Section C – Structures [Ans 1 Specified Qn from this Section]

1. **(computeCircle)** A structure called circle is defined below. The structure consists of the radius of the circle and the (x,y) coordinates of its centre.

```
struct circle {
    double radius;
    double x;
    double y;
};
```

(a) Implement the function `intersect()` that returns 1 if two circles intersect, and 0 otherwise. Two circles intersect when the distance between their centres is less than or equal to the sum of their radii. The function prototype is given below:

```
int intersect(struct circle c1, struct circle c2);
```

(b) Implement the function `contain()` that returns 1 if *c1* contains *c2*, i.e. circle *c2* is found inside circle *c1*. Otherwise, the function returns 0. Circle *c1* contains circle *c2* when the radius of *c1* is larger than or equal to the sum of the radius of *c2* and the distance between the centres of *c1* and *c2*. The function prototype is given below:

```
int contain(struct circle *c1, struct circle *c2);
```

Write a C program to test the functions.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter circle 1 (radius x y):
10 5 5
Enter circle 2 (radius x y):
5 1 1
intersect(): 1
contain(): 0
```

(2) Test Case 2:

```
Enter circle 1 (radius x y):
10 5 5
Enter circle 2 (radius x y):
1 1 1
intersect(): 1
contain(): 1
```

(3) Test Case 3:

```
Enter circle 1 (radius x y):
1 5 5
Enter circle 2 (radius x y):
1 10 10
intersect(): 0
contain(): 0
```

A sample program to test the functions is given below:

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
struct circle {
   double radius;
   double x;
   double y;
```

```
};
int intersect(struct circle c1, struct circle c2);
int contain(struct circle *c1, struct circle *c2);
int main()
{
   struct circle c1, c2;

   printf("Enter circle 1 (radius x y): \n");
   scanf("%lf %lf %lf", &c1.radius, &c1.x, &c1.y);
   printf("Enter circle 2 (radius x y): \n");
   scanf("%lf %lf %lf", &c2.radius, &c2.x, &c2.y);
   printf("intersect(): %d\n", intersect(c1, c2));
   printf("contain(): %d\n", contain(&c1, &c2));
   return 0;
}
int intersect(struct circle c1, struct circle c2)
{
   /* Write your program code here */
}
int contain(struct circle *c1, struct circle *c2)
{
   /* Write your program code here */
}
```

2. **(computeExp)** A structure is defined to represent an arithmetic expression:

```
typedef struct {
    float operand1, operand2;
    char op;      /* operator '+','-','*' or '/'  */
} bexpression;
```

(a) Write a C function that computes the value of an expression and returns the result. For example, the function will return the value of 4/2 if in the structure passed to it, operand1 is 4, operator is '/' and operand2 is 2. The function prototype is given as:

```
float compute1(bexpression expr);
```

(b) Write another C function that performs the same computation with the following function prototype:

```
float compute2(bexpression *expr);
```

Write a C program to test the functions.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter expression (op1 op2 op):
5 8 +
compute1(): 13.00
compute2(): 13.00
```

(2) Test Case 2:

```
Enter expression (op1 op2 op):
8 5 /
compute1(): 1.60
compute2(): 1.60
```

(3) Test Case 3:

```
Enter expression (op1 op2 op):
5 8 *
compute1(): 40.00
```

```
compute2(): 40.00
```

A sample template for the program is given below:

```
#include <stdio.h>
typedef struct {
   float operand1, operand2;
   char op;
} bexpression;
float compute1(bexpression expr);
float compute2(bexpression *expr);
int main()
{
   bexpression e;

   printf("Enter expression (op1 op2 op): \n");
   scanf("%f %f %c", &e.operand1, &e.operand2, &e.op);
   printf("compute1(): %.2f\n", compute1(e));
   printf("compute2(): %.2f\n", compute2(&e));
   return 0;
}
float compute1(bexpression expr)
{
   /* Write your program code here */
}
float compute2(bexpression *expr)
{
   /* Write your program code here */
}
```

3. (**findMiddleAge**) Write a function `findMiddleAge()` that takes in an array of three persons, finds the person whose age is the middle one of the three persons, and returns the name and age of that person to the caller. For example, if the array is `{{"Tom",18},{"John",19}, {"Jim",20}}`, then the person `John` and his age will be returned. The structure Person is defined below:

