香 港 中 文 大 學
The Chinese University of Hong Kong

Course Examination 1st Term, 2015 – 2016

Course Code & Title : ENGG 1110 A-F Problem Solving by Programming

Time allowed : 2 hours

Student I.D. No. : ..... Seat No. : .....

<< The Cover Page >>

Please read the following instructions carefully.

- There are **six** questions and you are required to answer **ALL** questions. Full Score is 100.
- Please write all your answers in **the space provided** of this question paper.
- Please write neatly.
- Remove all drafts. Otherwise, the drafts would be considered as a part of your answer.
- A list of C operators as well as the ASCII table are provided on the last page of this question paper.
- For all problems, you may assume `int` is 4 bytes.
- For all problems, you may assume all input values (from keyboard or from file) are valid.

Problem 1: Short Questions [2% each; 20% total]

For each of the following segments of code, show the output produced by the code.

		Answers

a.	`int x = (int) -3.6;`
	`double y = 4/10;`
	
	`printf("%d %.2f", x, y);`	

b.	`// Note: Indicate whether the output is`
	`// 0, +ve, or -ve`
	`printf("%d ", strcmp("ABC", "A B C"));`
	`printf("%d", strcmp("", "0"));`	

c.	`// Note: There is an ASCII table on the last page` `printf("%d %c", '1' + '2', 'a' + ('M' - 'A'));`	

d.	`char s1[100] = "ABC", s2[100] = "C\\D";`
	`s1[1] = '\0';`
	`printf("%s %d", s1, strlen(s2));`	

e.	// Note: There is a precedence table of C operators on the last page
	`int x = 5;`
	`printf("%.2f ", (double)x / 2 / 10);`
	`printf("%.2f", (double)(x / 2 / 10));`	

f.	// Note: There is a precedence table of C operators on the last page
	`int x = 2, y = 0, z = 0;`
	`printf("%d %d", x == y == z, x > y > z);`	

g.	`int i, count = 0;`
	`for (i = -9; i <= 1; i+=2)`
	`  count++;`
	
	`printf("%d %d", count, i);`	

h.	`void foo(int X[], int Y[]) {`
	`  X[0] = Y[0];`
	`}`
	
	`int main(void) {`
	`  int A[5] = { 4 }, B[3] = { 2, 7, 9 };`
	
	`  foo(A, B);`
	`  printf("%d %d", A[0], A[3]);`
	
	`  return 0;`
	`}`	

i.

```
void foo(int *x, int *y) {
  *x *= *y;
}

int main(void) {
    int x = 10, y = 5;

    foo(&y, &x);
    printf("%d %d", x, y);

    return 0;
}
```

j.

```
// Read the code carefully
int FOO = 0, bar = 0, i;

for (i = 0; i < 4; i++) {
  int FOO = i;

  if (i % 2 == 0)
    FOO += bar;
          bar++;
}

printf("%d %d", FOO, bar);
```

Problem 2: If-else Statements [10%]

Write a program fragment to **read three integers** from the user and then **print the maximum, the middle, and the minimum values** as shown in the sample output. You can assume that the three input integers are distinct. A sample run is shown as follows (italic, bold, underlined characters are inputs from the user):

```
5 7 2
maximum = 7
middle = 5
minimum = 2
```

Let us begin the program with:

```
int x, y, z;

// Start your solution below
// Hint: if-then-else is sufficient to solve this problem
```

Problem 3: 2D Arrays [20%]

Suppose now we have a game of Tic-Tac-Toe and you want to write code to analyse the final game board. The move of two players are represented by lower-case letters 'x' and 'o' respectively. You can assume the game board is always **fully filled** with either one player or none of the players winning.

a) [5%] Declare and initialize a character array, namely `b[3][3]`, to represent the following final game board:

x	o	o
o	x	x
o	o	x

Answer:

b) [15%] Write **a code segment** that analyses the array `b[3][3]` to determine the winner of the game. There is only one winner or no winners.

