3137 Data Structures and Algorithms in C++

Lecture 2

July 10 2006

Shlomo Hershkop

Announcements

- Homework due soon
  - make sure to submit the homework on time
  - ask if you need help
- I forgot to take feedback last class, please remind me at end of class
  - scribble/email me comments

Outline

- Review
- Proofs
- Recursive programming
- ADT
- Lists
- Stacks

3

From Last time

- anyone have any questions ??

- any questions on the homework ??
  - let me go over the programming section

4

Working environment

- will post submission instructions later
  - need to get class account setup
- some suggestions on working
- let me demo eclipse

5

Question

- Anyone know the difference between

  Mathematical induction

  vs

  Logical deduction ??

6

Deduction

- inference in which the conclusion about particulars follow from the general or universal premise

1. The picture is above the desk.
2. The desk is above the floor.
3. Therefore the picture is above the floor

7

Wrong deduction

1. Every terrorist opposes the government
2. Everyone in the opposition party opposes the government.
3. Therefore everyone in the opposition party is a terrorist

- what is wrong here ?

8

Induction

- Inference of generalized conclusion from particular instances
  - i.e. the process of reasoning in which the premises of an argument support the conclusion but do not ensure it
- The Street is wet
- When it rains the street becomes wet
- It must have rained

9

difference

- deduction is logical necessity
- Usually will see something and induce something
  - which might be disproved later
  - this is not the case with mathematical induction

10

SAP Method

- here is a quick and dirty method for mathematical induction

- Show
- Assume
- Prove

11

Show

- Here we show the theorem holds in the simplest case (base)

12

Assume

- Assume the theorem holds for a general case
- called inductive hypothesis
- Example: assuming the hypothesis to be true for some specific integer k.

13

Prove

- Prove that the theorem holds for the next larger case

14

Example

$$\sum_{i=1}^{n}(2i-1)=n^2$$

- How can we prove this is the case ??

15

Show

- Set n = 1
- is it true ??
- now assume it is true, how ca we prove it is true in the general case ??

16

proof

$$\sum_{i=1}^{k+1}(2i-1)=(k+1)^2$$

$$[2(k+1)-1]+\sum_{i=1}^{k}2i-1$$

$$(2k+2-1)+k^2\ (\ from-inductive-hypothesis)$$

$$k^2+2k+1$$

$$(k+1)^2$$

$$QED$$

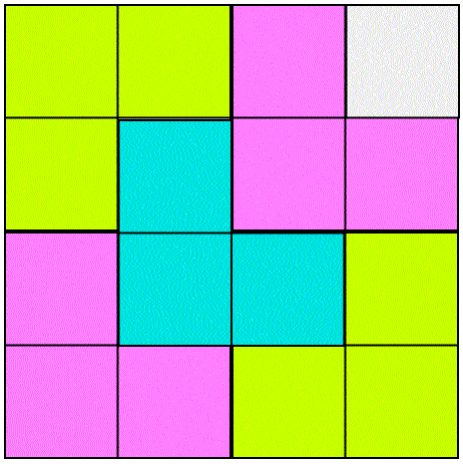
17

Next

- Does this make sense ??
- Can you do the same for :
- Given a 2^n by 2^n checkerboard with any one square deleted, it is possible to cover this board with L-shaped pieces.

18

Example



19

- what is the base case??

20

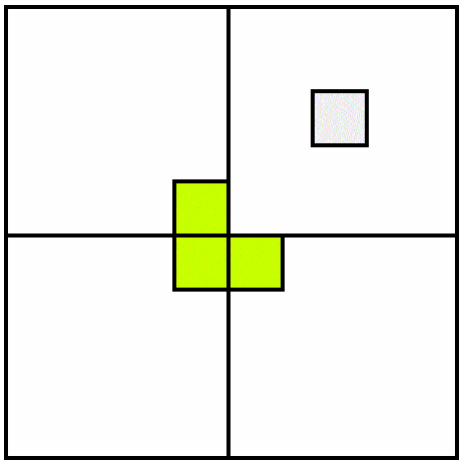
- What is the assumption ??

21

- How would the proof go ?

22

2^k by 2^k sections



23

Another Example

- We want to prove that for the fibinochi number series such that
- $F_0 = 1, F_1 = 1, F_2 = 2, F_3 = 3, F_4 = 5$, etc
- $F_i \leq (5/3)^i$

24

Show

- F1 ≤ $(5/3)^1$
- F2 ≤ $(5/3)^2$

- Assume:
  - true for all i

25

Proof

$$F_{k+1} \le \left(\frac{5}{3}\right)^{k+1}$$

$$F_{k+1} = F_k + F_{k-1}$$

$$\left(\frac{5}{3}\right)^{k} + \left(\frac{5}{3}\right)^{k-1} \le \left(\frac{5}{3}\right)^{k+1}$$

$$divide - by - \left(\frac{5}{3}\right)^{k-1}$$

$$\frac{3}{3} + \frac{5}{3} \le \frac{25}{9}$$

$$\frac{24}{9} \le \frac{25}{9}$$

26

Recursion

- fib is related to recursion
- recursion is code that is defined in terms of itself
- Many DS problems can be solved in a recursive fashion

27

Example

- can you code the power function as a recursive method ?

