Data Structure and Algorithm Lab Report

Name: Aritra Das Department: BCSE-2(Sem-1) Roll No-001910501075

Session-2020-21

Assignment - 1

Q1)

Problem Statement - Write a program to compute the factorial of an integer n iteratively and recursively. Check when there is overflow in the result and change the data types for accommodating higher values of inputs.

Solution approach - Program two functions named facrec(int n) which calculates the factorial of n recursively and faciter(int n) which calculates the value of factorial of n iteratively.

Structured Pseudo code:

```
Recursive Process:
facrec(n):
Begin
        If n>=1then
           return n*facrec(n-1)

       else
           return 1
End
Iterative Process:
faciter(n):
Begin
    res=1
    for i=1 to n do
        res*=i;
```

```
        return res;
    End
```

Results:

DataType(int) gives correct answer till 12!, but after that it overflows.

DataType(long int):gives correct answer till 20!

Discussion:

13! = 6,227,020,800, whereas int (signed) being 32 bit gives a max value of 2^(31) - 1= 2,147,483,647. As 13! is greater than the max range of signed int, an overflow is generated.

21! = 51,090,942,171,709,440,000, whereas long int (signed) being 64 bit gives a max value of 2^(63) - 1 = 9,223,372,036,854,775,807. As 21!is greater than the max range of signed long int, an overflow is generated.

Separate files containing commented source code:

Q2)

Problem statement: Write a program to generate the nth Fibonacci number iteratively and recursively. Check when there is overflow in the result and change the data types for accommodating higher values of inputs. Plot the Fibonacci number vs n graph.

Solution Approach: Program two functions named fibrec(int n) which calculates the fibonacci series of n recursively and fibiter(int n) which calculates the value of fibonacci of n iteratively.

Structured Pseudo code:

Recursive Process:

```
fibrec(n)
Begin
 If (n<=1)
     Return n;
Else
     Return fibrec(n-1) + fibrec(n-2);
Endif
End
```

Iterative Process:

```
Begin
Declare x=0,y=1,z=0;
for i=0 to (n-1) do
z=x+y;
x=y;
y=z;
```

```
return z;
End
```

Results:

Data Type(int) gives correct answer till 20!, but after that it overflows.

Data Type(unsigned long long int) gives correct answer till 91!.

Discussion:

Int (signed) being 32 bit gives a max value of 2^(31) - 1= 2,147,483,647. As 20th fibonacci number is greater than the max range of signed int, an overflow is generated.

Signed long long int being 64 bit gives a max value of 2^(63) - 1 = 9,223,372,036,854,775,807. As 91st is greater than the max range of signed long int, an overflow is generated.

Separate files containing commented source code:

Q3)

Problem statement: Write programs for linear search and binary search for searching integers, floating point numbers and words in arrays of respective types.

Solution Approach: A program with function search to search an element x in array arr of length n using the linear search method Another program to search elements in arrays using the binary search method

Structured Pseudo code

Linear Search Method:

```
search(arr,n,x):
Begin
  Int i;
  for(i=0;i<n;i++)
       if(arr[i] == x)
             Return i;
  Return -1;
End
```

Binary Search Method:

```
search(arr,l,r,x):
Begin
if(r>=l)
    Int m=l+(r-1)/2;
    if(arr[m] == x)
        Return m;
   if(arr[m]>x)
        Return search(arr,m+1,r,x);
   Return search(arr,m+1,r,x);
Return -1;
```

End

Result

Discussion:

For a list with n items, the best case is when the value is equal to the first element of the list, in which case only one comparison is needed. The worst case is when the value is not in the list (or occurs only once at the end of the list), in which case n comparisons are needed. And in general, the binary search takes lesser time than linear search. The time complexity for binary search worst case is O(log n) and for linear search is O(n).

Separate files containing commented source code:

Q4)

Problem statement: Write a program to generate 1,00,000 random integers between 1 and 1,00,000 without repetitions and store them in a file in character mode one number per line. Study and use the functions in C related to random numbers.

Solution Approach:

The function random trick assigns numbers from 1 to 100000 sequentially in an array. Then the rand() function generates random indexes and swaps out the elements of the array at those indexes creating a randomized order of sequences. This method is efficient in this scenario as all the numbers inside the limit needs to be utilized. Then the writefile function writes all the numbers to a file by one number in each line.

Structured Pseudo code:

```
randomstrict(arr[]):
Begin
Declare i;
For i=0 to 100000 do
        arr[i] = i+1;
For i=0 to 100000 do
        Declare temp = arr[i];
        Declare randomindex = rand() % 100000;
        arr[i] = arr[randomindex];
        arr[randomindex] = temp;
End
Writefile(Filepointer fpw, arr[]):
Begin
OpenFile(fpw,"data.txt")
For Declare i=0 to 100000 do
        WriteFile(arr[i])
CloseFile
End
```

Result:

In a separate file data.txt all the numbers are randomly printed without repetition.

Discussion:

The rand() function is used in C/C++ to generate random numbers in the range [0, RAND_MAX).

Note: If random numbers are generated with rand() without first calling rand(), your program will create the same sequence of numbers each time it runs. The srand() function sets the starting point for producing a series of pseudo-random integers. If srand() is not called, the rand() seed is set as if srand(1) were called at program start. Any other value for seed sets the generator to a different starting point.

Q5)

Problem Statement: Write a program to generate 1,00,000 random strings of capital letters of length 10 each, without repetitions and store them in a file in character mode one string per line.

Solution Approach: The function random picks randomly different characters from the array containing all the alphabets and builds n strings which has random characters of length 10 and writes them to a file.

