Lecture 1 Introduction

CS101 Algorithms and Data Structures

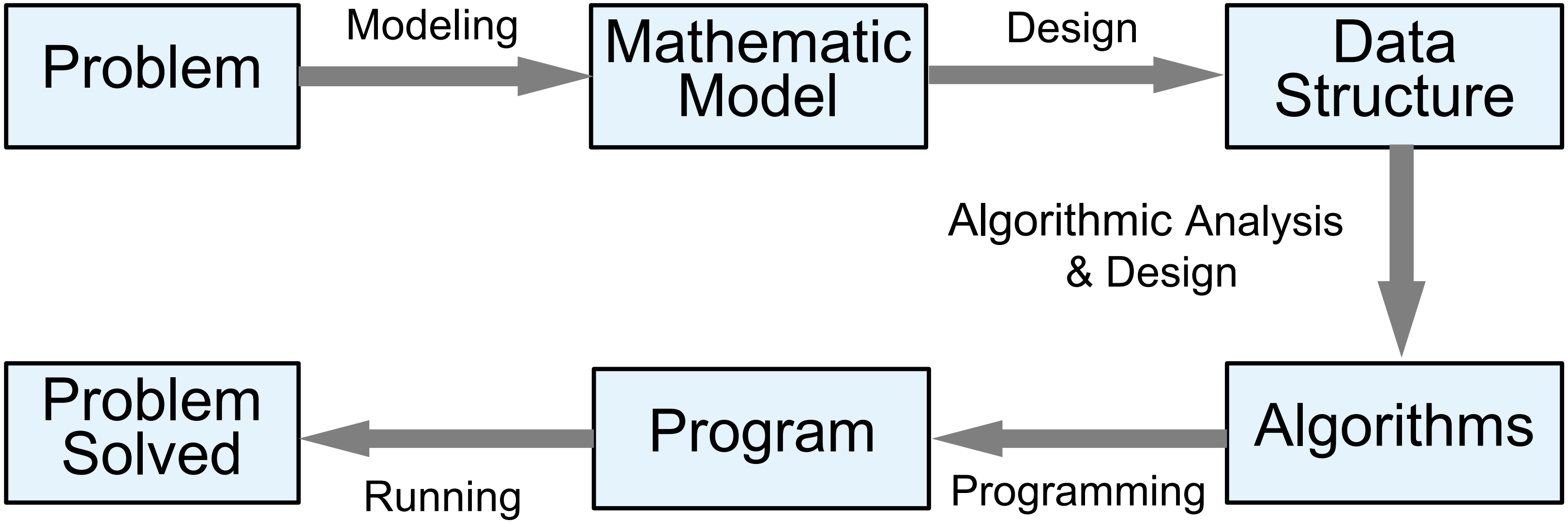
Instructor：Dengji Zhao; Yuyao Zhang; Xin Liu; Hao Geng

2022-09-05

Definition of Data Structure

- A data structure is a scheme for organizing data in the memory of a computer.
- The way in which the data is organized affects the performance of an algorithm for different tasks.
- 数据结构（data structure）是计算机中存储、组织数据的方式。通常情况下，精心选择的数据结构可以带来最优效率的算法（algorithm）。

How to combat problems via a computer



Ex1 How to arrange books on the bookshelf?

Ex1 How to arrange books on the bookshelf?

- The following two operations are essential for efficiently arranging your books:
  - **Operation 1:** how to insert new books?
  - **Operation 2:** how to find/access an existing book?

Ex1 How to arrange books on the bookshelf?

- **Method 1:** randomly insert new books.
  - **Operation 1:** how to insert new books?

    Insert the book wherever there is an available space.

    Nice and easy!
  - **Operation 2:** how to find/access an existing book?

    It depends …

Ex1 How to arrange books on the bookshelf?

- **Method 2:** insert new books according to the alphabets order of the first letter.
  - **Operation 1:** how to insert new books?

  EX: we bought a new book “Algorithm”.

  - **Operation 2:** how to find/access an existing book?

  EX: Binary search!

Ex1 How to arrange books on the bookshelf?

- **Discussion 1:** is **Method 2** absolutely better/more efficient than **Method 1**?

Method 1: randomly insert new books.

Method 2: insert new books according to the alphabets order of the first letter.

- **Discussion 2:** how can we further improve **Method 2**?

Ex1 How to arrange books on the bookshelf?

