveに配列scoreの値を一つずつ足していく。

15行目
- 変数aveを配列の要素数10で割って配列scoreの値の平均を求めている。

18-21行目
- 配列scoreの値から変数aveを引いて、平均からの差を求めている。
- 配列の要素番号、値、平均との差を出力。

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。

型	関数名	引数・パラメータ	機能	戻り値
float	get_ave	なし	平均値を求め表示	平均値
void	print_data	配列の添字、平均値	1つのデータと平均値との差を求め表示	なし

＊ プログラム ： average_1.c

```
 1  /*
 2    Program : average_1.c
 3    Comment : Average and difference are calculat.
 4  */
 5
 6  #include <stdio.h>
 7
 8  float get_ave();
 9  void print_data(float);
10
11  int score[10] ={3,5,8,9,10,6,7,9,8,3};
12  int i;
13
14  /* main-function */
15  int main(){
16    float ave = 0.0;
17
18    ave = get_ave();
19    print_data(ave);
20
21    return(0);
22  }
```

＊ 解　析 ： average_1.c

8行目
- float型の関数get_aveを定義。引数は定義していない。

9行目
- void型の関数print_dataを定義。引数は定義していない。

11行目
- int型配列scoreを定義(=グローバル変数)。値をそれぞれ初期化。

16行目
- float型変数aveを宣言。ave = 0.0で初期化。

18行目
- 変数aveに関数get_aveの戻り値(配列の平均値,float型)を代入。

19行目
- 関数print_dataに引数として、変数aveを渡す。
- 関数print_dataは、平均との差を求め、結果を出力している。

＊ プログラム ： get_ave.c

```
 1  /*
 2    Program : get_ave.c
 3    Comment : average of score
 4  */
 5
 6  #include <stdio.h>
 7  extern int score[10];
 8
 9  float get_ave(){
10    int i;
11    float ave = 0.0;
12
13    for(i=0; i<10; i++)
14      ave = ave + (float) score[i];
15
16    ave = ave / 10.0;
17    printf("ave = %3.1f\n",ave);
18
19    return(ave);
20  }
```

＊ 解　析 ： get_ave.c

7行目
- extern指定子を付けることで、「他の場所で宣言している」ということをコンパイラに知らせている。この場合、main関数の所で宣言している。

10-11行目
- int型変数i(=配列の添字)と、float型変数ave(=平均値)を宣言する。

13-14行目
- float型変数aveに配列scoreの値を一つずつ足していくという処理を行っている。

16行目
- 変数aveを配列の要素数10で割った値を変数aveに代入。

17行目
- 平均値を3桁で小数第一位まで出力する。

＊ プログラム ： print_data.c

```
 1  /*
 2    Program : print_data.c
 3    Comment : difference from average
 4  */
 5
 6  #include<stdio.h>
 7  extern int score[10];
 8
 9  void print_data(void){
10    int i;
11    float dif = 0.0,ave;
12
13    for(i=0; i<10; i++){
14      dif = score[i] - ave;
15      printf("score[%02d]=%2d Difference from average = %4.1f\n",i,score[i],dif);
16    }
17
18    return;
19  }
```

＊ 解　析 ： print_data.c

7行目
- 関数get_ave同様に、配列 scoreはmain関数で宣言しているので、extern指定子を付ける。

10-11行目
- int型変数 i(=配列の添字)と、float型変数dif(=平均値との差)を宣言する。

13-15行目
- for文。i=0から i=9になるまで i をインクリメントする。
- 変数dif に score[i] - aveの値を代入。(=平均との差)
- printf関数で配列の添字 i と配列の値 score[i]、平均との差 dif を出力。

＊ makeファイル ： makefile

```
 1  average_1: average_1.o get_ave.o print_data.o
 2          cc -o average_1 average_1.o get_ave.o print_data.o
 3
 4  average_1.o: average_1.c
 5          cc -c average_1.c
 6
 7  get_ave.o: get_ave.c
 8          cc -c get_ave.c
 9
10  print_data: print_data.c
11          cc -c print_data.c
```

＊考　察：makefile

1-2行目
- 実行可能ファイルaverage_1は、3つのオブジェクトファイル(average_1.o , get_ave.o , print_data.o)に依存している。
- 3つのオブジェクトファイルのいずれかに変更があれば、
　→「cc -o average_1.o get_ave.o print_data.o」を実行する。

4-5行目
- オブジェクトファイルaverage_1.oは、ソースファイルaverage_1.cに依存している。
- ソースファイルaverage_1.cに変更があれば、
　→「cc -c average_1.c」を実行する。

7-8 , 10-11行目
- 4-5行目と同様にそれぞれ、get_ave.oはget_ave.cに、print_data.oはprint_data.cに、依存していて、それぞれ変更があれば、2行目のコマンドを実行する。

＊コンパイル結果

```
new-host:Report5 kouta$ make -f makefile cc -c average_1.c cc -c get_ave.c cc -c
print_data.c cc -o average_1 average_1.o get_ave.o print_data.o
```

