School of Computer Science

COMP SCI 1103/2103 Algorithm Design & Data Structure

Recursion 4

adelaide.edu.au

seek LIGHT

Previously on ADDS

- Recursion
  - Checklist
  - Recursive helper function
  - Tail recursion
  - Memorization
- Indirect Recursion
  - Harder to track and control
  - Example: processing arithmetic expressions

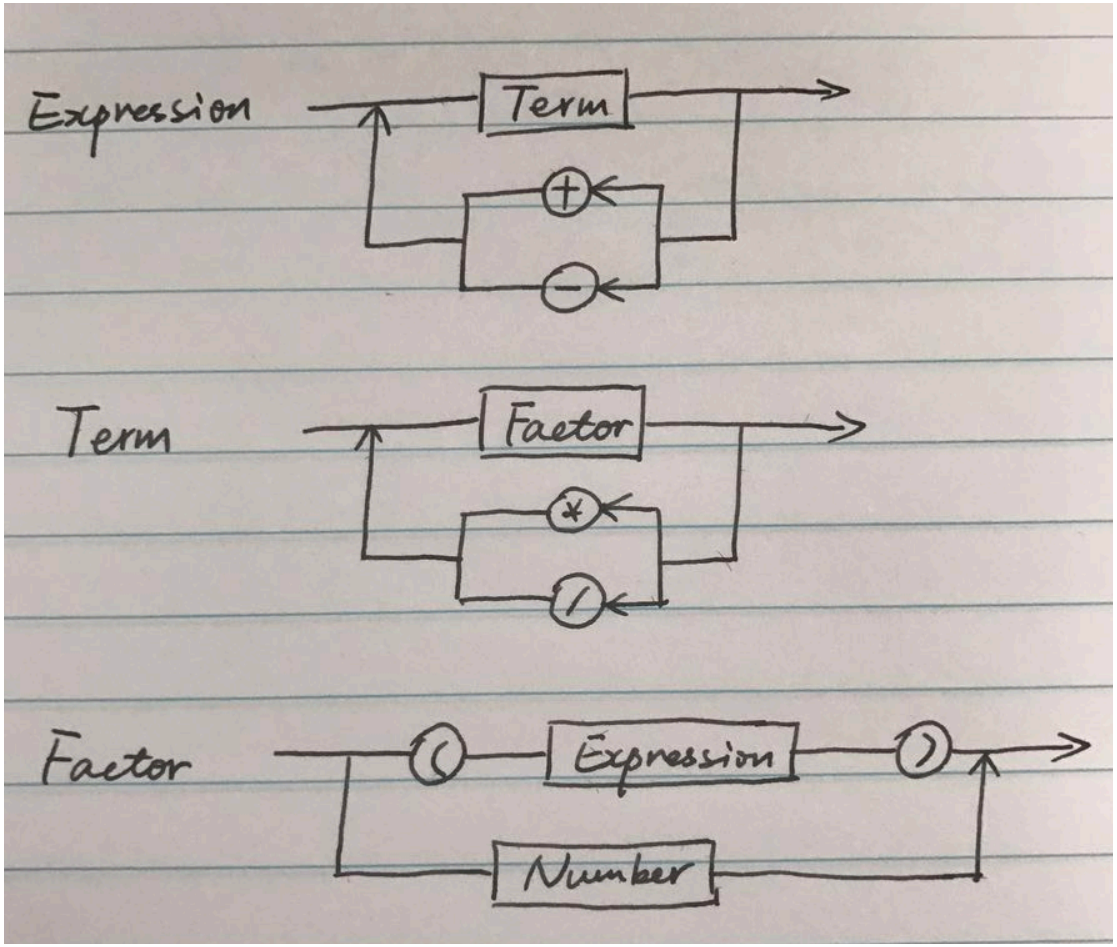
Overview

- In this lecture we will:
  - See a sample code for a function that can process arithmetic expressions (indirect recursion)
  - Discuss Dynamic Programming
    - Fibonacci
    - Counting Coin Problem

