Pointers (review and examples)

CSE 373
Data Structures
Lecture 2
Basic Types and Arrays

• Basic Types
  › integer, real (floating point), boolean (0,1), character

• Arrays
  › A[0..99] : integer array

A[5]
Records and Pointers

• **Record** (also called a struct)
  › Group data together that are related
    
    ```
    X : complex pointer
    real_part : real
    imaginary_part : real
    ```
  
  › To access the fields we use **“dot” notation**.

    ```
    X.real_part
    X.imaginary_part
    ```
Record Definition

- Record definition creates a new type

Definition

\[
\text{record complex : (}
\begin{align*}
&\text{real\_part : real,} \\
&\text{imaginary\_part : real}
\end{align*}
\)
\]

Use in a declaration

\[X : \text{complex}\]
Pointer

• A pointer is a reference to a variable or record (or object in Java world).

```
X : blob pointer

*X   blob
```

• In C, if X is of type pointer to Y then *X is of type Y
Creating a Record

- We use the “new” operator to create a record.

\[ P : \text{pointer to blob;} \]

\[ P := \text{new blob;} \]

\[ \text{(null pointer)} \]
Simple Linked List

• A linked list
  › Group data together in a flexible, dynamic way.
  › We’ll describe several list ADTs later.

\[
\begin{align*}
L &: \text{node pointer} \\
4 &\rightarrow 9 &\rightarrow 13 &\rightarrow 20
\end{align*}
\]

record node : (  
  data : integer,  
  next : node pointer
  )
Application
Sparse Polynomials

• $10 + 4x^2 + 20x^{40} + 8x^{86}$

record poly : (  
   exp : integer,  
   coef : integer,  
   next : poly pointer  
)  

Exponents in Increasing order
Identically Zero Polynomial

null pointer
Addition of Polynomials

\[ 10 + 4x^2 + 20x^{40} + 8x^{86} \]

\[ 7x + 10x^2 - 8x^{86} \]
Recursive Addition

Add(P, Q : poly pointer): poly pointer{
R : poly pointer
case {
P = null : R := Q ;
Q = null : R := P ;
P.exp < Q.exp : R := P ;
    R.next := Add(P.next,Q);
    P.exp > Q.exp : R := Q ;
        R.next := Add(P,Q.next);
P.exp = Q.exp : R := P ;
        R.coef := P.coef + Q.coef ;
    R.next := Add(P.next,Q.next);
}
return R
}
Example

Add

P → 0 → 2 → 40 → 86
10 → 4 → 20 → 8

Q → 1 → 2 → 86
7 → 10 → -8
Example (first call)

Add

P

R

Q

0
1
10
7
2
4
10
2
40
20
86
8

86
8

40
20

86
8
The Recursive Call

Add

```
P -> 0 10
    2 4
    40 20
    86 8

Q -> 1 7
    2 10
    86 -8
```
During the Recursive Call

Add

Represent return values

Return value

R
After the Recursive Call

- Add
- Return value

R

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>14</td>
<td>40</td>
</tr>
<tr>
<td>86</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>86</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-8</td>
</tr>
</tbody>
</table>
```
The final picture
Notes on Addition

• Addition is destructive, that is, the original polynomials are gone after the operation.

• We don’t salvage “garbage” nodes. Let’s talk about this.

• We don’t consider the case when the coefficients cancel. Let’s talk about this.
Unneeded nodes to Garbage

• How would you force the unneeded node to be garbage in the code on slide 11?

• Suggestions?
Memory Management –
Private Store

- **Private store** – get blocks from a private store when possible and return them when done.
  - Efficiently uses blocks of a specific size
  - The list of unused blocks can build up eventually using too much memory.
Private Store

R

unneeded

garbage

0 10
1 7

2 14
40 20

86 0

2 86
10 -8
Private Store

```
<table>
<thead>
<tr>
<th>R</th>
<th>0 10</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 7</td>
</tr>
<tr>
<td></td>
<td>2 14</td>
</tr>
<tr>
<td></td>
<td>40 20</td>
</tr>
</tbody>
</table>
```

```
FreeList
```

```
<table>
<thead>
<tr>
<th>FreeList</th>
</tr>
</thead>
<tbody>
<tr>
<td>86 0</td>
</tr>
<tr>
<td>86 -8</td>
</tr>
<tr>
<td>2 10</td>
</tr>
</tbody>
</table>
```
Memory Management – Global Allocator

• Global Allocator’s store – always get and return blocks to global allocator
  + Necessary for dynamic memory.
  + Blocks of various sizes can be merged if they reside in contiguous memory.
  - Allocator may not handle blocks of different sizes well.
  - Allocator may be slower than a private store.
Memory Management – Garbage Collection

• Garbage collection – run time system recovers inaccessible blocks from time-to-time. Used in Lisp, Smalltalk, Java.
  + No need to return blocks to an allocator or keep them in a private store.
  - Care must be taken to make unneeded blocks inaccessible.
  - When garbage collection kicks in there may be undesirable response time.
Solution for Polyn. Addition

P.exp = Q.exp : R := P ;

    R.coef := P.coef + Q.coef ;

    if R.coef = 0 then

        R := Add(P.next,Q.next);

    // The terms with coef = 0 have been removed from the
    // result

    else

        R.next := Add(P.next,Q.next);

    }


Use of Private Store or Global Allocator

```c
P.exp = Q.exp : R := P ;

    R.coef := P.coef + Q.coef ;

if R.coef = 0 then

    R := Add(P.next,Q.next);

    Free(P); Free(Q);

else

    R.next := Add(P.next,Q.next);

    Free(Q);

}
```