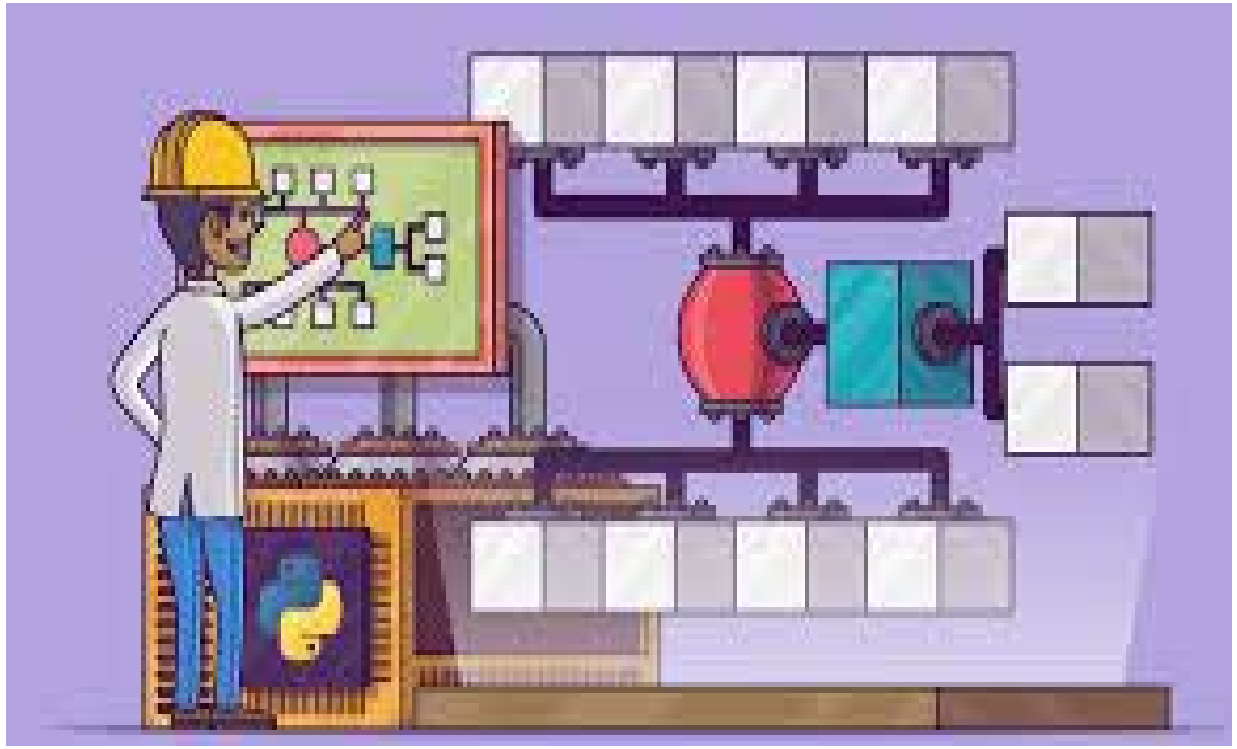
ساختمان داده ها

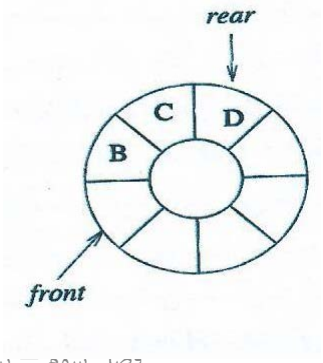


مدرس:
سمانه حسینی سمنانی

دانشگاه صنعتی اصفهان- دانشکده برق و کامپیوتر

Queue implementation using array

```
private:
    T *queue;       // array for queue elements
    int front,      // one counterclockwise from front
        rear,       // array position of rear element
        capacity;   // capacity of queue array

template <class T>
Queue<T>::Queue (int queueeCapacity) : capacity (queueCapacity)
{
    if (capacity < 1) throw "Queue capacity must be > 0";
    queue = new T [capacity];
    front = rear = 0;
}
```



چالش پر یا خالی بودن صف

پر یا خالی بودن صف را چطور تشخیص دهیم؟ برای هر دو حالت front = rear

راه حل برای حل مشکل:

