Data Structures TA Session #2

By Po-Chuan & Pei-Hsuan

104/11/02

Preface

Few things before we start

When doing you homework

1. Always write **in nice & clear format**.
2. Please **DO indent** your code, and use monospaced fonts.
3. Answer all requirements one by one. Don't skip them!
4. Read the problems carefully. **Answer what they want!** (Not all problems ask you to write code.)

Assignment #2

Problem 2-1

Recursive function: getSum()

What does this problem want?

- List the criteria of a recursive function
- **State how the function meets the criteria**
- Refer to your textbook for more information
- This problem doesn’t ask you to write code, nor does it ask to draw a demonstration graph

Criterion A.

- Define the problem in terms of a smaller problem of the same type
- One action of `getSum()` is to *call itself*
- Calculation of sum is made by adding the first element to the sum of the remaining array, which is smaller than the current array

Criterion B.

- How does each recursive call diminish the size of the problem?
- At each recursive call to `getSum()`, the size of the array you need to compute is *diminished by* `1`

Criterion C.

- What instance of the problem can serve as the base case?
- The function handles the sum of x differently from all the other ones: It does not generate a recursive call. Rather, you know that `getSum(x)` is the element itself (`x[lower]`). Thus, the *base case* occurs when *lower=upper*.

Criterion D.

- As the problem size diminishes, will you reach this base case?
- Given that `0 <= lower <= upper`, criterion B assures you that you will always *reach the base case*.

Grading policy

- There are 4 questions you need to ask when writing a recursive function
- Each accounts for 5 points

Problem 2-9

Digit sum of a given positive integer

Let’s think about recursion

- What’s the base case?
- When N < 10, or N == 0
- What’s the answer of that?
- The sum is the digit itself
- What’s the recursion formula?
- `N % 10 + getSum( N / 10 )`

```
int getSum( int n )
{
   if ( n < 10 ) // or if ( n == 0 )
      return n;
   else
      return n % 10 + getSum( n / 10 );
}
```

```
int getSum( int n )
{
   return n < 10? n : n % 10 + getSum( n / 10 );
}
```

Grading policy

• The base case	5
• The answer for the base case	5
• Other cases	5
• The answer for other cases	5

Problem 2-16

Box trace of binary search

Box trace

- Please refer to your textbook
- Shows the information during each iteration of the process

Part A. Box 1 & 2

- Target = 5
- First = 0
- Last = 7
- Mid = 3
- Target < x[3]
- Search the left part

- Target = 5
- First = 0
- Last = 2
- Mid = 1
- Target = x[1]
- Return 1

Part B. Box 1 & 2

- Target = 13
- First = 0
- Last = 7
- Mid = 3
- Target > x[3]
- Search the right part

- Target = 13
- First = 4
- Last = 7
- Mid = 5
- Target < x[5]
- Search the left part

Part B. Box 2 & 3

- Target = 13
- First = 4
- Last = 7
- Mid = 5
- Target < x[5]
- Search the left part

- Target = 13
- First = 4
- Last = 4
- Mid = 4
- Target < x[4]
- Search the left part!

Part B. Box 3 & 4

- Target = 13
- First = 4
- Last = 4
- Mid = 4
- Target < x[4]
- Search the left part!

- Target = 13
- First = 4
- Last = 3
- Mid = 3
- First > Last
- Return -1 (not found)

Grading policy

- Part A: 2 boxes, 5 points each
- Part B: 4 boxes, 2.5 points each (your score is rounded up to the nearest integer)

Problem 2-19

Indent the rabbit function

How to solve this problem?

- Keep the recursion depth, either as a parameter or as a global variable
- Print the information of each function call after tabs

```
int rabbit( int n, string prefix = "" )
{
    int child = n <= 2;

    cout << prefix << "Enter rabbit: n = " << n << endl;

    if( n > 2 )
        child += rabbit( n - 1, prefix + '\t' ) +
rabbit( n - 2, prefix + '\t' );

    cout << prefix << "Leave rabbit: n = " << n << "
value = " << child << endl;

    return child;
}
```

Grading policy

Function prototype	2
Indention	4
The “enter” statement	3
The “leave” statement	3
Base case	1
Recursive call	3
Return the answer of rabbit()	3
Syntax correctness	1

Problem 2-23

Euclidean algorithm

Part A. the proof

To prove gcd(a, b) = gcd(b, a mod b), given ab != o

Let X = gcd(a, b), then let a = mX, b = nX.

Let a = bq + r, q and r are integers, o <= r < b

a - bq = r

By **Common Divisor Divides Integer Combination**,

1. all common divisors of a and b divide r (from a – bq = r)
2. all common divisors of b and r divide a (from bq + r = a)

Part A. the proof (cont.)

1. all common divisors of a and b divide r (from a – bq = r)
2. all common divisors of b and r divide a (from bq + r = a)

Every common factor of (a, b) will appear in common factors of (b, r), or (b, a mod b). (The reverse also holds.)

