情報処理概論 中間試験

問１． 本講義の内容に関して，次の文章の空欄 (a) 〜 (w) を埋めよ。

- ストアードプログラム型計算機は，開発に携わった数学者の名をとって，(a) とも呼ばれる。
- 10 進数の 125 を 2 進数に直すと (b) であり，16 進数に直すと (c) である。
- CPU とメモリーの情報のやりとりで，アドレスは (d)，読み込み／書き込みは (e)，データは (f) である (上記 3 つの空欄には,「片方向」あるいは「双方向」が入る)。
- 高級言語で掛かれたプログラムを低級言語に変換するプログラムを，(g) と呼ぶ。
- 識別子は，(h)，クラス，メソッドを区別するための名前である。識別子に使うことができる文字は，アルファベット，数字，記号の_と$とであり，文字数に制限はないが，(i) から始まってはいけない。また，あらかじめ定められているキーワードと (j)，(k)，(l) を使ってはいけない。キーワードの例として，(m) を表す break，(n) を表す double がある。
- int 型の変数は，(o) から (p) までの整数を表すことができる (2^n という表現を使ってよい)。int 型の変数に，データとして FFFFFFE4H が格納されている場合，その変数が表す数は，10 進数では (q) となる。16 進数 A4BDH を 2 進数に直すと (r) となる。この 16 進数を 16 進数のまま long 型のリテラルで表すと (s) となる。
- float 型の内部データの 16 進表現が C0F00000H の場合，これを通常の数の表現で表すと (t) となる。
- l，m，n を整数型変数として，

```
l = 5;
m = --l;
n = l--;
```

を実行したあとの l，m，n の値は，それぞれ，(u) と (v) と (w) である。

問２． 次のプログラム (クラス Chukan2) を実行したときに出力されるものを記せ。

```
public class Chukan2 {
  public static void main(String[] args) {
    int a = 2, b = 4, c = 4;
    if (a < 3) {
      if (b < 4) {
        System.out.println("('-'*)");
      } else if (b < 6) {
        System.out.println("(^o^)/");
      } else {
        System.out.println("( ••; )");
      }
    } else {
      if (b >= 0) {
        System.out.println("<(_ _*)> ");
      }
    }
    switch(c) {
    case 1: System.out.println("c は 1 かな？"); break;
    case 2: System.out.println("c は 2 かな？");
    case 3: System.out.println("c は 3 かな？"); break;
    case 4: System.out.println("c は 4 かな？");
    default: System.out.println("わかんないや？");
    }
  }
}
```

問３. 次の式の演算子が評価される順序を，各式の中の全ての演算子の下に 1 から順番に番号付して記せ。

```
a = b + c;
a = b = c + d;
a = b + c - d + e;
a = b + c / d * e;
a = b + c + d * e;
a = b * ++ c + - -- d;
A = a >= b | c <= d & e >= f;
```

問４. 次のプログラムは変数 target の素因数分解を行い，表示するプログラムである。空欄 (a)〜(d) を埋めよ。

```
 1: public class Chukan4 {
 2:   public static void main(String[] args) {
 3:     int target = 234;
 4:     int prime  = 2;
 5:     boolean firstDividePrime = true; // target を最初に割り切った素数かどうか
 6:    System.out.printf("%d =", target);
 7:    while(target > 1) {
 8:      int nDivided = 0;
 9:      for (int divider = 1 ; divider <= prime ; ++divider) {
10:        if (prime % divider == 0) ____(a)____
11:      }
12:      if (nDivided == 2) { // prime が素数の場合
13:        ____(b)____ // beki は，prime が target を割り切る最大の回数
14:        // target が prime で割ることのできる回数を調べる。
15:        while (target % prime == 0) {
16:          target /= prime;
17:          ____(c)____
18:        }
19:        // target は prime で beki 回割ることができた。
20:        if (beki != 0) {
21:          if (firstDividePrime) {
22:            System.out.printf(" %d^%d", prime, beki);
23:            ____(d)____
24:          } else {
25:            System.out.printf(" * %d^%d", prime, beki);
26:          }
27:        }
28:      }
29:      ++prime;
30:    }
31:     System.out.println(); // 最後に改行を表示
32:   }
33: }
```

(この出力は，234 = 2^1 * 3^2 * 13^1 となる。)

さらに，Chukan4 と同様の計算を行い，より高速化できるようにプログラムを改造したい。その改造点と高速になる理由を説明せよ (行番号を使って良い，より速くなると思われるものほど高得点)。

解答用紙

学科・類：　　学籍番号：　　名前：

問1.