۱- ذخیره تعداد عناصر موجود در صف

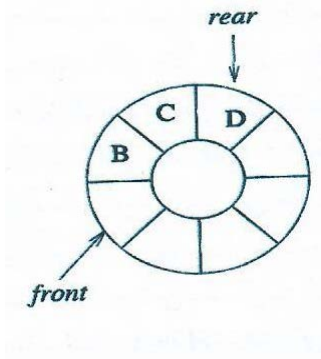
۲- اجازه ندهیم صف پر شود. همیشه یک عنصر صف را خالی نگه داریم

۳- نگه داشتن آخرین عمل

Queue ADT

```
template <class T>
inline bool Queue<T>::IsEmpty() {return front == rear;}

template <class T>
inline T& Queue<T>::Front()
{
        if (IsEmpty ()) throw "Queue is empty. No front element";
        return queue [(front + 1) % capacity];
}

template <class T>
inline T& Queue<T>::Rear()
{
        if (IsEmpty()) throw "Queue is empty. No rear element";
        return queue [rear];
}
```

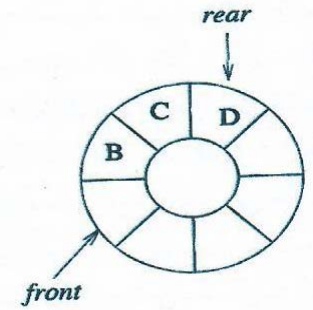
rear == front



Push in the Queue

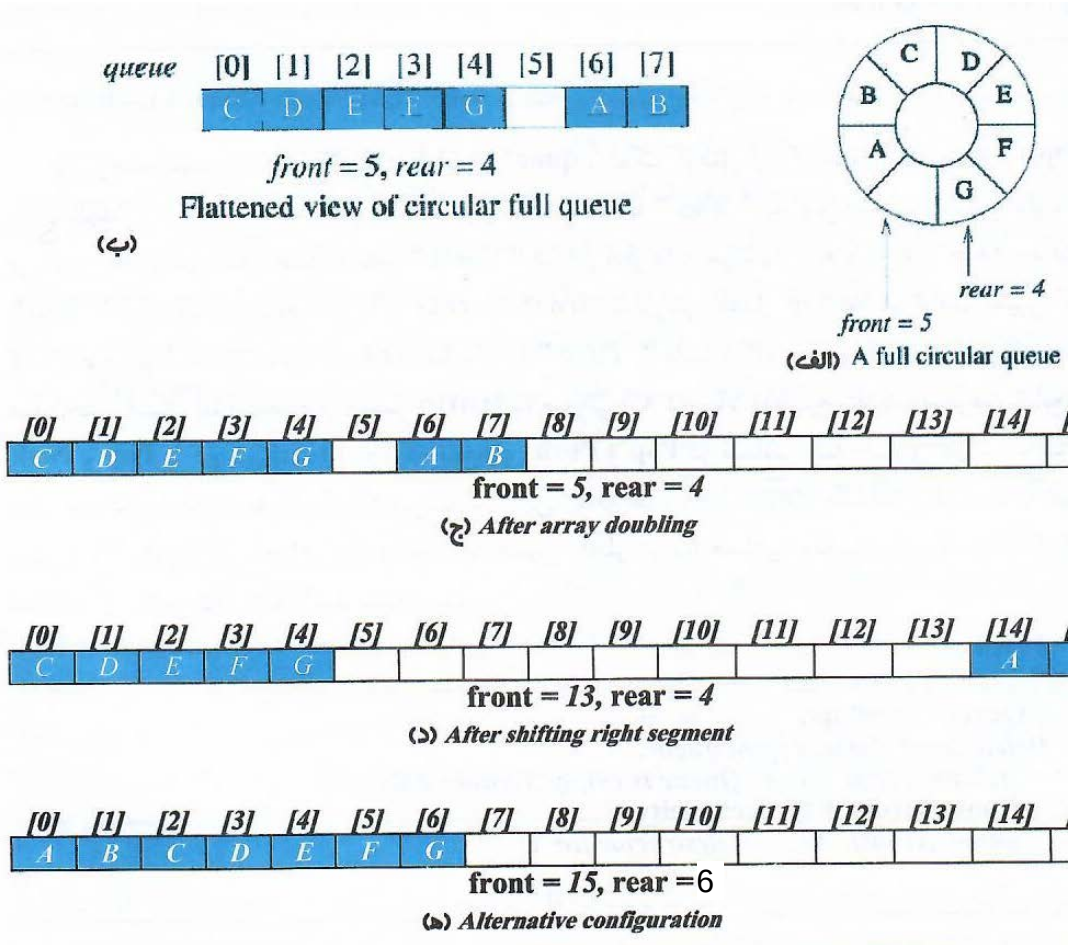
```
emplate <class T>
oid Queue<T>::Push(const& x)
// Add x at rear of queue.
        if ((rear + 1) % capacity == front)
        {// queue full, double capacity
                // code to double queue capacity comes here
         }

         rear = (rear + 1) % capacity;
         queue[rear] = x;
}
```

