**C++, Java Conversion Guide - Yeongu Choe**

**Data Type**

|  |  |  |  |
| --- | --- | --- | --- |
| C++ | | Java | |
| Type | Range | Type | Range |
| nullptr |  | null |  |
| void |  | void |  |
| byte | [0,255] | byte | [Byte.MIN\_VALUE,Byte.MAX\_VALUE] |
| short | [SHRT\_MIN,SHRT\_MAX] | short | [Short.MIN\_VALUE,Short.MAX\_VALUE] |
| int | [INT\_MIN,INT\_MAX] | int | [Integer.MIN\_VALUE,Integer.MAX\_VALUE] |
| long | [LONG\_MIN,LONG\_MAX] | long | [Long.MIN\_VALUE,Long.MAX\_VALUE] |
| float |  | float | [Float.MIN\_VALUE,Float.MAX\_VALUE] |
| double |  | double | [Double.MIN\_VALUE,Double.MAX\_VALUE] |
| char | [CHAR\_MIN,CHAR\_MAX] | char |  |
| string |  | String |  |
| bool | true,false | boolean | true,false |

**Data Structure Declaration**

|  |  |  |
| --- | --- | --- |
| Data Structure | C++ | Java |
| Array | int x[5] = {1,2,3,4,5};  int x[] = {1,2,3,4,5};  int x[5]{1,2,3,4,5};  int x[]{1,2,3,4,5}; | int[] x = {1,2,3};  int[] x = new int[3];  int x[] = {1,2,3};  int x[] = new int[3];  int[] x = new int[]{1,2,3};  int x[] = new int[]{1,2,3}; |
| 2D Array | int x[2][2]={{1,0},{0,1}};  int x[2][2]{{1,0},{0,1}};  int x[][2]={{1,0},{0,1}};  int x[][2]{{1,0},{0,1}}; | int[][] x = {{1,0},{0,1}};  int[][] x = new int[2][2];  int[][] x = new int[2][];  int x[][] = {{1,0},{0,1}};  int x[][] = new int[2][2];  int x[][] = new int[2][]; |
| List | vector<int> x = {1,2,3,4,5};  vector<int> x{1,2,3,4,5};  vector<int> x(5);  vector<int> x(5,1); | List<Integer> x = new ArrayList<>(); |
| Stack | stack<int> x; | Stack<Integer> x = new Stack<>(); |
| Queue | queue<int> x;  queue<int> x({1,2,3,4,5}); | Queue<Integer> x = new LinkedList<>(); |
| Map | map<int,int> x = {{1,2}};  map<int,int> x{{1,2}};  map<int,int> x({{1,2}}); | Map<Integer,Integer> x = new HashMap<>(); |
| Set | set<int> x = {1,2,3,4,5};  set<int> x{1,2,3,4,5};  set<int> x({1,2,3,4,5}); | Set<Integer> x = new HashSet<>(); |
| MaxHeap | priority\_queue<int> x; | PriorityQueue<Integer> x = new PriorityQueue<>((a,b)->{return b-a;}); |
| MinHeap | priority\_queue<int,vector<int>,greater<int>> x; | PriorityQueue<Integer> x = new PriorityQueue<>(); |

**Data Structure Operation**

Array

|  |  |  |
| --- | --- | --- |
| C++ | Java | Time Complexity |
| sizeof(x)/sizeof(x[0]); | x.length | O(1) |
| sort(x,x+sizeof(x)/sizeof(x[0]); | Arrays.sort(x) | O(n log n) |
| sort(x,x+sizeof(x)/sizeof(x[0]),greater<int>()); | Arrays.sort(x,Collections.reverseOrder()); | O(n log n) |
| equal(begin(x),end(x),begin(y)); | Arrays.equals(x,y) | O(n) |
| memcpy(y,x,sizeof(x)); | Arrays.copyOfRange(x,1,3); | O(3-1) |
| fill(begin(x),end(x),100); | Arrays.fill(x,1); | O(n) |
| fill(x,x+1,100); | Arrays.fill(x,0,1,100); | O(1-0) |
| for\_each(begin(x),end(x),[](int& x){x\*=2;}); | Arrays.setAll(x,i->x[i]\*2); | O(n) |
| string y(x); | Arrays.toString(x); | O(n) |
| \*find(x,x+sizeof(x)/sizeof(x[0]),100); |  | O(n) |

List

|  |  |  |
| --- | --- | --- |
| C++ | Java | Time Complexity |
| x.push\_back(1); | x.add(1); | O(1) |
| x.erase(x.begin()+1); | x.remove(1); | O(n) |
| x[0]=1;  x.at(0)=1; | x.set(0,1); | O(1) |
| x[0];  x.at(0); | x.get(0); | O(1) |
| x.size(); | x.size(); | O(1) |
| x.empty(); | x.isEmpty(); | O(1) |
| find(x.begin(),x.end(),1); | x.contains(1); | O(n) |
| x.clear(); | x.clear(); | O(n) |
| sort(x.begin(),x.end()); | Collections.sort(x); | O(n log n) |
| sort(x.rbegin(),x.rend()); | Collections.sort(x,Collections.reverseOrder()); | O(n log n) |
| vector<int> y(x.begin(),x.begin()+2); | x.subList(0,2); | O(2-0) |

Stack

|  |  |  |
| --- | --- | --- |
| C++ | Java | Time Complexity |
| x.push(1); | x.push(1); | O(1) |
| x.pop(); | x.pop(); | O(1) |
| x.top(); | x.peek(); | O(1) |
| x.empty(); | x.empty(); | O(1) |
| x.size(); | x.size(); | O(1) |

Queue

|  |  |  |
| --- | --- | --- |
| C++ | Java | Time Complexity |
| x.push(1); | x.add(1); | O(1) |
| x.pop(); | x.remove(); | O(1) |
| x.front(); | x.element(); | O(1) |
| x.size(); | x.size(); | O(1) |

Map

|  |  |  |
| --- | --- | --- |
| C++ | Java | Time Complexity |
| x[1]=1;  x.insert({1,1}); | x.put(1,1); | O(n) |
| x[1];  x.at(1); | x.get(1); | O(n) |
| x.erase(1); | x.remove(1); | O(n) |
| x.clear(); | x.clear(); | O(n) |
| x.size(); | x.size(); | O(1) |
| x.empty(); | x.isEmpty(); | O(1) |

Set

|  |  |  |
| --- | --- | --- |
| C++ | Java | Time Complexity |
| x.insert(1); | x.add(1); | O(n) |
| x.erase(1); | x.remove(1); | O(n) |
| x.find(1)!=x.end(); | x.contains(1); | O(n) |
| x.empty(); | x.isEmpty(); | O(1) |
| x.clear(); | x.clear(); | O(n) |
| copy(x.begin(),x.end(),y); | Integer[] y = x.toArray(new Integer[]{}); | O(n) |

Heap

|  |  |  |
| --- | --- | --- |
| C++ | Java | Time Complexity |
| x.push(1); | x.add(1); | O(log n) |
| x.pop(); | x.remove(); | O(log n) |
| x.top(); | x.peek(); | O(1) |
| x.size(); | x.size(); | O(1) |
| x.empty(); | x.isEmpty(); | O(1) |