01 - Pointers and Structures

Post-Lab Exercises

Exercise 1

Write, run, and analyse the following code in Code::Blocks. Afterwards, explain each line of the `bubbleSort()` function, specially detailing the pointer based method of the function call. You may get help from Chapter 8 of “C++ How to Program” by Deitel and Deitel.

```
#include <iostream>
using namespace std;

//FUNCTION PROTOTYPES
void bubbleSort(int arr[], int size, bool (*compare)(int a, int b));
void swap(int *a, int *b);          //same as lab exercise
bool ascending(int a, int b);       //transmitted as pointer
bool descending(int a, int b);      //transmitted as pointer

int main()
{
    const int SIZE = 10;
    int num[SIZE] = {2, 6, 4, 8, 10, 12, 89, 68, 45, 37};

    //Function call with ascending function sent as pointer
    bubbleSort(num, SIZE, ascending);

    //printing the sorted ascending array
    cout << "\nAscending sort:";
    for (int index = 0; index < SIZE; index++)
        cout << endl << "Element " << index  << " = " << num[index];

    //Function call with descending function sent as pointer
    bubbleSort(num, SIZE, descending);

    //Printing the sorted descending array
    cout << "\nDescending sort:";
    for (int index = 0; index < SIZE; index++)
        cout << endl << "Element " << index << " = " << num[index];
    cout << endl;
    return 0;
}   //end main()

//FUNCTION DEFINITIONS
void bubbleSort(int arr[], int SIZE, bool (*compare)(int a, int b))
{
    for (int i = 0; i < SIZE; i++)
        for (int j = i + 1; j < SIZE; j++)
            if ((*compare)(arr[i], arr[j])
                swap(&arr[i], &arr[j]);
            //end if
        //end inner for loop
    //end outer for loop
}//end bubbleSort
```

```
void swap(int *a, int *b)
{
    int temp = *a;      //store value pointed to by a
    *a = *b;            //assign value stored by b to a
    *b = temp;          //assign a's old value to b
}   //end swap

bool ascending(int a, int b)
{
    return a > b;
}   //end ascending

bool descending(int a, int b)
{
    return a < b;
}   //end descending
```

Output

```
Ascending sort:
Element 0 = 2
Element 1 = 4
Element 2 = 6
Element 3 = 8
Element 4 = 10
Element 5 = 12
Element 6 = 37
Element 7 = 45
Element 8 = 68
Element 9 = 89
Descending sort:
Element 0 = 89
Element 1 = 68
Element 2 = 45
Element 3 = 37
Element 4 = 12
Element 5 = 10
Element 6 = 8
Element 7 = 6
Element 8 = 4
Element 9 = 2
Program ended with exit code: 0
```

Explanation

This programme demonstrates the possibility and potential advantages of using pointers to functions. More specifically, it shows how a pointer to one function can be passed to another as an argument, and how this can make a programme more flexible.

Just as a pointer to a variable stores the memory address of that variable, a pointer to a function stores the memory address of the function's instructions. As such, instead of calling a function by its name or identifier, it is also possible to call a function using the memory address of its instructions through a pointer.

This is especially useful when a function needs to be passed as a parameter to another function i.e. in situations when one function may need to call any of a variety of similarly defined functions.

Consider the code for the `bubbleSort()` function below.

```
void bubbleSort(int arr[], int SIZE, bool (*compare)(int a, int b))
{
    for (int i = 0; i < SIZE; i++)
        for (int j = i + 1; j < SIZE; j++)
            if ((*compare)(arr[i], arr[j]))
                //*compare could be ascending or descending
                swap(&arr[i], &arr[j]);
            //end if
        //end inner for loop
    //end outer for loop
}//end bubbleSort
```

This function takes three arguments

1. an array of integers (passed by reference or as a memory address by default)
2. an integer that determines the size of the array - used for parsing the array.
3. a pointer to a 'generic' function that has the following properties:
    1. it takes two integer arguments, represented by the parameters a and b.
    2. it somehow processes these arguments to return a boolean.