```
typedef struct {
   char name[20];
   int age;
} Person;
```

The function prototype is given below:

```
Person findMiddleAge(Person *p);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter person 1:
john 23
Enter person 2:
peter 56
Enter person 3:
mary 31
findMiddleAge(): mary 31
```

(2) Test Case 2:

```
Enter person 1:
vincent 11
Enter person 2:
```

```
raymond 22
Enter person 3:
alex 12
findMiddleAge(): alex 12
```

A sample template for the program is given below:

```
#include <stdio.h>
typedef struct {
   char name[20];
   int age;
} Person;
Person findMiddleAge(Person *p);
int main()
{
   Person man[3], middle;
   int i;

   for (i=0; i<3; i++) {
      printf("Enter person %d: \n", i+1);
      scanf("%s", man[i].name);
      scanf("%d", &man[i].age);
   }
   middle = findMiddleAge(man);
   printf("findMiddleAge(): %s %d\n", middle.name, middle.age);
   return 0;
}
Person findMiddleAge(Person *p)
{
   /* Write your program code here */
}
```

4. **(computeAverage)** Assume the following structure is defined to represent a grade record of a student:

```
struct student{
      char name[20];     /* student name */
      double testScore; /* test score */
      double examScore; /* exam score */
      double total;      /* total = (testScore+examScore)/2 */
};
```

Write a C function `average()` that creates a database of maximum 50 students using an array of structures. The function reads in student name. For each student, it takes in the test score and exam score. Then it computes and prints the total score of the student. The input will end when the student name is `"END"`. Then, it computes and returns the average score of all students to the calling function (i.e. `main()`). The calling function then prints the average score on the display. The function prototype is given below:

```
double average();
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter student name:
Hui S
Enter test score:
35.5
Enter exam score:
```

```
43.5
Student Hui S total = 39.50
Enter student name:
END
average(): 39.50
```

(2) Test Case 2:

```
Enter student name:
Hui S
Enter test score:
34
Enter exam score:
45
Student Hui S total = 39.50
Enter student name:
Fong A
Enter test score:
67
Enter exam score:
56
Student Fong A total = 61.50
Enter student name:
END
average(): 50.50
```

(3) Test Case 3:

```
Enter student name:
END
average(): 0.00
```

A sample template for the program is given below:

```
#include <stdio.h>
#include <string.h>
struct student{
   char name[20]; /* student name */
   double testScore; /* test score */
   double examScore; /* exam score */
   double total;  /* total = (testscore+examscore)/2 */
};
double average();
int main()
{
   printf("average(): %.2f\n", average());
   return 0;
}
double average()
{
   /* Write your program code here */
}
```

5. (**encodeChar**) Write a function `encodeChar()` that accepts two character strings ***s*** and ***t***, and an array of structures as parameters, encodes the characters in ***s*** to ***t***, and passes the encoded string ***t*** to the caller via call by reference. During the encoding process, each *source* character is converted into the corresponding *code* character based on the following rules: `'a'→'d'; 'b'→'z'; 'z'→'a';` and `'d'→'b'`. For other source characters, the *code* will be the same as the *source*. For example, if the character string **s** is `"abort"`, then the encoded string **t** will be `"dzort"`. The structure `Rule` is defined below:

```
typedef struct {
   char source;
   char code;
} Rule;
```

The function prototype is given below:

```
void encodeChar(Rule table[5], char *s, char *t);
```

Write a C program to test the function.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Source string:
abort
Encoded string: dzort
```

(2) Test Case 2:

```
Source string:
fgh
Encoded string: fgh
```

A sample template for the program is given below:

```
#include <stdio.h>
typedef struct {
   char source;
   char code;
} Rule;
void encodeChar(Rule table[5], char *s, char *t);
int main()
{
   char s[80],t[80];
   Rule table[5] = {'a','d', 'b','z', 'z','a', 'd','b', '\0','\0' };

   printf("Source string: \n");
   gets(s);
   encodeChar(table,s,t);
   printf("Encoded string: %s\n", t);
   return 0;
}
void encodeChar(Rule table[5], char *s, char *t)
{
   /* Write your program code here */
}
```

