ساختمان داده ها به زبان C++

مدرس: دانش
دانشگاه قدسیه ساری

لیست پیوندی

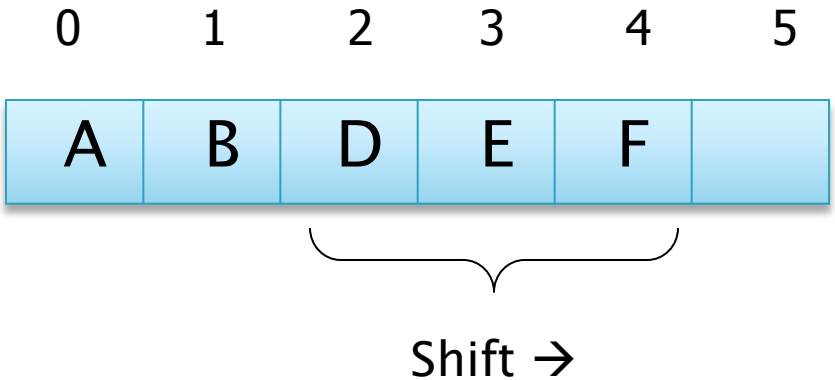
مشکلات ساختمان داده ثابت

- ✓ بازگشت ناپذیر بودن حافظه بعد از گرفتن آن
- ✓ لازم بودن پیش بینی بیشترین حافظه مورد نیاز
- ✓ پر هزینه بودن اضافه کردن عنصر
- ✓ پر هزینه بودن حذف کردن عنصر

مشکلات ساختمان داده ثابت

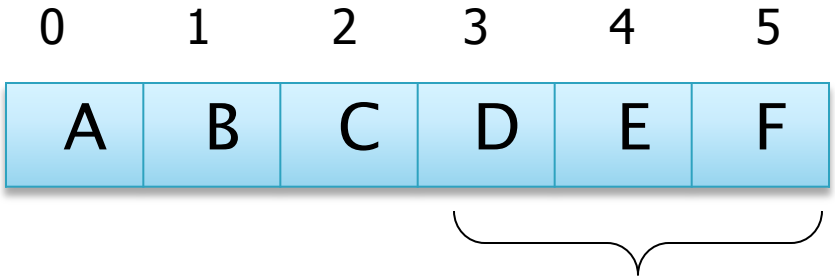
پر هزینه بودن اضافه کردن عنصر

- می خواهیم C را به آرایه زیر اضافه کنیم به طوری که ترتیب الفبایی آن حفظ بماند.



مشکلات ساختمان داده ثابت

پر هزینه بودن اضافه کردن عنصر-ادامه

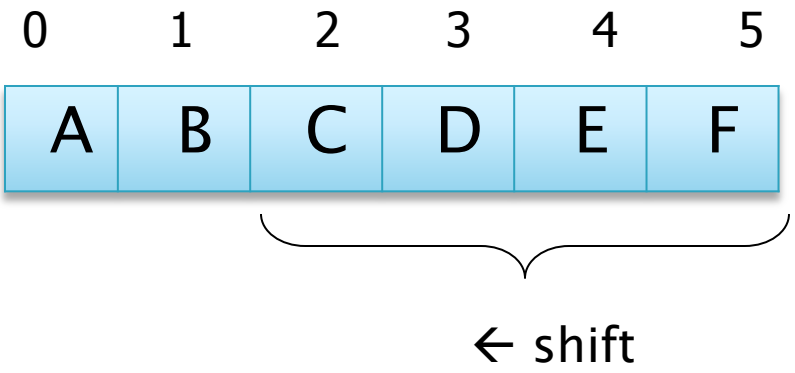


- سوال:مرتبه الگوریتم بالا چیست؟ O(n)

مشکلات ساختمان داده ثابت

پر هزینه بودن حذف کردن عنصر

- اگر بخواهیم B را از آرایه ی قبلی حذف کنیم باید تمام عناصر بعد از آن را یکی به عقب بیاوریم



- مرتبه این الگوریتم نیز O(n) است.

راه حل مشکلات ساختمان داده ثابت

- **استفاده از لیست پیوندی**

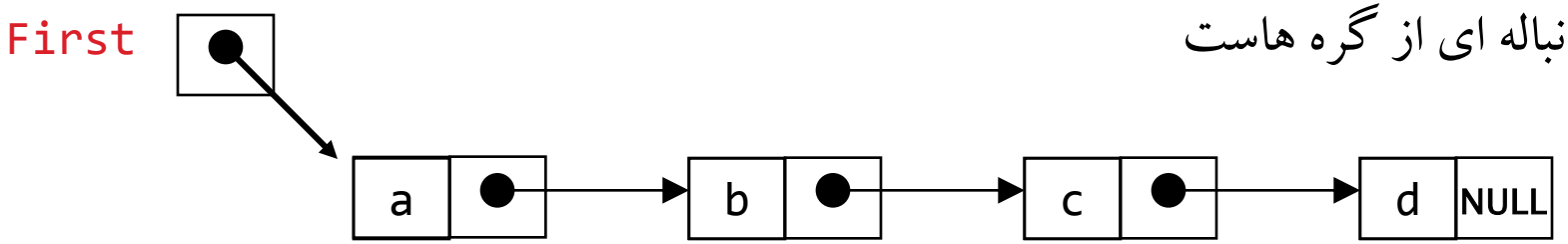
مزایا:

- مجبور نیستیم داده ها را به صورت پشت سر هم در حافظه ذخیره کنیم بلکه در لیست پیوندی عناصر می توانند در هر جای حافظه قرار گیرند. برای این منظور عناصر لیست پیوندی را به یکدیگر پیوند می زنند.

- می توان حافظه ی بدون استفاده را به کامپیوتر برگرداند.

لیست پیوندی (Linked List)

- لیست پیوندی شامل:
  - دنباله ای از گره هاست



- ✓ که هر گره شامل بخش داده (اطلاعاتی که لیست پیوندی نگهداری می کند و می تواند یک یا چند فیلد داده باشد)

  بخش پیوند (اشاره گری که آدرس گره بعدی را نگهداری می کند)

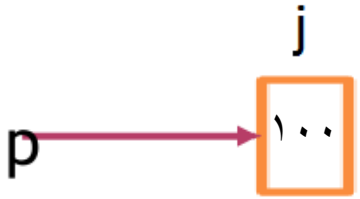
- ✓آخرین گره به NULL اشاره می کند (چون بعد از آن هیچ گره ای نیست).

- ✓در هر لیست پیوندی، آدرس شروع لیست پیوندی را در اشاره گری به نام First باید ذخیره نمود (در غیر این صورت لیست پیوندی در حافظه گم می شود)

پیاده سازی لیست پیوندی با اشاره گر

- **اشاره گر** (pointer): متغیری که آدرس متغیرهای دیگر در حافظه را نگهداری می کند.
- اشاره گر باید از نوع همان متغیر ولی با * باشد. برای مثال اشاره گری که آدرس یک متغیر int را نگهداری می کند باید به صورت int* تعریف شود.

- int *p, j; P اشاره گری به متغیری از نوع int و j متغیری از نوع int است.
- p=&j; این دستور آدرس متغیر j را در p قرار می دهد. یعنی p به j اشاره میکند.



- *p=100; این دستور عدد ۱۰۰ را در خانه ای که p به آن اشاره می کند قرار میدهد.

پیاده سازی لیست پیوندی با اشاره گر

- اشاره گرها زمانی نقش ایفا می کنند که بخواهیم از حافظه پویا استفاده کنیم، یعنی در حین اجرای برنامه در صورت نیاز حافظه ای را تخصیص دهیم و هنگامی که دیگر به این حافظه نیاز نباشد آن را آزاد کنیم و به سیستم برگردانیم.

- دستور **new**: جهت تخصیص حافظه به صورت پویا
- دستور **delete**: جهت آزاد سازی حافظه به صورت پویا

پیاده سازی لیست پیوندی با اشاره گر

- مثال :

```
Int *p;
P=new int;
*p=۵۰;
cout<<*p;
```

- ابتدا اشاره گری از نوع int به نام p تعریف میشود.
- سپس فضای جدیدی به اندازه ۲ بایت (int) در حافظه در نظر گرفته می شود و آدرس آن در p قرار میگیرد.
- سپس محتوای جایی که p به آن اشاره می کند را برابر ۵۰ قرار داده و آن را چاپ میکند.
- جهت حذف فضای تخصیص یافته به صورت زیر عمل میکنیم:
- delete p;
- به این ترتیب این حافظه در زمان اجرا به سیستم برگردانده می شود.

پیاده سازی لیست پیوندی با اشاره گر

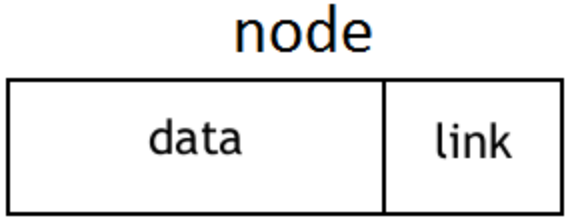
- چنانچه اشاره گرها به متغیری از نوع ساختار (struct) یا کلاس (class) اشاره کنند، جهت دسترسی به هر فیلد آن متغیر از عملگر → استفاده میکنیم:

```
struct temp {
  int i;
  float f;
 };

void  main() {
  temp* p;
  p→ i = 5;
  p→ f = 4.12;
  }
```

تعریف ساختار هر گره در لیست پیوندی

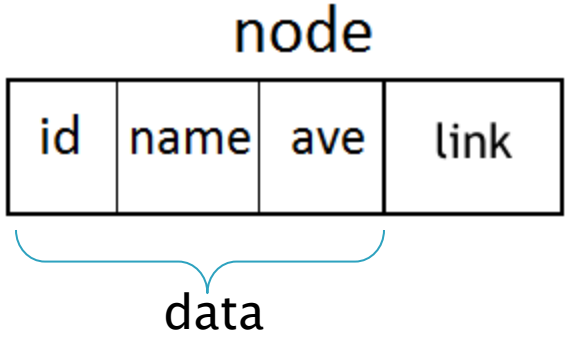
- ساختار هر گره در لیست پیوندی از دو بخش داده و پیوند تشکیل شده است.

```
class node {
        int  data;
        node* link;
}
```



- برای مثال اگر یک لیست پیوندی از دانشجویان داشته باشیم، مشخصات هر گره آن (یک دانشجو) به صورت زیر خواهد بود:

```
class node {
    int id;
    char name[30];
    float  ave;
    node* link;
}
```



تعریف ساختار هر گره در لیست پیوندی

- مثال:یک گره دانشجو با مشخصات زیر تعریف کنید و سپس آن را حذف کنید:

93125	ahmad	14.32	null

- node* stu;
- stu=new node();
- stu→id=93125;
- stu→name=ahmad;
- stu→ave=14.32;
- stu→link=null;
- delete stu;

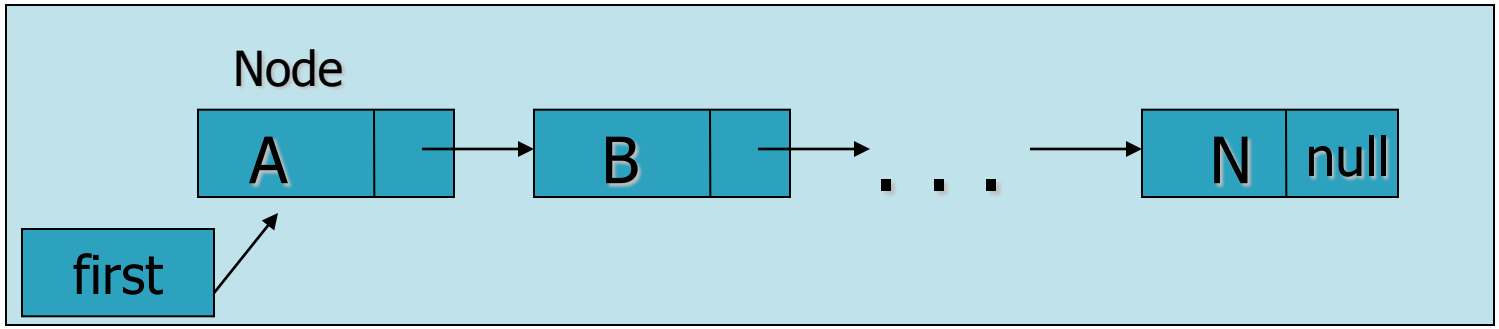
انواع لیست پیوندی

از نظر نوع پیوند چهار نوع مختلف لیست پیوندی داریم:

- لیست پیوندی یک طرفه (تک پیوندی)
- لیست پیوندی یک طرفه حلقوی
- لیست پیوندی دوطرفه (دوپیوندی)
- لیست پیوندی دوطرفه حلقوی (دوپیوندی حلقوی)

لیست پیوندی یک طرفه

لیست تک پیوندی فقط دارای یک اشاره گر (پیوند) است که جهت اشاره به گره بعدی به کار می رود و آخرین گره در این لیست پیوندی به جایی اشاره نمی کند (Null).



پیاده سازی لیست پیوندی یک طرفه

```
class Node
{
        public :
                char  data;
                Node * next;
                Node(char c) { data=c;  next=Null; }
};
```

ساختار هر گره لیست

```
class  LinkedList
{
        public :
                LinkedList ();
                ~ LinkedList ();
                //list manipulation operations
        private:
                Node * first;
};
```

ساختار لیست پیوندی یک طرفه

آدرس اولین گره لیست

عملیات لیست پیوندی یک طرفه

- بررسی خالی بودن لیست
- درج گره جدیدی در لیست
- حذف گره ای از لیست
- جستجوی گره ای در لیست
- پیمایش و نمایش اطلاعات گره های لیست
- حذف همه گره های لیست

پیاده سازی لیست پیوندی یک طرفه

```
class  LinkedList
{
     private:
          Node *  first;
     public :
         LinkedList ();
          bool  IsEmpty ();
          void  display();
          Node*  search (char);
          void  InsertAfter (Node* , char)
          void  InsertBefore (Node* , char);
          Node*  Delete (Node*);
          ~ LinkedList ();
};
```

عملیات لیست پیوندی یک طرفه

```
LinkedList :: LinkedList()
 {
    first = null;
 }
```

سازنده (ایجاد لیست خالی)

```
bool LinkedList :: IsEmpty()
 {
    if (first == null)
         return  true;
    else
         return  false;
 }
```

بررسی خالی بودن لیست

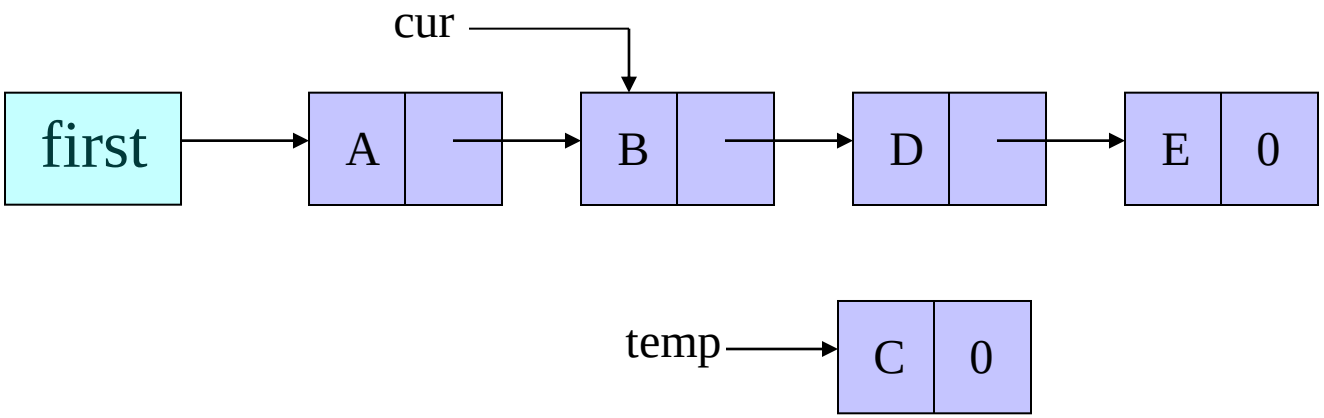
عملیات لیست پیوندی یک طرفه

افزودن گره جدیدی به لیست

1. **تخصیص حافظه برای گره جدید (عملگر new)**
2. **بروزرسانی فیلدهای آدرس جهت درج گره جدید در لیست**
   - درج گره جدید بعد از گره مشخصی در لیست
   - درج گره جدید قبل از گره مشخصی در لیست
     - قبل از گره اول (ابتدای لیست پیوندی)
     - قبل از سایر گره های لیست (جز گره اول)

