変数のスコープと記憶域クラスについての考察

a.スコープについて

宣言した変数の有効範囲のことをスコープよぶ。変数にはグローバル変数（外部変数）とローカル変数（局所変数）の２種類がある。

グローバル変数とはプログラム全体で使用することができる変数であり、そのプログラム内では先の値が保存され続ける。

ローカル変数とは変数を宣言した場所でのみ有効となる変数である。つまり、ある関数内で宣言した場合その関数の中でしか使用することができず、他の場所に値を引き継ぐことはできない。

サンプルプログラム「3．有効範囲の入れ子」のなかでint_nの中で変数nを宣言した後、その中のプロック内でもう一度nを宣言している。出力結果を見るとブロック内で宣言したnの値が出力されている。つまり、外部で宣言する変数よりも内部で宣言した変数の方が優先されるということである。

b.記憶域クラスについて

動的記憶：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）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解放する。

静的記憶：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する。

◎autoは動的記憶、externは静的記憶、staticの場合は宣言する場所によって動的か静的か分かれるが基本的に静的記憶に分類される。

c.記憶域クラスの違い

サンプルプログラムと実行結果から次のことがわかる

auto：使用した場所で１回終了するまで有効である。例えばある関数内で使用したならその関数が１度終了すれば数値はリセットされ、次に引き継がない。

static：宣言した場所においてつねに有効である。つまり、ある関数内で宣言したのならその関数をもう１度使用したときにも値は引き継がれる。しかしその他の関数内において先の値を引き継ぐことはできない。

extern：関数外、プログラムの一番外側で宣言した変数を引き継ぐことができる。しかしある関数内でexternを使っても他の関数内で指定した変数を呼び出すことはできない。常にメモリ領域を確保するため、メモリに負担をかける。

つまり、autoのように変数を呼び出すたびに初期化されるクラスはメモリ領域を割き続ける必要がないため使いやすいクラスである。staticとexternはメモリ領域の使用を抑えるためstaticを使用するのが好ましいと思われる。しかし今回のレポートのように分割してMakefileでコンパイルするような場合においてはその他の場所で数値の読み込みを可能にするためexternを使う方がよいだろう。

Report#4

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、各データと平均の差を表示するプログラムである。

1.プログラム「average.c」のソース

```
/*
  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);
}
```

2.average.cの実行結果

```
[softbank220057168077:~/prog/report4] ib0603% ./average
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
```

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_date	配列の添字、平均値	１つのデータと平均値の差を求め表示	なし

Report#4のページに「同様な動作をするオリジナルのプログラムを作成してもよい」とあるので、平均値からの差で評価を決めるプログラムを作ってみたいと思う。

ave.cのソース

```
/*
  Program    : ave.c
  Student-ID : 065745A
  Autor      : NAKASONE,Tomoya
  UpDate     : 2006/06/07(Wed)
  Comment    : heikin
*/

float get_ave();
void  print_date();

int score[10]={52,23,37,61,89,12,95,40,76,2};

int main() {
  int i;
  float ave;

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

  return(0);
}
```

get_ave.cのソース

```
#include <stdio.h>

extern score[10];

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

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

  return(ave);
}
```

print_date.cのソース

```
#include <stdio.h>

extern score[10];

void print_date(int i, float ave) {
  float dif;

  dif = score[i] - ave;
  printf("score[%02d]=%2d   Differense from average = %5.1f",i ,score[i], dif);
```

```
  if      (-40.0 <= dif && dif <= -20.1){
    printf("  ドンマイ・・・      ランクD\n");
  }else if(-20.0 <= dif && dif <= -0.1) {
    printf("  あと少し            ランクC\n");
  }else if(0.0 <= dif && dif <= 19.9)   {
    printf("  まぁまぁ            ランクB\n");
  }else if(20.0 <= dif && dif <= 39.9)  {
    printf("  スゴい！            ランクA\n");
  }else if(40.0 <= dif)                 {
    printf("  素晴らしい！！      ランクS\n");
  }else {
    printf("  出直してください    ランクF\n");
  }
}
```

これらをまとめてコンパイルするためのMakefileをつくる

Makefileのソース

```
#
# ave.c のコンパイル＆実行のための Makefile
#

ave: ave.o get_ave.o print_date.o
        cc -o ave ave.o get_ave.o print_date.o

ave.o: ave.c
        cc -c ave.c

get_ave.o: get_ave.c
        cc -c get_ave.c

print_date.o: print_date.c
        cc -c print_date.c
```

Makefileでのコンパイル

```
[ib0603-no-ibook-g4:~/prog/report4] ib0603% make -f Makefile
cc -c ave.c
cc -c get_ave.c
cc -c print_date.c
cc -o ave ave.o get_ave.o print_date.o
```

ave.cの実行結果

```
[ib0603-no-ibook-g4:~/prog/report4] ib0603% ./ave
ave = 48.7
score[00]=52   Differense from average =   3.3  まぁまぁ          ランクB
score[01]=23   Differense from average = -25.7  ドンマイ・・・    ランクD
score[02]=37   Differense from average = -11.7  あと少し          ランクC
score[03]=61   Differense from average =  12.3  まぁまぁ          ランクB
score[04]=89   Differense from average =  40.3  素晴らしい！！    ランクS
score[05]=12   Differense from average = -36.7  ドンマイ・・・    ランクD
score[06]=95   Differense from average =  46.3  素晴らしい！！    ランクS
score[07]=40   Differense from average =  -8.7  あと少し          ランクC
score[08]=76   Differense from average =  27.3  スゴい！          ランクA
score[09]= 2   Differense from average = -46.7  出直してください  ランクF
```

考察

ave.cでは変数と関数の宣言を行い、score[10]を他のファイルでも共有できるようにした。

get_ave.cではscore[10]の中に入ってる値の平均を取った。

※(float)score[i]というのはint型のscore[10]をfloat型に一時的に変換するということである。iの代わりに10を入れても同様に出力されると思い、入れ替えてみたところ-1610604544.0という数値が出力された。

どうやらfloat型は整数ではなく少数を扱うことしかできないため代わりにfor文から外に出てきたiで代用しているようである。

print_date.cでは、配列の添字と求めた平均値の値をうけとり、１つのデータと平均値の差を出力した。それをave.cのfor文で１０回繰り返し行った。

課題に少し改良を加えたのはここからである。

for文の中にReport#3で使用したif文とelse文を使用し平均値からの差に応じてランク分けを行った。ここで基準となる数値は「dif」とした。その理由は「dif」は『dif = score[i] - ave』とfor文の中で宣言されているため、繰り返されるごとにiの数値が変化し、それぞれの平均値からの差も表示することができるためである。

考えた通りに出力することができた。

感想・反省

今回のレポートは他の教科の課題と重なっている上、すぐ来週にはテストも待ち受けているため量的には多くなかったものの、正直今までで一番精神的につらかったです。