第56回

C言語プログラミング能力認定試験

２　級

解答時における注意事項

1.次の表に従って解答してください。

問題番号	問１～問８
選択方法	８問必須
試験時間	９０分

2.HB の黒鉛筆を使用してください。訂正する場合は，あとが残らないように消しゴムできれいに消し，消しくずを残さないでください。なお，ボールペンや万年筆等で記入した場合は，採点されません。

3.マークシート（解答用紙）の所定の欄に，級種，会場コード，受験番号を記入しマークしてください。また，会場名，氏名及びフリガナを所定の位置に記入してください。

4.解答は，次の例題にならって，「解答マーク欄」にマークしてください。

〔例題〕　日本の首都はどこか。

ア　東京　　イ　京都　　ウ　大阪　　エ　福岡

正しい答えは“ア　東京”ですから，次のようにマークしてください。

例題 

指示があるまで開いてはいけません。 **試験終了後，問題冊子を回収します。**

受験会場	
受験番号	
氏　　名	

サーティファイ
情報処理能力認定委員会

問１～問８は，すべて必須問題です。全問について解答してください。
各設問の答えは，解答群の中から一つだけ選び，括弧中の設問番号に対応したマークシートの解答番号の「解答マーク欄」にマークしてください。なお，二つ以上マークした場合には不正解になります。

問１　Ｃ言語の特徴に関する次の記述の正誤を，解答群の中から選べ。

(1)　標準ライブラリ関数 rand は，ヘッダ string.h で宣言されている。

(2)　列挙型指定子 enum により定義される列挙定数において，同じ列挙体の中で同じ値をもつ複数の列挙定数を定義することができる。

(3)　math.h で定義されている標準関数 pow は，pow(x, y)の呼出しにより，x の y 乗を計算する。

(4)　s1 の文字列が s2 の文字列より大きい場合，「strcmp(s1, s2)」の返却値は 0 より小さい整数となる。

(5)　標準ライブラリ関数 strrchr は，第 1 引数で指定した文字列の中で，第 2 引数で指定した文字が最後に現れる位置を返す。

(6)　標準ライブラリ関数 sprintf は，ヘッダ stdio.h で宣言されている。

(7)　論理 OR 演算子（||）を使用して複数の条件式を記述した場合，真偽の判定は左から右へ順に行われ，条件式が真と判定されても，それ以降の全ての条件式が判定される。

(8)　ヘッダファイルの指定には前処理指令である#include を使用するが，ヘッダファイルの中に，さらに#include を使用することはできない。

解答群

ア　正しい　　イ　誤り

問２　sizeof 演算子及び文字列関数に関する次の記述中の［　　］に入れる適切な字句を，解答群の中から選べ。

sizeof 演算子は，演算対象として指定した［　(9)　］や変数，配列などのメモリ上のサイズをバイト数で求める。ここで，書式は，

```
sizeof 単項式
sizeof (［　(9)　］)
```

である。また，単項式には，変数，配列，構造体，共用体の名前などを記述することができる。

int 型のサイズが 32 ビット，1 バイトあたりのビット数が 8 である仕様の処理系の場合に，次のプログラムを実行すると，

```
a =［　(10)　］
b =［　(11)　］
c =［　(12)　］
d =［　(13)　］
e =［　(14)　］
```

と標準出力に出力される。

＜プログラム＞

```
#include <stdio.h>
#include <string.h>

int main(void)
{
    int data[] = {-3, 8, -9, 5};
    char mountain[] = "Fuji";

    printf("a = %d¥n", sizeof(int));
    printf("b = %d¥n", sizeof data[2]);
    printf("c = %d¥n", sizeof data);
    printf("d = %d¥n", strlen(mountain));
    printf("e = %d¥n", sizeof mountain / sizeof(char));

    return 0;
}
```

