目次　　ページ

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。

型	関数	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	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

ソースプログラム：

```
/*
  program   : average.c
  comments  : 結合(linkage)を用いて、平均・差を求める。
*/

#include <stdio.h>
int score[10]={3,5,8,9,10,6,7,9,8,3};

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

  for(i=0; i<10; i++) ave = ave + (float)score[i];
  ave = ave / 10.0;
  printf("ave = %3.1f\n",ave);

  for(i=0; i<10; i++){
    dif = score[i] - ave;
    printf("score[%02d]=%2d  Difference from average
= %4.1f\n",i,score[i],dif);
  }

  return(0);
}
```

ソースプログラムの実行結果：

```
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
```

問題が理解できないので問題の文章を理解することからはじめる。

分からない用語

外部変数	関数の外部で宣言をした変数であり、記憶クラス指定子はない。Extern 宣言を行うと、他のモジュールから外部変数を参照することが可能。（＝グローバル変数）
自動変数	関数内部で宣言され、宣言された関数の中のみで使用可能。（＝ローカル変数）
翻訳単位	ソースプログラムをいくつかのファイルに分割してコンパイルする時、ファイルの１つ１つを翻訳単位という。
float	浮動小数点型の時に使用。％変換は％f を使う。
void	パラメータリストが空であることを指定。void(型なし)と記述する。 void 型の時は return 文を省略してもよい。
スコープ	ある変数や関数が特定の名前で参照される範囲のこと
記憶域クラス	指定されたオブジェクト（識別子）を格納する記憶場所の時間的な寿命のこと。 自動的、静的、動的がある
識別子	int data ;　でいう data のこと
make	ファイルが最後にコンパイルされた後に修正されたどうかを認識してコンパイルを行う

以上のことを参考にして問題の文章を考えると、

> ソースプログラムを、平均値を求め表示するプログラム、1 つのデータと平均値の差を求め表示するプログラムに分け、別々に作り、Makefile を使って同じ内容のプログラムを作りなさい。

ということ。

『float』『void』という型の関数を使って、ソースプログラムを三つに分ける。

average.c のプログラム（prgrmng17.c）

```
/*
    program    : average.c
    Student-ID : e075739A
    Author     : TSUHAKO Masaki
    Update     : 2007/6/25
    comments   : 結合(linkage)を用いて、平均・差を求める。
*/

float get_ave();
void print_date(int i, float ave);

int score[10]={3,5,8,9,10,6,7,9,8,3};

int main(){
  int i;
  float ave;

  ave = get_ave();

  for(i=0; i<10; i++)
    print_date(i,ave);

  return(0);
  }
```

float get_ave のプログラム（prgrmng15.c）

```
/*ヘッダー*/
#include <stdio.h>

extern score[10];
```

```
float get_ave(){
  int   i;
  float ave = 0.0;

  /*平均を求める*/
  for(i=0; i<10; i++) ave = ave + (float)score[i];
  ave = ave / 10.0;

  /*平均表示*/
  printf("ave = %3.1f\n",ave);

/*戻り値＝平均値*/
  return (ave);
}
```

void print_data のプログラム（prgrmng16.c）

```
/*ヘッダー*/
#include <stdio.h>

extern score[10];

/*平均値との差を求める*/
void print_date(int i, float ave){
  float dif = 0.0;
  dif = score[i] - ave;
  printf("score[%02d]=%2d  Difference from average = %4.1f\n",i,score[i],dif\
);
 }
```

average.c のプログラムと float get_ave のプログラムと void print_data のプログラムを Makefile を使って分割されたファイルをコンパイルする。

Makefile

```
#
# prgrmng17.c(プロレポ４) のコンパイル＆実行のための makefile
#


prgrmng17: prgrmng17.o prgrmng16.o prgrmng15.o
	gcc -o prgrmng17 prgrmng17.o prgrmng16.o prgrmng15.o


prgrmng17.o: prgrmng17.c
	gcc -c prgrmng17.c


prgrmng16.o: prgrmng16.c
	gcc -c prgrmng16.c


prgrmng15.o: prgrmng15.c
	gcc -c prgrmng15.c
```

出力結果：

```
[Masaki:~/PRGRMNG] e075739% ./prgrmng17
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
```

よって実行可能ファイルの完成

Makefile のやり方の手順

```
[Masaki:~/PRGRMNG] e075739% emacs -nw Makefile
```

emacs –nw Makefile　と入力

```
target: source1.o source2.o source3.o
<Tab>command

source1.o:source1.c
<Tab>command

source2.o:source2.c
<Tab>command

source3.o:source3.c
<Tab>command
```

上の形式でオブジェクトファイル、C 言語のソースファイル、コマンドの依存関係を記述

```
[Masaki:~/PRGRMNG] e075739% make -f Makefile
```

C 言語のソースファイルに何か変更があった時、

```
gcc -c source1.c
gcc -o target source1.o source2.o source3.o
```

