構造体・共有体

構造体

```
1.  #include <stdio.h>
2.
3.  struct smp{/*smpはタグ名、データ型はすべてint型、day,month,year,がメンバ名*/
4.          int day;
5.          int month;
6.          int year;
7.          }date,*pdate;
8.
9.  isleap1(struct smp d){
10.         int r4,r100,r400;
11.
12.         r4   = d.year % 4;
13.         r100 = d.year % 100;
14.         r400 = d.year % 400;
15.
16.         return ( ((r4 == 0) && (r100 != 0)) || (r400 == 0) );
17.
18. }
19.
20. isleap2(struct smp *d){
21.         int r4,r100,r400;
22.
23.         r4   = d->year % 4;
24.         r100 = d->year % 100;
25.         r400 = d->year % 400;
26.
27.         return ( ((r4 == 0) && (r100 != 0)) || (r400 == 0) );
28. }
29.
30. main(){
31.         date.day = 24;
32.         date.month = 2;
33.         date.year  = 1900;
34.
```

```
35.         printf("%4d年は閏年で%s¥n",date.year,(isleap1(date) != 0)? "す。":"ない。");
36.         printf("%x %x %x¥n",&date.day,&date.month,&date.year);
37.
38.         printf("%4d年は閏年で%s¥n",date.year,(isleap2(&date) != 0)? "す。":"ない。");
39.         printf("%x %x %x¥n",&date.day,&date.month,&date.year);
40.  }
```

実行結果

1900 年は閏年でない。

40d9c0 40d9c4 40d9c8

1900 年は閏年でない。

40d9c0 40d9c4 40d9c8

考察

3 行目~7 行目:構造体 smp の型枠の宣言

複数のデータ（メンバと呼ぶ）をまとめて 1 つの構造体の型枠を宣言している。

この宣言は単に構造体の型を作っただけであり、領域の割当ては行われていない。

この型枠の有効範囲は宣言する場所によって異なり、関数外で宣言した場合は、その位置より下の全関数で有効であり、関数内で宣言した場合は宣言した関数内でのみ有効である。

7 行目:構造体の型の宣言を行った後に、同時にそれを型名 date と*pdate として定義

9 行目~18 行目; isleap1 関数の宣言

12~14 行目:構造体系変数内の各メンバを参照している。

メンバを参照するには、変数名の後に「.」を付けた後にメンバ名を付ける。

ここではメンバ名を year、変数名を d として参照している。

参照している d.year を各数字で割ったあまりを r4、r100、r400 の変数に代入している。

16 行目で値を引き渡している。()の中身は閏年の条件 4 で割れて 100 の倍数を含まず 400 の倍数を含むを示している。

20 行目~28 行目; isleap2 関数の宣言

23~25 行目:構造体のポインタを参照している。

ポインタを参照するためには、変数名の後に「->」を付けて、そのあとにメンバ名を付ける。

ここではメンバ名を year、変数名を*d を参照している。

27 行目で値を引き渡している。

30 行目~main 関数

31~33 行目:構造体 int 型のメンバにそれぞれ値を代入している。

35 行目:変数 d に対する出力

date.year=1900 と宣言してあるので「1900 年は閏年で」と出力。

”す。”と”ない。”は「:」で区切られていおり、この:を使って、閏年であるか無いかと区別している。

入力された年が真であるなら、”す。”を、偽なら(閏年でないなら)”ない。”を出力する。

38 行目:ポインタ d による出力。

共有体

ソース

```
1.  #include <stdio.h>
2.
3.  union smp{
4.          char    b08;
5.          short b16;
6.          int     b32;
7.
8.  };
9.
10. main(){
11.         union smp var;
12.
13.         var.b08=2;
14.         puts("----------");
15.         puts("var.b08=2");
16.         printf("var.b08=0x          %02x=%d¥n",var.b08,var.b08);
17.         printf("var.b16=0x       %04x=%d¥n",var.b16,var.b16);
18.         printf("var.b32=0x%08x=%d¥n",var.b32,var.b32);
19.         printf("var ADDRESS¥n");
20.         printf("var.b08=0x%x +=%%0x%x¥n",&var.b08,&var.b08+1);
21.         printf("var.b16=0x%x +=%%0x%x¥n",&var.b16,&var.b16+1);
22.         printf("var.b32=0x%x +=%%0x%x¥n",&var.b32,&var.b32+1);
23.
24.         var.b16=260;
25.         puts("----------");
26.         puts("var.b16=260");
```

```
27.         printf("var.b08=0x          %02x=%d¥n",var.b08,var.b08);
28.         printf("var.b16=0x       %04x=%d¥n",var.b16,var.b16);
29.         printf("var.b32=0x%08x=%d¥n",var.b32,var.b32);
30.
31.         var.b32=66000;
32.         puts("---------------");
33.         puts("var.b32=66000");
34.         printf("var.b08=0x          %02x=%d¥n",var.b08,var.b08);
35.         printf("var.b16=0x       %04x=%d¥n",var.b16,var.b16);
36.         printf("var.b32=0x%08x=%d¥n",var.b32,var.b32);
37.
38.  }
```

実行結果

```
---------------
var.b08=2
var.b08=0x          02=2
var.b16=0x       0002=2
var.b32=0x00000002=2
var ADDRESS
var.b08=0x12ff74 +=%0x12ff75
var.b16=0x12ff74 +=%0x12ff76
var.b32=0x12ff74 +=%0x12ff78
---------------
var.b16=260
var.b08=0x          04=4
var.b16=0x       0104=260
var.b32=0x00000104=260
---------------
var.b32=66000
var.b08=0x          ffffffd0=-48
var.b16=0x       01d0=464
var.b32=0x000101d0=66000
```