عملیات لیست پیوندی یک طرفه

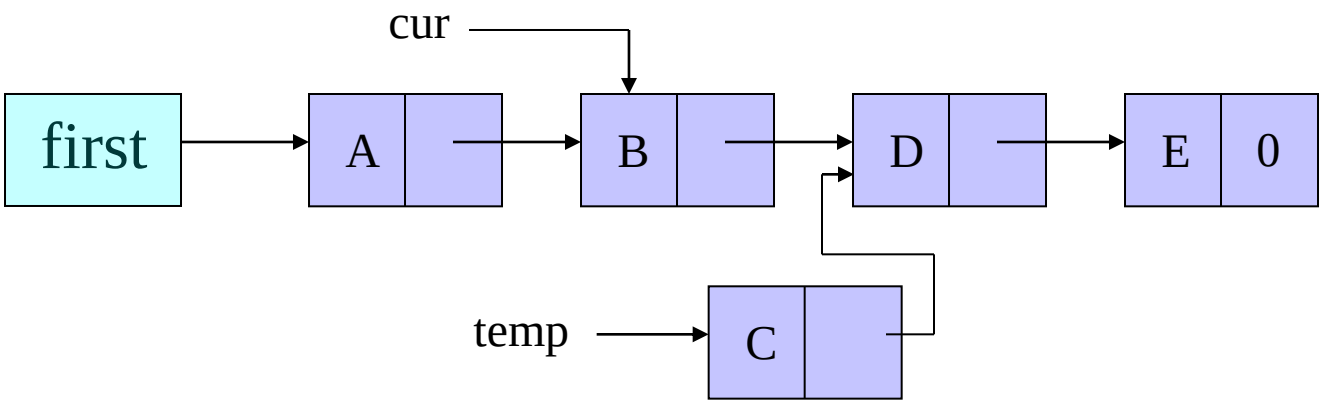
افزودن گره جدیدی بعد از گره مشخصی در لیست



Node *temp = new Node (‘C’);

عملیات لیست پیوندی یک طرفه

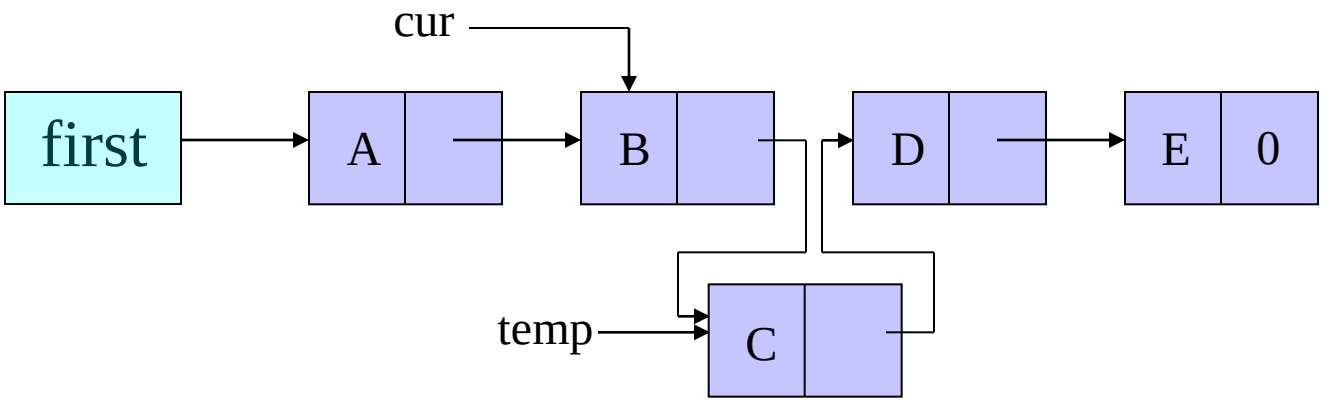
افزودن گره جدیدی بعد از گره مشخصی در لیست



temp → next = cur → next ;

عملیات لیست پیوندی یک طرفه

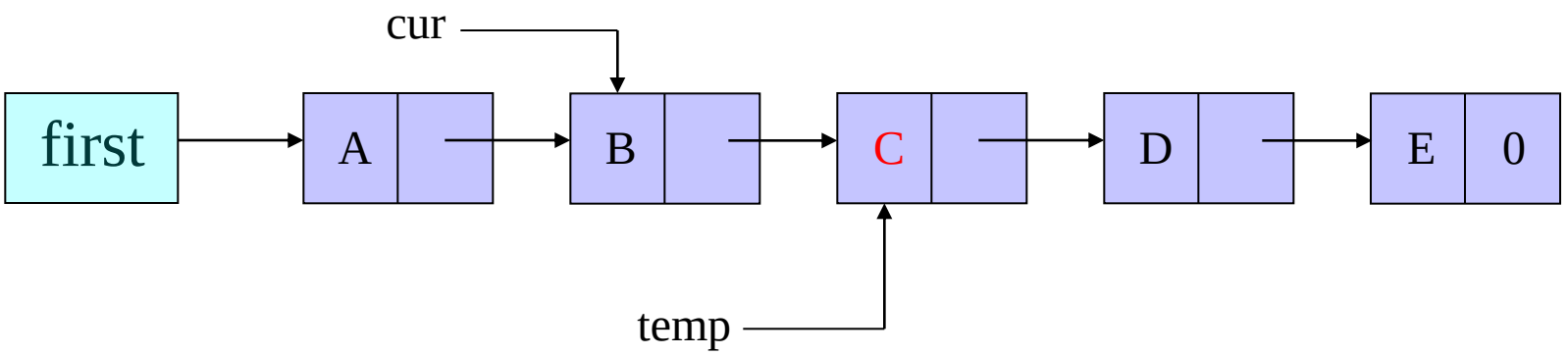
افزودن گره جدیدی بعد از گره مشخصی در لیست



cur → next= temp ;

عملیات لیست پیوندی یک طرفه

افزودن گره جدیدی بعد از گره مشخصی در لیست



عملیات لیست پیوندی یک طرفه

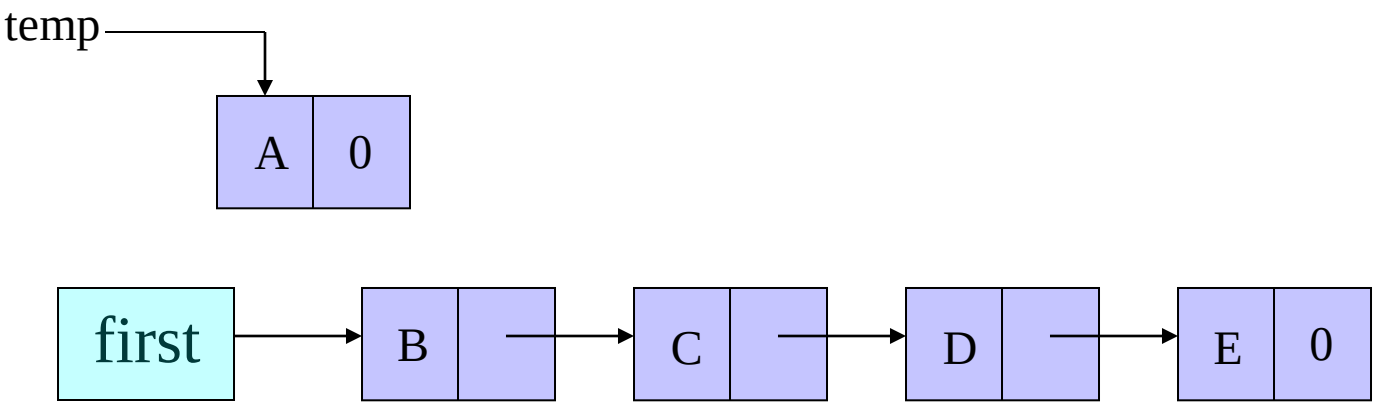
افزودن گره جدیدی بعد از گره مشخصی در لیست

```
void LinkedList :: InsertAfter (Node*cur , char value)
{
    Node  *temp = new  Node(value);
    temp → next = cur → next;
    cur → next= temp ;
}
```

عملیات لیست پیوندی یک طرفه

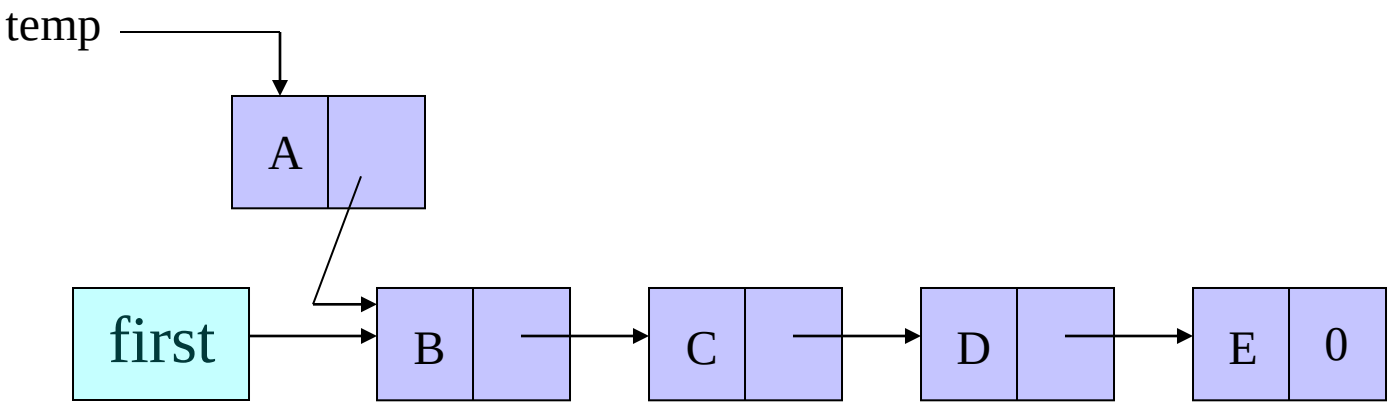
افزودن گره جدیدی در ابتدای لیست

Node *temp = new Node ('A');



عملیات لیست پیوندی یک طرفه

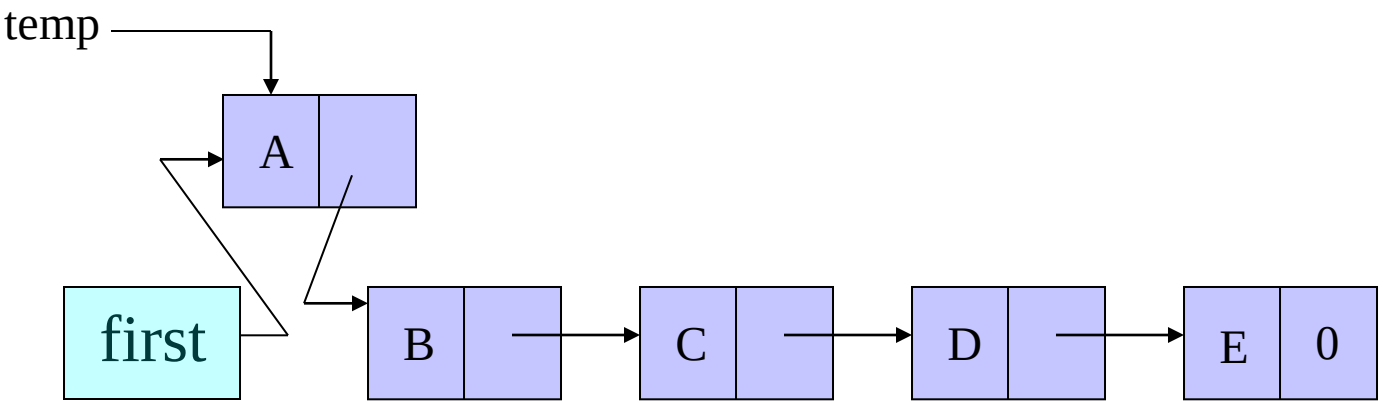
افزودن گره جدیدی در ابتدای لیست



temp → next = first;

عملیات لیست پیوندی یک طرفه

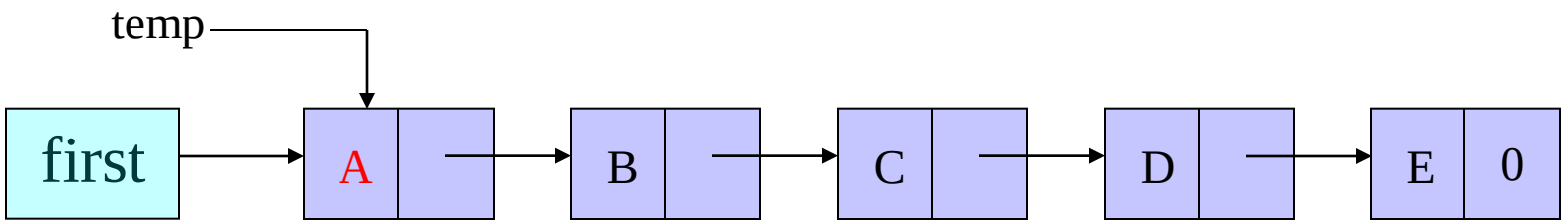
افزودن گره جدیدی در ابتدای لیست



first = temp;

عملیات لیست پیوندی یک طرفه

افزودن گره جدیدی در ابتدای لیست



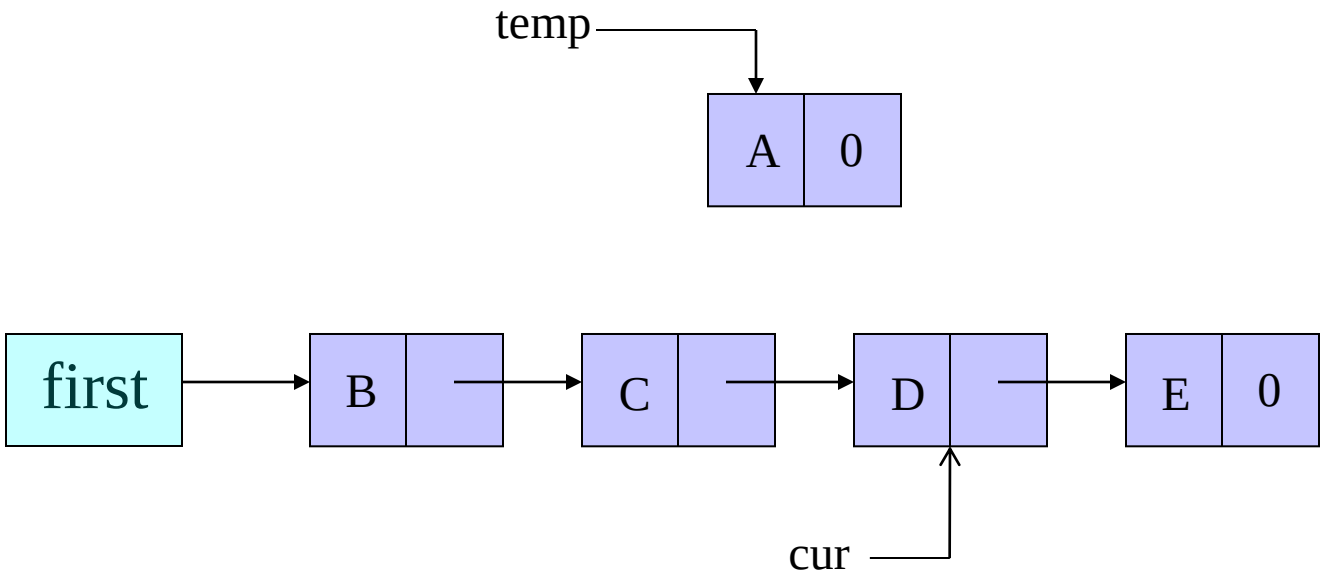
عملیات لیست پیوندی یک طرفه

افزودن گره جدیدی در ابتدای لیست

```
void  LinkedList :: InsertFirst (char value)
{
    Node * temp = new Node (value);

   if( IsEmpty() )      //no node
         first = temp;
    else
      {   temp → next= first;
          first = temp;
      }
}
```

