?? marks **CS 101 Quiz 2** **8:15-9:25, 26/10/12**

On the top right hand corner of your answerbook, please write the timings of your lab batch, e.g. "Thursday, 8:30-10:30".

Problem 1:[10 marks] We can describe a time duration by specifying 3 numbers: days, hours and minutes. It is expected that when a duration is specified, the number of minutes should be less than 60, and the number hours less than 24, i.e. 25 hours should be represented as 1 day, 1 hour and 0 minutes.

Define a struct `duration` to represent time durations specified in this manner. Include a member function which prints out the duration on `cout`. Write an ordinary function `dur_sum` which takes as arguments two `duration` structs and returns a `duration` struct which represents the sum of the argument durations.

```
struct duration{    //1 mark for correct syntax of structure
 double days, hours, minutes;    // 1 mark
 void print(){                   // 1 mark for correct syntax of function
    // 1 mark printing days, hours, minutes.  endl not need.
    cout << days <<' '<<hours<<' '<<minutes<<endl;
 }
};

duration dur_sum(duration d1, duration d2){
 duration res;
 res.minutes = d1.minutes + d2.minutes;
 res.hours   = d1.hours + d2.hours;
 res.days    = d1.days + d2.days;

 if(res.minutes >= 60){ res.minutes -= 60; res.hours++;}
 if(res.hours >= 24){ res.hours -= 24; res.days++;}
 return res;
}

// 1 mark: function declaration form.
// 2 mark: get the sum of the two durations
// 2 mark: manage the carry
// 1 mark: return a struct
```

Problem 2: (a)[4 marks] What is the output when the following code is executed?

```
#include<simplecpp>
struct foo{double x,y;};
double f(foo &f1, foo *f2, foo f3){
double res = (f1.x - f2->x) * (f3.y - f2->y);
f1.x = 20;
f2->x = 30;
f3.x = 40;
return res;
}
int main(){
foo p = {0,0}, q = {3,4}, r = {4,3};
double s = f(p, &q, r);
cout << s <<' '<<p.x<<' '<<q.x<<' '<<r.x<<endl;
}
```

3 20 30 4
1 mark for each blank
f1 is passed by reference. So changes made in f1 are reflected even outside of function.
f2 is passed by pointer and so changes made by dereferencing will be reflected outside function.
f3 is passed as copy by value and hence changes are only local to function.

(b)[5 marks] (i) Fill in the blanks in the code given for binary search. It is expected to determine whether the integer `t` is present in the subarray `a[start..end]` which is sorted in non-decreasing order.

```
bool bsearch(int t, int a[], int start, int end){
if(start == end) return (a[start] == t);
int mid = (start+end)/2;
if(a[mid] < t) return bsearch(t, a, ____, ____);
else return bsearch(t, a, ____, ____);
}
```

mid+1,end

start, mid

1 mark for each blank. Total - 4 marks
When $a[mid] < t$ then t belongs in second half of array from mid+1 to end. Note if we use indices as mid, end then it may run in infinte loop.
For example $a = [0, 1, 3]$ and $t = 2$ then $start = 0$ $end = 2$ $mid = 1$
$a[mid] < t$ so in next call if $start = mid$ $end = end$ then bsearch will be called with same start and end values in all subsequent calls leading to infinte loop. So we need $mid + 1$

For other case since value lies in first half of array so $start = start$ $end = mid$. Note mid-1 may be negative and hence it is not valid bound. Example $a = [0, 1]$, here $start = 0$ $end = 1$ $mid = 0$ so $mid - 1$ is negative and hence cannot be used. (ii) Suppose the first statement in the code above is changed to

```
if(start == end) return a[start];
```

Succinctly state what the function returns in this case in terms of `t` and the values in the array `a`.

1 mark
Two possible solutions:

1. Since return type of function is bool it returns true $a[start]$ is not 0 otherwise false. Note int vales are implicitly converted into bool with converted bool value being false if integer is 0 otherwise true.

2. Assuming that return type of function is also changed to int then function returns smallest value larger or equal to `t` if `t` is less than largest element in array. When `t` is greater than largest element then last element is returned.

Partial credit is given to people saying that there will be compile error or closest value near to `t` is returned.

