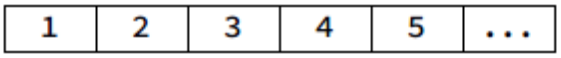
Unrolled linked list

> There are some instruction of unrolled linked list on Wikipedia.

Description

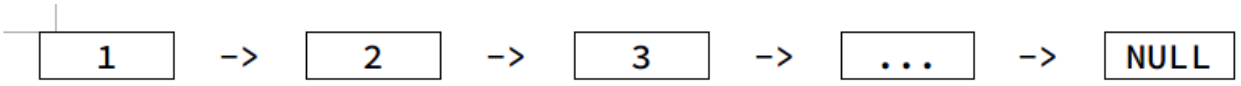
Usually, there are two ways for us to implement a link. The first one is array, which have a good cache preformance because the memory is adjacent. However, it will lead to a bad performance if we try to insert something in the middle of the array.

This is an example for an array.



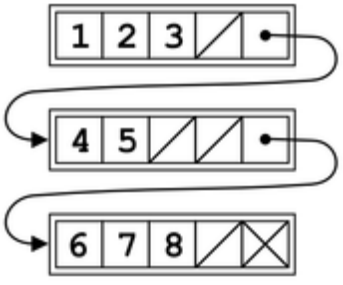
The second method is single linked list, it only need a few pointer manipulation when we insert some nodes in the middle of the list. But the cache performance is not good because the linked list spread data elements all over the memory of heap.

This is an example for an (single) linked list.



Our mission today is to implement a data structure called "Unrolled Linked List", which can combine the advantages of the array and the linked list.

This is an example of unrolled linked list.



For a unrolled linked list, the data in one **block** is adjacent, so the data can fill a cache line and get better performance. And the cost of insertion is acceptable.

```
struct Node {
    Node* next;
    int size;
    char values[NODE_SIZE];
};
```

```
};

class UnrolledLinkedList {
public:
    int get(size_t index);
    void insert(size_t index);
private:
    Node* head;
};
```

There is a skeleton of our unrolled linked list class. You have to implement the function `get` and `insert`.

- the "get" function
  You should return the k_{th} element in the list.
  As that example above, `get(0) == 1` and `get(3) == 4`.
- the "insert" function
  You should insert an element right after the k_{th} element.
  After `insert(4, 19)`, the list may look like this.

```
[---] -> [---] -> [---] -> NULL
[1]      [4]      [6]
[2]      [5]      [7]
[3]      [*19*]   [8]
[-]      [-]      [-]
[-]      [-]      [-]
```

Hint

There are so many corner cases and trival details when you try to implement this list.

It's not a hard one, but you need to keep sharp and try hard to avoid any possible mistakes.