Data Structure (I)

Luo Tsz Fung {pepper1208}

2025-03-12

Content

- Pointer
- C++ Structures
- Linked List
- Monotonic Data Structure

Content

- **Pointer**
- C++ Structures
- Linked List
- Monotonic Data Structure

Storing a value inside the computer

- We usually declare the variable like this:
- `int num = 5;`
- Intuitively, the variable “num” will allocate some memory in the CPU.
- How can the CPU identify the existence of the variable?
- Each variable is assigned with an **address**.

Reference variable

- A reference variable is a "***reference***" to an existing variable, and it is created with the & operator:

```
string school = "BSTC";
string &location = school;

cout << school << endl;
cout << location << endl;
```

```
BSTC
BSTC
```

- Look pretty useless?

Reference variable

- Let’s try the following program.

```
string school = "BSTC";
string &location = school;

location = "CTSB";

cout << school << endl;
cout << location << endl;
```

```
CTSB
CTSB
```

Reference variable

- The previous code segment shows the usage of a reference variable.
- When it ***references*** a variable, the change of the *reference variable* will also affect the value stored in the original variable begin *referenced*.
- You can try the following code segment to validate the idea:

```
string school = "BSTC";
string &location = school;
string &another_location = location;

another_location = "Computer Room";

cout << school << endl;
cout << location << endl;
cout << another_location << endl;
```

Memory address

- When a variable is created in C++, a memory address is assigned to the variable.
- When we assign a value to the variable, it is stored in this memory address.

Memory address

- Try the code segment below.

```
string school = "BSTC";

cout << school << endl;
cout << &school << endl;
```

