ASSIGNMENT – STRUCTURES

1. (**rectangle**) A structure called Point is defined to represent a point in 2D which is given as follows:

```
typedef struct {
   double x;
   double y;
} Point;
```

Another structure called Rectangle is defined as follows:

```
typedef struct {
   Point topLeft;     /* top left point of rectangle */
   Point botRight;    /* bottom right point of rectangle */
} Rectangle;
```

Write a C program that reads in the top left point and bottom right point of a rectangle, computes the area of the rectangle and prints the area of the rectangle on the screen. Your program should include the following three functions with prototypes:

(1) `void getRect(Rectangle *r);` /* read in the two points of rectangle */
(2) `void printRect(Rectangle r);` /* print the coordinates of two points of rectangle */
(3) `double findArea(Rectangle r);`/* return the area of rectangle */

A sample program template is given below to test the functions:

```
#include <stdio.h>
#include <math.h>
typedef struct {
   double x;
   double y;
} Point;
typedef struct {
   Point topLeft;     /* top left point of rectangle */
   Point botRight;    /* bottom right point of rectangle */
} Rectangle;
void getRect(Rectangle *r);
void printRect(Rectangle r);
double findArea(Rectangle r);
int main()
{
   Rectangle r;
   int choice;

   printf("Select one of the following options:\n");
   printf("1: getRect()\n");
   printf("2: findArea()\n");
   printf("3: printRect()\n");
   printf("4: exit()\n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("getRect(): \n");
            getRect(&r);
            break;
```

```
            case 2:
               printf("findArea(): %.2f\n", findArea(r));
               break;
            case 3:
               printf("printRect(): \n");
               printRect(r);
               break;
            default:
               break;
         }
      } while (choice < 4);
      return 0;
}
void getRect(Rectangle *r)
{
    /* write your code here */
}
void printRect(Rectangle r)
{
    /* write your code here */
}
double findArea(Rectangle r)
{
    /* write your code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:

```
Select one of the following options:
1: getRect()
2: findArea()
3: printRect()
4: exit()
Enter your choice:
1
getRect():
Enter top left point:
1 2
Enter bottom right point:
2 1
Enter your choice:
3
printRect():
Top left point: 1.00 2.00
Bottom right point: 2.00 1.00
Enter your choice:
4
```

(2) Test Case 2:

```
Select one of the following options:
1: getRect()
2: findArea()
3: printRect()
4: exit()
Enter your choice:
1
getRect():
Enter top left point:
```

```
1 2
Enter bottom right point:
2 1
Enter your choice:
2
findArea(): 1.00
Enter your choice:
4
```

(3) Test Case 3:

```
Select one of the following options:
1: getRect()
2: findArea()
3: printRect()
4: exit()
Enter your choice:
1
getRect():
Enter top left point:
1 5
Enter bottom right point:
5 5
Enter your choice:
2
findArea(): 0.00
Enter your choice:
4
```

(4) Test Case 4:

```
Select one of the following options:
1: getRect()
2: findArea()
3: printRect()
4: exit()
Enter your choice:
1
getRect():
Enter top left point:
1 5
Enter bottom right point:
2 8
Enter your choice:
3
printRect():
Top left point: 1.00 5.00
Bottom right point: 2.00 8.00
Enter your choice:
2
findArea(): 3.00
Enter your choice:
4
```

2. (**encodeChar**) Write a function that accepts two character strings ***s*** and ***t***, an array of structures and the size of array 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 user defined rules. For other source characters, the `code` will be the same as the `source`. In addition, write a function `createTable()` that accepts two parameters, `table` and `size`. It allows users to define encoding rules in `table`. For example, when

the following rules `'a'→'d'; 'b'→'z'; 'z'→'a'; 'd'→'b'` are defined with `size = 4`, 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 prototypes are given below:

```
void createTable(Rule *table, int *size);
void encodeChar(Rule *table, int size, char *s, char *t);
```

A sample program template is given below to test the function:

```
#include <stdio.h>
typedef struct {
   char source;
   char code;
} Rule;
void createTable(Rule *table, int *size);
void printTable(Rule *table, int size);
void encodeChar(Rule *table, int size, char *s, char *t);
int main()
{
   char s[80], t[80], dummychar;
   int size, choice;
   Rule table[100];

   printf("Select one of the following options:\n");
   printf("1: createTable()\n");
   printf("2: printTable()\n");
   printf("3: encodeChar()\n");
   printf("4: exit()\n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("createTable(): \n");
            createTable(table, &size);
            break;
         case 2:
            printf("printTable(): \n");
            printTable(table, size);
            break;
         case 3:
            scanf("%c",&dummychar);
            printf("Source string: \n");
            gets(s);
            encodeChar(table,size,s,t);
            printf("Encoded string: %s\n", t);
            break;
         default:
            break;
      }
   } while (choice < 4);
   return 0;
```

