Problem 1: Short Questions [2% each box; 30% total]

Write clearly the output produced by each of the following segments of code.

Problem 1 Con'd.		Answers

a.	`int x = (int)(60.40 + 4.70);`
	`double y = (int)60.40 + (int)4.70;`
	`printf("%c%5.1f", x, y);`	

b.	`printf("%.2f\n", 10 / 3 * 3.0 );` `printf("%.2f",  1.0 / 3 * 3 );`	

c.	`int x = -10, y = 0, z = -1;`
	`if (x < z < y)`
	`  printf("%d %d", 10 + x, y != y != y);`
	`else`
	`  printf("%d %d", 20 + x, z == z == z);`	

d.	`printf("%d %c", 'D' - 'a', '2' + 2);`	
e.	`char S[9] = "/\n", T[9] = {'0', '\0', 'B'};` `printf("%d %d", strlen(S), strlen(T));`	
f.	`printf("%d\n", strcmp("a", "B") > 0);` `printf("%d\n", strcmp(" A", "A ") > 0);`	
g.	`int A[10] = { 1, 3, 5, 7, 9 };` `printf("%d\n%d\n", A[ A[0] ], A[0] + A[5]);`	

h.	`int x = 5, y = 7, tmp;`
	`if (x < y) {`
	`  tmp = x;`
	`  x = y;`
	`  y = tmp;`
	`}`
	`printf("%d %d", x, y);`	

Problem 1 Con'd.		Answers

i.	`void foo(int i, int sum) {`
	`  sum += i;`
	`  i++;`
	`}`
	
	`int main() {`
	`  int i, sum = 10;`
	
	`  for ( i = 5; i > 2; i -= 2 )`
	`  foo(i, sum);`
	
	`  printf("%d\n", sum);`
	`  printf("%d\n", i);`
	`  return 0 ;`
	`}`	

j.	`void bar(int A[], int B) {`
	`  A[1] += 1;`
	`  B *= 2;`
	`}`
	
	`int main(void) {`
	`  int A[5] = { 0 }, B[] = { 3, 4, 5, 6, 7 };`
	`  bar(B, A[1]);`
	`  printf("%d\n", A[1]);`
	`  printf("%d\n", B[1]);`
	`  return 0;`
	`}`	

k.	`int recursion(int x, int n) {`
	`  int sum = x;`
	`  if (n < 0) return -sum;`
	`  if (n > 0) {`
	`  n--;`
	`  sum = recursion(x-1, n-1) + x;`
	`  }`
	`  return sum;`
	`}`
	
	`int main(void) {`
	`  printf("%d\n", recursion( 2, 1 ) );`
	`  printf("%d\n", recursion( 3, 2 ) );`
	`  return 0;`
	`}`	

l.	`int x1 = 0, y1 = 3;`
	`int x2 = 4, y2 = 0;`
	
	`printf("%d\n",  x1 * y2 - x2 * y1);`
	`printf("%d", (x1-x2)*(x1-x2)/(y1-y2)*(y1-y2));`	

Problem 1 Con'd.		Answers

m.	`int i = 1, j = 2, k = 3;`
	`int *p = &i, *q = &j;`
	`*p = k;`
	`j = i;`
	`printf("%d\n%d", i, *q);`	

n.	`char x = 'X', y = 'Y', k = 'Z';`
	`char *X = &x, *Y = &y;`
	`printf("%d\n%d", x - *Y, k - *X);`	

o.	`char name[] = "The CUHK";`
	`int i = 4;`
	`char *p = &name[strlen(name)];`
	`printf("%d\n%d", name[i-1], *p);`	

Problem 2: Branching [6%]

Complete the program below to simulate a login system. The user inputs two integers: user ID and password. Then the program judges whether they are valid based on the user account table below. After that, the program terminates regardless of the outcome; see three sample runs on the right.