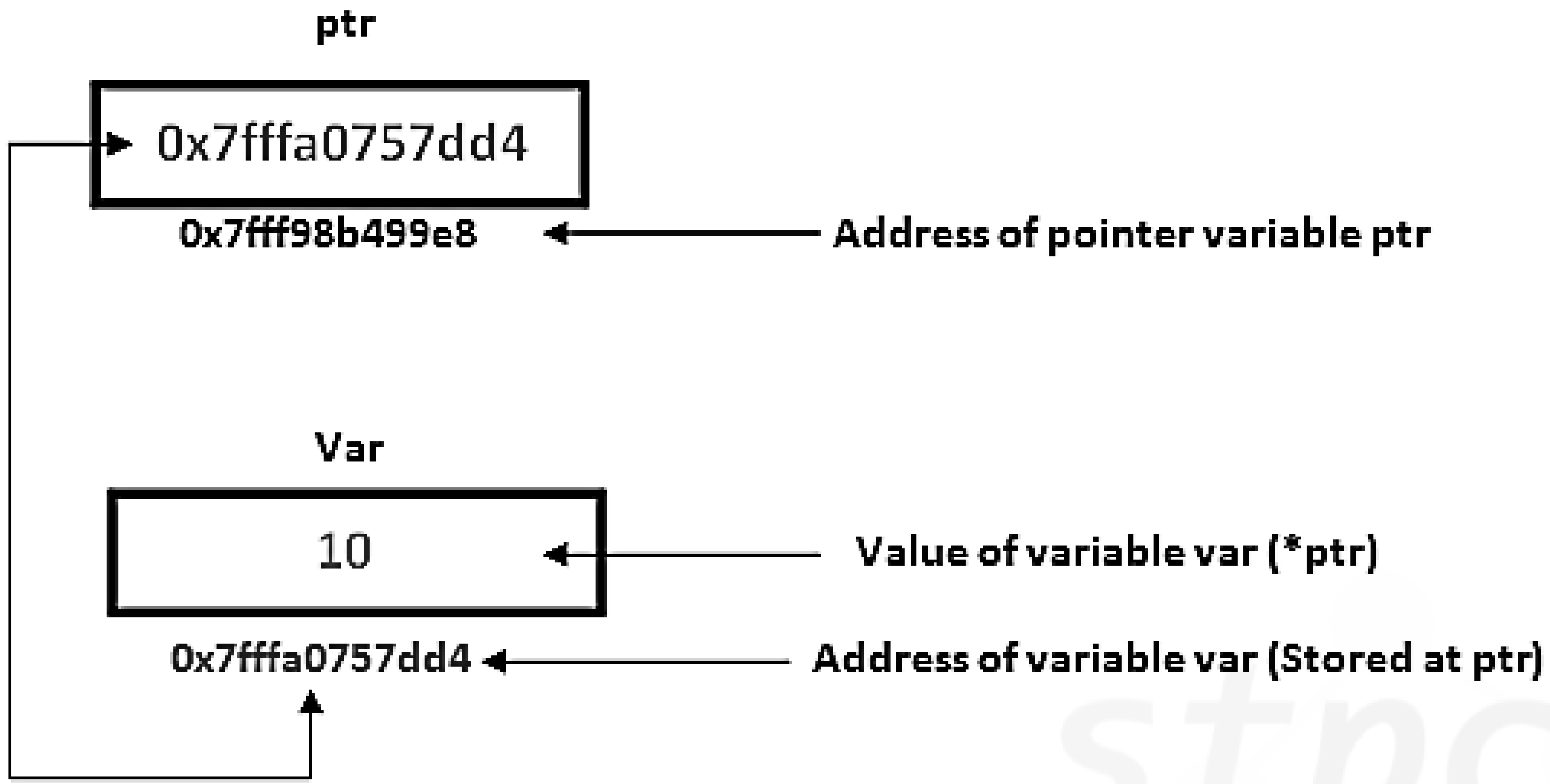
```
BSTC
0x7ffe4aa930a0
```

- The second line of output indicates the ***address*** of the variable. The address is represented in the form of a hexadecimal value. It might **NOT BE FIXED** during each execution of the code segment.

C++ Pointers

- Pointers are symbolic representations of addresses.
- A pointer acts like a normal variable. However, the content of them is different.
- A normal variable stored some value (mutable / immutable).
- A pointer variable stored an address (of something).

C++ Pointers


ptr
0x7fffa0757dd4
0x7fff98b499e8
Address of pointer variable ptr
Var
10
Value of variable var (*ptr)
0x7fffa0757dd4
Address of variable var (Stored at ptr)

C++ Pointers

- Try to execute the following code segment.

```
string school = "BSTC";
string* ptr = &school;

cout << ptr << endl;
```

- Pay heed to the second line of code for the declaration of a pointer.
- Remind that a pointer stores an address. A pointer is **NOT** a reference variable. The output of the code segment will be an address.

C++ Pointers

- How we can retrieve the value stored in a specific address?
- Use a ***dereference operator*** *.
- Try to execute the following code segment.

```
string school = "BSTC";
string* ptr = &school;

cout << ptr << endl;
cout << *ptr << endl;
```

```
0x7ffe88291070
BSTC
```

C++ Pointers

- Mini-exercise:
  What is the output of the following code segment?

```
BSTC
0x7ffd4f56fd60
BSTC
Computer Room
Computer Room
```

```
string location = "BSTC";
string* ptr = &location;

cout << location << endl;
cout << &location << endl;
cout << *ptr << endl;

*ptr = "Computer Room";

cout << *ptr << endl;
cout << location << endl;
```

Content

- Pointer
- **C++ Structures**
- Linked List
- Monotonic Data Structure

C++ Structures

- Structures (also called ***structs***) are a way to group several related variables into one place. Each variable in the structure is known as a ***member*** of the structure.
- Unlike an array, a structure can contain many different data types.

C++ Structures

- A struct is declared as follows.

```
struct {
    int num;
    string s;
} custom_struct;
```

- The struct above contains two members, an integer and a string.
- The name of the struct is `custom_struct`.

C++ Structures

- We can assign custom values to members of the structure.

```
struct {
    int num;
    string s;
} custom_struct;

custom_struct.num = 5;
custom_struct.s = "BSTC";

cout << custom_struct.num + 2 << endl;
cout << custom_struct.s[2] << endl;
```

```
7
T
```

C++ Structures

- Furthermore, the structure can be named.
- Therefore, the structure can be treated as a unique data type.

```
struct myStruct{
    int num;
    string s;
};

myStruct custom_struct;
myStruct this_is_a_struct_array[50];
```

Pointers and Structure

- The following code segment is executed.

```
struct myStruct {
    int num;
} newStruct;

newStruct.num = 5;

myStruct* ptr = &newStruct;
cout << *ptr.num << endl;
```

```
request for member ‘num’ in ‘ptr’,
which is of pointer type
‘main()::myStruct*’ (maybe you meant
to use ‘->’ ?)
```

- What is the output of the program?

