Q1: gprof [10 marks]

VARIANT 1

You use gprof to get the call graph of the following C program.

```
void f1(), f2(), f3();

void f1() { f2(); f3(); }
void f2() { f3(); }
void f3() { }

int main()
{
   int x, y, z, i;

   scanf("%d%d%d", &x, &y, &z);
   for (i=0; i<x; ++i) { f1(); f2(); }
   for (i=0; i<y; ++i) { f1(); f3(); }
   for (i=0; i<z; ++i) { f2(); f3(); }

   exit(0);
}
```

A part (contiguous) of the output supplied by gprof is given below.

```
-----------------------------------------------
             0.00    0.00     7/19        main [9]
             0.00    0.00    12/19        f1 [3]
[2]    0.0   0.00    0.00    19        f2 [2]
             0.00    0.00    19/40        f3 [1]
-----------------------------------------------
```

What are the values of x, y and z supplied by the user? Show the equations involving x,y,z that can be derived from this gprof output, with one sentence stating how each of these equations is obtained. Finally, write the solution for x,y,z. No need to show the detailed calculations.

ANS

x + y = 12 [number of times f2 is called by main]
x + z = 7 [number of times f2 is called by f1]
3x + 3y + 2z = 40 [total number of calls of f3]

Solving gives x = 5, y = 7, and z = 2.

VARIANT 2

You use gprof to get the call graph of the following C program.

```
void f1(), f2(), f3();

void f1() { f2(); f3(); }
void f2() { f3(); }
void f3() { }

int main()
{
   int x, y, z, i;

   scanf("%d%d%d", &x, &y, &z);
   for (i=0; i<x; ++i) { f1(); f2(); }
   for (i=0; i<y; ++i) { f2(); f3(); }
   for (i=0; i<z; ++i) { f3(); f1(); }

   exit(0);
}
```

A part (contiguous) of the output supplied by gprof is given below.

```
-----------------------------------------------
             0.00    0.00    11/23         main [9]
             0.00    0.00    12/23         f1 [3]
[2]    0.0   0.00    0.00    23        f2 [2]
             0.00    0.00    23/42         f3 [1]
-----------------------------------------------
```

What are the values of x, y and z supplied by the user? Show the equations involving x,y,z that can be derived from this gprof output, with one sentence stating how each of these equations is obtained. Finally, write the solution for x,y,z. No need to show the detailed calculations.

ANS

x + y = 11 [number of times f2 is called by main]
x + z = 12 [number of times f2 is called by f1]
3x + 2y + 3z = 42 [total number of calls of f3]

Solving gives x = 8, y = 3, and z = 4.

VARIANT 3

You use gprof to get the call graph of the following C program.

```
void f1(), f2(), f3();

void f1() { f2(); f3(); }
void f2() { f3(); }
void f3() { }

int main()
{
   int x, y, z, i;

   scanf("%d%d%d", &x, &y, &z);
   for (i=0; i<x; ++i) { f1(); f2(); }
   for (i=0; i<y; ++i) { f1(); f3(); }
   for (i=0; i<z; ++i) { f2(); f3(); }

   exit(0);
}
```

A part (contiguous) of the output supplied by gprof is given below.

```
-----------------------------------------------
             0.00    0.00    10/10          main [9]
[3]    0.0   0.00    0.00    10         f1 [3]
             0.00    0.00    10/42          f3 [1]
             0.00    0.00    10/17          f2 [2]
-----------------------------------------------
```

What are the values of x, y and z supplied by the user? Show the equations involving x,y,z that can be derived from this gprof output, with one sentence stating how each of these equations is obtained. Finally, write the solution for x,y,z. No need to show the detailed calculations.

ANS

x + y = 10 [number of times f1 is called by main]
2x + y + z = 17 [total number of calls of f2]
3x + 3y + 2z = 42 [total number of calls of f3]

Solving gives x = 1, y = 9, and z = 6.

