Generalized Lists

- Need a new underlying representation:
  - A node in the list needs to be able to contain either an atomic piece of information or point to another list

Tag = 0 (Data) / 1 (Sublist)	Data/Sublist Pointer	Next Pointer

Generalized Lists

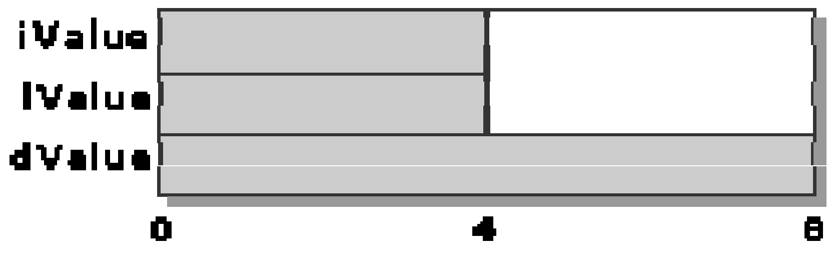
```
class GenListNode {
        friend class GenList;
        private:
          bool tag;
          GenListNode *next;
        union {
                char data;          // or any other type of interest
                GenListNode* sublist; }
}

class GenList {
        private:
         GenListNode *front;       // using name front because
                                   // we will use the term head
                                   // as something like a function
}
```

Generalized Lists

- Union definition:
  - User-defined data type that, at any given time, contains only one object from its list of members (although that object can be an array or a class type). The *member-list* of a union represents the kinds of data the union can contain. A union requires enough storage to hold the largest member in its *member-list*.

```
union NumericType
{
        int iValue;
        long lValue;
        double dValue;
};
```



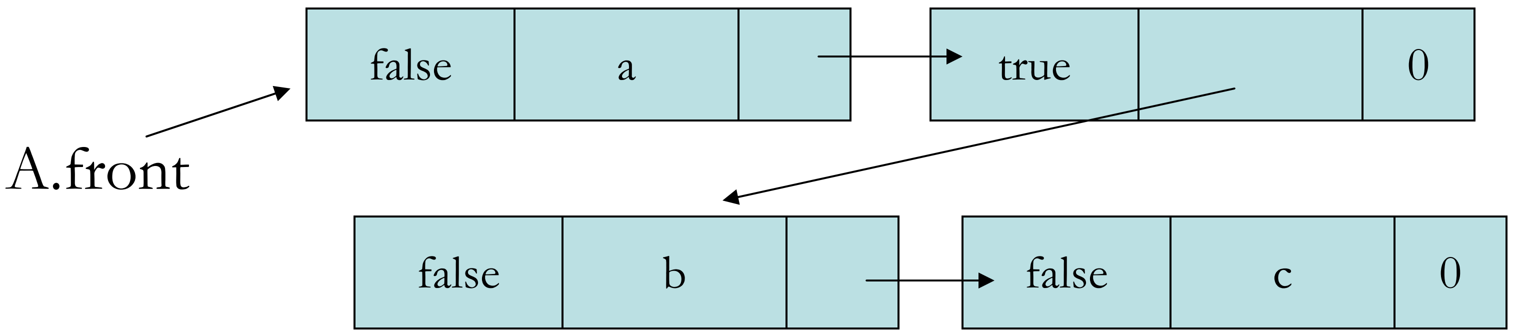
Generalized Lists

- Example representations
  - D = ( ) Length 0, Null List

  D.front = 0;