(9) の解答群

ア　データ型　　イ　演算子　　ウ　配列要素数　　エ　関数　　オ　予約語

(10) の解答群

ア　2　　イ　4　　ウ　8　　エ　16　　オ　32

(11) の解答群

ア　1　　イ　2　　ウ　3　　エ　4　　オ　5

(12) の解答群

ア　2　　イ　4　　ウ　8　　エ　16　　オ　32

(13) の解答群

ア　2　　イ　3　　ウ　4　　エ　5　　オ　6

(14) の解答群

ア　2　　イ　3　　ウ　4　　エ　5　　オ　6

問３　次のプログラムを実行したとき，(15)～(19)で出力される値を，解答群の中から選べ。なお，short 型は 2 バイト長であるものとする。

＜プログラム＞

```
#include <stdio.h>

int main(void)
{
    unsigned short a = 0x5afd;
    unsigned short b = 0xe439;

    printf("%04hx¥n", a & b);                 … (15)
    printf("%04hx¥n", a | b);                 … (16)
    printf("%04hx¥n", b ^ ~b);                … (17)
    printf("%04hx¥n", a << 4);                … (18)
    printf("%04hx¥n", a ^ (b << 3));          … (19)

    return 0;
}
```

(15) の解答群

ア　3f36　　イ　4039　　ウ　76c4　　エ　bec4　　オ　fefd

(16) の解答群

ア　3f36　　イ　4039　　ウ　76c4　　エ　bec4　　オ　fefd

(17) の解答群

ア　0000　　イ　1bc6　　ウ　e439　　エ　ff00　　オ　ffff

(18) の解答群

ア　02d7　　イ　0b5f　　ウ　5fa0　　エ　afd0　　オ　d7e8

(19) の解答群

ア　00c8　　イ　196d　　ウ　7b35　　エ　7bfd　　オ　f620

問４　main 関数に関する次の記述中の［　　］に入れる適切な字句を，解答群の中から選べ。

main 関数は，二つの仮引数をもつ場合，次のように定義できる。

```
int main(int argc, [ (20) ] *argv[])
{
     ⋮
    return 0;
}
```

仮引数 argc には，実行時にコマンド行で指定された［ (21) ］が格納されて渡される。また，配列 argv の要素には，プログラム名と各引数の文字列のポインタが格納されて渡される。ここで，argv[argc]には［ (22) ］が格納されている。

コマンド行で

```
job△105△AZ△-opt△2021
```

（△は空白文字を表す）

と指定して，次のプログラム（プログラム名 job）を実行すると，

```
a = [ (23) ]
b = [ (24) ]
c = [ (25) ]
```

と標準出力に出力される。

＜プログラム＞

```
#include <stdio.h>

int main(int argc, [ (20) ] *argv[])
{
    printf("a = %d¥n", argc);
    printf("b = %s¥n", argv[2]);
    printf("c = %c¥n", argv[4][1]);

    return 0;
}
```

(20) の解答群

ア　char　　イ　double　　ウ　float　　エ　int　　オ　void

(21) の解答群

ア　プログラム名と引数の個数の合計値
イ　プログラム名の文字数
ウ　第1引数の値
エ　引数の個数
オ　引数の個数 － 1

(22) の解答群

ア　EOF　　イ　FILE　　ウ　NULL　　エ　"¥n"　　オ　"¥t"

(23) の解答群

ア　3　　イ　4　　ウ　5　　エ　105　　オ　2021

(24) の解答群

ア　-opt　　イ　105　　ウ　AZ　　エ　job　　オ　opt

(25) の解答群

ア　-　　イ　0　　ウ　1　　エ　2　　オ　o

問５　ファイルアクセスに関する次の記述中の［　　］に入れる適切な字句を解答群の中から選べ。

プログラムでファイルを使用する際には，［（26）］関数を呼び出し，ファイル名で指定したファイルにストリームを結びつける。［（26）］関数の呼び出しに失敗した場合の戻り値は，［（27）］となる。ストリームから1文字読み込むには，［（28）］関数を呼び出す。ストリームがファイルの終わりに達している場合，［（28）］関数は［（29）］を返却する。

（26）の解答群

ア　fclose　　イ　fgetc　　ウ　fgets　　エ　fopen　　オ　fscanf

（27）の解答群

ア　EOF　　イ　FILE　　ウ　NULL　　エ　stderr　　オ　不定値

（28）の解答群

ア　fgetc　　イ　fgets　　ウ　fputc　　エ　fputs　　オ　fscanf

（29）の解答群

ア　EOF　　イ　NULL　　ウ　'¥0'　　エ　'¥n'　　オ　不定値

問６　次のプログラムを実行したとき，(30)～(34)で出力される値を，解答群の中から選べ。

＜プログラム＞

```
#include <stdio.h>

int main(void)
{
    char *numbers[] = {
        "One", "Two", "Three", "Four", "Five",
        "Six", "Seven", "Eight", "Nine", "Ten" };
    char *p, **pp;
    int i;

    p = numbers[3];
    printf("%s¥n", p);                          ･･･ (30)
    printf("%s¥n", p + 2);                      ･･･ (31)

    pp = numbers;
    printf("%s¥n", *(pp + 7));                  ･･･ (32)
    printf("%s¥n", *pp + 1);                    ･･･ (33)

    for (i = 0; i < 9; i++) {
        if (**pp == **(pp + 1)) {
            break;
        }
        pp++;
    }
    printf("%s¥n", *pp);                        ･･･ (34)

    return 0;
}
```