VARIANT 4

You use gprof to get the call graph of the following C program.

```
void f1(), f2(), f3();

void f1() { f2(); f3(); }
void f2() { f3(); }
void f3() { }

int main()
{
   int x, y, z, i;

   scanf("%d%d%d", &x, &y, &z);
   for (i=0; i<x; ++i) { f1(); f2(); }
   for (i=0; i<y; ++i) { f2(); f3(); }
   for (i=0; i<z; ++i) { f3(); f1(); }

   exit(0);
}
```

A part (contiguous) of the output supplied by gprof is given below.

```
-----------------------------------------------
             0.00    0.00    13/13        main [9]
[3]    0.0   0.00    0.00    13       f1 [3]
             0.00    0.00    13/43        f3 [1]
             0.00    0.00    13/18        f2 [2]
-----------------------------------------------
```

What are the values of x, y and z supplied by the user? Show the equations involving x,y,z that can be derived from this gprof output, with one sentence stating how each of these equations is obtained. Finally, write the solution for x,y,z. No need to show the detailed calculations.

ANS

$x + z = 13$ [number of times f1 is called by main]
$2x + y + z = 18$ [total number of calls of f2]
$3x + 2y + 3z = 43$ [total number of calls of f3]

Solving gives $x = 3$, $y = 2$, and $z = 10$.

Grading Guidelines for Q1

- 3 marks for each equation (1 mark for correctness, 2 marks for explanation)
- 1 mark for the correct solution
- No credit for linearly dependent equations

Q2: grep [6 marks]

VARIANT 1

You have a C file program.c in which the strings (if any) do not contain the characters { and }. These characters are used solely to indicate the beginnings and the ends of blocks. Show how you can run grep in order to print only those lines in program.c, in which a block is opened using { but not closed by } in the same line. For example, the following lines are to be printed.

```
} else {
n = 10; if (x == y) { n++; } for (i=0; i<n; ++i) {
```

The following line is not to be printed.

```
if (x == y) { n++; } for (i=0; i<n; ++i) { k += 2; }
```

Assume that there are no nested blocks in the same line, that is, situations like { { } or { } } or { { } { } } do not occur in a line. Write only the grep command.

ANS: `grep '{[^}]*$' program.c`

VARIANT 2

You have a C file utility.c in which the strings (if any) do not contain the characters { and }. These characters are used solely to indicate the beginnings and the ends of blocks. Show how you can run grep in order to print only those lines in utility.c, in which a block is closed using } but not opened by { in the same line. For example, the following lines are to be printed.

```
} else {
n++; sum += x; } else { sum -= x; }
```

The following line is not to be printed.

```
if (x == y) { n++; } for (i=0; i<n; ++i) { k += 2; }
```

Assume that there are no nested blocks in the same line, that is, situations like { { } or { } } or { { } { } } do not occur in a line. Write only the grep command.

ANS: `grep '^[^{]*}' utility.c`

VARIANT 3

You have a C file splprog.c in which the strings (if any) do not contain the characters { and }. These characters are used solely to indicate the beginnings and the ends of blocks. Show how you can run grep in order to print only those lines in splprog.c, in which a block is both opened by { and closed by } in the same line. Only the blocks in the line should be matched. For example, the following line should match like

```
} else if (x > 0) { --i; ++j; } else { ++i; --j; }
                  ------------      ------------
```

but not like

```
} else if (x > 0) { --i; ++j; } else { ++i; --j; }
                  --------------------------------
```

Assume that there are no nested blocks in the same line, that is, situations like { { } or { } } or { { } { } } do not occur in a line. Write only the grep command.

ANS: `grep '{[^}]*}' splprog.c`

Q3: grep + sed [6 + 8 marks]

VARIANT 1

Professor Foojit has a text file foonums.txt in which each line contains five positive integers separated by single spaces. There is no extra space at the beginning or at the end of any line. The integers may be arbitrarily large, but do not contain leading zero digits. Professor Foojit considers all integers in the range 500-5000 unlucky. Out of these unlucky numbers, the number 876 is considered super-unlucky.

(a) Write a grep command for Professor Foojit to print all the lines in foonums.txt, that do not contain any unlucky number. Write only the grep command.