28

power

```
int power(int x, int y) {
  if(y==0) {
     return 1;
  }
  else {
     return x * power(x,y-1);
   }
}
```

29

some important points

- biggest problem with recursive code is getting stuck in infinite loop
- here are some quick guidelines

30

Rules

1. Base Case:
   make sure you have a base case which can be computed without recursion

2. Progress
   make sure you are making progress towards solution

31

More Rules

3. Assume all recursive calls return correctly

4. Never duplicate work

32

Question

- when is recursion necessary to solve a problem ??

33

- Never
- if you can do it recursively, can do it iteratively
- can you prove this ??

34

- proof: cpu is not a recursive cpu
- generally a non recursive solution will run faster
- BUT
  - harder to read

35

Example

- count number of digits in an int recursively

```
int numDigits(int x) {
  if(abs(x) < 10) {
     return 1;
  }
  else {
     return numDigits(x/10) + 1;
  }
}
```

36

tail recursion

- most of the time, when you see recursion its slowed down by the fact that it needs to wait for another function call before returning its value

37

Factorial

```
int factorial(int num) {
  if (num ==1)
     return 1;
  else
     return num * factorial(num-1);
}
```

38

Better version

```
int fact2(int num, int result) {
  if(num==1)
     return result;
  else
     return fact2(num1-,result*num);
}
```

39

switch gears

- let us start to talk about how to organize data

40

Abstraction

- one important concept for DS is the idea of abstraction
- anyone have a pilots license ??
- anyone know how to fly a plane ??

41

- Abstraction is hiding the details
  - focus on important bits
  - simplify
  - allow change to happen later
    - we can replace underlying structure without changing the outside view
    - same idea of a standard api

42

List ADT

- We want to represent a group of items
- with a list which operations would you have ??

43

List ADT

- insert
- remove
- sort
- find first
- find last
- count
- Notice how we aren't even talking about how to store the list
- Any IDEAS ??

44

Arrays to implement List ADT

- Positive ?
- Negative ?

- what is the cost of insertion ?
- what is the cost of insert at beginning ?
- find by value ?
- find by index ?

45

Linked List

- structure which has list and links to elements
- each node points to next
- not necessarily adjacent in memory
- last has null pointer
- need to keep link to first element

46

Linked Lists

- Positive ?
- Negative ?

- what is the cost of insertion ?
- what is the cost of insert at beginning ?
- find by value ?
- find by index ?

47

improvements

- can improve linked list DS by adding another set of links going backwards
- Double linked lists

- header / tail nodes

48

```
1       struct Node
2       {
3           Object  data;
4           Node   *prev;
5           Node   *next;
6
7           Node( const Object & d = Object( ), Node *p = NULL, Node *n = NULL )
8             : data( d ), prev( p ), next( n ) { }
9       };
```

49

quick question

- here is a short code segment, see if you can type it up and compile on your computer….for those who don't have one, can you tell me the output…

50

```
#include <iostream.h>

class X {
public:
X() { cout << 1 << ' '; }

X( const X& ){ cout << 2 << ' '; }

~X(){ cout << 3 << ' '; }

X& operator=( const X& ){ cout << 4 << ' '; }
};

X f( X x ){ return x; }
X& g( X& x ){ return x; }

int main() {
   X a;
   X b = a;
   cout << endl;
   a = b;
   cout << endl;
   a = f( b );
   cout << endl;
   b = g( a );
   cout << endl;
   return 0;
 }
```

51

```
1   template <typename Object>
2   class List
3   {
4     private:
5       struct Node
6         { /* See Figure 3.13 */ };
7
8     public:
9       class const_iterator
10        { /* See Figure 3.14 */ };
11
12      class iterator : public const_iterator
13        { /* See Figure 3.15 */ };
14
15    public:
16      List( )
17        { /* See Figure 3.16 */ }
18      List( const List & rhs )
19        { /* See Figure 3.16 */ }
20      ~List( )
21        { /* See Figure 3.16 */ }
22      const List & operator= ( const List & rhs )
23        { /* See Figure 3.16 */ }
24
25      iterator begin( )
26        { return iterator( head->next ); }
27      const_iterator begin( ) const
28        { return const_iterator( head->next ); }
29      iterator end( )
30        { return iterator( tail ); }
31      const_iterator end( ) const
32        { return const_iterator( tail ); }
33
34      int size( ) const
35        { return theSize; }
36      bool empty( ) const
37        { return size( ) == 0; }
38
39      void clear( )
40      {
41          while( !empty( ) )
42              pop_front( );
43      }
```

```
44      Object & front( )
45        { return *begin( ); }
46      const Object & front( ) const
47        { return *begin( ); }
48      Object & back( )
49        { return *--end( ); }
50      const Object & back( ) const
51        { return *--end( ); }
52      void push_front( const Object & x )
53        { insert( begin( ), x ); }
54      void push_back( const Object & x )
55        { insert( end( ), x ); }
56      void pop_front( )
57        { erase( begin( ) ); }
58      void pop_back( )
59        { erase( --end( ) ); }
60
61      iterator insert( iterator itr, const Object & x )
62        { /* See Figure 3.18 */ }
63
64      iterator erase( iterator itr )
65        { /* See Figure 3.20 */ }
66      iterator erase( iterator start, iterator end )
67        { /* See Figure 3.20 */ }
68
69    private:
70      int   theSize;
71      Node *head;
72      Node *tail;
73
74      void init( )
75        { /* See Figure 3.16 */ }
76  };
```