(30) の解答群

ア Five　イ Four　ウ One　エ Three　オ Two

(31) の解答群

ア Fo　イ ree　ウ Seven　エ Six　オ ur

(32) の解答群

ア Eight　イ Nine　ウ Seven　エ Six　オ Ten

(33) の解答群

ア e　イ ne　ウ One　エ Two　オ wo

(34) の解答群

ア Five　イ Four　ウ Six　エ Three　オ Two

問７　次のプログラムの説明を読んで，プログラム中の［　　］に入れる適切な字句を，解答群の中から選べ。

＜プログラムの説明＞

関数 RadixConv は，与えられた文字列を基数変換した文字列として返す。指定できる基数は 2 ～ 16 とする。

例えば，10 進数の文字列"123"を，関数 RadixConv を用いて 8 進数の文字列に変換すると"173"，6 進数の文字列"51042"を関数 RadixConv を用いて 12 進数の文字列に変換すると"3A82"となる。0 ～ 15 の数値を各基数で表すときに用いる数字文字を図１に示す。

基数＼数値	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
2	0	1														
3	0	1	2													
4	0	1	2	3												
5	0	1	2	3	4											
6	0	1	2	3	4	5										
7	0	1	2	3	4	5	6									
8	0	1	2	3	4	5	6	7								
9	0	1	2	3	4	5	6	7	8							
10	0	1	2	3	4	5	6	7	8	9						
11	0	1	2	3	4	5	6	7	8	9	A					
12	0	1	2	3	4	5	6	7	8	9	A	B				
13	0	1	2	3	4	5	6	7	8	9	A	B	C			
14	0	1	2	3	4	5	6	7	8	9	A	B	C	D		
15	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	
16	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

図１　2～16 進数で用いる文字

(1)　関数RadixConvは，引数で渡された配列snumに格納された文字列（以後，変換前文字列と呼ぶ）をsradixで指定された基数（以後，変換前基数と呼ぶ）の整数値とみなし，同じく引数で指定されたdradixの基数（以後，変換後基数と呼ぶ）の整数値に変換し，配列dnumに文字列（以後，変換後文字列と呼ぶ）として格納する。ここで，sradixとdradixは2 ～ 16の値が指定されるものとする。関数RadixConvは，変換前文字列に，変換前基数として不当な文字が含まれていた場合は-1を返却し，正しく変換できた場合は0を返却する。ここで，変換後文字列を格納するための領域は十分に確保されており，基数変換過程において，int型変数で扱える値の範囲を超えることはないものとする。

関数RadixConvの引数と返却値の仕様を表１に示す。また，関数RadixConvから呼び出される関数ToNumberと関数ToCharの引数と返却値の仕様を，それぞれ表２，表３に示す。

表１　関数 RadixConv の引数と返却値の仕様

int RadixConv(char snum[], int sradix, char dnum[], int dradix)	
引　数	snum　　：変換前文字列 sradix　：変換前基数（2 ～ 16） dnum　　：変換後文字列 dradix　：変換後基数（2 ～ 16）
返却値	0：正常，-1：不当な文字あり

表２　関数 ToNumber の引数と返却値の仕様

int ToNumber(char ch)	
引　数	ch　　：数値化したい数字文字（'0' ～ 'F'）
返却値	数値（0 ～ 15）。ただし変換できない場合は-1