- **Method 3: cluster books according to different topics** (computer science, economics, agriculture, politics…), then insert new books according to the alphabets order of the first letter.
  - **Operation 1:** how to insert new books?

  EX: we bought a new book “Algorithm”.

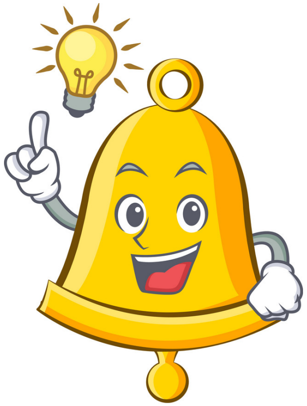
  - **Operation 2:** how to find/access an existing book?

  EX: Binary search for topic first, then binary search for book title.

- **Discussion 3:** how much space we should preserve for each topic?

How many topics is an optimism option?

The efficiency of a method/algorithm highly depends on the organization&amount of the data.



Ex2 How to implement a function PrintN?

- Implement a function named PrintN, when input a positive integer N, print all the positive integer from 1 to N.

看似递归和循环的代码相似甚至于递归代码还比较方便但是在输入的数字为十万或者更多的时候，使用递归的程序却无法执行了。这恰恰反映了递归对程序的内存占用过大

```
void PrintN ( int N )
{ int i;
  for ( i=1; i<=N; i++ ) {
        printf( “%d\n” , i);
}
return;
}
```

Loop implementation

```
void PrintN ( int N )
{ if (N);
    PrintN( N-1);
    printf( “%d\n” , N);
}
return;
}
```

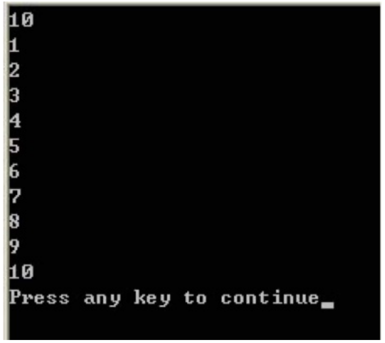
Recursive implementation

- Let N = 100, 1000, 10000, 100000, … …

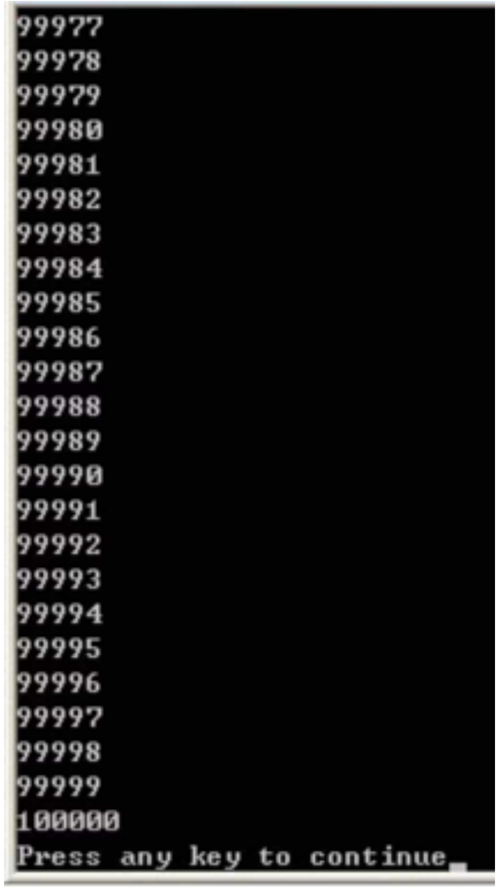
Ex2 How to implement a function PrintN?

- Implement a function named PrintN, when input a positive integer N, print all the positive integer from 1 to N.

```
# include <stdio.h>
void PrintN ( int N );
int main ()
{ int N;
  scanf (“%d”, &N);
  PrintN(N);
  return 0;
}
```



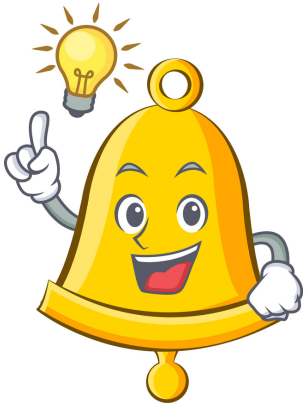
```
10
1
2
3
4
5
6
7
8
9
10
Press any key to continue_
```



```
99977
99978
99979
99980
99981
99982
99983
99984
99985
99986
99987
99988
99989
99990
99991
99992
99993
99994
99995
99996
99997
99998
99999
100000
Press any key to continue_
```

Loop implementation

The efficiency of a method/algorithm depends on the occupation of RAM.



