CS101 Quiz 2 Answers

General instructions: **Time: 1 hour**

- There are 4 questions in this quiz. Write your answers directly on this paper in the spaces provided.
- Do not use a more general type where a simpler type would work (eg. do not use a float type where int will work.)
- Follow C++ syntax strictly when completing missing parts of code.

Q1) **[7 marks]** Fill up the blanks to match the output of code given below:

```
#include <simplecpp>
int main()
{
    int n = 4;
    int x[n][n];
    int count=1;
    for(int i=0;i<n;i++)
    for(int j=0;j<n;j++)
    {
          x[i][j]=count;count++;
    }

    for (int slice = 0; slice < 2* n - 1; slice++)
    {
          cout<<"Slice "<<slice<<": ";
          int z1=0;
          if(!(slice < n) )z1=slice - n + 1;

    for (int j = slice - z1; j >= z1; --j)
    {
          cout<<x[j][slice - j]<<" ";
    }
    cout<<endl;
    }
    return 0;
}
```

Output:

Slice 0: ______

Slice 1: ______

Slice 2: ______

Slice 3: ______

Slice 4: ______

Slice 5: ______

Slice 6: ______

Answer:

Slice 0: 1

Slice 1: 5 2

Slice 2: 9 6 3

Slice 3: 13 10 7 4

Slice 4: 14 11 8

Slice 5: 15 12

Slice 6: 16

Evaluation instructions:

1 mark per Slice blank

Marks will be allotted only if the output completely matches per blank.

Q2) **[14 Marks]** Structures as a programming construct are useful in implementing geometric objects. The following code contains the definition of two structs, describing a 2D point and a 2D polygon respectively. Given the the structure definitions, answer the questions that follow.

```
struct Point {
        int x, y;    // x and y coordinates of the point
        bool isEqual(Point p) {
            if(___________) return true;
            return false;
        }
        double distance(Point p) {
                return sqrt((x-p.x)*(x-p.x) + (y-p.y)*(y-p.y));
        }
};

struct Polygon1 {
        int num_vertices;     // actual number of vertices (<= 100)
        Point vertices[100];   // max number of vertices = 100
};
```

a. **[2 Marks]** The function **`bool isEqual(Point p)`** returns true if coordinates of **p** are exactly equal to the coordinates of the **Point** object calling the function and returns false otherwise. Fill in the blank with the appropriate condition.

Answer:

x==p.x && y==p.y

Evaluation instructions:

1. No marks to be awarded if answer differs from answer key (y==p.y && x==p.x is okay)
2. 2 marks for writing the condition properly

b. **[4 Marks]** The function **`void set(Point vertex_arr[], int n)`** is added to the **Polygon1** struct. Given an array **vertex_arr** of type Point, and an integer **n** as arguments, the function **void set(...)** initializes the data members of the **Polygon1** struct. Fill up the blank with appropriate lines of code to do the same(note that the code will span multiple lines).

The vertices in **vertex_arr** are such that the struct now represents a polygon obtained by connecting the given vertices in order they appear in the array(i.e. consecutive vertices in the array are connected and also the vertex at 0th and last index are connected. This knowledge can be used to answer the next parts.

```
struct Polygon1 {
        int num_vertices;
        Point vertices[100];

        void set(Point vertex_arr[], int n) {
                ___________________________________________
                ___________________________________________
                ___________________________________________
        }
};
```

Answer:

```
num_vertices = n;
for(int i = 0 ; i < num_vertices ; i++) {
        vertices[i] = vertex_arr[i];
}
```

Evaluation instructions:

1. Loop may be written in some other way (for example, looping from i = num_vertices-1 to i = 0) - full marks will be awarded for this.
2. There might be multiple answers to this part. Please use your own judgement while correcting this part.
3. 1 mark for num_vertices = n;
4. 3 marks for writing the loop perfectly(may not be the same as sample solution but should result in correct code)

c. **[2 Marks]** The function **double perimeter(Polygon1 p)** computes the perimeter of the polygon **p**. Complete the following code template so that it implements the perimeter function correctly.

```
double perimeter(Polygon1 p) {
  double sum = 0;
  for(int i=0; i<p.num_vertices-1; i++) {
        // use the distance() function defined previously
        sum = sum + ____________;
     }
   // use the distance() function defined previously
   sum = sum + __________;
   return sum;
}
```

Answer:

Blank-1: Two possible answers:

1. p.vertices[i].distance(p.vertices[i+1])
2. p.vertices[i+1].distance(p.vertices[i])

Blank-2: Two possible answers:

1. p.vertices[0].distance(p.vertices[p.num_vertices-1])
2. p.vertices[p.num_vertices-1].distance(p.vertices[0])