表３　関数 ToChar の引数と返却値の仕様

char ToChar(int n)	
引　数	n　　：数字文字化したい数値（0 ～ 15）
返却値	数字文字（'0' ～ 'F'）

(2)　以下に，関数RadixConv関数の処理手順を示す。

①　基数重みを 1 に初期化する。
②　変換前文字列の末尾文字の要素位置を求める。
③　変換値を 0 に初期化する。
④　変換前文字列の末尾文字から先頭文字まで，各文字を参照し，④－１～④－３の処理を繰り返す。
　④－１　関数 ToNumber を呼び出し対象文字を数値化する。対象文字が数値化できなかった場合は，-1 を返却し関数を終了する。
　④－２　変換値に，数値化した値×基数重みを加算する。
　④－３　基数重みに変換前基数を乗じた値を，新たな基数重みとする。
⑤　変換後文字列の文字格納要素位置（以後，変換後文字位置と呼ぶ）を 0 に初期化する。
⑥　変換値が変換後基数以上の間，⑥－１～⑥－２の処理を繰り返す。
　⑥－１　変換値を変換後基数で割った余りを引数として関数 ToChar を呼び出し，取得した数字文字を変換後文字列の変換後文字位置に格納し，変換後文字位置を次の文字位置に進める。
　⑥－２　変換値を変換後基数で割った商を新たな変換値とする。
⑦　変換値を引数として関数 ToChar を呼び出し，取得した数字文字を，変換後文字列の変換後文字位置に格納する。
⑧　変換後文字列の（変換後文字位置＋1）の位置に'¥0'を格納する。
⑨　変換後文字列を，1 文字目と末尾の文字，2 文字目と末尾－1 の文字，…というように前後を順に入れ替える。
⑩　0 を返却して関数を終了する。

＜プログラム＞

```
#include <stdio.h>
#include <string.h>

char nums[] = "0123456789ABCDEF";

int ToNumber(char ch)
{
    int i;

    for (i = 15; i >= 0; i--) {
        if (nums[i] == ch) {
            break;
        }
    }

    return i;
}

char ToChar(int n)
{
    return nums[n];
}

int RadixConv(char snum[], int sradix, char dnum[], int dradix)
{
    int num, d, wt, slen, si, di, i;
    char wk;

    wt = 1;
    slen = strlen(snum);
    si = slen - 1;
    num = 0;
    while ( (35) ) {
        d = ToNumber(snum[si--]);
        if ( (36) ) {
            return -1;
        }
        num += d * wt;
        (37);
    }

    di = 0;
    while ( (38) ) {
        dnum[di++] = ToChar(num % dradix);
        (39);
    }
```

```
    dnum[di] = ToChar(num);
    dnum[di + 1] = '¥0';
    for (i = 0; i < di; i++, di--) {
        wk = dnum[di];
        dnum[di] = dnum[i];
        dnum[i] = wk;
    }

    return 0;
}
```

(35) の解答群

ア　si > 0
イ　si > sradix
ウ　si >= 0
エ　si >= sradix

(36) の解答群

ア　d == -1
イ　d == -1 || d > sradix
ウ　d == -1 || d >= sradix
エ　d >= sradix

(37) の解答群

ア　wt *= dradix
イ　wt *= sradix
ウ　wt /= dradix
エ　wt /= sradix

(38) の解答群

ア　num > 0
イ　num > dradix
ウ　num >= 0
エ　num >= dradix

(39) の解答群

ア　num += dradix
イ　num -= dradix
ウ　num /= dradix
エ　num %= dradix

問８　次のプログラムの説明を読んで，プログラム中の［　　］に入れる適切な字句を，解答群の中から選べ。

＜プログラムの説明＞

関数ParseRecordsは，ヘッダレコード，データレコード，トレーラレコード，エンドレコードの四つのレコードで構成される入力レコード群について，レコード構成の妥当性チェックをすると共に，IDとそのIDに属するデータレコードの数の情報を返却するプログラムである。

入力レコードは長さ1～79の文字列であり，図１で示すように，各レコードの先頭にレコード種別（ヘッダレコード，データレコード，トレーラレコード，エンドレコード）を識別する文字（以後，識別コードと呼ぶ）が，2文字目以降にはそのレコード固有の情報を保持する文字列（以後，固有情報と呼ぶ）が格納されている。固有情報の末尾には'¥0'が付加される。

識別コード	固有情報
1	2 ～ 79

図１　入力レコードの形式

識別コードにより認識される四つのレコード種別の内容を図２に示す。

レコード種別	識別コード	固有情報
ヘッダ	'H'	ID（文字列）
データ	'D'	データ情報（文字列）
トレーラ	'T'	データレコード数（文字列）
エンド	'E'	（なし）

