ASSIGNMENT NO:4 **DATE: / /2015**

PROGRAM TITLE:Implement a hash table using:

1. **Division method**
2. **Mid Square method**
3. **Folding method**

Now implement another hashing function of your own and compare the performance of all the four methods in terms of number of collisions.

PROGRAM ALGORITHM:

```
divmethod(d[],x)
{
      add=x%arraysize;      //which is 100 in this program
      call place(d,add,x);
}
midsqmethod(m[],x)
{
      square x;
      t=middle 2 digits of x;    //as arraysize has been taken 100
      call place(m,t,x);
}
foldmethod(f[],x)
{
      partition the number into groups of 2 from the right and add them up;
      get 2 digit index from the sum and store in s;
      call place(f,s,x);
}
usermethod(u[],x)
{
      t=x,s=0;
      while(t not equal to 0)
      {
            s=s+(t%97);      //replace 97 by the greatest prime before arraysize
            t=t/97;
      }
      s=take rightmost 2 digits from s;
      call place(u,s,x);
}
place(a[],add,x)
{
      if(a[add]is empty)
      {
            a[add]=x;
      }
      else
      {
            add=find nearest empty place by linear probing;
            if(add is a valid index)
                  a[add]=x;
            else
                  printf("\n\tHash Table is full.");
      }
}
```

PROGRAM CODE:

```
#include <stdio.h>
#define ARRSIZE 100
#define DIG 2//number of digits of the index
int methdiv(int a[],int n);//function to implement the division method, works
on d
int methmidsq(int a[],int n);//function to implement the mid-square method,
works on m
int methfold(int a[],int n);//function to implement the folding method, works
on f
int methuser(int u[],int x);//user defined hash function
int place(int a[],int index,int n);
int collres(int a[],int index);//sends the array and the index where it is
colliding, returns an empty index or -1 if the array is full. Searches by
linear probing.
int upow(int x,int y);
int main()
{
	int i,n=1,d[ARRSIZE]={0},m[ARRSIZE]={0},f[ARRSIZE]={0},u[ARRSIZE]={0};
	int cd=0,cm=0,cf=0,cu=0,count=0;
	printf("\n\tONLY POSITIVE NON-ZERO NUMBERS. Entering any negative number
will stop taking inputs.\n");
	printf("\tEnter the numbers::");
	while((n>0)&&(count<ARRSIZE))
	{
		scanf("%d",&n);
		if(n<=0)
			break;
		cd+=methdiv(d,n);
		cm+=methmidsq(m,n);
		cf+=methfold(f,n);
		cu+=methuser(u,n),
		count++;
	}
	printf("\n\tThe final individual hash tables
are::\n\tIndex\tDivision\tMid-Square\tFolding\t\tUser");
	for(i=0;i<ARRSIZE;i++)
	{
		printf("\n\t%d|\t%d\t\t%d\t\t%d\t\t%d",i,d[i],m[i],f[i],u[i]);
	}
	printf("\nCollisions:\t%d\t\t%d\t\t%d\t\t%d\n",cd,cm,cf,cu);
	return 0;
}
int methdiv(int d[],int x)
{
	int add=x%ARRSIZE;
	return(place(d,add,x));
}
int methmidsq(int m[],int x)
{
	int i=0;
	unsigned long int t;
	t=x*x;
	while(t!=0)
	{
		t=t/10;
```

```
            i++;
      }
      t=x*x;
      i=i-DIG;
      i=i/DIG;
      t=t/upow(10,i);
      t=t%upow(10,DIG);
      return(place(m,(int)t,x));
}
int methfold(int f[],int x)
{
      int t=x,s=0;
      while(t!=0)
      {
            s=s+(t%upow(10,DIG));
            t=t/upow(10,DIG);
      }
      s=s%upow(10,DIG);
      return(place(f,s,x));
}
int methuser(int u[],int x)
{
      int t=x,s=0;
      while(t!=0)
      {
            s=s+(t%97);//replace 97 by the greatest prime before ARRSIZE
            t=t/97;
      }
      s=s%upow(10,DIG);
      return(place(u,s,x));
}
int place(int a[],int add,int x)
{
      if(a[add]==0)
      {
            a[add]=x;
            return 0;
      }
      else
      {
            add=collres(a,add);
            if(add!=-1)
                  a[add]=x;
            else
                  printf("\n\tHash Table is full.");
            return 1;
      }
}
int collres(int a[],int index)
{
      int i,j;
      for(i=index,j=index;(i>=0)||(j<ARRSIZE);i--,j++)
      {
            if(i>=0 && a[i]==0)
                  return i;
            if(j<ARRSIZE && a[j]==0)
                  return j;
```

```
     }
     return -1;
}
int upow(int x,int y)
{
     int i,s=1;
     for(i=0;i<y;i++)
     {
     s=s*x;
     }
     return s;
}
```

