[1] 課題内容

与えられたプログラムを参考に、自分自身のオリジナルプログラムを作成し、変数スコープについて考察せよ。

[2]

プログラム 1

```
/*****************************************
  Program    : scope_original.c
  Student-ID : 045713C
  Author     : OSHIRO,Kazuya
  Date       : 04/06/09
  Comment    : 課題、オリジナルプログラム
*****************************************/

#include <stdio.h>

int en = 10;                    /* int(整数)型、 グローバル変数enを10に初期化 */
void inc_n(void);        /* 戻り値なしで引数なしの関数inc_nを宣言する */
void dec_n(void);

int main()
{
  printf("before function      en=%2d¥n¥n", en); /* enの初期値を出力 */
  inc_n();
  dec_n();
  dec_n();
  inc_n();                      /* 関数int_nをそれぞれ呼び出す */
  printf("aftor  function      en=%2d¥n¥n", en); /* 最終的なenの値を出力 */
}

void inc_n()                    /* 関数inc_nの定義 */
{
  static int sn = 0;            /* 整数型の静的変数snを0に初期化 */
  auto   int an = 0;            /* 整数型の動的変数anを0に初期化 */
  extern int en;         /* グローバル変数en */

  printf("before n++ sn=>%2d an=>%2d en=>%2d¥n", sn, an, en); /* 各変数sn,an,enを出力
する */
  sn++;                                                  /* snに1を足す */
  an++;                                                  /* anに1を足す */
  en++;                                                  /* enに1を足す */
  printf("after  n++ sn=>%2d an=>%2d en=>%2d¥n¥n",sn,an,en);       /* 上の処理をし
た後のそれぞれを出力する */
}

void dec_n(void)         /* 関数dec_nの定義 */
{
  static int sn = 0;        /* 静的関数で整数型のsnを0に初期化(上のsnとは別物) */
  auto   int an = 0;         /* 整数型の動的変数anを0に初期化(上のanとは別物) */
  extern int en;     /* グローバル変数en */

  printf("before n-- sn=>%2d an=>%2d en=>%2d¥n", sn, an, en); /* sn,an,enの値を出力
```

```
*/
  sn--;                                                  /* snから1を引く */
  an--;                                                  /* anから1を引く */
  en--;                                                  /* enから1を引く */
  printf("after  n-- sn=>%2d an=>%2d en=>%2d¥n¥n", sn, an, en); /* sn,an,enの値を出力
*/
}
```

結果１

```
[Kazuya-OSHIRO:~/prog1/report/#3] j04013% ./scope_original
before function      en=10

before n++ sn=> 0 an=> 0 en=>10
after  n++ sn=> 1 an=> 1 en=>11

before n-- sn=> 0 an=> 0 en=>11
after  n-- sn=>-1 an=>-1 en=>10

before n-- sn=>-1 an=> 0 en=>10
after  n-- sn=>-2 an=>-1 en=> 9

before n++ sn=> 1 an=> 0 en=> 9
after  n++ sn=> 2 an=> 1 en=>10

aftor  function      en=10
```

考察１

このプログラムは初めにenに10を代入し、void型(戻り値のない)関数inc_n(void)とdec_n(void)を宣言する。その後まずenを出力そして関数inc_n()を呼び出す。inc_n関数内でstaticのsnを0に初期化、autoのanを0に初期化、そして外部変数のenを使うことを宣言する。それぞれを出力した後、1を足しまた出力する。ここまできて処理をmainに戻し関数dec_nに処理を移す。inc_nと同じような処理をする(ただし、1を引く)。同様のことを行い、最後にmain関数のprintfでenを出力する。

結果の数値よりsnは関数inc_nで行った変更がdec_nでは影響していない、同じくdec_nでの変更もinc_nには影響していないのでinc_nでのsnとdec_nでのsnは別の変数だということがわかる。同様にanも同じことが言える、さらにanは呼び出されるたび0を出力するので毎回初期化が行われているのがわかる。enは関数外で宣言されており結果よりすべての関数の中で使われているenは同一のものだとわかった。

