Chapter 20:

Recursion

20.3

The Recursive gcd Function

Jobs with experience

The Recursive gcd Function

- Greatest common divisor (gcd) is the largest factor that two integers have in common
- Computed using Euclid's algorithm:

  `gcd(x, y) = y` if `y` divides `x` evenly

  `gcd(x, y) = gcd(y, x % y)` otherwise
- `gcd(x, y)= y` is the base case

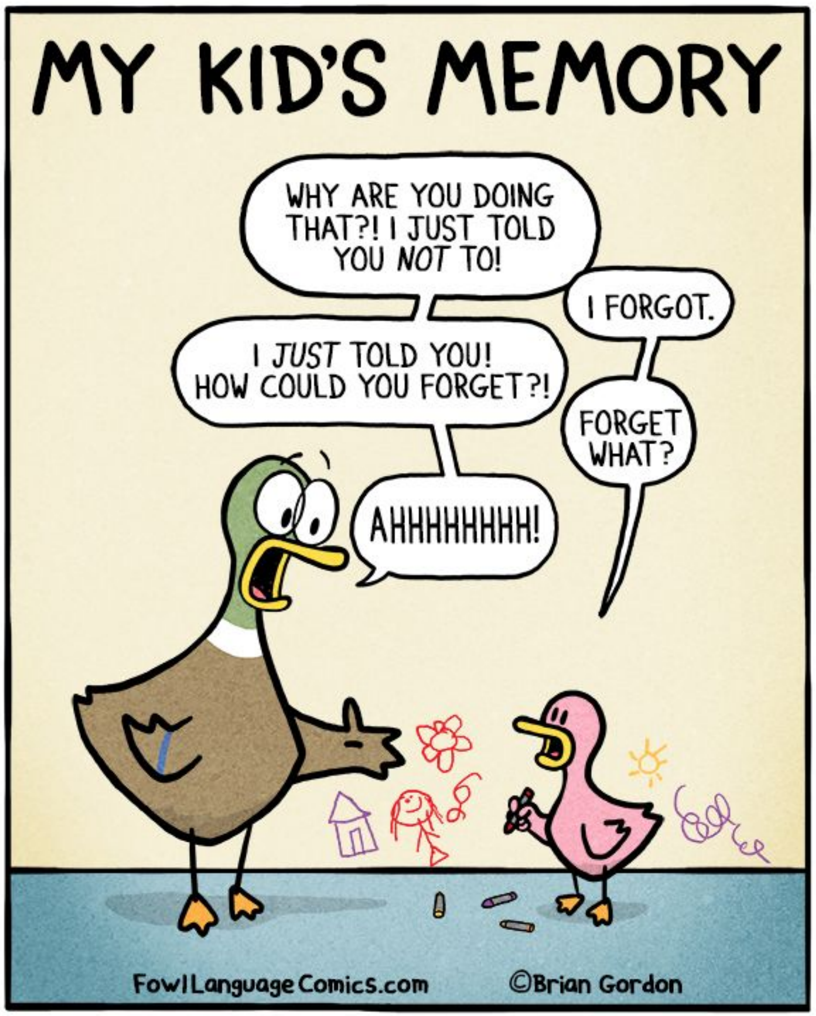
The Recursive gcd Function

```
int gcd(int x, int y)
{
     if (x % y == 0)
        return y;
     else
        return gcd(y, x % y);
}
```

20.4

Solving Recursively Defined Problems

I hope you remember that ...



Solving Recursively Defined Problems

- The natural definition of some problems leads to a recursive solution
- Example: Fibonacci numbers:

  `0, 1, 1, 2, 3, 5, 8, 13, 21, ...`
- After the starting `0, 1`, each number is the sum of the two preceding numbers
- Recursive solution:

  `fib(n) = fib(n - 1) + fib(n - 2);`
- Base cases: `n <= 0, n == 1`

Solving Recursively Defined Problems

```
int fib(int n)
{
    if (n <= 0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n - 1) + fib(n - 2);
}
```

20.5

Recursive Linked List Operations

What is grey and ...

Recursive Linked List Operations

- Recursive functions can be members of a linked list class
- Some applications:
  - Compute the size of (number of nodes in) a list
  - Traverse the list in reverse order

Contents of a List in Reverse Order

- Algorithm:
  - pointer starts at head of list
  - If the pointer is null pointer, return (base case)
  - If the pointer is not null pointer, advance to next node
  - *Upon returning from recursive call*, display contents of current node

Algorithm: displayNode function

```
void displayNode()
{
    cout <<"\nThe elements are the following: " << endl;
    Node *currentPtr = headPtr;
    displayNodePrivate(currentPtr);
}

// recursive method to display the content of the node
void displayNodePrivate(Node * current)
{
    if(current == nullptr)
        return; // we are done
    else
    {   // display the current data, and go to the next node
        cout << current->data << endl;
        displayNodePrivate(current ->next);
    };
}
```

Algorithm: displayNode function

What about displaying the elements in reverse order?

```
void displayNode()
{
    cout <<"\nThe elements are the following: " << endl;
    Node *currentPtr = headPtr;
    displayNodePrivate(currentPtr);
}

// recursive method to display the content of the node
void displayNodePrivate(Node * current)
{
    if(current == nullptr)
        return; // we are done
    else
    {
        displayNodePrivate(current ->next);
        // display the current data, and go to the next node
        cout << current->data << endl;
    };
}
```

Counting the Nodes in a Linked List

- Uses a pointer to visit each node
- Algorithm:
  - pointer starts at head of list
  - If pointer is null pointer,
    - return 0 (base case)
  - else,
    - return 1 + number of nodes in the list pointed to by current node
- See the `NumberList` class in Chapter 19

The countNodes function, a private member function

```
// driver function
int sizeList()
{
    Node *currentPtr = headPtr;
    int size = sizeListPrivate(currentPtr);
    return size;
}

// recursive method to count the number of nodes
int sizeListPrivate(Node * current)
{
    if(current == nullptr)
        return 0; // we are done
    else
    {
        return 1 + sizeListPrivate(current ->next);
    };
}
```

The countNodes function, a private member function

20.6

A Recursive Binary Search Function

Let's exchange numbers ...

A Recursive Binary Search Function

- Binary search algorithm can easily be written to use recursion
- Base cases: desired value is found, or no more array elements to search
- Algorithm (array in ascending order):
  - If middle element of array segment is desired value, then done
  - Else, if the middle element is too large, repeat binary search in first half of array segment
  - Else, if the middle element is too small, repeat binary search on the second half of array segment

A Recursive Binary Search Function (Continued)

```
int BinarySearch(int array[], int start_index, int end_index, int element){

   // when end_index < start_index is true, that would indicate
   // that the element has not been found

   if (end_index <start_index) return -1; // one base case

   // calculate the middle value
   int middle = (start_index + end_index)/ 2;

   // if the value is found, we stop
   // this is one base case
   if (array[middle] == element) // second base case
      return middle;


   if (array[middle] > element) // the element is on the left side of the middle
      return BinarySearch(array, start_index, middle-1, element);


   else // the element is on the right side of the middle value
      return BinarySearch(array, middle+1, end_index, element);

}
```

A Recursive Binary Search Function (Continued)

Thank you

Please let me know if you have any further questions!