لیست های پیوندی

آرایه

❖ خاصیت نمایش عناصر با آرایه این بود که عناصر پشت سرهم داده مقصود با فواصل ثابت و معینی ذخیره می شدند.

❖اشکالات کار با آرایه

۱- حذف و درج عناصر در آرایه ها بسیار وقت گیر است.

۲- وقتی دارای چندین لیست مرتب شده با طول های متفاوت هستیم. باذخیره کردن هر لیست در آرایه ای با حداکثر اندازه ، حافظه هدر می رود.

۳- با نگهداری لیستها در یک آرایه ، انتقال حجم زیادی از داده ها لازم است.

لیست های پیوندی

❖ راه حل مناسب برای حل مشکل شیفت داده ها در نگاشت ترتیبی ، استفاده از لیست پیوندی می باشد.

❖ بر خلاف نمایش ترتیبی که عناصر در فواصل ثابتی از هم قرار می گرفتند، در لیست پیوندی عناصر می توانند در هر جای حافظه قرار گیرند.

❖ برای دستیابی صحیح به عناصر یک لیست، بایستی به همراه هر عنصر، آدرس یا موقعیت عنصر بعدی نیز ذخیره شود. بنابراین برای هر عنصر ، اشاره گری به عنصر بعدی در لیست وجود دارد.

اشاره گرها

❖ مهمترین عملگرهای استفاده شده با نوع اشاره گر عبارتند از :

- & عملگر آدرس
- * عملگر غیررجوعی(indirection or dereferencing)

❖ **مثال:**

```
int  i ،* pi ;
```

آنگاه i یک متغیر صحیح و pi یک اشاره گر به یک متغیر صحیح است.

```
pi = &i ;
```

&i آدرس i را برگشت و به ان مقدار pi را نسبت می دهد.

اشاره گرها

❖ در هنگام برنامه نویسی به زبان C بهتر است که تمام اشاره گرهایی را که عملا به جایی اشاره نمی کنند را برابر NULL قرار دهیم. این عمل از تلاش برای دستیابی به قسمتی از حافظه که خارج از دامنه برنامه شما بوده و یا شامل اشاره گری به یک داده مقصود قابل دسترسی نیست ، جلوگیری می کند.

اشاره گرها

- هر زمان که نیاز به حافظه جدیدی باشد ، می توان تابعی به نام malloc را فراخوانی و مقدار فضای لازم را درخواست کرد. اگر حافظه لازم وجود داشته باشد ، اشاره گری به ابتدای ناحیه حافظه مورد نیاز برگردانده می شود.

- زمانی که دیگر نیازی به آن حافظه نباشد ، می توان آن را با فراخوانی تابعی به نام free ، آزاد نمود.

اشاره گرها

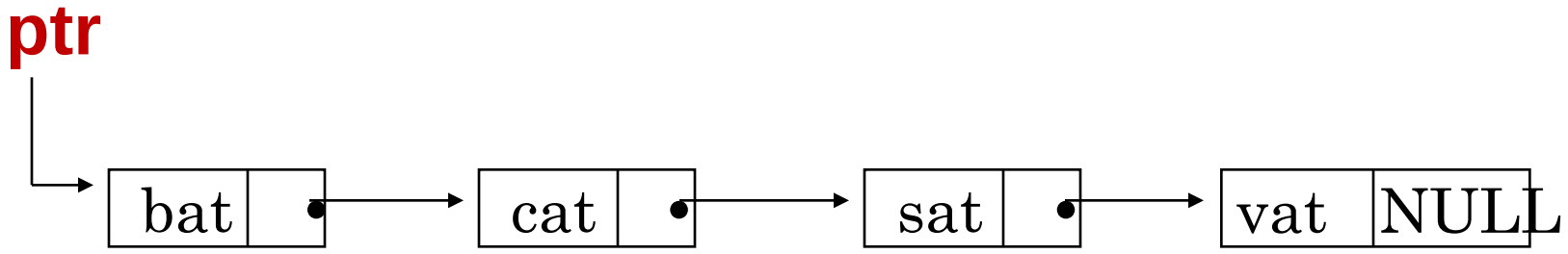
```
int i, *pi;
float f, *pf;
pi = (int *) malloc(sizeof(int));
pf = (float *) malloc (sizeof(float));
*pi =1024;
*pf =3.14;
printf(”an integer = %d, a float = %f\n”, *pi, *pf);
free(pi);
free(pf);
```

اختصاص حافظه

آزاد کردن حافظه

لیست های تک پیوندی

❖ لیست های پیوندی معمولا به وسیله گره هایی متوالی با اتصالاتی که به صورت فلش هایی نشان داده شده اند ارایه می گردند. از نام اشاره گر به اولین عنصر لیست به عنوان نام کل لیست استفاده می شود. بنابراین لیست شکل زیر ptr نامیده می شود.