Structured Pseudocode:

```
void random(n):

Begin
   char str[26] =
{'A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z'}
   if(n==0)
        n=100000;
   filePointer fp
   fileOpen("test.txt")
   if (fp == NULL)
                print NO FILE
                exit
   while (n != 0)
                for Declare i = 0 to 10
                fileWrite(fp, str[rand() % 10])
                n--
End
```

Results: The file is written by random strings of length 10 on each line.

Discussion

Separate files containing commented source code

Q6)

Problem Statement: Store the names of your classmates according to roll numbers in a text file one name per line. Write a program to find out from the file, the smallest and largest names and their lengths in number of characters. Write a function to sort the names alphabetically and store in a second file.

Solution Approach: Two functions created and used a structure to store the names and role numbers of different students. One function to take input from user and the other one directly reads all the names from a file which contains the list of students. The function arrange() arranges the students structure array according to roll number. The function lexico() arranges the names according to alphabets in lexicographical order and write them to a file. The function find() finds the name with maximum and minimum length and displays them to user. The function writefile() writes the names from structure array in another file.

Structured Pseudocode:

```
find(n,struct Student arr[]):
Begin
    Declare len1,len2,mx,mn,max,min;
    mx=stringLength(arr[0].name);
    for Declare i=0 to n
        len1= stringLength (arr[i].name)
        len2= stringLength (arr[i].name)
        if(len1>mx)
            max=i;
            mx=len1
        if(len2<mn)
            min=i;
            mn=len2
    print: The largest name is arr[max].name of length stringLength(arr[max].name)
    print: The largest name is arr[min].name of length stringLength(arr[min].name)

End
```

```
lexico(n,struct Student arr[]):
Begin
  for (int i = 0; i < n; ++i)
        for (int j = i+1 ; j < n; ++j)
              Declare char temp[51]
              if (stringCompare(arr[i].name, arr[j].name) > 0)
                    stringCopy(temp, arr[i].name)
                    stringCopy (arr[i].name, arr[j].name)
                    stringCopy (arr[j].name, temp
   filePointer fpw;
   fileOpen("student1.txt")
   for Declare i=0 to n
        fprintf(fpw,"%s\n",arr[i].name)
   fileClose(fpw)
End
arrange(n,struct Student arr[]):
Begin
        for Declare i = 0 to n-1
              for Declare j = 0 to n-i-1
                    if (arr[j].roll > arr[j+1].roll)
                          struct Student temp;
                          temp = arr[j];
                          arr[j] = arr[j+1];
                          arr[j+1] = temp;
End
```

Q7)

Problem statement: Take a four-digit prime number P. Generate a series of large integers L and for each member Li compute the remainder Ri after dividing Li by P. Tabulate Li and Ri. Repeat for seven other four digit prime numbers keeping Li fixed.

Solution approach: The function generate creates number of large numbers and stores them in li array. The function rem generates remainder of each li elements divided by prime number argument and store them in ri array. The function show generates a tabular form of data in li and ri. The function full contains the last two functions and is run for each element in prime number array.

Structured Pseudo code:

```
generate( li[], n):
Begin
    For Declare i=0 to n
        li[i]= (rand() % (100000-10000)) + 10000
End

rem(ri[], li[], n, prime):
Begin
    for(int i=0;i<n;i++)
         ri[i]=li[i] % prime
End

void show(uss li[],uss ri[],int n)
Begin
    print _____
    print Li            Ri
    print _____
    for Declare i=0 to n
        print (li[i])     (ri[i])
```

```
End

full(li[], ri[], n, prime)
Begin
    rem(ri,li,n,prime)
    show(li,ri,n)
End
```

Result:

Discussion:

Q8)

Problem Statement: Convert your Name and Surname into large integers by juxtaposing integer ASCII codes for alphabet. Print the corresponding converted integer. Cut the large integers into two halves and add the two halves. Compute the remainder after dividing the by the prime numbers P in problem 7.

Solution Approach: The function digit calculates the number of digits in a number. The function convert takes in a string input and it initializes a result with zero initial value and run a loop which converts each character to integer and adds it to result by multiplying the result variable with 10 raised to the power of digits in the converted integer from character. The function convert2 halves the converted string to juxtaposed integers and adds them and returns the sum. The function final divides the juxtaposed name and surname by individual elements of the primes array and then prints the remainders.

Structured Pseudocode:

```
int digit(n):
Begin
    Declare digit=0
    while(n!=0)
        digit++;
        n/=10;
    return digit
End

convert(char str[51]):
Begin
    Declare res=0;
    For Declare i=0 to stringLength(str)
            res = power(10,digit(convertToInt(str[i]))*res + convertToInt(str[i])
    return res
End

convert2(n):
Begin
```

```
    Declare n1=0,n2=0;
    n1= n/power(10,(digit(n)/2))
    n2 = n-(n1*power(10,(digit(n)/2)))
    return (n1+n2)
End

final(n1, n2):
Begin
    Declare prime[8] = {1009,1013,1291,1151,1451,1597,1613,2003}
    For int i=0 to 8
        Print: For (i+1)'th prime: (prime[i])
        Print: The remainder(name) for convert2(n1), divided by prime (prime[i]) is (convert2(n1)%prime[i])
        Print: The remainder(surname) for convert2(n2), divided by prime (prime[i]) is (convert2(n2)%prime[i])
End
```

Results: The program outputs the converted name and surname and then tabulates the remainder generated for the juxtaposed number divided by prime numbers.

Discussion

Separate files containing commented source code