(b) Write a single sed command for Professor Foojit that modifies and prints the lines in foonums.txt as follows. If the line does not contain the super-unlucky number 876, then the line is printed as it is. If the line contains one or more occurrences of the super-unlucky number, then only the first occurrence of the number in that line is replaced by the text BADNUM. For example, consider the following lines.

```
115 116 117 118 119
87654 9876 876 87 8
876 8765 87 876 876
8 87 878 877 876
```

These lines are to be printed as follows.

```
115 116 117 118 119
87654 9876 BADNUM 87 8
BADNUM 8765 87 876 876
8 87 878 877 BADNUM
```

ANS

```
(a) grep -v -w -e '[5-9][0-9][0-9]' -e '[1-4][0-9][0-9][0-9]' -e 5000 foonums.txt
(b) sed -e '1,$s/^876 /BADNUM /' -e '/BADNUM/!s/ 876 / BADNUM /'
        -e '/BADNUM/!s/ 876$/ BADNUM/' foonums.txt
```

VARIANT 2

Professor Foojit has a text file foonums.txt in which each line contains five positive integers separated by single spaces. There is no extra space at the beginning or at the end of any line. The integers may be arbitrarily large, but do not contain leading zero digits. Professor Foojit considers all integers in the range 399-3999 unlucky. Out of these unlucky numbers, the number 789 is considered super-unlucky.

(a) Write a grep command for Professor Foojit to print all the lines in foonums.txt, that do not contain any unlucky number. Write only the grep command.

(b) Write a single sed command for Professor Foojit that modifies and prints the lines in foonums.txt as follows. If the line does not contain the super-unlucky number 789, then the line is printed as it is. If the line contains one or more occurrences of the super-unlucky number, then only the first occurrence of the number in that line is replaced by the text BADNUM. For example, consider the following lines.

```
7 75 753 7531 777
789 789 79 78 789
6789 7890 1234 56 789
567 678 789 890 789
```

These lines are to be printed as follows.

```
7 75 753 7531 777
BADNUM 789 79 78 789
6789 7890 1234 56 BADNUM
567 678 BADNUM 890 789
```

ANS

```
(a) grep -v -w -e 399 -e '[4-9][0-9][0-9]' -e '[1-3][0-9][0-9][0-9]' foonums.txt
(b) sed -e '1,$s/^789 /BADNUM /' -e '/BADNUM/!s/ 789 / BADNUM /'
        -e '/BADNUM/!s/ 789$/ BADNUM/' foonums.txt
```

VARIANT 3

Professor Foojit has a text file foonums.txt in which each line contains five positive integers separated by single spaces. There is no extra space at the beginning or at the end of any line. The integers may be arbitrarily large, but do not contain leading zero digits. Professor Foojit considers all integers in the range 600-6000 unlucky. Out of these unlucky numbers, the number 963 is considered super-unlucky.

(a) Write a grep command for Professor Foojit to print all the lines in foonums.txt, that do not contain any unlucky number. Write only the grep command.

(b) Write a single sed command for Professor Foojit that modifies and prints the lines in foonums.txt as follows. If the line does not contain the super-unlucky number 963, then the line is printed as it is. If the line contains one or more occurrences of the super-unlucky number, then only the first occurrence of the number in that line is replaced by the text BADNUM. For example, consider the following lines.

```
1963 196 19 9631 963963
9 96 963 963 963
963 964 965 966 967
959 960 961 962 963
```

These lines are to be printed as follows.

```
1963 196 19 9631 963963
9 96 BADNUM 963 963
BADNUM 964 965 966 967
959 960 961 962 BADNUM
```

ANS

```
(a) grep -v -w -e '[6-9][0-9][0-9]' -e '[1-5][0-9][0-9][0-9]' -e 6000 foonums.txt
(b) sed -e '1,$s/^963 /BADNUM /' -e '/BADNUM/!s/ 963 / BADNUM /'
        -e '/BADNUM/!s/ 963$/ BADNUM/' foonums.txt
```

VARIANT 4

Professor Foojit has a text file foonums.txt in which each line contains five positive integers separated by single spaces. There is no extra space at the beginning or at the end of any line. The integers may be arbitrarily large, but do not contain leading zero digits. Professor Foojit considers all integers in the range 299-2999 unlucky. Out of these unlucky numbers, the number 593 is considered super-unlucky.

(a) Write a grep command for Professor Foojit to print all the lines in foonums.txt, that do not contain any unlucky number. Write only the grep command.

