サンプル問題

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。

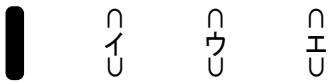
３.マークシート（解答用紙）の所定の欄に，級種，会場コード，受験番号を記入しマークしてください。また，会場名，氏名及びフリガナ，性別を所定の位置に記入してください。

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	

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言語の文法に関する次の記述の正誤を，解答群の中から選べ。

(1)　関数の型の宣言は関数の定義において行うが，関数の型を宣言しない場合はその関数は int 型として扱われる。

(2)　#include 命令で指定したヘッダファイル中で，さらに#include 命令を記述して使用することはできない。

(3)　const 修飾子は，値の変更ができない変数を指定するものである。

(4)　atoi 関数を使用すると，浮動小数点数を int 型に変換できる。

(5)　#define 命令や#include 命令は，プリプロセッサにより解釈される。

(6)　変数名に予約語は使用できないが，予約語を含む名前は使用できる。

(7)　型名を void として定義された関数であっても，return 文を使用して関数の戻り値を返すことは可能である。

(8)　外部変数名と同じ名前の局所変数を宣言することはできない。

解答群

ア　正しい　　イ　誤り

問２　列挙型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。

列挙型は，データの取り得る値を列挙定数と呼ばれる名前で列挙するデータ型であり，次のように定義される。

［(9)］ 列挙タグ { 列挙定数のリスト } 列挙変数;

列挙定数のリストに記述した定数のデータ型は整数型となる。デフォルトでは，先頭の値が［(10)］となり，それ以降は，列挙順に［(11)］ずつ加算された値が設定される。

また，各列挙定数の値は“=”に続き，定数式を記述することにより明示的に値を設定することができる。

例えば，次のプログラムを実行すると，画面には，

A = ［(12)］，B = ［(13)］，C = ［(14)］

と出力される。

＜プログラム＞

```
#include <stdio.h>

int main(void)
{
    ［(9)］ code {
        HOKKAIDO, AOMORI, IWATE, SAITAMA = 11, CHIBA, TOKYO, OKINAWA = 47
    };

    printf("A = %d, B = %d, C = %d¥n", AOMORI, TOKYO, OKINAWA - CHIBA);
    return 0;
}
```

(9) の解答群

ア const　　イ #define　　ウ enum　　エ struct　　オ typedef

(10)，(11) の解答群

ア -1　　イ 0　　ウ 1　　エ 10　　オ 不定値

(12)，(13) の解答群

ア 0　　イ 1　　ウ 2　　エ 11　　オ 13

(14) の解答群

ア 12　　イ 34　　ウ 35　　エ 36　　オ 47

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，char型は１バイトの領域を確保するものとする。

＜プログラム＞

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

int main(void)
{
    char str1[8] = "static";
    char str2[] = "100¥t59¥t36";
    char str3[] = "int¥0char¥0";

    printf("%d¥n", strlen(str1));          … (15)
    printf("%d¥n", sizeof str1);           … (16)
    printf("%d¥n", strlen(str2));          … (17)
    printf("%d¥n", strlen(str3));          … (18)
    printf("%d¥n", sizeof str3);           … (19)

    return 0;
}
```

(15)，(16)の解答群

ア 6　　イ 7　　ウ 8　　エ 9　　オ 10

(17)の解答群

ア 8　　イ 9　　ウ 10　　エ 11　　オ 12

(18)の解答群

ア 3　　イ 4　　ウ 11　　エ 12　　オ 13

(19)の解答群

ア 6　　イ 7　　ウ 9　　エ 10　　オ 11

問４　ビット演算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。

Ｃ言語のビット演算子には，

［(20)］演算子 ･･･ ビットごとの論理積
［(21)］演算子 ･･･ ビットごとの論理和
［(22)］演算子 ･･･ ビットごとの排他的論理和
［(23)］演算子 ･･･ ビット反転

などがある。

なお，ビット反転した値においては，［(23)］演算子を用いず求めることができる。また，ビットごとの排他的論理和の値においては，［(22)］演算子を用いず求めることもできる。

例えば，次のプログラムを実行すると，変数 b と変数 c が同じ値に，変数 d と変数 e が同じ値になる。

＜プログラム＞