Example

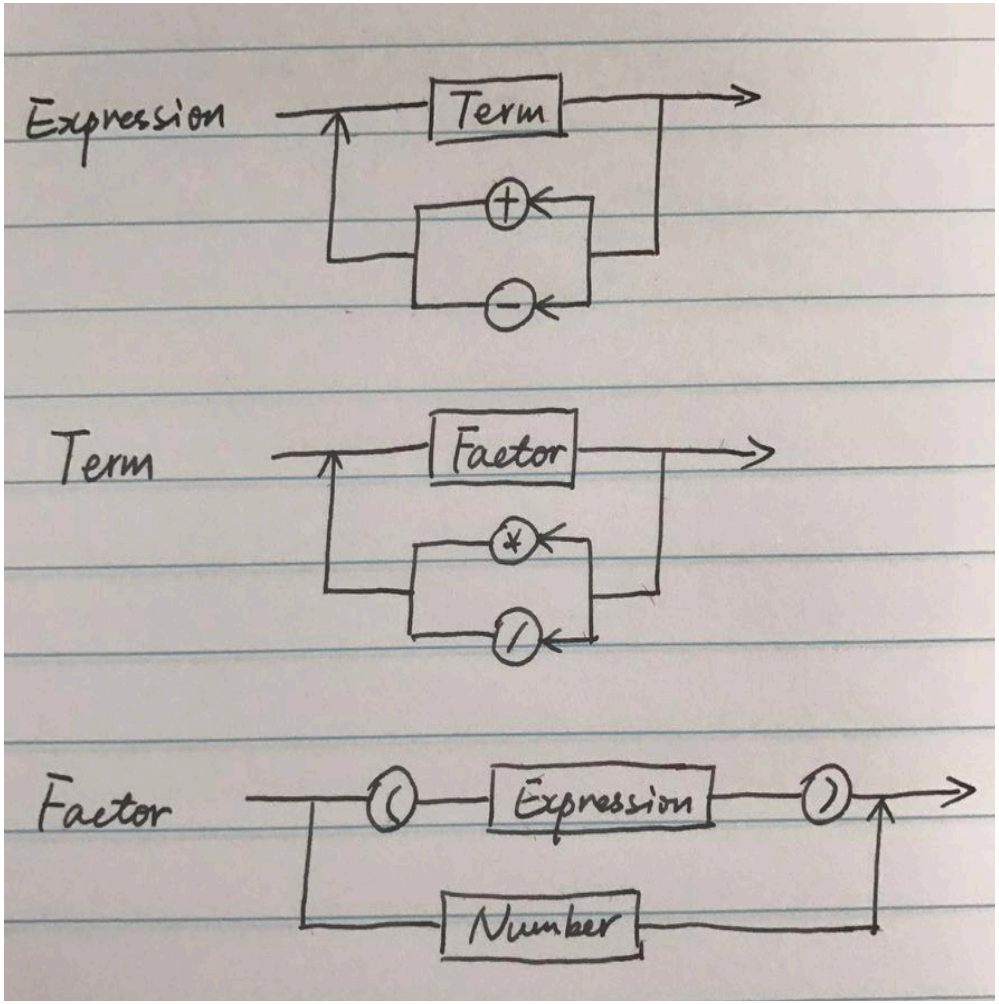
- Compute the values of arithmetic expressions