- If player '`x`' wins, print out "`x won!`"
- Else if player '`o`' wins, print out "`o won!`"
- Otherwise, if there is no single winner, print "`draw!`"

Answer:

```
char b[3][3];
// Assume that the variable 'b' is initialized properly.
// Write your answer below.
```

Problem 4: Looping [15%]

a) [7%] Write **a function** to calculate and return the factorial of a given non-negative integer n. Factorial of a non-negative integer n, defined by **n!** which is the product of all positive integers less than or equal to n. For example,

```
5! = 5 * 4 * 3 * 2 * 1 = 120
```

In principle,

```
0! = 1
n! = n * (n-1)* (n-2)... * 2 * 1   ,with n > 0
```

Answer:

```
int factorial(int n) {

// Write your answer below




} // End of factorial function
```

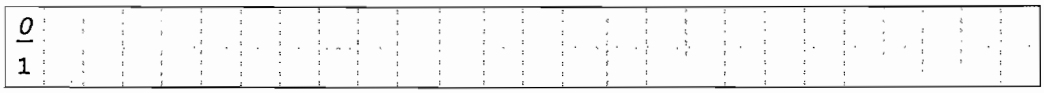
b) [8%] Write a **program fragment** to calculate and return the factorial of a given n. In this part, you are required to write the program code such that the user is required to repeatedly input an integer until the input value is a non-negative integer. By then the program will calculate and display the result. You may directly call the result in part (a) if it is useful here.

Two sample runs are shown as follows (italic, bold, underlined characters are from the user):

Example 1 of 2

```
-5
Non-negative int!
-10
Non-negative int!
5
120
```

Example 2 of 2



Answer:

```
// You can assume that the factorial function is defined
// correctly based on the requirements stated in Part (a)
```

Problem 5: Recursion [10%]

Given the following recursive function, namely **function1**:

```
int function1(int array[3], int pos) {
  int result;
  if(pos == 3) {
    printf("Recursion stops\n");
    return 0;
  }
  else {
    result = array[pos] + function1(array, pos+1);
    printf("(%d, %d)\n", pos, result);
    return result;
  }
}
```

Please write down the output generated by the following main function:

```
int main(void) {
  int array[3] = {2, 3, 5};
  printf("Answer = %d\n", function1(array, 0));
  return 0;
}
```

Answer:

Problem 6: Array, Structure and Pointer [25%]

Consider the following type definition for representing a NumberList in C:

```
typedef struct {
    int data[100];  // For keeping at most 100 data items;
    int listSize;   // storing actual number of data items
} NumberList;
```

a) [5%] Write a code segment to declare a **NumberList** variable named **primes** and the value of that variable should be initialized to represent a list of prime numbers 19, 2, 31, 7 and 23.

Answer:

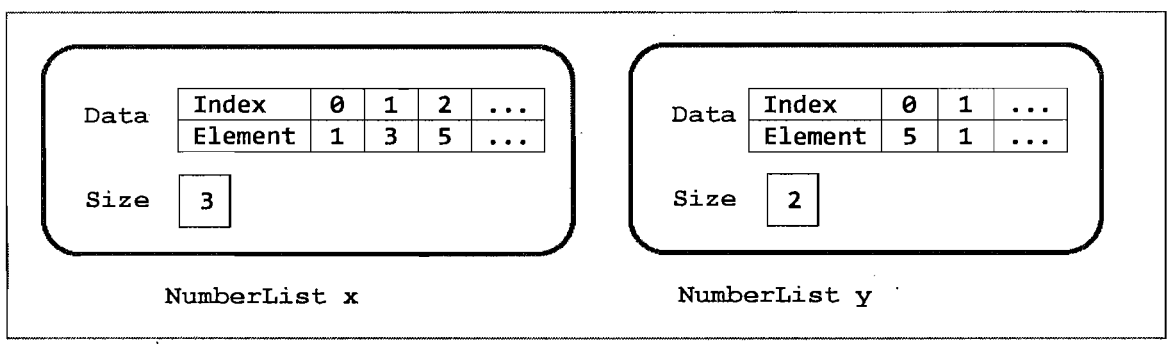
b) [5%] Complete the implementation of the following function for concatenating two NumberList's, namely **concat**. Assume parameters a and b of **concat** are valid NumberList's. The function creates and returns a new combined NumberList with a copy of all the data items in a followed by another copy of all the data items in b. The list size of the result is the sum of that of both parameters. The function shall not print anything.