عملیات لیست پیوندی یک طرفه

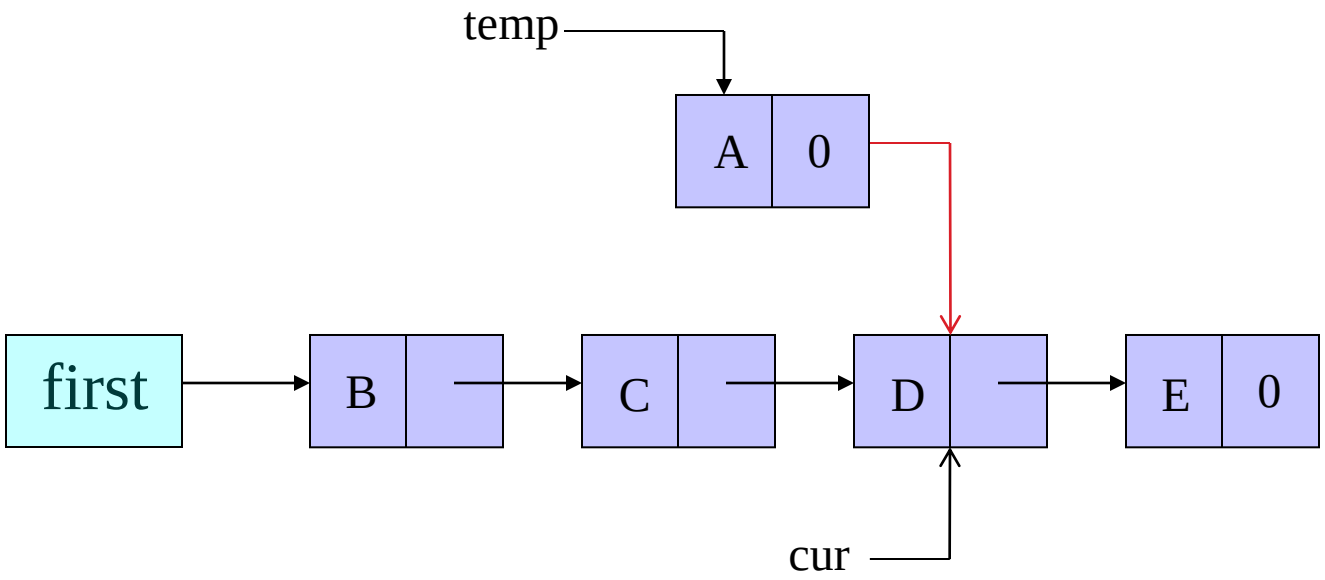
افزودن گره جدیدی قبل از گره مشخصی در لیست



Node *temp = new Node (‘A’);

عملیات لیست پیوندی یک طرفه

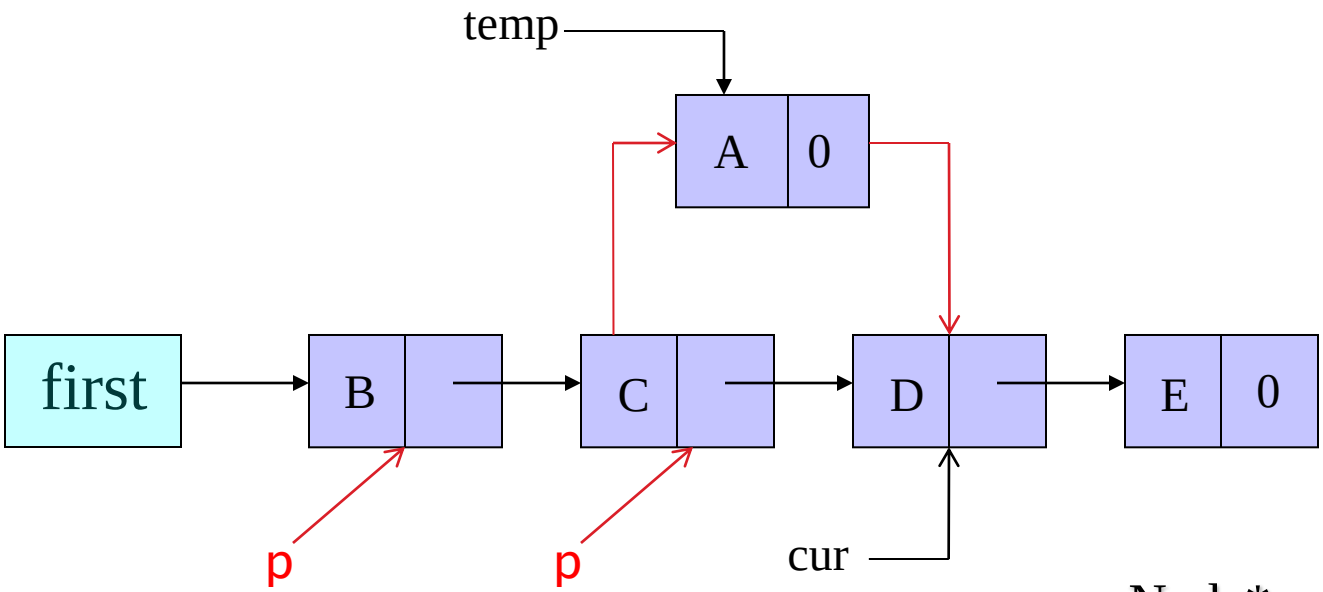
افزودن گره جدیدی قبل از گره مشخصی در لیست



temp → next = cur;

عملیات لیست پیوندی یک طرفه

افزودن گره جدیدی قبل از گره مشخصی در لیست




```
Node*  p = first;
While (p → next != cur)
        p = p → next;
p → next = temp;
```

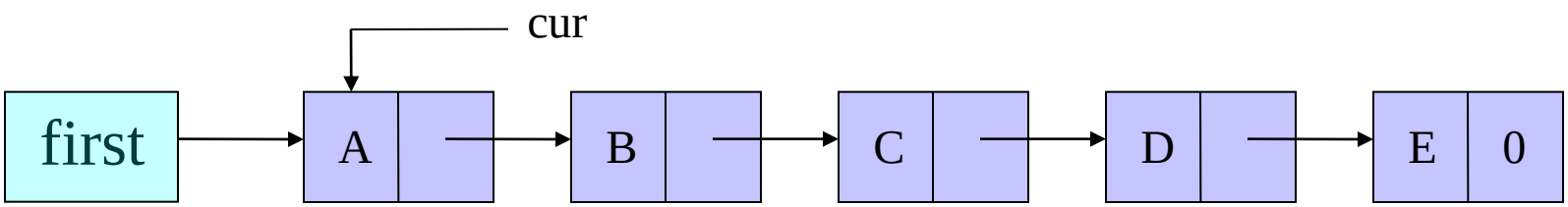
عملیات لیست پیوندی یک طرفه

افزودن گره جدیدی قبل از گره مشخصی در لیست

```
void  LinkedList :: InsertBefore(Node* cur, char value)
{
    Node * temp= new Node (value);

   if( cur == first )
          InsertFirst (value);
    else
   {      Node*  p = first;
          While (p → next != cur)
               p = p → next;
          p → next = temp;
    }
}
```

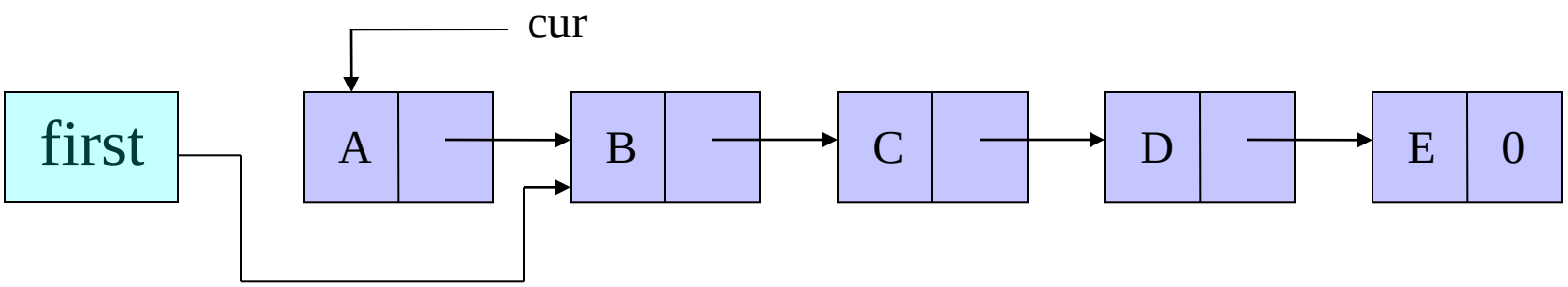
عملیات لیست پیوندی یک طرفه

حذف گره از ابتدای لیست



عملیات لیست پیوندی یک طرفه

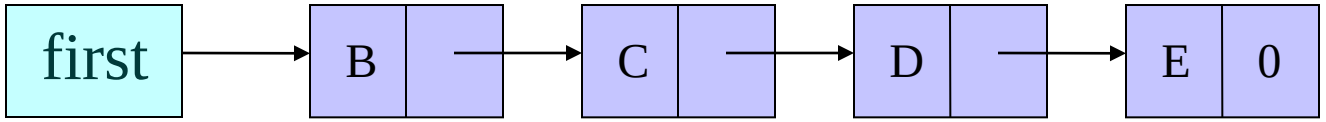
حذف گره از ابتدای لیست



first = first →next;

عملیات لیست پیوندی یک طرفه

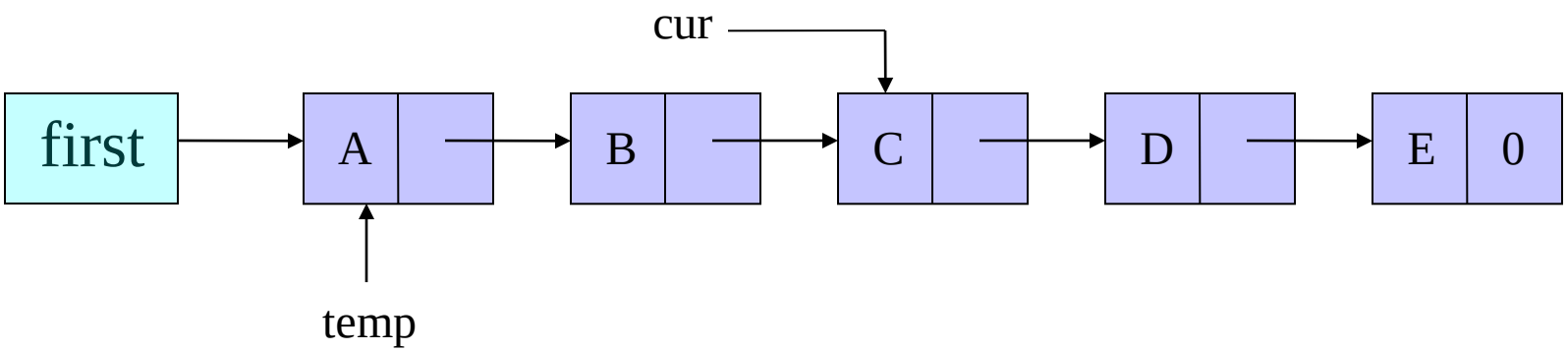
حذف گره از ابتدای لیست



delete cur;

عملیات لیست پیوندی یک طرفه

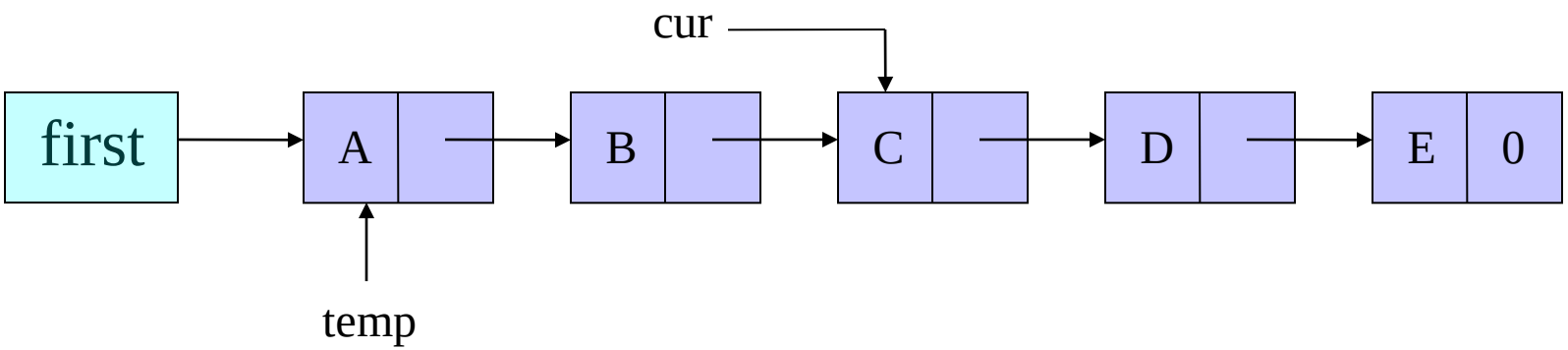
حذف گره از وسط یا انتهای لیست



Node *temp = first;

عملیات لیست پیوندی یک طرفه

حذف گره از وسط یا انتهای لیست

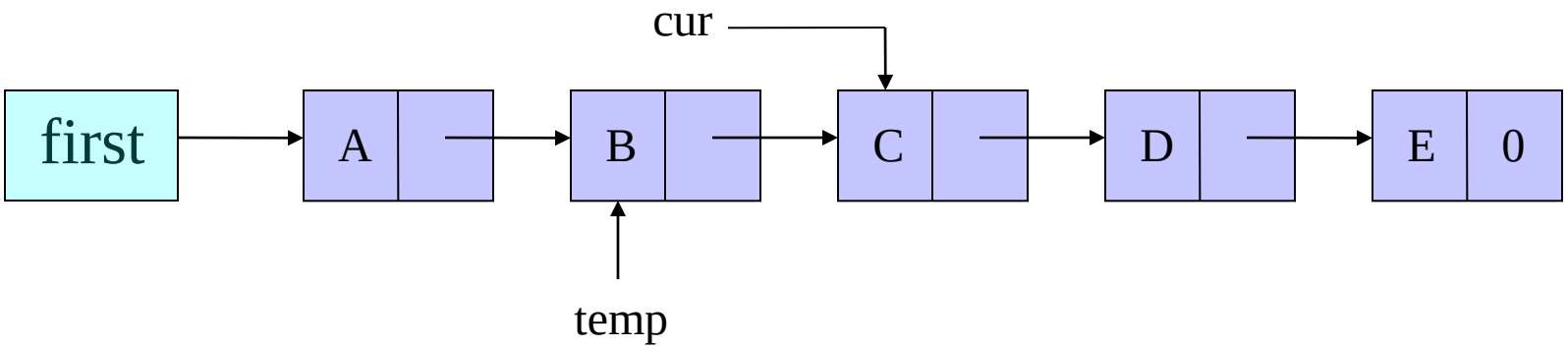


while(temp → next != cur)

temp = temp → next;

عملیات لیست پیوندی یک طرفه

حذف گره از وسط یا انتهای لیست

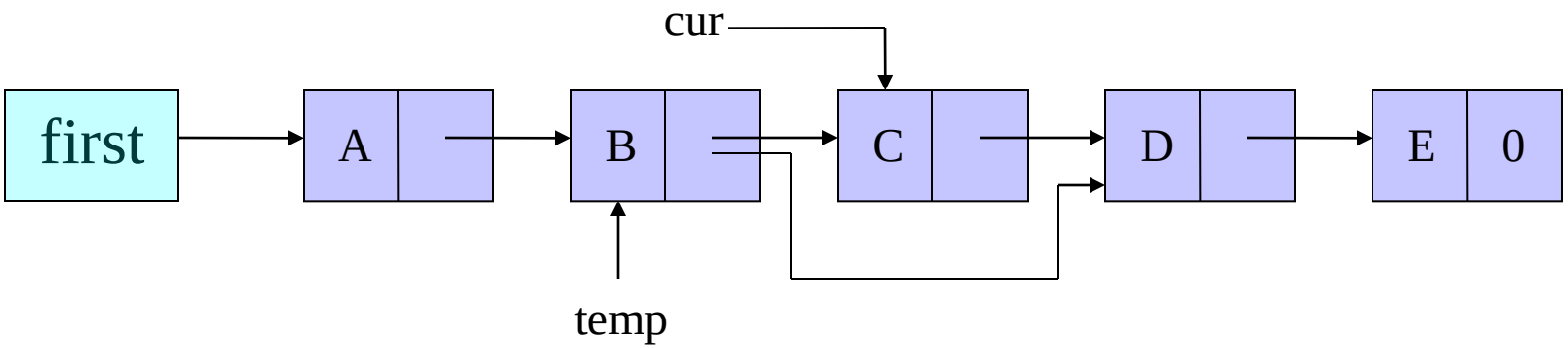


while(temp → next != cur)

temp = temp → next;

عملیات لیست پیوندی یک طرفه

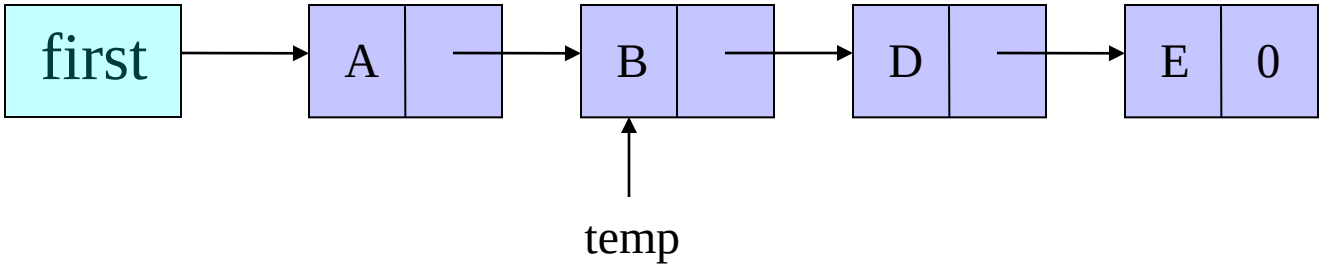
حذف گره از وسط یا انتهای لیست



temp → next = cur → next;

