日商プログラミング検定 EXPERT（C 言語）サンプル問題

知識科目

第 1 問（知識 4 択：20 問）

1. 変数宣言において、変更することができない変数を指定する修飾子はどれか。
次の中から最も適切なものを選びなさい。

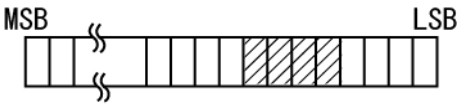
① const
② volatile
③ extern
④ static

2. 以下のように宣言した 2 次元配列において、x[2][1]の値はどれか。次の中から最も適切なものを選びなさい。

```
int x[3][4] = {
        {1, 2},
        {3, 4, 5},
        {6}
};
```

① 0
② 4
③ 5
④ 6

3. unsigned int 型変数 x に対し、以下の演算を行った。
x ^ (~0 & 0xf0)
下図が変数 x のビット配列を表しているとき、演算の結果はどれか。次の中から最も適切なものを選びなさい。



① 図の斜線部分のビットを反転する。
② 図の斜線部分以外のビットを反転する。
③ 図の斜線部分のビットを 0 にする。

④ 図の斜線以外の部分のビットを 0 にする。

4. 以下のプログラムがコンパイルエラーとなる理由として、ふさわしい説明はどれか。次の中から最も適切なものを選びなさい。ただし、省略部分は正しく記述されており、この部分に標準入出力に関する記述はないものとする。

```
#define N    10;

int main(void){
      int i = 0;
      for ( ; i < N; i++){
            <省略>
      }
}
```

① プリプロセッサの文に「;」があるから。
② #include<stdio.h>がないから。
③ for 文中に初期化の式がないから。
④ 「}」が不足しているから。

5. 以下のプログラム片において、（A）に入れる「その他」にあたるキーワードはどれか。次の中から最も適切なものを選びなさい。

```
int y = 1;
switch (n) {
      case 2:
            y = y * 10;
      case 1:
            y = y * 10;
      (A):
            y = 0;
}
```

① default
② break
③ continue
④ while

6. 以下のプログラム片の動作の説明を選びなさい。次の中から最も適切なものを選びなさい。ただし、関数 func() の型は int 型であり、正しく定義されているものとする。

```
int i;
for(i = 0; i < 20; i++){
      if ( !func()){
            break;
      }
}
```

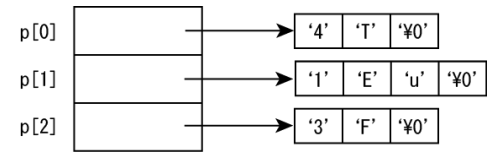
① 関数 func() を繰り返し実行し、戻り値が 0 となったとき、または 20 回繰り返したとき、
繰り返しを終了する。
② 関数 func()を 20 回繰り返し実行する。
③ 関数 func()の戻り値が 0 になるまで、繰り返し実行する。
④ 関数 func()を繰り返し実行し、戻り値が 0 になり、かつ、20 回繰り返したとき、繰り返しを終了する。

7. int 型を指すポインタ p を宣言する文はどれか。次の中から最も適切なものを選びなさい。

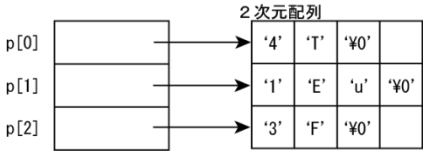
① int *p;
② int p;
③ int &p;
④ int **p:

8．以下のようにポインタ配列が宣言されているとき、どのようなデータ構成になるか。次の中から最も適切なものを選びなさい。
char *p[] = { "4T" , "1Eu" , "3F" };

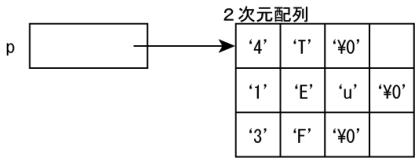
①



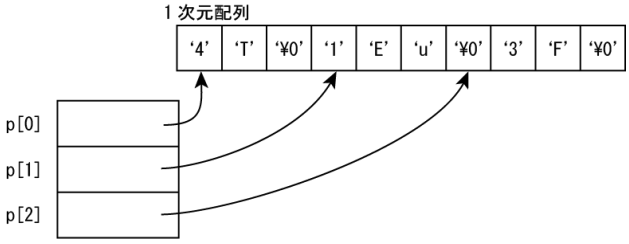
②



③



④



9. 関数に関する説明のうち、誤っているものはどれか。次の中から最も適切なものを選びなさい。

① 関数呼出し文と、呼び出される関数は、互いに同じファイル内に記述していなければならない。
② 関数内には return 文を複数記述することができる。
③ プロトタイプ宣言を収めたヘッダファイルを include することで、プロトタイプ宣言の記述に代えることができる。
④ ライブラリ関数のヘッダファイルを include するときは、ファイル名を「<」と「>」で囲むが、ソースプログラムと同じフォルダに保存した自作ヘッダファイル名は「"」で囲んで記述する。

10. 関数の型として指定することができないのはどれか。次の中から選びなさい。

① int[]
② int
③ void
④ char

11. 関数を呼び出したとき、呼出し側の実引数は影響を受けない関数はどれか。次の中から最も適切なものを選びなさい。

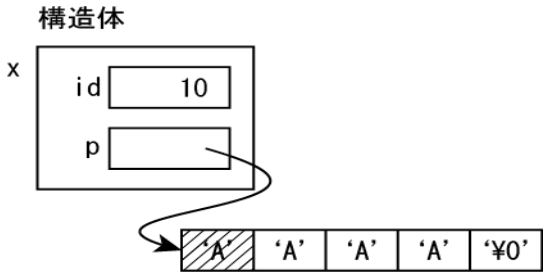
①
```
void func(int x){
      x++;
}
```
②
```
void func(int[] x){
      x[0]++;
}
```
③
```
void func(int *px){
      *px++;
}
```
④
```
void func(int *px){
      px[0]++;
}
```

12. 以下のような int 型関数が定義されているとき、関数呼出し func(4) による戻り値はいくつか。次の中から選びなさい。

```
int func(int x) {
    if (x == 1)
        return 1;
    else
        return func(x - 1) * x;
}
```

① 24
② 4
③ 1
④ 3

13. 下図のような構造体型変数 x において、斜線部分の値を'B'に変更する文はどれか。次の中から選びなさい。ただし、メンバ id は int 型、メンバ p は char 型へのポインタとする。



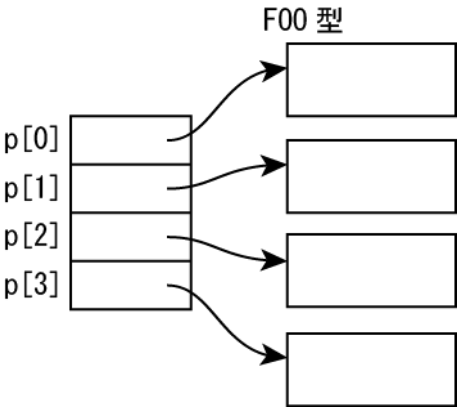
① *x.p = 'B';
② *x.p = ''B'';
③ x*p = 'B';
④ x*p = ''B'';

14. 自己参照の構造を実現するために構造体を定義する。(A)に入るプログラム片はどれか。次の中から最も適切なものを選びなさい。

```
typedef struct foo{
        int data1;
        int data2;
        (A)  p_next;
}FOO;
```

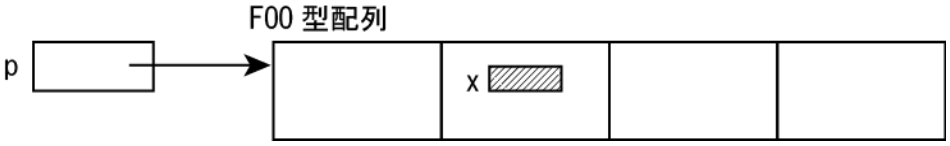
① struct foo *
② FOO *
③ foo *
④ struct foo

15. 下図のようなメモリ構造を構築したい。配列 p を宣言する文はどれか。次の中から最も適切なものを選びなさい。ここで、FOO 型は構造体を利用した型である。



① FOO *p[4]
② FOO[4] *p
③ FOO p[4]
④ FOO p

16. 構造体を利用した FOO 型の配列をポインタ p が下図のように指すとき、FOO 型のメンバ x を指すプログラム片はどれか。次の中から選びなさい。



① (p+1)->x
② p->x
③ (p+1).x
④ p.x

17. 下記の記述があるとき、MACRO1(10 , 20)の値はどれか？次の中から選びなさい。

```
#define MACRO1(a, b) ((a) +(b))
```