Problem 3:(a) [5 marks]Define a two dimensional array `Q` having 9 rows and 9 columns, of type `int`. Also give code to read in numbers into it in row major order, i.e. the rows are read in one after another from top (row 0) to bottom (row 8), and each each row is read from left (column 0) to right (column 8).

```
 int Q[9][9];              // 1 mark
 for(int i=0; i<9; i++)    // 1 mark
    for(int j=0; j<9; j++)  // 1 mark
     cin >> Q[i][j];       // 2 mark
if cin  && << is written only one mark is cut
if cout && >> is written only one mark is cur
```

(b)[8 marks] A one dimensional array of n elements is said to be good if it contains each integer from 0 to $n-1$ exactly once. Write a function good that takes a one dimensional array and determines if it is good. How will you call this function to determine if the row of `Q` with index 5 is good?

```
bool good(int *a, int n){
 bool res = true;
 for(int i=0; i<n; i++){
    int count = 0;
    if(a[i] < 0 || a[i] >= n){ res = false; break;}
    for(int j=0; j<n; j++)
     if(a[i] == a[j]) count++;
    if(count != 1){res = false; break;}
 }
 return res;
}
// 1 mark: function declaration
// if function is written as void no marks are cut
// if function input is taken as int 0.5 marks are cut
// 2 marks: check that numbers are in the region 0 to n-1
// 3 marks: check uniqueness
// 2 mark: call: good(Q[5],9)
//only 1 mark cur if the call is good(Q[5][ ], 9). all other cases 2 marks are cut
```

(c)[8 marks] Consider the array `Q` as being made of 9 square subarrays, each 3x3. Say a subarray is even if it only contains even integers. Give code which prints whether each subarray is even, considering subarrays in row major order. No credit if you give separate code to check each subarray.

```
 for(int i=0; i<3; i++){
    for(int j=0; j<3; j++){
     bool even = true;
     for(int k=0; k<3; k++)
       for(int l=0; l<3; l++)
          if(Q[i*3+k][j*3+l] % 2 != 0) even = false;
     cout << even <<' ';
    }
    cout << endl;
 }
```

3 marks : being able to determine whether a single subarray, e.g. a[0..2][0..2] is even. Includes checking for even-ness (1 mark) and accumulating even-ness over rows and columns (2 marks). Ex:-writting a function 1 mark if the accumulation is partially correct

5 marks : being able to go over each subarray without duplicating the code. 3 marks: correctly goes over either subarrays in a row or in a column. 2 marks: correctly goes over the remaining.

full marks are awarded if in pass you noted all the odd numbers and then outputted the even sub arrays

Problem 4:[10 marks] The term permutation is used to denote the rearrangement of n distinct objects which can be considered to be placed at positions $0, 1, \ldots, n-1$ on the real line. After the rearrangement, the objects must also occupy the positions $0, 1, \ldots, n-1$. Further, at any position there must be exactly one object, both before and after the movement. Clearly, a permutation can be specified, by giving for each position i, the position $f(i)$ where the object at position i moves.

A permutation over n objects can be represented as an array where the ith element gives the position $f(i)$ to which the position causes the element at position i to move.

The order of a permutation P is defined as the smallest number $R > 0$ such that if you apply the permutation P repeatedly, R times, all the objects return to their initial positions.

Write a function that determines the order of a permutation given the array representing it. First describe in 7-8 sentences what your algorithm does. If you are uncomfortable writing a function, you may give a program for less credit.

```
/*
The basic idea is to start with i placed in position i.  Then we apply
the given permutation and move around these numbers.  We keep on doing
this.  After each movement, we check if all numbers have returned to
their positions, i.e. i is back at a[i].  At this point we break and
repeat how many times we have had to apply the permutation.
*/

// 3 marks correct general idea.  Has to be stated to get credit.
// Note that you were explicitly told to give a description.
// 7 marks to implement the idea correctly.

int order(int perm[], int n){
 int position[n];
 for(int i=0; i<n; i++) position[i] = i;  // 1 mark

 int result = 0;
 while(true){
    int c[n];
    for(int i=0; i<n; i++)
     c[perm[i]] = position[i];

    // 4 marks moving to new position.  Need a new array,
    // index expressions have to be correct.  It is possible they
    // could be written differently.
    // 3 marks deducted if new array is not used, but everything else is correct

    for(int i=0; i<n; i++)
     position[i] = c[i];
    result++;
    for(int i=0; i<n; i++) cout << position[i];
    cout << endl;
    bool identity = true;
    for(int i=0; i<n; i++) if (position[i] != i){identity = false; break;}
    if(identity) break;
 }
 return result;    // 2 marks getting result correctly.
}
```