Ex3 compute the summation for a polynomial at a fixed value x.

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_{n-1}x^{n-1} + a_nx^n$$

```
double fpoly1 ( int n, double a[ ], double x )
{ int i;
  double p = a[0];
  for (i = 1; i <=n; i++)
       p += (a[i] * pow( x, i) );
  return p;
}
```

$$f(x) = a_0 + x(a_1{+}x(a_2 + \cdots x(a_{n-1} + x(a_n))\cdots))$$

```
double fpoly2 ( int n, double a[ ], double x )
{ int i;
  double p = a[n];
  for (i = n; i > 0; i-- )
       p = a[i-1]  + x* p;
  return p;
}
```

- clock(): capture consumed time for running a function. The unit of the captured time is ***clock tick***, which depends on the CPU.
- CLOCKS_PER_SEC is a constant that presents the number of ***clock ticks*** per second.

```
#include <stdio.h>
#include <time.h>

clock_t  start, stop;
/* Clock_t is the variable returned by function clock(). */
double duration;
/* Record the running time for a function. Time unit is second. */
int main ()
{
  start = clock ();  /* Start timing. */
  Myfunction();
  stop = clock ();  /* Stop timing. */
  duration = ((double) (stop - start))/CLOCKS_PER_SEC;

 return 0;
}
```

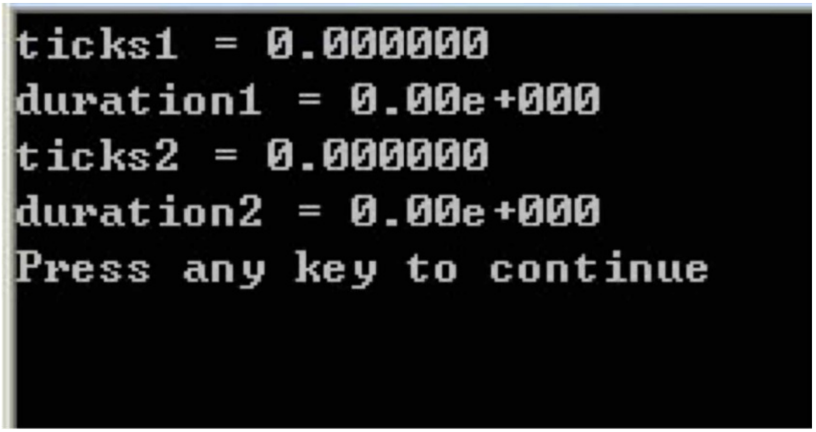
Ex3 compute the summation for a polynomial $f(x) = \sum_{i=0}^{9} i \cdot x^i$ at a fixed value $x = 1.1$, $f(1.1)$.

```
double fpoly1 ( int n, double a[ ], double x )
{ int i;
  double p = a[0];
  for (i = 1; i <=0; i++)
      p += (a[i] * pow( x, i) );
  return p;
}
```

```
double fpoly2 ( int n, double a[ ], double x )
{ int i;
  double p = a[n];
  for (i = n; i > 0; i-- )
      p = a[i-1]  + x* p;
  return p;
}
```

