UNIVERSITY OF DIYALA

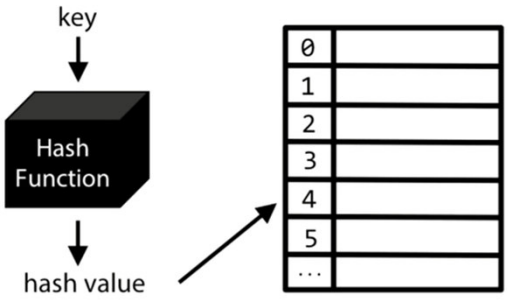
Data Structure

Lecture 5:

Recursion, Hash Table

Instructor

Ali A. Al-Ani



Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

Recursion

- *Recursion is a programming technique in which a function calls itself* ***directly or indirectly.*** *This powerful technique produces repetition without using loops (e.g., while loops or for loops). Thus it can produce substantial results from very little code. Recursion allows elegantly simple solutions to difficult problems. But it can also be misused, producing inefficient code. Recursive code is usually produced from recursive algorithms. The recursive solution for a problem involves a two-way journey:*

  1. ***First we decompose the problem from the top to the bottom***
  2. ***Then we solve the problem from the bottom to the top.***

- *Recursion allows us to write elegant solutions to problems that may otherwise be very difficult to program., e.g. Games of all types like towers of hanoi problem or chess.*

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

Recursion

- ***The Three Laws of Recursion:*** *All recursive algorithms must obey three important laws:*

1. ***A recursive algorithm must have a base case.***

- *For example: consider the following recursive method:*

```
void badPrint(int k)
{
  cout<<k;
  badPrint(k + 1);
}
```


UNIVERSITY OF DIYALA

Recursion

- *Note that a runtime error will occur when the call* ***badPrint(2)*** ***(k = 2)*** *is made (in particular, an error message like* ***"Stack Over flow Error"*** *will be printed, and the program will stop).*
- *This is because there is* ***no code that prevents*** *the recursive call from being made again and again and eventually the program runs out of memory. This is an example of an infinite recursion.*
- *So the role is "Every recursive method must have a* ***base case*** *-- a condition under which no recursive call is made -- to prevent infinite recursion".*

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

Recursion

2. ***A recursive algorithm must change its state and move toward the base case to prevent infinite recursion.*** *Consider this example:*

```
void badPrint(int k) {
    if (k < 0) {      return;      }
    cout<< k;      badPrint2(k + 1);  }
```

- *The above example does have a base case, but the call* ***badPrint(2) (k = 2)*** *will still cause an infinite recursion due to there is no progress toward the base case. So Every recursive method must make progress toward the base case to prevent infinite recursion.*

UNIVERSITY OF DIYALA

Recursion

- ***A recursive algorithm must call itself, recursively:*** *A recursive subroutine is one that calls itself, either directly or indirectly. To say that a subroutine calls itself directly means that its definition contains a subroutine call statement that calls the subroutine that is being defined. To say that a subroutine calls itself indirectly means that it calls a second subroutine which in turn calls the first subroutine (either directly or indirectly).*
- ***Either Directly:***

```
void f() {
  ... f() ...  }
```

- ***Or Indirectly:***

```
void f() {   ... g() ...}
void g() {   ... f() ... }
```

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

Recursion: Factorial Function

- ***A Recursive Implementation of the Factorial Function:*** *To demonstrate the mechanics of recursion, we begin with a simple mathematical example of computing the value of the* ***Factorial Function****.*
- *The factorial of a positive integer* ***n****, denoted* ***n!****, is defined as the product of the integers from* ***1 to n****. If* ***n = 0,*** *then* ***n!*** *is defined as* ***1*** *by convention. More formally, for any integer* ***n>=0****;*
- ***For Example****,* ***5! = 5 * 4 * 3 * 2 * 1 = 120****. The factorial function is important because it is known to equal the number of ways in which* ***n*** *distinct item can be arranged into a sequence, that is, the number of permutations of* ***n*** *items.*

UNIVERSITY OF DIYALA

Recursion: Factorial Function

- *A recursive solution often leads to short and elegant code. Compare the recursive solution with the iterative solution* ***(n = 3)*** *:*
- ***Recursive solution***

```
int factorial(int n){
    if(n == 0)
      return 1;
    else
      return n * factorial(n - 1);
}
```