`(*compare)` represents a pointer to a generic function, which we have chosen to represent with the name 'compare'. The name 'compare' is a parameter name for a function, just as 'a' and 'b' are parameter names for two integers.

It is important to enclose the *compare in parentheses because writing something like

```
bool *compare(int a, int b)
```

would not represent a pointer to a generic function 'compare' with two parameters a and b, but rather a pointer to the boolean variable returned by this function.

The rest of the `bubbleSort` function is fairly standard. It consists an outer and an inner loop, with the inner loop starting at one index after the outer loop. The idea is to eventually bubble values to the right of the array to form an array of numbers in ascending or descending order. At any given point in the algorithm's execution, index i is always less than index j.

If the pointer to the `ascending` function was given to this function as an argument, then the if conditional checks if the values at indexes i and j in the array are in ascending order. If the the value at index i is greater than the value at index j, `ascending` will return a true to indicate that the values are not in ascending order i.e. they NEED to be sorted in ascending order. This causes the `swap` function to be called with the values at these indexes in the array, which shifts the smaller value to the smaller index i.

Similarly, if `descending` was passed as an argument to the `bubbleSort( )` function, the if conditional would check if the value at index i was smaller than that at j, and return true to tell the caller that a swap is necessary because the two values are NOT in descending order. s

In this way, values are swapped and ‘bubbled’ to the end of the array through several iterations, eventually resulting in a sorted array.

Exercise 2

Write a C++ programme that defines a structure to store employee data for a company with 10 employees.

```
struct employee
{
    string  firstName;
    string  lastName;
    int     age;
    int     serviceInMonths;
    int     currentSalary;
    char    annualPerformance;      //a grade between 'a' and 'f'
};
```

The programme needs to go through the following steps:

1. **Declare an array of structures for the employees.**
2. **Ask user to enter credentials for all employees and store them.**
3. **Sort the complete array of structures on the basis of annual performance grade - highest first - using bubble sort. You must realise that this sorting shall require comparison on the basis of annual performance grade but shuffling will occur for all members of that specific array location.**
4. **Display the reordered array to verify that all changes were made successfully.**

```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

struct employee
{
    string  firstName;
    string  lastName;
    int     age;
    int     serviceInMonths;
    float   currentSalary;
    char    annualPerformance;      //a grade between 'a' and 'f'
};


void bubbleSort(employee myArray[], int arraySize);
void getEmployeeData(employee myArray[], int arraySize);
void printDatabase(employee myArray[], int arraySize);

int main()
{
    const int TOTAL_EMPLOYEES = 10;
    employee employeeArray[TOTAL_EMPLOYEES];

    getEmployeeData(employeeArray, TOTAL_EMPLOYEES);
    cout << "Employee database before sorting" << endl;
    printDatabase(employeeArray, TOTAL_EMPLOYEES);

    cout << "Sorting employee database" << endl;
    bubbleSort(employeeArray, TOTAL_EMPLOYEES);

    cout << "Employee database after sorting" << endl;
    printDatabase(employeeArray, TOTAL_EMPLOYEES);

    cout << "END OF PROGRAMME" << endl;
    return 0;
}


void bubbleSort(employee myArray[], int arraySize)
{
    int i, j;
    for (i = 0; i < arraySize - 1; i++)
        for (j = 0; j < arraySize - i - 1; j++)
            if (myArray[j].annualPerformance >
                      myArray[j + 1].annualPerformance)
            {
                employee temp = myArray[j];
                myArray[j] = myArray[j+1];
                myArray[j+1] = temp;
            }
}
```

```
void getEmployeeData(employee myArray[], int arraySize)
{
    const int FIELD_WIDTH = 24;
    for (int i = 0; i < arraySize; i++)
    {
        cout << "Entering data for employee number " << i + 1
           << "." << endl << endl;
        cout << setw(FIELD_WIDTH) << "First Name:\t";
        cin >> myArray[i].firstName;

        cout << setw(FIELD_WIDTH) << "Last Name:\t";
        cin >> myArray[i].lastName;

        cout << setw(FIELD_WIDTH) << "Age:\t";
        cin >> myArray[i].age;

        cout << setw(FIELD_WIDTH) << "Service (in months):\t";
        cin >> myArray[i].serviceInMonths;

        cout << setw(FIELD_WIDTH) << "Salary:\t";
        cin >> myArray[i].currentSalary;

        cout << setw(FIELD_WIDTH) << "Annual Performance:\t";
        cin >> myArray[i].annualPerformance;

        cout << endl << endl;
    }
}

void printDatabase(employee myArray[], int arraySize)
{
    cout << setw(12) << "First Name" << setw(12) << "Last Name"
    << setw(4) << "Age" << setw(8) << "Service" << setw(8) << "Salary"
    << setw(12) << "Performance" << endl;

    for (int i = 0; i < arraySize; i++)
    {
        cout << setw(12) << myArray[i].firstName
           << setw(12) << myArray[i].lastName
           << setw(4) << myArray[i].age
           << setw(8) << myArray[i].serviceInMonths
           << setw(8) << myArray[i].currentSalary
           << setw(12) << myArray[i].annualPerformance << endl;
    }
    cout << endl;
}
```