① 30
② 10
③ 20
④ -10

18. キーボードから入力されたデータを構造体を利用した FOO 型に格納して返すという機能を持った関数 func() を作成する。この関数のプロトタイプ宣言として、適切でないものはどれか。次の中から選びなさい。

① void func(FOO x)
② FOO func(void)

③ void func(FOO *p)
④ FOO *func(FOO *p)

19. 以下のように定義されているとき、ビットフィールド nengo に格納できる最大値はいくつか。次の中から選びなさい。

```
typedef struct  {
    unsigned int nengo : 3;
    unsigned int year : 8;
    unsigned int month : 5;
    unsigned int day : 6;
}WAREKI;
```

① 7
② 3
③ 17
④ 8

20. 以下のプログラム片を含むプログラムをコンパイルならびに実行したときの動作についての説明はどれか。次の中から最も適切なものを選びなさい。

```
char *p = “ABC” ;
*p = ‘B’ ;
```

① コンパイルは成功し、実行可能ファイルが作成されるが、実行時エラーが発生して、実行することはできない。
② コンパイルは成功し、実行可能ファイルが作成され、実行すると、ポインタ p の指す領域は'B'に変更される。
③ コンパイルエラーとなり、実行可能ファイルは作成されず、実行できない。
④ コンパイル終了時に警告が発生するが、実行可能ファイルは作成され実行すると、ポインタ p の指す領域は'B'に変更される。

第 2 問（穴埋め：2 問）

【問題 1 】

列車の座席を自動販売機で予約することをシミュレートするプログラムである。予約のルールは以下のとおりである。

(1) 停車駅と座席のシート名は、グローバル変数 p_station と p_seat から与えられるものとする。
(2) 乗車駅番号と降車駅番号を入力すると、購入可能な座席番号を表示する。
(3) 座席番号を入力すると、予約が完了する。予約可能な座席がないときは、座席番号の入力はできない。
(4) 乗車駅と降車駅の間が連続して空席であれば予約することができる。
(5) 切符は 1 回の操作で 1 枚しか購入することはできない。
(6) 1 回購入するごとに、購入操作を続けるか、終了するか問合せを行い、1 と入力すると、次の切符の購入操作を行うことができる。0 と入力すると、プログラムを終了する。

なお、問題簡略化のため、車両の情報は省略しており、また、キーボードからの入力の誤りはないものとする。

例えば、座席番号 4A に上野から土浦まで予約が確定しているとき、土浦から友部までは新に予約をすることができるが、柏から友部までの予約はできない。

駅番号	駅名
0	上野
1	柏
2	土浦
3	石岡
4	友部

座席番号 4A

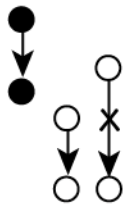


図 1　予約例

このプログラムは、以下の構造体を利用している。

表 1　構造体 Seat 型の構成

型	メンバ	説明
char *	p_name	座席名
Int	reserved[STATION_N-1]	駅間の予約状況。0：空席　1：予約済み

表 2　グローバル領域に宣言されている配列

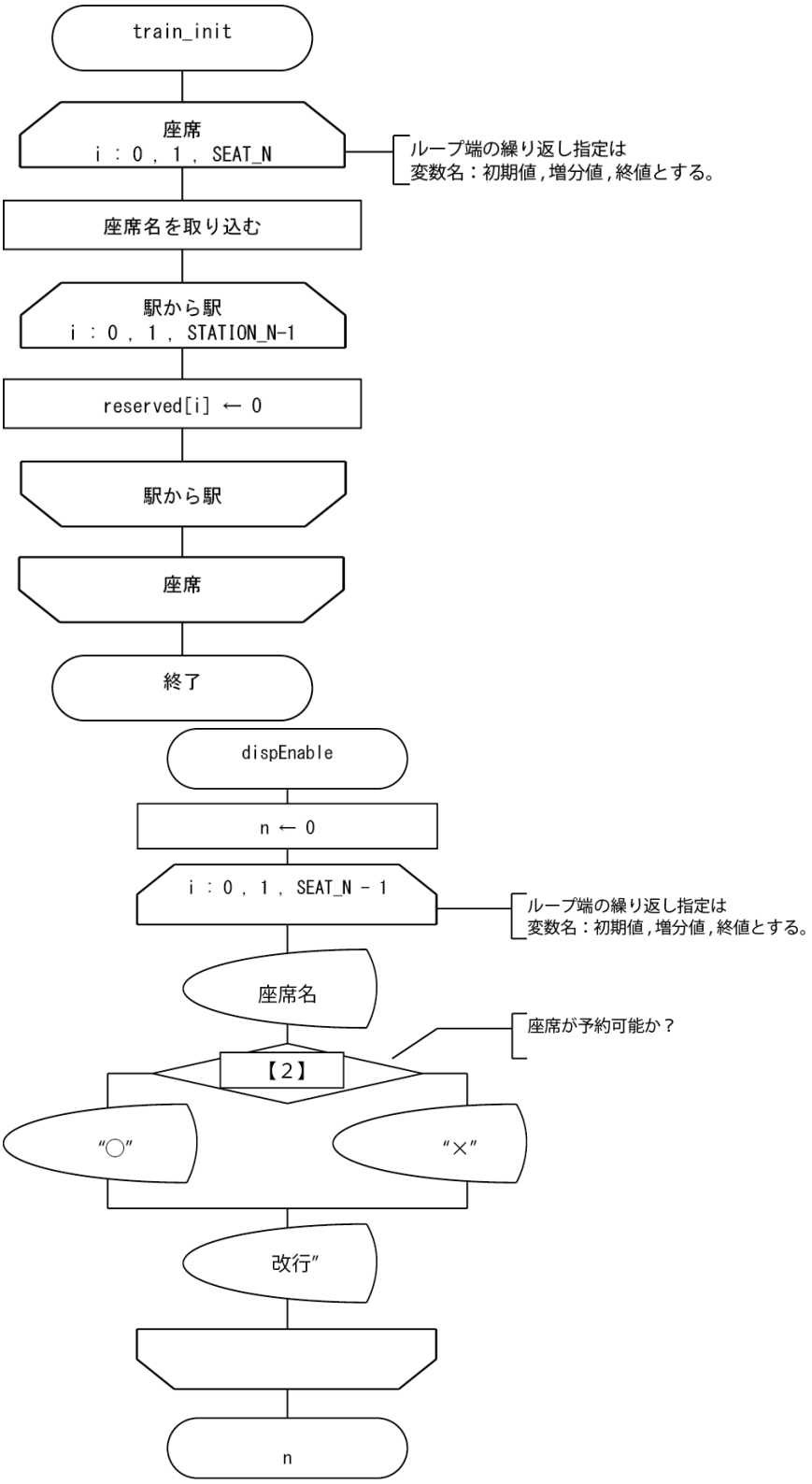
型	メンバ	説明
char *	p_station[STATION_N]	駅名を定義
char *	p_seat[SEAT_N]	座席名を定義