(a)

(b)

(c)

(d)

(e)

(f)

(g)

(h)

(i)

(j)

(k)

(l)

(m)

(n)

(o)

(p)

(q)

(r)

(s)

(t)

(u)

(v)

(w)

問2.

問3.

```
a = b + c;
```

```
a = b = c + d;
```

```
a = b + c - d + e;
```

```
a = b + c / d * e;
```

```
a = b + c + d * e;
```

```
a = b * ++ c + - -- d;
```

```
A = a >= b | c <= d & e >= f;
```

問４.

(a)

(b)

(c)

(d)

高速化法とその説明

解答用紙

学科・類：　　学籍番号：　　名前：

問1.

(a) ノイマン型計算機

(b) 1111101

(c) 7DH (H は付いていなくても良い)

(d) 片方向

(e) 片方向

(f) 双方向

(g) コンパイラ

(h) 変数

(i) 数字

(j) `true`　　((j),(k),(l) は順不同)

(k) `false`

(l) `null`

(m) 実行する文を，ループ文や switch 文 の中から外に移すこと

(n) 倍精度 (64 bit) 浮動小数点数

(o) -2^{31}

(p) $2^{31}-1$

(q) -28

(r) 1010 0100 1011 1101

(s) 0xa4bdL

(t) -7.5

(u) 3

(v) 4

(w) 4

問2.

```
(^o^)/
c は 4 かな？
わかんないや？
```

問3.

```
a = b + c;
  2   1

a = b = c + d;
  3   2   1

a = b + c - d + e;
  4   1   2   3

a = b + c / d * e;
  4   3   1   2

a = b + c + d * e;
  4   1   3   2

a = b * ++ c + - -- d;
  6   2  1   5 4  3

A = a >= b | c <= d & e >= f;
  6   1    5    2   4    3
```

問4.

(a) `++nDivided;`

(b) `int beki = 0;`

(c) `++beki;`

(d) `firstDividePrime = false;`

高速化法とその説明

8 行目から 12 行目までを

```
int divider = 2;
int upperBound = (int) (Math.sqrt(prime + 0.1));
for (divider = 3 ; divider <= upperBound ; divider += 2) {
  if (prime% divider == 0) break;
}
if (divider > upperBound) { // prime が素数の場合
```

と変え，29 行目を

```
if (prime == 2) prime = 3;
else prime += 2;
```

と変える。

理由は，次の通りである。29 行目の変更により，素数の候補 prime について，2 と奇数の場合だけを調べるようにしている。整数 x を，1 と x 以外に割り切る数があるとすれば，$\sqrt{x}$ までに存在する。$x \geq 2$ のとき，$\sqrt{x} < x$ であるから，prime を 3 以上の奇数とすれば，3 から prime の平方根までの奇数に割りきれる数がなければ，素数であることが分かる。そのため，その中に割りきれる数がなかったときは，break 文を使って for から抜け出る。prime が素数であるときは，for 文を break 文で出てこなかったときであるので，divider か prime の平方根より大きくなっている。この条件が成立するとき，prime を素数として，素因数分解の因数として後の処理に利用する。

従って，偶数を調べないため，prime を割り切る数が 1 つでも存在すれば，それ以上調べないため，高速化される。

なお，平方根の計算で 0.1 を加えている理由は，prime が平方数であるときに，型変換などによって，本当の平方根よりも -1 だけ小さい数が出力されることが考えられるからである (0.1 を加えたことによって $+1$ 大きい数が出力されることはない)。

別解 8 行目から 12 行目までと，28 行目を取り去り，29 行目を

```
if (prime == 2) prime = 3;
else prime += 2;
```

と変える。

理由は次の通りである。最初のプログラムでは prime が素数かどうかチェックしていた。一般に，x，y を 2 以上の整数とし，x が y 未満の素数で割りきることができず y で割りきれるならば，y は素数である。target は既にそれ以下の素数で割って残っているものであり，prime 未満の素数では割りきれない。従って，prime に対する beki が 0 でないとすると，prime は素数であるので，プログラムの prime は素数が素数かどうか判定する部分を省略することができる。

従って，素数かどうか調べないため高速化される。