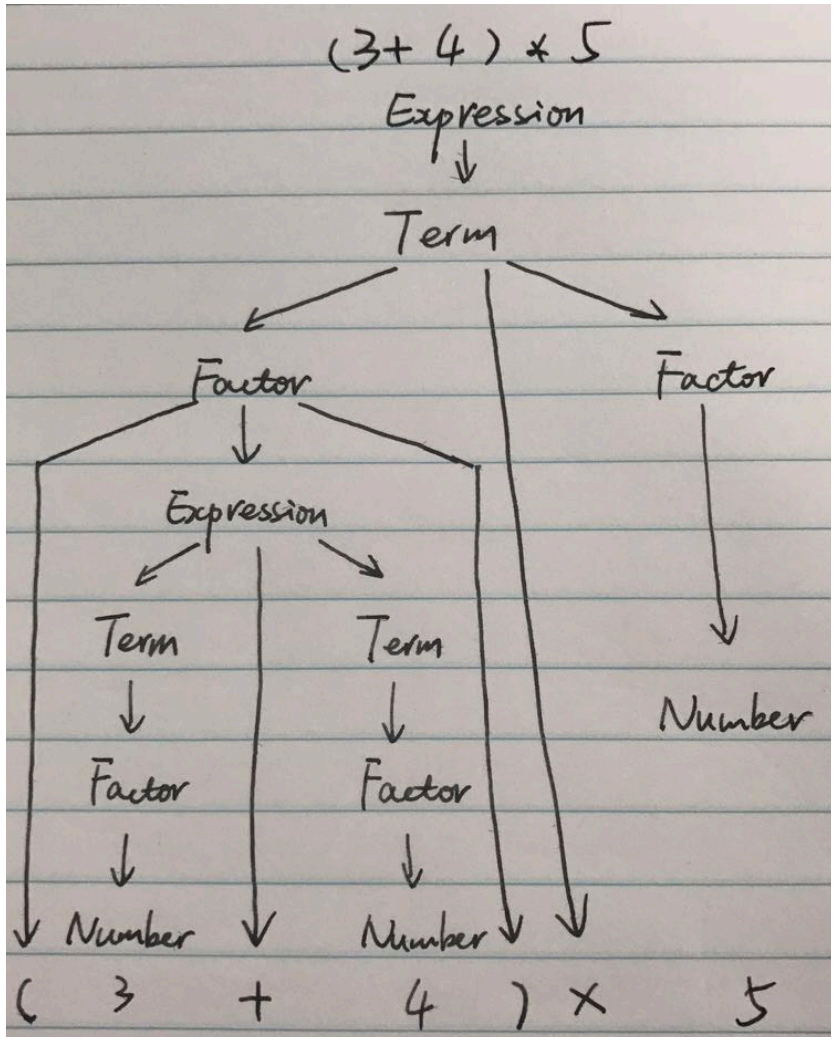
- Example
  - 3+4*5
  - (3+4)*5



Example

- (3+4*5)


Expression
Term
+
-
Term
Factor
*
/
Factor
(
Expression
)
Number


(3+4)*5
Expression
Term
Factor
Factor
Expression
Term
Term
Factor
Factor
Number
Number
Number
(
3
+
4
)
×
5

Sample code for getExptession

- Assume we have
  - getChar()
    looks ahead in input string
  - removeChar()
    removes a char from input string
  - getNumber()
    reads a number from the input string and removes the corresponding characters from that

A sample code

```
int getExp(){
   int value = getTerm();
   char next = getChar();
   while ((next ==‘+’) ||(next==‘-’)){
            removeChar();
            int value2 = getTerm();
            if(next ==‘+’)
               value += value2;
            else
               value -= value2;
   }
   return value;
}
```

```
int getTerm(){
   int value = getFactor();
   char next = getChar();
   while ((next ==‘*’) ||(next==‘/’)){
            removeChar();
            int value2 = getFactor();
            if(next ==‘*’)
               value *= value2;
            else
               value /= value2;
   }
   return value;
}
```

```
int getFactor(){
   int value;
   char next = getChar();
   if(next==‘(’ ){
         removeChar();
         value = getExp();
         removeChar(); // discard ‘)’
   }else
      value = getNumber();
   return value;
}
```

Example

- For calculating (3+4)*5
- getExp()
  - getTerm()
    - getFactor() ->consume ‘(’
      - getExp() ->3+4, return 7
    - ->consume ‘)’
    - getFactor() –> return 5
  - 7*5 -> return 35
- return 35

Dynamic Programming

- What do we do in dynamic programming?
  - Again, break the problem down to some smaller subproblems
  - solving each of them once
  - storing the solutions into some data structure (usually a table).

