解答

解答とは言っても一例にすぎません．若干異なっていたとしてもプログラムが動きさえすれば問題ありません．

・ソフトゼミ A　練習問題解答

1．円の直径の長さを引数として受け取って、その面積を返す関数
　ただし円周率は 3.14 としてよい。

　　double Circle(double d)

　引数が「直径」であることに注意しましょう．半径ではありません．計算した面積の値は実数になるので，関数の型（返り値の型）は double 型です．

```
//円の直径の値を受け取って円の面積の値を返す関数
#include<stdio.h>

double Circle(double d); //関数プロトタイプ宣言

int main(void){
   double d,s;

   scanf("%lf",&d);   //scanf で double 型を代入するときは%f ではなく%lf です
   s=Circle(d);       //関数 Circle で返ってきた円の面積の値を s に代入
   printf("直径%f の円の面積は%f です¥n",d,s);

   return 0;
}
double Circle(double d){
   double x,r;
   r=d/2;             //半径
   x=r*r*3.14;    //面積の計算
   return x;       //面積 x を返す
}
```

2. 整数の値を 3 つ、引数として受け取ってそのうち最大の値を返す関数
int max(int a,int b,int c)

3 つの中からの最大値，と値が少ないので，配列を使わずにごり押しで最大値を求めてみました. 配列を使う方法については今までの解答を見ればわかるはずです. その場合受け取った引数をいったん配列に入れる必要があります.

```
//整数を 3 つ受け取って、最大値を返す関数
#include<stdio.h>
int max(int a,int b,int c);
int main(void){
	int x,y,z;
	scanf("%d%d%d",&x,&y,&z); //x,y,z の値をキーボードから入力

	printf("最大の値は%d です¥n",max(x,y,z)); //最大値を表示
	return 0;
}
int max(int a,int b,int c){//配列を使ってもいいかも
	int m;	//最大値を保存するための変数
	//まずは a,b でどちらが大きいか. 大きいほうをとりあえず最大値とする
	if(a>=b){
		m=a;
	}
	else{
		m=b;
	}
	//c が m より大きいか否か
	if(m>=c){
		;  //m より小さいなら何もしない（；だけの文は空文と言います）
	}
	else{
		m=c; //c のほうが大きいなら c が最大値ということになる
	}

	return m;
}
```

3．整数の値を 2 つ(x,y)、引数として受け取って x の y 乗を返す関数。ただし y は 0 以上の整数とします。

int power(int x,int y)

x の y 乗とは，x を y 回だけ掛けることです．このことを for 文などのループ文で実現してやればいいわけです．ちなみに四則演算子（+ , - , * , / ）のような演算子は，べき乗には定義されてません．

```
//x,y の値を受け取って x の y 乗を返す関数
#include<stdio.h>
int power(int x,int y);
int main(void){
	int x,y;
	printf("x の y 乗。x と y を入力してください¥n");
	scanf("%d%d",&x,&y);
	printf("%d^%d=%d¥n",x,y,power(x,y));
	return 0;
}
int power(int x,int y){
	int k=1,i;

	for(i=0;i<y;i++){
		k=k*x;
	}

	return k;
}
```

4．整数の値を 4 つ引数として受け取って大きい順に出力する関数

void order(int a,int b,int c,int d)

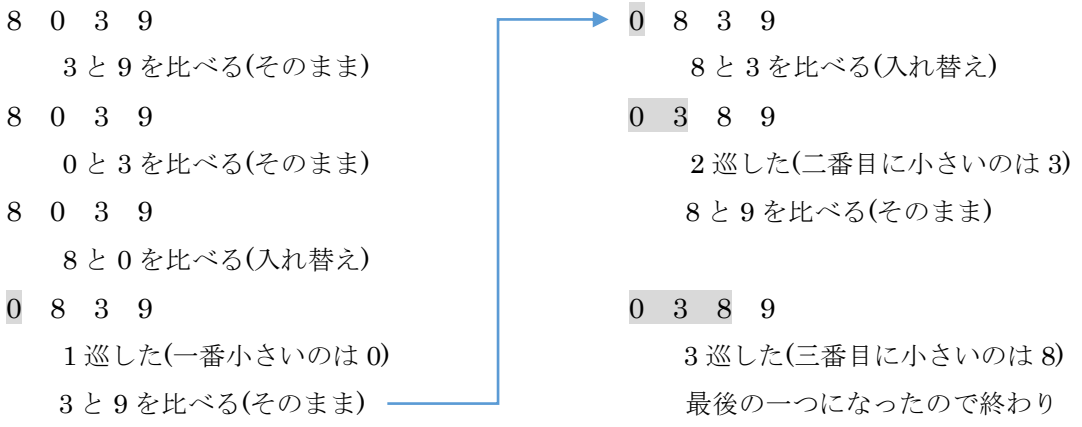
大きさを比べる関数で，引数が 4 つ以上になると問 2 のようにごり押しするのは大変です．そこで関数側で配列を用意して，引数をそれぞれ代入し，for 文を使って大きい順に並び変えています．

```
//整数を 4 つ受け取って大きい順に並び替えて表示
#include<stdio.h>
void order(int a,int b,int c,int d);   //関数プロトタイプ宣言
int main(void){
	int w,x,y,z;
	printf("整数を 4 つ入力してください¥n");
	scanf("%d%d%d%d",&w,&x,&y,&z);   //整数を 4 つ入力
	printf("¥n");                  //見やすくするため 1 行開ける
	order(w,x,y,z);                //関数呼び出し

	return 0;
}
void order(int a,int b,int c,int d){
	int sort[4],i,j,k;
	sort[0]=a;
	sort[1]=b;
	sort[2]=c;
	sort[3]=d;
	for(i=0;i<3;i++){ //大きい順に並び替える
		for(j=3;j>i;j--){
			if(sort[j-1]>sort[j]){
				k=sort[j];
				sort[j]=sort[j-1];
				sort[j-1]=k;
			}
		}
	}
	//並べ替え終わったら表示
	for(i=3;i>=0;i--){
		printf("%d¥n",sort[i]);
	}
}
```