- ***Iterative solution***

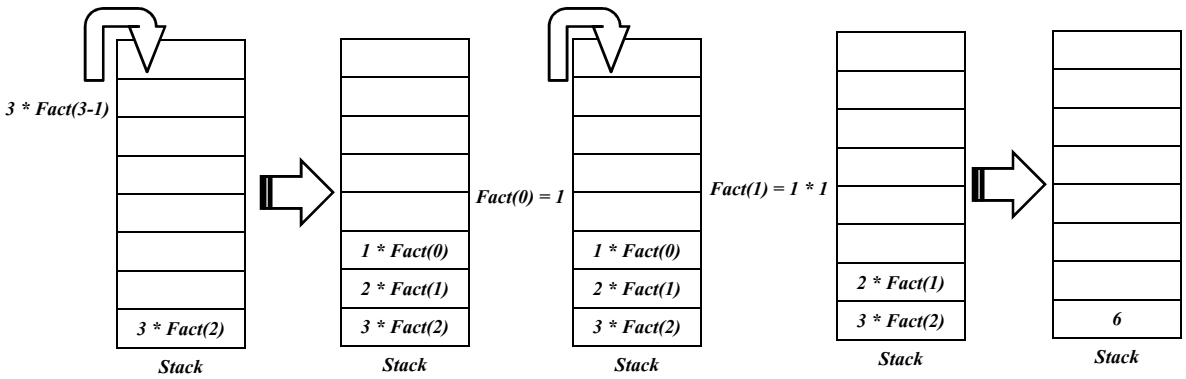
```
int factorial(int n){
    int product = 1;
    while(n >1){
    product * = n;
    n --;   }
  return product;  }
```

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

Recursion: Factorial Function

- *We can illustrate the execution of a recursive function definition by means of a* ***recursion trace****.*



UNIVERSITY OF DIYALA

Recursion: Home Work

1. *Write a recursive procedure to compute the Fibonacci sequence and Graph the resulting (Trace).* ***Noted*** *The Fibonacci numbers are 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, …. Each number after the second is the sum of the two preceding numbers. This is a naturally recursive definition:*

$$Fn \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ f_{n-1} + f_{n-2} & \text{if } n > 1 \end{cases}$$

2. *Write a recursive procedure to compute the* ***greatest common divisor (GCD)*** *algorithm? and Graph the resulting (Trace).*

UNIVERSITY OF DIYALA

Hash Table

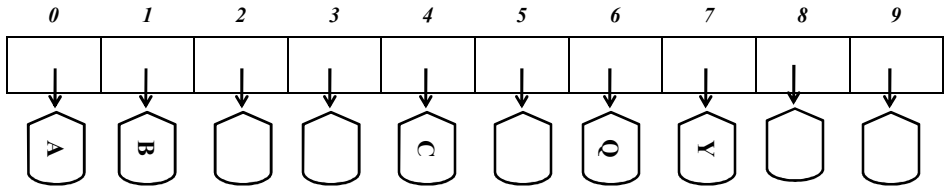
- *A **hash table** (also called a **map**, a **lookup table**, an **associative array**, or a **dictionary**) is a container that allows direct access by any index type.*
- *Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, **where each data value has its own unique index value**. Access of data becomes very fast if we know the index of the desired data.*
- *Examples of such applications include a **compiler's symbol table** and a **registry of environment** variables. Both of these structures consist of a collection of symbolic names where each name serves as the "address" for properties about a variable's type and value.*

UNIVERSITY OF DIYALA

Hash Table

- *In general, a hash table consists of two major components, **a bucket array** and a **hash function**.*

1. *A **bucket array** for a hash table is an array A of size N, where each cell of A is thought of as a "bucket" (that is, a collection of key-value pairs) and the integer N defines the capacity of the array. If the keys are integers well distributed in the range [0,N −1], this bucket array is all that is needed.*



Department of Computer Science
College of Science

Hash Table

- *If our keys are unique integers in the range [0,N −1], then each bucket holds at most one entry. Thus,* ***searches, insertions****, and* ***removals*** *in the bucket array* ***take O(1) time****. This sounds like a great achievement, but it has two drawbacks.*

A. *First, the space used is proportional to N. Thus, if N is much larger than the number of entries n actually present in the map, we have a waste of space.*

B. *The second drawback is that keys are required to be integers in the range [0,N−1], which is often not the case. if there are two or more keys with the same hash value, then two different entries will be mapped to the same bucket in A. In this case, we say that a* ***collision*** *has occurred.*

UNIVERSITY OF DIYALA

Hash Table

2. *A* ***hash function****, A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table. The values returned by a hash function are called* ***hash values****,* ***hash codes****,* ***hash sums****, or simply* ***hashes****. To achieve a good hashing mechanism, It is important to have a good hash function with the following basic requirements:*
   1. *Easy to compute: It should be easy to compute and must not become an algorithm in itself.*
   2. *Uniform distribution: It should provide a uniform distribution across the hash table and should not result in clustering.*
   3. *Less collisions: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided or resolve.*

Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

Hash Table

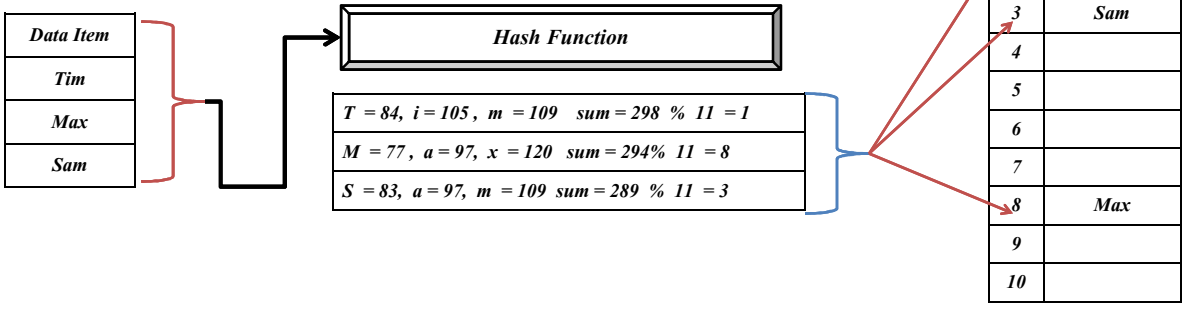
- *A collision occurs when two pieces of data when run through the hash function yield the same hash code. Presumably we want to store both pieces of data in the hash table, so we shouldn't simply overwrite the data that happened to be placed in the first. We need to find a way to get both elements in to hash table while trying to preserve quick insertion and lookup. Various techniques are used to manage this problem:*

1. ***chaining,***
2. ***re-hashing,***
3. ***using neighboring slots (linear probing),quadratic probing, random probing.***

UNIVERSITY OF DIYALA

Hash Table

- ***Ex 1: Suppose we have names items {Tim, Max, Sam}, and the method of determining a hash key is Key = sum ASCII codes Mod (%) Table Size***

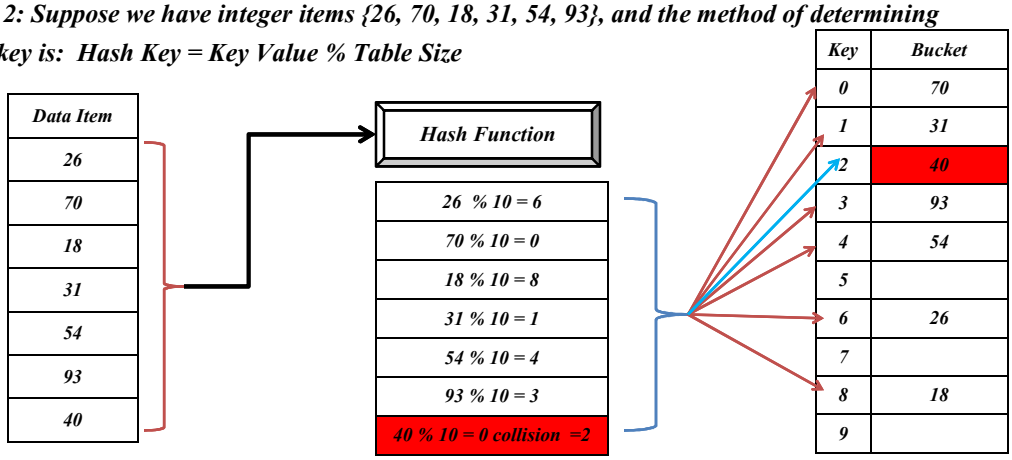


Department of Computer Science
College of Science

UNIVERSITY OF DIYALA

Hash Table

- ***Ex 2: Suppose we have integer items {26, 70, 18, 31, 54, 93}, and the method of determining a hash key is: Hash Key = Key Value % Table Size***



UNIVERSITY OF DIYALA

Hash Table: collision

- *If we insert item* ***(40)*** *in our data items, it would have a hash value of* ***(0) (40 % 10 = 0).*** *But* ***70*** *also had a hash value of* ***0****, it becomes a* ***collision problem****. To solve this problem there are different methods and we will use one of the them that is the Linear probing method:*
- ***Hash Key for (40) = 40 % 10 = 0,*** *Position 0 is occupied by 70. Using Linear Probing method:*
- ***Hash Key (40) = (Hash Key (40) + 1) % table-size → 0 + 1 % 10 = 1. But also the position 1 is occupied by 31, Then we keep try to the next position and so on until we find the empty position:***
- ***Next position = 1 + 1 % 10 = 2 Position 2 is empty, so 40 is inserted there.***

UNIVERSITY OF DIYALA

Table of characters ASCII codes

a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	*j*	*k*	*l*	*m*	*n*
97	*98*	*99*	*100*	*101*	*102*	*103*	*104*	*105*	*106*	*107*	*108*	*109*	*110*
o	*p*	*q*	*r*	*s*	*t*	*u*	*v*	*w*	*x*	*y*	*z*		
111	*112*	*113*	*114*	*115*	*116*	*117*	*118*	*119*	*120*	*121*	*122*		

A	*B*	*C*	*D*	*E*	*F*	*G*	*H*	*I*	*J*	*K*	*L*	*M*	*N*
65	*66*	*67*	*68*	*69*	*70*	*71*	*72*	*73*	*74*	*75*	*76*	*77*	*78*
O	*P*	*Q*	*R*	*S*	*T*	*U*	*V*	*W*	*X*	*Y*	*Z*		
79	*80*	*81*	*82*	*83*	*84*	*85*	*86*	*87*	*88*	*89*	*90*		

UNIVERSITY OF DIYALA

The End

Department of Computer Science
College of Science