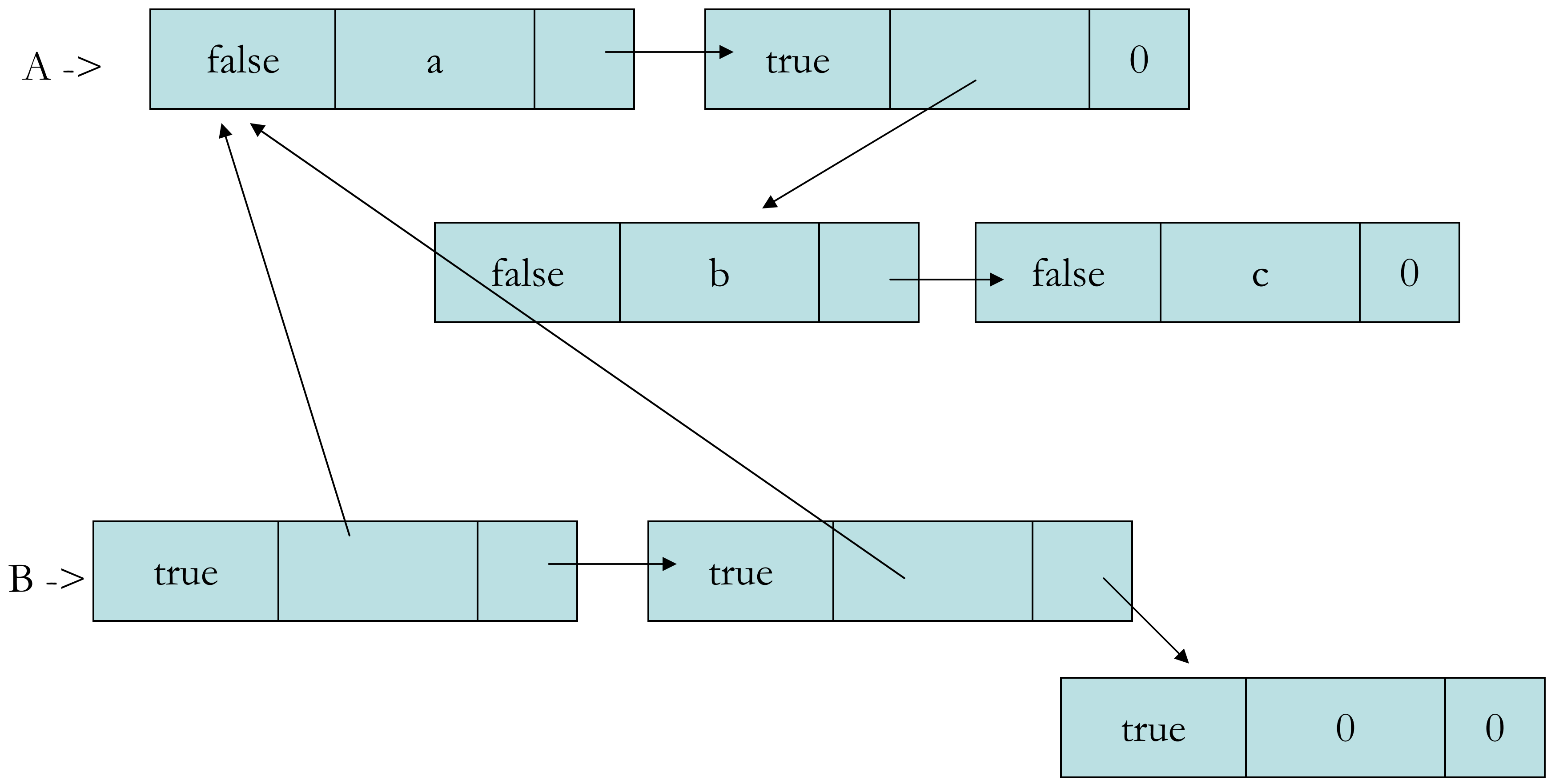
  - A = (a, (b,c)) Length 2

    Head = a Tail = ((b,c))



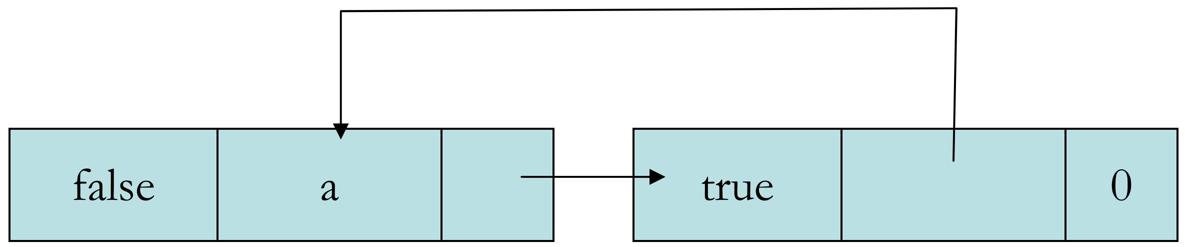
Generalized Lists

- B = (A, A, ( )) {where A is defined previously}



Generalized Lists

- C = (a, C)



Generalized List Algorithms

- 4 Key Properties:
  - Handle null pointers
  - Look at tag
  - Depending on tag
    - Handle item locally or handle sublist with recursive call
  - Handle next pointer with recursive call

Generalized List Copy

```
// Driver
void GenList::Copy(const GenList &rhs)
{ first = Copy(rhs.first); }

// Workhorse
GenListNode* GenList::Copy(GenListNode* p)
{
    GenListNode* q = 0;
    if (p != 0) {
            q = new GenListNode();
            q-> tag = p->tag;
            if (q->tag == false) q-> data = p->data;
            else q->sublist = Copy(p->sublist);
            q->next = Copy(p->next);
    }
    return q;
}
```

Generalized List Equality

- Test for Equality
  - Requires:
    - Same list structure (placement of atoms and sublists)
    - Same list data
- Essential properties of algorithm:
  - Check equality of tags
  - If equal
    - If data elements, check equality for data type
    - If list elements, recursively check equality on sublist

```
bool operator==(const GenList& l, const GenList& r)
{ return equal(l.first, r.first); }

bool equal(GenListNode* s, GenListNode* t)
{
    bool equalSoFar;
    if ((!s) && (!t)) return true; // both empty
    if (s && t && (s->tag == t->tag)) // data in lists, same
   {                                  // type in this position
        // check data if not sublists
        if (s->tag == 0)
        {
                if (s->data == t->data) equalSoFar = true;
                else return false;
        }
        // check recursively on sublists otherwise
        else equalSoFar = equal(s->sublist, t->sublist);

        // if equal so far, recurse on next nodes
        if (equalSoFar) return equal(s->next, t->next);
    }
    else return false;  //otherwise return false
}
```