● 実習内容

【C言語】

1. 以下の ex05-1.c を入力してコンパイルおよび実行を行ってみよう.
   - 乗法演算子に関する「算術変換」とは何かを考えよう.
   - どのような時に算術変換が行われるかを理解しよう.
2. 以下の ex05-2.c を入力してコンパイルおよび実行を行ってみよう.
   - 今回用いている環境では char 型は「符号つき」である.
   - char 型変数に「最上位ビットが 1 となる値」を代入すると, どのような変換が行われるかをよく理解しよう.
   - 整数型の値が「より広い」型に変換されるときには**値が保存**されることに注意しよう.
3. 以下の ex05-3.c を入力してコンパイルおよび実行を行ってみよう.
   - $\sum_{n=1}^{10} n$ を求めるプログラムである. for, while, do--while の動作の様子をよく理解しよう.
4. 以下の ex05-4.c を入力してコンパイルおよび実行を行ってみよう.
   - 与えられた整数が 6 の倍数かどうかを判定している. この 4 つの方法のうち, 意図通りに動作しないものが 2 つある. それはなぜかを考えてみよう.
5. 以下の ex05-5.c を入力してコンパイルおよび実行を行ってみよう.
   - これら 3 つの繰り返し文の挙動が異なる理由を考えてみよう.
   - もし, プログラムが「止まらなくなった」ら CTRL+C を入力すればよい.

電子メールで「今日の講義の感想や意見」を送ってください.

★ ex05-1.c の内容

```
/* 算術変換 */
/* $Id: ex05-1.c,v 1.3 2004-05-02 17:26:54+09 naito Exp $ */
/* ex05-1.c */

#include <stdio.h>

int     i,j,k ;
double  x,y,z ;

int main(int argc, char **argv)
{
  i = 3   ; j = 2   ; x = 3.0 ; y = 2.0 ;

  k = i/j ;              printf("%d\n", k) ;
  k = (double)i/j ;      printf("%d\n", k) ;
  k = i/(double) j ;     printf("%d\n", k) ;
  k = (double)(i/j) ;    printf("%d\n", k) ;

  z = i/j ;              printf("%f\n", z) ;
  z = x/j ;              printf("%f\n", z) ;
  z = i/y ;              printf("%f\n", z) ;
  z = x/y ;              printf("%f\n", z) ;

  return 0 ;
}
```

★ ex05-2.c の内容

```
/* 整数への格上げと符号拡張 */
/* $Id: ex05-2.c,v 1.1 2004-05-02 16:48:18+09 naito Exp $ */
/* ex05-2.c */

#include <stdio.h>

char    c ;
int     i ;
unsigned char   d ;

int main(int argc, char **argv)
{
  c = 'A' ;
  printf("%x (%d) ", c, c) ;
  i = c ; c = i ;
  printf("\t%x (%d) ", i, i) ;
  printf("\t%x (%d)\n", c, c) ;

  i = 0x41 ;
  printf("%x (%d) ", i, i) ;
  c = i ; i = c ;
  printf("\t%x (%d) ", c, c) ;
  printf("\t%x (%d)\n", i, i) ;

  i = 0x81 ;
  printf("%x (%d) ", i, i) ;
  c = i ; i = c ;
  printf("\t%x (%d) ", c, c) ;
  printf("\t%x (%d)\n", i, i) ;

  i = 0x81 ;
  printf("%x (%d) ", i, i) ;
  d = i ; i = d ;
  printf("\t%x (%d) ", d, d) ;
  printf("\t%x (%d)\n", i, i) ;

  c = 0x81 ;
  printf("%x (%d) ", c, c) ;
  i = c ;
  printf("\t%x (%d)\n", i, i) ;

  return 0 ;
}
```


★ ex05-3.c の内容

```
/* １から１０までを加える */
/* $Id: ex05-3.c,v 1.2 2004-05-02 17:08:29+09 naito Exp $ */
/* ex05-3.c */

#include <stdio.h>

int     i, sum ;

int main(int argc, char **argv)
{

  sum = 0 ;
  for(i=0;i<10;i++) sum += i+1 ;
  printf("sum = %d\n", sum) ;

  sum = 0 ; i = 0 ;
  while(i < 10) {
    i += 1 ;
    sum += i ;
  }
  printf("sum = %d\n", sum) ;

  sum = 0 ; i = 0 ;
  do {
    i += 1 ;
    sum += i ;
  } while(i < 10) ;
  printf("sum = %d\n", sum) ;

  return 0 ;
}
```

