Christopher McDaniel

COSC 3319.01

Submitted: 2 December 2019

Grading Option: "**C**"

----------Linear hash for a 50% full table----------

Minimum number of probes for the first, or last, 30 keys: 1
Maximum number of probes for the first, or last, 30 keys: 30
Average number of probes for the first, or last, 30 keys: 15

Minimum number of probes for the first, or last, 30 keys: 1
Maximum number of probes for the first, or last, 30 keys: 64
Average number of probes for the first, or last, 30 keys: 49

Theoretical expected number of probes to locate a random item: 1.50000E+00

1. 1234567890123456 Hash Address: 1 Probes: 1
2. Aguirrie Hash Address: 1 Probes: 2
3. Alcantara Hash Address: 1 Probes: 3
4. Bhandari Hash Address: 1 Probes: 4
5. Carmona Hash Address: 1 Probes: 5
6. Casper Hash Address: 1 Probes: 6
7. Cook Hash Address: 1 Probes: 7
8. Daniels Hash Address: 1 Probes: 8
9. Nienberg Hash Address: 1 Probes: 9
10. Paschal Hash Address: 1 Probes: 10
11. Red Hash Address: 1 Probes: 11
12. Salkowski Hash Address: 1 Probes: 12
13. Zulfiqar Hash Address: 1 Probes: 13
14. Qamruddin Hash Address: 1 Probes: 14
15. Acevedo Hash Address: 1 Probes: 15
16. Ajose Hash Address: 1 Probes: 16
17. Arauza Hash Address: 1 Probes: 17
18. Buck Hash Address: 1 Probes: 18
19. Clark Hash Address: 1 Probes: 19
20. Crouch Hash Address: 1 Probes: 20
21. Davies Hash Address: 1 Probes: 21
22. Dugger Hash Address: 1 Probes: 22
23. Egbe Hash Address: 1 Probes: 23
24. Ellington Hash Address: 1 Probes: 24
25. Farral Hash Address: 1 Probes: 25
26. Garza Hash Address: 1 Probes: 26
27. Gurung Hash Address: 1 Probes: 27
28. Joseph Hash Address: 1 Probes: 28
29. Kelly Hash Address: 1 Probes: 29
30. Corey Hash Address: 1 Probes: 30
31. Adam Hash Address: 1 Probes: 31
32. Clayton Hash Address: 1 Probes: 32
33. Dustin Hash Address: 1 Probes: 33
34. Robert Hash Address: 1 Probes: 34
35. Kyle Hash Address: 1 Probes: 35
36. Scott Hash Address: 1 Probes: 36
37. Octavio Hash Address: 1 Probes: 37
38. Judy Hash Address: 1 Probes: 38
39. Derek Hash Address: 1 Probes: 39
40. Jeffrey Hash Address: 1 Probes: 40
41. Jordon Hash Address: 1 Probes: 41
42. Vinnela Hash Address: 1 Probes: 42

43. Lisa Hash Address: 1 Probes: 43
44. Todd Hash Address: 1 Probes: 44
45. Veronica Hash Address: 1 Probes: 45
46. Matthew Hash Address: 1 Probes: 46
47. Michael Hash Address: 1 Probes: 47
48. Akhila Hash Address: 1 Probes: 48
49. John Hash Address: 1 Probes: 49
50. Charles Hash Address: 1 Probes: 50
51. James Hash Address: 1 Probes: 51
52. Chris Hash Address: 1 Probes: 52
53. Wade Hash Address: 1 Probes: 53
54. Christopher Hash Address: 1 Probes: 54
55. Fernando Hash Address: 1 Probes: 55
56. Batbold Hash Address: 1 Probes: 56
57. Joel Hash Address: 1 Probes: 57
58. Fabulous Hash Address: 1 Probes: 58
59. Misogamist Hash Address: 1 Probes: 59
60. Maiden Hash Address: 1 Probes: 60
61. Eye Hash Address: 1 Probes: 61
62. Constriction Hash Address: 1 Probes: 62
63. Necromancer Hash Address: 1 Probes: 63
64. Syncopate Hash Address: 1 Probes: 64
65. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
66. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
67. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
68. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
69. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
70. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
71. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
72. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
73. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
74. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
75. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
76. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
77. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
78. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
79. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
80. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
81. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
82. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
83. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
84. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
85. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
86. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
87. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
88. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
89. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
90. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
91. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
92. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
93. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
94. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
95. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
96. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0

97. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
98. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
99. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
100. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
101. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
102. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
103. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
104. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
105. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
106. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
107. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
108. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
109. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
110. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
111. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
112. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
113. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
114. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
115. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
116. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
117. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
118. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
119. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
120. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
121. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
122. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
123. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
124. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
125. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
126. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
127. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0
128. &&&&&&&&&&&&&&&& Hash Address: 0 Probes: 0

~~Random hash for a 50% full table~~

~~raised CONSTRAINT_ERROR : main.adb:154 index check failed~~

----------Main.adb----------

```
with Ada.Text_IO; use Ada.Text_IO;
with Ada.Integer_Text_IO; use Ada.Integer_Text_IO;
with Ada.Float_Text_IO; use Ada.Float_Text_IO;
with Ada.Long_Integer_Text_IO; use Ada.Long_Integer_Text_IO;
with Ada.Numerics.Generic_Elementary_Functions;
with Ada.Unchecked_Conversion;

--Table Size = 128
procedure Main is

   --pragma Suppress(Overflow_Check); --Check for numeric overflow. Constraint_Error
suppressed.

   --Global variables.
   TempHash : Long_Integer;
   TempKey : String(1..16);
   InputFile: File_Type;

   MArray: array (1..128) of Integer; --p: array

   type InputData is record
      Key : String(1..16) := "&&&&&&&&&&&&&&&&"; --Empty location
      HA : Long_Integer := 0; --Hash Address.
      Probes : Integer := 0; --Number of probes.
   end record;

   Table: array (1..128) of InputData;

   String16: String(1..16);

   --Convert character to integer.
   function ConvertString16 is new Ada.Unchecked_Conversion(String, Long_Integer);

   package Logarithm is new Ada.Numerics.Generic_Elementary_Functions(Float);
   use Logarithm;

   -------------------------Functions---------------------------
   --Burris Hash function.
   function BurrisHash(X: in String) return Long_Integer is
      Sum : Long_Integer;
      Divider : Long_Integer := 65535;
   begin
      --HA = {abs[ slice(4..5) ] + abs[slice(13..14) ] } / 65,535 + abs[slice(10..10)]
      Sum := (ConvertString16(String16(4..5))) + (ConvertString16(String16(13..14)));
      Sum:= (Sum / Divider);
      Sum:= (Sum + (ConvertString16(String16(10..10))));

      TempHash := ((abs(Sum)) mod 128 + 1); --Folding remainder with table size 128.
      return TempHash;
   end BurrisHash;
   -------------------------Procedures-------------------------
```

```
--Displays the table.
--Uses records from InputData.
procedure Display is
   Num : Integer := 1;

begin
   for K of Table
   loop
      put(Num, 0);
      put(".    "); --#.
      put(K.Key);
      put("  Hash Address:"); --#
      put(K.HA);
      put("  Probes:"); --#
      put(K.Probes);
      New_Line(1);
      Num := Num + 1; --Increase Address number (1-128).
   end loop;
end Display;

--Empties the table for next run through.
--Resets to original InputData record.
procedure Empty is
begin
   for K of Table
   loop
      K.Key := "&&&&&&&&&&&&&&&&&&"; --Empty location.
      K.HA := 0; --Hash address is 0.
      K.Probes := 0; --Number of probes is 0.
   end loop;
end Empty;

--Inserts linear probe.
--Page 144 and 163 of DS Notes.
procedure InsertLinear (CheckHA: in Long_Integer; Insert: in out InputData) is
   TempHA : Integer;

begin
   --If TempHA = 0, set it to 1.
   if TempHA = 0 then
      TempHA := 1;
      Insert.HA := 1;
   end if;

   --If an empty spot is found, insert the Insert.
   if Table(TempHA).HA = 0 then
      Insert.Probes := 1;
      Table(TempHA) := Insert;
   else --Keep looking.
      Insert.Probes := 1;
      while Table(TempHA).HA /= 0
      loop --A collision has occured. Continue linearly until empty spot is found.
         Insert.Probes := Insert.Probes + 1;
         if TempHA < 128 then --HA doesn't exceed the size of the table.
```

```
          TempHA := TempHA + 1;
        else
          TempHA := 1;
        end if;
      end loop;
      Table(TempHA) := Insert;
    end if;
end InsertLinear;

--Generates random numbers for use with random probe.
--Page 160 of DS Notes.
procedure GenerateRandomProbe is
   R, K : Integer := 1;
   Done : Boolean := False;
begin
   while not Done
   loop
      R := (5 * R);
      R := (R mod(2**9)); --Table size of 128
      if R = 1 then --Initialization seed.
        Done := True; --Return initial seed.
      end if;
      MArray(K) := (R / 4); --Set random number = (R/4).
      K := K + 1;
   end loop;
end GenerateRandomProbe;

--Inserts random probe.