解説

3 行目~8 行目:共有体の型枠の宣言

タグ名は smp

b08 は char 型なので 1 バイト、b16 は int 型なので 2 バイト、b32 は long 型なので 4 バイトである。

まだこの時点ではメモリ上に領域を確保してはいない。

10 行目~:main 関数

11 行目:smp 共有体、変数 Ver を宣言。変数この時点でメモリ上に領域を確保する。

構造体と違うのは char 型、int 型、long 型の各変数の先頭アドレスが同じである

13~18 行目:char 型メンバに値 2 を代入した場合の b08,b16,b32 の各値の表示。

先頭アドレスが一緒なので b16、b32 にも char 型つまり 1bit で 2 が入っているものと思われる。

char 型は-128 ~127 までしか入らないのでそれ以外の値を代入するとループする。

20~23 行目

各アドレスに 1 を足したときの結果を表示させる b08 は char 型なので 1 バイトされる

b16 は int 型なので 2 バイト足され、b32 は long 型なので 4 バイト足される。

24~29 行目:int 型で 260 を各変数に代入する。

Int 型なので -32768～32767 までの数字を代入することが可能である。

31 行目~

Long 型に 66000 を代入したときの各型の値を表示させる。Long 型は-2147483648～2147483647

までを代入することができる。

考察

「char 型は-128 ~127 までしか入らないのでそれ以外の値を代入するとループする」を実際にやってみると下記のようになる。

[上限]

var.b08=127

var.b08=0x 7f=127

var.b16=0x 007f=127

var.b32=0x0000007f=127

var.b08=128

var.b08=0xffffff80=-128

var.b16=0x 0080=128

var.b32=0x00000080=128

[下限]

var.b08=-128

var.b08=0x ffffff80=-128

var.b16=0x　　0080=128
var.b32=0x00000080=128

var.b08=-129
var.b08=0x　　7f=127
var.b16=0x　　007f=127
var.b32=0x0000007f=127

「Int 型なので -32768～32767 までの数字を代入することが可能である。」を実際に調べてみると下記のようになる。
[上限]
var.b16=32767
var.b08=0x　　ffffffff=-1
var.b16=0x　　7fff=32767
var.b32=0x00007fff=32767

var.b16=32768
var.b08=0x　　00=0
var.b16=0x　　ffff8000=-32768
var.b32=0x00008000=32768

[下限]
var.b16=-32768
var.b08=0x　　00=0
var.b16=0x　　ffff8000=-32768
var.b32=0x00008000=32768

var.b16=-32769
var.b08=0x　　ffffffff=-1
var.b16=0x　　7fff=32767
var.b32=0x00007fff=32767

「Long 型は-2147483648～2147483647 までを代入することができる」について実際に調べてみると下記のようになる。
[上限]

var.b32=2147483647

var.b08=0x ffffffff=-1

var.b16=0x ffffffff=-1

var.b32=0x7fffffff=2147483647

var.b32=2147483648

var.b08=0x 00=0

var.b16=0x 0000=0

var.b32=0x80000000=-2147483648

[下限]

var.b32=-2147483648

var.b08=0x 00=0

var.b16=0x 0000=0

var.b32=0x80000000=-2147483648

var.b32=-2147483649

var.b08=0x ffffffff=-1

var.b16=0x ffffffff=-1

var.b32=0x7fffffff=2147483647

union について

union により共有体となった変数は次のようにあらわせる。

先頭アドレスを 0x12ff7 とする。char 型の 1 つの箱を 1bit とする。

char 型	0x12ff74	0x12ff75	0x12ff76	0x12ff77	0x12ff78	0x12ff79	0x12ff7a	0x12ff7b
int 型	0x12ff74		0x12ff76		0x12ff78		0x12ff7a	
long 型	0x12ff74				0x12ff78			

Error 報告

```
#include <stdio.h>

int main(){

	{
		int *p,**q;
		*(100)=200;
		*(200)=300;
		p=100;
		q=100;
		printf("Q7.次の文を実行した後の*p,*q,**qの値を示せ。¥n");
		printf("*p = %d¥n",*p);
		printf("*q = %d¥n",*q);
		printf("**q = %d¥n",**q);
	}
}
```

上記のように自分でアドレスを割り当てる作業をさせても下記のようにエラーが出る。
error.c(8)：error C2100: 間接指定演算子 (*) の使い方が正しくありません。
error.c(8)：error C2106: '='： 左のオペランドが、左辺値になっていません。
これは上記で述べたようにメモリの領域を確保していないところに確保しようとするので
エラーが出る。

```
#include <stdio.h>

#define MAX 5;

int main(){

	{
		char i,*p,m[MAX] = {1,2,3,4,5};
		printf("Q19.次の文をポインタの代わりにint k;を宣言し、配列を用いた文に書き換えよ。¥n");
		printf("書き換え前¥n");
		for(p=m;*p;++p,i++){
```

```
                    *p+=1;
                    printf("*(p+%d) = %d¥n",i,*p);
                }

        }
}
```

上記のように define MAX 5 の後ろに「 ; 」を付けてしまい、下記のようにエラーが出でしまいきづくまで大変だった。

error.c(8)：error C2143: 構文エラー ：']' が ';' の前にありません。

error.c(8)：error C2143: 構文エラー ：';' が ']' の前にありません。

感想

テストで下らないミスで点数が落ちてしまったがそれだけではなく理解不足もうまく点数がのびなかった原因であると痛感した。

このレポートを通してよりポインタを理解することができたと思う。

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