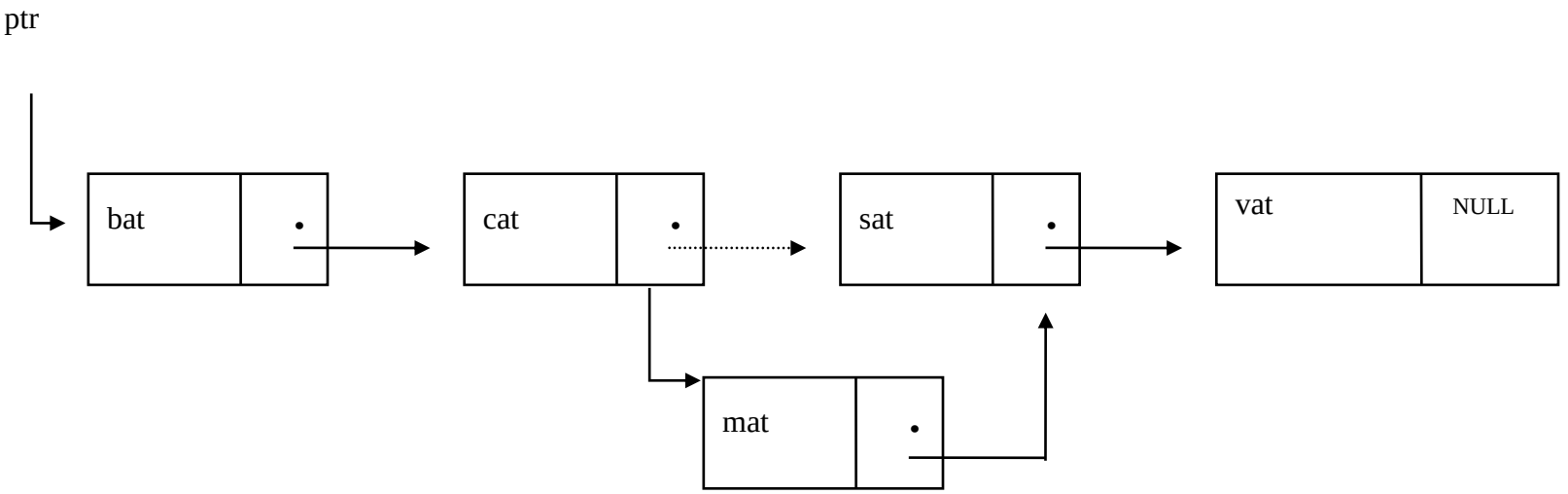


❖ **نکات:**

- گره ها واقعا در مکانهای پشت سر هم حافظه قرار نمی گیرند
- موقعیت گره ها در اجراهای مختلف می تواند تغییر کند

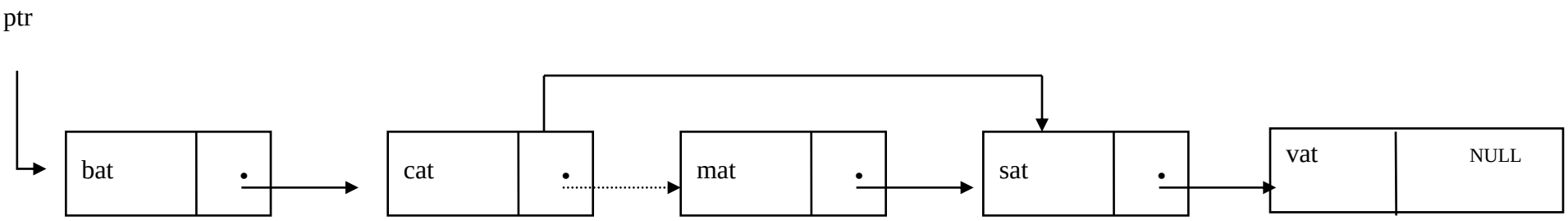
لیست های تک پیوندی

❖ اضافه کردن گره mat بعد از گره cat



لیست های تک پیوندی

❖ حذف گره mat



امکانات لازم برای ایجاد لیست های تک پیوندی

- مکانیزمی برای تعریف ساختار گره ها یعنی فیلدهایی که در گره وجود دارد
- روشی برای ایجاد و اضافه کردن گره هایی که مورد نیاز می باشد
- روشی برای آزاد کردن گره هایی که دیگر مورد نیاز نمی باشد

امکانات لازم برای ایجاد لیست های تک پیوندی

❖ تعریفات لازم

```
typedef struct list_node* list_pointer;
typedef struct list_node {
        char data [4];
        list_pointer link;
        };
```