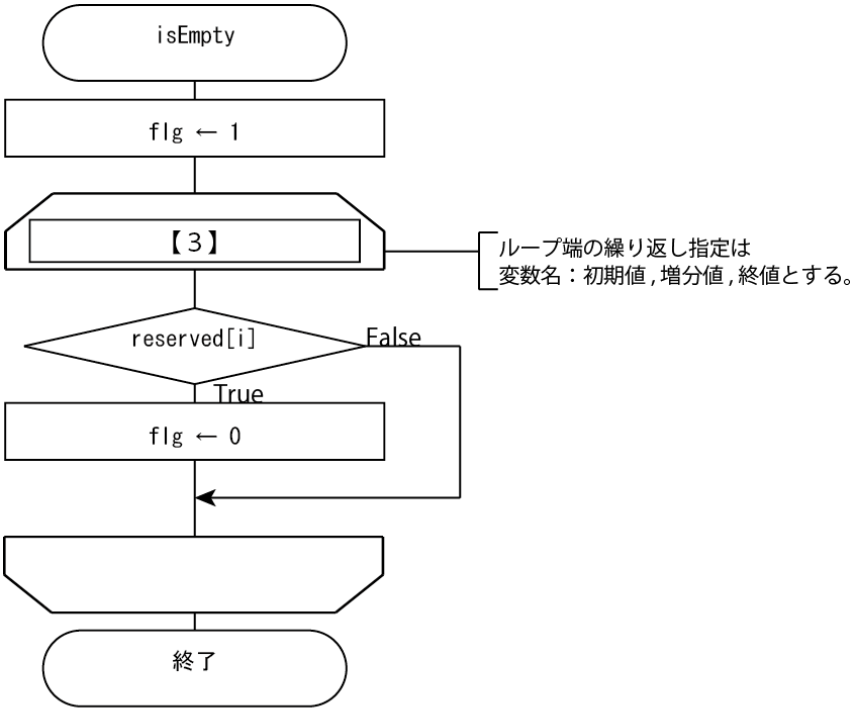
プログラムで定義し、利用している関数の仕様は以下のとおりである。

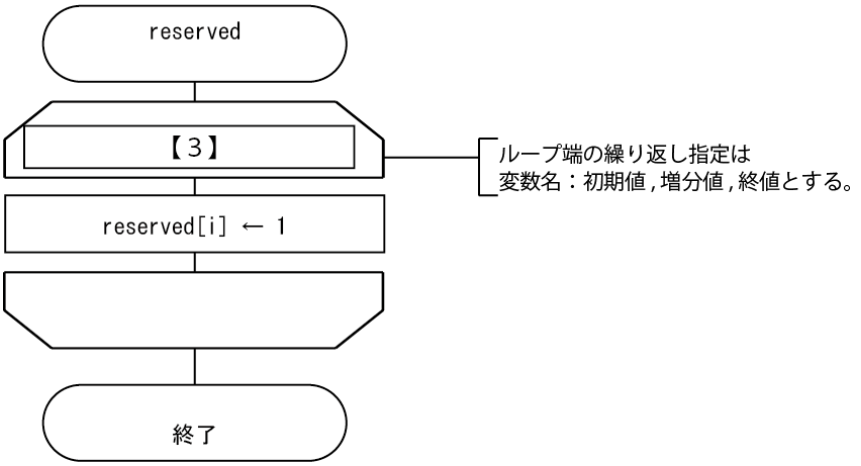
表 3　定義している関数

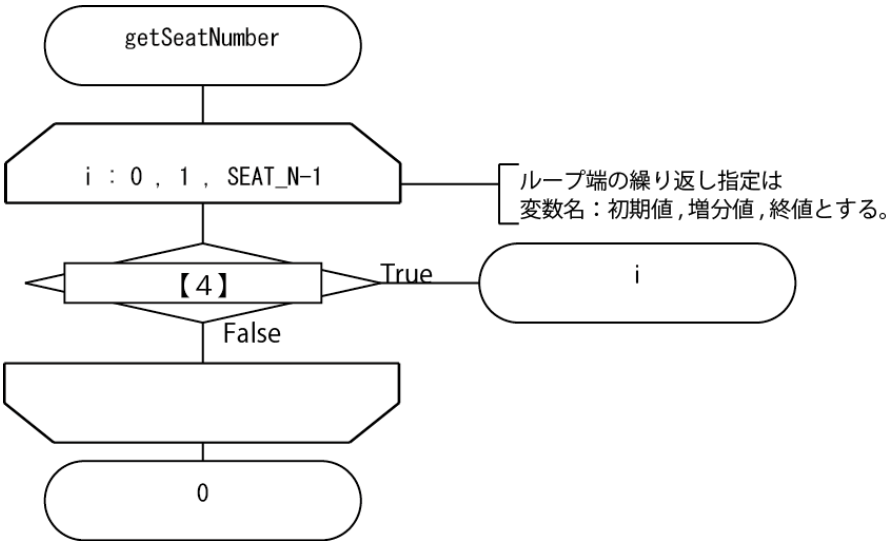
戻り値	関数の形式と機能
void	train_init(TRAIN *p_train) 列車情報を初期化する。座席名を取り込み、すべて空席とする。
int	input(TRAIN *p_train) 乗車駅番号と降車駅番号をキーボードから入力し、各座席が予約可能かどうかを表示する。予約可能な座席が１つ以上あるときは、キーボードから選択する座席を入力し、予約を確定する。 予約動作を終了する場合は 0 を、再度予約を行う場合は 0 以外をキーボードから入力する。
int	dispEnable(TRAIN train , int bording, int getOff) train : 列車情報 bording : 乗車駅番号 getOff : 降車駅番号 指定された乗車駅・降車駅間の全座席の予状況について、乗車駅から降車駅までの間がすべて空席であれば○、１区間でも予約済みであれば×を表示する。○の座席の数を返す。
int	isEmpty(TRAIN train, int bording, int getOff , int index) train : 列車情報 bording : 乗車駅番号 getOff : 降車駅番号 index : 座席番号 指定された座席について、乗車駅・降車駅間がすべて空席であれば 1 を、１区間でも予約済みであれば 0 を返す。
void	reserved(SEAT *p_seat ,int bording, int getOff) p_seat : 座席を指すポインタ bording : 乗車駅番号 getOff : 降車駅番号 指定座席の乗車駅・降車駅間をすべて予約済みとする。
int	getSeatNumber(SEAT seats[] , char *p_name) seats : 列車の全座席 p_name : 座席名 seats の座席から指定の座席名を検索し、座席番号を返す。

【アルゴリズム】


train_init
座席
i ： 0 ， 1 ， SEAT_N
ループ端の繰り返し指定は
変数名：初期値,増分値,終値とする。
座席名を取り込む
駅から駅
i ： 0 ， 1 ， STATION_N-1
reserved[i] ← 0
駅から駅
座席
終了
dispEnable
n ← 0
i ： 0 ， 1 ， SEAT_N － 1
ループ端の繰り返し指定は
変数名：初期値,増分値,終値とする。
座席名
座席が予約可能か？
【2】
“○”
“×”
改行”
n


isEmpty
flg ← 1
【3】
ループ端の繰り返し指定は
変数名：初期値,増分値,終値とする。
reserved[i]
False
True
flg ← 0
終了


reserved
【3】
ループ端の繰り返し指定は
変数名：初期値,増分値,終値とする。
reserved[i] ← 1
終了


getSeatNumber
i : 0 , 1 , SEAT_N-1
ループ端の繰り返し指定は
変数名：初期値,増分値,終値とする。
【4】
True
i
False
0

【プログラム】

```
#include <stdio.h>
#include <string.h>
#define SEAT_N          4
#define STATION_N       5

char *p_station[STATION_N] = { "上野" , "柏" , "土浦" , "石岡" , "友部" };
char *p_seat[SEAT_N] = { "1A" , "2A" , "3A" , "4A" };

typedef struct seat {
        char *p_name;
        int   reserverd[STATION_N - 1];
}SEAT;

typedef struct train {
        SEAT            seats[SEAT_N];
}TRAIN;

void train_init(TRAIN *p_train);
int input(TRAIN *p_train);
int isEmpty(TRAIN train, int bording, int getOff, int index);
int dispEnable(TRAIN train, int bording, int getOff);
void reserved(SEAT *p_seat, int bording, int getOff);
int getSeatNumber(SEAT seats[], char *p_name);

int main(void) {
        TRAIN tokiwa55;
        train_init(&tokiwa55);

        while (input(&tokiwa55));
}

void train_init(TRAIN *p_train) {
        int i , j;
        for (i = 0; i < SEAT_N; i++) {
                p_train->seats[i].p_name = p_seat[i];
                for (j = 0; j < STATION_N - 1; j++) {
                        p_train->seats[i].reserverd[j] = 0;
                }
        }
```

```
}

int input(TRAIN *p_train) {
        int bording;
        int getOff;
        char seatName[10];
        int c;
        int n;

        printf("乗車駅番号：");
        scanf("%d", &bording);
        printf("降車駅番号：");
        scanf("%d", &getOff);
        if (dispEnable(*p_train, bording, getOff)) {
                printf("座席を選んでください：");
                scanf("%s", seatName);
                n = getSeatNumber(p_train->seats, seatName);
                reserved(【１】, bording, getOff);
        }
        printf("継続しますか？  0:End other:Continue：");
        scanf("%d", &c);
        return c;
}

int dispEnable(TRAIN train , int bording, int getOff) {
        int i;
        int n = 0;
        for (int i = 0; i < SEAT_N; i++) {
                printf("%s ", train.seats[i].p_name);
                if (【２】) {
                        printf("○  ");
                        n++;
                }
                else
                        printf("×  ");
        }
        printf("¥n");
        return n;
```

```
}

int isEmpty(TRAIN train, int bording, int getOff , int index) {
        int i;
        int flg = 1;
        for (【３】) {
                if (train.seats[index].reserverd[i]) flg = 0;
        }
        return flg;
}

void reserved(SEAT *p_seat ,int bording, int getOff) {
        int i;
        for (【３】) {
                p_seat->reserverd[i] = 1;
        }
}

int getSeatNumber(SEAT seats[] , char *p_name) {
        int i;
        for (int i = 0; i < SEAT_N; i++) {
                if(【４】) return i;
        }
        return 0;
}
```

