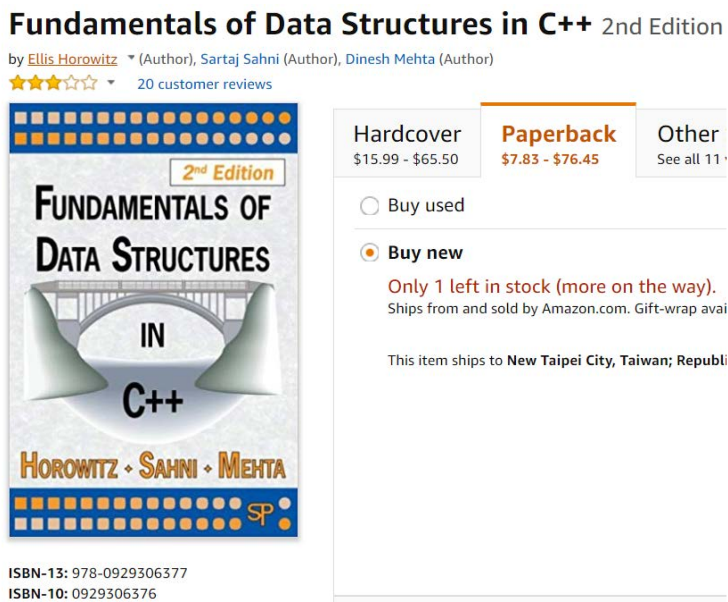
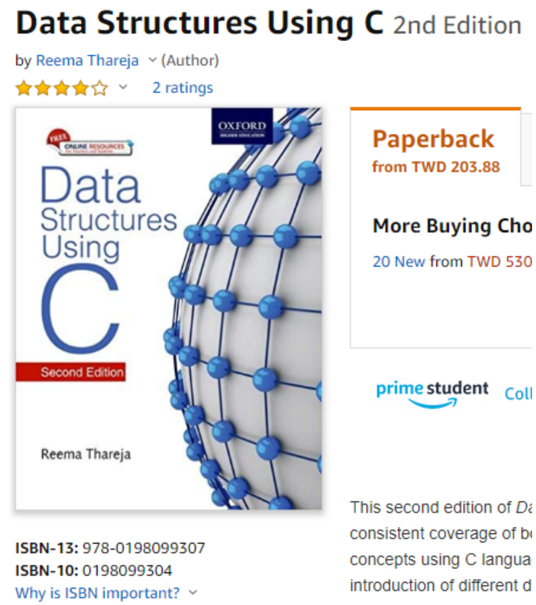
Program, Algorithm & Recursion

Kuan-Yu Chen (陳冠宇)

2020/09/16 @ TR-212, NTUST

Review





- Homework: 55%
- Midterm: 25%
- Final: 30%
  - Additional Enrollment: **ONLY** for senior students

Data Type

- Data type determines the set of values that a data item can take and the operations that can be performed on the item

Data Type	Size in Bytes	Range	Use
char	1	-128 to 127	To store characters
int	2	-32768 to 32767	To store integer numbers
float	4	3.4E-38 to 3.4E+38	To store floating point numbers
double	8	1.7E-308 to 1.7E+308	To store big floating point numbers

  - Note that C does not provide any data type for storing text
    - Text is made up of individual characters
    - “char” is supposed to store characters not numbers
    - In the memory, characters are stored in their ASCII codes

ASCII Codes

- In C language
  - ‘5’ == int 53
  - ‘5’-’0’==int 53 – int 48 == int 5
  - If char c = ‘B’+32, then c==‘b’

ASCII TABLE

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Program

- A program contains one or more functions, where a function is defined as a group of statements that perform a well-defined task
  - A program should undoubtedly give correct results, but along with that it should also run efficiently
- The definition of a good program is
  - runs correctly
  - easy to read and understand
  - easy to debug
  - easy to modify

```
main()
{
            Statement 1;
            Statement 2;
            ............
            ............
            Statement N;
}
Function1()
{
            Statement 1;
            Statement 2;
            ............
            ............
            Statement N;
}
Function2()
{
            Statement 1;
            Statement 2;
            ............
            ............
            Statement N;
}
..................
..................
FunctionN()
{
            Statement 1;
            Statement 2;
            ............
            ............
            Statement N;
}
```

Algorithm.

- The typical definition of algorithm is "a formally defined procedure for performing some calculation"
  - In general terms, an algorithm provides a blueprint to write a program to solve a particular problem
  - A program **does not** have to satisfy the fourth condition

Definition: An *algorithm* is a finite set of instructions that, if followed, accomplishes a particular task. In addition, all algorithms must satisfy the following criteria:

(1) **Input.** Zero or more quantities are externally supplied.

(2) **Output.** At least one quantity is produced.