プログラム2

```
/*********************************
*  Program    : scope2.c         *
*  Student-ID : 045713C          *
*  Author     : OSHIRO,Kazuya    *
*  Date       : 04/06/06         *
*  Comment    : 有効範囲の入れ子 *
*********************************/
```

```
#include <stdio.h>

int   n = 0;                    /* 整数型のグローバル変数nを0に初期化 */
void inc_n(void);        /* 戻り値なし、引数なしの関数inc_nを宣言 */

int main()
{
  printf("before function        n=%2d¥n¥n", n); /* グローバル変数のnを出力する */
  inc_n();                                    /* 関数inc_nを呼び出す */
  printf("after  function        n=%2d¥n", n); /* グローバル変数のnを出力する */
}

void inc_n(void)         /* 関数inc_nの定義 */
{
  static int n = 10;             /* 静的変数で整数型の変数nを10に初期化 */

  printf(" before n++            n=%2d¥n", n); /* nの値を出力 */
  {
    int n=20;                    /* ローカル変数で整数型のnを20に初期化 */
    printf("         in block      n=%2d¥n", n); /* nの値を出力 */
  }
  n++;                           /* nに1を足す */
  printf(" after  n++            n=%2d¥n¥n", n); /* 10に初期化したnに1を足す */
}
```

結果2

```
[Kazuya-OSHIRO:~/prog1/report/#3] j04013% ./scope2
before function        n= 0

 before n++            n=10
         in block      n=20
 after  n++            n=11

after  function        n= 0
```

考察2

このプログラムは初めにnを０に初期化、そしてvoid型関数inc_nを宣言する。main関数内でnの値を出力する。そして処理をinc_n関数に移る。inc_nではstaticの変数nを10に初期化し、その値を出力する。ブロック文の中でまた変数n=20を宣言、初期化し出力する。ブロック文を抜け、staticのnに1を足し出力を行う。処理をmain関数に戻し、最後にnの値を出力し終了する。

結果を見ると初めと終わりの数値はどちらとも0、これは関数外で宣言したnが出力されているのがわかる。次にinc_n関数の中のブロック内で宣言されたnだがその下の結果を見ると11になっていてこれはstaticのnに処理が行われたのがわかり、そのためブロック内で宣言された変数はそのブロック内で有効なのがわかる。staticで宣言されたnにも同様なことが言え、有効な範囲は関数内だとわかる、しかしブロック内はその中で宣言された変数が優先されるので関数の変数nは隠蔽されている。

[3] 全体のを通しての考察

a.変数スコープの説明

変数が有効である範囲のことである。

b.スコープの種類と有効範囲

ファイル・スコープ---------- ソースファイル全体で有効
ブロック・スコープ---------- 関数やブロック内で有効
ファンクション・スコープ---------- 関数内の全体で一意的に有効
ファンクション・プロトタイプ・スコープ---------- 関数内の全域で一意的に有効

c.記憶クラス指定子の種類と説明

auto(自動変数)

関数が呼び出されると、関数内で宣言された変数がスタック上に作成され、関数が実行を終えると変数領域ごと消去される。

static(静的変数)

関数の呼び出しに関係なく、常時存在する。関数の中で宣言したときスコープは関数内になる。初期化は一度のみであり、値はプログラムが終了するまで保持される。

register(レジスタ変数)

レジスタ変数は自動変数の内、プログラム実行中に頻繁に使う変数をレジスタに格納し、処理速度を高めようと設計されています。この記憶クラスを指定することで、頻繁に使う変数であることをコンパイラに知らせる。

exterm(外部参照変数)

extermは、ほかのソースファイルにある同一の変数を参照する為に用いる。長いCプログラムを分割コンパイルする時、分割したプログラム間でデータを受け渡す事ができる。

d.記憶クラス指定子を用いた宣言方法

記憶クラス指定子　変数の型　変数名；

ちなみに関数内で記憶クラス指定子を用いないで宣言した場合、そのクラスはautoと見なされる。

[4] 感想・反省

今回のレポートの反省はオリジナルプログラムですかね。正直何をしていいのかわからなかったため与えられていたプログラムを合体させただけ・・・、後半に至ってはそのまんま使用ですからね。その点で今回のレポートは難しかったです。だけど、前々から疑問に思ってた変数の宣言する位置の意味がようやくわかったので良かったと思います。後、あれですね。やっぱ始めるのが遅い。前回も前々回もこれは書いてる、この調子じゃ毎回書くことになりそうだ・・・そうならないようがんばろうと思う。

参考文献

C実践プログラミング 第3版　(Steve Oualline　著)
変数と記憶クラス　(http://www.komonet.ne.jp/~c/chap4.htm)