DESIGN OF GENERIC LIBRARIES

Alexander A. Stepanov
Concurrent Computing Dept.
Hewlett-Packard Labs
Palo Alto, CA
HISTORY OF THE ACTIVITY

1. 79-84 TECTON
   - specialized functional forms
   - algorithms on algebraic structures
   - complexity signatures

2. 84-87 Higher Order Imperative Programming
   - extensive library
   - complex graph algorithms
   - extensive use of state

3. 87-89 Ada Generic Library
   - Common Lisp functionality
   - deep layering
   - high efficiency

4. 88-91 C++ Component Library
   - production quality
   - developed structuring
OBJECTIVE

To develop a way to construct libraries of software components.

Components should be:

- useful;
- efficient;
- generic;
- correct.

Libraries should be:

- comprehensive;
- well-structured;
- well-documented.
Bentley and Binary Search

L:=1; U:=N
loop
if L>U then
    P:=0; break
M:=(L+U) div 2
case
    X[M]<T : L:=M+1
    X[M]=T : P:=M; break
    X[M]>T : L:=M-1
lessThan:

returns the smallest index $K$ in the sorted segment such that COMP is negative for all indices smaller than $K$.

greaterThan:

returns the smallest index $K$ in the sorted segment such that COMP is non-positive for all indices smaller than $K$.

equalTo:

returns a pair of $(\text{lessThan}(), \text{greaterThan}())$.
But is faster than calling them separately.
**lessThan**

**Declaration**

```
Index SortedSegment::lessThan();
```

**Description**

`lessThan` returns the index of the rightmost element in the sorted segment such that for all indices in the segment that are smaller than it `comp` is negative. Normally, `lessThan()-first()` is equal to the number of the elements in the segment for which `comp` is negative, and `length()`-(`lessThan()-first()`) is equal to the number of the elements in the segment for which `comp` is non-positive. (Subtraction of indices is not required by the algorithm, but is normally defined.)

**See Also** `greaterThan`, `equalTo`, `insert`, `setInsert`

**Time Complexity** Logarithmic. If \( n \) is the number of elements in the segment then at most \( \lfloor \log_2 n \rfloor + 1 \) operations `comp` are performed.

**Space Complexity** Constant

**Mutative?** No

**Implementation**

```
Index SortedSegment::lessThan()
{
    register Index start = first();
    register Integer len = length();

    while (len>0)
    {
        register Integer half = len>>1;
        register Index middle = half+start;

        if (comp(middle)<0)
        {
            start = middle+(Integer)1;
            len = (len-half)-(Integer)1;
        }
        else
            len = half;
    }
```
Index SortedSegment::lessThan()
{
    register Index start = first();
    register Integer len = length();

    while (len>0)
    {
        register Integer half = len>>1;
        register Index middle = start+half;

        if (comp(middle)<0)
        {
            start = middle+(Integer)1;
            len = (len-half)-(Integer)1;
        }
        else
            len = half;
    }

    return start;
}
Index SortedSegment::lessThan()
{
    Index start = first();
    Integer len = length();

    while (len>0)
    {
        Integer half = len>>1;
        Index middle = start+half;

        if (comp(middle)<0)
        {
            start = middle+(Integer)1;
            len = (len-half)-(Integer)1;
        }
        else
            len = half;
    }

    return start;
}
CLASS ASSUMPTIONS

classtype Integer;
// an (unsigned) integral type

classtype Index
{
public:
    Index operator+(Integer);
implied:
    int operator<(Index);
meaning:
    {Index s; Integer x, y;
     (s+x)+y==s+(x+y);
    }
    {Index s; Integer x;
     s<s+x;
    }
};

classtype SortedSegment
{
public:
    Index first();
    Integer length();
    int comp(Index);
meaning:
    {Integer x, y;
     x>y ||
     y>length() ||
     comp(first()+x)<=comp(first()+y);
    }
};
MEANING OF BINARY SEARCH

SortedSegment::lessThan::meaning()
{
    {Integer x;
        !(x>=0) ||
        !(x<length()) ||
        !(first()+x<lessThan()) ||
        comp(x)<0;
    }

    { !(lessThan()<length()) ||
        comp(lessThan())>=0;
    }
}
typedef float Type;
typedef int Integer;
typedef Type* Index;

class Fvector
{
private:
    Index b;
    Integer l;
    Type v;
public:
    Fvector(Index x, Integer y, Type z) :
        b(x), l(y), v(z){};
    Integer length() {return l;}
    Index first() {return b;}
    void setValue(type z){v=z;}
    int comp(Index x){return int(*x-t);}
    Index lessThan();
};

#define SortedSegment Fvector
#include "lessThan.H"
typedef float Type;
typedef int Integer;

struct Node
{
    Node* cdr;
    Type car;
};

struct Index
{
    Node* i;
    Index(Node* j) : i(j);
    Index operator+(Integer n)
    {
        Node* t = i;
        while (n-->0) t = *t;
        return Index(t);
    }
};
class Flist
{
private:
   Index b;
   Type v;
public:
   Flist(Index x, Type z) :
      b(x), v(z) {};
   Integer length()
   {
      for(int i = 0; b.cdr(); i++);
      return i;
   }
   Index first() {return b;}  
   void set_value(type z){v=z;}
   int comp(Index x){return int(*x-t);}  
   Index lessThan();
};

#define SortedSegment Fvector
#include "lessThan.H"
D. R. Musser & A. A. Stepanov
*Ada Generic Library: Linear List Processing Packages*
Springer-Verlag, 1989

170 Components in 8 packages

- Double Ended Lists
- Stacks
- Deques
- System Allocated
- User Allocated
- Singly Linked Lists
- Auto Reallocating
- Linked List Algorithms

implemented in terms of

plugs together with
PARTIAL AND APPROXIMATE MODELS

A partial model is isomorphic to the complete model when the partial model is defined.

E.g.,

- bounded sequences
- fixed precision arithmetic

An approximate model may also disagree with the complete model, but all the disagreements are within well defined tolerance

E.g.,

- floating point arithmetic
- pixel representation of continuous images
ORGANIZATION OF THE LIBRARY

1. Structures
   - set of operations
   - generic meaning
   - generic algorithms

sequences

2. Realizations
   - relative complexity
   - specific meaning
   - specific algorithms

vectors, lists

3. Implementations
   - constraints
   - exact complexity
   - exception handling

bounded vectors,
extensible vectors

singly linked lists with
   incremental garbage collection
CLASSIFICATION OF OPERATIONS

Pseudo-permutations
{some of the same elements in a different order}

Improper permutations:
index based:
  subrange, even-numbered
predicate-based:
  remove, stable remove
comparison-based:
  remove-duplicates, select largest half

Proper permutations:
index based:
  reverse, rotate, random-shuffle
predicate-based:
  partition, stable partition
comparison-based:
  sort, stable sort, partial sort

FUNCTIONAL vs. MUTATIVE
FUTURE PLANS

* Community selection
  - formal
  - software engineering
  - systems

* CSP (Maxwell, Datamesh)
  - classification of disk algorithms
  - build DataMesh from reusable components