図２　四つのレコード種別

入力レコード群は，「ヘッダレコード×1＋データレコード×N（N≧0）＋トレーラレコード×1」の１個以上の連続＋エンドレコード×1という構成になっていて，エンドレコードの次にはレコードは存在しない。図３に入力レコード群の構成例を識別コードで示す。

H	D	D	D	T	H	D	D	D	D	T	…	H	D	D	T	E

図３　入力レコード群の構成例

最小のレコード群は，データレコードが0個の場合で，図４のように，ヘッダレコード＋トレーラレコード＋エンドレコードの3レコードの構成例となる。

H	T	E

図４　最小のレコード構成例

＜レコード構成の妥当性チェック＞

レコード構成の妥当性チェックは，表１の状態遷移表に従って行う。ここで，状態の初期値は「①H 待ち」であり，結果が「error」となった場合，レコード構成に誤りがあると判定する。

表１　状態遷移表

		対象レコードの識別コード				
		H	D	T	E	その他
状態	①H 待ち	②へ	error	error	error	error
	②D or T 待ち	error	状態維持	③へ	error	error
	③H or E 待ち	②へ	error	error	④へ	error
	④E 後	error	error	error	error	error

状態＝「①H 待ち」のとき，
　対象レコードの識別コードがHであれば，状態を「②D or T 待ち」へ遷移する。
　対象レコードの識別コードが上記以外であれば，errorとなる。

状態＝「②D or T 待ち」のとき，
　対象レコードの識別コードがDであれば，状態を「②D or T 待ち」のまま維持する。
　対象レコードの識別コードがTであれば，状態を「③H or E 待ち」へ遷移する。
　対象レコードの識別コードが上記以外であれば，errorとなる。

状態＝「③H or E 待ち」のとき，
　対象レコードの識別コードがHであれば，状態を「②D or T 待ち」へ遷移する。
　対象レコードの識別コードがEであれば，状態を「④E 後」へ遷移する。
　対象レコードの識別コードが上記以外であれば，errorとなる。

状態＝「④E 後」のとき，
　対象レコードの識別コードがいずれであっても，errorとなる。

また，最終レコードまで走査し終えた後の状態が「④E 後」でなかった場合は，errorとする。
さらに，ヘッダレコードとトレーラレコード間のデータレコード数をカウントし，トレーラレコードの固有情報であるデータレコード数と一致しているかを判定し，不一致であった場合もerrorとする。

＜関数ParseRecordsの関数仕様＞

関数ParseRecordsは，引数nで渡されたレコード数の入力レコード群（配列records）を先頭から走査し，ヘッダレコードの固有情報であるIDとヘッダレコードからトレーラレコードの間にあるデータレコードの数を，引数で渡された構造体INFOの配列infosの要素に格納して返す。このとき，infosに格納した要素数を返却値として返す。ただし，レコード構成の妥当性チェックでerrorとなった場合は，負の値を返却値として返す。関数ParseRecordsの引数と返却値の仕様を表２に示す。

ここで，REC_SIZEは1レコード（最大79文字）を格納するための領域サイズ80（文字列末尾の'¥0'を含めたサイズ）として定義されている。また，構造体INFOは，以下のように定義されている。

```
struct INFO {
    char id[REC_SIZE - 1];      /* ヘッダレコード ID */
    int cnt;                    /* データレコード数 */
};
```

表２　関数 ParseRecords の引数と返却値の仕様

int ParseRecords(char records[][REC_SIZE], int n, struct INFO infos[])	
引　数	records　：入力レコード群 n　　：入力レコード群の要素数 infos　：ID とそのデータレコード数を格納する配列 （以後，情報配列と呼ぶ）
返却値	0～　　：infos の要素数 -1　　：レコード構成異常 -2　　：データレコード数異常

＜関数ParseRecordsの処理の流れ＞

(1) 状態を「①H 待ち」に設定する。

(2) 情報配列の編集対象要素位置（以後，要素位置と呼ぶ）を0に初期化する。

(3) 入力レコード群の先頭から末尾までの各レコードに対して，(case－1)～(case－4)の場合分け処理を繰り返す。

(case－1) 状態が「①H 待ち」の場合：
対象レコードがヘッダレコードであれば，ヘッダレコードの固有情報（ID）を情報配列の要素位置のメンバーidに複写し，メンバーcntを0に初期化後，状態を「②D or T 待ち」に設定する。