Evaluation instructions:

1. No marks to be awarded if the answer differs from the answer key
2. 1 mark each for the two blanks

d. **[6 Marks]** The function **bool `cyclicpermutation`(Polygon1 p1, Polygon1 p2)** checks whether the array of vertices of one of the polygons is a cyclic permutation of the other or not. It returns true if one of the arrays can be obtained by cyclically rotating the elements in the array of the other polygon and false otherwise.

arr1 is a cyclic permutation of **arr2** iff they have equal no. of elements (say **n**) and there exists an integer **p** such that **arr1[i] == arr2[(i+p)%num_vertices]** for all **0<=i<=num_vertices-1**.

```
bool cyclicpermutation(Polygon1 p1, Polygon1 p2) {
       // compare number of vertices in the 2 polygons
       if(______________________) return false;
       for(int i=0; i<______________________; i++) {
              int j = 0;
              while(_____________________) {
              // use isEqual member function of Point in above blank

                     j++;
                   if(___________________) return true;
              }
       }
      return false;
}
```

Answer:

Blank-1: p1.num_vertices != p2.num_vertices

Blank-2: Two possible answers:

1. p1.num_vertices
2. p2.num_vertices

Blank-3: Two possible answers:

1. p1.vertices[(i+j)%p1.num_vertices].isEqual(p2.vertices[j])
2. p2.vertices[j].isEqual(p1.vertices[(i+j)%p1.num_vertices])

Blank-4: j==p1.num_vertices

Evaluation instructions:

1. 1 mark each for first two blanks,
2. 2 marks each for last two blanks

Q3) **[9 Marks]** Given below is a struct called **complex_no** to facilitate storage of complex numbers. A complex number is a number that can be expressed in the form **a + ib,** where **a** and **b** are real numbers.

The struct **complex_no** has the following member functions:

set: Takes 2 numbers (a, b) that form the complex number as arguments

complex_add: Takes a complex number as its argument and returns the sum of this number with itself

complex_mult: Takes a complex number as its argument and returns the product of this number with itself

scalar_mult: Takes a scalar as argument and returns the product of this scalar with itself i.e., cmp.scalar_mult(scalar) gives the complex number scalar*cmp.

is_root: This functions takes a complex number and 3 integers **p , q , r** as arguments. These 3 integers form the coefficients of the quadratic equation **p*x^2 + q*x + r=0**. Utilising the member functions of the struct **complex_no**, find whether the given complex number is a root of this quadratic equation. This function returns a boolean value.

Now fill the blanks in the code template below.

```
#include <simplecpp>
struct complex_no {
   double a, b;
   void set(double a1, double b1) {
               a = a1;
               b = b1;
   }
   complex_no complex_add(complex_no c) {
               complex_no ret;
         ret.a = a + ______________;
               ret.b = b + ______________;
               return ret;
   }
   complex_no complex_mult(complex_no cmp) {
         complex_no ret;
```

```
            ret.a = ______________ * cmp.a + ______________ * cmp.b;

            ret.b = ______________ * cmp.a + ______________ * cmp.b;

            return ret;

    }

    complex_no scalar_mult(int c) {

                complex_no ret;

                ret.a = ______________;

                ret.b = ______________;

                return ret;

    }

};


bool is_root(complex_no cmp, int p, int q, int r) { // p*x^2 + q*x + r

    complex_no px2, qx, r1, ret;

    px2 = ____________________________________________;

    qx = _____________________________________________;

    r1.a = r;

    r1.b = 0;

    ret = px2.complex_add(qx).complex_add(r1);

    return ret.a == __________  && ret.b == __________;

}
```

Answer:

```
#include <simplecpp>

struct complex_no {
double a, b;
void set(double a1, double b1) {
    a = a1;
    b = b1;
}
complex_no complex_add(complex_no c) {
    complex_no ret;
    ret.a = a + c.a;
    ret.b = b + c.b;
    return ret;
}
complex_no complex_mult(complex_no cmp) {
  complex_no ret;
  ret.a = a*cmp.a + (-b)*cmp.b;
  ret.b = (a)*cmp.b + (b)*cmp.a;
  return ret;
}
complex_no scalar_mult(int c) {
    complex_no ret;
    ret.a = a*c;
    ret.b = b*c;
    return ret;
}
};

bool is_root(complex_no cmp, int p, int q, int r) { // p*x^2 + q*x + r
complex_no px2, qx, r1, ret;
px2 = cmp.complex_mult(cmp).scalar_mult(p);
qx = cmp.scalar_mult(q);
r1.a = r;
r1.b = 0;
ret = px2.complex_add(qx).complex_add(r1);
return ret.a == 0 && ret.b == 0;
}
```

Evaluation instructions:

1. 2 marks each for the values of px2 and qx
2. 0.5 marks per blank for the rest
3. Total 9 marks
4. Order of function calls in px2 could be different

Q4) **[11 Marks]** You have solved the finding mean and mode of students marks problems in one of the labs. Here you have to find median of students marks.