user ID	password
1	12
2	3

Three Sample Runs *(user input underlined)*

1 12
OK

2 12
Wrong Password

3 12
Invalid User ID

- If the input user ID exists, print **OK** or **Wrong Password** according to the correctness of the password;
- otherwise, print **Invalid User ID**.

```
#include <stdio.h>
int main(void) {
    int id, key;
    // Answer:









    return 0;
}
```

Problem 3: Array Processing [14%]

A sound is often stored in a digital format represented as a series of floating numbers (called ***samples***), e.g.,

```
double input[8000] = {-0.088, -0.083, 0.055, 0.112, 0.135, 0.095, 0.083, 0.097,
                      -0.089, ..., -0.007, 0.013, -0.057, -0.019, 0.079};
```

where `-1.0` ≤ sample value ≤ `-1.0`. By manipulating these samples, we can achieve many interesting effects. Write C program fragments to achieve the following sound effects on **`input[]`** and store the output in **`output[]`**. In your answers, you must use the variables **`numOfSamples`** and **`outputSize`** instead of using hard-coded constants, e.g., 8000.

a) Doubling the speed and pitch:
- copy the input array elements while skipping half of the elements alternatively, i.e. keep only the 0th, 2nd, 4th, … samples. (3%)

```
double output[4000] = {0};
int    noOfSamples  = 8000;
int    outputSize   = 4000;
// Answer:
```

Example Run

input[]

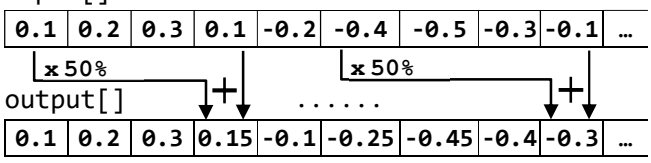
0.1	0.2	0.3	0.1	-0.2	-0.4	-0.6	-0.3	-0.1	…

output[]

0.1	0.3	-0.2	-0.6	-0.1	…	…	…	…	…

b) Adding an echo:
1. ask the user for the number of samples before an echo (e.g., ***n***);
2. for every array element, on top of the original value, add 0.5 times the value of the array element ***n*** samples (if any) before it.
3. beware of array boundaries, i.e., the first ***n*** output elements shall stay the same. (5%)

Example Run with n=3




```
double output[8000] = { 0.0 } ;
int    numOfSamples = 8000 ;
int    outputSize   = 8000 ;
printf("How many samples before echo?\n");

// Answer:
```

Problem 4 Con'd.

c) Normalizing the sound: (6%)

1. find the lowest and highest sample values in **`input`**
2. determine the maximum absolute value ***x = max(\|lowest\| , \|highest\|)***
3. if ***x*** is non-zero, divide all samples by ***x*** and store results in **`output`**; if ***x*** is zero, keep zeros in output.

```
double output[8000] = {0.0};
int    numOfSamples = 8000 ;
int    outputSize   = 8000 ;
double x , lowest , highest ;

// Answer:
```

Example Run: *lowest=-0.5, highest=0.3, x=0.5*

input[] **highest** **lowest**

0.1	0.2	*0.3*	0.1	-0.2	-0.4	*-0.5*	-0.3	-0.1	…

output[]

0.2	0.4	0.6	0.2	-0.4	-0.8	-1	-0.6	-0.2	…

```
// print the results at last

printf( "lowest = %f, highest = %f\n" , lowest , highest );

printf( "absolute max x = %f\n" , x );
```

Problem 4: File IO [8%]

Complete the part of the following C program marked by **TODO**.

