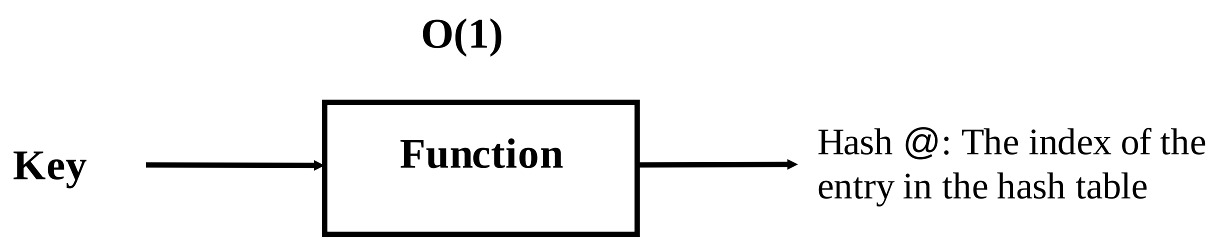
Hashing

- **Goal:**
  - Perform insertions, deletions, and search in constant time: O(1).
  - Example:
    - Many compilers use Hashing to implement their symbol tables

- **Hash Tables:**
  - A hash table is an ADT where insertion, deletion, and search take a constant time: O(1).



  - Synonyms: A set of all keys that maps to the same hash @

  - Let HT be an array [0..b-1] a table of size b and $x_1, x_2, ..., x_n$ are n entities to be stored in HT

  - Question:
    - Where should xi be stored in HT so that it can be found later in O(1)?
  - Answer:
    - Find a function F called hash function such that

$$0 \leq F(x_i) \leq b-1$$

x_i is stored in $HT[F(x_i)]$.

- Example 1:
  - Given a set of elements: ab, bit, chair, table, zebra, group
  - Hash function:

F(word) = (rank of the first letter of "word" in the alphabet) mod 7

F(ab) = 1 mod 7 = 1
F(bit) = 2 mod 7 = 2
F(chair) = 3 mod 7 = 3
F(table) = 20 mod 7 = 6
F(zebra) = 26 mod 7 = 5
F(group) = 7 mod 7 = 0

0	group
1	ab
2	bit
3	chair
4	
5	zebra
6	table

- Example 2:
  - Given a set of elements: Given a set of numbers: 12, 100015, 7533457, 4
  - Hash function:

    F(x) = x mod 5

F(12) = 2 = 1	0	100015
F(100015) = 0	1	12
F(7533457) = 2	2	7533457
	3	
F(4) = 4	4	4

- In general the size of the hash table is much smaller that the set the inputs: How to handle the insertion of synonyms?

Collision Problems !!!!

- Operations:
  - Insert(x):

    Compute F(x)
    insert x in HT: HT[F(x)] = x;

  - Delete(x):

    Compute F(x)
    delete x from HT: HT[F(x)] = 0;

  - Search(x):

    Compute F(x)
    return(HT[F(x)]);

- Hashing Requirements:

  - A hashing function
  - A collision resolution policy

  - Hashing functions

    - Mid-square: F(x) = middle k digits in x^2.

    - Division: F(x) = x mod M

      - The best value of M is prime.
    - Folding: Given a Key $x_1x_2...x_r$

      - $F1(x_1x_2...x_r) = x_1x_2+x_3x_4+...+x_{r-1}x_r$

      - $F2(x_1x_2...x_r) = x_2x_1+x_4x_3+...+x_rx_{r-1}$

- Example: X= 251367

    - F1(K) = 25+13+67 = 125
    - F2(K= 52+31+76 = 159

- Truncation:
    - F(x) = last k digits of x or first k digits of x
        - IF the student ID of a student is G123456789 and we consider the last k=3 digits:

          F(G123456789) = 789

- **Collision resolution policies**

    - **Definition**:
        - If one of the synonyms of a key x is already stored in the hash table and another element y of the synonym set arrives. What would be the alternative entry in the hash table?