53

Iterators

- some data structures have an idea of a position
- want to abstract that away
- use of helpers known as Iterators which allow you to iterate over a group of items

54

Issues

- getting Iterators
- operations
- when required ?

55

operations

- ++
- *
- ==
  !=
- anything else ??

56

When do we need Iterators ?

- list manipulations can be made safer and easier using Iterators

- Example:
  - inserts
  - range inserts
  - deletion
  - range deletion

57

```
class const_iterator
{
  public:
    const_iterator( ) : current( NULL )
      { }

    const Object & operator* ( ) const
      { return retrieve( ); }

    const_iterator & operator++ ( )
    {
        current = current->next;
        return *this;
    }

    const_iterator operator++ ( int )
    {
        const_iterator old = *this;
        ++( *this );
        return old;
    }

    bool operator== ( const const_iterator & rhs ) const
      { return current == rhs.current; }
    bool operator!= ( const const_iterator & rhs ) const
      { return !( *this == rhs ); }

  protected:
    Node *current;

    Object & retrieve( ) const
      { return current->data; }

    const_iterator( Node *p ) : current( p )
      { }

    friend class List<Object>;
};
```

58

```
39      class iterator : public const_iterator
40      {
41        public:
42          iterator( )
43            { }
44
45          Object & operator* ( )
46            { return retrieve( ); }
47          const Object & operator* ( ) const
48            { return const_iterator::operator*( ); }
49
50          iterator & operator++ ( )
51          {
52              current = current->next;
53              return *this;
54          }
55
56          iterator operator++ ( int )
57          {
58              iterator old = *this;
59              ++( *this );
60              return old;
61          }
62
63        protected:
64          iterator( Node *p ) : const_iterator( p )
65            { }
66
67          friend class List<Object>;
68      };
```

59

```
1       // Erase item at itr.
2       iterator erase( iterator itr )
3       {
4           Node *p = itr.current;
5           iterator retVal( p->next );
6           p->prev->next = p->next;
7           p->next->prev = p->prev;
8           delete p;
9           theSize--;
10
11          return retVal;
12      }
13
14      iterator erase( iterator start, iterator end )
15      {
16          for( iterator itr = from; itr != to; )
17              itr = erase( itr );
18
19          return to;
20      }
```

60

Applications

- Tons of applications for List DS
- depending on application will choose implementation
- basic sorts

61

Selection Sort

- Given an array of size n
- for every position 1..n
  - current is min
  - run through rest
  - swap if less

62

- lets do the code

63

```
void selectionsort(int numbers[],int size) {
int i,j,min,tmp;

for(i=0;i<size,i++) {
  min = numbers[i];
  for(j=i; j<size; j++) {
      if(min> numbes[j]) {
            tmp = numbers[j];
            numbers[j] = min;
            min = tmp;
      }
  }
numbers[i] = min;
}
```

64

- run time ?

65

Bucket sort

- this is a really good sort for values range < m with distinct values
- create an array of size m, for each item throw it in the correct bucket
- when done read off all values in sorted order
- what is the run time ?

66

Radix sort

- variation of bucket sort
- anyone hear of this ?

67

- need to know largest value
- will iterate over all digits from last to first on each iteration

68

Example

- 155
- 024
- 197
- 922
- 874
- 137
- 256
- 156
- 207
- 027

- lets do each iteration

69

- run time ?

70

Analysis

- O (P (N + B))
- N = number of values
- B = number of buckets
- P = number of passes
- linear because it grows slowly in relation to n

71

C++ issue

- in many dynamic DS the gain from reorganizing the DS will be lost with the overhead of new/delete
- any ideas ??

72

Solution

- allocate memory in large blocks
- will not be doing this generally, but should be aware of the technique
- example, create a class to ask for space, which allocates a 100 items at a time, giving back references until it needs to ask for another block of 100

73

Next Time

- Finish homework
- Reading:
  - chapter 3
  - 4.1
- Will be releasing homework 2 Wednesday

74