rear + 1 == front



front = 0
rear = 7
(rear+1) % capacity =
8 % 8 = 0
Front = 0 ->
queue is full

Push in the Queue



1. آرایه جدید را با دو برابر ظرفیت ایجاد کنید.
2. قطعه دوم از آرایه قبلی (q[capacity-1] تا q[front+1]) را به اول آرایه جدید کپی کنید.
3. قطعه اول از آرایه قبلی (q[rear] تا q[0]) را به محل capacity-front-1 آرایه جدید کپی کنید.

Push in the Queue

```
// allocate an array with twice the capacity
T* newQueue = new T[2*capacity];

// copy from queue to newQueue
int start = (front + 1) % capacity;
if (start < 2)
    // no wrap around
    copy (queue + start,queue + start + capacity – 1, newQueue);
else
// queue wraps around
    copy (queue + start,queue + capacity, newQueue);
    copy (queue,queue + rear + 1, newQueue + capacity – start);

// switch to newQueue
front = 2*capacity – 1;
rear = capacity – 2;
capacity *=2;
delete [] queue;
queue = newQueue;
```

1. آرایه جدید را با دو برابر ظرفیت ایجاد کنید.
2. قطعه دوم از آرایه قبلی (q[capacity-1] تا q[front+1]) را به اول آرایه جدید کپی کنید.
3. قطعه اول از آرایه قبلی (q[rear] تا q[0]) را به محل capacity-front-1 آرایه جدید کپی کنید.

Pop from the Queue

- هزینه push و pop از صف: (بدون نظر گرفتن هزینه دو برابر کردن طول صف):$O(1)$

```
template<class T>
void Queue<T>::Pop()
// Delete front element from queue.
    if (IsEmpty()) throw "Queue is empty. Cannot delete.";
    front = (front + 1) % capacity;
    queue [front].~T();    // destructor for T
```

- راه حل برای چالش پر یا خالی بودن صف:

۱- ذخیره تعداد عناصر موجود در صف

۲- اجازه ندهیم صف پر شود. همیشه یک عنصر صف را خالی نگه داریم.