大きい順への並び替えですが，何をしているのかというと，配列の隣同士を右側から比べて，右側のがちいさかったら入れ替える．という操作を繰り返します．1 巡すると一番左に一番小さい数が入っていることになるので，次は一番左の数を除いて同じ操作をします．こ

れを最後の 1 つになるまで続けています.
例:



・ソフトゼミ∀　解答

練習問題

1. その関数が呼び出された回数を随時出力する void 型の関数 times を作成してください。

static を使います. 初期化は一回だけ行われるので, times 内の呼び出された回数を記録する変数 n の値は保持されます.

```
#include<stdio.h>
void times(void){
        static int n=1;
        printf("%d 回目の呼び出し¥n",n);
        n++;
}

int main(void){
        int n;
        int i;

        printf("何回 times を呼び出しますか: ");
        scanf("%d",&n);
```

```
        for(i=0;i<n;i++){
                times();
        }
        return 0;

}
```

2.　正の整数を入力したときに、その数の階乗を返す int 型の関数 factorial を作成してください。

階乗の定義を数学的に示すと以下のようになります.

$$n! = \begin{cases} 1 & (n = 0) \\ n * (n-1)! & (n > 0) \end{cases}$$

この定義に従って, 再帰関数をつくればいいのです. 入力される数は負にならないと仮定します.

```
#include<stdio.h>
int factorial(int n){//n の階乗を返す関数
        if(n==0){
                return 1;
        }
        else{
                return (n*factorial(n-1));
        }
}

int main(void){
        int n;
        scanf("%d",&n);//0 もしくは正の整数を入力

        printf("%d! = %d¥n",n,factorial(n));

        return 0;
}
```

応用問題

1. みみみ君の main 文がとても汚いです。main 文のなかで以下の作業をしています。

・入力された数字を 3 倍して 5 足す

・その計算あとの値から降順に 0 まで数字を出力

・また計算結果の回数分「がんばるびぃ」と出力

面倒くさがりのみみみ君の汚い main 文をきれいにするため関数を用意してあげましょう。用意する関数は 3 つです。（るびぃちゃん問題）

```
#include<stdio.h>

int cal(int n){
        n = n * 3 + 5;
        return n;
}
//計算する関数

void turn(int n){
        int i;
        for(i = n;i >= 0;i--){
                printf("%d ",i);
        }
        printf("¥n");
}
//降順に出力する関数
void ruby(int n){
        int i;
        for(i = 0;i < n;i++){
                printf("がんばるびぃ¥n");
        }
}
//がんばるびぃを出力する関数

int main(void){
        int num;
        scanf("%d",&num);          //入力
        num = cal(num);            //計算関数の呼び出し
        turn(num);                          //降順関数の呼び出し
        ruby(num);                          //るびぃちゃん
        return 0;
}
```