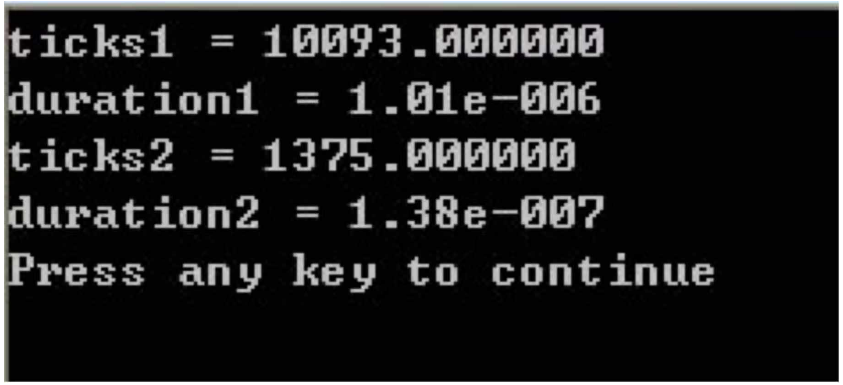
$$f(x) = \sum_{i=0}^{9} i \cdot x^i$$

```
#include <stdio.h> #include <time.h> #include <math.h>
clock_t start, stop;
double duration;
#define MAXN 10  /*maximum order of the polynomial */
double fpoly1 ( int n, double a[ ], double x )
double fpoly2 ( int n, double a[ ], double x )
int main ()
{  int i;
   double a[MAXN]; /*save the coefficient of the
polynomial*/
   for ( i=0; i<MAXN; i++) a[i] = (double) i;

   start = clock ();
   fpoly1(MAXN-1, a, 1.1);
   stop = clock ();
   duration = ((double) (stop - start))/CLOCKS_PER_SEC;
   prinft (“ticks1 = %f\n”,(double) (stop - start));
   prinft (“duration1 = %6.2e\n”, duration));

   start = clock ();
  fpoly2(MAXN-1, a, 1.1);
  stop = clock ();
  duration = ((double) (stop - start))/CLOCKS_PER_SEC;
  prinft (“ticks1 = %f”,(double) (stop - start));
  prinft (“duration1 = %6.2e\n”, duration));
 return 0;
}
```



```
ticks1  = 0.000000
duration1  = 0.00e+000
ticks2  = 0.000000
duration2  = 0.00e+000
Press any key to continue
```

```
#include <stdio.h>
#include <time.h>
#include <math.h>
......
#define MAXK 1e7
/*maximum repeat time of the test function */
......
int main ()
{ ......

  start = clock ();

  for ( i=0; i<MAXK; i++)
        fpoly1(MAXN-1, a, 1.1);  /* repeat the test function to get enough clock ticks*/

  stop = clock ();
  duration = ((double) (stop - start))/CLOCKS_PER_SEC/MAXK;
   /* compute running time for single function duration */
  prinft ("ticks1 = %f\n",(double) (stop - start));
  prinft ("duration1 = %6.2e\n", duration));


   ......

  return 0;
}
```



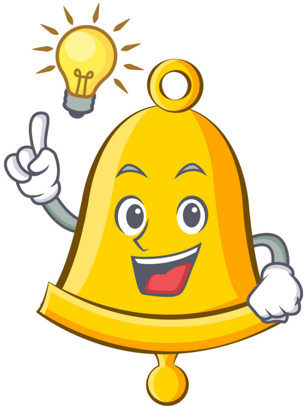
```
ticks1 = 10093.000000
duration1 = 1.01e-006
ticks2 = 1375.000000
duration2 = 1.38e-007
Press any key to continue
```

算法选择时效率的考虑：

虽然我们希望所选的算法占用额外空间小，运行时间短，其他性能也好，但计算机的时间和空间这两大资源往往相互抵触。所以，一般算法选择的原则是：

1. 对于反复使用的算法应选择运行时间短的算法；
2. 而使用次数少的算法可力求简明、易于编写和调试；
3. 对于处理的数据量较大的算法可从如何节省空间的角度考虑。

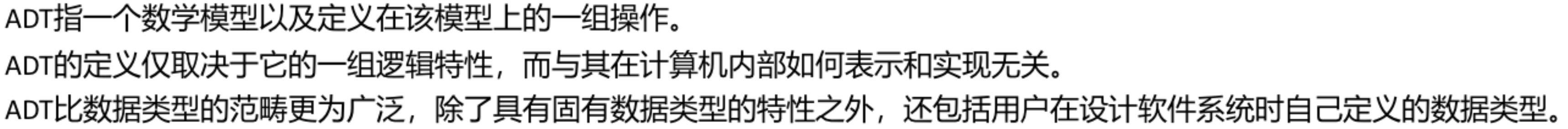
The efficiency of a method/algorithm depends on the design of the algorithm.



Definition of Data Structure

- ***Data structure***, way in which data are stored for efficient search and retrieval.
- Different data structures are suited for different ***operations***.
- ***Algorithm*** is a procedure for solving a mathematical problem in a finite number of steps that frequently involves repetition of an operation.