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

int main( void ) {
    int gridSize ;
    int numberGrid[10][10];
    int row , col ;
    FILE * fptr ;

    /* TODO */

    return 0 ;
}
```

Read a Number Grid from the File NumberGrid.txt (8%)

The program reads a number grid from the file named **NumberGrid.txt** in the following format. The first two numbers indicate the number of rows and columns in the grid, denoted as **row** and **col**, which are between 2 and 10, inclusively. The subsequent **row** x **col** numbers indicate the numbers in the grid, which will be read into the 2D array **numberGrid** in the program. An example is shown on the right.

Example file NumberGrid.txt
2 3 5 5 8 -5 1000 0

If the file cannot be opened successfully, the program will terminate immediately. But if the file can be opened, you can assume that the file format is correct and no validation is required.

Answer:

Problem 5: Function [10%]

a) Write a function **`extract`** that accepts an integer **n** as its argument and returns a value computed from **n**. Let X and Y be the tens digit and unit digit of **n**, respectively. Your function should compute and return $X^2 - Y^2$. For example, if **n** is 53, your function should return $5^2 - 3^2 = 16$. You may assume the input argument is in the range from 0 to 99. (3%)

Answer:

b) Write a function **`dump`** that accepts an integer **`diff`** as its argument and prints all integers in the range from 0 to 99 in ascending order, such that the *difference* of the square of the tens digit and the square of the unit digit of the integer equals to **`diff`** or **`-diff`**. The function should also return the number of integers printed. (7%)

For example, calling **`dump(9);`** will print "**`3 30 45 54`**"
while calling **`dump(10);`** will print nothing.

Answer:

Problem 6: Structure [10%]

Consider the following type definitions for representing a 2D point and a triangle, respectively:

```
typedef struct {
    double x , y ;  // 2D coordinates of a point
} Point ;

typedef struct {
    Point   p1 , p2 , p3 ;  // 3 points in a triangle
} Triangle ;
```

a) Declare an array named **`triangles`** with three elements of type **`Triangle`**, and write a program fragment to ask for user inputs for the first element of array **`triangles`**. A sample run is shown below with ***user inputs underlined***. (3%)

```
Input p1: 0.0 2.3
Input p2: -1.5 4.6
Input p3: -5.0 2.3
```

Answer:

You may use the **`sqrt()`** function defined in **`<math.h>`** to complete the following task:

b) Write function **`double calDis(p1,p2)`** to compute and return the distance between the two **`Point`** parameters. Distance of two points (x_1, y_1), (x_2, y_2) is given by (2%)

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Answer:

Problem 6 con'd.

c) Complete the following function for finding and returning the area of a **`Triangle t`**. Heron's formula states that the area of a triangle whose sides have lengths *a, b* and *c* is

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}\ ,$$

where *s* is the semiperimeter of the triangle; that is, $s = \frac{a+b+c}{2}$.

You may refer to definitions and use function defined in previous parts. (5%)

```
double area( Triangle t ) {
     // Answer:




}
```

Problem 7: Character and String Processing [12% ESTR only]

Write a function that reads an array of digits (as a C-string) and decode it into an output string. For example, if the input is "72101108108111326710897115115533", the output string is "Hello Class!" because from the ASCII table, we know that 72 is H, 101 is e, 108 is l, 111 is o, 32 is a space, etc.

Note that we assume that the output string contains only the alphabets (lowercase or uppercase), space (ASCII code 32), and the following five basic punctuations: period (.), question mark (?), the exclamation mark (!), single quote ('), and double quote ("), so that we can decode the input.

Below is the skeleton of the overall program.

```
#include <stdio.h>

...... // your two functions to be put here

void main( void )
{
    char input[] = "72101108108111326710897115115533" ;
    char output[1024] ;
    decode( input , strlen(input) , output , 1024 );
    printf( "input  = [%s]\n" , input  );
    printf( "output = [%s]\n" , output );
}
```

[3%] Write function **is_valid_char**, which returns 1 if its argument **ch** is an alphabet, a space, or one of the five punctuations listed earlier in the question; otherwise, it should return zero.

Note that in your code below, you SHOULD NOT write down explicit ASCII numbers, e.g., 32 and 97. Marks will be deducted for each explicit ASCII number written in your code.