Output

```
Entering data for employee number 1.

            First Name:    Saad
             Last Name:    Siddiqui
                   Age:    21
   Service (in months):    10
                Salary:    1000
    Annual Performance:    d


Entering data for employee number 2.

            First Name:    Faiq
             Last Name:    Siddiqui
                   Age:    18
   Service (in months):    12
                Salary:    1200
    Annual Performance:    a


Entering data for employee number 3.

            First Name:    Usman
             Last Name:    Malik
                   Age:    32
   Service (in months):    24
                Salary:    2000
    Annual Performance:    c


Entering data for employee number 4.

            First Name:    Hasan
             Last Name:    Rehman
                   Age:    24
   Service (in months):    6
                Salary:    1800
    Annual Performance:    e


Entering data for employee number 5.

            First Name:    Zain
             Last Name:    Hasan
                   Age:    20
   Service (in months):    20
                Salary:    3000
    Annual Performance:    a
```

```
Entering data for employee number 6.

            First Name:    Gum
             Last Name:    Naam
                   Age:    32
   Service (in months):    10
                Salary:    1200
    Annual Performance:    b


Entering data for employee number 7.

            First Name:    Amjad
             Last Name:    Sohail
                   Age:    40
   Service (in months):    36
                Salary:    4000
    Annual Performance:    f


Entering data for employee number 8.

            First Name:    Haseeb
             Last Name:    Qadri
                   Age:    42
   Service (in months):    1
                Salary:    1200
    Annual Performance:    b


Entering data for employee number 9.

            First Name:    General
             Last Name:    Specific
                   Age:    55
   Service (in months):    12
                Salary:    4000
    Annual Performance:    c


Entering data for employee number 10.

            First Name:    Private
             Last Name:    Public
                   Age:    32
   Service (in months):    11
                Salary:    4500
    Annual Performance:    e
```

```
Employee database before sorting
  First Name   Last Name Age Service  Salary Performance
        Saad    Siddiqui  21      10    1000           d
        Faiq    Siddiqui  18      12    1200           a
       Usman       Malik  32      24    2000           c
       Hasan      Rehman  24       6    1800           e
        Zain       Hasan  20      20    3000           a
         Gum        Naam  32      10    1200           b
       Amjad      Sohail  40      36    4000           f
      Haseeb       Qadri  42       1    1200           b
     General    Specific  55      12    4000           c
     Private      Public  32      11    4500           e

Sorting employee database
Employee database after sorting
  First Name   Last Name Age Service  Salary Performance
        Faiq    Siddiqui  18      12    1200           a
        Zain       Hasan  20      20    3000           a
         Gum        Naam  32      10    1200           b
      Haseeb       Qadri  42       1    1200           b
       Usman       Malik  32      24    2000           c
     General    Specific  55      12    4000           c
        Saad    Siddiqui  21      10    1000           d
       Hasan      Rehman  24       6    1800           e
     Private      Public  32      11    4500           e
       Amjad      Sohail  40      36    4000           f

END OF PROGRAMME
Program ended with exit code: 0
```