    - There are two methods of **collision handling**:
        - Open addressing or linear probing
        - Chaining or bucket hashing

    - **Open addressing** or **linear probing**

        - Insert x in the next available entry following HT[F(x)] with a wrap around.
        - Insert(x)

```
begin
        if the has table is full then print("Error");
        else begin
                probe = Compute F(x);
                While(table[probe] is occupied)
                        begin
                                probe = (probe+1) mod M;
                        end
                table[probe] = x; //insert x
end;
```

- **Example**:
  - Given
    - a set of elements: ab, chair, table, zebra
    - A hash table of size M=7

  - F(word) = (rank of the first letter of “word” in the alphabet) mod 7

F(ab) = 1 mod 7 = 1
F(chair) = 3 mod 7 = 3
F(table) = 20 mod 7 = 6
F(zebra) = 26 mod 7 = 5

0	
1	ab
2	
3	chair
4	
5	zebra
6	table

- Insert tool, able, apple?

F(tool) = 20 mod 7 + 1 (collision)
F(able) = 1 mod 7 + 1 (collision)
F(apple) = 1 mod 7 + 3 (collision)

0	tool
1	ab
2	able
3	chair
4	apple
5	zebra
6	table

- **Delete Operation**
  - Empty entry should be classified as either **never occupied** or **formerly occupied**.
  - Example:
    - Let us consider the previous hash table

0	tool
1	ab
2	able
3	chair
4	apple
5	zebra
6	table

    - Delete "table"
      - F(table) = 20 mod 7 = 6
      - If HT[6] = "table" //Yes
        - Then delete "table"

0	tool
1	ab
2	able
3	chair
4	apple
5	zebra
6	

- If we leave the HT[6] to null what would happen if we are trying to delete "tool"?
  - F(table) = 20 mod 7 = 6
  - Check HT[6]?

- It is null. Does that mean that the HT does not have “tool”?
- So instead of setting HT[6] to null when we delete “table”, we need to set it to “Previously Occupied” so we can keep looking for “tool”.

0	tool
1	ab
2	able
3	chair
4	apple
5	zebra
6	**“Previously Occupied”**

- **Problems**:
  - Slower
  - Clustering:
    - A group of synonyms will be located adjacently and mixed together with some official occupants.
    - Too many “Previously Occupied” and “Never Occupied” markers degrades **delete performance**
    - **Rehash** if there are too many markers.

- **Non-linear probing**
  - Insert x in the next available entry following HT[F(x)] with a wrap around.
  - Insert(x)

```
begin
    if the has table is full then print("Error");
    else begin
        probe = Compute F(x);
        While(table[probe] is occupied)
            begin
                probe = (probe+INC(i)) mod M;
            end
        HT[probe] = x; //insert x
end;
```

  - **Example**:
    - Quadratic probing: INC(i) = i^2
    - The insertion is in the next available position: 1, 4, 9, etc.
    - Given a set of elements: chair, table, zebra
    - F(word) = (rank of the first letter of "word" in the alphabet) mod 7

F(chair) = 3 mod 7 = 3
F(table) = 20 mod 7 = 6
F(zebra) = 26 mod 7 = 5

0	
1	
2	
3	chair
4	
5	zebra
6	table

- Insert tool, zoo, apple?

F(tool) = (6 + 1) mod 7 = 0
F(zoo) = (5 + 4) mod 7 = 2
F(apple) = 1 mod 7 = 1

0	tool
1	apple
2	Zoo
3	chair
4	
5	zebra
6	table

- **Double Hashing**
  - Use two hashing functions
  - Distributes keys more uniformly than linear probing does
  - Insert x in the next available entry following HT[F(x)] with a wrap around.
  - Insert(x)

```
begin
        if the has table is full then print("Error");
        else begin
                probe = Compute F1(x);
                offset = Compute F2(x);
                While(table[probe] is occupied)
                        begin
                                probe = (probe+offset) mod M;
                        end
                HT[probe] = x; //insert x
end;
```

  - **Example**
    - Two hash functions:
      - F1(x) = x mod 7
      - F2(x) = 4 - x mod 4
    - Given the following list of elements: 15, 20, 13

      F1(15) = 15 mod 7 = 1
      F1(20) = 20 mod 7 = 6
      F1(13) = 13 mod 7 = 6
      HT[6] is occupied, compute the offset
      Offset = F2(13) 4 – 13 mod 4 = 3

0	**1**	**2**	**3**	**4**	**5**	**6**
	15		**13**			**20**

- **Chaining or bucket hashing**

    - In this strategy insertion of synonyms is in a separate storage (Linked lists)

    - Example:
        - Given a set of elements: 3, 4, 7, 40, 45, 50, 80
        - F(x) = x mod 4