次の中から、上の空欄【１】～【４】に入る最も適切なものを選びなさい。

【１】　（１）　&(p_train->seats[n])
　　（２）　p_train->seats[n]
　　（３）　&p_train
　　（４）　p_train

【２】　（１）　isEmpty(train, bording , getOff , i)
　　（２）　isEmpty(&train, bording , getOff , i)
　　（３）　isEmpty(train.seats, bording , getOff , i)

（４）　isEmpty(train->seats, bording , getOff , i)

【３】　（１）　i = bording; i < getOff; i++

（２）　i = bording; i <= getOff; i++

（３）　int i = 0; i < SEAT_N; i++

（４）　int i = 0; i <= SEAT_N; i++

【４】　（１）　strcmp(seats[i].p_name, p_name) == 0

（２）　strcmp(seat.p_name, p_name) == 0

（３）　strcmp(train.seats[i].p_name, p_name) == 0

（４）　strcmp(train->seats[i].p_name, p_name) == 0

【問題 2】

音符を LED で表現するプログラムをシミュレートする。

鍵盤の一つひとつに LED が付いており、音を LED の発光で表現する。各鍵盤には図 1 に示す番号が付いている。ここでは、0 番から 12 番まで 13 個の LED の点灯・消灯を制御する。

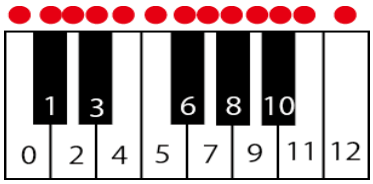


図 2 鍵盤の番号

音符は、開始拍と音階、それに音の長さで表現する。例えば、図 2 の譜面の例では、表 1 のように表現する。第 0 拍のように音のないときもあれば、第 8 拍のように複数の音が重なることもある。音の長さは 8 分音符を 1 とする。

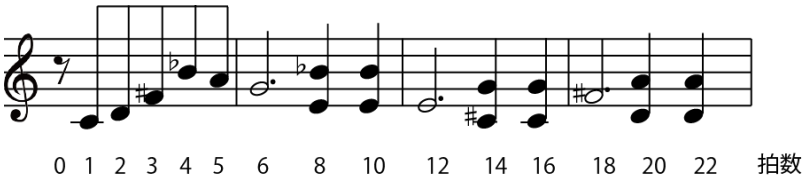


図 3 譜面の例

表 4 音の表現

拍	音符 1	長さ 1	音符 2	長さ 2	音符 3	長さ 3
0						
1	0	1				
2	2	1				
3	6	1				
4	10	1				
5	9	1				
6	7	6				
7	7	6				
8	7	6	4	2	10	2
9	7	6	4	2	10	2

上の例では、第 0 拍では音がなく、LED はすべて消灯している。第 8 拍は、第 6 拍からの音の続きがあり、さらに 2 音が追加されて同時に 3 つの LED が点灯する。

13 個の LED を制御するため、図 3 のような仮想のコンピュータと仮想のシフトレジスタ 2 個を利用する。シフトレジスタ 1 個には 8 個の LED を接続することができる。

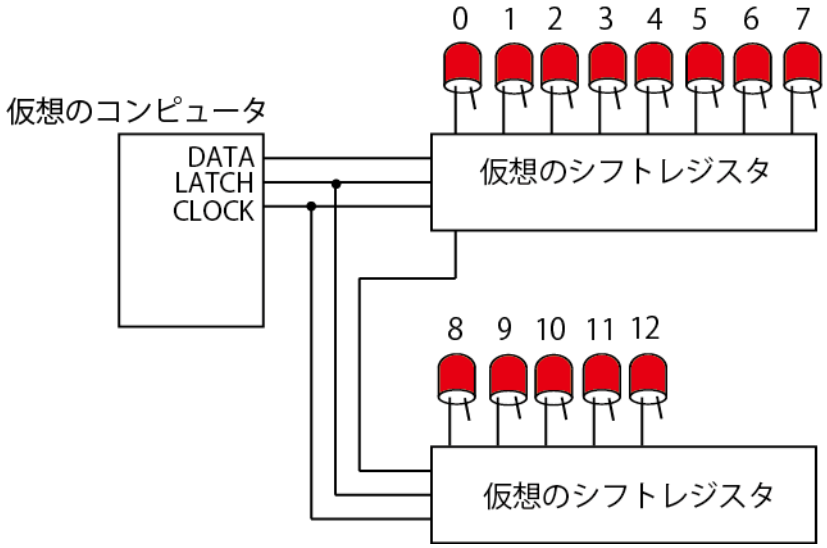


図 4　仮想コンピュータによる LED 制御回路図

シフトレジスタは、LED 1 個ずつの ON/OFF の状態を順に出力すると、各出力ピンで状態を保持することができる。13 個の LED を制御する手順は以下のとおりである。ただし、本プログラムは、仮想であるため、実際の信号出力の代わりに、LATCH 信号と CLOCK 信号についてはコメント文を記載し、各 LED の状態については、コマンドプロンプトに表示することとする。

LATCH信号LOW	:
LED 0の状態を出力	:
CLOCK信号HIGHT	LED 12の状態を出力
CLOCK信号LOW	CLOCK信号HIGHT
LED 1の状態を出力	CLOCK信号LOW
CLOCK信号HIGHT	LATCH信号HIGHT
CLOCK信号LOW	
:	
:	

図 2 の音符の一つひとつは、以下の構造体に記録されており、これが配列になっている。

表 5　音符を扱う構造体の構成

型	メンバ名	説明
Int	start_time	開始拍数
Int	onkai	音の番号（図1による）
Int	length	音の長さ。八分音符を 1 とする。

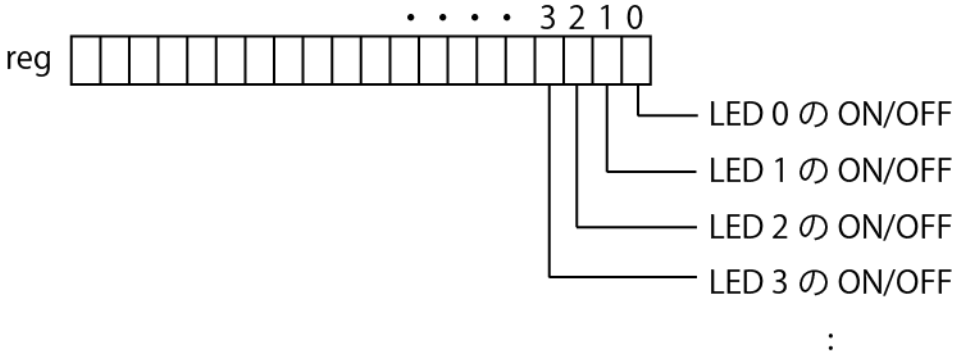
例えば図 2 の 1 泊目は八分音符のドであり、構造体の内容は以下のように記録される。

start_time	1	第 1 泊から始まる。

onkai	0	音階 0 （ド）
length	1	音の長さは八分音符

1 回分の 13 個の LED への出力を行う関数 output の仕様は以下のとおりである。

表 6　関数 output の仕様

戻り値	関数の形式と機能
void	output(int i , int reg) i : 拍数（表示用） reg：13 個の LED の状態をビットごとに記録。LED ON を 1、OFF を 0 とする。 

1 拍分の 13 個の LED の出力を int 型変数 reg に保持し、各音符の残りの長さを配列 count で保持する。例えば、表 1 の第 6 拍では、音符 7 に 6 拍分の長さの音がある。このときの reg と count の内容は以下のとおりである。

