WEEK 6 ASSIGNMENT – STRUCTURES

Assignment on Coding Questions

For this assignment – you will need to answer 3 questions (randomly generated via APAS) from the following question list:

1. findMiddleAge
2. complexNumber
3. rectangle
4. encodeChar
5. student
6. customer
7. employee

Questions

1. (**findMiddleAge**) Write a function 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);
```

In addition, you are also required to write another function to read the three persons' information. The input data are passed to the calling function via the pointer parameter `p`. The function prototype is given below:

```
void readData(Person *p);
```

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

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

   readData(man);
   middle = findMiddleAge(man);
   printf("findMiddleAge(): %s %d\n", middle.name, middle.age);
   return 0;
}
void readData(Person *p)
{
   /* Write your program code here */
}
Person findMiddleAge(Person *p)
{
```

```
    /* Write your program code here */
}
```

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:

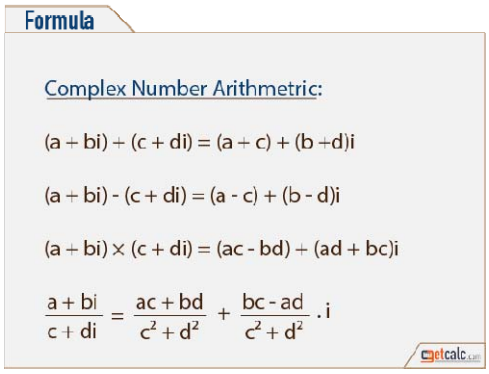
```
Enter person 1:
vincent 11
Enter person 2:
raymond 22
Enter person 3:
alex 12
findMiddleAge(): alex 12
```

2. (**complexNumber**) A structure called complex is defined to represent a complex number. Each complex number consists of the real and imaginary parts as follows:

```
typedef struct {
   double real;
   double imag;
} Complex;
```

Write C functions that perform addition, subtraction, multiplication and division operations on two complex numbers. The function prototypes are given as follows:

```
Complex add(Complex c1, Complex c2);
Complex mul(Complex c1, Complex c2);
Complex sub(Complex *c1, Complex *c2);
Complex div(Complex *c1, Complex *c2);
```



[source: https://getcalc.com/math-complex-numbers-arithmetic-calculator.htm]

Write a C program that handles complex number arithmetic using the arithmetic functions implemented. The program will read the choice of operation (e.g. addition, subtraction, multiplication, division and quit) and two complex numbers, and display the result.

A sample program to test the functions is given below:

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

```c
typedef struct {
   double real;
   double imag;
} Complex;
Complex add(Complex c1, Complex c2);
Complex mul(Complex c1, Complex c2);
Complex sub(Complex *c1, Complex *c2);
Complex div(Complex *c1, Complex *c2);
int main()
{
   int choice;
   Complex input1, input2, result;

   printf("Complex number operations: \n");
   printf("1 - addition \n");
   printf("2 - subtraction \n");
   printf("3 - multiplication \n");
   printf("4 - division \n");
   printf("5 - quit \n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      if (choice == 5)
         return 0;
      printf("Enter Complex Number 1: \n");
      scanf("%lf %lf", &input1.real, &input1.imag);
      printf("Enter Complex Number 2: \n");
      scanf("%lf %lf", &input2.real, &input2.imag);
      switch (choice) {
         case 1: result = add(input1, input2);
            break;
         case 2: result = sub(&input1, &input2);
            break;
         case 3: result = mul(input1, input2);
            break;
         case 4: result = div(&input1, &input2);
            break;
      }
      printf("complex(): real %.2f imag %.2f\n",
             result.real, result.imag);
   } while (choice<5);
   return 0;
}
Complex add(Complex c1, Complex c2)
{
    /* write your code here */
}
Complex sub(Complex *c1, Complex *c2)
{
    /* write your code here */
}
Complex mul(Complex c1, Complex c2)
{
    /* write your code here */
}
Complex div(Complex *c1, Complex *c2)
{
    /* write your code here */
}
```