--Page 144, 162, and 163 of DS Notes.
procedure InsertRandom (CheckHA: in Long_Integer; Insert: in out InputData) is
   TempHA : Integer;
   Root : Integer;
   K : Integer := 1;

begin
   --If TempHA = 0, set it to 1.
   if TempHA = 0 then
      TempHA := 1;
      Insert.HA := 1;
   end if;

   --If an empty spot is found, insert the Insert.
   if Table(TempHA).HA = 0 then
      Insert.Probes := 1;
      Table(TempHA) := Insert;
   else --Keep looking.
      Insert.Probes := 1;
      while Table(TempHA).HA /= 0
      loop --A collision has occured. Generate a random offset and try again.
        Insert.Probes := Insert.Probes + 1;
        TempHA := Root + MArray(K);
        if TempHA > 128 then --Wrap around HA exceeds size of the table.
          TempHA := TempHA - 128;
        end if;
```

```
        K := K + 1;
      end loop;
      Table(TempHA) := Insert;
    end if;
end InsertRandom;

--Finds and prints the Minimum, Maximum, and Average number of probes.
procedure MinMaxAvg (Start: in Integer) is
   TTL : Integer := 0; --Total.
   AVG : Integer := 0; --Average.
   MAX : Integer := 0; --Maximum.
   MIN : Integer := 1; --Minimum.
   KeyNum : Integer := 30;
   K : Integer := 1;

begin
   if Start /= 1 then
      K := Start;
      while KeyNum /= 0
      loop
        if Table(K).Key /= "&&&&&&&&&&&&&&&&" then

          TTL := TTL + Table(K).Probes; --Total number of probes.

          if Table(K).Probes < MIN then
            MIN := Table(K).Probes; --Minimum number of probes.
          end if;

          if Table(K).Probes > MAX then
            MAX := Table(K).Probes; --Max number of probes.
          end if;

          K := K - 1;
          KeyNum := KeyNum - 1;

        else
          K := K - 1;
        end if;
      end loop;

   else
      while KeyNum /= 0
      loop
        if Table(K).Key /= "&&&&&&&&&&&&&&&&" then

          TTL := TTL + Table(K).Probes; --Total number of probes.

          if Table(K).Probes < MIN then
            MIN := Table(K).Probes; --Minimum number of probes.
          end if;

          if Table(K).Probes > MAX then
            MAX := Table(K).Probes; --Max number of probes.
          end if;
```

```
            K := K + 1;
            KeyNum := KeyNum - 1;

         else
            K := K + 1;
         end if;
      end loop;
   end if;

   AVG := (TTL / 30);
   put("Minimum number of probes for the first, or last, 30 keys: ");
   put(MIN, 0);
   New_Line(1);
   put("Maximum number of probes for the first, or last, 30 keys: ");
   put(MAX, 0);
   New_Line(1);
   put("Average number of probes for the first, or last, 30 keys: ");
   put(AVG, 0);
   New_Line(1);

end MinMaxAvg;

--Calculate the theoretical expected number of Linear probes.
--DS notes page 157.
procedure TheoreticalLinear (X: in Float) is
   alpha : Float := (X / Float(128)); --alpha = (number keys in the table) / (table size).
   Sum : Float;

begin

   Sum := ((1.0 - alpha / 2.0) / (1.0 - alpha)); --E = (1-alpha/2)/(1-alpha).
   put("Theoretical expected number of probes to locate a random item: ");
   put(Sum);
end TheoreticalLinear;

--Calculate the theoretical expected number of Random probes.
--DS notes page 159.
procedure TheoreticalRandom (X: in Float) is
   alpha : Float := (X / Float(128)); --alpha = (number keys in the table) / (table size).
   Sum : Float;

begin

   Sum := (-(1.0 / alpha) * Log(1.0 - alpha)); --E = -(1/alpha) log(1-alpha).
   put("Theoretical expected number of probes to locate a random item: ");
   put(Sum);

end TheoreticalRandom;

--Creates table with X percentage full.
procedure Fill (X: Integer; LinearRandom: Integer) is
   PtV : Integer;
begin
```

```
    PtV := (128 * X / 100);
    --Opens "Words200D16.txt" file.
    Open(File => InputFile,
        Mode => In_File,
        Name => "Words200D16.txt");

    for K in 1..PtV
    loop
      declare
        InData : InputData;
      begin
        TempKey := Get_Line(InputFile);
        InData.Key := TempKey;
        InData.HA := BurrisHash(TempKey);
        if LinearRandom = 1 then --Linear approach.
          InsertLinear(InData.HA, InData);
        elsif LinearRandom = 2 then --Random approach.
          InsertRandom(InData.HA, InData);
        end if;
      end;
    end loop;

    --Closes "Words200D16.txt" file.
    Close(File => InputFile);

    --Checks if InputFile is already open and closes it if it is.
  exception
    when End_Error =>
      if Is_Open(File => InputFile) then
        Close(File => InputFile);
      end if;

  end Fill;

begin
  --50% Linear.
  put("Linear hash for a 50% full table");
  New_Line(2);
  Fill(50, 1);
  MinMaxAvg(1);
  New_Line(1);
  MinMaxAvg(128);
  New_Line(1);
  TheoreticalLinear(64.0);
  New_Line(2);
  Display;
  Put("------------------------------------------------------");
  New_Line(1);

  Empty;

  --50% Random.
  put("Random hash for a 50% full table");
  New_Line(2);
```

```
   Fill(50, 2);
   MinMaxAvg(1);
   New_Line(1);
   MinMaxAvg(128);
   New_Line(1);
   TheoreticalRandom(64.0);
   New_Line(2);
   Display;
   Put("-----------------------------------------------------");
   New_Line(1);

   Empty;

   --90% Linear.
   put("Linear hash for a 90% full table");
   New_Line(2);
   Fill(90, 1);
   MinMaxAvg(1);
   New_Line(1);
   MinMaxAvg(128);
   New_Line(1);
   TheoreticalLinear(115.0);
   New_Line(2);
   Display;
   Put("-----------------------------------------------------");
   New_Line(1);

   Empty;

   --90% Random.
   put("Random hash for a 90% full table");
   New_Line(2);
   Fill(90, 2);
   MinMaxAvg(1);
   New_Line(1);
   MinMaxAvg(128);
   New_Line(1);
   TheoreticalRandom(115.0);
   New_Line(2);
   Display;
   Put("-----------------------------------------------------");
   New_Line(1);

end Main;
```