A sequence of marks of **n** (1 <= n <= 100) students out of 50 are given as input. Marks are integers(they can take any integer value between 0 and 50, both inclusive). We want to find out the median of marks. We will use **frequency_array** to find the median.

Median : Elements are first arranged in ascending order. If **n** is odd, median is the middle element. If **n** is even, then median is the average of the two middle elements.

For example:
Median of a = {3, 5, 3, 9, 11} is 5
Explanation : Middle element after arranging as {3, 3 , 5 , 9 , 11}.
frequency_array[3] = 2, frequency_array[5] = 1, frequency_array[9] = 1, frequency_array[11] = 1, remaining values of the array are 0.

Median of a = {9, 5, 3, 11} is 7
Explanation : Average of middle marks 5 and 9 after arranging as {3, 5, 9, 11}
frequency_array[3] = 1, frequency_array[5] = 1, frequency_array[9] = 1, frequency_array[11] = 1, remaining values of the array are 0.

Median of a = {32, 32, 3, 3} is 17.5
Explanation : Average of middle marks 3 and 32 after arranging as {3, 3, 32, 32 }
frequency_array[3] = 2, frequency_array[32] = 2, rest are 0.

Now fill in the blanks in the following code :

```
#include <simplecpp>

double findMedian(int a[],int n){

   int frequency_array[51];

   for(int i = 0 ; i < 51 ; i++) frequency_array[i] = 0;

   for(int i = 0 ; i < n ; i++) frequency_array[________] ++ ; //blank1

   int count_so_far = 0;


   if(n%2 == 1){

         for(int i=0;i<=50;i++){

               count_so_far += frequency_array[i];

               if(count_so_far >= (n+1)/2) return i;
```

```
        }
    }
    else{
    //middle1 and middle2 are the numbers that need to be averaged in order to find
    the median. The initial values of -1 indicate that they haven't been assigned
    yet.
        int middle1 = -1 , middle2 = -1;

        for(int i=0;i<=50;i++){

                count_so_far += frequency_array[i];

                if(count_so_far >= (____________) && middle1 == -1){ //blank2

                        middle1 = ____________; //blank3

                    }

                if(count_so_far >= ____________){ //blank4

                    middle2 = ____________; //blank5

                    break;

                }
        }
        return (middle1+middle2)/2.0;
    }
}

int main(){

    int a[100];

    int size;

    cin>>size;

    for(int i=0;i<size;i++) cin>>a[i];

    cout << findMedian(_________ , _________);

}
```

Answer:

Blank-1: a[i]

Blank-2: n/2

Blank-3: i

Blank-4: (n/2) + 1

Blank-5: i

Blank-6: a

Blank-7: size

Code :

```
#include <simplecpp>
double findMedian(int a[],int n){
   int frequency_array[51];
   for(int i=0;i<51;i++) frequency_array[i]=0;

   // filling up the frequency array
   for(int i=0;i<n;i++) frequency_array[a[i]] ++ ;

   int count_so_far = 0;

   //When number of students is Odd.
   if(n%2 == 1){

         for(int i=0;i<=50;i++){
               count_so_far += frequency_array[i];
               if(count_so_far >= (n+1)/2 ) return i;
         }

   }

   //When number of students is Even.
   else{

       //middle1 and middle2 are the numbers that need to be averaged in order to
       find the median.

       //Their initial value of -1 indicates that they haven't been assigned yet.
         int middle1=-1,middle2=-1;

         for(int i=0;i<=50;i++){
               count_so_far += frequency_array[i];

               if(count_so_far >= (n/2) && middle1 == -1){
                    middle1 = i;
               }
               if(count_so_far >= (n/2) + 1){
                      middle2 = i;
```

```
                        break;
                    }
                }
                return (middle1+middle2)/2.0;
            }

        }
        int main(){
            int a[100];
            int size;
            cin>>size;
            for(int i=0;i<size;i++) cin>>a[i];
            cout << findMedian(a,size);
        }
```

Evaluation Remarks :

1. Blank-1, Blank-6, Blank-7: 1 mark each
2. Blank-2, Blank-3, Blank-4, Blank-5: 2 marks each
3. Total 11 marks
4. For blank-2 and blank-5, consider other answers also which yield same final value (as there is integer division)