Array Doubling

* Arrays are fixed size.
* Array doubling algorithm dynamically creates a new array that is twice the size of the original array.
* Copy over the values from old to new.
* Free the memory assigned to the old array

Int \*p

Int size=2

P=new int[size]

P[0]=1

P[1]=2

Array is full, double

Size=size\*2

Int \*p2=new int[size]

P2[0]=p[0]

P2[1]=p[1]

Delete[]p

Write more data to p2

P2[3]=3

P2[4]=4

P=p2 (means p and p2 are pointing to the same thing)

Size=size\*2

P2=new int[size]

//now p2 and p are not pointing to the same memory.

P2[0]=p[0]

P2[3]=p[3]

Delete[ ]p //we can delete p because we have its data

P=p2

(now they both point to the same thing again)

If we tried to delete p2, anything that points to the same place p2 is, will also be affected. So p will be deleted too.

Delete[ ]p2

Delete[ ]p

Main()

A= new int[n]

B = doubleArray(A)

Delete[ ] A

Arrays

Limitations = array doubling is expensive

Ne wApproach

1. Allocate memory for individual elements as needed.
2. Link them together with pointers

This structure is called a linked list.

Data -🡪 Data 🡪 Data 🡪 Null.

End of list points to null.

Each element in a linked list is called a Node.

Simplest Node defined with a struct

Struct node

{

Int key; //any data

Node \*next //pointer to another instance that is the same type as itself. To another node.

Node(int k, node \*n) //Constructor

{

Key=k;

next=n;

}

}

Singly-linked list – pointers only go in one direction.

Doubly-linked list – Node \*next and Node \*previous

Data 🡪 🡨Data 🡪 🡨 Data 🡪 🡨 Data🡪Null

Ex: Create a single linked list with 3 nodes with key values5, 6 ,7.

5 🡪6🡪7🡪Null

5

Node \*x=new node(5, null)



Node \*xx2 = new node(6, null)

X -> next = x2 //x-> next stores address of x2

Node x3= new node(7, null)

X2 -> next=x3