Hashing Lab Report

Given function: HA = {abs[slice(4..5)] + abs[slice(13..14)] } / 65,535 + abs[slice(10..10)]

A.

1. Tasks:
    i. Create a hash table in main memory and fill it 50% full. Use the linear probe technique developed in class to handle collisions.

ii. Now look up the first 30 entries placed in the table. Print the minimum, maximum, and average number of probes required to locate the first 30 keys placed in the table.
iii. Now search for the last 30 keys placed in the table. Print the minimum, maximum, and average number of probes required to locate the last 30 keys placed in the table.
iv. Print the contents of the hash table clearly indicating open entries (this should allow you to see the primary/secondary clustering effect).
v. Calculate and print the theoretical expected number of probes to locate a random item in the table.
vi. Explain your empirical results in light of the theoretical results. **Your grade points will be highly dependent on your explanation!**

2. Response:
   i. When using the linear probe, clustering is to be expected, and with the given hash function the clustering compounds onto itself. The result is a large primary cluster, about, in the middle of the table.
   ii. For the first 30 keys the average probe count is ??? and for the final 30 keys it's ???, these results are worse than the expected results of 1.50 probes per key, according to the DS Notes on page 157. This proves the point that the given hash function was extraordinarily poor and far from being ideal.
   iii. The lack of variance between the values shows just how terrible the given function was at producing unique values.

B.

1. Tasks:
   i. Create a hash table in main memory and fill it 90% full. Use the linear probe technique developed in class to handle collisions.
   ii. Now look up the first 30 entries placed in the table. Print the minimum, maximum, and average number of probes required to locate the first 30 keys placed in the table.
   iii. Now search for the last 30 keys placed in the table. YOU MUST PHYSICALLY SEARCH FOR EACH KEY TO CALCULATE THE STATISTICS! Print the minimum, maximum, and average number of probes required to locate the last 30 keys placed in the table.
   iv. You must start filling the table with the same keys in the same order as used when only filling the table 50% full. Print the contents of the table clearly indicating open entries (this should allow you to see the primary clustering effect).
   v. Calculate and print the theoretical expected number of probes to locate a random item in the table.
   vi. Discuss your empirical results in light of the theoretical results. **Your grade points will be highly dependent on your explanation!**
2. Response:

i. Similar to the previous response, the given hash function shows just how poorly it performs as the percentage of being full increases. The data would appear to be in a large primary cluster within the printed hash table.