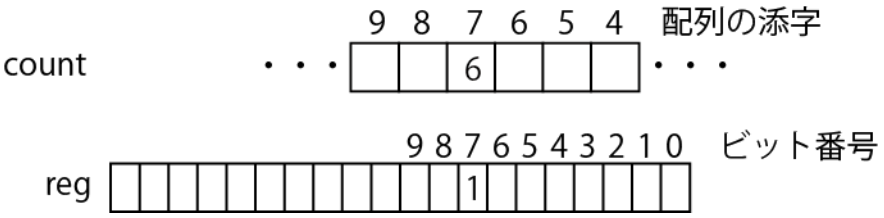
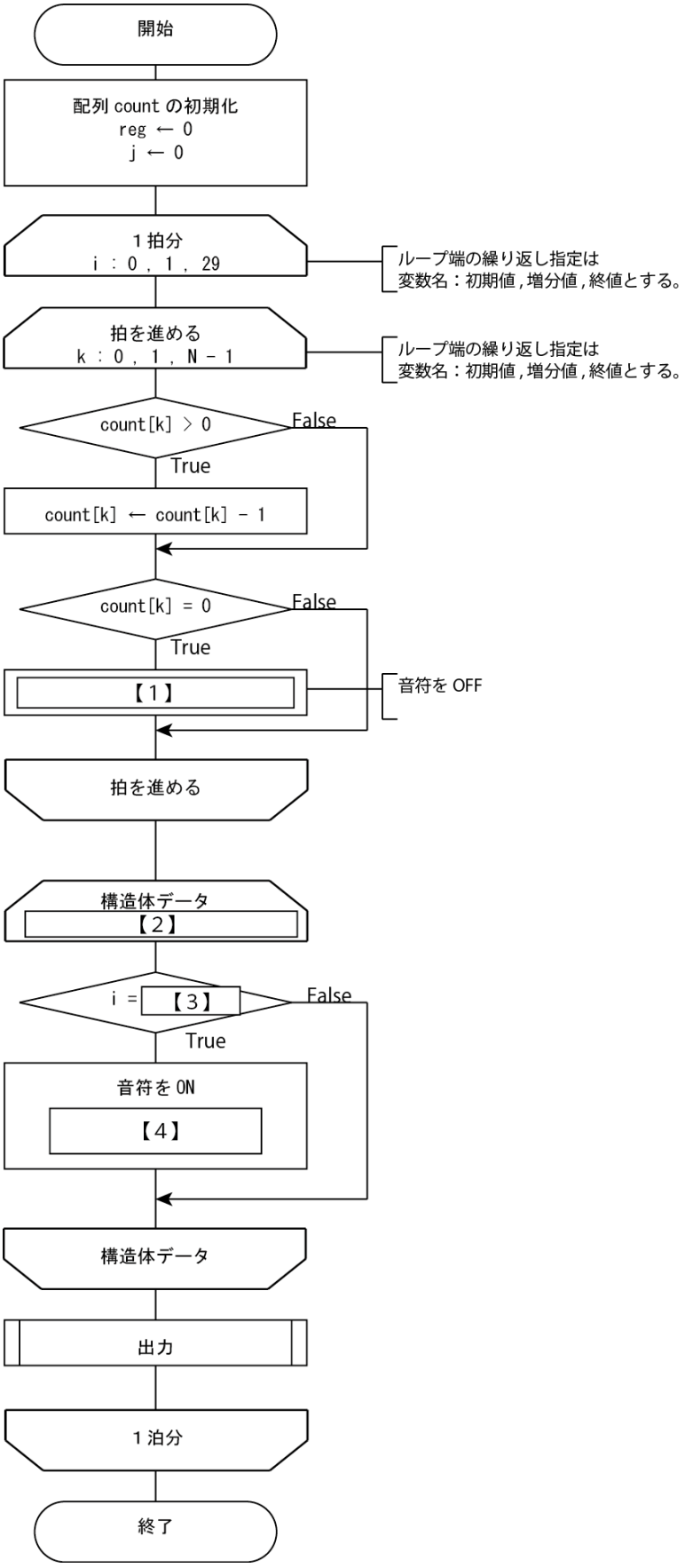


図 5　第 6 拍における変数 reg と配列 count

拍が進むにつれ、count の値を 1 ずつ減らし、0 になると LED を OFF とする。
なお、プログラムでは、30 拍分の出力を行っている。

【アルゴリズム】


開始
配列 count の初期化
reg ← 0
j ← 0
1 拍分
i : 0 , 1 , 29
ループ端の繰り返し指定は
変数名：初期値,増分値,終値とする。
拍を進める
k : 0 , 1 , N − 1
ループ端の繰り返し指定は
変数名：初期値,増分値,終値とする。
count[k] > 0
False
True
count[k] ← count[k] − 1
count[k] = 0
False
True
【1】
音符を OFF
拍を進める
構造体データ
【2】
i = 【3】
False
True
音符を ON
【4】
構造体データ
出力
1 泊分
終了

【プログラム】

```
#include <stdio.h>
#define N 13

typedef struct onpu {
        int start_time;
        int onkai;
        int length;
}ONPU;

void output(int i, int reg);

int main(void) {
        ONPU data[] = {
                {1,0,1},{2,2,1},{3,6,1},{4,10,1},{5,9,1},
                {6,7,6},{8,4,2},{8,10,2},{10,4,2},{10,10,2},
                {12,4,6} ,{14,1,2},{14,7,2},{ 16,1,2 },{ 16,7,2 },
                {18,6,6},{20,2,2},{20,9,2},{ 22,2,2 },{ 22,9,2 }
        };

        int reg;
        int count[N];
        int i, j, k;

        for (k = 0; k < N; k++) {
                count[k] = 0;
        }
        reg = 0;
        j = 0;
        for (i = 0; i < 30; i++) {
                for (k = 0; k < N; k++) {
                        if (count[k] > 0) {
                                count[k]--;
                                if (count[k] == 0)
                                        【1】;
                        }
                }
                for (; i >= data[j].start_time; j++) {
                        if (i == 【3】) {
                                reg |= 1 << data[j].onkai;
```

```
                                【４】;
                        }
                }
                output(i , reg);
        }

        return 0;
}

void output(int i , int reg) {
        int k;

        //LATCH LOW
        printf("%2d : output to DATAPIN " , i);
        for (k = 0; k < N; k++) {
                printf(" %x ", (reg >> k) & 1);
                //CLOCK HIGHT
                //CLOCK LOW
        }
        printf("¥n");
        //LATCH HIGHT
        //0.5秒休み

}
```

次の中から、上の空欄【１】～【４】に入る最も適切なものを選びなさい。

【１】　（１）　reg ^= 1 << k
　　（２）　reg = k
　　（３）　reg |= 1 << k
　　（４）　reg ^= 1

【２】　（１）　; i >= data[j].start_time; j++
　　（２）　j = 0; i >= data[j].start_time; j++
　　（３）　j = 0; i > data[j].start_time; j++
　　（４）　; i > data[j].start_time; j++

【３】　（１）　data[j].start_time
　　（２）　data[j]

（３）　data[j].onkai

（４）　data[j].length

【４】　（１）　count[data[j].onkai] = data[j].length

（２）　count[j] = data[j].length

（３）　count[i] = data[j].length

（４）　count[data[i].onkai] = data[j].length

第３問（読解：１問）

【問題】

社会現象をシミュレートするときに欠かすことができないのが乱数である。C言語には一様乱数を発生する関数が整備されており、これを使って各種乱数を発生することができる。一様乱数を発生する関数は以下のとおりである。

ヘッダファイル	戻り値	関数の形式
math.h	Int	rand() 0 から RAND_MAX のうちの１つの整数を得る。

以下は乱数を１つ発生するプログラムである。

【プログラム】

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

//関数のプロトタイプ宣言
int func(int a, double b);
int One(double b);

int func(int a, double b) {
        int i;
        int k = 0;
        for (i = 0; i < a; i++) {
                k += One(b);
        }
        return k;
}