عملیات لیست پیوندی یک طرفه

حذف گره از وسط یا انتهای لیست



delete cur;

عملیات لیست پیوندی یک طرفه

حذف گره از لیست

```
char  LinkedList :: Delete( Node *cur)
{
  char  value = cur → data;

  if( cur == first )
       first = first → next;

  else {
      Node *temp = first;
      while ( temp → next != cur)
           temp = temp → next;

       temp → next= cur → next;
    }

    delete cur;
    return  value;
}
```

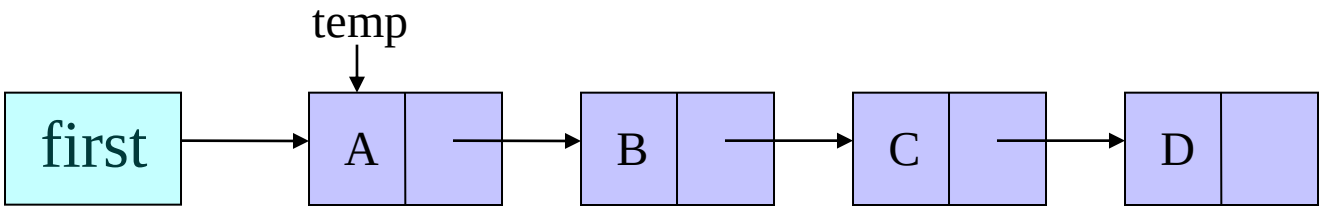
حذف از ابتدای لیست

حذف از وسط یا انتهای لیست

عملیات لیست پیوندی یک طرفه

نمایش لیست پیوندی (پیمایش)

Output : A

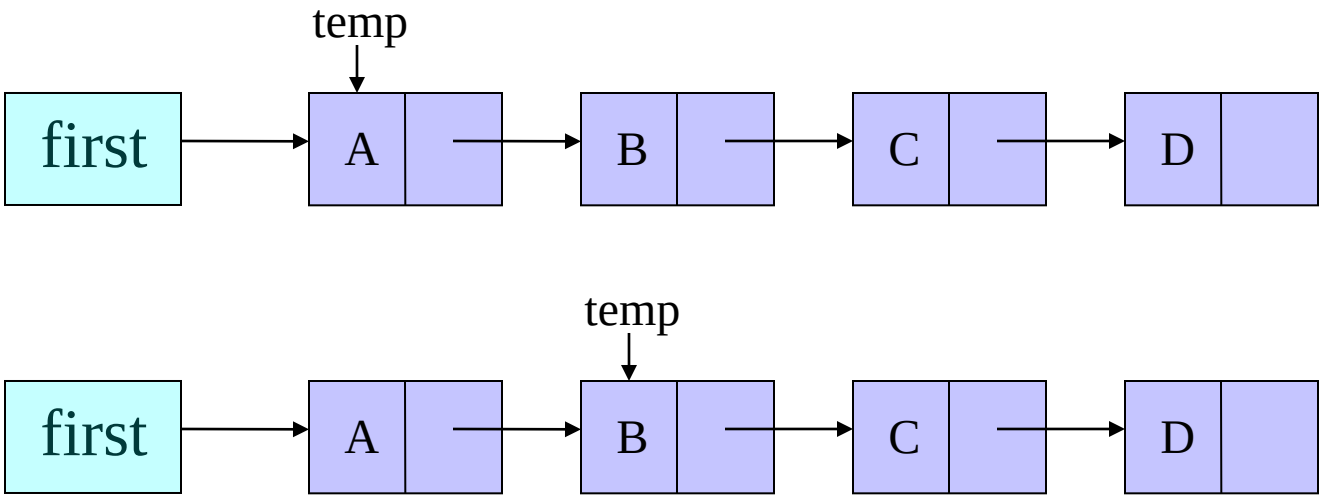


temp = first;
cout << temp → data;

عملیات لیست پیوندی یک طرفه

نمایش لیست پیوندی (پیمایش)

Output : A B



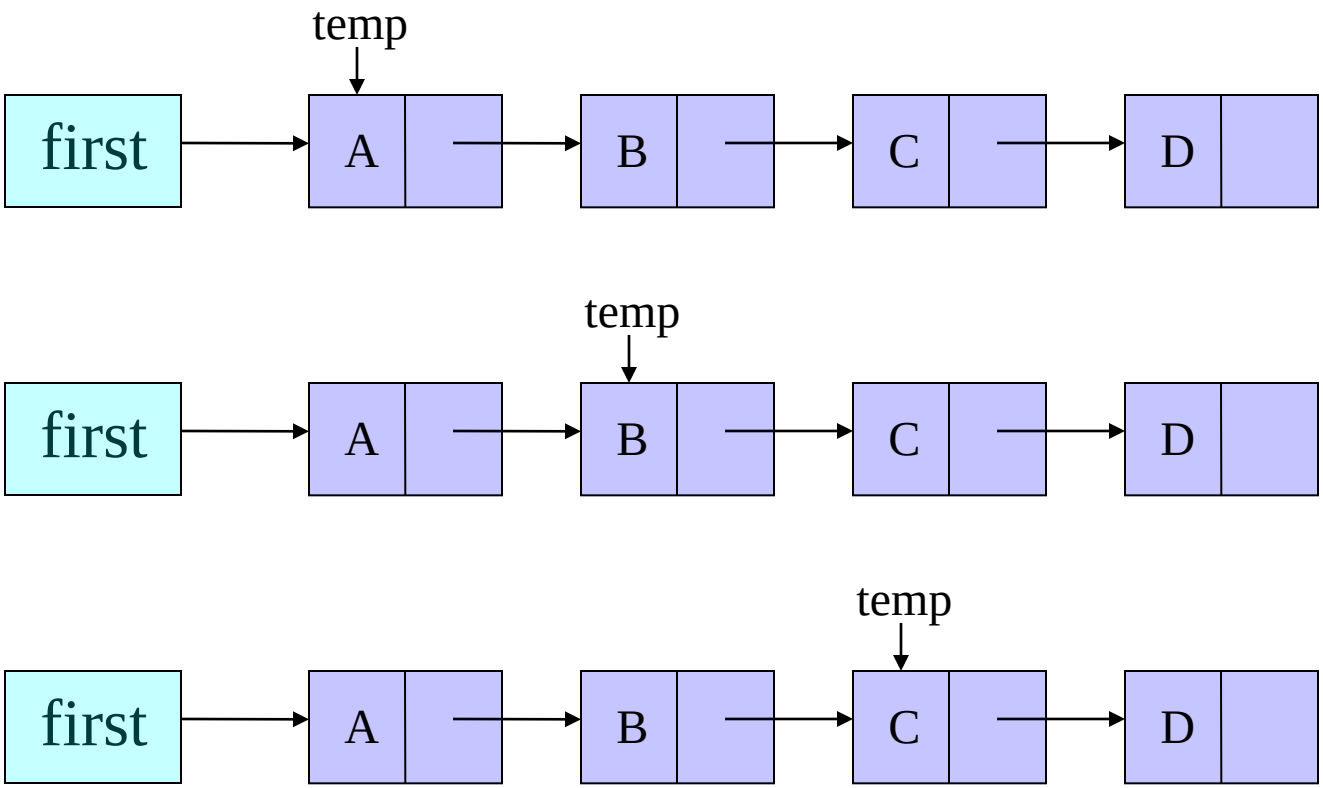
temp = temp → next;

cout << temp → data;

عملیات لیست پیوندی یک طرفه

نمایش لیست پیوندی (پیمایش)

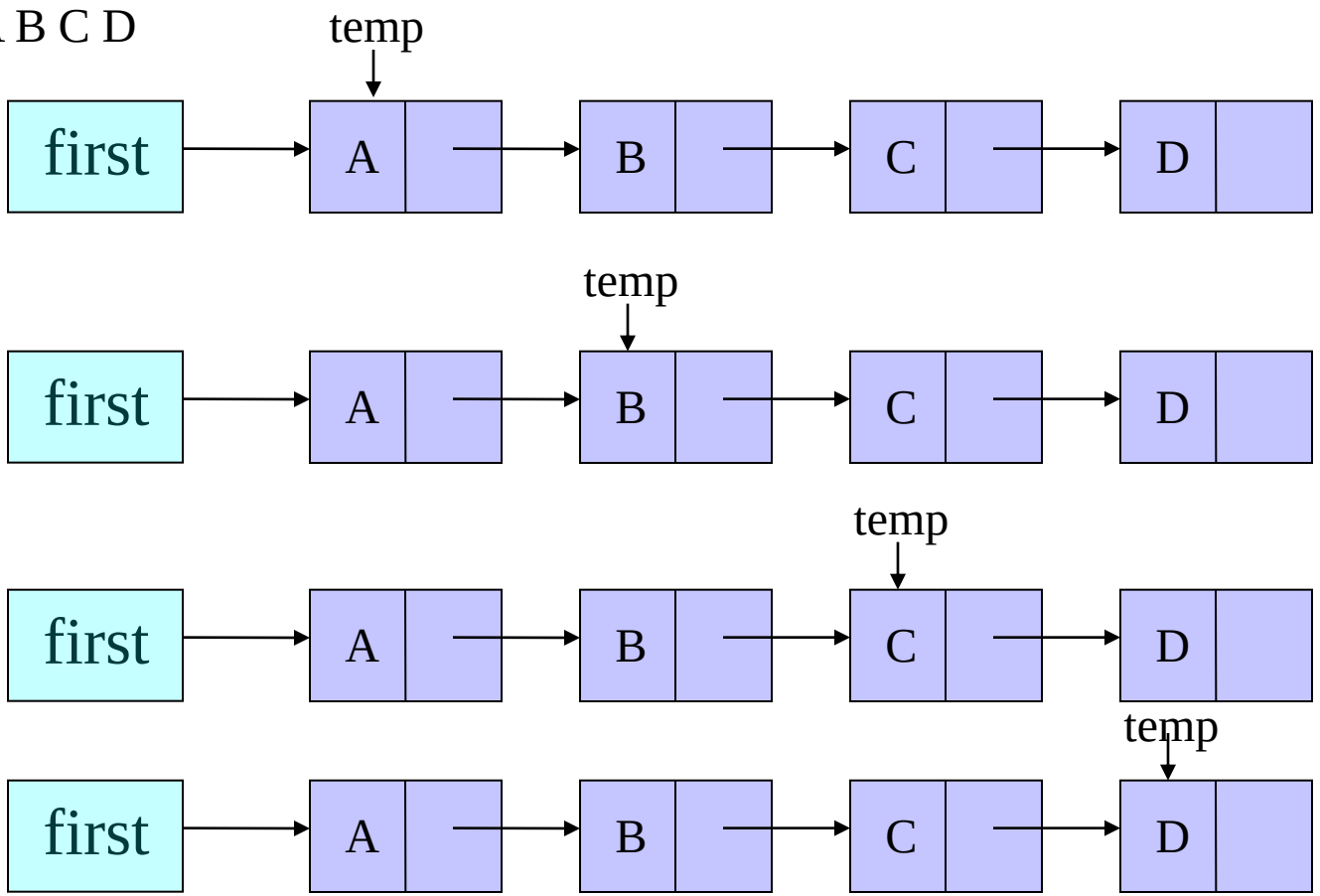
Output : A B C



عملیات لیست پیوندی یک طرفه

نمایش لیست پیوندی (پیمایش)

Output : A B C D



عملیات لیست پیوندی یک طرفه

نمایش لیست پیوندی (پیمایش)

```
void LinkedList :: display()
{
   if( IsEmpty() ) return;                    //no node

   Node *temp = first;

   while ( temp)                              //traverse list
   {
         cout << temp → data;
         temp = temp → next; //next node
   }
}
```

عملیات لیست پیوندی یک طرفه

جستجوی گره ای در لیست پیوندی

```
Node* LinkedList :: search (char key)
{
   if( IsEmpty() ) return Null;

   Node *temp = first;
   while ( temp)
   {
        if (temp->data == key)
             return  temp;

         temp = temp → next;   //next node
   }

   return Null;
}
```

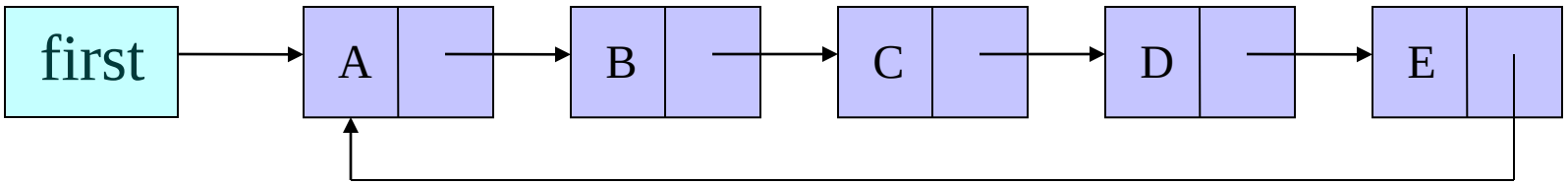
عملیات لیست پیوندی یک طرفه

حذف همه گره های لیست پیوندی

```
LinkedList::~ LinkedList()
{
   Node *current = first;
   Node *temp;

   while (current)
    {
        temp = current;
        current = current →next;
        delete temp;
    }

}
```

لیست پیوندی یکطرفه حلقوی

- لیست پیوندی یک طرفه ای است که آخرین گره آن به گره اول اشاره میکند.
- به عبارت دیگر، به وسیله ی اتصال آخرین گره به اولین گره می توان لیست حلقوی را تولید کرد.
- در این گونه لیست ها به راحتی می توان از آخر لیست به اول لیست پرش کرد.



لیست پیوندی یکطرفه حلقوی

- با توجه به اینکه در لیست پیوندی یک طرفه، پیمایش از ابتدا به انتهای آن بوده و امکان دسترسی به گره های قبلی وجود ندارد، در لیست حلقوی می توان با یک بار پیمایش، به گره های قبلی دسترسی داشت.

- بدین منظور، اگر لیست پیوندی حلقوی دارای n گره باشد، با حداکثر n-1 پیمایش میتوان به کلیه گره های قبلی دسترسی داشت.

عملیات لیست یکطرفه حلقوی

- اکثر عملیاتی که روی لیست معمولی انجام می دادیم می توانیم روی لیست حلقوی انجام دهیم. فقط اینکه باید بعد از تغییرات، لیست همچنان حلقوی بماند.

نمایش(پیمایش) لیست حلقوی

```
void  CLinkedList :: display()
{
   if( first==Null) return;          //no node

   Node *temp = first;
   do                                //traverse list
   {
         cout << temp → data;
         temp = temp → next;   //next node
   } while (temp != first)
}
```

عملیات لیست یکطرفه حلقوی

درج در لیست حلقوی

درج بعد و قبل از یک گره مشخص در لیست یکطرفه حلقوی:

- همانند لیست ساده یکطرفه است.

درج در ابتدای لیست یکطرفه حلقوی:

چون گره آخر لیست باید به گره اول لیست اتصال یابد، باید تابع جدیدی نوشته شود. با دو روش می توان این عمل را پیاده سازی کرد:

۱- لیست را برای پیدا کردن آخرین عنصر پیمایش کرد. پیچیدگی زیاد: O(n)

۲- یک اشاره گر جدید last به عنصر آخراشاره کند. پیچیدگی کم: O(1) ✓

عملیات لیست حلقوی

درج در ابتدای لیست حلقوی (افزودن اشاره گر Last)

```
void  CLinkedList :: InsertFirst (char value)
{
    Node* temp =new  Node (value);

    if( !first )                    //no node
    {
        last = first = temp;
        first → next = first;
    }
  else
  {
        temp → next= first;
        last → next= temp;
        first = temp;
  }
}
```

مرتبه زمانی: O(1)

✓این روش به دلیل پیچیدگی زمانی کمتر مناسب تر است

حذف از لیست یکطرفه حلقوی

```
void  CLinkedList :: Delete ( Node *cur)
{
  char  value = cur → data;
  if( cur == first )
   {
      if (first → next== first)
            first=last=Null;
      else
       {   first = first → next;
           last → next = first;
       }
   }
else {
     Node *temp = first;
     while( temp → next != cur)
           temp = temp → next;

    temp → next = cur → next;
    }
    delete cur;
    return value;
}
```

حذف از لیست با یک گره

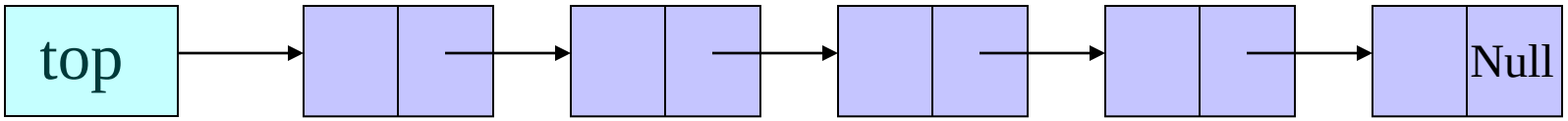
حذف از ابتدای لیست با بیش از یک گره

حذف سایر گره های لیست

پیاده سازی پشته با لیست پیوندی

- همان طور که گفته شد، پشته نیمه ثابت است. به عبارتی ثابت یا پویا بودن پشته به روش پیاده سازی آن بستگی دارد.
- اگر پشته با آرایه پیاده سازی شود (ثابت) و اگر با لیست پیوندی پیاده شود (پویا) است.
- در این قسمت با پیاده سازی پویای پشته آشنا می شویم.