OUTPUT:

```
     ONLY POSITIVE NON-ZERO NUMBERS. Entering any negative number will stop
taking inputs.
     Enter the numbers::87460 79983 02414 08563 36998 78251 81051 99209 18177
08159 64770 94937 30994 69706 06686 95746 75451 36335 53684 54752 51548 61161
32321 18840 32726 91069 16040 14568 45543 52616 57957 41935 90001 19245 04954
74383 39539 44071 83592 38471 47275 73979 93869 34199 60093 35526 09227 61565
04550 02155 56225 20572 54089 26981 34862 84660 35267 56888 45139 80387 74642
18436 60497 07754 92541 85064 07495 40203 68638 09891 86133 04291 49816 39798
96409 00018 62634 01345 11768 27385 07091 67974 85324 86392 83187 92800 37403
50480 64365 22708 47680 46207 77183 84255 89596 73574 21640 50884 36594 06023

     The final individual hash tables are::
     Index Division   Mid-Square Folding         User
     0|    92800      4550       16040      87460
     1|    90001      38471      36335      35526
     2|    37403      6686       45543      44071
     3|    40203      18177      4954       52616
     4|    6023       6023       54752      16040
     5|    46207      92800      40203      61161
     6|    69706      68638      30994      69706
     7|    22708      20572      60497      78251
     8|    96409      46207      90001      9227
     9|    99209      35526      69706      26981
     10|   36594      36594      99209      2414
     11|   50884      50884      75451      79983
     12|   21640      4291       95746      34862
     13|   73574      21640      34862      18436
     14|   2414       73574      14568      45139
     15|   49816      89596      44071      8563
     16|   52616      45543      93869      53684
     17|   89596      84255      84660      60497
     18|   18         77183      86133      85064
     19|   84255      47680      9227       34199
     20|   77183      22708      85064      7495
     21|   32321      18         18436      86133
     22|   47680      7754       35267      96409
     23|   85324      36335      64770      81051
     24|   56225      8563       73979      18840
     25|   35526      78251      53684      61565
     26|   32726      14568      49816      18
     27|   9227       44071      18         20572
```

```
28|  64365      30994      83187      54752
29|  50480      40203      18840      18177
30|  83187      9891       68638      80387
31|  62634      79983      7754       35267
32|  86133      64365      4291       4550
33|  68638      39539      74383      95746
34|  41935      50480      54089      36998
35|  36335      34862      83592      45543
36|  18436      37403      39539      54089
37|  94937      9227       19245      99209
38|  39539      27385      2414       62634
39|  16040      83187      8159       27385
40|  18840      2155       78251      67974
41|  45139      86392      57957      2155
42|  74642      4954       87460      47275
43|  45543      85324      34199      36335
44|  92541      67974      92800      19245
45|  19245      18840      64365      90001
46|  95746      74642      52616      74642
47|  1345       7091       32321      83187
48|  51548      96409      8563       64365
49|  4550       99209      22708      46207
50|  81051      49816      84255      89596
51|  78251      86133      47275      22708
52|  75451      45139      6686       37403
53|  54752      93869      26981      11768
54|  4954       41935      77183      39798
55|  2155       92541      67974      73979
56|  7754       80387      32726      56888
57|  57957      34199      38471      91069
58|  84660      56888      41935      6686
59|  8159       69706      18177      4954
60|  87460      84660      1345       92800
61|  61161      95746      56888      56225
62|  34862      56225      27385      64770
63|  8563       47275      7091       84255
64|  85064      32321      86392      47680
65|  61565      19245      47680      32321
66|  35267      57957      62634      73574
67|  93869      52616      7495       4291
68|  14568      51548      51548      83592
69|  91069      8159       81051      57957
70|  64770      53684      36998      7754
71|  44071      81051      73574      38471
72|  38471      87460      21640      14568
73|  20572      2414       46207      93869
74|  11768      64770      36594      40203
75|  47275      54752      92541      74383
76|  67974      61161      2155       85324
77|  18177      91069      6023       21640
78|  86392      90001      61161      50884
79|  73979      83592      20572      41935
80|  7091       60093      37403      84660
81|  26981      16040      50884      50480
82|  74383      61565      96409      30994
83|  79983      54089      85324      39539
84|  53684      26981      35526      7091
```

```
      85|  27385      36998      11768      36594
      86|  6686       74383      61565      32726
      87|  80387      35267      50480      86392
      88|  56888      75451      91069      49816
      89|  54089      18436      79983      94937
      90|  4291       60497      9891       6023
      91|  9891       85064      89596      75451
      92|  83592      73979      56225      51548
      93|  60093      7495       45139      60093
      94|  30994      39798      4550       92541
      95|  7495       62634      94937      8159
      96|  39798      1345       74642      68638
      97|  60497      11768      39798      1345
      98|  36998      32726      80387      77183
      99|  34199      94937      60093      9891
Collisions:     46         60         44         44
```

DISCUSSION:

1. Linear probing is done in both directions simultaneously for faster collision resolution.
2. The number of collisions differs for each set of input.
3. To change the dimension of the array, only the macros at the top of the code would have to be changed along with a minor change in the user method.