- Similar to Memorization, but DP does the computation from problems with smaller values to larger values.

- We need base cases and recursion relationship as well.

Example 1

- Let's look back to the Fibonacci number again.

- What about calculating in a bottom-up order?

```
int fib(int n){
  int * fibTable= new int[n];

  fibTable[0] = fibTable[1] = 1;

  for(int i=2; i<n; i++){
    fibTable[i] = fibTable[i-1]+fibTable[i-2];
  }

  return fibTable[n-1];
}
```

Example 2

- Counting Coins
- Given a value n, if we need n cents, how many ways can we make the change?
  - Assume infinite supply of each kind of coins
- int coinValue[] = {5, 10, 20, 50, 100, 200};

- How many configurations can you find for 20 cents?

- What is the result for value n when you include i (0<=i<=maxPossible) coins of value x?
  - The result for n-i*x, without considering coins of value x for that.

Counting Coins Recursive Algorithm

```
int facevalue[] = {5, 10, 20, 50, 100, 200};
int count(int n, int coinIndex){

    if (n == 0)
            return 1;
    if (n < 0)
            return 0;
    if (coinIndex >= 6 )
            return 0;

   int counter = 0;
   for (int i = 0; i<= n/facevalue[coinIndex]; i++)
            counter += count(n - i*facevalue[coinIndex], coinIndex+1);

  return counter;
}
```

How can we also do this loop recursively?

Example 2 version 2

- Counting Coins
- Given a value n, if we need n cents, how many ways can we make the change?
  - Assume infinite supply of each kind of coins

- int coinValue[] = {5, 10, 20, 50, 100, 200};

- What is the result for value n when you include 0 or 1 coins of value x?
  - The results for n and next type of coins + the results for n-x and coins of value x again.

Counting Coins Recursive Algorithm v2

```
int facevalue[] = {5, 10, 20, 50, 100, 200};
int countV2(int n, int coinIndex){

    if (n == 0)
            return 1;
    if (n < 0)
            return 0;
    if (coinIndex >= 6 )
            return 0;

     return countV2(n, coinIndex+1) + countV2(n-facevalue[coinIndex], coinIndex);
}
```

Example 2

	Coin types used					
						5
					10	10
				20	20	20
			50	50	50	50
		100	100	100	100	100
Total Amt	200	200	200	200	200	200
0	1	1	1	1	1	1
5	0	0	0	0	0	1
10	0	0	0	0	1	2
15	0	0	0	0	0	2
20	0	0	0	1	2	4
25	0	0	0	0	0	4
30	0	0	0	0	2	6
35	0	0	0	0	0	6

Counting Coins with DP

```
int dptable[1000][6];

int countDP (int n, int coinIndex){
  // initialize the first row
  for(int j=0; j<6; j++){
    dptable[0][j] = 1;
  }

  //fill in the table downwards
  for(int i=1; i<=n; i++){
    for(int j=5; j>=0; j--){
      if(i>=facevalue[j])
          dptable[i][j] = dptable[i][j+1] + dptable[i-facevalue[j]][j];
      else
          dptable[i][j] = dptable[i][j+1];
    }
  }
  return dptable[n][coinIndex];
}
```