(b) Write a single sed command for Professor Foojit that modifies and prints the lines in foonums.txt as follows. If the line does not contain the super-unlucky number 593, then the line is printed as it is. If the line contains one or more occurrences of the super-unlucky number, then only the first occurrence of the number in that line is replaced by the text BADNUM. For example, consider the following lines.

```
589 590 591 592 593
593593 3593 5934 59 5
593 594 595 596 597
59 593 5937 593 593
```

These lines are to be printed as follows.

```
589 590 591 592 BADNUM
593593 3593 5934 59 5
BADNUM 594 595 596 597
59 BADNUM 5937 593 593
```

ANS

```
(a) grep -v -w -e 299 -e '[3-9][0-9][0-9]' -e '[1-2][0-9][0-9][0-9]' foonums.txt
(b) sed -e '1,$s/^593 /BADNUM /' -e '/BADNUM/!s/ 593 / BADNUM /'
        -e '/BADNUM/!s/ 593$/ BADNUM/' foonums.txt
```

Q4: gawk [10 marks]

VARIANT 1

Consider a file containing the employee details of a company in the following format:

```
EmployeeID Name PayScale Salary Perks TotalPay
```

Here, PayScale is an integer in the range 1-10, and Salary and Perks are positive integers. TotalPay is the sum of the Salary and the Perks components.

```
9001 Ashish_Gupta 10 20000 3000 23000
9002 Atul_Kumar 7 15340 3120 18460
9003 Bikash_Krishna_Das 9 16310 3560 19870
```

An increment is to be paid to all employees at the rate of 10% for PayScale 10, 8% for PayScale 9, and 6% for lower PayScale values, on the salary part only. The new salary will become the old salary plus the increment. So for, say Ashish Gupta, the new Salary will be 22000, and the new TotalPay will be 22000 + 3000 = 25000. In case the new salary has a fractional part, the fraction is ignored, and the integer part is taken.

Write a gawk script that takes the name of the employee file as the only argument, reads each line, computes the increment as per the above formula, and replaces the Salary in the line with the new salary, and the TotalPay with the new TotalPay, in the original file. The script is not supposed to print anything to the screen.

ANS

```
#!/usr/bin/gawk -f

BEGIN {
   FNAME = ARGV[1]
   n = 0
}

{
   ++n
   EmpID[n] = $1
   EmpName[n] = $2
   PayScale[n] = int($3)
   Salary[n] = int($4)
   Perks[n] = int($5)
   TotalPay[n] = int($6)
}

END {
   printf("") > FNAME
   for (i=1; i<=n; ++i) {
      if (PayScale[i] == 10) { Salary[i] = int(1.10 * Salary[i]) }
      else if (PayScale[i] == 9) { Salary[i] = int(1.08 * Salary[i]) }
      else { Salary[i] = int(1.06 * Salary[i]) }
      TotalPay[i] = Salary[i] + Perks[i];
      printf("%s %s %d %d %d %d\n", EmpID[i], EmpName[i], PayScale[i], Salary[i], Perks[i], TotalPay[i]) >> FNAME
   }
}
```

VARIANT 2

Consider a file containing the details of PhD students of an institute in the following format:

```
StudentID Name Year Scholarship BookGrant TotalPay
```

Here, Year is an integer in the range 1-8, and Scholarship and BookGrant are positive integers. TotalPay is the sum of the Scholarship and the BookGrant components.

```
FB9001 Ashish_Gupta 5 20000 3000 23000
FB9002 Atul_Kumar 2 15340 3120 18460
FB9003 Bikash_Krishna_Das 4 16310 3560 19870
```

An increment is to be paid to all students at the rate of 15% for Year 4, 12% for Year 5, and 9% for other Years, on the Scholarship part only. The new Scholarship will become the old Scholarship plus the increment. So for, say Ashish Gupta, the new Scholarship will be 22400, and the new TotalPay will be 22400 + 3000 = 25400. In case the new scholarship has a fractional part, the fraction is ignored, and the integer part is taken.

Write a gawk script that takes the name of the student file as the only argument, reads each line, computes the increment as per the above formula, and replaces the Scholarship in the line with the new scholarship, and the TotalPay with the new TotalPay, in the original file. The script is not supposed to print anything to the screen.