```
int main(void)
{
    unsigned short x = 0x1f57, y = 0xa023;
    unsigned short b, c, d, e;

    b = [ (23) ] x;
    c = x [ (24) ] (unsigned short)0xffff;

    d = x [ (22) ] y;
    e = (x & ~y) [ (25) ] (~x & y);

    return 0;
}
```

(20) ～ (23) の解答群

ア　&　　イ　?　　ウ　^　　エ　|　　オ　~

(24) の解答群

ア　&　　イ　<<　　ウ　>>　　エ　^　　オ　|

(25) の解答群

ア　&　　イ　*　　ウ　+　　エ　-　　オ　|

問５　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。なお，問題文中および解答群の␣は空白文字を表す。

コマンドラインから三つの引数（Tennis，Baseball，Soccer）を指定して，プログラム sample を実行する。ここで，コマンドラインから，

```
sample␣Tennis␣Baseball␣Soccer
```

とキー入力して実行したときの，argc と argv の出力結果を次に示す。

＜プログラム＞

```
#include <stdio.h>

int main ( [ (26) ] argc, [ (27) ] *argv[])
{
    printf("argc = %d¥n", argc);
    printf("argv = %s¥n", argv[2]);

    return 0;
}
```

＜出力結果＞

```
argc = [ (28) ]
argv = [ (29) ]
```

(26)，(27) の解答群

ア　char　　イ　double　　ウ　float　　エ　int　　オ　void

(28) の解答群

ア　0　　イ　1　　ウ　2　　エ　3　　オ　4

(29) の解答群

ア　Baseball　　イ　Baseball␣Soccer
ウ　sample　　エ　Soccer
オ　Tennis

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)
{
    int data[] = {9, -1, 7, 6, 3, -5, 6, 2, -4, 0};
    int *ip, *iq;
    int i, value, flag;

    ip = data;
    printf("%d¥n", *ip + 1);                          … (30)

    ip = data;
    printf("%d¥n", *(ip + 3));                        … (31)

    ip = data;
    iq = ip + 6;
    iq--;
    printf("%d¥n", *iq);                              … (32)

    ip = &data[1];
    iq = &data[6];
    value = 0;
    for (i = 0; i < 3; i++) {
        value += *ip + *iq;
        ip++;
        iq--;
    }
    printf("%d¥n", value);                            … (33)
```

```
    flag = 0;
    for (ip = data; *ip != 0 && flag == 0; ip++) {
        for (iq = ip + 1; *iq != 0; iq++) {
            if (*ip == *iq) {
                flag = 1;
                break;
            }
        }
    }
    printf("%d¥n", *iq);                                    … (34)

    return 0;
}
```