6. (**customer**) Write a C program that repeatedly reads in customer data from the user and prints the customer data on the screen until the customer name "`End Customer`" (i.e., first_name last_name) is read. Your program should include the following two functions: the function `nextCustomer()` reads and returns a record for a single customer to the caller via a pointer parameter ***`acct`***, and the function `printCustomer()` takes a parameter ***`acct`*** and then prints the customer information. The prototypes of the two functions are given below:

```
void  nextCustomer(struct account *acct);
void  printCustomer(struct account acct);
```

The structure definition for **`struct account`** is given below:

```
struct account {
   struct
   {
      char lastName[10];
      char firstName[10];
   } names;
   int accountNum;
```

```
    double balance;
};
```

You are required to implement the functions `printCustomer()` and `nextCustomer()`. Write a C program to test the functions. In your code, you should follow the exact format of the required input and output given in the following test sample sessions.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter names (firstName lastName):
SC Hui
Enter account number:
123
Enter balance:
6789.89
Customer record:
SC Hui 123 6789.89
Enter names (firstName lastName):
End Customer
```

(2) Test Case 2:

```
Enter names (firstName lastName):
SC Hui
Enter account number:
123
Enter balance:
6789.89
Customer record:
SC Hui 123 6789.89
Enter names (firstName lastName):
FY Tan
Enter account number:
13
Enter balance:
69.89
Customer record:
FY Tan 13 69.89
Enter names (firstName lastName):
End Customer
```

(3) Test Case 3:

```
Enter names (firstName lastName):
End Customer
```

A sample program to test the functions is given below:

```
#include <stdio.h>
#include <string.h>
struct account {
   struct
   {
      char lastName[10];
      char firstName[10];
   } names;
   int accountNum;
   double balance;
};
void  nextCustomer(struct account *acct);
void  printCustomer(struct account acct);
int main()
{
```

```
    struct account record;
    int flag = 0;

    do {
       nextCustomer(&record);
       if ((strcmp(record.names.firstName, "End") == 0) &&
           (strcmp(record.names.lastName, "Customer") == 0))
          flag = 1;
       if (flag != 1)
          printCustomer(record);
    } while (flag != 1);
}
void nextCustomer(struct account *acct)
{
    /* Write your program code here */
}
void  printCustomer(struct account acct)
{
    /* Write your program code here */
}
```

7. (**phoneBook**) Write a C program that implements the following two functions. The function `readin()` reads a number of persons' names and their corresponding telephone numbers, passes the data to the caller via the parameter p, and returns the number of names that have entered. The character `'#'` is used to indicate the end of user input. The function `search()` finds the telephone number of an input name ***`target`***, and then prints the name and telephone number on the screen. If the input name cannot be found, then it will print an appropriate error message. The prototypes of the two functions are given below:

```
int readin(PhoneBk *p);
void search(PhoneBk *p, int size, char *target);
```

The structure definition for `PhoneBk` is given below:

```
typedef struct {
   char name[20];
   char telno[20];
} PhoneBk;
```

You are required to implement the two functions. Write a C program to test the functions.

Some test input and output sessions are given below:

(1) Test Case 1:

```
Enter name:
Hui Siu Cheung
Enter tel:
1234567
Enter name:
Philip Fu
Enter tel:
2345678
Enter name:
Chen Jing
Enter tel:
3456789
Enter name:
#
Enter search name:
Philip Fu
Name = Philip Fu, Tel = 2345678
```

(2) Test Case 2:

```
Enter name:
Hui Siu Cheung
Enter tel:
1234567
Enter name:
Chen Jing
Enter tel:
3456789
Enter name:
#
Enter search name:
Philip Fu
Name not found!
```

(3) Test Case 3:

```
Enter name:
#
Enter search name:
Philip Fu
Name not found!
```

A sample program to test the functions is given below:

```
#include <stdio.h>
#include <string.h>
#define MAX 100
typedef struct {
   char name[20];
   char telno[20];
} PhoneBk;
int readin(PhoneBk *p);
void search(PhoneBk *p, int size, char *target);
int main()
{
   PhoneBk s[MAX];
   char t[20];
   int size;

   size = readin(s);
   printf("Enter search name: \n");
   gets(t);
   search(s,size,t);
   return 0;
}
int readin(PhoneBk *p)
{
   /* Write your program code here */
}
void search(PhoneBk *p, int size, char *target)
{
   /* Write your program code here */
}
```

8. (**mayTakeLeave**) Given the following information, write the code for the functions `getInput()`, `mayTakeLeave()` and `printList()` with the following function prototypes:

```
typedef struct {
   int id;           /* staff identifier */
   int totalLeave;   /* the total number of days of leave allowed */
   int leaveTaken;   /* the number of days of leave taken so far */
} leaveRecord;
```

(a) `void getInput(leaveRecord list[ ], int *n);`

Each line of the input has three integers representing one staff identifier, his/her total number of days of leave allowed and his/her number of days of leave taken so far respectively. The function will read the data into the array *list* until end of input and returns the number of records read through *n*.

(b) `int mayTakeLeave(leaveRecord list[ ], int id, int leave, int n);`

It returns 1 if a leave application for *leave* days is approved. Staff member with identifier *id* is applying for *leave* days of leave. *n* is the number of staff in *list*. Approval will be given if the leave taken so far plus the number of days applied for is less than or equal to his total number of *leave* days allowed. If approval is not given, it returns 0. It will return -1 if no one in *list* has identifier *id*.

(c) `void printList(leaveRecord list[ ], int n);`

It prints the *list* of leave records of each staff. *n* is the number of staff in *list*.

Write a C program to test the functions. You do not need to check any errors in the input. In your code, you should follow the exact format of the required input and output given in the following test sample sessions.

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Enter the number of staff records:
2
Enter id, totalleave, leavetaken:
11 28 25
Enter id, totalleave, leavetaken:
12 28 6
The staff list:
id = 11, totalleave = 28, leave taken = 25
id = 12, totalleave = 28, leave taken = 6
Please input id, leave to be taken:
11 6
The staff 11 cannot take leave
```

(2) Test Case 2:

```
Enter the number of staff records:
2
Enter id, totalleave, leavetaken:
11 28 25
Enter id, totalleave, leavetaken:
12 28 6
The staff list:
id = 11, totalleave = 28, leave taken = 25
id = 12, totalleave = 28, leave taken = 6
Please input id, leave to be taken:
12 6
The staff 12 can take leave
```

(3) Test Case 3:

```
Enter the number of staff records:
2
Enter id, totalleave, leavetaken:
11 28 25
Enter id, totalleave, leavetaken:
12 28 6
The staff list:
id = 11, totalleave = 28, leave taken = 25
id = 12, totalleave = 28, leave taken = 6
Please input id, leave to be taken:
13 6
The staff 13 is not in the list
```

A sample program to test the functions is given below:

```
#include <stdio.h>
#define INIT_VALUE 1000
typedef struct {
   int id;           /* staff identifier */
   int totalLeave;   /* the total number of days of leave allowed */
   int leaveTaken;   /* the number of days of leave taken so far */
} leaveRecord;
int mayTakeLeave(leaveRecord list[], int id, int leave, int n);
void getInput(leaveRecord list[], int *n);
void printList(leaveRecord list[], int n);
int main()
{
   leaveRecord listRec[10];
   int len;
   int id, leave, canTake=INIT_VALUE;

   getInput(listRec, &len);
   printList(listRec, len);
   printf("Please input id, leave to be taken: \n");
   scanf("%d %d", &id, &leave);
   canTake = mayTakeLeave(listRec, id, leave, len);
   if (canTake == 1)
      printf("The staff %d can take leave\n", id);
   else if (canTake == 0)
      printf("The staff %d cannot take leave\n", id);
   else if (canTake == -1)
      printf("The staff %d is not in the list\n", id);
   else
      printf("Error!");
   return 0;
}
void printList(leaveRecord list[], int n)
{
   int p;

   printf("The staff list:\n");
   for (p = 0; p < n; p++)
      printf ("id = %d, totalleave = %d, leave taken = %d\n",
         list[p].id, list[p].totalLeave, list[p].leaveTaken);
}
void getInput(leaveRecord list[], int *n)
{
   /* Write your program code here */
}
int mayTakeLeave(leaveRecord list[], int id, int leave, int n)
{
   /* Write your program code here */
}
```