Some test input and output sessions are given below:

(1) Test Case 1:

```
Complex number operations:
1 - addition
2 - subtraction
3 - multiplication
4 - division
5 - quit
Enter your choice:
1
Enter complex number 1:
1 1
Enter complex number 2:
2 2
complex(): real 3.00 imag 3.00
Enter your choice:
5
```

(2) Test Case 2:

```
Complex number operations:
1 - addition
2 - subtraction
3 - multiplication
4 - division
5 - quit
Enter your choice:
2
Enter complex number 1:
3 3
Enter complex number 2:
2 2
complex(): real 1.00 imag 1.00
Enter your choice:
5
```

(3) Test Case 3:

```
Complex number operations:
1 - addition
2 - subtraction
3 - multiplication
4 - division
5 - quit
Enter your choice:
3
Enter complex number 1:
3 3
Enter complex number 2:
4 6
complex(): real -6.00 imag 30.00
Enter your choice:
5
```

(4) Test Case 4:

```
Complex number operations:
1 - addition
2 - subtraction
3 - multiplication
4 - division
5 - quit
Enter your choice:
4
Enter complex number 1:
1 1
Enter complex number 2:
```

```
2 2
complex(): real 0.50 imag 0.00
Enter your choice:
5
```

3. (**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 2
Enter your choice:
3
printRect():
Top left point: 1.00 5.00
Bottom right point: 2.00 2.00
Enter your choice:
2
findArea(): 3.00
Enter your choice:
4
```

4. (**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>
#include <string.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, *p;
   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");
            fgets(s, 80, stdin);
            if (p=strchr(s,'\n')) *p = '\0';
            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:

(5) 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
```

(6) Test Case 2:

```
<use Test Case 1>
Enter your choice:
3
Source string:
abcd
Encoded string: bcdd
Enter your choice:
4
```

(7) Test Case 3:

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

(8) 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
```

5. (**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], *p;
   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");
            fgets(target, 80, stdin);
            if (p=strchr(target,'\n')) *p = '\0';
            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
```

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;
};
```

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

```
#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 */
}
```

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
```

7. (**employee**) Write a C program that creates an array of structures to hold the employee information below:

```
typedef struct {
   char name[40];
   char telno[40];
   int id;
   double salary;
} Employee;
```

You are required to implement 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 `search()` allows the user to query the array using the name field. If the name is found, the program displays the message "`Employee found at index location: x`". The function `search()` finds the employee data of an input name `target`, and then prints the name, telephone number, id and salary on the screen. If the input name cannot be found, then it will print the error message "`Name not found`" on the screen.
- If the name is not found and the array is not full, the user can then add the name as a new record into the array. Assume that the maximum size of the array is 100. The function `addEmployee()` adds a new employee record into the array. The message "`Added at position: x`" will be printed. If the database is fill, an error message "`Database is full`" will be printed on the screen. The function returns the updated size of the array after adding the new employee record.

The prototypes of the functions are given below:

```
int readin(Employee *emp);
int search(Employee *emp, int size, char *target);
int addEmployee(Employee *emp, int size, char *target);
```