Pointers and Structure

- To access the member of a structure which is pointed by a pointer, we will use a special syntax, shown as follows:

```
struct myStruct {
    int num;
} newStruct;

newStruct.num = 5;

myStruct* ptr = &newStruct;
cout << ptr->num << endl;
```

C++ Unions

- Union is a special structure in C++.
- The class specifier for a union declaration is similar to class or struct declaration.

```
union {
    int num;
    char c;
};
```

- The syntax of `union` is basically the same as `struct`.

C++ Unions

- However, one thing that makes it different from structures is that the member variables in a union share the **same memory location**, unlike a structure that allocates memory separately for each member variable.
- The size of the union is equal to the size of the largest data type.
- Memory space can be used by **one member variable at one point** in time, which means if we assign value to one member variable, it will automatically **deallocate the other member variable** stored in the memory which will lead to loss of data.

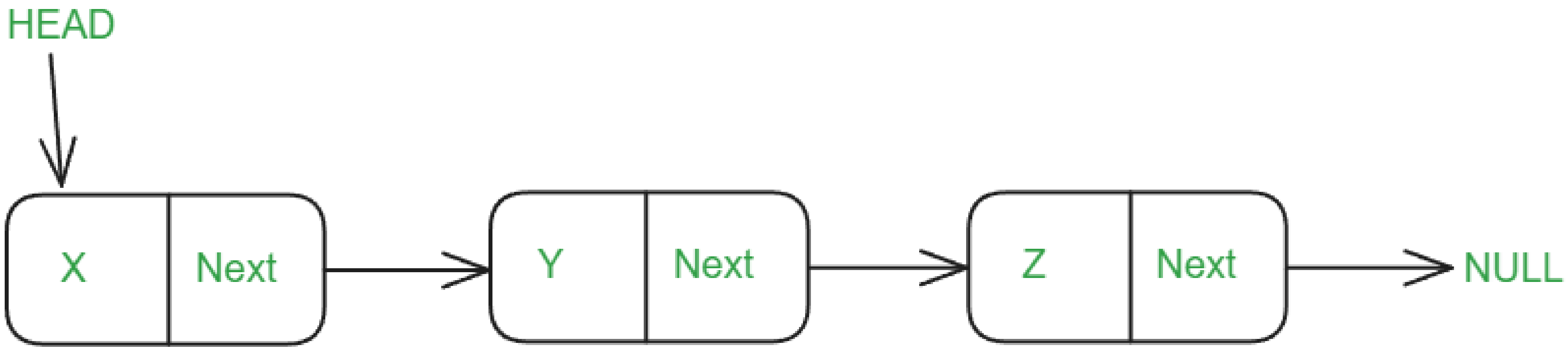
Content

- Pointer
- C++ Structures
- **Linked List**
- Monotonic Data Structure

Linked List

- A linked list is a linear data structure that allows us to store data in ***non-contiguous*** memory locations.
- A linked list is defined as a collection of nodes where each node consists of two members which represents its ***value*** and a next pointer which stores the ***address for the next node***.

Linked List



SINGLY LINKED LIST

Linked List

- When a node contains a pointer storing the address of the next node, the list is called a ***singly linked list***.
- When a node contains two pointers storing the address of the previous node and the next node respectively, the list is called a ***doubly linked list***.
- When the list is found to be circular, the list is called a ***circular linked list***.

Linked List

- Why linked list?
- Advantage: High efficiency to insert / delete a node.
- Disadvantage: Sequential access is needed.