(30) の解答群

ア -1　　イ 0　　ウ 9　　エ 10　　オ 11

(31) の解答群

ア 3　　イ 6　　ウ 7　　エ 11　　オ 12

(32) の解答群

ア -5　　イ -4　　ウ 2　　エ 3　　オ 6

(33) の解答群

ア 7　　イ 16　　ウ 19　　エ 22　　オ 25

(34) の解答群

ア -1　　イ 0　　ウ 2　　エ 3　　オ 6

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＞

スタック操作とキュー（待ち行列）操作を行うための関数群である。スタックとキューで管理されるデータは int 型とする。

スタックとキューは，それぞれ次に示す構造体で管理される。なお，キューに格納されるデータはリングバッファ（循環バッファ）として管理され，配列 buffer の末尾と先頭が連続しているように扱われる。

```
#define STACK_SIZE  10
struct t_stack {                /* スタック管理構造体          */
    int result;                 /* 操作結果 0:正常, -1:異常    */
    int sp;                     /* スタックポインタ            */
    int buffer[STACK_SIZE];     /* スタック領域                */
};

#define QUEUE_SIZE  10
struct t_queue {                /* キュー管理構造体            */
    int result;                 /* 操作結果 0:正常, -1:異常    */
    int count;                  /* 格納されているデータ数      */
    int rp;                     /* 読込みポインタ              */
    int wp;                     /* 書込みポインタ              */
    int buffer[QUEUE_SIZE];     /* キュー領域                  */
};
```

スタック操作には，表1～表3の三つの関数を用いる。

表１　関数 stack_init の引数と戻り値の仕様

void stack_init(struct t_stack *p)	
引　数	p : スタック管理構造体のポインタ
戻り値	なし
機　能	引数で渡されたスタック管理構造体のメンバーを初期化する。 ・スタックポインタ sp に 0 を設定する。

表２　関数 push の引数と戻り値の仕様

void push(struct t_stack *p, int value)	
引　数	p ： スタック管理構造体のポインタ value ： スタックへ積むデータ
戻り値	なし
機　能	引数で渡されたデータをスタックに積む。 ・スタックが満杯のときは，result に-1 を設定して終了する。 ・sp が示すスタック領域にデータを格納した後，sp をインクリメントし，result に 0 を設定して終了する。

表３　関数 pop の引数と戻り値の仕様

int pop(struct t_stack *p)	
引　数	p ： スタック管理構造体のポインタ
戻り値	スタックから取り出したデータ
機　能	スタックからデータを取り出して返す。 ・スタックが空のときは，result に-1 を設定して 0 を返却する。 ・sp をデクリメントし，sp が示すスタック領域のデータを取得し，result に 0 を設定して終了する。スタックから取り出したデータを返却する。

キュー操作には，表 4～表 6 の三つの関数を用いる。

表４　関数 queue_init の引数と戻り値の仕様

void queue_init(struct t_queue *p)	
引　数	p ： キュー管理構造体のポインタ
戻り値	なし
機　能	引数で渡されたキュー管理構造体のメンバーを初期化する。 ・データ保持数 count を 0 に設定する。 ・読込みポインタ rp と書込みポインタ wp を 0 に設定する。

表５　関数 set の引数と戻り値の仕様

void set(struct t_queue *p, int value)	
引　数	p ： キュー管理構造体のポインタ value ： キューへ追加するデータ
戻り値	なし
機　能	引数で渡されたデータをキューに追加する。 ・キューが満杯のときは，result に-1 を設定して終了する。 ・wp が示すキュー領域にデータを格納した後，wp をインクリメントする。インクリメントの結果 wp がキュー領域の末尾を超えた場合は，wp に 0 を再設定する。 ・count をインクリメントし，result に 0 を設定して終了する。

表６　関数 get の引数と戻り値の仕様

int get(struct t_queue *p)	
引　数	p ： キュー管理構造体のポインタ
戻り値	キューから取り出したデータ
機　能	キューからデータを取り出して返す。 ・キューが空のときは，result に-1 を設定して 0 を返却する。 ・rp が示すキュー領域のデータを取得した後，rp をインクリメントする。インクリメントの結果 rp がキュー領域の末尾を超えた場合は，rp に 0 を再設定する。 ・count をデクリメントし，result に 0 を設定して終了する。