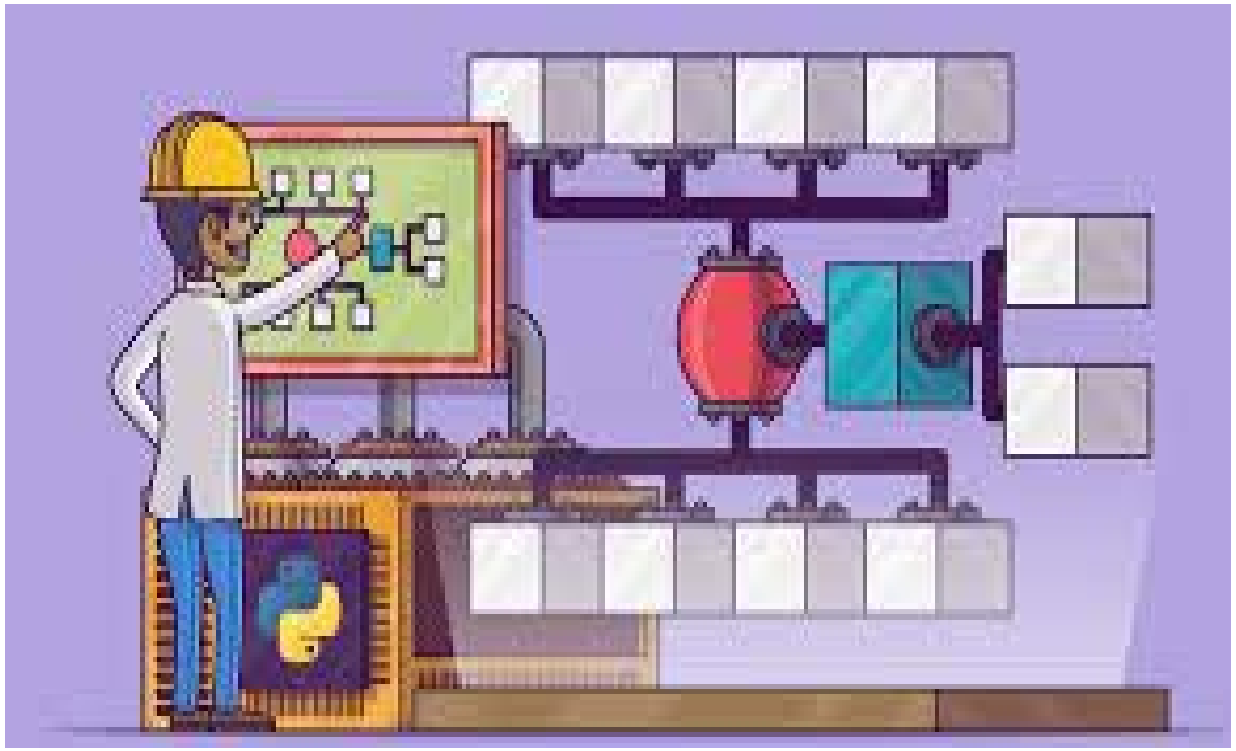
۳- نگه داشتن آخرین عمل

- تعریف متغیر LastOp
- آخرین عمل انجام شده روی صف

$$\begin{cases} full & if \quad front == rear \quad and \quad lastOp == push \\ empty & if \quad front == rear \quad and \quad lastOp == pop \end{cases}$$

کدام راه حل بهتر است و چرا؟

ساختمان داده ها



مدرس:
سمانه حسینی سمنانی

دانشگاه صنعتی اصفهان- دانشکده برق و کامپیوتر

Linked List لیست پیوندی

- لیست های یک پیوندی و زنجیرها
- لیست های حلقوی
- پشته ها و صف های پیوندی
- چند جمله ای ها
- لیستهای دو پیوندی

لیست ها

- لیست مرتب شده: ordered list

Ordered (linear) list: ($item_1$, $item_2$, $item_3$, …, $item_n$)

- مثال
  - (Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday)
  - (1941, 1942, 1943, 1944, 1945)
  - (a_1, a_2, a_3, …, a_{n-1}, a_n)

لیست ها

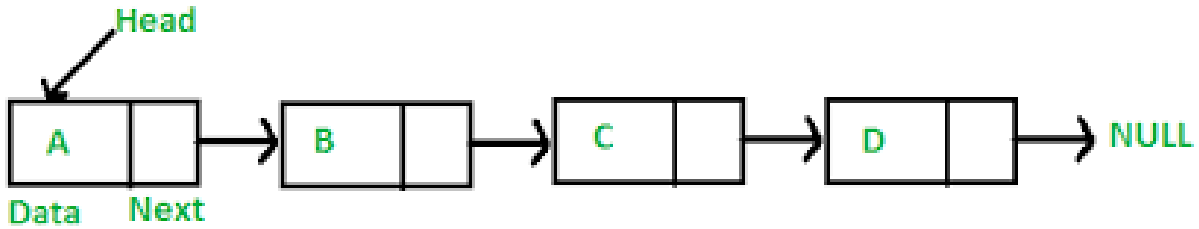
- طراحی
  1. پیدا کردن طول یک لیست
  2. خواندن اقلام داده یک لیست از چپ به راست یا بر عکس
  3. بازیابی i امین عنصر از یک لیست
  4. تعویض یک قلم اطلاعاتی در i امین موقعیت یک لیست
  5. درج یک قلم داده جدید در i امین موقعیت یک لیست
  6. حذف یک قلم اطلاعاتی از i امین موقعیت یک لیست
- پیاده سازی
  - نگاشت ترتیبی sequential mapping (1)~(4) **مثلا با استفاده از آرایه**
  - نگاشت غیر ترتیبی non-sequential mapping (5)~(6) **مثلا با استفاده از لیست پیوندی**