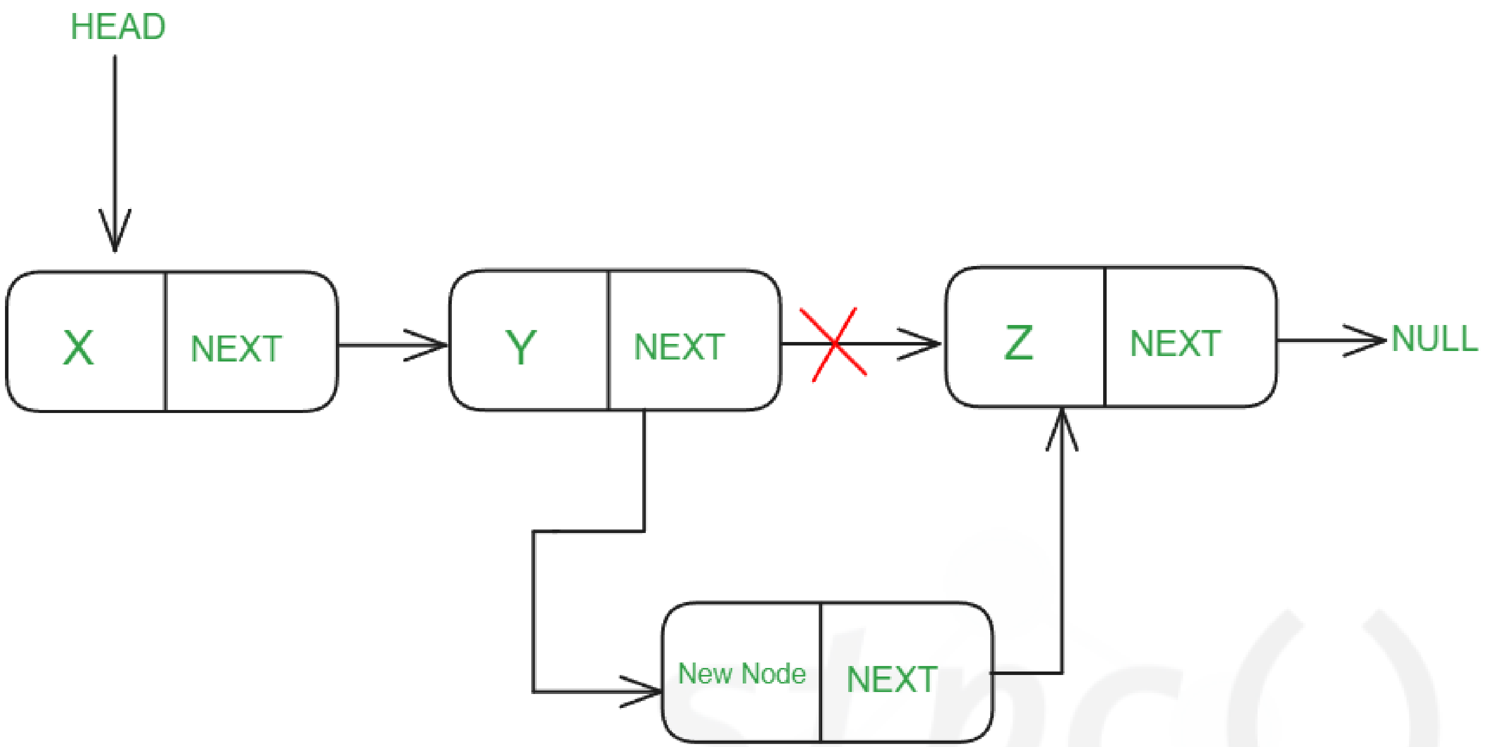
Linked List

- There are two major method to implement a linked list.
  - By linear array
  - By structures
- We will introduce some basic operation on the linked list. The actual implementation is left as exercise. ☺
- The following operation is based on a singly linked list.