(3) **Definiteness.** Each instruction is clear and unambiguous. (明確性)

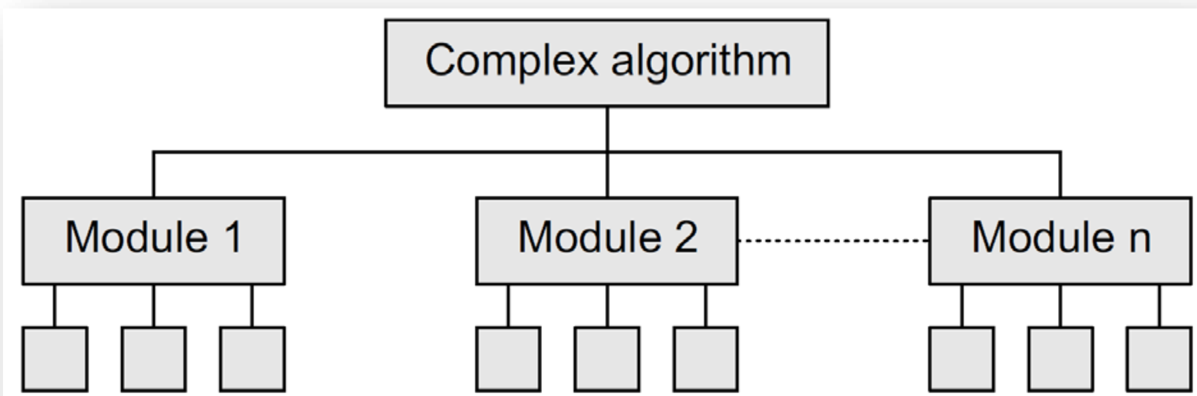
(4) **Finiteness.** If we trace out the instructions of an algorithm, then for all cases, the algorithm terminates after a finite number of steps. (有限性)

(5) **Effectiveness.** Every instruction must be basic enough to be carried out, in principle, by a person using only pencil and paper. It is not enough that each operation be definite as in (3); it also must be feasible. □ (有效性)

Algorithm..

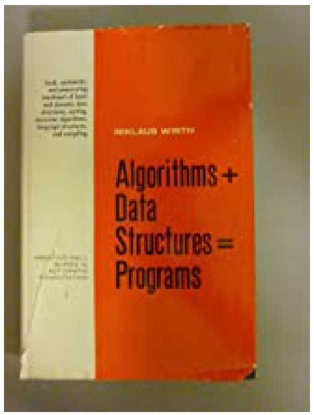
- A complex algorithm is often divided into smaller units called modules
  - This process of dividing an algorithm into modules is called modularization

- The key advantages of modularization are as follows:
  - It makes the complex algorithm simpler to design and implement
  - Each module can be designed independently



Each module can be divided into one or more sub-modules

Algorithm & Program



Algorithms + Data Structures = Programs (Prentice-Hall Series in Automatic Computation)

by Niklaus Wirth | Feb 1, 1976

17

Hardcover

TWD957.31 to rent

Only 2 left in stock - order soon.

More Buying Choices

TWD 249.62 (60 used & new offers)

Paperback

More Buying Choices

TWD 4,272.82 (5 used offers)

Function

- Let us analyze the reasons why segmenting a program (algorithm) into manageable chunks is an important aspect of programming
  - Dividing the program into separate well-defined functions facilitates each function to be written and tested separately
  - Understanding, coding, and testing multiple separate functions is easier than doing the same for one big function
  - Maintaining a huge program will be a difficult task
  - When a big program is broken into comparatively smaller functions, then different programmers working on that project can divide the workload by writing different functions

Recursive.

- The recursive mechanisms are extremely powerful, because they often can express a complex process very clearly

- Recursive functions can be categorized into three classes
  - Direct Recursion
    - The function may call itself before it is done
  - Indirect Recursion
    - The function may call other functions that again invoke the calling function
  - Tail Recursion
    - The function may call itself at the end of the function
    - A special case of direct recursion

Recursive..

Direct Recursion

```
function A()
{
    ...
    A() ;
    ...
}
```

Indirect Recursion

```
function A()
{
    ...
    B() ;
    ...
}
function B()
{
    ...
    A() ;
    ...
}
```

calling cycle

Tail Recursion

```
function A()
{
    ...
    A() ;
}
```

Recursive…

- Let's make a comparison

Recursion	Non-Recursion
Codes are more compact	Codes are complicated
Easy to understand	Hard to read
Time-consuming	Time-saving

Examples – 1

- Please write down a recursive program to do factorial

$$0! = 1$$

$$1! = 1$$

$$2! = 1 \times 2$$

$$3! = 1 \times 2 \times 3$$

$$n! = 1 \times 2 \times 3 \times \cdots \times n$$

```
int factorial( int a )
{
    if( a == 0 )
        return 1 ;
    else
        return factorial(a-1)*a ;
}
```

Examples – 2.

- Please (1) write a recursive program, $Fib(\ int\ a\ )$, to calculate the Fibonacci number; (2) how many function calls do you need to do when we want to calculate $Fib(5)$?