＜プログラム＞

```
/* スタックに関する操作 */
#define STACK_SIZE  10
struct t_stack {
    int result;                 /* 操作結果 0:正常, -1:異常      */
    int sp;                     /* スタックポインタ              */
    int buffer[STACK_SIZE];     /* スタック領域                  */
};

void stack_init(struct t_stack *p)
{
    p->sp = 0;
}
```

```
void push(struct t_stack *p, int value)
{
    if ( (35) ) {
        p->buffer[p->sp] = value;
        p->sp++;
        p->result = 0;
    }
    else {
        /* スタックが満杯の状態である */
        p->result = -1;
    }
}

int pop(struct t_stack *p)
{
    int value = 0;
    if (p->sp > 0) {
        (36) ;
        p->result = 0;
        value = p->buffer[p->sp];
    }
    else {
        /* スタックが空の状態である */
        p->result = -1;
    }
    return value;
}

/* キューに関する操作 */
#define QUEUE_SIZE  10
struct t_queue {
    int result;                 /* 操作結果 0:正常, -1:異常      */
    int count;                  /* 格納されているデータ数        */
    int rp;                     /* 読込みポインタ                */
    int wp;                     /* 書込みポインタ                */
    int buffer[QUEUE_SIZE];     /* キュー領域                    */
};
```

```
void queue_init(struct t_queue *p)
{
    p->count = 0;
    p->rp = 0;
    p->wp = 0;
}

void set(struct t_queue *p, int value)
{
    if (p->count < QUEUE_SIZE) {
        (37);
        p->wp++;
        if (p->wp == (38))
            p->wp = 0;
        p->count++;
        p->result = 0;
    }
    else {
        /* キューが満杯の状態である */
        p->result = -1;
    }
}

int get(struct t_queue *p)
{
    int value = 0;
    if (p->count > 0) {
        value = p->buffer[p->rp];
        p->rp++;
        if (p->rp == (38))
            p->rp = 0;
        (39);
        p->result = 0;
    }
    else {
        /* キューが空の状態である */
        p->result = -1;
    }
    return value;
}
```

(35) の解答群

ア　p->sp == 0
イ　p->sp == STACK_SIZE
ウ　p->sp < STACK_SIZE
エ　p->sp < STACK_SIZE - 1

(36) の解答群

ア　p->sp++
イ　p->sp--
ウ　p->sp = 0
エ　p->sp = STACK_SIZE

(37) の解答群

ア　p->buffer[0] = value
イ　p->buffer[p->count] = value
ウ　p->buffer[p->rp] = value
エ　p->buffer[p->wp] = value

(38) の解答群

ア　0
イ　p->count
ウ　p->rp
エ　QUEUE_SIZE

(39) の解答群

ア　p->count++
イ　p->count--
ウ　p->count = 0
エ　p->count = QUEUE_SIZE

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＞

プログラムは，8×8 の盤面に配置された風船をすべて割るゲームである。図 1 に，8×8 の盤面に 3 か所の風船が配置されている例を示す。ここで，図 1 中の“B”で示される三つの風船の位置は，（x 座標，y 座標）の形式で指定したとき，それぞれ（2, 1），（5, 2），（4, 6）で表される。

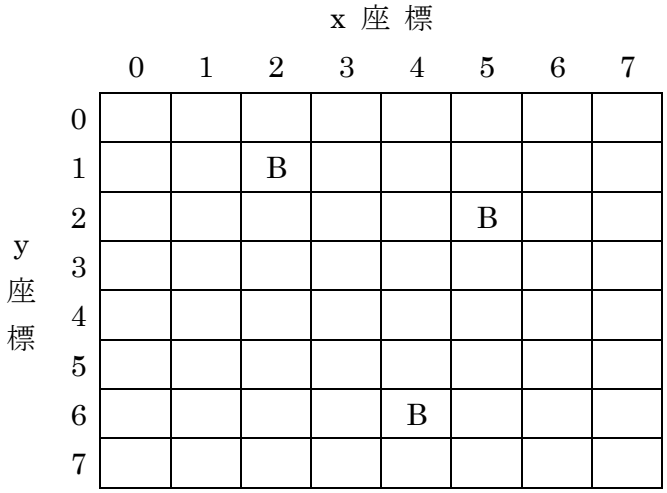


図１　盤面と風船の指定

盤面は 2 次元配列 matrix で管理し，（x 座標，y 座標）で与えられた風船の位置は，matrix[y][x]で表される。例えば，図 1 に示される三つの風船においては，matrix[1][2]，matrix[2][5]，matrix[6][4]に存在する。ここで，matrix においては，風船が存在する位置には 1（BALOON），風船が存在しない位置には 0（EMPTY）が格納される。