ANS

```
#!/usr/bin/gawk -f

BEGIN {
   FNAME = ARGV[1]
   n = 0
}

{
   ++n
   StudID[n] = $1
   StudName[n] = $2
   Year[n] = int($3)
   Schol[n] = int($4)
   BookGrant[n] = int($5)
   TotalPay[n] = int($6)
}

END {
   printf("") > FNAME
   for (i=1; i<=n; ++i) {
      if (Year[i] == 4) { Schol[i] = int(1.15 * Schol[i]) }
      else if (Year[i] == 5) { Schol[i] = int(1.12 * Schol[i]) }
      else { Schol[i] = int(1.09 * Schol[i]) }
      TotalPay[i] = Schol[i] + BookGrant[i];
      printf("%s %s %d %d %d %d\n", StudID[i], StudName[i], Year[i], Schol[i], BookGrant[i], TotalPay[i]) >> FNAME
   }
}
```

VARIANT 3

A company sells products, the details of which are stored in a file in the following format:

```
ProductID Name Category MRP Discount Cost
```

Here, Category is an integer in the range 1-3, MRP (Maximum Retail Price) and Discount are positive integers, and Cost is MRP minus Discount.

```
LT2197 Bar_Laptop 2 30000 2000 28000
OM5110 Optical_Mouse 1 490 50 440
TV1011 Foo_Widescreen 3 35000 1200 33800
```

After the new budget, the MRP of a Category 1 item increases by 5%, the MRP of a Category 2 item increases by 11%, and the MRP of a Category 3 item increases by 16%. The Discount is not affected by the budget. For example, the new MRP of Bar Laptop becomes 33300, and its new cost becomes 33300 - 2000 = 31300. In case the new MRP has a fractional part, the fraction is ignored, and the integer part is taken.

Write a gawk script that takes the name of the product file as the only argument, reads each line, computes the increment as per the above formula, and replaces the MRP in the line with the new MRP, and the Cost with the new Cost, in the original file. The script is not supposed to print anything to the screen.

ANS

```
#!/usr/bin/gawk -f

BEGIN {
   FNAME = ARGV[1]
   n = 0
}

{
   ++n
   ProdID[n] = $1
   ProdName[n] = $2
   Category[n] = int($3)
   MRP[n] = int($4)
   Discount[n] = int($5)
   Cost[n] = int($6)
}

END {
   printf("") > FNAME
   for (i=1; i<=n; ++i) {
      if (Category[i] == 1) { MRP[i] = int(1.05 * MRP[i]) }
      else if (Category[i] == 2) { MRP[i] = int(1.11 * MRP[i]) }
      else { MRP[i] = int(1.16 * MRP[i]) }
      Cost[i] = MRP[i] - Discount[i];
      printf("%s %s %d %d %d %d\n", ProdID[i], ProdName[i], Category[i], MRP[i], Discount[i], Cost[i]) >> FNAME
   }
}
```

Q5: Shell programming [10 marks]

VARIANT 1

Write a bash script classify.sh to do the following.

1. Write a function classify which takes a directory name as argument, and finds the size of each regular file in the directory. For each such file, the function prints the size of the file (in bytes), and classify the file as small (of size <= 1Kb), medium (of size > 1Kb but <= 1Mb), or large (of size > 1Mb).
2. If the script is called without any argument, exit with an error message.
3. If the first argument is not readable or not a directory, exit with appropriate error messages.
4. Call the function classify on the first argument.

Assume that no file or directory name contains spaces. In order to get the size of a file, use ls -l or wc -c. But these commands print some text in addition to the size.

Sample output:

```
$ ./classify.sh ~
/home/foobar/CampusMap.jpg is a large file of size 2341348
/home/foobar/takebackup is a small file of size 473
/home/foobar/zyzzyva.pdf is a medium file of size 29328
$
```

ANS

```
#!/bin/bash

function classify ( )
{
   dirname=$1
   for file in `ls $dirname`; do
      fname=$dirname/$file
      if [ -f $fname ] ; then
         lsout=(`ls -l $fname`)
         fsize=${lsout[4]}
         if [ $fsize -le 1024 ]; then
            echo "$fname is a small file of size $fsize"
         elif [ $fsize -le 1048576 ]; then
            echo "$fname is a medium file of size $fsize"
         else
            echo "$fname is a large file of size $fsize"
         fi
      fi
   done
}

if [ $# -eq 0 ]; then echo "Usage: Run with a directory name"; exit 1; fi
if [ ! -r $1 ]; then echo "$1: Permisssion denied"; exit 2; fi
if [ ! -d $1 ]; then echo "$1: Not a directory"; exit 3; fi
classify $1
```

VARIANT 2

Write a bash script classify.sh to do the following.