لیست پیوندی - Linked List

- ***linked list*** : data structure in which the objects are arranged in a linear order.
- ***Array***: the linear order is determined by the array indices

However

- ***linked list*** : the order is determined by a pointer in each object.
- Simple and flexible representation for dynamic lists



مقایسه لیست پیوندی و آرایه

- مشکل اصلی آرایه ها هزینه بالای عملیات درج و حذف در آنها است.

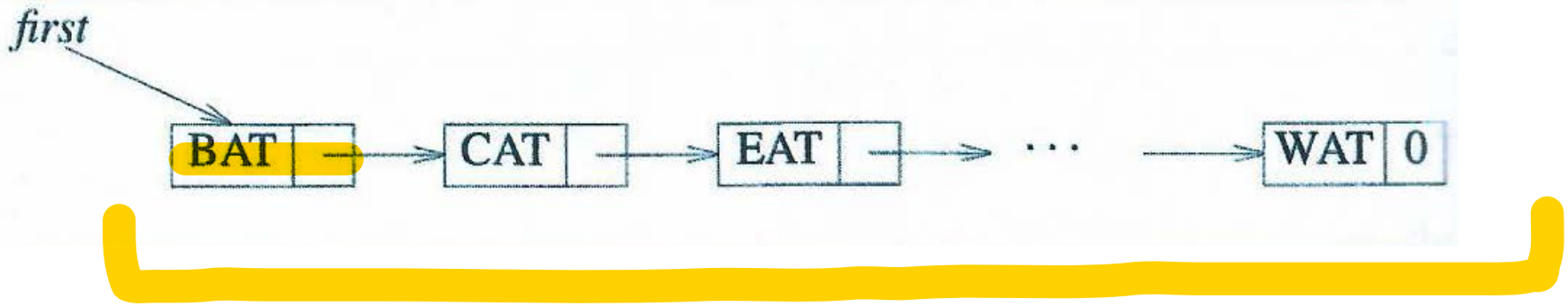
- ***Array***:

- ***linked list***:

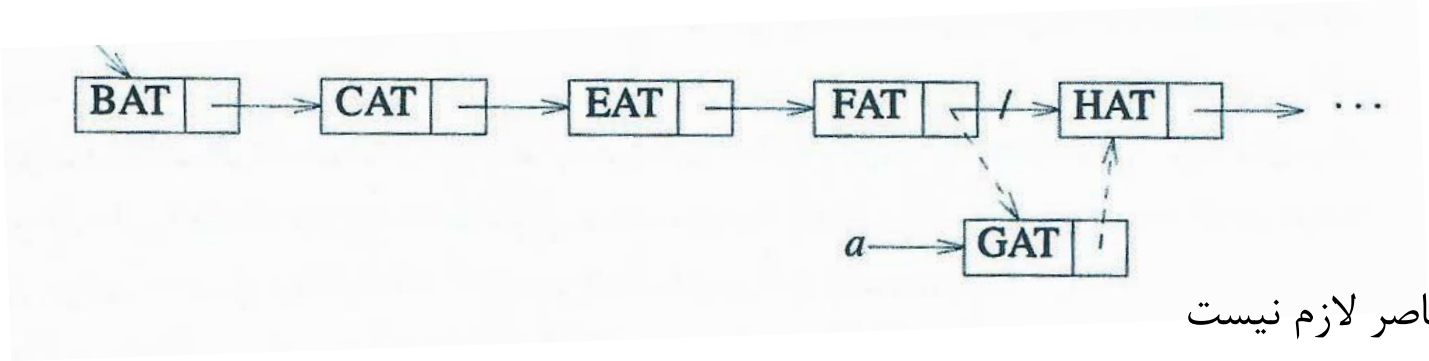
سمانه حسینی سمنانی
هیات علمی دانشکده برق و کامپیوتر- دانشگاه صنعتی اصفهان

مقایسه لیست پیوندی و آرایه

- مکان گره ها در هر بار اجرای برنامه می تواند تغییر کند.