```
}
void printTable(Rule *table, int size)
{
   int i;

   for (i=0; i<size; i++)
   {
      printf("%d: %c->%c\n", i+1, table->source, table->code);
      table++;
   }
}
void createTable(Rule *table, int *size)
{
   /* Write your program code here */
}
void encodeChar(Rule *table, int size, char *s, char *t)
{
   /* Write your program code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Select one of the following options:
1: createTable()
2: printTable()
3: encodeChar()
4: exit()
Enter your choice:
1
createTable():
Enter number of rules:
3
Enter rule 1:
Enter source character:
a
Enter code character:
b
Enter rule 2:
Enter source character:
b
Enter code character:
c
Enter rule 3:
Enter source character:
c
Enter code character:
d
Enter your choice:
2
printTable():
1: a->b
2: b->c
3: c->d
Enter your choice:
4
```

(2) Test Case 2:

```
<use Test Case 1>
```

```
Enter your choice:
3
Source string:
abcd
Encoded string: bcdd
Enter your choice:
4
```

(3) Test Case 3:

```
<Use Test Case 1>
Enter your choice:
3
Source string:
abort abort
Encoded string: bcort bcort
Enter your choice:
4
```

(4) Test Case 4:

```
Select one of the following options:
1: createTable()
2: printTable()
3: encodeChar()
4: exit()
Enter your choice:
1
createTable():
Enter number of rules:
2
Enter rule 1:
Enter source character:
a
Enter code character:
d
Enter rule 2:
Enter source character:
t
Enter code character:
b
Enter your choice:
3
Source string:
abort
Encoded string: dborb
Enter your choice:
4
```

3. (**student**) You are required to write a C program to process an array of student records. For each student record, it stores the student `id` and `name`. In the program, you need to write the following three functions:
   - The function `inputStud()` reads in students' information according to an input student size.
   - The function `printStud()` prints the student information on the display. It will print the message "`Empty array`" if the student list is empty.
   - The function `removeStud()` takes in three parameters. It removes the `target` student name from the array which has `*size` numbers in it. If `*size` is equal to zero the function should issue an error message "`Array is empty`". If the `target` name does not appear in the array, the function should issue an error message "`The target does not`

`exist`". The program defines the constant *SIZE* as the maximum number of student records which can be stored in the array. The function will return 0 if the removal operation is successful, 1 if the array is empty or 2 if the number does not exist in the array.

The prototypes of the three functions are given below:

```
void inputStud(Student *s, int size);
void printStud(Student *s, int size);
int removeStud(Student *s, int *size, char *target);
```

The structure definition for the structure `Student` is given below:

```
typedef struct {
   int id;
   char name[50];
} Student;
```

A sample program template is given below to test the functions:

```
#include <stdio.h>
#include <string.h>
#define SIZE 50
typedef struct {
   int id;
   char name[50];
} Student;
void inputStud(Student *s, int size);
void printStud(Student *s, int size);
int removeStud(Student *s, int *size, char *target);
int main()
{
   Student s[SIZE];
   int size=0, choice;
   char target[80];
   int result;

   printf("Select one of the following options: \n");
   printf("1: inputStud()\n");
   printf("2: removeStud()\n");
   printf("3: printStud()\n");
   printf("4: exit()\n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            printf("Enter size: \n");
            scanf("%d", &size);
            printf("Enter %d students: \n", size);
            inputStud(s, size);
            break;
         case 2:
            printf("Enter name to be removed: \n");
            scanf("\n");
            gets(target);
            printf("removeStud(): ");
            result = removeStud(s, &size, target);
            if (result == 0)
               printf("Successfully removed\n");
```

```
                else if (result == 1)
                   printf("Array is empty\n");
                else if (result == 2)
                   printf("The target does not exist\n");
                else
                   printf("An error has occurred\n");
                break;
             case 3:
                printStud(s, size);
                break;
          }
      } while (choice < 4);
      return 0;
}
void inputStud(Student *s, int size)
{
    /* write your code here */
}
void printStud(Student *s, int size)
{
    /* write your code here */
}
int removeStud(Student *s, int *size, char *target)
{
    /* write your code here */
}
```

Some sample input and output sessions are given below:

(1) Test Case 1:

```
Select one of the following options:
1: inputStud()
2: removeStud()
3: printStud()
4: exit()
Enter your choice:
1
Enter size:
3
Enter 3 students:
Student ID:
11
Student Name:
Hui Siu Cheung
Student ID:
12
Student Name:
Kenny B
Student ID:
13
Student Name:
Victor Leong
Enter your choice:
3
The current student list:
Student ID: 11 Student Name: Hui Siu Cheung
Student ID: 12 Student Name: Kenny B
Student ID: 13 Student Name: Victor Leong
```

```
Enter your choice:
4
```

(2) Test Case 2:

```
Select one of the following options:
1: inputStud()
2: removeStud()
3: printStud()
4: exit()
Enter your choice:
1
Enter size:
3
Enter 3 students:
Student ID:
11
Student Name:
Hui Siu Cheung
Student ID:
12
Student Name:
Kenny B
Student ID:
13
Student Name:
Victor Leong
Enter your choice:
2
Enter name to be removed:
Victor Leong
removeStud(): Successfully removed
Enter your choice:
3
The current student list:
Student ID: 11 Student Name: Hui Siu Cheung
Student ID: 12 Student Name: Kenny B
Enter your choice:
4
```

(3) Test Case 3:

```
Select one of the following options:
1: inputStud()
2: removeStud()
3: printStud()
4: exit()
Enter your choice:
1
Enter size:
3
Enter 3 students:
Student ID:
11
Student Name:
Hui Siu Cheung
Student ID:
12
Student Name:
Kenny B
Student ID:
```

```
13
Student Name:
Victor Leong
Enter your choice:
2
Enter name to be removed:
Victor Hui
removeStud(): The target does not exist
Enter your choice:
3
The current student list:
Student ID: 11 Student Name: Hui Siu Cheung
Student ID: 12 Student Name: Kenny B
Student ID: 13 Student Name: Victor Leong
Enter your choice:
4
```

(4) Test Case 4:

```
Select one of the following options:
1: inputStud()
2: removeStud()
3: printStud()
4: exit()
Enter your choice:
2
Enter name to be removed:
Victor Hui
removeStud(): Array is empty
Enter your choice:
3
The current student list:
Empty array
Enter your choice:
4
```

4. (**phoneBook**) Write a C program that implements the following three 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 `printPB()` prints the phonebook information on the display. It will print the message "`Empty phonebook`" if the phonebook list is empty.
   - 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:

The prototypes of the three functions are given below:

```
void printPB(PhoneBk *p, int size);
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;
```

A sample program template is given below to test the functions:

```
#include <stdio.h>
#include <string.h>
#define MAX 100
typedef struct {
   char name[20];
   char telno[20];
} PhoneBk;
void printPB(PhoneBk *p, int size);
int readin(PhoneBk *p);
void search(PhoneBk *p, int size, char *target);
int main()
{
   PhoneBk s[MAX];
   char t[20];
   int size=0, choice, dummychar;

   printf("Select one of the following options: \n");
   printf("1: readin()\n");
   printf("2: search()\n");
   printf("3: printPB()\n");
   printf("4: exit()\n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            scanf("%c", &dummychar);
            size = readin(s);
            break;
         case 2:
            scanf("%c", &dummychar);
            printf("Enter search name: \n");
            gets(t);
            search(s,size,t);
            break;
         case 3:
            printPB(s, size);
            break;
      }
   } while (choice < 4);
   return 0;
}
void printPB(PhoneBk *p, int size)
{
   /* Write your program code here */
}
int readin(PhoneBk *p)
{
   /* Write your program code here */
}
void search(PhoneBk *p, int size, char *target)
{
   /* Write your program code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:

```
Select one of the following options:
1: readin()
2: search()
3: printPB()
4: exit()
Enter your choice:
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 your choice:
3
The phonebook list:
Name: Hui Siu Cheung
Telno: 1234567
Name: Philip Fu
Telno: 2345678
Name: Chen Jing
Telno: 3456789
Enter your choice:
4
```

(2) Test Case 2:

```
Enter your choice:
2
Enter search name:
Philip Fu
Name = Philip Fu, Tel = 2345678
Enter your choice:
4
```

(3) Test Case 3:

```
Enter your choice:
2
Enter search name:
Tommy Fu
Name not found!
Enter your choice:
4
```

(4) Test Case 4:

```
Select one of the following options:
1: readin()
2: search()
3: printPB()
```

```
4: exit()
Enter your choice:
1
Enter name:
#
Enter your choice:
3
The phonebook list:
Empty phonebook
Enter your choice:
4
```