Therefore, these 2 sets are equal.

gcd(a, b) = gcd(b, a mod b)

Part B. when b > a…

- Suppose b > a in `gcd( a, b )`
- The next recursive call will be
  `gcd( b, a mod b ) = gcd( b, a )`
- The recursion swaps these 2 numbers and continues as usual without a problem

Part C-1. Will it end?

- If b | a, then b is their GCD, and the function ends immediately (though the problem excludes such cases)
- Otherwise, the parameters a and b will get smaller each time
- Also assume that **a > b > 0** (refer to part B when b > a)
- **a > b**, and **b > a mod b** (by definition)
- But a and b are both always greater than 0, so termination of the process can be done in finite steps

Part C-2: Why the base case is appropriate?

- When a mod b = 0, b is the greatest common divisor of a and b (by definition.)
- Therefore, no further recursion calls are required, and the base case is appropriate.

Grading policy

• The proof	4
• When b > a	10
• Reach the base case	3
• Why is base case appropriate	3

Assignment #3

Exercise 1.9

a.

```
cout << p.coefficient( p.degree() );
```

b.

```
p.changeCoeffcient( p.coefficient( 3 ) + 8, 3 );
```

Exercise 1.9

c.

```
polynomial<double> add( polynomial<double> a,
polynomial<double> b )
{
      polynomial<double> sum;
      int high = max( a.degree(), b.degree() );

      for ( int i = 0; i <= high; ++i )
            sum.changeCoeffcient( a.coefficient( i )
+ b.coefficient( i ), i );
      return sum;
}
```

Exercise 1.9 - Grading Policy

- a(5)
  - correctness 1, display result 2, using ADT operation 2
- b(5)
  - correctness 3, using ADT operation 2
- c(10)
  - correctness 4, syntax correctness 3, using ADT operation 3

Exercise 2.24(a)

c(1) = 0
c(2) = 1
c(3) = 3
c(4) = 7
c(5) = 15

```
C(n) = 0, when n=1
     = 1, when n=2
     = 2*C(n-1)+1, otherwise
```

```
int C(int n)
{
      if(n==1) return 0;
      else if(n==2) return 1;
      else return 2*C(n-1)+1;
}
```

Exercise 2.24(b)

c(1) = 0
c(2) = 1
c(3) = 2
c(4) = 4
c(5) = 6
c(6) = 10
c(7) = 14
c(8) = 21
c(9) = 29

```
C(n) = b(n,n-1)
b(n, m) = 0, when n<1 or m<1
        = 1, when n=1 or m=1
        = 1+b(n,n-1), when n=m
        = b(n-m, m)+b(n,m-1), otherwise
```

```
int b(int n, int m)
{
    if(n<1||m<1) return 0;
    else if(n==1||m==1) return 1;
    else if(n==m) return 1+b(n,n-1);
    else return b(n-m,m)+b(n,m-1);
}
```

Exercise 2.24(b)

- Some reference for you
- OEIS: online encyclopedia of integer sequences
- A000041 number of partitions of n
- A000065 -1 + number of partitions of n.

Exercise 2.24 - Grading Policy

- Each has 10 points
- Recursion function, 4
- Definition correctness, 6

Exercise 3.1

```
Class ArrayBag: public BagInterface<ItemType>
{
   ...
public:
        double getAvg() const;
}
```

```
double ArrayBag::getAvg() const
{
    double avg = 0, sum = 0;
    for ( int I = 0; I < itemCount; i++)
        sum += items[i];
    avg = sum / itemCount;
    return avg;
}
```

Exercise 3.1

- Reference: [accumulate()]() in [#include<numeric>]()
- You can use it to sum up the values

Exercise 3.1 - Grading Policy

- using client function, 4
- function correctness, 4
- syntax correctness, 2

Exercise 3.5

```
class Inventory {

private:

   string name;
   int cost, quantity;

public:

   Inventory( const string Name, const int Cost,
const int Quantity ): name( Name ), cost( Cost ),
quantity( Quantity ) {}

Inventory(): name( "" ), cost( 0 ), quantity( 0 ) {}
```

Exercise 3.5 (cont'd)

```
string getName() const { return name; }
void setName( const string val ) { name = val; }
int getCost() const { return cost; }
void setCost( const int val ) { cost = val; }
int getQuantity() const { return quantity; }
void setQuantity( const int val ) { quantity = val; }


}
```

Exercise 3.5 - Grading Policy

- class, 4
- syntax correctness, 5
- operation (look at, change value), 16
- attribute (product, price, quantity, date, rating…), 15

Exercise 3.9

```
template<typename T>
ArrayBag<T>::ArrayBag( const int val[], int size )
{
    itemCount = std::min( size, DEFAULT_CAPACITY );
    maxItems = DEFAULT_CAPACITY;
    for ( int i = 0; i < itemCount; ++i )
        items[ i ] = val[ i ];
}
```

Exercise 3.9

- Reference: copy() in #include<algorithm>
- You can use it to copy the values into the bag

Exercise 3.9 - Grading Policy

- template, 2
- constructor function, 2
- initializing itemCount and maxItems, 2
- create a bag, 3
- syntax correct, 1

The end~

Hope you did a good job in this assignment.

Average score for assignment #2 & #3 is 79.7

By the TAs

104/11/02