1. Write a function classify which takes a directory name as argument, and finds the number of entries in each subdirectory in the directory. An entry is any file (a regular file, a directory, or anything else). For each such subdirectory, the function prints the number of entries in the subdirectory, and classify the subdirectory as small (with <= 10 entries), medium (with more than 10 but <= 100 entries), or large (with > 100 entries).
2. If the script is called without any argument, exit with an error message.
3. If the first argument is not readable or not a directory, exit with appropriate error messages.
4. Call the function classify on the first argument.

Assume that no file or directory name contains spaces.

Sample output:

```
$ ./classify.sh ~
/home/foobar/books is a medium directory with 16 entries
/home/foobar/Downloads is a large directory with 295 entries
/home/foobar/snap is a small directory with 3 entries
$
```

ANS

```
#!/bin/bash

function classify ( )
{
   dirname=$1
   for file in `ls $dirname`; do
      fname=$dirname/$file
      if [ -d $fname ] ; then
         lsout=(`ls $fname`);
         dirsize=${#lsout[@]}
         if [ $dirsize -le 10 ]; then
            echo "$fname is a small directory with $dirsize entries"
         elif [ $dirsize -le 100 ]; then
            echo "$fname is a medium directory with $dirsize entries"
         else
            echo "$fname is a large directory with $dirsize entries"
         fi
      fi
   done
}

if [ $# -eq 0 ]; then echo "Usage: Run with a directory name"; exit 1; fi
if [ ! -r $1 ]; then echo "$1: Permisssion denied"; exit 2; fi
if [ ! -d $1 ]; then echo "$1: Not a directory"; exit 3; fi
classify $1
```

VARIANT 3

Write a bash script classify.sh to do the following.

1. Write a function classify which takes a directory name as argument, and computes the total size (in bytes) of all the regular files residing in the directory. Depending upon the computed total size, it classifies the directory as small (total size <= 1Kb bytes), medium (total size > 1Kb but <= 1Mb), or large (total size > 1Mb).
2. If the script is called without any argument, exit with an error message.
3. If the first argument is not readable or not a directory, exit with appropriate error messages.
4. Call the function classify on the first argument.

Assume that no file or directory name contains spaces. In order to get the size of a file, use ls -l or wc -c. But these commands print some text in addition to the size.

Sample output:

```
$ ./classify.sh ~
Total file size in /home/foobar is 31725615
/home/foobar is a large directory
$ ./classify.sh ~/.ssh/
Total file size in /home/foobar/.ssh/ is 666
/home/foobar/.ssh/ is a small directory
$ ./classify.sh ~/spl
Total file size in /home/foobar/spl/ is 19354
/home/foobar/spl/ is a medium directory
$
```

ANS

```
#!/bin/bash

function classify ( )
{
   dirname=$1
   declare -i totalsize=0
   for file in `ls $dirname`; do
      fname=$dirname/$file
      if [ -f $fname ] ; then
         lsout=(`ls -l $fname`)
         fsize=${lsout[4]}
         totalsize=$(( totalsize + fsize ))
      fi
   done
   echo "Total file size in $dirname is $totalsize"
   if [ $totalsize -le 1024 ]; then
      echo "$dirname is a small directory"
   elif [ $totalsize -le 1048576 ]; then
      echo "$dirname is a medium directory"
   else
      echo "$dirname is a large directory"
   fi
}

if [ $# -eq 0 ]; then echo "Usage: Run with a directory name"; exit 1; fi
if [ ! -r $1 ]; then echo "$1: Permisssion denied"; exit 2; fi
if [ ! -d $1 ]; then echo "$1: Not a directory"; exit 3; fi
classify $1
```