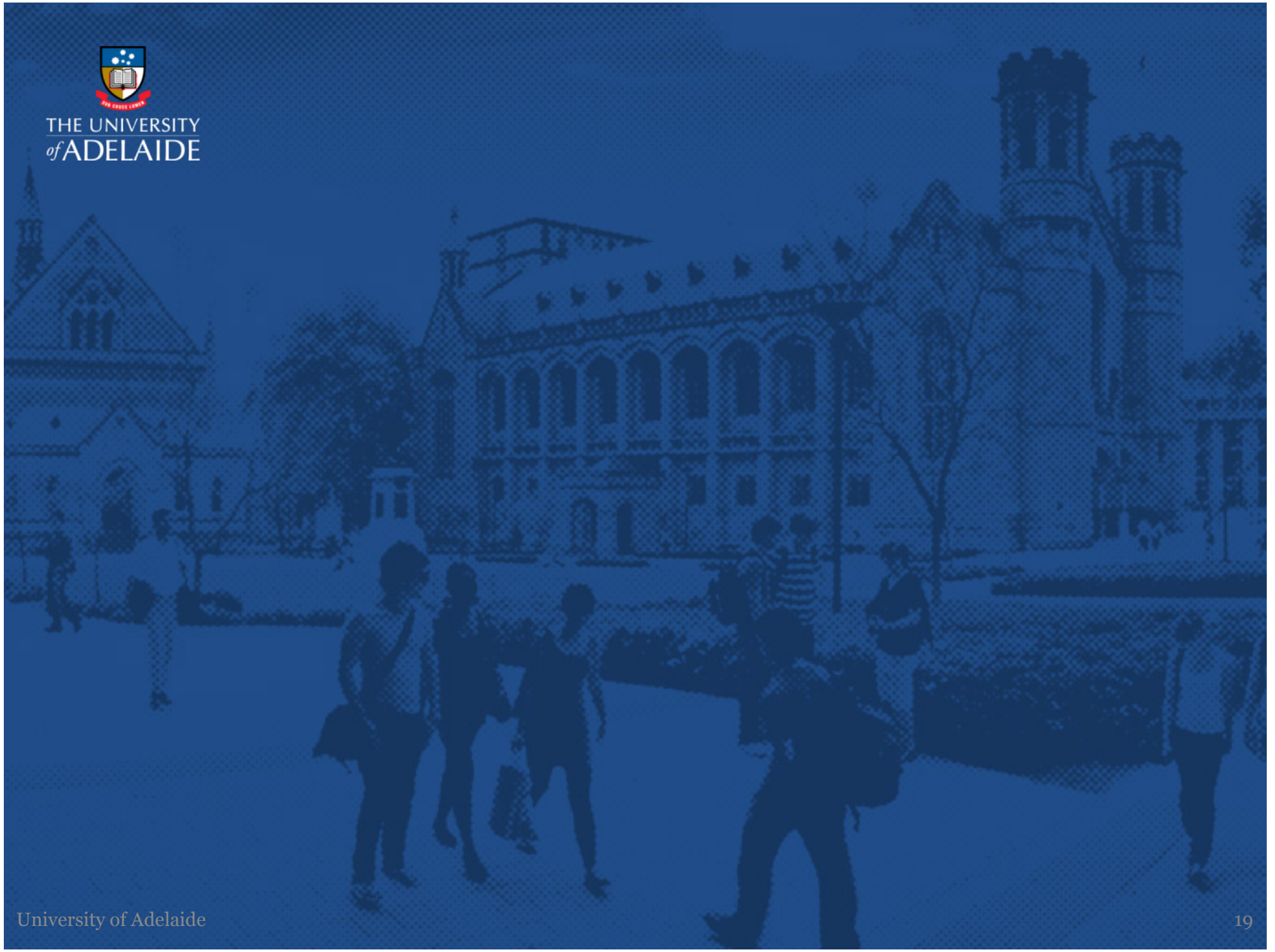
Dynamic Programming

- DP is just about filling table.

- Memory usage!
  - In some situations we can save the memory usage by reusing the table, e.g. using a circular data structure.

Summary

- Indirect recursion, we saw an example
- Dynamic programming
  - Fill out some tables and use the values recursively


THE UNIVERSITY
of ADELAIDE