int One(double b) {
        double rnd = (double)rand() / RAND_MAX;
        if (rnd < b) return 1;
        else return 0;
}
```

上のプログラムにより生成される乱数の特徴を表しているのは次のうちどれか。最もふさわしいものを選択しなさい。

【選択肢】

【１】　関数 func を何度も呼び出すことで、成功する確率が b である事象を a 回行ったとき、成功する回数の分布が得られる。

【２】　関数 func を何度も呼び出すことで、時間 b のあいだに a 回事象が発生する分布が得られる。

【３】　関数 func を何度も呼び出すことで、平均 a、分散 b の正規分布が得られる。

【４】　関数 func を何度も呼び出すことで、生起率 b で事象が起きるとき、次に起きるまでの時間の分布が得られる。

実技科目

【問題１】

スタート地点からゴール地点に移動するにあたり、複数のルートの中から、もっとも短時間での移動が見込まれるルートを選択する。ただし、途中に休憩できる施設があるルートでなければならない。例えば、下表のように５つのルートがある場合、ルートＣを選択する。

表 7 ルート一覧の例

	ルートA	ルートB	ルートC	ルートD	ルートE
休憩施設	なし	あり	あり	なし	あり
見込み時間	3.8 時間	4.5 時間	4.2 時間	3.7 時間	4.4 時間

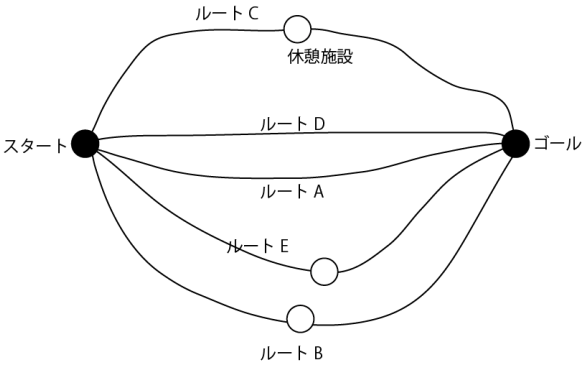


図 6　ルート例

各地点に関する情報は、表２のような構造体に収められており、ルート数を要素数とする構造体配列を構成している。

表 8　構造体の構成

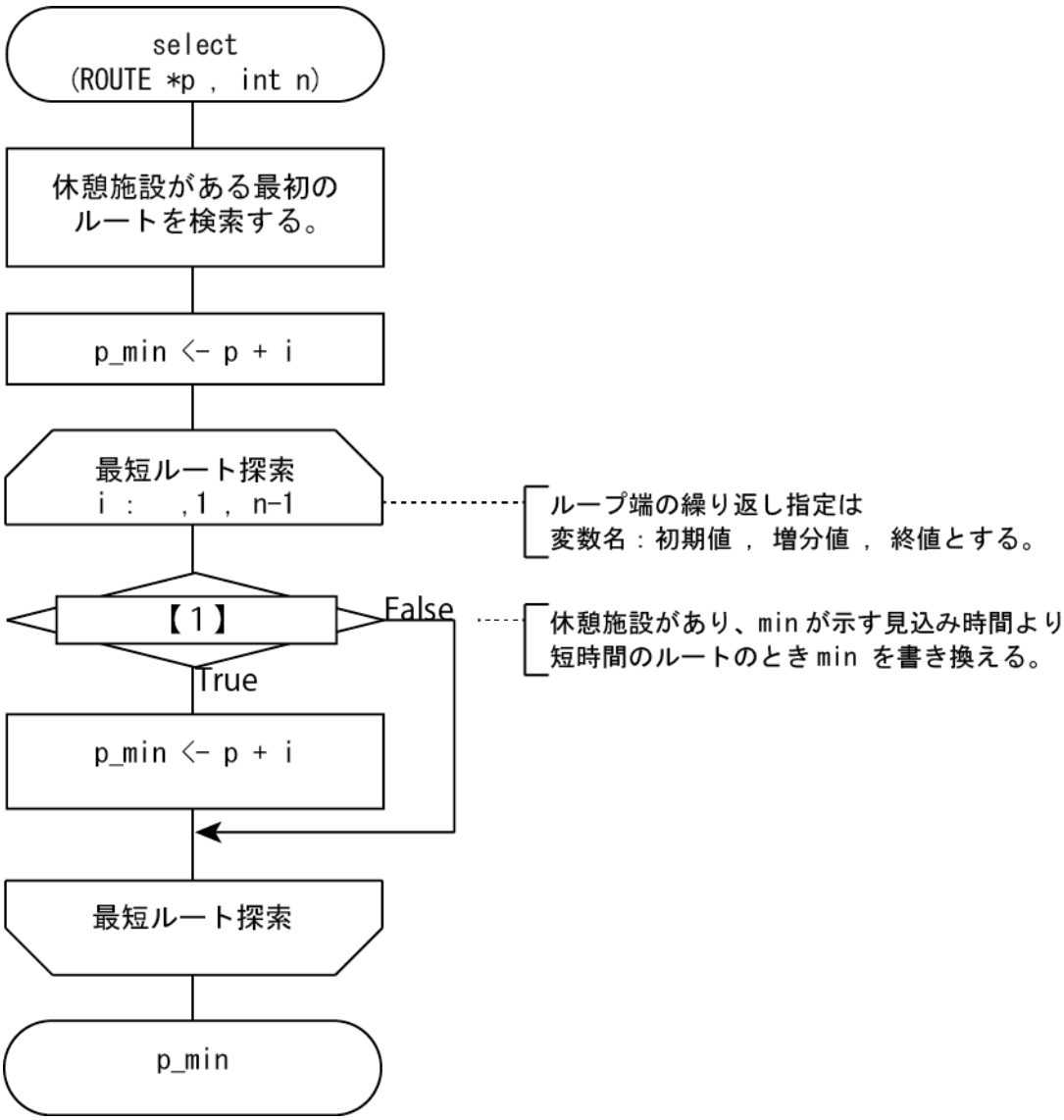
型	メンバ名	説明
char *	Name	ルート名
int	restPlace	休憩施設　0:なし 1:あり
double	requiredTime	見込み時間（単位：時間）

関数 select は、以下の仕様である。

表 9　関数 select の仕様

戻り値	関数の形式と機能
ROUTE *型	select(ROUTE　*p , int　n) p : ルート情報を収めた ROUTE 型配列へのポインタ n：ルートの個数

【流れ図】



以下のプログラムの空欄に入る適当なプログラム片を入力してプログラムを完成させなさい。

```
#include <stdio.h>
#define N 5

typedef struct root {
	char		*name;		//ルート名
	int		restPlace;	//休憩施設
	double		requiredTime;//見込み時間
}ROUTE;

//プロトタイプ宣言
ROUTE *select(ROUTE *p, int n);
```

```
int main(void) {
        ROUTE data[] = {
            {"Route-A" , 0 , 3.8},{"Route-B" , 1 , 4.5} ,{"Route-C" , 1 , 4.2},
            {"Route-D" , 0 , 3.7},{"Route-E" , 1 , 4.4}
        };

        ROUTE *selected;
        selected = select(data, N);
        printf("ルート：%s　見込み時間：%4.1f時間¥n", selected->name, selected->requiredTime);
}

ROUTE *select(ROUTE *p , int n) {
        int i;
        ROUTE *p_min;

        for (i = 0; i < n; i++) {
            if ((p + i)->restPlace == 1)break;
        }

        p_min = p+i;
        for (; i < n; i++) {
            if ((p + i)->restPlace == 1 && 【1】) {
                        p_min = p + i;
            }
        }