main 文がすっきりしました．このように関数を使うと，まとまった操作ごとに分けられるので，プログラムが見やすくなります．

2．ざわちん君の x 座標、y 座標、腕の長さ、タカヤ君の x 座標、y 座標、腕の長さ、深山君の x 座標、y 座標の順に入力します（すべて整数値）。距離を考えるときは三平方の定理を使っています．（距離と腕の長さの 2 乗同士で比べます）

```
#include<stdio.h>
struct person {//構造体
	int x, y, arm;
};

void lariat(struct person a, struct person b, struct person c); //プロトタイプ宣言

int main(void) {
	struct person ozawa, takaya, miyama;
	//8 つの値を入力
	scanf("%d%d%d%d%d%d%d%d", &ozawa.x, &ozawa.y, &ozawa.arm,
&takaya.x, &takaya.y, &takaya.arm, &miyama.x, &miyama.y);

	lariat(ozawa, takaya, miyama);
	return 0;
}

//判定
void lariat(struct person a, struct person b, struct person c) {
	//ざわちん君の腕の長さがざわちん君とタカヤ君の距離より長い　かつ
	//タカヤ君の腕の長さがざわちん君とタカヤ君の距離より長い　かつ
	//ざわちん君の腕の長さがざわちん君と深山君の距離より長い　または
	//タカヤ君の腕の長さがタカヤ君と深山君の距離より長い　とき
	if(	(a.x - b.x)*(a.x - b.x)+(a.y - b.y)*(a.y - b.y) < =a.arm * a.arm &&

		(a.x - b.x)*(a.x - b.x)+(a.y - b.y)*(a.y - b.y) <= b.arm * b.arm &&

		( (a.x - c.x)*(a.x - c.x)+(a.y - c.y)*(a.y - c.y) < =a.arm * a.arm ||
```

```
            (b.x - c.x)*(b.x - c.x)+(b.y - c.y)*(b.y - c.y) <= b.arm * b.arm )   )
{
        puts("dead");                   //全員死ぬ
                                        //puts は文字列を出力して改行する標準関数
}
//ざわちん君の腕の長さがざわちん君とタカヤ君の距離より長い　かつ
//タカヤ君の腕の長さがざわちん君とタカヤ君の距離より長い　かつ
//ざわちん君の腕の長さがざわちん君と深山君の距離より短い　かつ
//タカヤ君の腕の長さがタカヤ君と深山君の距離より短い　とき
else if(   (a.x - b.x)*(a.x - b.x)+(a.y - b.y)*(a.y - b.y) < =a.arm * a.arm &&

        (a.x - b.x)*(a.x - b.x)+(a.y - b.y)*(a.y - b.y) < =b.arm * b.arm &&

        (a.x - c.x)*(a.x - c.x)+(a.y - c.y)*(a.y - c.y) > a.arm * a.arm &&

          (b.x - c.x)*(b.x - c.x)+(b.y - c.y)*(b.y - c.y) > b.arm * b.arm   )
{
        puts("victory");                //2 人が自滅する
}

//ざわちん君の腕の長さがざわちん君と深山君の距離より短い　かつ
//タカヤ君の腕の長さがタカヤ君と深山君の距離より短い　とき
else if( (a.x - c.x)*(a.x - c.x)+(a.y - c.y)*(a.y - c.y) > a.arm * a.arm &&

                    (b.x - c.x)*(b.x - c.x)+(b.y - c.y)*(b.y - c.y) > b.arm * b.arm ) {

        puts("win");                            //深山君は死なない
}
else {
        //ざわちん君の腕の長さがざわちん君と深山君の距離より長い　とき
        if((a.x - c.x)*(a.x - c.x)+(a.y - c.y)*(a.y - c.y) < =a.arm * a.arm) {

                puts("ざわちん"); //深山君はざわちん君に殺される
        }
        //タカヤ君の腕の長さがタカヤ君と深山君の距離より長い　とき
```

```
                if((b.x - c.x)*(b.x - c.x)+(b.y - c.y)*(b.y - c.y) <= b.arm * b.arm) {

                        puts("タカヤ");                 //深山君はタカヤ君に殺される
                }
        }
}
```

3. 二進数問題

ある整数(負でない)N を次のように分解したとします.

$$N = n_m 2^m + n_{m-1} 2^{m-1} + \cdots + n_2 2^2 + n_1 2^1 + n_0 2^0$$

ただし m, n_m は負でない整数とします. このとき式を次のように変形します.

$$N = 2(n_m 2^{m-1} + n_{m-1} 2^{m-2} + \cdots + n_2 2^1 + n_1 2^0) + n_0 2^0$$

このとき, $n_0 2^0 = n_0$ は N を 2 で割った余り, ( )内は N を 2 で割った商となります. また n_0 は N の 2 進数表現の 1 の位(2^0 の位)の数字と同じです. ( )内を 2 で割ればその余りは n_1, すなわち N の 2 進数表現の 2 の位(2^1 の位)の数字です.

このことを再帰関数で繰り返します.

```
#include<stdio.h>

void print_binary(int n){
        if(n == 0){                     //0 になったら戻る
                return;
        }
        else{
                print_binary(n / 2);        //どんどん割る
                printf("%d",n % 2);                 //余りを出力
        }
}

int main(void){
        int num;
        scanf("%d",&num);                   //入力
        print_binary(num);                  //2 進数関数の呼び出し
        printf("¥n");                       //見やすく改行
        return 0;
}
```