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

```
#include <stdio.h>
#include <string.h>
#define MAX 100
typedef struct {
   char name[40];
   char telno[40];
   int id;
   double salary;
} Employee;
void printEmp(Employee *emp, int size);
int readin(Employee *emp);
int search(Employee *emp, int size, char *target);
int addEmployee(Employee *emp, int size, char *target);
int main()
{
   Employee emp[MAX];
   char name[40], *p;
   int size, choice, result;

   printf("Select one of the following options: \n");
   printf("1: readin() \n");
   printf("2: search() \n");
   printf("3: addEmployee() \n");
   printf("4: print() \n");
   printf("5: exit() \n");
   do {
      printf("Enter your choice: \n");
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            size = readin(emp);
            break;
         case 2:
            printf("Enter search name: \n");
            scanf("\n");
            fgets(name, 40, stdin);
            if (p=strchr(name,'\n')) *p = '\0';
            result = search(emp,size,name);
            if (result != 1)
               printf ("Name not found! \n");
            break;
         case 3:
            printf("Enter new name: \n");
            scanf("\n");
            fgets(name, 40, stdin);
            if (p=strchr(name,'\n')) *p = '\0';
            result = search(emp,size,name);
            if (result != 1)
               size = addEmployee(emp, size, name);
            else
               printf("The new name has already existed in the database\n");
            break;
         case 4:
            printEmp(emp, size);
            break;
         default:
            break;
      }
```

```
    } while (choice < 5);
    return 0;
}
void printEmp(Employee *emp, int size)
{
    int i;

    printf("The current employee list: \n");
    if (size==0)
        printf("Empty array\n");
    else
    {
        for (i=0; i<size; i++)
            printf("%s %s %d %.2f\n",emp[i].name,emp[i].telno,emp[i].id,
                                emp[i].salary);
    }
}
int readin(Employee *emp)
{
    /* write your code here */
}
int search(Employee *emp, int size, char *target)
{
    /* write your code here */
}
int addEmployee(Employee *emp, 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: readin()
2: search()
3: addEmployee()
4: print()
5: exit()
Enter your choice:
1
Enter name:
Hui Siu Cheung
Enter tel:
12345678
Enter id:
11
Enter salary:
123.45
Enter name:
#
Enter your choice:
4
The current employee list:
Hui Siu Cheung 12345678 11 123.45
Enter your choice:
5
```

(2) Test Case 2:

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

```
4: print()
5: exit()
Enter your choice:
1
Enter name:
Hui Siu Cheung
Enter tel:
12345678
Enter id:
11
Enter salary:
123.45
Enter name:
Kenny B
Enter tel:
23456789
Enter id:
12
Enter salary:
1234.45
Enter name:
#
Enter your choice:
4
The current employee list:
Hui Siu Cheung 12345678 11 123.45
Kenny B 23456789 12 1234.45
Enter your choice:
2
Enter search name:
Kenny B
Employee found at index location: 1
Kenny B 23456789 12 1234.45
Enter your choice:
5
```

(3) Test Case 3:

```
Select one of the following options:
1: readin()
2: search()
3: addEmployee()
4: print()
5: exit()
Enter your choice:
1
Enter name:
Hui Siu Cheung
Enter tel:
12345678
Enter id:
11
Enter salary:
123.45
Enter name:
Kenny B
Enter tel:
23456789
Enter id:
12
Enter salary:
1234.45
Enter name:
#
Enter your choice:
```

```
4
The current employee list:
Hui Siu Cheung 12345678 11 123.45
Kenny B 23456789 12 1234.45
Enter your choice:
2
Enter search name:
Kenny BB
Name not found!
Enter your choice:
5
```

(4) Test Case 4:

```
Select one of the following options:
1: readin()
2: search()
3: addEmployee()
4: print()
5: exit()
Enter your choice:
1
Enter name:
Hui Siu Cheung
Enter tel:
12345678
Enter id:
11
Enter salary:
123.45
Enter name:
Kenny B
Enter tel:
23456789
Enter id:
12
Enter salary:
1234.45
Enter name:
#
Enter your choice:
4
The current employee list:
Hui Siu Cheung 12345678 11 123.45
Kenny B 23456789 12 1234.45
Enter your choice:
3
Enter new name:
Kenny Tan
Enter tel:
12344321
Enter id:
13
Enter salary:
2345.67
Added at position: 2
Enter your choice:
4
The current employee list:
Hui Siu Cheung 12345678 11 123.45
Kenny B 23456789 12 1234.45
Kenny Tan 12344321 13 2345.67
Enter your choice:
5
```