- می دانید که در پشته داده ها از بالای آن (top) درج و حذف می شوند، یعنی ورود و خروج داده ها از یک طرف انجام می شود.

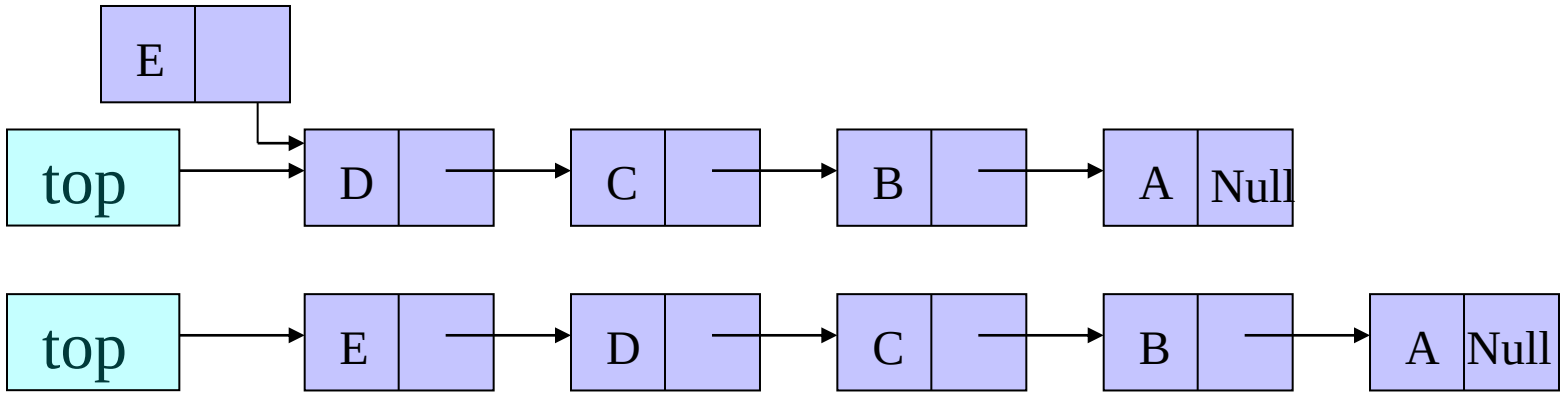


- در پشته پیوندی، به جای اشاره گر first از اشاره گر top استفاده می کنیم.

کلاس پشته

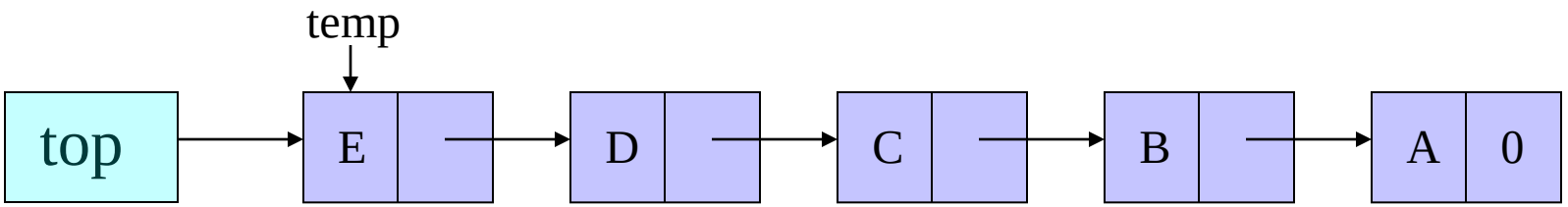
```
class  Stack
{
   public :
         Stack()    { top = Null; }
         void  Push( int );
         int  Pop();
   private:
         Node *top;
};
//.....

class  Node
{
   public:
         int  data;
         Node  *next;
         Node (int d) { data=d;  next=null; }
};
```

درج در پشته پیوندی



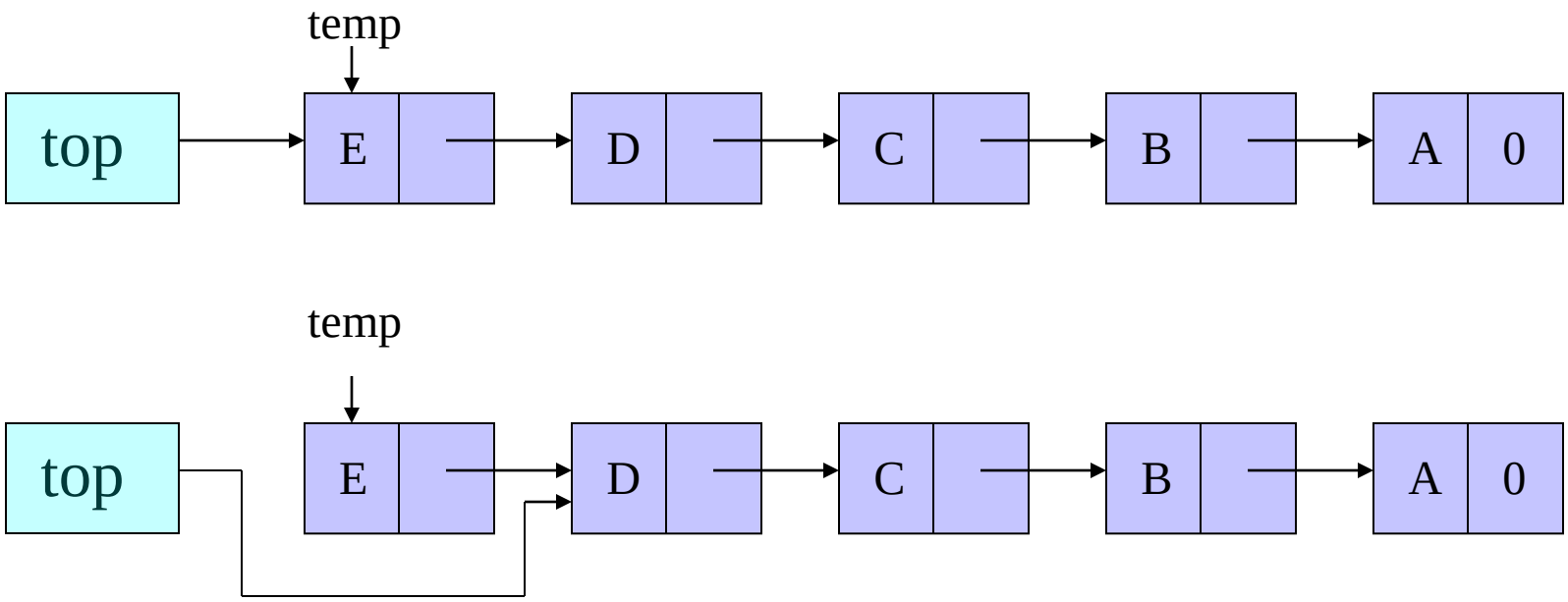

```
void  Stack :: Push (int value)
{
    Node* temp = new Node(value);
    temp → next = top ;
    top = temp;
}
```

حذف از پشته پیوندی



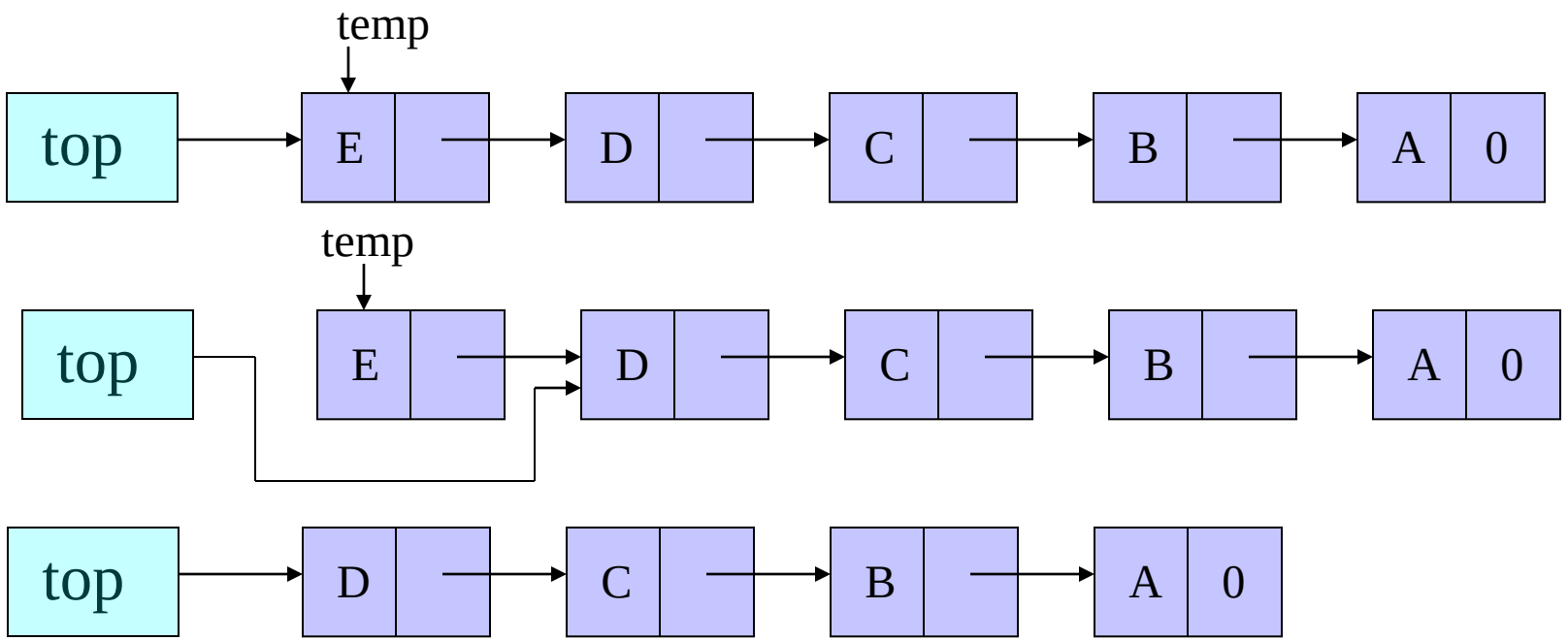
temp = top;

حذف از پشته پیوندی



top = temp → next;

حذف از پشته پیوندی



delete temp;

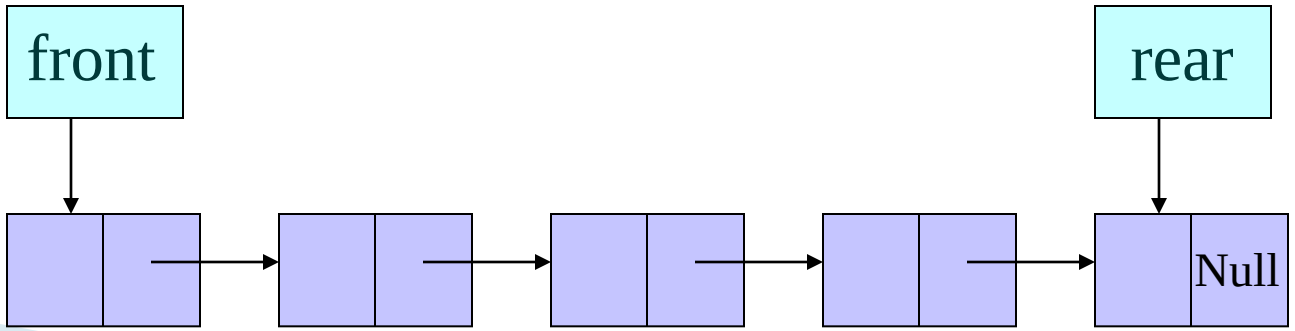
حذف از پشته پیوندی

```
int  Stack :: Pop()
{
   if( !top )  {cout << “empty stack”;   return -1;}

   Node *temp = top;

   int  retVal = temp → data;

   top = top → next;

   delete  temp;

   return  retVal;
}
```

صف های پیوندی

- صف نیز مانند پشته ساختمان داده نیمه ثابت است که اگربا آرایه پیاده سازی شود (صف ثابت) و اگر با لیست پیوندی پیاده شود (صف پویا) است.
- در این قسمت با پیاده سازی پویای صف آشنا می شویم.

- می دانید که در صف داده ها از انتها (rear) به آن اضافه می شوند و از ابتدا (front) از آن حذف میشوند. به عبارتی ورود اطلاعات از یک طرف و خروج اطلاعات از طرف دیگر است.
- به این دلیل به دو اشاره گر به نام های rear و front نیاز داریم.

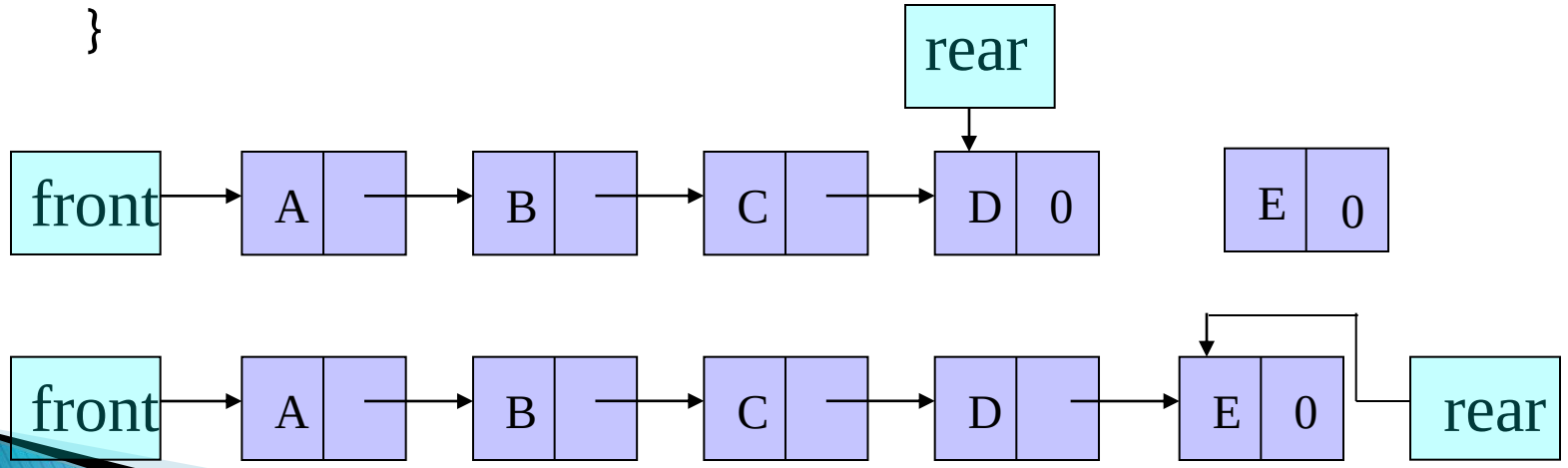


کلاس صف پیوندی

```
class  Queue
{
   public :
         Queue()   {front = rear = Null;}
         void   AddQ (int);
         int   DelQ();
   private:
          Node *rear , *front;
};
//.....................................
class  Node
{
   public :
         int  data;
         Node  *next;
         Node (int d) { data=d; next=Null; }
};
```

درج در صف پیوندی(درج در انتها)

```
void  Queue :: AddQ( int  value)
{
    Node* temp = new  Node(value);

    if( !front )          // no node
        front = rear = temp;
    else
        {   rear → next= temp;
            rear = temp;
        }
}
```



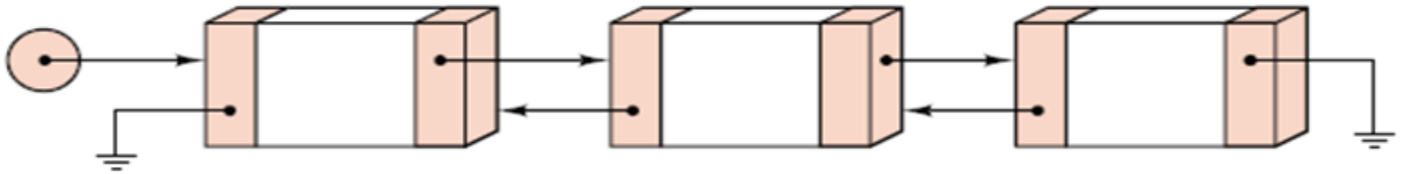
حذف از صف پیوندی(حذف از ابتدا)

```
int  Queue :: DelQ()
{
   if( !front )  { cout << “empty queue.”;  return -1; }

   Node  *temp = front;
   int  retVal = front → data;
   front = front → next;
   delete  temp;
   return  retVal;
}
```

لیست پیوندی دوطرفه(لیست دوپیوندی)

- مشکل اساسی لیست های یک طرفه عدم دسترسی به گره قبلی است.
- لیست های دوطرفه با اضافه کردن یک اشاره گر به گره ی قبلی این مشکل را حل کرده اند.

