Hashes

Introduction - 1/2

- Hash is a computer term used to describe a mechanism for efficiently storing and retrieving values in sparse array
- A **sparse array** is an array that may have indexes from one to a million but contains only a small no of actual values
- A hash table uses an algorithm to determine the index of where to store or retrieve data value

Introduction - 2/2

- Index is called a key and it is linked to the value associated with it
- These keys and values are referred to as **key/value pairs**
- **Hashes do not have last cell index as arrays**
- Starts with **%** symbol

Hash data storage and retrieval

- **Cannot retrieve data from a hash sequentially**

- Hash uses some kind of algorithm to determine where in the array is the best place to store the data

Hash keys

- A hash key is any **literal** that is convenient for us to use to **look up value**

Example

```
%pricelist = ( “Leather bag” => 500.00,
                    “Jacket” => 450.00,
                    “Shirt” => 200.00
                    “Skirt” => 250.00);

printAA(%pricelist);

print “could use hash cell like this $pricelist{‘Jacket’} \n”;

%pricelist = ( 1234=> 500.00,
                  1235 => 450.00,
                   1345 => 200.00
                   1355 => 250.00);

print “or like this $pricelist{1345}\n”;

printAA(%pricelist)
```

```
sub printAA
{
 my %mylist = @_;

 foreach $key (keys(%mylist))
 {
    print "$key price = $pricelist{$key} \n";
 }
}
```

- Result

  The price of item Leather bag = 500.00
  Skirt price = 250.00
  Shirt price = 200.00
  Jacket price = 450.00
  could use hash cell like this 450.00
  or like this 200.00
  The price of item 1345 = 200.00
  1234 price = 500.00
  1235 price = 450.00
  1355 price = 250.00

Hash value assignment – 1/3

- Can use **string literals** as keys

"Leather bag" => 500.00 The key is Leather bag and value is 500.00

- Can use **numeric literals** also as keys

1234 => 500.00 The key is 1234 and value is 500.00

- Value of hash cell is retrieved by a scalar variable that begins with $

- **Hash uses { } brackets around key**

$hashname{"key"}

Hash value assignment – 2/3

- Rules for assigning values to hash cell

  $hash{key} = scalar;

  – $ sign identifies assignment as scalar

  – {} identify assignment as hash

  – **Key may be numeric or string literal or scalar variable**

  – If key resolves to string literal that includes spaces, string literal must be surrounded by quotation marks

  – Only scalars may be assigned to hash cell

Hash value assignment – 3/3

- Syntax of hash list assignment

  %hash = (key/value pair list)

- % sign is required in hash list assignment
- Parentheses required around key/value pair list

- Key/value pair is comma-separated list. Each key is index into hash and must be associated with value

- **Associative operator (=>) is used in place of comma to identify key/value pair**

Hash value retrieval

- **Cannot retrieve values stored in the hash in same sequential order in which they were stored in the array**

- Values of the array stored with string literal keys are retrieved in one order and the values stored with numeric keys are retrieved in different order

- To retrieve value from hash use

  $hashname{“key”};

Key retrieval functions – 1/5

- **keys** function
  - –Takes a hash and returns a list of keys or indexes in hash
  - –Syntax

    @keylist = keys %hash
  - –Can use keys function with or w/o parentheses

    @list **= keys(%hash)**

    @list **= keys %hash**
  - –The list returned by the function is processed in some type of loop and is never stored into array

Key retrieval functions – 2/5

- In foreach stmt, the keys function is executed first, and only once, returning a list

```
foreach $key (@list)
{
        print "$key = $hash{$key}\n";
}
```

- List is then processed one element at a time

- Each cell of the array @list is accessed and stored iteratively into variable $key

Key retrieval functions – 3/5

- **values** function
  - –Returns all of the values of hash
  - – Syntax

    values %hash;
  - – List returned is exactly in the same order as list returned from keys function

    @valuelist = values %hash;
  - – If we are not interested in values of the hash, but just wanted total value

```
@valuelist = values %list;
foreach (@valuelist)
{
        $total = $total + $_;
}
```

Key retrieval functions – 4/5

- **each** function
  - Returns a single key/value pair
  - It allows to iterate thro a hash one key/value at a time
  - It returns both key and value in one step

```
while (($key, $value) = each %pricelist)
  {
   print “$key => $value \n”;
  }
  foreach $key (sort keys %pricelist)
  {
   print qq| $key => $pricelist{“$key”}\n |;
  }
```

Key retrieval functions – 5/5

- The each statement can be used in a loop
- Loop will terminate at the end of processing the hash since **each returns a null list** when list is completely read
- The each function iterates thro the entire list starting from first element and moving to next element after each subsequent call

Item removal with delete function

$pricelist{‘Jacket’} = NULL;

- This **sets the key to null**, but **does not delete the item Jacket** from %pricelist

- To **remove completely use delete function**

delete $pricelist{‘Jacket’};

Item verification with exists function

- Can tell if item exists in hash by using **exists** function
- Even if key reference to null value, exists function returns true if key is part of the hash

```
$res = exists $hash{$key};

$inlist = exists $pricelist{‘Jacket’};
```

- Can also use exists in an if expression

```
if (exists $pricelist{‘Jacket’})
{
    block of stmts
}
```

Hash slices – 1/2

- Like an array (or list), a hash can be sliced to access a collection of elements instead of just one element at a time.

- For example, consider the bowling scores set individually:

```
$score{"fred"} = 205;
$score{"barney"} = 195;
$score{"dino"} = 30;
```

($score{"fred"},$score{"barney"},$score{"dino"}) = (205,195,30);

OR,

@score{"fred","barney","dino"} = (205,195,30);

Hash slices – 2/2

- **Hash slices** can also be used to merge a smaller hash into a larger one.

- In this example, the smaller hash takes precedence in the sense that if there are duplicate keys, the value from the smaller hash is used:

%league{keys %score} = values %score;

- Here, the values of %score are merged into the %league hash.

%league = (%league, %score); # merge %score into %league

(slower)

Extra for practical use

- Read standard input in a list context

```
@a = <STDIN>;
```

- Type 3 lines then press CTRL-D to indicate "end of file"
- The array ends up with three elements.
- Each element will be a string that ends in a newline, corresponding to the three newline-terminated lines entered.