Pseudocode: Dictionaries, Examples

Customers buying food items

- Find the customer who buying the highest amount of food items

Customers buying food items

- Find the customer who buying the highest amount of food items
- Create a dictionary to store food purchases
  - Customer names as keys
  - Number of food items purchased as values

```
foodD = {}
while (Table 1 has more rows) {
    Read the first row X in Table 1
    customer = X.CustomerName
    items = X.Items
    foreach row in items {
      if (row.Category == "Food") {
        if (isKey(foodD,customer)) {
          foodD[customer]
                  = foodD[customer] + 1
        }
        else {
          foodD[customer] = 1
        }
      }
    }
    Move X to Table 2
}
```

Birthday paradox

- Find a birthday shared by more than one student

Birthday paradox

- Find a birthday shared by more than one student
- Create a dictionary with dates of births as keys
- Record duplicates in a separate dictionary

```
birthdays = {}
duplicates = {}
while (Table 1 has more rows) {
    Read the first row X in Table 1
    dob = X.DoB
    if (isKey(birthdays,dob)) {
      duplicates[dob] = True
    }
    else {
      birthdays[dob] = True
    }
    Move X to Table 2
}
```

Birthday paradox

- Find a birthday shared by more than one student
- Create a dictionary with dates of births as keys
- Record duplicates in a separate dictionary
- If we want to record the names of those who share the birthday, store a list of student ids against each date of birth

```
birthdays = {}
duplicates = {}
while (Table 1 has more rows) {
    Read the first row X in Table 1
    dob = X.DoB
    seqno = X.SeqNo
    if (isKey(birthdays,dob)) {
      duplicates[dob] = True
      birthdays[dob] =
          birthdays[dob] ++ [seqno]
    }
    else {
      birthdays[dob] = [seqno]
    }
    Move X to Table 2
}
```

Birthday paradox

- Find a birthday shared by more than one student
- Create a dictionary with dates of births as keys
- Record duplicates in a separate dictionary
- If we want to record the names of those who share the birthday, store a list of student ids against each date of birth
- Can also store the students associated with each date of birth as a dictionary

```
birthdays = {}
duplicates = {}
while (Table 1 has more rows) {
    Read the first row X in Table 1
    dob = X.DoB
    seqno = X.SeqNo
    if (isKey(birthdays,dob)) {
      duplicates[dob] = True
      birthdays[dob][seqno] = True
    }
    else {
      birthdays[dob] = {}
      birthdays[dob][seqno] = True
    }
    Move X to Table 2
}
```

Resolving pronouns

- Resolve each pronoun to matching noun
  - Nearest noun preceding the pronoun

Resolving pronouns

- Resolve each pronoun to matching noun
  - Nearest noun preceding the pronoun
- Create a dictionary with part of speech as keys, sorted list of card numbers as values.

```
partOfSpeech = {}
partOfSpeech['Noun'] = []
partOfSpeech['Pronoun'] = []

while (Table 1 has more rows) {
    Read the first row X in Table 1
    if (X.PartOfSpeech == 'Noun') [
      partOfSpeech['Noun'] =
          partOfSpeech['Noun']
          ++ [X.SerialNo]
    }
    if (X.PartOfSpeech == 'Pronoun') [
      partOfSpeech['Pronoun'] =
          partOfSpeech['Pronoun']
          ++ [X.SerialNo]
    }
    Move X to Table 2
}
```

Resolving pronouns

- Resolve each pronoun to matching noun
  - Nearest noun preceding the pronoun
- Create a dictionary with part of speech as keys, sorted list of card numbers as values.
- Iterate through the dictionary to match pronouns
- Note that `partOfSpeech['Noun']` and `partOfSpeech['Pronoun']` are both sorted in ascending order of `SerialNo`

```
matchD = {}
foreach p in partOfSpeech['Pronoun'] {
    matched = -1
    foreach n in partOfSpeech['Noun'] {
      if (n < p) {
        matched = n
      }
      else {
        exitloop
      }
    }
    matchD[p] = matched
}
```