Abstract Data Type (ADT 抽象数据类型)



ADT指一个数学模型以及定义在该模型上的一组操作。

ADT的定义仅取决于它的一组逻辑特性，而与其在计算机内部如何表示和实现无关。

ADT比数据类型的范畴更为广泛，除了具有固有数据类型的特性之外，还包括用户在设计软件系统时自己定义的数据类型。

- Abstract: The method that we describe the data type, does not depend on the implementations.
  - Not related to the computer that stores the data.
  - Not related to the physical structure that stores the data.
  - Not related to the algorithm and language that implements the operation.

- We only care about “*how to design*” the objective data sets and related operations, not how to “*implement*” a data structure.

EX4 Abstract data type of a $\boldsymbol{matrix}$

Array?
Structural array? Orthogonal list?

- Data type: $Matrix$
- Objects: a $M{\times}N$ matrix $\underline{\mathrm{A}_{M\times N} = (a_{ij})}$ $(i = 1,\cdots,M; j = 1,\cdots,N)$ is composed by a number of $M{\times}N$ array of $< a,i,j >$, where a present the value of the matrix element, i present the no. of row, and j present the no. of column.
- Operations: for an arbitrary matrix $\mathrm{A},\mathrm{B},\mathrm{C} \in Matrix$, and integers i,j,M,N
- $Matrix\ create\ (int\ M, int\ N)$: return an empty matrix of $M{\times}N$;
- $int\ GetMaxRow(\ Matrix\ A\ )$: return the number of rows;
- $int\ GetMaxCol(\ Matrix\ A\ )$: return the number of columns;
- $\underline{ElementType}\ GetEntry(\ Matrix\ A, int\ i, int\ j)$: return the element of matrix A in row i, column j;
- M$atrix\ Add\ (Matrix\ A,\ \ Matrix\ B)$: if the dimension of matrix A and B are the same, return matrix $\underline{C = A + B,}$ otherwise error;
- $Matrix\ multiply\ (Matrix\ A,\ \ Matrix\ B)$: if the number of columns of matrix A is equals to the number of rows of matrix B, return matrix $C = AB$, otherwise return error;
- ……

The elements are added in order of rows or columns? C,C++,Python,…?

Course info

- **Instructors**
  - □ Yuyao Zhang zhangyy8@shanghaitech.edu.cn SIST 3-420
  - □ Dengji Zhao zhaodj@shanghaitech.edu.cn SIST 1A-304E
  - □ Xin Liu liuxin7@shanghaitech.edu.cn SIST 2-302H
  - □ Hao Geng genghao@shanghaitech.edu.cn SIST 3-332

Course Schedule

Week	Date	Content
1	9/05 Mon	Introduction
	9/07 Wed	Elementary Data Structures: Array and Lists
2	9/12 Mon	Mid-Autumn Festival
	9/14 Wed	Stack and Queue
3	9/19 Mon	Big O/Theta/Omega
	9/21 Wed	Hash Table
4	9/26 Mon	Sorting: Insertion, Bubble
	9/28 Wed	Sorting: Merge
5	10/03 10/05 Mon Wed	National Day
	10/08 Sat	Sorting: Quick
6	10/10 Mon	Divide and Conquer
	10/12 Wed	Trees: Introduction, DFS, BFS
7	10/17 Mon	Binary Trees
	10/19 Wed	Heap and Heap Sort
8	10/24 Mon	Binary Search Trees + Huffman Coding
	10/26 Wed	Balanced Binary Search Trees: AVL

Course Schedule

Week	Date	Content
9	10/31 Mon	Red-Black Tree + Disjoint sets 1
	11/02 Wed	Middle Term Exam
10	11/07 Mon	Disjoint sets 2+ Graphs: Introduction, Traversal
	11/09 Wed	Graphs: Traversal
11	11/14 Mon	Minimum Spanning Trees
	11/16 Wed	Greedy
12	11/21 Mon	Topological Sorts
	11/23 Wed	Shortest Path Algorithm: Dijkstra
13	11/28 Mon	A*
	11/30 Wed	Floyd-Warshall Algorithm
14	12/05 Mon	Dynamic Programming
	12/07 Wed	Knapsack Problem
15	12/12 Mon	Reductions
	12/14 Wed	P+NP
16	12/19 Mon	NPC
	12/21 Wed	Review

Course Policy

- **Grading**
  - □ Exams (45%): middle term: 20%; final: 25%
  - □ Weekly Homework (20%): non-programming questions
  - □ Programming Tasks (20%): 4 programming tasks (each lasts 3 weeks)
  - □ In-Class Quizzes (15%): in lectures and discussions