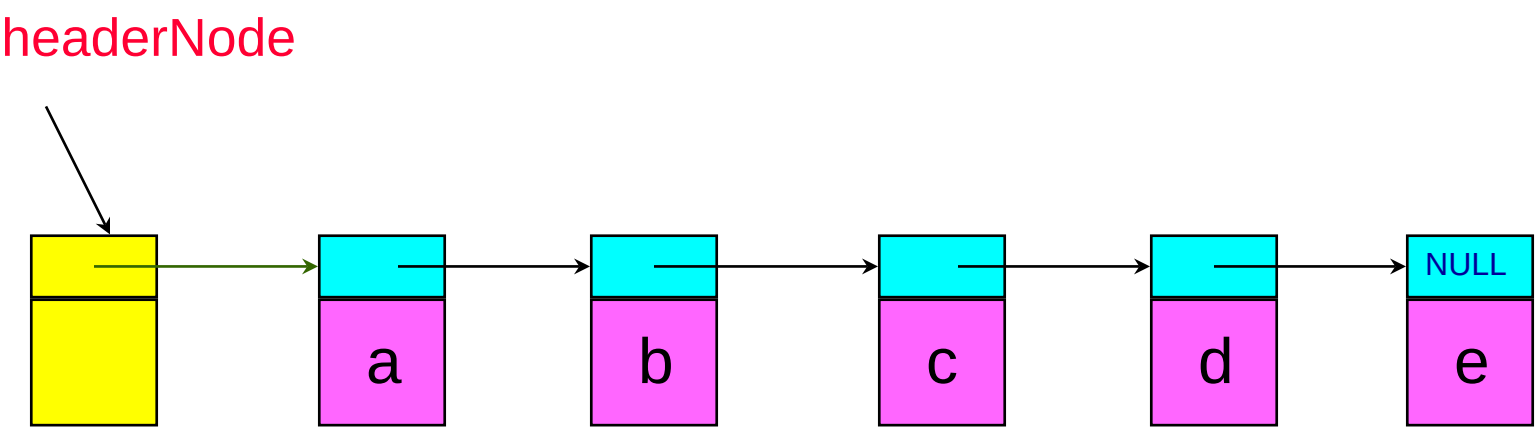
- اضافه کردن GAT به لیست :