★ ex05-4.c の内容

```
/* 条件文および繰り返し文 */
/* $Id: ex05-4.c,v 1.3 2004-05-02 17:13:51+09 naito Exp $ */
/* ex05-3.c */

#include <stdio.h>

int     n ;
int     n2, n3 ;
int     k2, k3 ;

int main(int argc, char **argv)
{
  for(n=0;n<10;n++) {
    printf("n = %d\n", n) ;
    n2 = n3 = k2 = k3 = -1 ;

    if ((n%2 == 0)&&(n%3 == 0)) printf("n は 6 の倍数\n") ;
    else if (n%2 == 0) printf("n は 2 の倍数だが 3 の倍数ではない\n") ;
    else if (n%3 == 0) printf("n は 3 の倍数だが 2 の倍数ではない\n") ;
    else printf("n は 2 の倍数でも 3 の倍数でもない\n") ;

    if ((n&2 == 0)&&(n%3 == 0)) printf("n は 6 の倍数\n") ;
    else if (n&2 == 0) printf("n は 2 の倍数だが 3 の倍数ではない\n") ;
    else if (n%3 == 0) printf("n は 3 の倍数だが 2 の倍数ではない\n") ;
    else printf("n は 2 の倍数でも 3 の倍数でもない\n") ;

    k2 = n%2 ; k3 = n%3 ;
    if ((k2 == 0)&&(k3 == 0)) printf("n は 6 の倍数\n") ;
    else if (k2 == 0) printf("n は 2 の倍数だが 3 の倍数ではない\n") ;
    else if (k3 == 0) printf("n は 3 の倍数だが 2 の倍数ではない\n") ;
    else printf("n は 2 の倍数でも 3 の倍数でもない\n") ;

    if (((n2 = n%2) == 0)&&((n3 = n%3) == 0)) printf("n は 6 の倍数\n") ;
    else if (n2 == 0) printf("n は 2 の倍数だが 3 の倍数ではない\n") ;
    else if (n3 == 0) printf("n は 3 の倍数だが 2 の倍数ではない\n") ;
    else printf("n は 2 の倍数でも 3 の倍数でもない\n") ;

    printf("k2 = %d, k3 = %d\n", k2, k3) ;
    printf("n2 = %d, n3 = %d\n\n", n2, n3) ;
  }
  return 0 ;
}
```


★ ex05-5.c の内容

```
/* 浮動小数点数 */
/* $Id: ex05-5.c,v 1.2 2004-05-06 08:30:52+09 naito Exp $ */
/* ex05-5.c */

#include <stdio.h>

double  x ;
float   y ;

int main(int argc, char **argv)
{
  for(x=0.0;x<=1.0;x+=0.1) printf("%f ", x) ;
  printf("\n") ;

  for(y=0.0F;y<=1.0F;y+=0.1F) printf("%f ", y) ;
  printf("\n") ;

  for(x=0.0;x != 1.0;x+=0.1) printf("%f ", x) ;
  printf("\n") ;

  return 0 ;
}
```

【課題】

☆ exercise-04-2 ビット演算子と + のみを用いて int 型の値の符号を反転させるプログラムを書きなさい.

★ exercise-05-1 西暦 1900 年から西暦 2004 年までの閏年をすべて出力するプログラムを書きなさい.

★ exercise-05-2 ２つの int 型の整数の和を計算するプログラムを書きなさい.

★ exercise-05-3 ２つの int 型の整数の差を計算するプログラムを書きなさい.

★ exercise-05-4 ２つの unsigned int 型の整数の積を計算するプログラムを書きなさい.

★ exercise-05-5 **（やや難）** ２つの unsigned int 型の整数の商と剰余を計算するプログラムを書きなさい.

★ exercise-05-6 **（難）** unsigned int 型の整数の平方根の整数部分を計算するプログラムを書きなさい.

★ exercise-05-7 **（やや難）** 次の式変形を利用して，２つの int 型の整数の積を計算するプログラムを書きなさい.

$$-a_{n-1}\times 2^{n-1}+\sum_{k=0}^{n-2}a_k\times 2^k=-a_0\times 2^0+\sum_{k=0}^{n-2}(a_k-a_{k+1})\times 2^{k+1}.$$

exercise-05-2 から exercise-05-7 のプログラムを書く際に, 必要なら次の事実を用いてもよい.

- int, unsigned int のビット幅が 32 ビットである.
- 整数は２進表現であり, 負の数は２の補数によって表現されている.

● 前回の講義のキーポイント及び補足

- char 型とは「１文字」を格納する変数の型であり, 「整数型」の一部である. 「文字」の「数値」とはその文字の「**文字コード**」の値であり, 多くのＯＳでは文字コードとして「**ASCII コード**」を用いている.
- Ｃ言語における「**１バイト**」とは char 型変数の占めるメモリ領域のことを指す.
- char 型が「符号なし」か「符号付き」かは処理系に依存する.