＊実行結果：average_1.c

```
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
```

＊考 察

サンプルプログラムのような1つのプログラムより、関数の翻訳単位にファイルを分けた、プログラムの方が、どの様な処理をしているかが分かり易い気がした。

また、makefileは複数のソースファイルに変更があっても、makeコマンドを実行するだけで、実行可能ファイルが作成できるので、ソースファイル、オブジェクトファイルがたくさんある大規模なプログラムの場合、とても便利である。

今回使った指定子extern(=外部参照)の他にも、auto(=自動変数)、static(=静的変数)、register(=レジスタ変数)、typedef(=型定義)などの記憶クラスがあることが分かった。

２．変数スコープと記憶域クラスについて。

＊ 変数スコープ

（1）ローカル変数

- 関数内で定義され、その関数内でのみ使用可能な変数。
- 普通はブロック文の最初で宣言する。
- 複数の関数が同一の関数名を用いても構わない。

（2）グローバル変数

- 関数外で定義され、どの関数からでも使用可能な変数。
- ローカル変数とグローバル変数とで同じ変数名がある場合、その関数内では、ローカル変数が優先される。

グローバル変数は、どの関数からでも参照できるので便利であるが、変数の衝突が起こり易く、どの関数からでも値を変えられるため分かりにくい。
なので、通常はローカル変数を主に使い、関数間のやりとりは引数で行った方が良いと思う。

＊ 記憶域クラス

（1）自動変数

- 関数内で宣言され、その関数内でのみ使用可能(ローカル変数)
- 実行中のみメモリが確保され、実行終了と同時にメモリが解放される。
- 関数が呼ばれるたびに初期化される。
- 明確的に初期化しないと初期値は不安定である。
- auto int a; のように宣言する（通常autoは省略される）。

（2）外部変数

- 関数外で定義され、定義以降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される。
- 初期値を明確的にしてないと初期値は0になる。

（3）静的変数

- 関数内で宣言され、その関数内のみで使用可能(ローカル変数)
- プログラム実行中は常に同じ場所に配置された値を保持。
- プログラム開始処理の前に一度だけ初期化される。
- 初期値を明確的にしてないと初期値は0になる。
- static int a; のように宣言する。

普段使用している変数は自動変数であったことが分かる。外部変数と静的変数は初期値を設定しなくても0であるが、最初から初期値を設定しておいたほうが良いと思われる。

また、静的変数は、自動変数のように、実行終了ごとにメモリが解放されないので、関数内で値を保持したいときなどに利用すると便利だなと思った。

３．オリジナルのプログラム、平均からの差でA,B,C,D,Eの評価を付けるプログラム。

＊ プログラム ： rank.c

```
 1  /*
 2    program   : rank.c
 3    comments  : 平均からの差でA,B,C,D,Eの評価を付ける.
 4  */
 5
 6  #include <stdio.h>
 7
 8  int score[10]={3,5,8,9,10,6,7,9,8,3};
 9
10  int main(){
11    int   i;
12    float ave = 0.0, dif;
13
14    for(i=0; i<10; i++) ave = ave + (float)score[i];
15    ave = ave / 10.0;
16    printf("ave = %3.1f\n",ave);
17
18    for(i=0; i<10; i++){
19      dif = score[i] - ave;
20      printf("score[%02d]=%2d  Difference from average = %4.1f",i,score[i],dif);
21      if(3.0 <= dif)
22        printf("  Rank : A\n");
23      else if(3.0 > dif && dif >= 1.5)
24        printf("  Rank : B\n");
25      else if(1.5 > dif && dif > -1.5)
26        printf("  Rank : C\n");
27      else if(-1.5 >= dif && dif > -3.0 )
28        printf("  Rank : D\n");
29      else if(-3.0 >= dif)
30        printf("  Rank : E\n");
31    }
32
33    return(0);
34  }
```

＊ 実行結果 ： rank.c

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

＊考　察

8-20行目
- サンプルプログラムと同様なので省略する。

21-31行目
- if文。
- dif(=平均との差)の値が、3.0以上である場合　…… ランクA
- 3.0より小さい かつ 1.5以上である場合ランクB　…… ランクB
- 1.5より小さい かつ -1.5より大きい場合ランクC　…… ランクC
- -1.5以下 かつ -3.0より大きい場合ランクD　…… ランクD
- -3.0以下である場合　…… ランクE

今回は、if文でランク分けをした。条件式を数字のみで組み立てたが、#define文などを使用すれば、配列の値や平均値に応じてランクの条件を柔軟に変えることができると思う。

4. 感　想

今回の課題は、まず、”翻訳単位”という言葉の意味から調べ始めて、サンプルプログラムを翻訳単位に分けるときに、引数のエラーだったり、変数のエラーだったりと色々なエラーが出て、正しく動作するプログラムを作るのに苦労した。
特に引数のエラーは、void型の変数にすることで解決したが、原因がなんだったのかよく分からなかったので、もっと変数の型や、関数の引数などについて勉強していきたいと思いました。

5. 参考文献

- Steve Oualline 著、谷口功 訳　『C実践プログラミング　第3版』　オーム社
- 初心者のためのポイント学習C言語　http://www9.plala.or.jp/sgwr-t/（ 09/06/17 access ）