Let us consider the following example:

```
NumberList x, y;
NumberList result1, result2;

// assume that x and y are initialized properly
result1 = concat( x, y );
result2 = concat( y, x );
```

The following figure illustrates the resulting structures result1 and result2. Note that "..." in the following figure means data elements that are not covered by **size**.



```
NumberList result1;
result1 = concat( x, y );
```

Data

Index	0	1	2	3	4	...
Element	1	3	5	5	1	...

Size 5

```
NumberList result2;
result2 = concat( y, x );
```

Data

Index	0	1	2	3	4	...
Element	5	1	1	3	5	...

Size 5

Answer:

```
NumberList  concat ( NumberList a,  NumberList b ) {




} // End of concat function
```

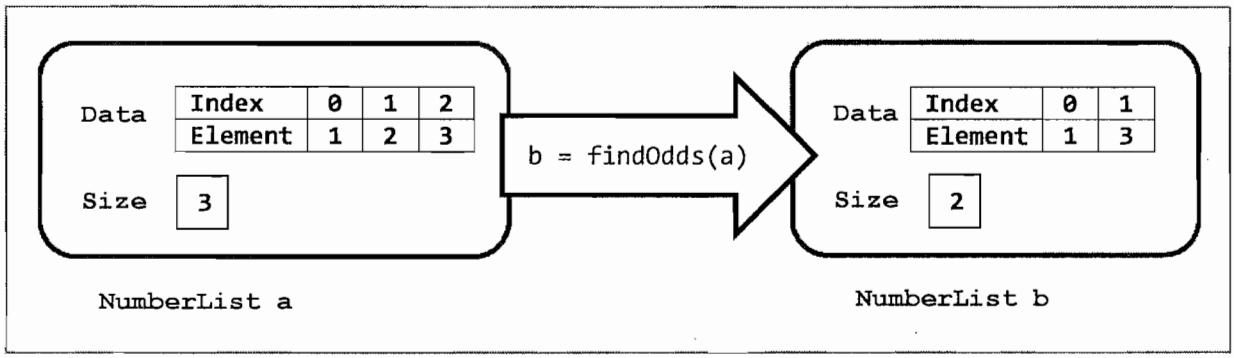
c) [5%] Complete the implementation of the following function, namely **findOdds**, for finding all odd data items in a NumberList. Assume parameter c of **findOdds** is a valid NumberList. The function creates and returns a **new NumberList** which contains only all the odd data items in the parameter c. The function shall not remove duplicates in the result even if there are repeated data items in c. The function shall not print anything.

Let us consider the following example:

```
NumberList a;
NumberList b;

// assume that a is initialized properly
b = findOdds( a );

// The following figure illustrates the resulting structure
```



Answer:

```
NumberList  findOdds ( NumberList c ) {




} // End of findOdds function
```

d) [10%] Complete the implementation of the following function for sorting all the data items in a NumberList in **ASCENDING** order. The parameter p is passed by address, assumed to be pointing to a valid NumberList. The function makes all changes in the NumberList pointed by **p**. It returns nothing and prints nothing.

For example use of the **sort** function.

```
NumberList a;      // assume that a is initialized properly.
sort( &a );        // "data" of a is sorted in ascending order.
```

Answer:

```
void sort ( NumberList * p ) {




} // End of sort function
```

<< Appendix >>

List of Partial C Operators in Decreasing Precedence	Associativity
() [] . -> ++ (postfix) -- (postfix)	left-to-right
+ (unary) - (unary) ++ (prefix) -- (prefix) !	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>>