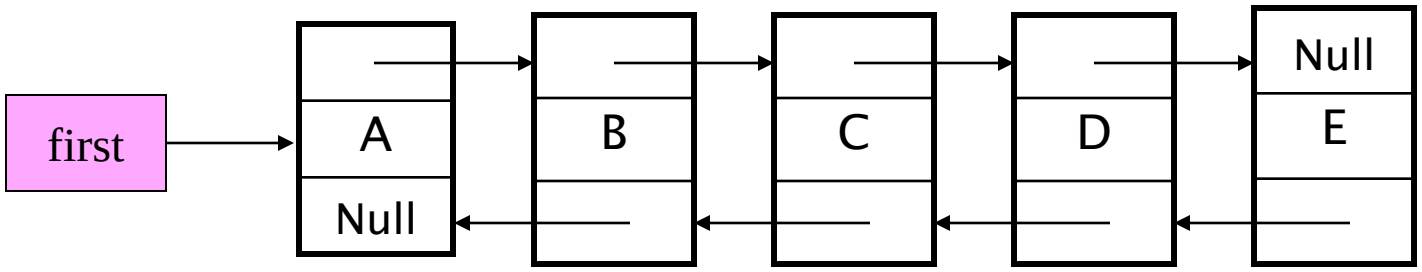
ساختار هر گره لیست دوپیوندی



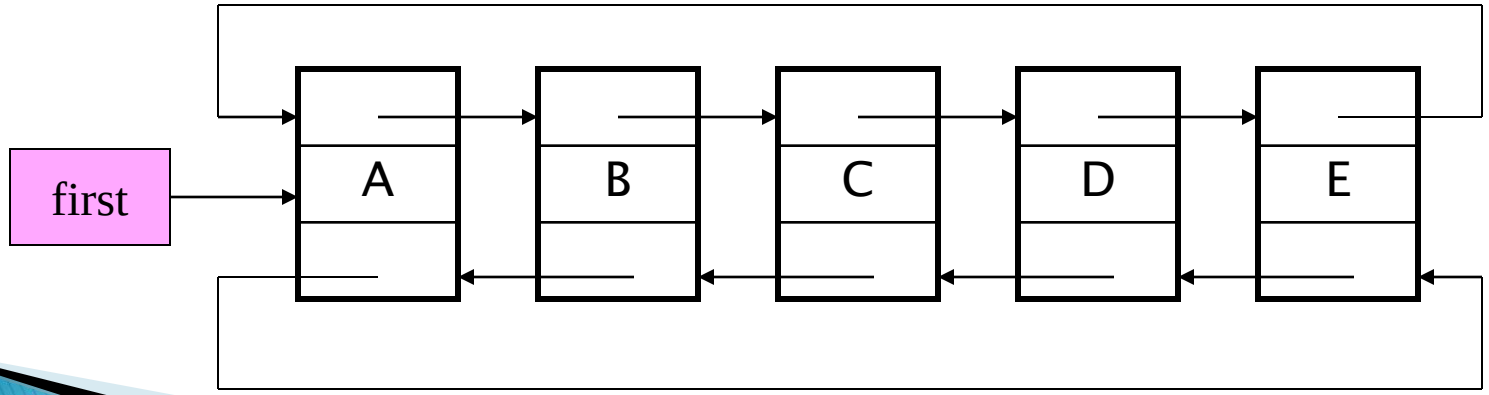
یک لیست دوپیوندی

انواع لیست های دوپیوندی

لیست دوپیوندی ساده

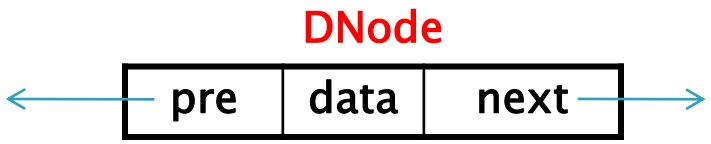


لیست دوپیوندی حلقوی



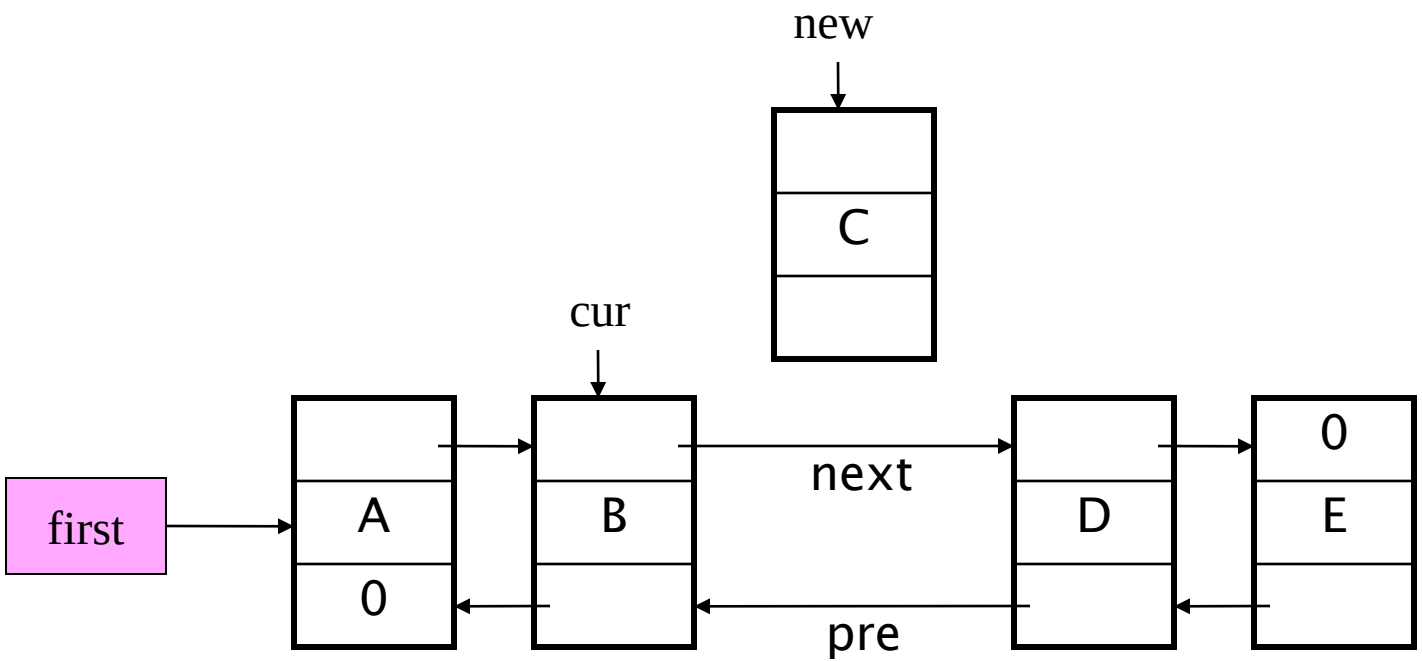
کلاس لیست دوپیوندی ساده

```
class  DNode
{
   public:
         int  data;
         DNode  *next , *pre;
         Dnode (int d) : data(d), next(Null), pre(Null) { }
};
//---------------------------------------
class  DLinkedList
{
   public :
          DLinkedList();
          ~DLinkedList();
          //list manipulation operations
   private:
          DLinkedList  *first;
};
```



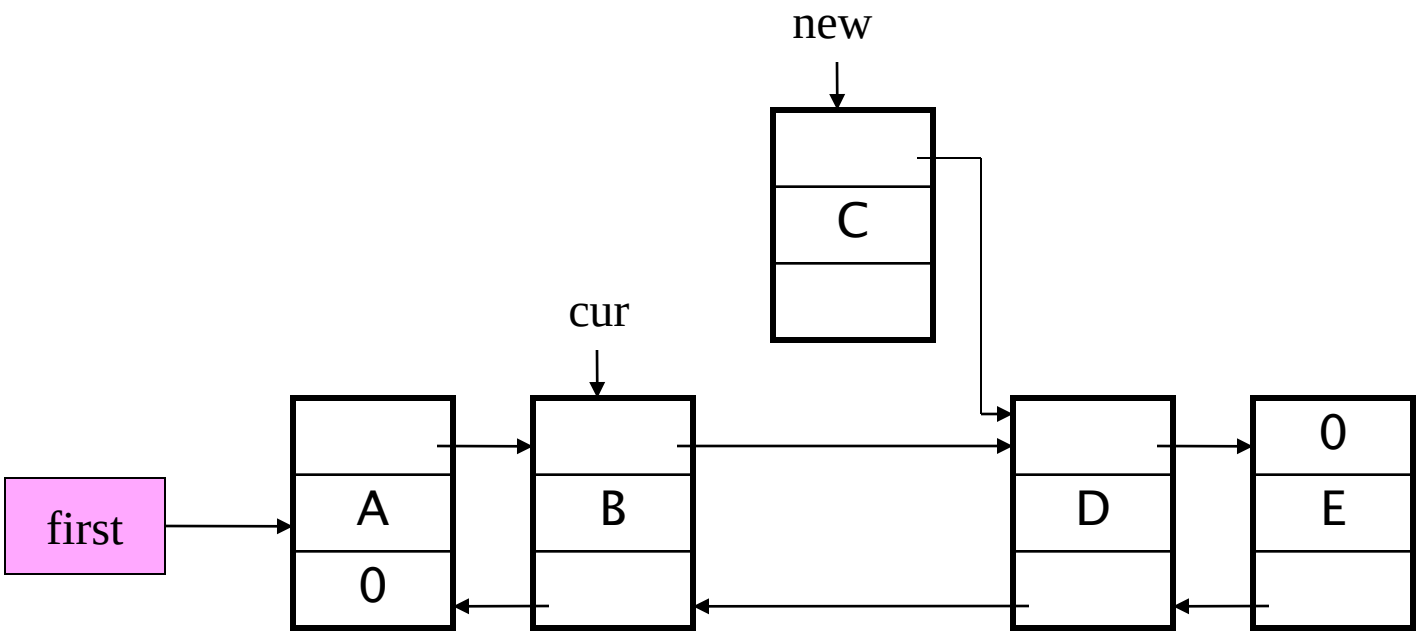
عملیات لیست دوپیوندی ساده

افزودن گره جدیدی در وسط لیست



عملیات لیست دوپیوندی ساده

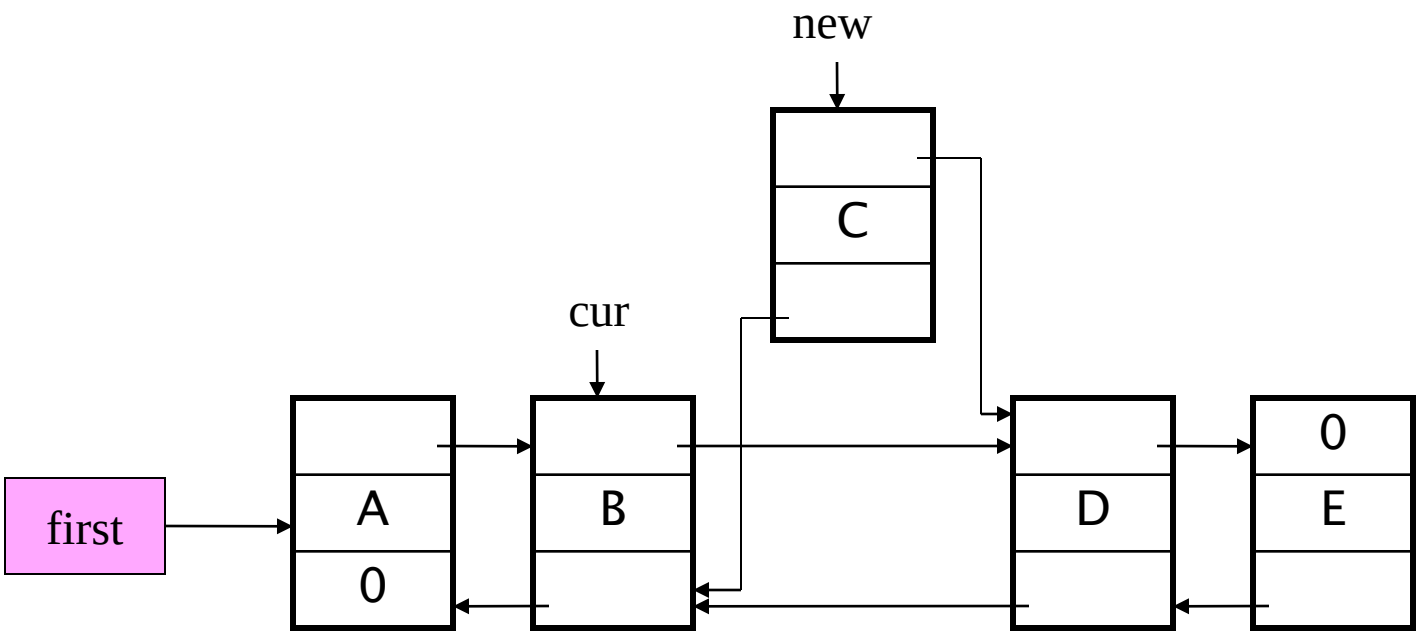
افزودن گره جدیدی در وسط لیست



new → next = cur → next;

عملیات لیست دوپیوندی ساده

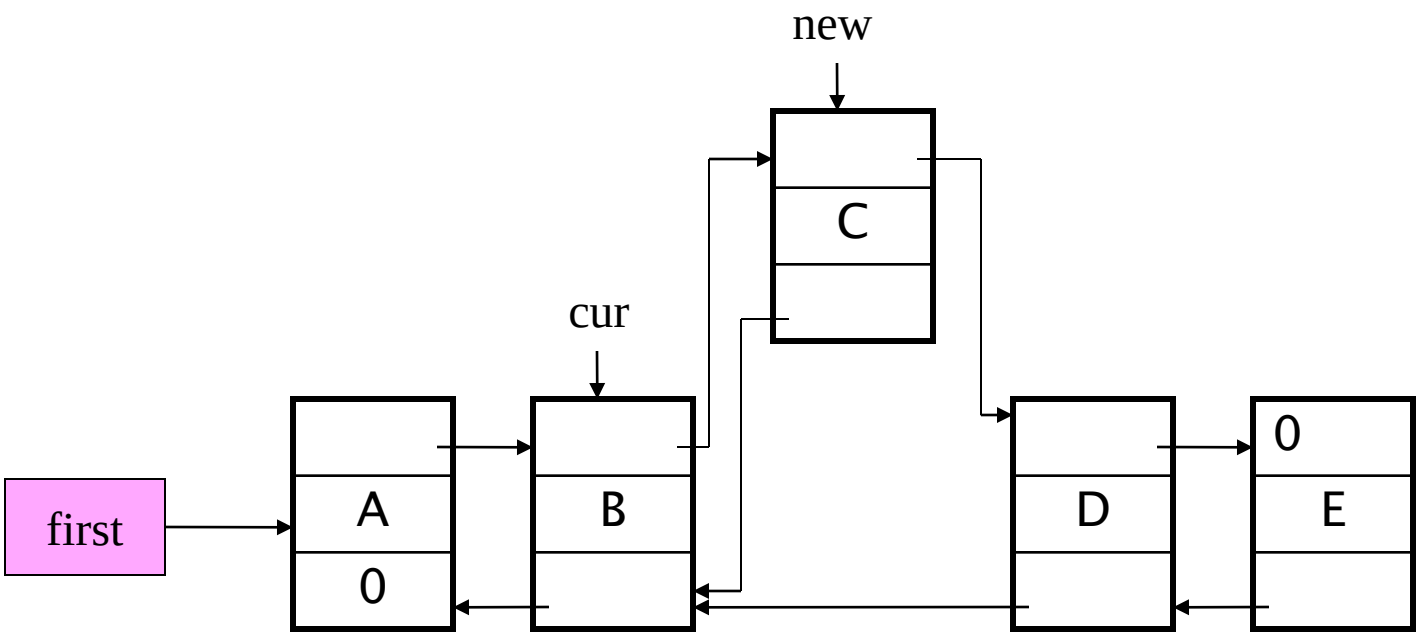
افزودن گره جدیدی در وسط لیست



new→ pre = cur;