Linked list: Insert at position

```
newNode.next ← Z
Y.next ← newNode
```

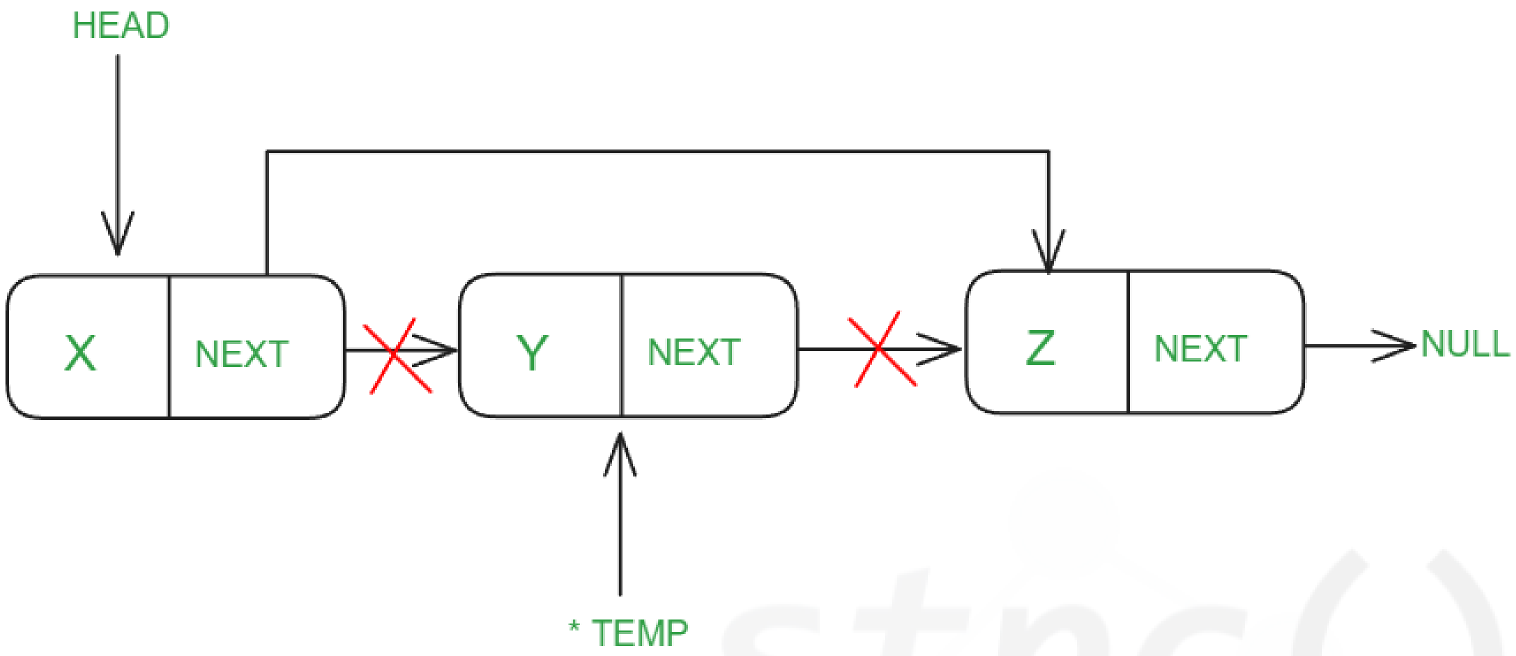
Time complexity: O(1)



Linked list: Delete at position

`X.next ← X.next.next`

Time complexity: O(1)



Linked list: Sequential access

```
curr ← HEAD
while curr <> NULL
  Output curr
  curr ← curr.next
```

Time complexity: O(N)

[N is the number of node inside the linked list]

CSP-J 2023 Q4

- A node in the linked list is defined as follow:

```
struct Node {
    int data;
    Node* next;
}
```

- Currently, there is a pointer pointing to the head of the linked list:

```
Node* head;
```

- If we want to insert a new node with the value stored in `data` is 42, and the new node will become the first node in the linked list. Which of the following operation is correct?

CSP-J 2023 Q4

```
A.    Node* newNode = new Node; newNode->data = 42; newNode->next = head; head = newNode;
B.    Node* newNode = new Node; head->data = 42; newNode->next = head; head = newNode;
C.    Node* newNode = new Node; newNode->data = 42; head->next = newNode;
D.    Node* newNode = new Node; newNode->data = 42; newNode->next = head;
```

CSP-J 2023 Q4

- As newNode has to be the first node in the linked list, the next node of newNode must be the node pointed by head.
- After that, update the head pointer to newNode will be correct.