```
int is_valid_char( char ch )
{
    // Answer:









}
```

Problem 7 con'd.

[9%] Write function **decode**, which reads an array of digits (i.e., **digits**) as a string and the number of digits in the array (i.e., **number_digits**), and then outputs the decoded string in **output_str**. Note that we ignore the digits remained in **digits** when **output_str** is full.

```
void decode( const char *digits , int number_digits   ,
             char *output_str   , int output_max_size )
{
    // Answer:









}
```

Problem 8: Permutation – K Sum [10% ESTR only]

[10%]

Given a list of N integers, find all subsets of K integers such that their sum equals a target integer T, where K <= N.

For example, considering the list of integers [-1 2 1 4 -3] stored in **input_data** inside the main function below, if K is 3 and target T is 2, the program should be able to find two sets of three integers, i.e., [-1 1 2] and [4 1 -3], each with a sum equal to 2, and report that there are two cases in total.

```
#include <stdio.h>

...... // your function on next page to be put here

void main( void )
{
    int N , K , T , total_found ;
    int output_numbers[10] ;

    int input_data[] = { -1 , 4 , 1 , 2 , -3 };
    N = 5 ;
    K = 3 ;
    T = 2 ;

    total_found = print_Ksum( input_data , 0 , N-1 , K , T ,
                              output_numbers , K );

    printf( "Total cases found = %d\n" , total_found );
}
```

Note: we have provided to you the function declaration of **print_Ksum**, where **begin** and **end** together (see the next page) mark the range of values in **input_data** to be explored in the function call. Moreover, you should not create and use any global variable in the program.

Problem 8 con'd.

```
// You may follow the print_Ksum declaration below to complete your code, or
// design your own print_Ksum; however, if you design your own print_Ksum, you
// have to write down also how to call your function in main()

int print_Ksum( int * input_data , int begin , int end ,
                int remainK , int T , int * output , int K )
{
     // Answer:









}
```

<< Appendix >>

Partial List of C Operators in Decreasing Precedence	Associativity
() [] . -> ++ (postfix) -- (postfix)	left-to-right
+ (unary) - (unary) ++ (prefix) -- (prefix) ! * (unary)	right-to-left
* / %	left-to-right
+ (addition) - (subtraction)	left-to-right
< <= > >=	left-to-right
== !=	left-to-right
&	left-to-right
&&	left-to-right
||	left-to-right
= += -= *= /= etc.	right-to-left
, (comma operator)	left-to-right

ASCII Table							
0 NUL	1 SOH	2 STX	3 ETX	4 EOT	5 ENQ	6 ACK	7 BEL
8 BS	9 HT	10 NL	11 VT	12 NP	13 CR	14 SO	15 SI
16 DLE	17 DC1	18 DC2	19 DC3	20 DC4	21 NAK	22 SYN	23 ETB
24 CAN	25 EM	26 SUB	27 ESC	28 FS	29 GS	30 RS	31 US
32 SP	33 !	34 "	35 #	36 $	37 %	38 &	39 '
40 (	41)	42 *	43 +	44 ,	45 -	46 .	47 /
48 0	49 1	50 2	51 3	52 4	53 5	54 6	55 7
56 8	57 9	58 :	59 ;	60 <	61 =	62 >	63 ?
64 @	65 A	66 B	67 C	68 D	69 E	70 F	71 G
72 H	73 I	74 J	75 K	76 L	77 M	78 N	79 O
80 P	81 Q	82 R	83 S	84 T	85 U	86 V	87 W
88 X	89 Y	90 Z	91 [	92 \	93]	94 ^	95 _
96 `	97 a	98 b	99 c	100 d	101 e	102 f	103 g
104 h	105 i	106 j	107 k	108 l	109 m	110 n	111 o
112 p	113 q	114 r	115 s	116 t	117 u	118 v	119 w
120 x	121 y	122 z	123 {	124 |	125 }	126 ~	127 DEL

<< END OF PAPER >>