عملیات لیست دوپیوندی ساده

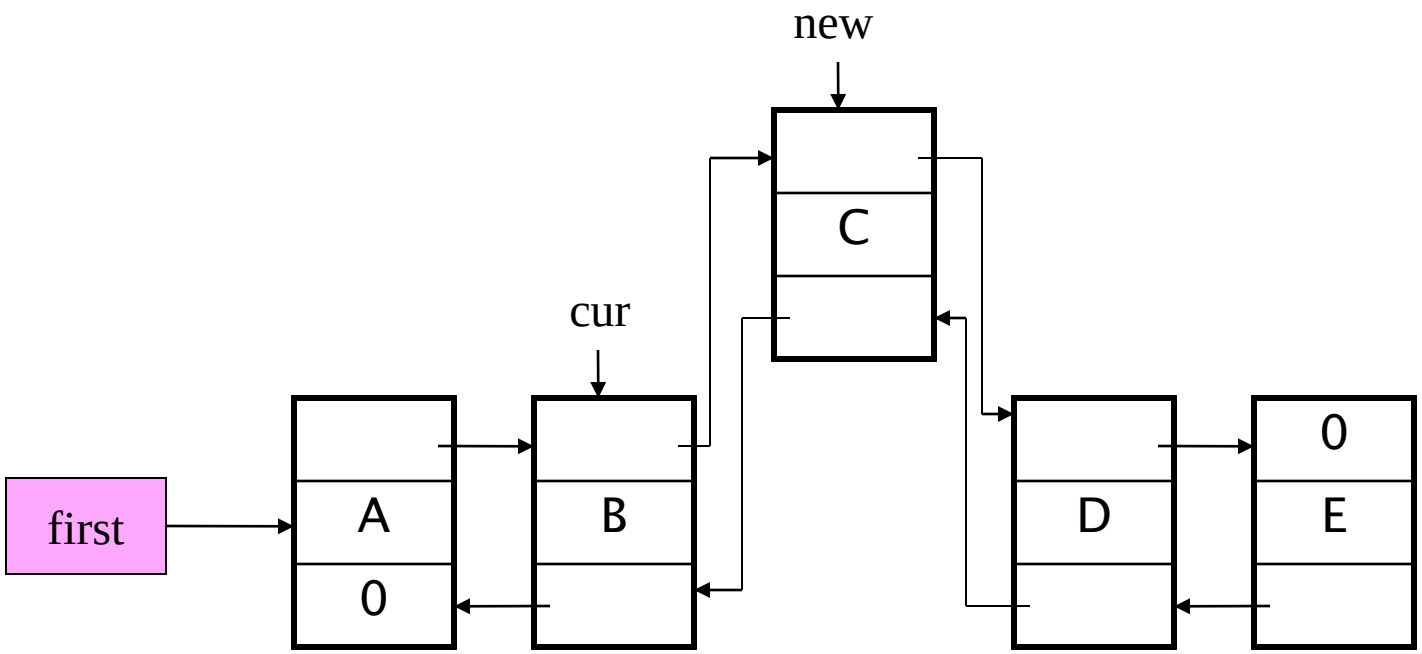
افزودن گره جدیدی در وسط لیست



cur → next = new;

عملیات لیست دوطرفه

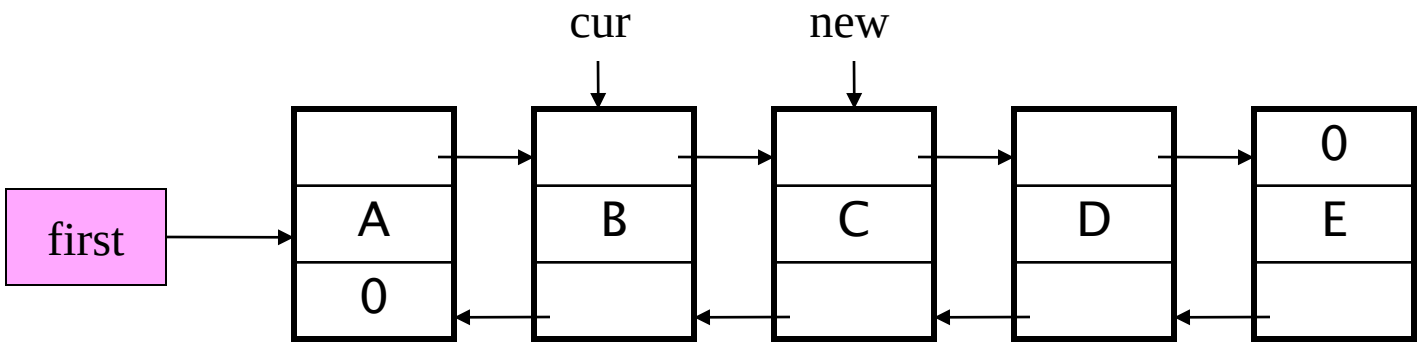
افزودن گره جدیدی در وسط لیست



(new → next) → pre = new;

عملیات لیست دوطرفه

افزودن گره جدیدی در وسط لیست



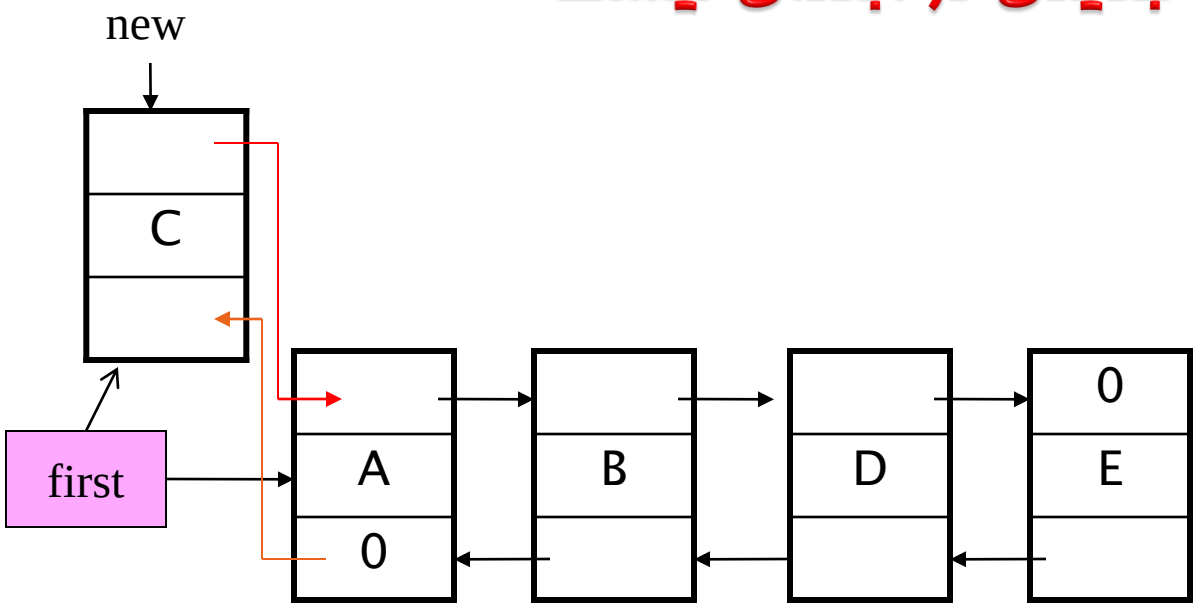
عملیات لیست دوطرفه

افزودن گره جدیدی در وسط یا انتهای لیست

```
void  DLinkedList :: InsertAfter( DNode *cur, int value)
{
   Dnode *new = new Dnode(value);
   new → next = cur → next;
   new → pre = cur;
   cur → next = new;
   if(new → next)                    // no last node
         (new → next) → pre = new;
}
```

عملیات لیست دوطرفه

افزودن گره جدیدی در ابتدای لیست



new → next = first;

first → pre = new;

first = new;

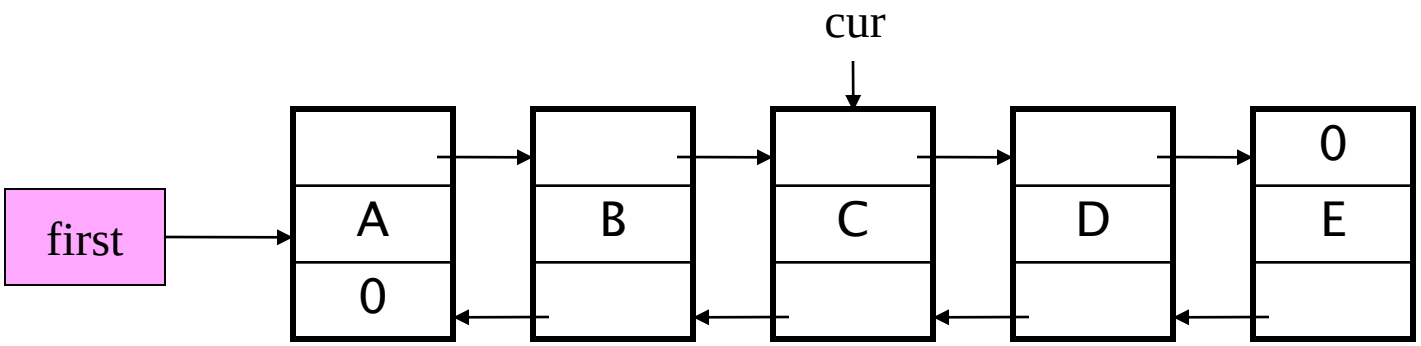
عملیات لیست دوطرفه

افزودن گره جدیدی در ابتدای لیست

```
void  DLinkedList :: InsertFirst( int value)
{
   Dnode *new = new Dnode(value);
   if (!first)    first = new;   // no node

   new → next = first;
   first → pre = new;
   first = new;
}
```

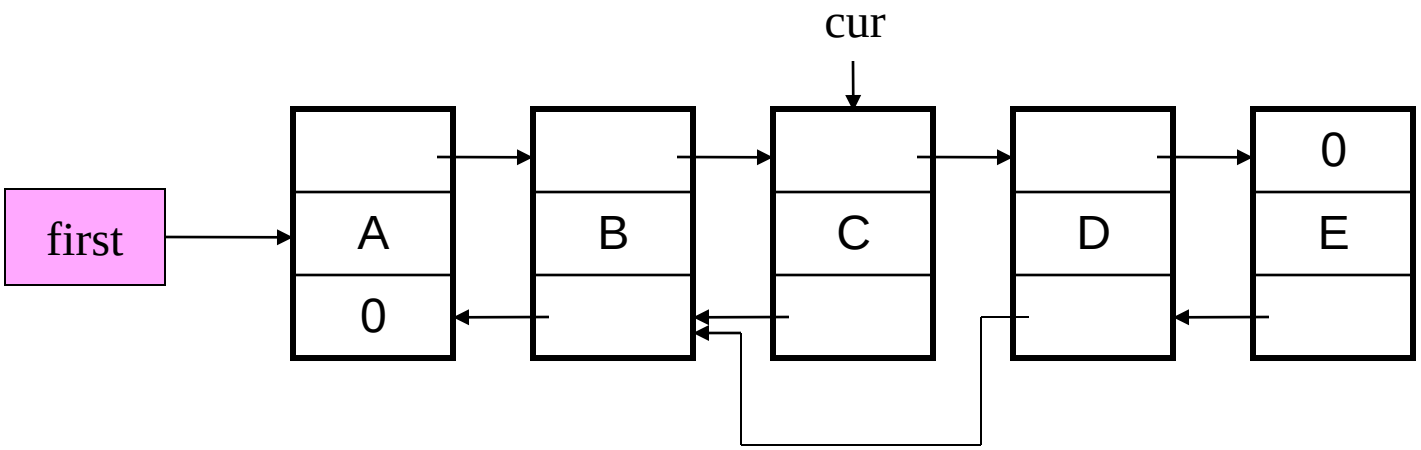
عملیات لیست دوپیوندی ساده

حذف از وسط لیست دو پیوندی



عملیات لیست دوپیوندی ساده

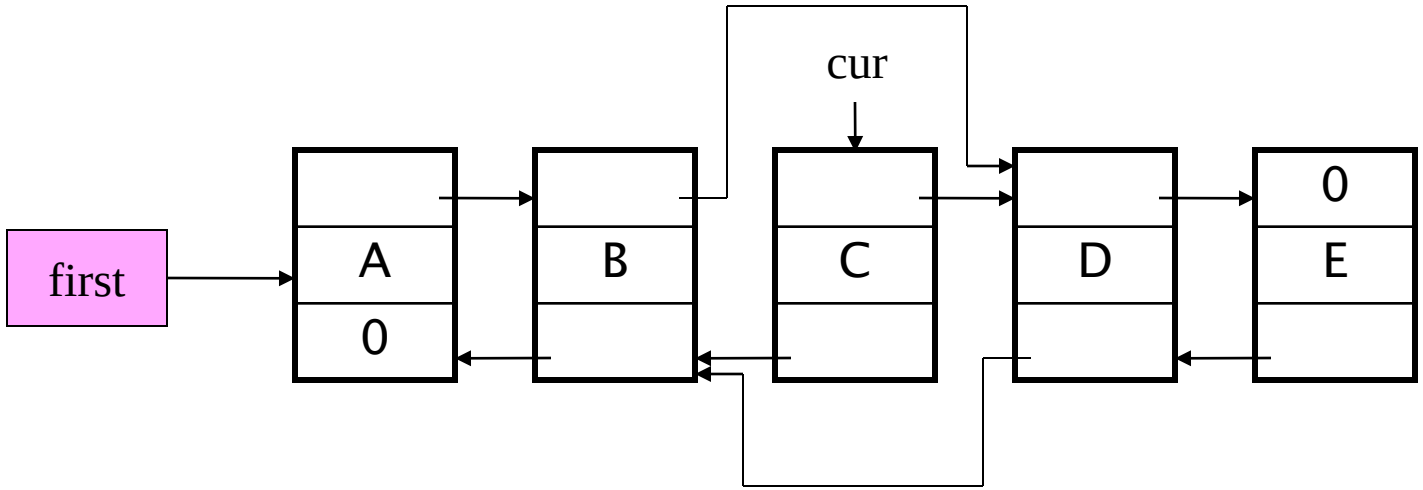
حذف از وسط لیست دو پیوندی



(cur → next) → pre = cur → pre;

عملیات لیست دوپیوندی ساده

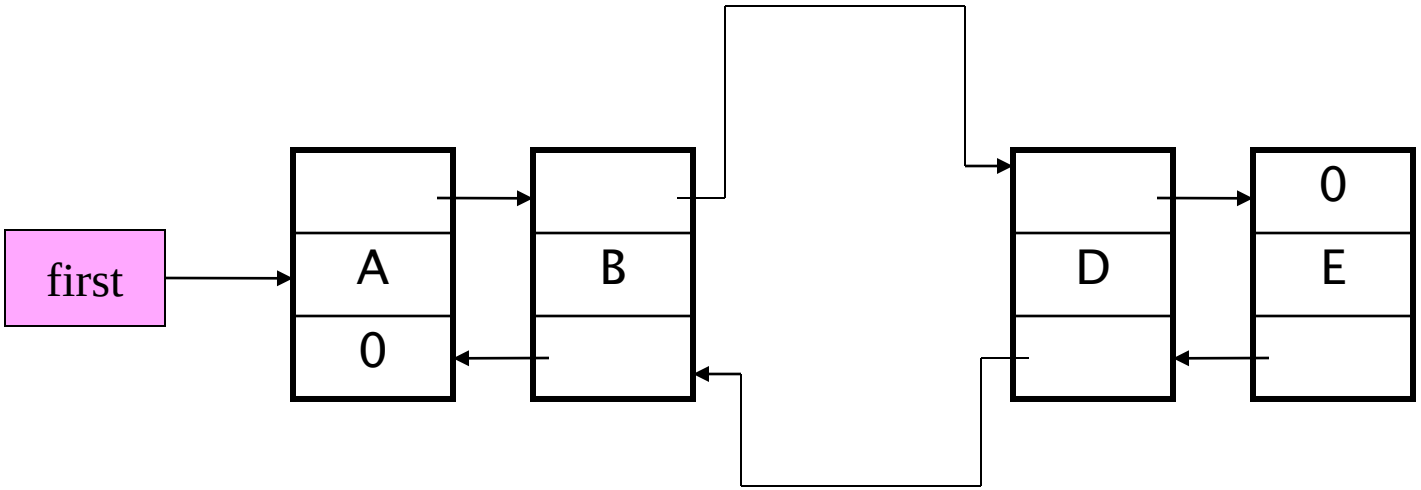
حذف از وسط لیست دو پیوندی



(cur → pre) → next = cur → next;

عملیات لیست دوپیوندی ساده

حذف از وسط لیست دو پیوندی



delete cur ;