風船をすべて割るまで，x 座標，y 座標を指定して関数 attack_baloon を呼び出す。関数 attack_baloon は，指定された x 座標，y 座標により盤面を調べ，表 1 で示す結果を返す。さらに，指定した位置に風船があった場合，その位置の matrix の要素に 0（EMPTY）を設定する。

表１　関数 attack_baloon の結果とその内容

結果	内容
0 （MISS）	（x，y）で指定した盤面の位置およびその位置に接するマス（以下，周辺のマスという）に風船がない。図 2 を参照のこと。
1 （NEAR）	（x，y）で指定した盤面の位置に風船はないが，その周辺のマスのいずれかに風船がある。
2 （HIT）	（x，y）で指定した盤面の位置に風船があり，その位置の風船を割っても，まだ盤面上に別の風船が残っている。
3 （COMPLETED）	（x，y）で指定した盤面の位置に風船があり，その位置の風船を割ることで，盤面上のすべての風船が割れた。

図2に示すように，x＝1，y＝3を指定した場合は0（MISS）が，x＝5，y＝6を指定した場合は1（NEAR）が，x＝5，y＝2を指定した場合は2（HIT）が返却される。

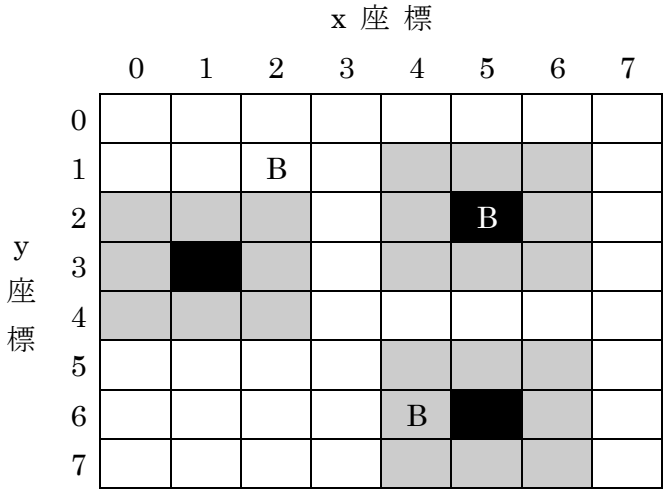


図2　指定した位置と風船の位置との関係

風船を割るゲームでは，表2～表4に示す関数を使用する。なお，RANGEは，盤面の1辺のサイズを示す。

表2　関数attack_baloonの引数と戻り値の仕様

int attack_baloon(int matrix[][RANGE], int x, int y)	
引　数	matrix：盤面 x：x座標，y：y座標
戻り値	0，1，2，3（それぞれMISS，NEAR，HIT，COMPLETED）のいずれかが返却される。
機　能	指定されたx座標，y座標により盤面を調べ，以下の結果を返す。 ・関数check_hitを呼び出し，戻り値をresultに設定する。 ・resultが2（HIT）のとき指定された位置の風船を割った後，check_completed関数を呼び出し，その結果が3（COMPLETED）ならresultに3（COMPLETED）を設定する。 ・resultの値を返却する。

表3　関数check_hitの引数と戻り値の仕様

int check_hit(int matrix[][RANGE], int x, int y)	
引　数	matrix：盤面 x：x座標，y：y座標
戻り値	0，1，2（それぞれMISS，NEAR，HIT）のいずれかが返却される。
機　能	指定されたx座標，y座標により盤面を調べ，以下の結果を返す。 ・指定された位置に風船があれば2（HIT）を返す。 ・指令された位置の風船の周辺のマスに風船があれば1（NEAR）を返す。 ・上記以外のときは0（MISS）を返す。

