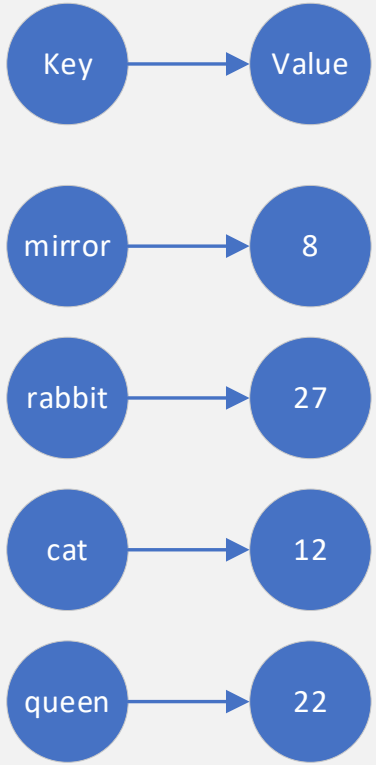
BINARY TREE EXAMPLE 2: TWO TEMPLATE VARIABLES

Mapping keys to values

K-V Pairs

Delroy A. Brinkerhoff

MAPPING KEYS TO VALUES



- The K-V mapping
  - May have a "natural" association
  - May be meaningful only in a given problem
- Key and Value can be simple or complex
- Program searches for the Key to use the Value
- Implemented with a "fast" search algorithm
  - Binary tree
  - Hash table

THE KVTree CLASS

```
template <class K, class V>
class KVTree
{
    private:
        K                key;
        V                value;
        KVTree<K,V>*     left = nullptr;
        KVTree<K,V>*     right = nullptr;

    public:
        ~KVTree();
        V* insert(K key, V value);
        V* search(K key);
        void remove(K key);
};
```

THE KVTree insert FUNCTION

```
template <class K, class V>
V* KVTree<K, V>::insert(K key, V value)
{
    KVTree<K, V>* top = this;
    KVTree<K, V>* bottom = right;

    while (bottom != nullptr)
    {
        if (bottom->key == key)
            return &bottom->value;

        top = bottom;
        bottom = (key < bottom->key) ? bottom->left : bottom->right;
    }

    bottom = new KVTree;
    bottom->key = key;
    bottom->value = value;
    ((top != this && key < top->key) ? top->left : top->right) = bottom;

    return &bottom->value;
}
```

MAPPING AN ID TO AN EMPLOYEE

```
KVTree<int, Employee> tree;

tree.insert(400, Employee("Dilbert", "225 Elm"));
                .
                .
                .
cout << "Search: " << *tree.search(500) << endl;
tree.remove(800);
```

THE WordCount PROGRAM

```
KVTree<string, int>   tree;

while ((c = file.get()) != EOF)
{
    if (isalpha(c))
        word += tolower(c);
    else if (word.length() > 0)
    {
        int* count = tree.search(word);
        if (count != nullptr)
            (*count)++;
        else
            tree.insert(word, 1);
        word.clear();
    }
}
```