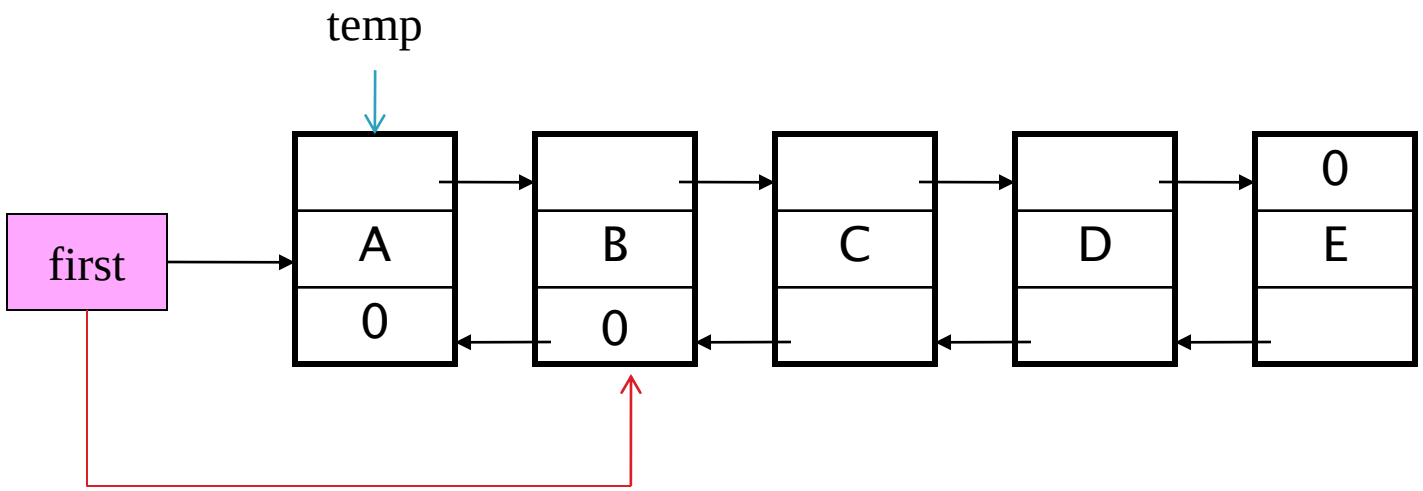
عملیات لیست دوپیوندی ساده

حذف از وسط یا انتهای لیست

```
void  DLinkedList :: DeleteAfter( DNode * cur)
{
   if( cur → next)
   {
         ( cur → next ) → pre = cur → pre;
         ( cur → pre ) → next = cur → next;
   }
  else
         ( cur → pre ) → next = Null;        // last node

   delete cur;
}
```

عملیات لیست دوپیوندی ساده

حذف از ابتدای لیست



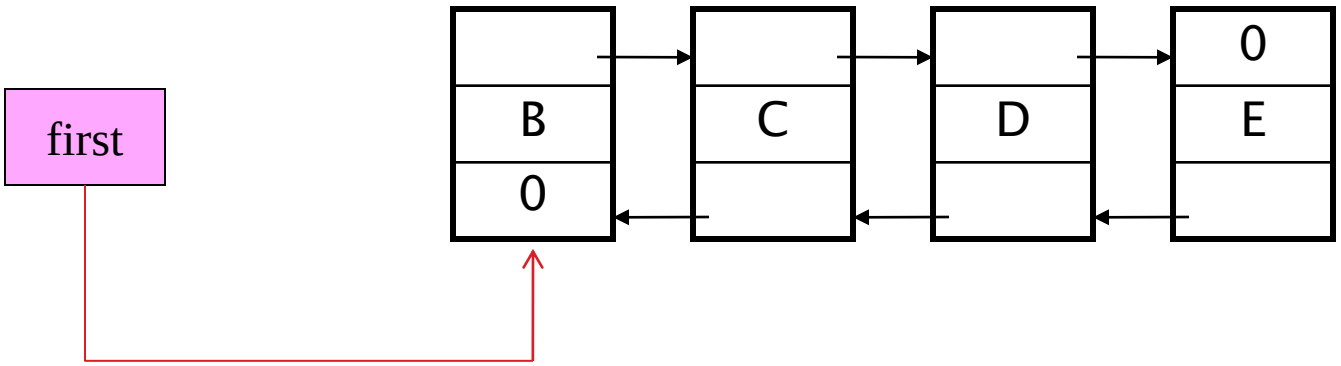
temp = first;

first = first → next ;

first → pre = Null;

عملیات لیست دوپیوندی ساده

حذف از ابتدای لیست




```
delete  temp;
```

عملیات لیست دوپیوندی ساده

حذف از ابتدای لیست

```
void  DLinkedList :: DeleteFirst ( )
{
        if ( (first → next) == Null )    //one node
        {
                delete first;
                return;
        }

        Node *temp = first ;
        first = first → next;
        first → pre = Null;
        delete temp;
        return;
}
```

مشکل لیست دوپیوندی ساده

- در لیست دوپیوندی ساده، برای عمل درج و حذف از انتها، نیاز است کل لیست تا انتها پیمایش شود تا گره آخر لیست به دست آید. ← O(n)

- جهت رفع این مشکل دو راه حل وجود دارد:

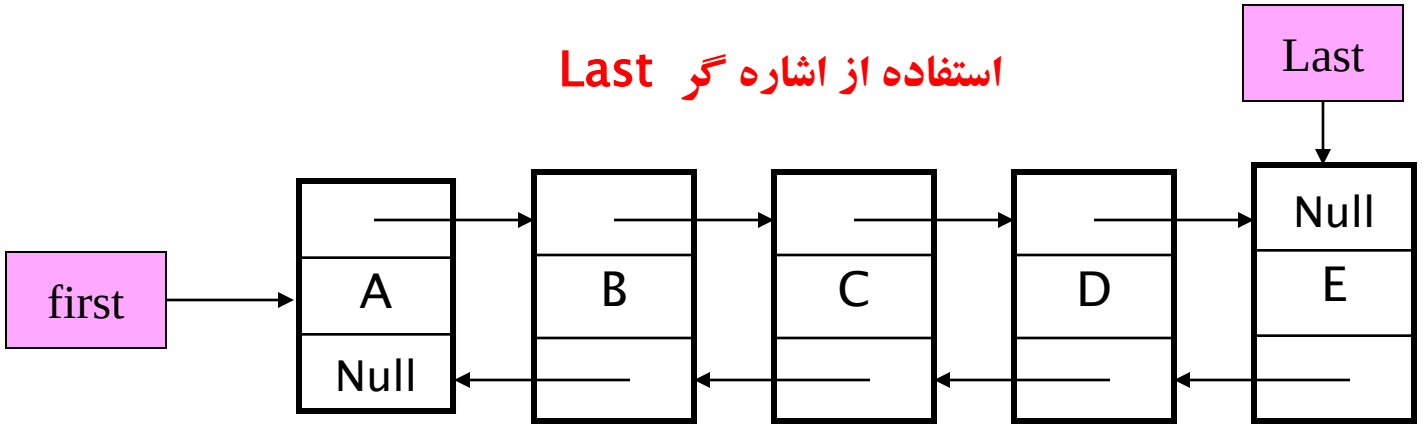
۱- استفاده از اشاره گر دیگری (last) که به گره آخر لیست اشاره کند.

۲- استفاده از لیست دوطرفه حلقوی که در این حالت گره آخر لیست به گره اول لیست اشاره می کند و بالعکس.

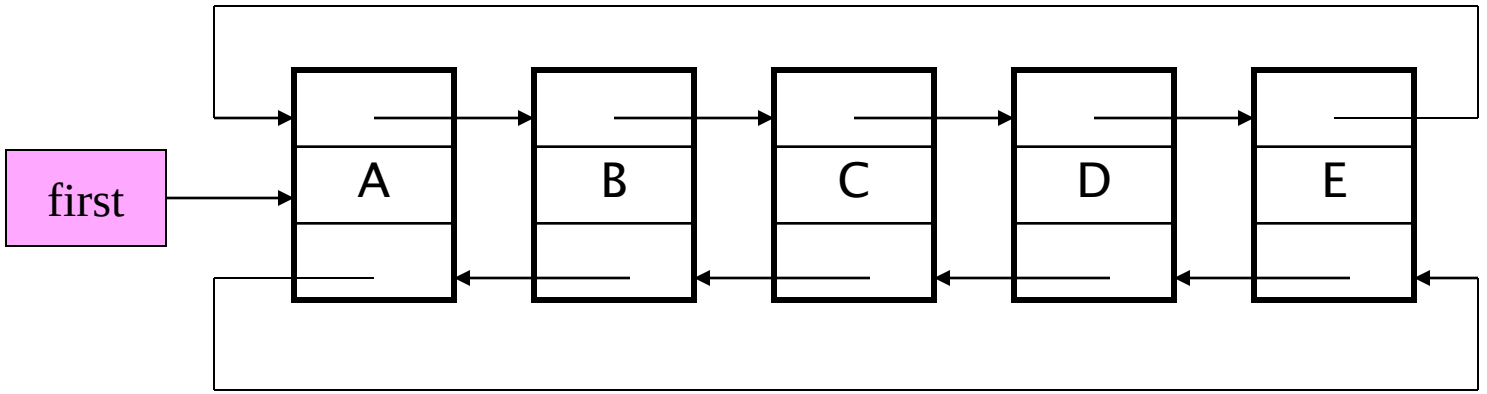
- در هر دو روش پیچیدگی عملیات درج و حذف از انتهای لیست دوطرفه با O(1) انجام می شود.

حل مشکل لیست دوپیوندی ساده

استفاده از اشاره گر Last

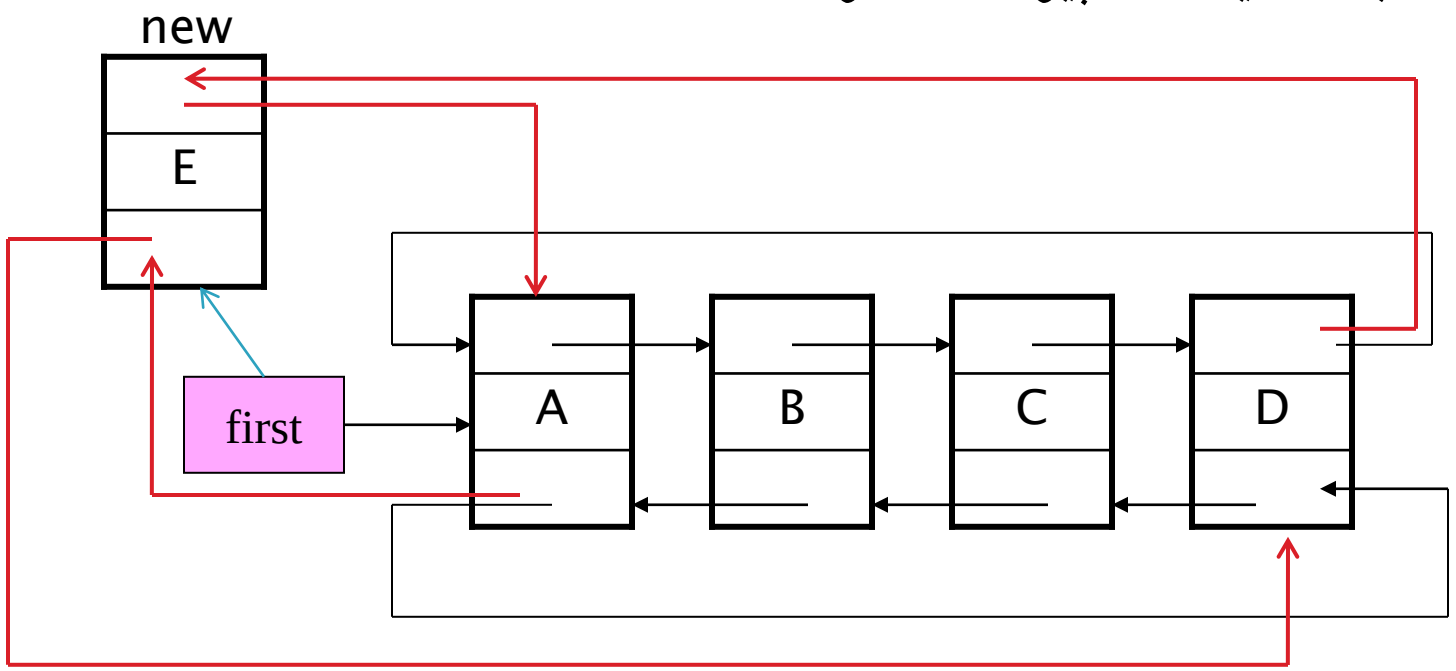


استفاده از لیست دوطرفه حلقوی



درج در لیست دوپیوندی حلقوی

- درج در وسط یا انتهای لیست دوپیوندی حلقوی همانند لیست دوپیوندی ساده است.
- درج در ابتدای لیست دوپیوندی حلقوی:



new → next = first;
new → pre = first → pre;
(first→ pre) → next = new;
first → pre = new;
first = new;

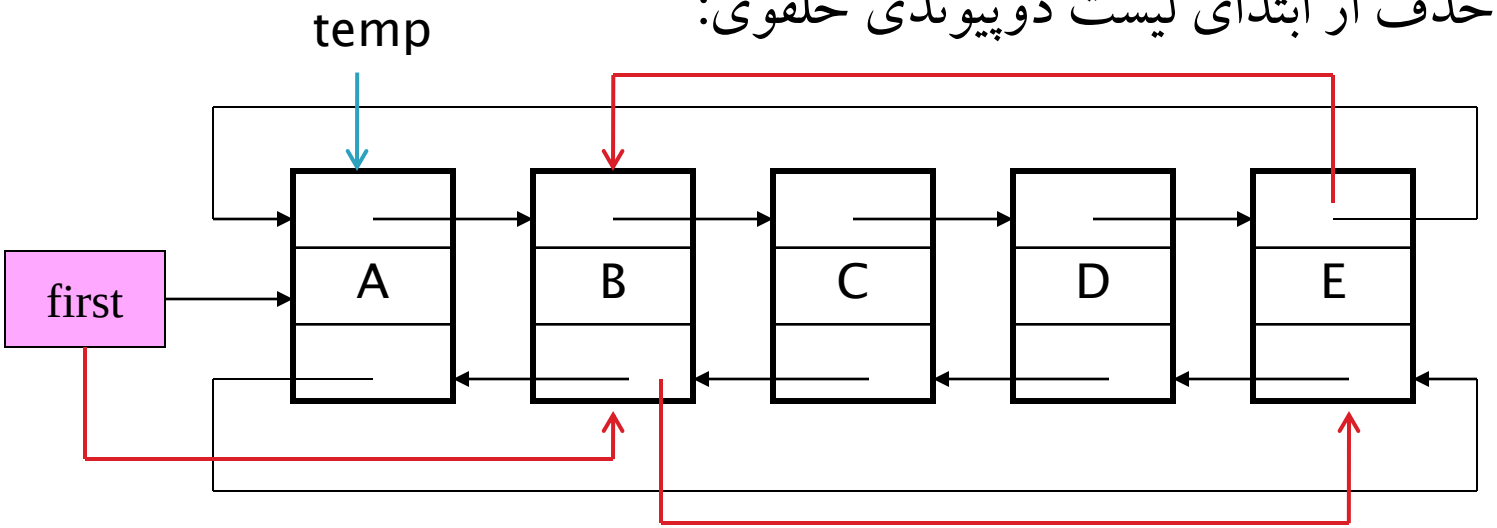
درج در لیست دوپیوندی حلقوی

درج در ابتدای لیست دوطرفه حلقوی

```
void  CDLinkedList :: InsertFirst( int value)
{
   Dnode  *new = new Dnode (value);

   if (!first)                    // no node
        {  first = new;
           first → next = first;
           first → pre = first;
        }

   new → next = first;
   new → pre = first → pre;
   (first→ pre) → next = new;
   first → pre = new;
   first = new;
}
```

حذف از لیست دوپیوندی حلقوی

- حذف از وسط یا انتهای لیست دوپیوندی حلقوی همانند لیست دوپیوندی ساده است.
- حذف از ابتدای لیست دوپیوندی حلقوی:




```
temp = first;
first = first → next;
first → pre = temp → pre;
(temp → pre ) → next = first;
delete  temp;
```

حذف از لیست دوپیوندی حلقوی

حذف از ابتدای لیست دوطرفه حلقوی

```
void  DLinkedList :: DeleteFirst ( )
{
        if ( (first → next) == first)    //one node
        {
                delete first;
                return;
        }

        Node *temp = first ;
        first = first → next;
        first → pre = temp → pre;
        (temp → pre ) → next = first;
        delete  temp;
}
```

سوال

- تابعی بنویسید که عناصر لیست دوطرفه ساده را معکوس کند.
- تابعی بنویسید که عناصر لیست دوطرفه حلقوی را معکوس کند.
- اگر بخواهیم ماتریس اسپارس را با استفاده از لیست پیوندی پیاده سازی کنیم، ساختار هر گره و ماتریس اسپارس حاصل چگونه خواهد بود.

پایان مبحث

لیست پیوندی