100 marks **CS 101 Midsem** **8:30-10:30, 14/9/12**

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:(a)[5 marks] Write a program that reads in a real number and prints its square. Thus on reading 1.6 your program should print 2.56.

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

int main() {
  double realNum; // or float realNum
  cin >> realNum;
  cout << realNum*realNum << endl;
  return 0;
}
```

1 mark : Declaring a variable of type float/double for reading a single real number
1 mark : Correctly reading the real number from command line
3 mark : calculating the square and printing it.

No marks in case more than one numbers are read of same or other types. marks are deducted if you forget to print the square value or print incorrect value. partial marks for calculating square correctly but not printing it.

(b)[5 marks] Write a function that takes a real number as an argument and returns true if the number is greater than or equal to 0, and false otherwise.

```
bool foo(double realNum) {
  if (realNum >= 0) return true;
  else return false;
}

bool foo(double realNum) {
  return (realNum>=0);
}
```

0 marks : if not used funtion
1 marks : correct return type
1 marks : correct argument type for function (float/double)
3 marks : comparison and returning boolean
-1 marks for returning 1 or 0 and not true or false respectively marks are deducted for printing true,false instead of returning true,false

(c)[5 marks] Write a program that reads in one character and prints "digit" if it is a digit, and "letter" if it is a lower or upper case letter.

```
#include <iostream>
using namespace std;
int main() {
  char x;
  if (x >= '0' && x<= '9') cout << "digit\n";
  else if ( (x>='a' && x<='z') || (x>='A' && x<='Z') ) cout << "letter\n";
  return 0;
}
```

1 mark : declaring and reading a character variable
1 mark : correct condition to check digit and printing ”digit”
3 mark : correct condition to check character and printing ”letter”
marks deducted if you print the ascii value/ actual character instead of the strings
marks deducted if more than one variable are taken as input

(d)[5 marks] Write a program that reads in an integer denoting an year (e.g. 2012), and prints out the number of leap years from the date 1/1/0001 till the given year (including the given year). Note that an year is said to be a leap year if (i) it is divisible by 4, but not by 100, or (ii) if it is divisible by 400.

```
#include <iostream>
using namespace std;
int main() {
  int y;
  cin >> y;
  cout << y/4 - y/100 + y/400 << endl;
  return 0;
}
```

1 mark : correct declaring and reading the year
4 mark : calcuating correctly the number of leap years
marks deducted for printing the years instead of counting
partial marks for correct condition to check if year is leap year or not provided iterating through all years from 1 till y
partial marks for maintaining a counter to add number of leap years

Problem 2: (a)[10 marks] Write a program to compute ${}^nC_r = \frac{n!}{r!n-r!}$. A simple (non-recursive) program is expected. Note that $0! = 1$.

Non-Efficient Solution :

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

int fact(int num) {
int i, res=1;
for(i=1;i<=num;i++)
res = i*res;
return res;
}

int main(){
int n,r,ans;
cin>>n;
cin>>r;

ans = fact(n) / (fact(r)*fact(n-r));

cout<<ans<<endl;
return 0;
}
```

Marking Scheme :
Only 9 marks for such solution
6 marks for factorial computation (either using function or using 3 for loops)
3 marks for overall correctness

Efficient Solution :

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

int main(){
int i,n,r,j,ans,numerator=1,denominator=1;
cin>>n;
cin>>r;
i=n-r;
for(j=1;j<=r;j++) {
numerator *= (i+j);
denominator *=j;
}

ans = numerator / denominator;

cout<<ans<<endl;
return 0;
}
```

Marking Scheme :
3 for numerator
3 for denominator
3 for nearly correct solution
1 for overall

(b)[10 marks] Note that ${}^{n}C_r = {}^{n-1}C_r + {}^{n-1}C_{r-1}$. Use this to write a recursive function to compute ${}^{n}C_r$. Clearly mention the base case(s) you will use.

```
int rec_fun(int n, int r){
if(r==0 || n==r)
return 1;
else
return rec_fun(n-1,r) + rec_fun(n-1,r-1);
}
```

Marking Scheme :

4 - 2 marks for each base case x 2
4 - marks for each recursive call x 2
2 marks for overall correctness

Problem 3: (a)[10 marks] What happens when you execute the following code:

```
void flake(int level, double length){
  if(level>0){
    forward(length/3);
    left(60);
    flake(level-1, length/3);
    right(120);
    flake(level-1, length/3);
    left(60);
    forward(length/3);
  }
  else forward(length);
}

int main(){
  turtleSim();
  flake(0,200);
  flake(1,200);
  flake(2,200);
}
```

Marking Scheme:

- 1st function call: 1 marks, if correct
- 2nd function call: 3 marks
  - 1 mark, if 1st angle is correct
  - 1.5 marks (0.5 x 3), if rest of the angles are correct
  - 0.5 marks, correctness
- 3rd function call: 6 marks
  - 1 mark, 1st angle is correct
  - 4.5 marks (0.5 x 9), if rest of the angles are correct

  - 0.5 marks, correctness

(b)[15 marks] Modify `flake` so that it returns the distance travelled by the turtle, in addition to drawing the figure. Hint: the recursive calls will also do that..

Solution:

```
double flake(int level, double length){
  if(level>0){
    forward(length/3);
    left(60);
    double dist_1=flake(level-1, length/3);
    right(120);
    double dist_2=flake(level-1, length/3);
    left(60);
    forward(length/3);
    return (2*(length/3) + dist_1 + dist_2);
  }
  else
  {
    forward(length);
    return length;
  }
}
```

Marking Scheme:

- Calculating distance recursively, by changing the return type if the function
  - 2 marks, correct return type *i.e.* double/float
  - 2 marks, correct return value for base case (in else part)
  - 5 marks, capturing distance traversed from both the recursive call (-2 marks, if calling the same function again, since the figure becomes distorted)
  - 3 marks, calculating total distance (in if part)
  - 3 marks, return type and return value
- Calculating distance using global variable
  - 3 marks, if the base case is working correct
  - No more marks have been given, since the use of global variable was not expected
- Calculating distance using local variable/extra argument
  - 3 marks, if the base case is working correct (only the base case will work unless and until the local variable is used properly)
  - No more marks have been given, since by using a local variable, only base case will work. Similarly the solution by using an extra argument in function will only work for the base case

  – Some of the students have got correct solution using both, a local variable and an extra argument in function. Full marks have been given to them

Problem 4:[15 marks] Suppose a cargo ship containing some packages has just arrived at a dock. Each package has its weight written on it. The packages will be unloaded from the ship, one at a time, and must be immediately loaded onto a truck waiting at the dock. Each truck has a certain capacity C, i.e. it can carry total weight at most C. If the just arrived package will cause the total weight loaded on the waiting truck to go above its capacity C, then the package is not loaded, but the truck is asked to leave. Another truck is guaranteed to arrive immediately. The arriving packages are now loaded into this new truck, until its capacity is about to be exceeded, and so on. You are to write a program which takes as input the number C and then the weights of the packages as they arrive. After receiving the weight of a package, your program must print out either "Current" which means that the package must be loaded into the currently waiting truck, or "Next" indicating that the current truck must be asked to leave and the package should go into the next truck. If the number -1 is received as input, it should be treated as a signal indicating that all packages have been processed. In that case the program must terminate and print the number of trucks that were used. Assume each package weighs less than C.

Solution 4:

```
#include<simplecpp>
int main()
{
  int capacity,current,p,count;
  cout<<"enter capacity of truck";
  cin>>capacity;   //capacity of each trucks
  current=0;       // represents current added weight of package
  p=0;             //weight of the package
  count=0;         //number of trucks used
  cout<<"enter weight of pakage   ";
  cin>>p;          //initial input of first package
  if(p!=-1)
    count++; //if there is atleast on package then truck used should be nonzero(>0)

  while (p!=(-1))
  {
     current=current+p;  // add the package weight to total weight
     if(current<=capacity) // if current weight is less then or equal to capacity
     {
        cout<<"current"<<endl; //it should go in current truck
     }
     else
     {
        cout<<"next"<<endl;  //it should go in next truck
        current=p;  //the current pakage left should become total weight for next truck
        count++;  //new truck is used so increment count of truck
```

