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

Records and Pointers

- Record (also called a struct)
  - Group data together that are related
    - \( X : \text{complex pointer} \)
      \[
      X \rightarrow \begin{cases} \text{real part} : \text{real} \\ \text{imaginary part} : \text{real} \end{cases}
      \]
  - To access the fields we use “dot” notation.
    - \( X . \text{real part} \)
    - \( X . \text{imaginary part} \)

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).

Creating a Record

- We use the “new” operator to create a record.
  - \( P : \text{pointer to blob;} \)
    - \( P \rightarrow (\text{null pointer}) \)
  - \( P := \text{new blob;} \)
Simple Linked List

- A linked list
  - Group data together in a flexible, dynamic way.
  - We'll describe several list ADTs later.

```
record node : {
  data : integer
  next : node pointer
}
```

Sparse Polynomials

- \( 10 + 4x^2 + 20x^{40} + 8x^{86} \)

```plaintext
P 0 4 40 86
```

Identically Zero Polynomial

```
P null pointer
```

Addition of Polynomials

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

Recursive Addition

```cpp
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
Example

The Recursive Call

After the Recursive Call

Example

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 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

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.exp = Q.exp ; R := P ;
R.coef := P.coef + Q.coef ;
if R.coef = 0 then
  R := Add(P.next,Q.next) ;
else
  R.next := Add(P.next,Q.next) ;
```

Use of Private Store or Global Allocator

```
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) ;
```