Basics on Pointers

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]

10/2/02 Basics on Pointers - Lecture 2
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
record complex : (  
    real_part : real,  
    imaginary_part : real  
)

Use in a declaration
X : complex
**Pointer**

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

\[ X : \text{blob pointer} \]

\[ X \rightarrow *X \]

- 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.

\[
\begin{aligned}
L : \text{node pointer} \\
\quad \downarrow \\
\quad 4 \quad \rightarrow \quad 9 \quad \rightarrow \quad 13 \quad \rightarrow \quad 20 \\
\text{record node : (}
\quad \text{data : integer}
\quad \text{next : node pointer}
\quad \text{)}
\end{aligned}
\]
Sparse Polynomials

- $10 + 4x^2 + 20x^{40} + 8x^{86}$

Exponents in Increasing order

```
record poly : (  
    exp : integer,  
    coef : integer,  
    next : poly pointer  
)
```
Identically Zero Polynomial

\( P \) null pointer

\[ \begin{align*}
1 \quad 2 \quad 86 \\
0 \quad 0 \quad 0
\end{align*} \]
Addition of Polynomials

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

\[P \rightarrow \begin{array}{cccc}
0 & 2 & 40 & 86 \\
10 & 4 & 20 & 8 \\
\end{array}\]

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

\[Q \rightarrow \begin{array}{cccc}
1 & 2 & 86 \\
7 & 10 & -8 \\
\end{array}\]
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 \rightarrow 0 10 2 4 40 20 86 8

Q \rightarrow 1 7 2 10 86 -8
Example

Add

P \rightarrow 0 \rightarrow 10 \rightarrow 2 \rightarrow 4 \rightarrow 20 \rightarrow 40 \rightarrow 86

Q \rightarrow 1 \rightarrow 7 \rightarrow 2 \rightarrow 10 \rightarrow 86 \rightarrow -8
The Recursive Call
After the Recursive Call

Add

Return value

Add

R

0
10

2
14

40
20

86
0

1
7

2
10

86
-8
Example

Add

R

Add

Return value

1
2
7
2
14
10
-8
0
86
20
40
86
0
Example

```
  0
  10

  1
  7

  2
  14

  40
  20

  86
  0

R

unneeded

garbage
```
Notes on Addition

• Addition is destructive, that is, the original polynomial are gone after the operation.
• We don’t salvage “garbage” nodes. We’ll talk about this later.
• We don’t consider the case when the coefficients cancel. We’ll talk about that later.
Unneeded to Garbage

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

FreeList

10/2/02 Basics on Pointers - Lecture 2 22
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 to Class Work

\[
P \cdot \text{exp} = Q \cdot \text{exp} : R := P ; \\
    R \cdot \text{coef} := P \cdot \text{coef} + Q \cdot \text{coef} ; \\
    \text{if } R \cdot \text{coef} = 0 \text{ then} \\
        R := \text{Add}(P \cdot \text{next}, Q \cdot \text{next}) ; \\
    \text{else} \\
        R \cdot \text{next} := \text{Add}(P \cdot \text{next}, Q \cdot \text{next}) ; \\
\]

\]
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) ;

}  
```