が表示。

何も変更がなかった時、

```
make: `target' is up to date.
```

が表示。

実行

```
[Masaki:~/PRGRMNG] e075739% ./target
```

Makefile の一連の流れは以上である。

変数のスコープと記憶域クラスについてのサンプルプログラム：

```
 1  #include <stdio.h>
 2
 3  int   en = 10;      /* File Scope */
 4  void  inc_n(void);
 5
 6  main(){
 7    printf("befor function           en=%2d\n\n",en);
 8    inc_n();
 9    inc_n();
10    inc_n();
11    printf("after function           en=%2d\n\n",en);
12  }
13
14  void inc_n(void){
15    static  int sn = 0;  /* Block Scope */
16    auto    int an = 0;  /* Block Scope */
17    extern  int en;      /* File Scope  */
18
19    printf("befor n++  sn=%2d  an=%2d  en=%2d\n",sn,an,en);
20    sn++;
21    an++;
22    en++;
23    printf("after n++  sn=%2d  an=%2d  en=%2d\n\n",sn,an,en);
24  }
```

実行結果

```
befor function           en=10


befor n++  sn= 0  an= 0  en=10
```

```
after n++  sn= 1  an= 1  en=11


befor n++  sn= 1  an= 0  en=11
after n++  sn= 2  an= 1  en=12


befor n++  sn= 2  an= 0  en=12
after n++  sn= 3  an= 1  en=13


after function           en=13
```

このプログラムでは、auto、static、extern という三つの記憶域クラス指定子が使われている。

変数のスコープと記憶域クラスについてのサンプルプログラムを見てみると、

```
int sn が static   初期値 0
int an が auto   初期値 0
int en が extern   初期値 10
```

となっている。

19 行目から 22 行目より、値を＋1 ずつ足していき、出力結果より三回繰り返しているのがわかる。ということは、加算していくのだから、最終結果（加算していく三回目）は、

```
sn＝3  an＝3  en＝13
```

ということが予想される。しかし、実際の実行結果は

```
sa＝3  an＝1  en＝13
```

である。

この実際の結果と予想が違う原因は auto、static、extern の性質の違いにある。

auto、static、extern の違いは、

auto…自動変数、または、一時的変数とも言われる。この変数は呼び出されるたびに初期化される。

記憶域クラスについてのサンプルプログラムでは、8、9、10 行目と、それれぞれ三回呼び出されている。

例

一回目 an＝0 → an＝1 → 初期化 → 二回目 an＝0 → an＝1

static…静的変数、または、恒久的変数とも言われる。この変数はプログラムの実行開始時に 1

度だけ初期化される。

extern…どこか他の場所で定義されている外部変数を、ここから（extern が使われている場所）でも参照できるようにする」といった意味をもつ。

つまり、

static は値を保持し、auto は関数の実行が終了すると値が削除される（関数が呼ばれるたびに初期化）。extern は値を保持し、定義以降のどの関数からでも使用可能である。extern は、他の翻訳単位で定義されれたオブジェクト（変数、関数等）のデータの読み出しをしたいときにも使う（今回の float get_ave と void print_data にも使っている）。

まとめ

変数のスコープ

変数の有効範囲のことを変数のスコープという。変数を最初にどこで使ったかによってその変数の有効範囲がきまる。

関数内のブロックで宣言された変数の識別子が、そのブロック内で特有の時、その変数をローカル変数という。それに対して、関数の外部で宣言された変数をグローバル変数という。

プログラムを書く上でグローバル関数はできる限り用いるべきではない。なぜなら、ソース内のどの関数からでも数値を変更できてしまうため、何がどうなっているかわからなくなってしまうから。

記憶域クラス

指定された識別子を格納する記憶場所の寿命のこと。

種類は extern（外部変数）、static（静的変数）、auto（自動変数）等がある。

auto　…自動変数。変数の適応範囲はローカル（local）である。

static…静的変数。変数の適応範囲は、宣言されている場所による。

extern…外部変数。変数の適応場所はグローバル（global）である。

感想：

今回は変数のスコープと記憶域クラスについての課題だったが、自分にとっては Makefile をテーマにした課題だった。なんといってもエラーが多い…。しかもどこが間違っているのかがわからないときた。強敵だ。そして課題をクリアしていく為にとった方法が『わからない用語を調べまくる作戦』だ。そのまんまだ。この方法は思っていた以上に効率が悪かった。おかげで時間はかかったが、インターネットで参考になるページを見つけることができるといういい収穫もあった。
まだよくこの課題についてよくわかっていないというのが正直な感想だ。重要なポイントっぽいので復習を沢山しようと思った。

参考文献

先輩たちのページ

http://www.ie.u-ryukyu.ac.jp/~e065745/

http://www.ie.u-ryukyu.ac.jp/~e065762/

初心者の為のポイント学習 C 言語

http://search.jword.jp/cns.dll?type=lk&fm=101&agent=1&partner=Excite&name=%BD%E9%BF%B4%BC%D4%A4%CE%A4%BF%A4%E1%A4%CE%A5%DD%A5%A4%A5%F3%A5%C8%B3%D8%BD%ACC%B8%C0%B8%EC&lang=euc&prop=500&bypass=0&dispconfig=

記憶域クラスについて

http://cai.cs.shinshu-u.ac.jp/sugsi/Lecture/c/e_02-01.html

記憶域クラス指定子

http://homepage3.nifty.com/c-master/column/storage_class.html