        return p_min;
}
```

【問題２】

図１のような迷路をスタート地点からゴール地点に最短で移動する経路をすべて列挙するプログラムである。■は壁であり、進むことができない。また、最短で移動するために、右と下にのみ移動できる。

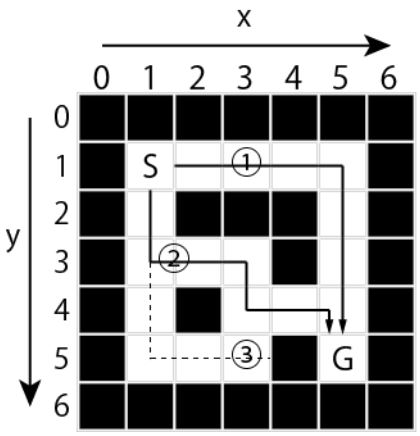


図 7　迷路の例

図１の例では、①と②はゴールにたどり着くことができるが、③は、途中で行き止まりになってしまい、ゴールにたどり着くことができない。

プログラムでは、壁の構成は、２次元配列 maze_data で与えられ、各地点は(y , x)で表現する。スタート地点 S は、(1,1)、ゴール地点 G は(5,5)である。また、地点を表す x と y の組は、構造体型を用いた POINT 型を利用する。POINT 型の構成は表１のとおりである。

通過した位置は、POINT 型の配列 route に記録していく。ゴールに到達したときは、配列 route の内容を先頭から順に表示する。その後、順に戻りながら、次のルートを探す。このとき、route の内容は１つずつ破棄する。行き止まりに阻まれたときも、同様に、route の内容を破棄しながら戻っていく。　なお、配列 route は十分な領域が確保されており、領域チェックは省略している。

グローバル領域に表２の配列を宣言して利用する。

表 10　POINT 型の構成

型	メンバ名	説明
int	X	横方向の添字
int	Y	縦方向の添字

表 11　グローバル領域に宣言されている配列

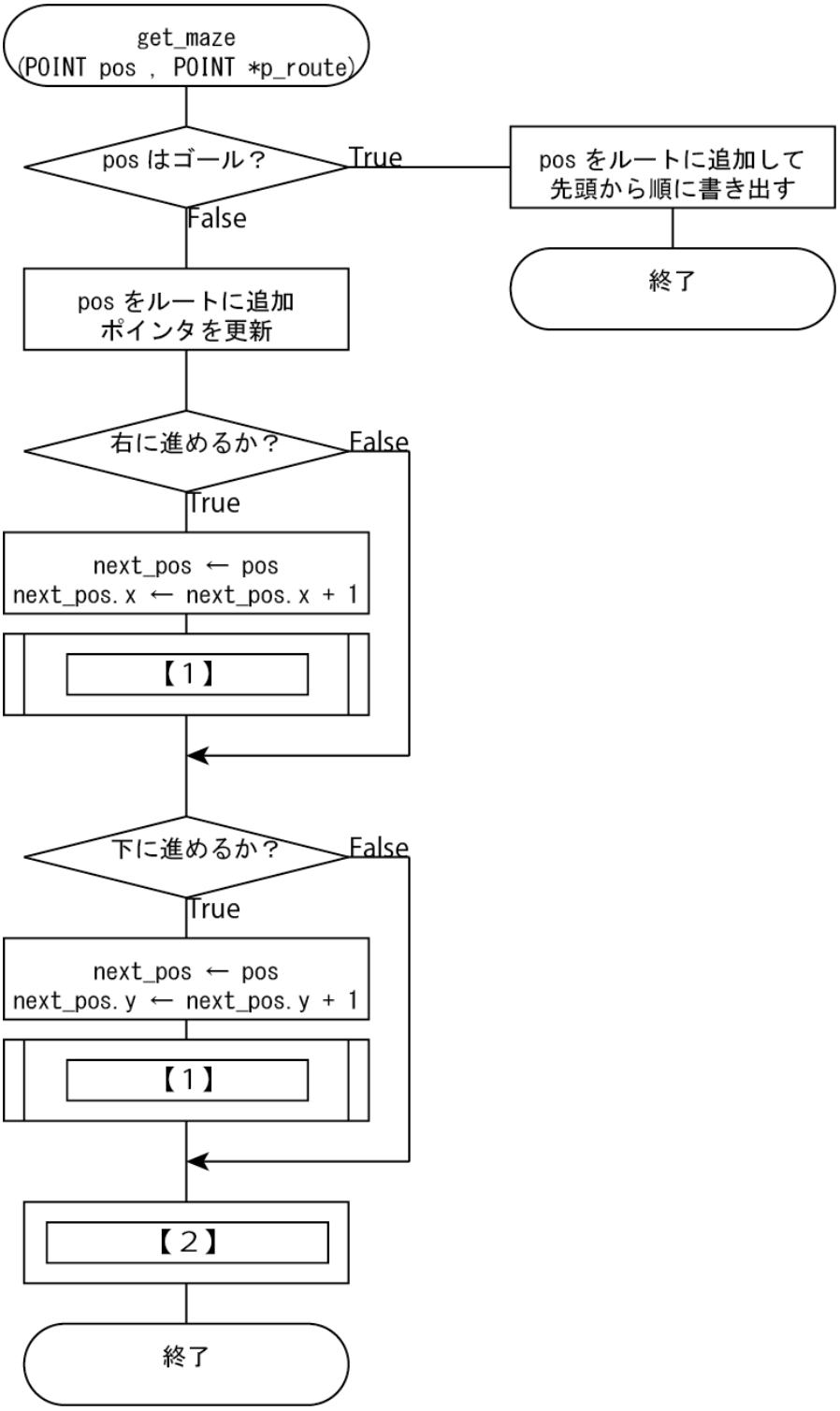
型	配列名	説明
int	maze_data[7][7]	壁の位置を格納する配列　0：通路　1：壁
POINT	route[30]	経路を格納する配列

プログラムで定義し、利用している関数の仕様は以下のとおりである。

表 12　定義している関数

戻り値	関数の形式と機能
void	get_maze(POINT pos, POINT *p_route) スタート位置からゴール位置までの経路を表示する。 pos ： 開始位置を指定する。 p_route ： 指す先に経路を格納する。
int	isGoal(POINT pos) pos がゴールであれば 1　ゴールでなければ 0 を返す。
void	print_route(POINT *p) 経路が格納されている配列 route の内容を p の前まで表示する。
int	check_go(POINT pos , int direction) pos の右または下の壁の状態を返す。 direction　1 ： 右　　2 ： 下

【流れ図】



以下のプログラムの空欄に入る適当なプログラム片を入力してプログラムを完成させなさい。

```
#include <stdio.h>


typedef struct {
        int x;
```

```
        int y;
}POINT;

int maze_data[7][7] = {
        { 1,1,1,1,1,1,1 },
        { 1,0,0,0,0,0,1 },
        { 1,0,1,1,1,0,1 },
        { 1,0,0,0,1,0,1 },
        { 1,0,1,0,0,0,1 },
        { 1,0,0,0,1,0,1 },
        { 1,1,1,1,1,1,1 }
};
POINT route[30];

void get_maze(POINT pos, POINT *p_route);
int isGoal(POINT pos);
void print_route(POINT *p);
int check_go(POINT pos, int direction);

int main(void) {
        POINT pos = { 1,1 };
        get_maze(pos, route);
}

void get_maze(POINT pos, POINT *p_route) {
        POINT next_pos;
        if (isGoal(pos)) {
            *p_route = pos;
            print_route(p_route+1);
            return;
        }

        *p_route = pos;
        p_route++;
        if (!check_go(pos, 1)) {
            next_pos = pos;
            next_pos.x++;
            【1】;
        }
```

```
        if (!check_go(pos, 2)) {
            next_pos = pos;
            next_pos.y++;
            【1】;
        }
        【2】;
}

int isGoal(POINT pos) {
        if (pos.x == 5 && pos.y == 5)
            return 1;
        else
            return 0;
}

void print_route(POINT *p) {
        int i;
        for (i = 0; &route[i] < p; i++) {
            printf("(%2d , %2d)", route[i].y, route[i].x);
        }
        printf("¥n");
}

int check_go(POINT pos , int direction) {
        int i, j;
        switch (direction) {
        case 1:
            i = pos.x + 1;
            j = pos.y;
            break;
        case 2:
            i = pos.x;
            j = pos.y + 1;
            break;
        }