- برخلاف آرایه ها جابجایی عناصر لازم نیست

انواع لیست پیوندی

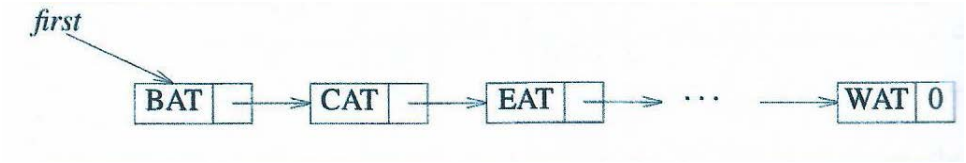
- زنجیر



- لیست حلقوی

تعریف یک زنجیر در C++ - طراحی ۱

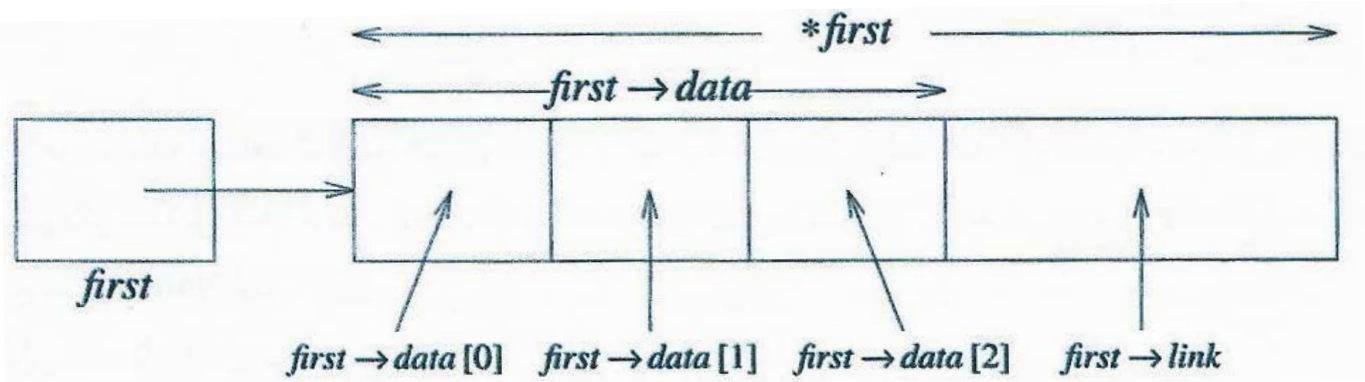
```
class ThreeLetterNode {
private:
   char data[3];
   ThreeLetterNode *link;
};
```




```
TreeLetterNode *first;
```

first →data, first → link

first →data[*0*], first →data[*1*], first→data[*2*]



- مشکل: چون data و link ، private تعریف شده اند، دستورات بالا خطای زمان اجرا دارد.

تعریف یک زنجیر در C++- طراحی ۲

```
class ThreeLetterNode {
private:
   char data[3];
   ThreeLetterNode *link;
};
```

- همان طراحی قبلی با داده های public
- مشکل: نقص اصل پنهان سازی داده ها
- راه حل: تعریف توابع عمومی Getlink, Setlink, Getdata, Setdata

بر روی کلاس ThreeLetterNode

```
TreeLetterNode *first;
```

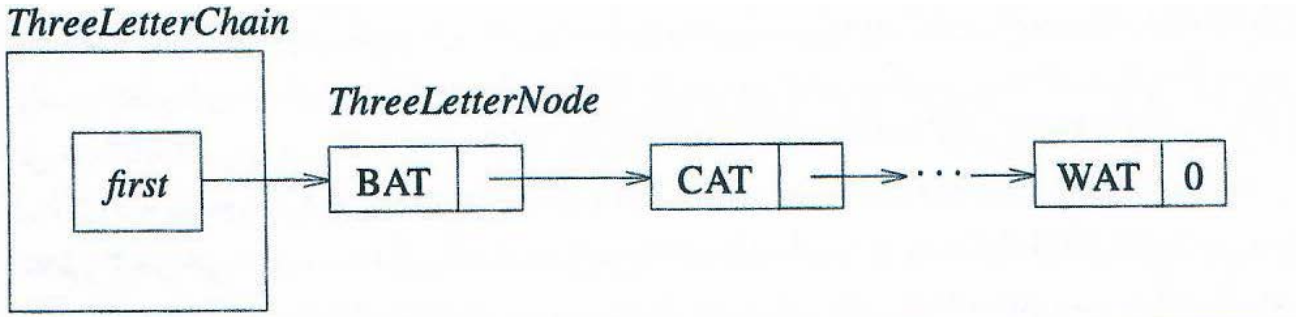
first →data, first → link

first →data[0], first →data[1], first→data[2]

- مشکل: هر تابع داخل برنامه همچنان می تواند به داده های خصوصی از طریق توابع عمومی بالا دسترسی داشته باشد

تعریف یک زنجیر در C++- طراحی ۳

```
calss ThreeLetterChain; // forward declaration

class ThreeLetterNode {
friend class ThreeLetterChain;
private:
    char data[3];
    ThreeLetterNode *link;
};

class ThreeLetterChain {
public:
    // Chain manipulation operations
.
.
private:
    ThreeLetterNode *first;
};
```



- تنها توابع عضو ThreeLetterChain و ThreeLetterNode می توانند به داده های خصوصی ThreeLetterNode دسترسی داشته باشند.
- تعریف توابع پردازش لیست (اضافه کردن یک گره در زنجیر یا حذف گره از زنجیر) به صورت عمومی