CMSC 100, FALL 2009

SOME BASIC ALGORITHMS
Examples from class 9/24/09

```
// COMPUTING INTEREST ON A LOAN
// Procedure to compute compound interest on an investment
// Prints balance after each month

procedure CompoundInterest (Principal, Rate, Months) {
  print ("Initial investment: ", Principal)
  loop from 1 to Months do {
    Principal = Principal * (1 + Rate)
    print ("After one month, balance is", Principal)
  }
}
```

```
// AVERAGING A LIST OF NUMBERS
// Function that returns the average of a list of numbers

function Average (List) {
  Sum = 0
  loop for i from 1 to Length(List) do {
    Sum = Sum + List[i]
  }
  return (Sum / Length(List))
}
```

```
// MAKING CHANGE
// Procedure that prints how much of each type of change
// to give for a payment over the amount due
// Due and Paid are in cents (e.g., $2.50 would be the value 250)

procedure MakeChange (Due, Paid) {
  if (Paid < Due) {
    print ("You owe ", Due, " but you only paid ", Paid, "!")
  } else {
    Change = Paid - Due
    // Max change: $20
    if ( Change >= 2000 ) {
      print ("You get ", Truncate (Change / 2000), " $20 bills")
      Change = Change - 2000 * Truncate (Change / 2000)
    }
    if ( Change >= 1000 ) {
      print ("You get ", Truncate (Change / 1000), " $10 bills")
      Change = Change - 1000 * Truncate (Change / 1000)
    }
    if ( Change >= 500 ) {
      print ("You get ", Truncate (Change / 500), " $5 bills")
      Change = Change - 500 * Truncate (Change / 500)
    }
    if ( Change >= 100 ) {
      print ("You get ", Truncate (Change / 100), " $1 bills")
      Change = Change - 10 * Truncate (Change / 10)
    }
    if ( Change >= 25 ) {
      print ("You get ", Truncate (Change / 25), " quarters")
      Change = Change - 25 * Truncate (Change / 25)
    }
    if ( Change >= 10 ) {
      print ("You get ", Truncate (Change / 10), " dimes")
      Change = Change - 10 * Truncate (Change / 10)
    }
    if ( Change >= 5 ) {
      print ("You get ", Truncate (Change / 5), "nickels")
      Change = Change - 5 * Truncate (Change / 5)
    }
    if ( Change >= 1 ) {
      print ("You get ", Change, "pennies")
    }
  }
```

```
// MAKING CHANGE - Shorter version
// Procedure that prints how much of each type of change
// to give for a payment over the amount due.
// Uses a subfunction to make each type of change.
// Due and Paid are in cents (e.g., $2.50 would be the value 250)

procedure MakeChange (Due, Paid) {
  new Change

  if (Paid < Due) {
    print ("You owe ", Due, " but you only paid ", Paid, "!")
  } else {
    Change = Paid - Due
    // Max change: $20
    MakeChangeType (Change, 2000, " $20 bills")
    MakeChangeType (Change, 1000, " $10 bills")
    MakeChangeType (Change, 500, " $5 bills")
    MakeChangeType (Change, 100, " $1 bills")
    MakeChangeType (Change, 25, " quarters")
    MakeChangeType (Change, 10, " dimes")
    MakeChangeType (Change, 5, " nickels")
    MakeChangeType (Change, 1, " pennies")
  }
}

function MakeChangeType (Change, UnitAmount, UnitName) {
    if ( Change >= UnitAmount ) {
      print ("You get ", Truncate (Change / 5), "nickels")
      Change = Change - 5 * Truncate (Change / 5)
    }
   return (Change)
}



// LINEAR SEARCH
// Find a number by looking through a list.  If found,
// print the location (index).

procedure LinearSearch (List, Value) {
  for i from 1 to Length(List) do {
    if ( List[i] == Value ) do {
      print (Value, " found at index ", i)
    }
  }
}
```

```
// LINEAR SEARCH, TAKE TWO
// Find a number by looking through a list.  If found,
// print the location (index) *and stop looking*.
// If not found anywhere, print "not found."

procedure LinearSearch (List, Value) {
  Found = false
  i = 1
  while ( ( not Found ) && ( i <= Length (List) ) ) do {
    if ( List[i] == Value ) do {
      print (Value, " found at index ", i)
      Found = true
    }
    i = i + 1
  }
  if ( not Found ) do {
    print (Value, " not found in list")
  }
}



// SMART LINEAR SEARCH
// Same as linear search but assumes list is ordered,
// so stop after you get higher than the number you're
// looking for.  Print location if found, and "not found"
// otherwise.

procedure LinearSearch (List, Value) {
  i = 1
  while ( ( i <= Length (List) ) && ( List[i] < Value ) ) do {
    i = i + 1
  }
  if ( ( i <= Length (List) ) && ( List[i] == Value ) ) do {
      print (Value, " found at index ", i)
  else {
    print (Value, " not found in list")
  }
}
```