Fibonacci number

From Wikipedia, the free encyclopedia

In mathematics, the **Fibonacci numbers** are the numbers in the following integer sequence, called the **Fibonacci sequence**, and characterized by the fact that every number after the first two is the sum of the two preceding ones:[1][2]

$$1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$

Often, especially in modern usage, the sequence is extended by one more initial term:

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,\ 89,\ 144,\ \ldots$$ [3]

By definition, the first two numbers in the Fibonacci sequence are either 1 and 1, or 0 and 1, depending on the chosen starting point of the sequence, and each subsequent number is the sum of the previous two.

The sequence F_n of Fibonacci numbers is defined by the recurrence relation:

$$F_n = F_{n-1} + F_{n-2},$$

with seed values[1][2]

$$F_1 = 1,\ F_2 = 1$$

or[5]

$$F_0 = 0,\ F_1 = 1.$$

Examples – 2..

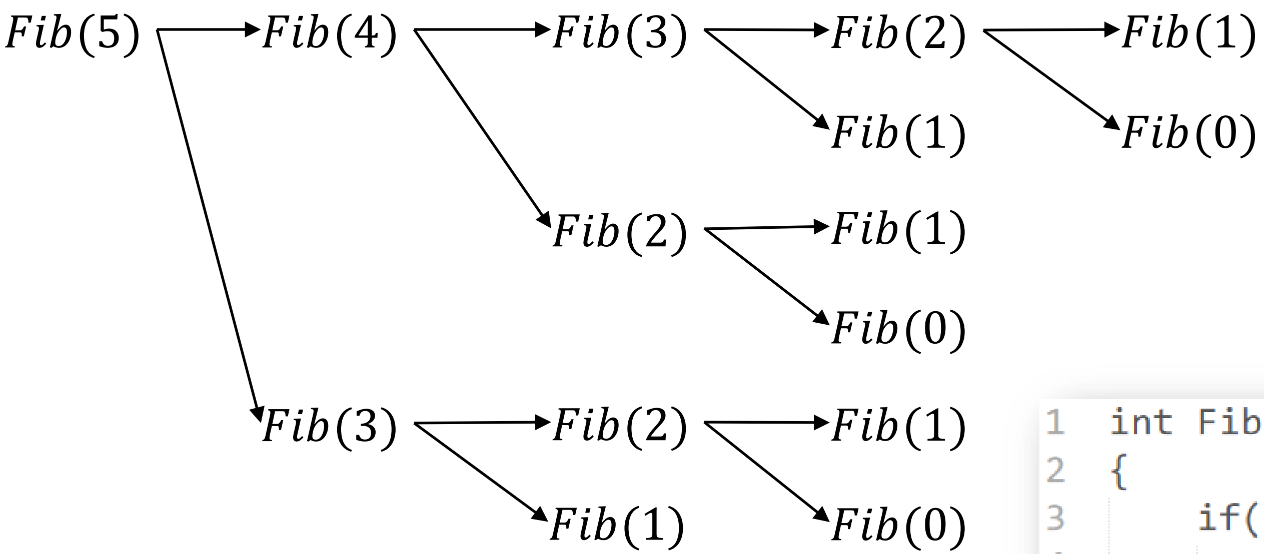
- Please (1) write a recursive program, $Fib(\ int\ a\ )$, to calculate the Fibonacci number; (2) how many function calls do you need to do when we want to calculate $Fib(5)$?

$$Fib(a) = \begin{cases} 0, & if\ a = 0 \\ 1, & if\ a = 1 \\ Fib(a-1) + Fib(a-2), & otherwise \end{cases}$$

```
int Fib( int a )
{
    if( a == 0 )
        return 0 ;
    else if( a == 1 )
        return 1 ;
    else
        return Fib(a-1)+Fib(a-2) ;
}
```

Examples – 2…

- Please (1) write a recursive program, $Fib(\ int\ a\ )$, to calculate the Fibonacci number; (2) how many function calls do you need to do when we want to calculate $Fib(5)$?




```
int Fib( int a )
{
    if( a == 0 )
        return 0 ;
    else if( a == 1 )
        return 1 ;
    else
        return Fib(a-1)+Fib(a-2) ;
}
```

Examples – 3.

- Given an Ackerman's function $A(m,n)$, please calculate $A(1,2)$.

$$A(m,n) = \begin{cases} n+1, & if\ m = 0 \\ A(m-1,1), & if\ n = 0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

$$A(1,2) = A\big(0, A(1,1)\big)$$

$$A(1,1) = A\big(0, A(1,0)\big)$$

$$A(1,0) = A(0,1)$$

$$A(0,1) = 2$$

Examples – 3..

- Given an Ackerman's function $A(m,n)$, please calculate $A(1,2)$.

$$A(m,n) = \begin{cases} n+1, & if\ m=0 \\ A(m-1,1), & if\ n=0 \\ A\big(m-1, A(m,n-1)\big), & otherwise \end{cases}$$

$$A(1,2) = A\big(0, A(1,1)\big) = A(0,3) = 4$$

$$A(1,1) = A\big(0, A(1,0)\big) = A(0,2) = 3$$

$$A(1,0) = A(0,1) = 2$$

$$A(0,1) = 2$$

Questions?

kychen@mail.ntust.edu.tw