- Answer: **A**

Linked List

- The Josephus Problem
- Notice the high demand of deletion at position and the low demand of retrieval.
- Therefore, we can implement it by a circular singly linked list.
- Furthermore, this question can also be implemented by a circular queue.

Linked List

- There is a special way to store the pointer towards the next node without storing its actual address.
- You can search **XOR linked list** by yourself for more information!

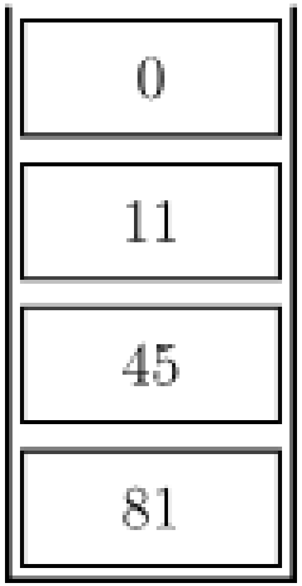
Content

- Pointer
- C++ Structures
- Linked List
- **Monotonic Data Structure**

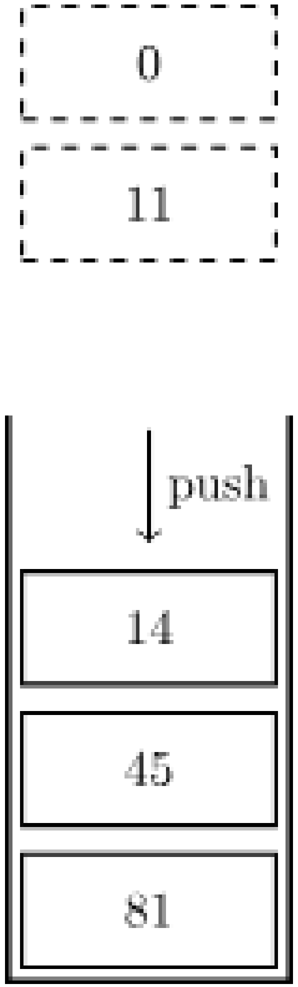
Monotonic stack

- Consider a stack supporting push and pop operation.
- A monotonic stack maintains the monotonicity of the elements stored inside the stack.
- For example, pop the element underneath it before pushing it inside, if the top element is smaller than the element which is ready to be pushed.

Monotonic stack



Monotonic stack



Monotonic stack

- With the maintenance of the monotonicity, some operation can be easily done.
- Given a list of integers a_1 to a_n. For each integer a_i, find the **closest** integer a_j such that $j < i$ and $a_j < a_i$.
- Brute force: **O(N²)**
- Implement a monotonic increasing stack: **O(N)**

- A stack supports *min query* is called a ***min stack***.

Monotonic stack

- Example: {4, 8, 5, 9, 2}

Monotonic stack

- Function
- Analyse the given function carefully.
- Actually, the function requires us to find the index j of the first element which is at the right of a[i] and bigger than a[i]!
- Sounds familiar?

Monotonic queue

- We can also maintain the monotonicity in a queue.
- Notice the “pop” operation when maintaining the monotonicity of a stack.
- We need to use a special data structure ***deque*** to maintain a monotonic queue.
- Deque supports pushing and popping from two ends.

Monotonic queue

- Sliding Window Maximum
- Given a distinct integer array A, there is a sliding window of size k that slides from the beginning to the end of the array. Find the maximum element in the sliding window for every window in A.
- Queue: **O(k)** for each window
- Heap: **O(log k)** for each window
- Monotonic queue: **O(1)** for each window

Monotonic queue

- Example: A = {7, 3, 1, 5, 0, 4, -3, -2, -1}, k = 3

Monotonic queue

- [Sliding Window]
- Implement the version of sliding window minimum by yourself!

Monotonic structure

- Unfortunately, we seldom use monotonic structure directly to implement a problem.
- However, the technique of using monotonic structure is frequently used to optimize dynamic programming transitions!
  - Wait for Dynamic Programming (IV)!

Q&A