        return maze_data[j][i];
}
```

【問題 3】

シャーレの中でバクテリアがどのように増殖するのかをシミュレートするプログラムである。シャーレを微小な空間に分け、その小さな一つひとつの空間にバクテリアが最大 1 匹存在できるものとする。図 1 では初期状態として、6 匹のバクテリアが存在している。

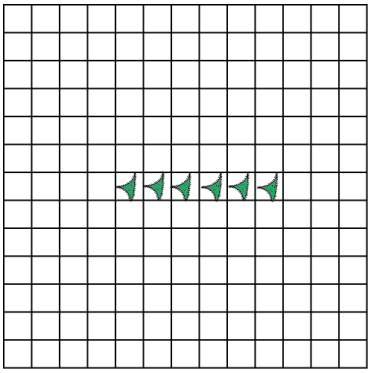


図 8　微小空間にいるバクテリア

バクテリアは、環境により次のように変化する。

① バクテリアは、図 2 に示す近傍 8 個のセルのうち 2 個または 3 個のセルにバクテリアが存在すれば、生き続けられる。しかし、2 個未満のときは、寂しくて死んでしまう。また、4 個以上の時は、過密すぎて死んでしまう。

② バクテリアが存在しないセルでは、図 3 に示す近傍 8 個のセルのうち 3 個のセルにバクテリアが要れば、分裂により新たに誕生することができる。

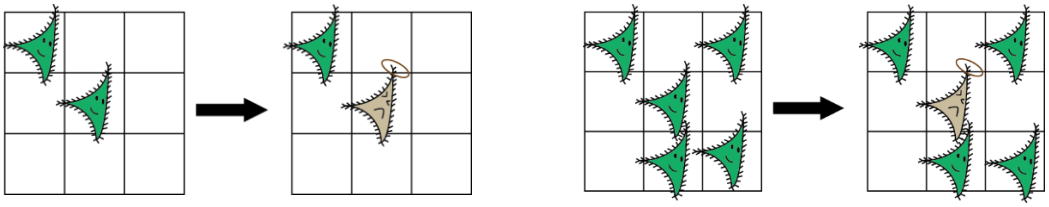


図 9　周りのバクテリアが少なくても多くても死んでしまう

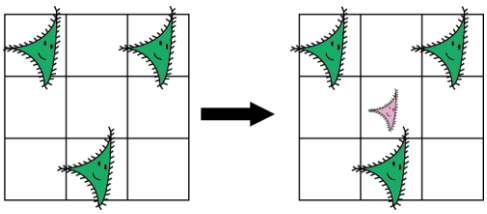


図 10　近傍が 3 匹であれば赤ちゃん誕生

以上のルールに従って、時間が進んだとき、バクテリアがどのように変化しているかをシミュレートする。ただし、領域の外側にはバクテリアは生存せず、生誕することもできないものとする。

プログラムでは、13×13 の領域に分けており、定数 TATE と YOKO で定義している。1 次元配列 cell に各微小空間のバクテリアの状態を記録しており、バクテリアが生きている状態を 1、いない状態を 0 とする。図 1 の初期状態を表したものが図 4 である。ここで、j 行 i 列の空間は

cell[j*YOKO+i]

で指定することができる。

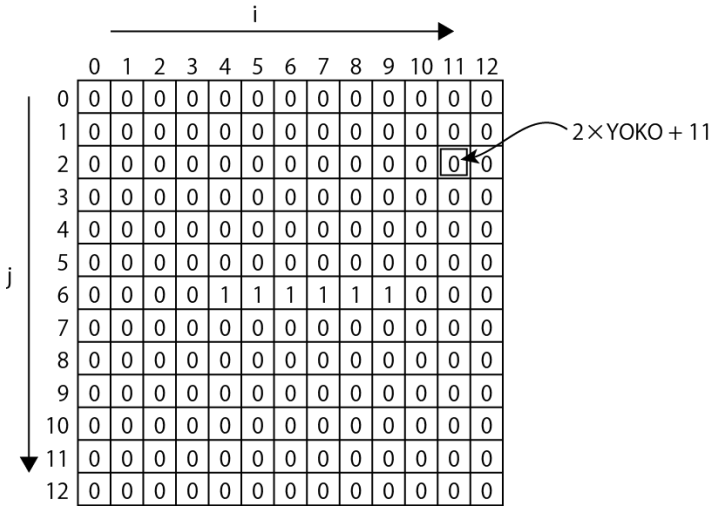


図 11　初期状態

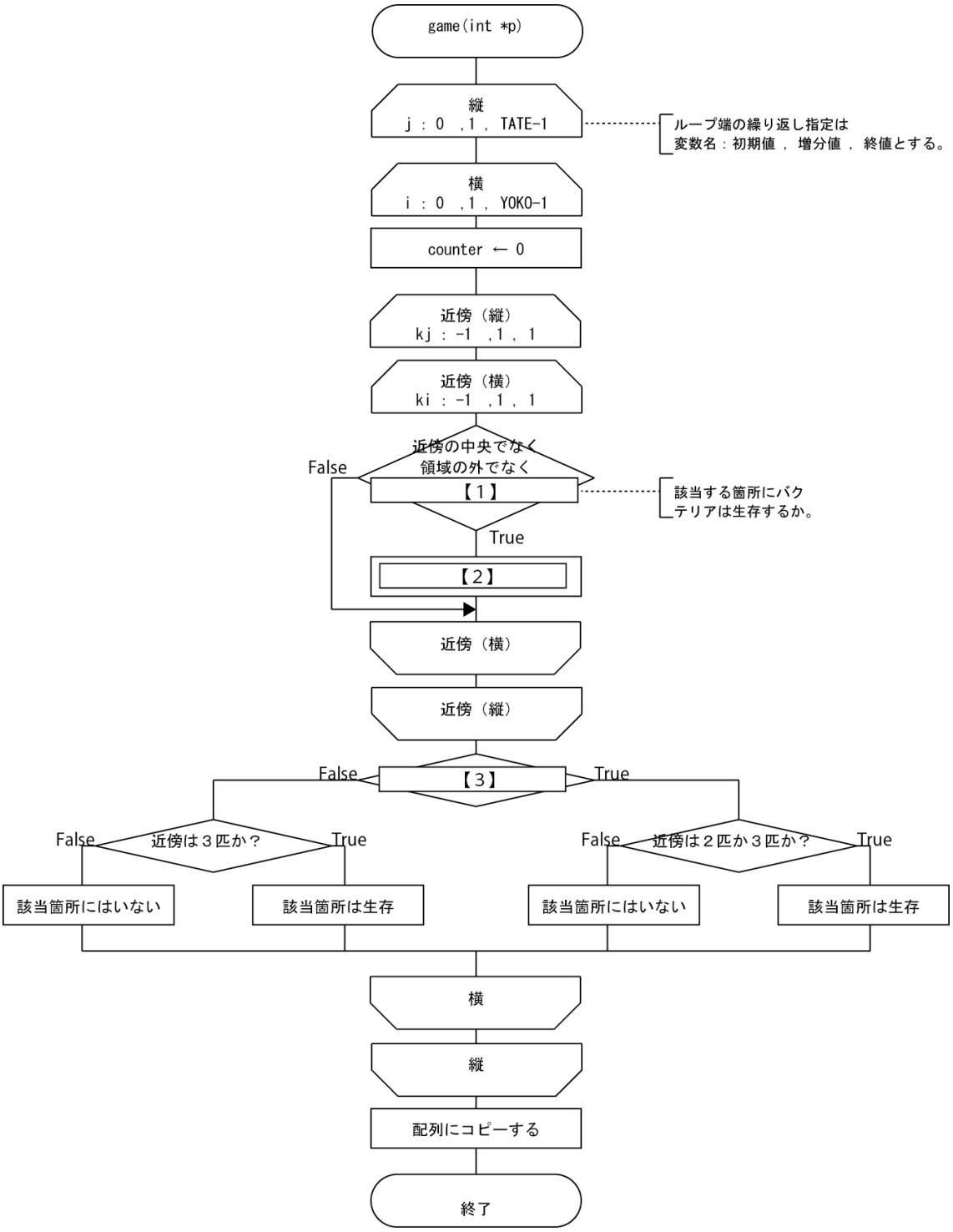
バクテリアの状態を１２回更新したときのバクテリアの状態を表示する。

プログラムで定義し、利用している関数の仕様は以下のとおりである。

表 13　定義している関数

戻り値	関数の形式と機能
void	init(int *p) 配列を初期化する。 p ： バクテリアの状態を表す配列を指すポインタ
void	disp(int *p) 配列の内容に従って画面に□または■を表示する。 p ： バクテリアの状態を表す配列を指すポインタ
void	game(int *p) バクテリアの状態から次の状態を求めて、配列を更新する。 p ： バクテリアの状態を表す配列を指すポインタ

【流れ図】



以下のプログラムの空欄に入る適当なプログラム片を入力してプログラムを完成させなさい。

```
#include <stdio.h>
#define TATE 13
#define YOKO 13

void init(int *p);
void disp(int *p);
void game(int *p);
```

```
int main(void) {
	int cell[TATE*YOKO];
	int i;

	init(cell);
	disp(cell);
	for (i = 0; i < 12; i++) game(cell);
	disp(cell);
}

void init(int *p) {
	int i, j;
	for (j = 0; j < TATE; j++) {
		for (i = 0; i < YOKO; i++) {
			*(p + j*YOKO + i) = 0;
		}
	}
	*(p + TATE / 2 * YOKO + YOKO / 2 - 2) = 1;
	*(p + TATE / 2 * YOKO + YOKO / 2 - 1) = 1;
	*(p + TATE / 2 * YOKO + YOKO / 2) = 1;
	*(p + TATE / 2 * YOKO + YOKO / 2 + 1) = 1;
	*(p + TATE / 2 * YOKO + YOKO / 2 + 2) = 1;
	*(p + TATE / 2 * YOKO + YOKO / 2 + 3) = 1;
}

void disp(int *p) {
	int i, j;

	for (j = 0; j < TATE; j++) {
		for (i = 0; i < YOKO; i++) {
			if (*(p + j*YOKO + i)) printf("■");
			else printf("□");
		}
		printf("¥n");
	}
}

void game(int *p) {
	int next[TATE*YOKO];
	int i, j, ki, kj;
	int counter;

	for (j = 0; j < TATE; j++) {
		for (i = 0; i < YOKO; i++) {
			counter = 0;
			for (kj = -1; kj <= 1; kj++) {
				for (ki = -1; ki <= 1; ki++) {
					if ( 【1】 ) {
```

```
                        if (j + kj >= 0 && j + kj < TATE && i + ki >= 0 && i + ki < YOKO) {
                            if (*(p + (j + kj)*YOKO + (i + ki)) == 1)  【2】  ;
                        }
                    }
                }
            }

            if (  【3】  ) {
                if (counter < 2 || counter >3)
                    next[j*YOKO + i] = 0;
                else
                    next[j*YOKO + i] = 1;
            }
            else {
                if (counter == 3)
                    next[j*YOKO + i] = 1;
                else
                    next[j*YOKO + i] = 0;
            }
        }
    }
    for (j = 0; j < TATE; j++) {
        for (i = 0; i < YOKO; i++) {
            *(p + j*YOKO + i) = next[j*YOKO + i];
        }
    }
}
```