```
      }
      cout<<"enter weight of pakage";
      cin>>p;                //input next package weight
   }
   cout<<"truck used are "<<count<<endl; //print truck used at the exit
}
```

Grading scheme:
2 marks:For unbounded stream of input, even if one uses arrays it must be made sure that that no information is asked about number of packages, nor a very large array size to rely on.
10 marks for logic:The logic consisted of two core parts:correctness of current and next

- NEXT
  1. case when current sum of weights > capacity (3 marks)
  2. case when current sum of weights == capacity (2 marks)
- CURRENT
  1. case when current sum of weights < capacity (3 marks)
  2. case when current sum of weights == capacity (2 marks)
- Counting Truck Used
  correctness of counting total number of trucks used (5 marks)

NOTE: In many cases the packages have been dropped, that is if current sum of weights > capacity, the current sum of weights is reset to 0 .which is WRONG.(Refer to solution). In such case only 6 marks have been deducted, as the case of printing CURRENT is still correct but NEXT and total trucks is WRONG.
3 marks:The correct stopping condition. The input must stop at -1 and only then must the program end giving the total number of trucks. 1 mark is deducted if the following test case is wrong:
INPUT:-1
OUTPUT:total trucks used is 0
(means if not a single package weight is given in input, the total number of trucks used is 0)
TOTAL = 2+(5+5)+3

Problem 5: The input to your program is an integer T (target) and a sequence of positive integers $a_0, \ldots, a_{99}$. The goal is to determine whether there exists a subsequence $a_i, a_{i+1}, \ldots, a_{j-1}, a_j$ which adds up to exactly T.

(In this problem when we speak of a subsequence, we will only mean a subsequence with consecutive subscript values, as above.)

(a)[5 marks] Give the code that defines an array to read in the sequence, and then reads in the sequence and also T.

```
int a[100],T;
for(int i=0;i<100;i++) cin>>a[i];
cin>>T;
```

1 : Array declaration; array should be statically defined

1 : Declaring T

1 : loop for reading array

1 : cin statement in loop

1 : reading T

(b)[5 marks] Give the code that determines whether a subsequence of length exactly 37 (i.e. a subsequence of the form $a_i, a_{i+1}, \ldots, a_{i+36}$) exists such that it adds up to exactly T. Hint: try all the possibilities.

```
int sum=0;
for(int i=0;i<64;i++){
    sum=0;
    for(int j=i;j<37+i;j++) {sum+=a[j];}
    if(sum==T) break;
}
if(sum==T) cout<<"Exists\n";
else cout<<"Doesn't Exist\n";
```

1 : Bound 64 in outer for loop (63 if $i <= 63$ is used)

1 : Bound 37 in inner for loop (36 if $j <= 36 + i$ is used)

2 : Logic and Correctness

- initialising *sum* to 0 before every iteration of outer loop

1 : Proper output

Instead of summing the elements from i to $i + 36$ every time in the inner loop, one can just do

```
sum = sum+a[i+37]-a[i];
```

in every outer loop and properly initialize 'sum' and appropriately change boundary conditions. This is more efficient.

(c)[5 marks] Give the code that determines if some subsequence adding upto exactly T exists. Print the starting and ending subscript of the subsequence if it exists. You are expected to extend the idea of part (b).

```
int sum=0;
for(int i=0;i<100;i++){
    sum=0;
    for(int j=i;j<100;j++){
        sum+=a[j];
        if(sum==T) break;
    }
}
if(sum==T) cout<<i<<" "<<j<<endl;
else cout<<"Doesn't Exist\n";
```

1 : Bounds for inner and outer loops

3 : Logic and Correctness

- initialising *sum* to 0 before every iteration of outer loop
- note that checking if $sum == T$ should be in the inner loop, this is the crucial difference from the previous part

1 : Correctly printing the start index and ending index of subsequence

No marks are awarded if the student just copies the code from part 5b

(d)[5 marks] Suppose you know that $a_0 + a_1 + \cdots + a_8 < T$, but $a_0 + a_1 + \cdots + a_8 + a_9 > T$. What are the smallest possible values of i, j for which it is possible that $a_i + \ldots + a_j = T$? Explain in 4-5 sentences.

The smallest values of i, j are 1,9 respectively
If i=0, then $a_i + \cdots + a_j = a_0 + \cdots + a_j$ which is $> T$ if $j \geq 9$ and $< T$ if $j \leq 8$. So $i \geq 1$.
If $j \leq 8$ then $a_i + \cdots + a_j \leq a_0 + \cdots + a_8 < T$. So $j \geq 9$.

Also $i = 1, j = 9$ is possible if $a_0 = \cdots = a_8 = 1, a_9 = 2, T = 10$

2 : $i = 1$

2 : $j = 9$

1 : Proof

Extra credit: 10 marks. Based on part (d), give an algorithm which is more efficient than the algorithm of part (c).

The algorithm crucially rests on the fact that a_i are non-negative just as in part (d). The basic idea of the algorithm is this:

- Maintain two counters i, j and *sum* which equals $a_i + \cdots + a_j$
- If $sum < T$ increment j and update *sum*
- If $sum > T$ increment i and update *sum*
- If $sum == T$ break and print i, j

In addition we should take care of boundaries and break appropriately.
10 marks are awarded if the student presents this idea clearly. Partial marks are awarded if he doesn't express clearly or writes an incomplete algorithm.