C.

1. Tasks:
    i. Repeat A and B above but use the random probe for handling collisions as developed in class.
    ii. When you print the contents of the table you should be able to visually see the secondary clustering affect.
    iii. Calculate and print the theoretical expected number of probes to locate a random item in the table.
    iv. Discuss your empirical results in light of the theoretical results. **Your grade points will be highly dependent on your explanation!**
2. Response:
    i. Due to random probe generation the keys may have the same initial value, but they will be spread out due to the randomization. One can see this because of the secondary clustering that will occur within the hash table. Since the keys are mainly going to the same value, the random probe generation technique, unfortunately, is not a beneficial technique to use for this. The main requirement for the random number generator is that results can be repeated as needed so as to make it easier and more efficient to search the hash table.
    ii. Unless the random number generator's seed is updated, the so-called "Random Offsets" will be continuously generated in the same order, causing the hash addresses to have the same sequences over and over with every search. Just like the linear probe, 2 keys can hash to the same value and will search through the same order of locations until an empty spot is reached. Instead of being sequential, the resulting hash locations that are accessed will be random.
    iii. Since the given function is so inefficient, the random probes efficiency is only slightly higher than that of the linear probe, when normally the random probe technique is far superior. In order to improve the average number of probes needed, the keys need to be varied and combined with the random number generator.

D.

1. Tasks:
    i. Present all your results in a neat tabular format (typed naturally).
    ii. Compare your results to the theoretical results. If there are differences, please explain why.
    iii. I expect you to physically calculate the theoretical values for comparison to your empirical results for both the linear and random probes! You may not ask a friend, you may not look them up in some table! Providing someone else with the results will not only cause them to fail the lab but you as well. You may however teach them to use a calculator, spreadsheet, log tables, or other techniques.
2. Response:

i. Figure 1.

	Burris Hash			**First 30**	(Sum/Knt)		Last 30			
Percent Full	Probe	Storage	**Min.**	**Max.**	**Avg.**	Min.	Max.	Avg.	**Probes**	**Keys**
50%	Linear	Mem.	**1**	**30**	**15.5**	….	….	….	….	….
50%	Linear	File	**1**	**30**	**15.5**	….	….	….	….	….
50%	Random	Mem.	**1**	**30**	**15.5**	….	….	….	….	….
50%	Random	File	**1**	**30**	**15.5**	….	….	….	….	….
90%	Linear	Mem.	**1**	**30**	**15.5**	….	….	….	….	….
90%	Linear	File	**1**	**30**	**15.5**	….	….	….	….	….
90%	Random	Mem.	**1**	**30**	**15.5**	….	….	….	….	….
90%	Random	File	**1**	**30**	**15.5**	….	….	….	….	….

ii. The table shows that since the function is using the same hash for multiple inputs, there is no difference between a linear and random probe technique. As previously gone over, when unique values are introduced into the function, the average number of random probes needed to find the address improve by a hair over the linear probes average, but not enough to really matter. The theoretical expected average of probes and actual average of probes have one glaring difference, in that, based on the given load (Percent full) it can be seen that the hash is no where near efficient for the given set of data and wouldn't be able to produce an ample amount of unique hash values.

E. I was not able to get this far in my program.

1. Tasks:
   i. My required hash function has multiple weaknesses. Criticize the hash function on a technical basis. To receive full credit, you must state clearly and explicitly state why my hash function should fail!
   ii. Based on your criticism, write a better hash function. You must explain explicitly why your hash function should be better from both a theoretical and empirical standpoint. You will not receive credit for simply writing a hash function that performs better.
   iii. Implement the hash function and generate the same results as required for parts A thru D presented in tabular format for comparison.
   iv. Formally evaluate the results as part of your lab (typed evaluation). The results for all parts of the lab should appear in a single table. Shame on you if your hash function's performance does not exceed the theoretical values for both the linear and random probe collision handling techniques. Script Kiddies may have trouble evaluating the failures of my hash function and creating a better function.
2. Response:
   i. The given function "HA = {abs[slice(4..5)] + abs[slice(13..14)] } / 65,535 + abs[slice(10..10)]" uses…
   ii. The function did not account for the characteristics of the expected keys and could be improved upon by utilizing a different modulas?
   iii. The comparison for of my hash function and the given one….