対象レコードがヘッダレコード以外であれば，「レコード構成異常」を返却値として関数を終了する。

(case－2) 状態が「②D or T 待ち」の場合：
対象レコードがデータレコードであれば，情報配列の要素位置のメンバーcntをインクリメントする。

対象レコードがトレーラレコードであれば，トレーラレコードの固有情報（データレコード数）を数値化した値と，情報配列の要素位置のメンバーcntの値が等しいか判定し，等しくなかった場合は，「データレコード数異常」を返却値として関数を終了する。等しかった場合は，要素位置をインクリメントし，状態を「③H or E 待ち」に設定する。

対象レコードがデータレコードでもトレーラレコードでもなければ,「レコード構成異常」を返却値として関数を終了する。

(case－3) 状態が「③H or E 待ち」の場合：
対象レコードがヘッダレコードであれば，ヘッダレコードの固有情報（ID）を情報配列の要素位置のメンバーidに複写し，メンバーcntを0に初期化後，状態を「②D or T 待ち」に設定する。

対象レコードがエンドレコードであれば，状態を「④E 後」に設定する。

対象レコードがヘッダレコードでもエンドレコードでもなければ，「レコード構成異常」を返却値として関数を終了する。

(case－4) 状態が「④E 後」の場合：
「レコード構成異常」を返却値として関数を終了する。

(4) 状態が「④E 後」でなければ，「レコード構成異常」を返却値として関数を終了する。

(5) 要素位置を返却値として関数を終了する。

＜プログラム＞

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#define REC_SIZE   80

enum STATE {
    W_HEADER,           /* H 待ち */
    W_DATA_OR_TRAILER,  /* D or T 待ち */
    W_HEADER_OR_END,    /* H or E 待ち */
    W_AFTER_END         /* E 後 */
};

#define C_HEADER   'H'
#define C_DATA     'D'
#define C_TRAILER  'T'
#define C_END      'E'

struct INFO {
    char id[REC_SIZE - 1];     /* ヘッダレコード ID */
    int cnt;                   /* データレコード数 */
};

int ParseRecords(char records[][REC_SIZE], int n, struct INFO infos[])
{
    int i, j;
    enum STATE state;

    state = [ (40) ];
    i = 0;
    for (j = 0; j < n; j++) {
        switch (state) {
            case W_HEADER:
                if (records[j][0] == C_HEADER) {
                    strcpy(infos[i].id, &records[j][1]);
                    [ (41) ];
                    state = W_DATA_OR_TRAILER;
                } else {
                    return -1;
                }
                break;
```

```
            case W_DATA_OR_TRAILER:
                if ( (42) ) {
                    infos[i].cnt++;
                } else if (records[j][0] == C_TRAILER) {
                    if ( (43) ) {
                        return -2;
                    }
                    i++;
                    state = W_HEADER_OR_END;
                } else {
                    return -1;
                }
                break;
            case W_HEADER_OR_END:
                if (records[j][0] == C_HEADER) {
                    strcpy(infos[i].id, &records[j][1]);
                    (41);
                    state = W_DATA_OR_TRAILER;
                } else if (records[j][0] == C_END) {
                    state = W_AFTER_END;
                } else {
                    return -1;
                }
                break;
            case W_AFTER_END:
                return -1;
        }
    }
    if ( (44) ) {
        return -1;
    }

    return i;
}
```

(40) の解答群

ア　W_AFTER_END
イ　W_DATA_OR_TRAILER
ウ　W_HEADER
エ　W_HEADER_OR_END

(41) の解答群

ア　infos[i].cnt = 0
イ　infos[i].cnt = 1
ウ　infos[j].cnt = 0
エ　infos[j].cnt = 1

(42) の解答群

ア　records[j][0] == C_DATA
イ　records[j][0] == C_HEADER
ウ　records[j][0] != C_DATA
エ　records[j][0] != C_HEADER

(43) の解答群

ア　atoi(&records[j][1]) == infos[i].cnt
イ　atoi(&records[j][1]) == infos[i].cnt - 1
ウ　atoi(&records[j][1]) != infos[i].cnt
エ　atoi(&records[j][1]) != infos[i].cnt - 1

(44) の解答群

ア　state == W_AFTER_END
イ　state == W_HEADER_OR_END
ウ　state != W_AFTER_END
エ　state != W_HEADER_OR_END