表４　関数 check_completed の引数と戻り値の仕様

int check_completed(int matrix[][RANGE])	
引　数	matrix : 盤面
戻り値	3，0（それぞれ COMPLETED，NOTCOMPLETED）のいずれかが返却される。
機　能	盤面のすべてのマスを調べ，風船が残っていなければ 3（COMPLETED）を， 一つでも風船が残っていれば 0（NOTCOMPLETED）を返す。

＜プログラム＞

```
#define RANGE   8           /* 盤面の大きさ RANGE × RANGE */

#define EMPTY   0           /* 風船なし      */
#define BALOON  1           /* 風船あり      */

#define COMPLETED     3     /* すべてヒット  */
#define NOTCOMPLETED  0     /* 残っている    */
#define HIT           2     /* ヒット        */
#define NEAR          1     /* 近くにある    */
#define MISS          0     /* ミス          */

/* 指定された位置をチェックする */
int check_hit(int matrix[][RANGE], int x, int y)
{
    int px, py;

    if ( [ (40) ] )
        return HIT;

    for (py = y - 1; py <= y + 1; py++) {
        if ( [ (41) ] )
            continue;
        for (px = x - 1; px <= x + 1; px++) {
            if (px < 0 || px >= RANGE)
                continue;
            if (matrix[py][px] == BALOON)
                return [ (42) ];
        }
    }
    return [ (43) ];
}
```

```
/* すべての風船が割れたかチェックする */
int check_completed(int matrix[][RANGE])
{
    int x, y;

    for (y = 0; y < RANGE; y++) {
        for (x = 0; x < RANGE; x++) {
            if ( (40) )
                return NOTCOMPLETED;
        }
    }
    return COMPLETED;
}

/* 指定された位置の風船を割る */
int attack_baloon(int matrix[][RANGE], int x, int y)
{
    int result;

    result = check_hit(matrix, x, y);
    if (result == HIT) {
        (44);
        if (check_completed(matrix) == COMPLETED)
            result = COMPLETED;
    }
    return result;
}
```

(40) の解答群

ア　matrix[y][x] == BALOON
イ　matrix[y][x] == EMPTY
ウ　matrix[0][x] == BALOON
エ　matrix[y][0] == BALOON

(41) の解答群

ア　py == 0 || py == RANGE
イ　py < 0 && py >= RANGE
ウ　py < 0 || py >= RANGE
エ　py > 0 || py < RANGE

(42), (43) の解答群

ア　COMPLETED
イ　HIT
ウ　MISS
エ　NEAR

(44) の解答群

ア　matrix[y][x] = BALOON
イ　matrix[y][x] = EMPTY
ウ　matrix[y][x] = HIT
エ　matrix[y][x] = COMPLETED

サンプル問題

C言語プログラミング能力認定試験

2級

＜正答＞

問 1

(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
ア	イ	ア	イ	ア	ア	イ	イ

問 2

(9)	(10)	(11)	(12)	(13)	(14)
ウ	イ	ウ	イ	オ	ウ

問 3

(15)	(16)	(17)	(18)	(19)
ア	ウ	イ	ア	エ

問 4

(20)	(21)	(22)	(23)	(24)	(25)
ア	エ	ウ	オ	エ	オ

問 5

(26)	(27)	(28)	(29)
エ	ア	オ	ア

問 6

(30)	(31)	(32)	(33)	(34)
エ	イ	ア	イ	オ

問 7

(35)	(36)	(37)	(38)	(39)
ウ	イ	エ	エ	イ

問 8

(40)	(41)	(42)	(43)	(44)
ア	ウ	エ	ウ	イ

サーティファイ
情報処理能力認定委員会