

Data Structure & Abstract Data Type:

These two words are very similar. The differences between them are very subtle. In order to distinguish them easily, I shall give you a very detail introduction first.

Data Structure:

Definition: An organization of information, usually in memory, for better algorithm efficiency, such as queue, stack, linked list, heap, dictionary, and tree, or conceptual unity, such as the name and address of a person. It may include redundant information, such as length of the list or number of nodes in a subtree.

Abstract Data Type (ADT):

Definition: A set of data values and associated operations that are precisely specified independent of any particular implementation.

Such a data type is abstract in the sense that it is independent of various concrete implementations. The definition can be mathematical, or it can be programmed as an interface.

Difference between Data Structure and ADT:

An **abstract data type** is a general specification of the operations allowed on an object, without concerning the actual implementation of these operations, or how the data is stored in memory.

A **data structure** is a particular implementation of an abstract data type, and may serve a more specific purpose than the general abstract data type.

For example, stack and queue are ADT. Array is data structure.

In the whole training, I will teach you at least four ADT and one Data Structure.

ADT: Queue (隊列), Stack (棧), Tree (樹), Graph (圖)

Data Structure: Linked-List (鏈表)

One more Data Structure if possible: Heap (堆)

p.s. The Chinese Name is very meaningless and misunderstanding ...
Especially, for Heap and Stack ...

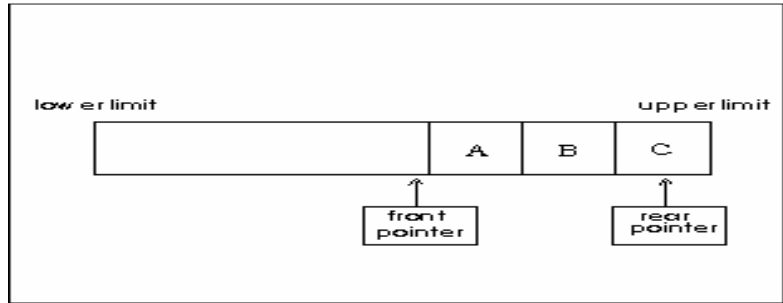
Unlike using array, when using abstract data type, we limit our operations. For example, in a queue, we can only insert value at the end, but we can't insert value at the top. Though, we can directly insert a value when using array.

Queue:

A queue in computer science,

Let q be a queue.

Has the following operations:



Empty(q);	Check if the queue q is empty
Full(q);	Check if the queue q is full.
Enqueue(q,item);	Put the item to the end of queue q.
Dequeue(q,item);	Let the first item of q go out.
Clear(q)	Clear the whole queue q.

This is a **First-In-First-Out (FIFO)** data type.

Real Life Example I:

Queuing up at a cinema to buy ticket:

You have to go in the queue at the end.

The first person of the queue can go to a counter to buy tickets.

Real Life Example II:

Drinks Selling Machine:

Drinks like coke, lemon tea are dropped from the top of the tunnel.

When you insert coins and choose a kind of drinks, the bottommost one will go out.

Pascal Implementation:

<pre>Type item=integer; {any data type} queue=record Data:array[1..100] of item; Front,rear:integer; End; Function Empty(var q:queue):Boolean; Begin Empty:=q.rear<q.front; End; Function Full(var q:queue):Boolean; Begin Full:=q.rear>=100; End; Procedure enqueue(var q:queue;k:item); Begin If not full(q) then Begin Inc(q.rear); q.data[q.rear]:=k; End; End; {Continue on R.H.S.}</pre>	<pre>{Continue from L.H.S.} function dequeue(var q:queue):item; begin dequeue:=q.data[q.front]; inc(q.front); end; procedure clear(var q:queue); begin fillchar(q,sizeof(q),0); q.front:=1; q.rear:=0; end; {End of Implementation} {Example of usage} {Demo of FIFO property} var q1:queue; I:integer; begin clear(q1); for I:=1 to 10 do enqueue(q1,i); while not empty(q1) do writeln(dequeue(q1)); end.</pre>
---	---

Stack:

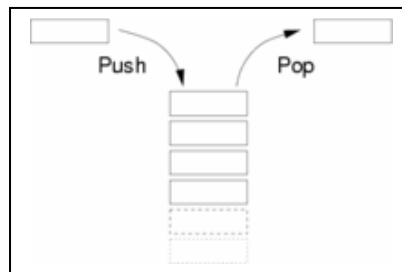
A stack is an abstract data type having **First-In-Last-Out (FILO)** property and the following operations.

Let s be a stack.

Empty(s);	Check if the stack s is empty
Full(s);	Check if the stack s is full.
Push(s ,item);	Put the item at the top of stack s .
Pop(s ,item);	Let the top item of s go out.
Clear(s)	Clear the whole stack s .

Real Life Example I:

Putting books to a one-side-opened box:
 You must put your book at the top.
 You can only take out the topmost book.



Pascal Implementation:

<pre>Type item=integer; {any data type} stack=record Data:array[1..100] of item; top:integer; End; Function Empty(var s:stack):Boolean; Begin Empty:=s.top<1; End; Function Full(var s:stack):Boolean; Begin Full:=s.top>=100; End; Procedure push(var s:stack;k:item); Begin If not full(s) then Begin Inc(s.top); s.data[s.top]:=k; End; End; {Continue on R.H.S.}</pre>	<pre>{Continue from L.H.S.} function pop(var s:stack):item; begin pop:=s.data[s.top]; dec(s.top); end; procedure clear(var s:stack); begin fillchar(s,sizeof(s),0); s.top:=0; end; {End of Implementation} {Example of usage} {Demo of FILO property} var s1:stack; I:integer; begin clear(s1); for I:=1 to 10 do push(s1,i); while not empty(s1) do writeln(pop(s1)); end.</pre>
---	--

Exercise I:

1999hse.doc	25-27,31,32
2001hse.doc	40
2002hje.doc	10,11,25,26
2004hse.doc	19
2005hje.doc	11,13,16,18,22,26

Exercise II:

Register an account at <http://acm.uva.es/problemset/>

This account will be used for the coming lessons.

Tell me your USER ID next lesson.

(Not only the numbers, but also the two alphabetical letters.)

Exercise III:

Complete all exercises and practical exercises in previous lessons.

Submit your program, (the source code) to my e-mail ami@fsstudio.net

Exercise IV: (1st – Aug)

If you have already known about ACM, (the website above)

Finish the following two questions.

<http://acm.uva.es/p/v104/10424.html>

<http://acm.uva.es/p/v104/10432.html>