● 前回の課題の解説

☆ exercise-03-4 先週のプログラム ex02-4.c の出力がなぜそうなるかを, 「int 型の値の演算結果」, 「演算の結合規則」というキーワードを用いて正しく説明しなさい.

【解答】 乗法演算子 / は被演算数がともに int 型の場合にはその商を返す. また, 乗法演算子 / および * は左結合であるので, 「3/2*2」 は先に「3/2」の評価が行われ, 式の値は 1 となる. その後に「1*2」の評価が行われるため, 式「3/2*2」の値は 1 となる. 一方, 「3*2/2」 は先に「3*2」の評価が行われ, 式の値は 6 となる. その後に「6/2」の評価が行われるため, 式「3*2/2」の値は 2 となる.

★ exercise-04-3 ex04-2.c の中の「やってはいけないこと」となっている部分を, なぜ「やってはいけないのか」の理由を述べなさい.

【解答】 シフト演算子 << および >> は, 「右被演算数が負の場合には結果は 不定 となる」とされている. また右シフト演算子に関しては, 「左被演算数が負の値を持つ場合には結果は 処理系に依存する」とされている. (cf. K&R A7.8)
したがって「やってはいけない」理由は, その演算結果が**不定**または**処理系依存**となり, プログラムの移植性を保証しないからである.

【注意】 このプログラムをコンパイルするとき, 「warning: left shift count is negative」という警告が発生していたはずである. つまり, シフト演算子の右被演算数が負の定数の場合には警告が発生する. しかし, 上記の「やってはいけないこと」の対象が変数の場合にはこのような警告は発生しない.

★ exercise-04-4 ex04-3.c の中の「エラーになる」となっている部分を, なぜ「やってはいけないのか」の理由を述べなさい.

【解答】 インクリメント演算子 ++ および -- の規定では, 「被演算数は左辺値でなければならない」, 「結果は左辺値ではない」とされている. したがって「(i++)++」に関しては, 左辺値ではない「i++」をインクリメントしようとしているためにエラーとなる. (cf. K&R A7.4.1)

また, 「i+++++j」 に関しては, 一つの正しい解釈として 「i ++ + ++ j」が存在しうるが, Ｃ言語処理系は「左からの最大一致規則」を適用するため, 「i ++ ++ + j」と解釈する.[1] したがってインクリメント演算子の制約に違反するためエラーとなる.

さらに, 「i+--+-+-+-+-j」は本来は「エラー」と判定される可能性の高いものである. この式の唯一の解釈は「(i)+ -- (+-+-+-+-j)」であり, 「+-+-+-+-j」は既に左辺値ではあり得ない. したがって, 本来ならば「エラー」と判定される.[2] しかし, gcc では「+-+-+-+-j」を「----j」と書き換え, さらに - が偶数個であるので「j」と書き換えてしまうため「エラー」とはならない.[3]

【注意】 もし, プログラム中に「i+++++j」ではなく「i++␣+␣++j」と記述すれば, 意図通りに「(i++)+(++j)」として解釈される.

【注意】 このようなＣ言語の「非常に難しい部分」を完全に理解することは非常に困難である. ここでこのようなことを紹介した理由は「**明らかに正しいとわかっていることしか利用してはいけない**」ことを理解してほしいからである.

● 第４回の講義資料の訂正

下線があるところが誤りです. （この資料に書かれていることが正しい）

演算子の優先順位 (cf. K&R p. 65)

演算子	結合規則
() [] -> .	⇒
! ~ ++ -- +（単項） -（単項） & (type) sizeof	⇐
* / %	⇒
+ -	⇒
>> <<	⇒
> < <= >=	⇒
== !=	⇒
&	⇒
^	⇒
|	⇒
&&	⇒
||	⇒
?:	⇐
= += -= *= /= %= &= |= ^= <<= >>=	⇐
,	⇒

[1] Ｃ言語の規格書（日本語版） JIS X 3010, 「6.1. 字句要素」によれば, 「入力ストリームをある文字まで前処理字句に解析し終わったとき, 次の前処理字句は, 前処理字句を構成することのできる最も長い文字のならびとする.」とされている. これによって, 処理系の字句解析規則が「左からの最大一致規則」を利用することがわかる.
[2] 実際, gcc とは異なる処理系ではこれをエラーと判定するものがある.
[3] これを「i+--+-+-+-+-+-j」とすると gcc でもエラーが発生する.