Creation

```
list_pointer ptr =NULL;
```

Testing

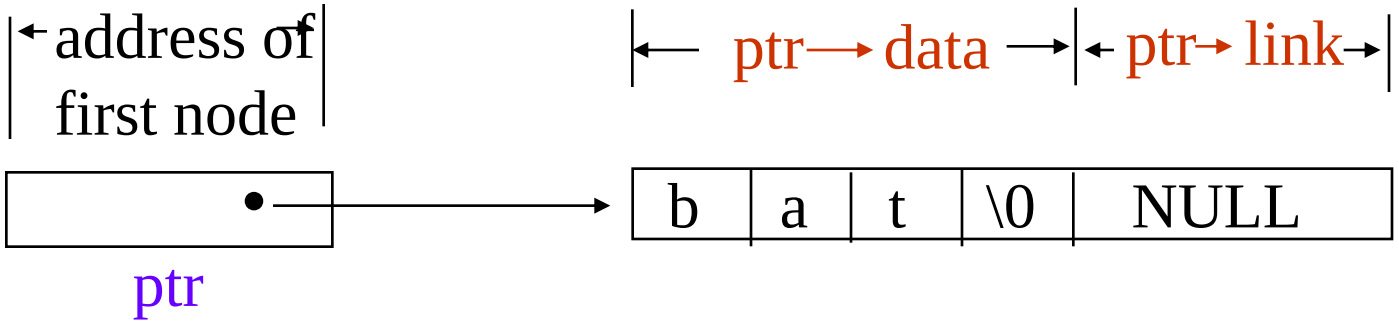
```
#define IS_EMPTY(ptr) (!(ptr))
```

Allocation

```
ptr=(list_pointer) malloc (sizeof(list_node));
```

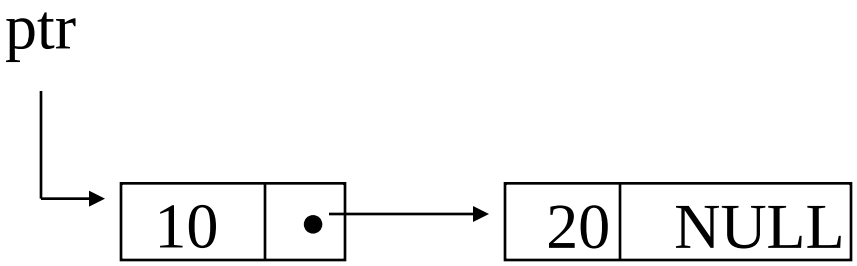
امکانات لازم برای ایجاد لیست های تک پیوندی

```
ptr -> data ⇨ (*ptr).data
strcpy(ptr -> data, “bat”);
ptr -> link = NULL;
```



لیست های تک پیوندی

❖ ایجاد لیست تک پیوندی با دو گره

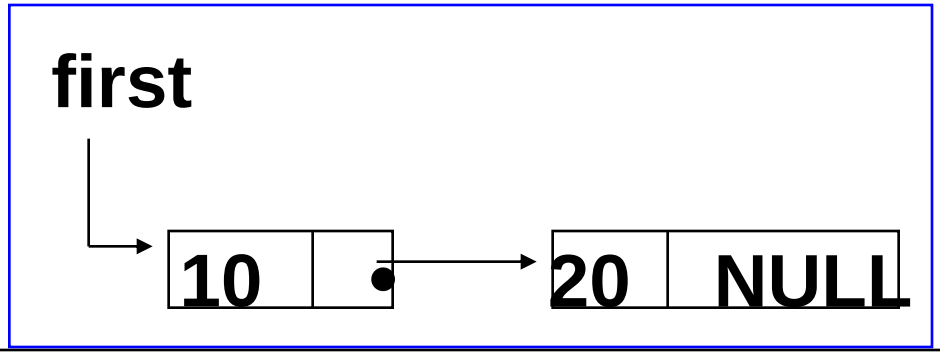



```
typedef struct list_node *list_pointer;
typedef struct list_node {
        int data;
        list_pointer link;
        };
list_pointer ptr =NULL
```

لیست های تک پیوندی

❖ ایجاد لیست تک پیوندی با دو گره

```
list_pointer create2( )
{
/* create a linked list with two nodes */
   list_pointer first, second;
   first = (list_pointer) malloc(sizeof(list_node));
   second = ( list_pointer) malloc(sizeof(list_node));
   second -> link = NULL;
   second -> data = 20;
   first -> data = 10;
   first ->link = second;
   return first;
}
```



لیست های تک پیوندی

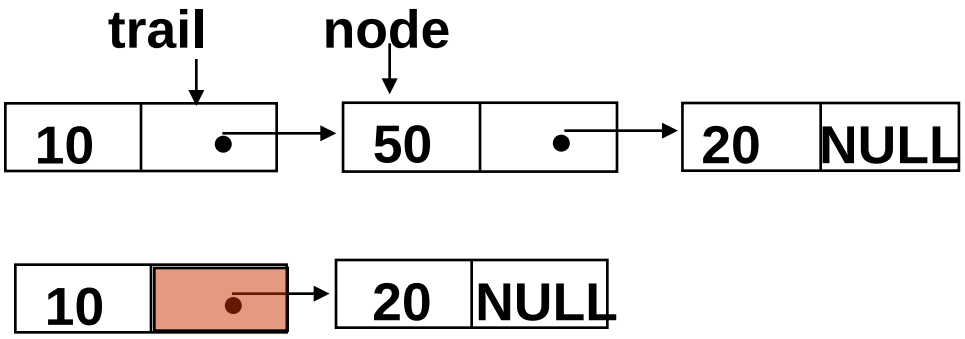
❖ تابع اضافه کردن یک گره با فیلد داده ۵۰ بعد از گره node در لیست پیوندی با نام ptr

```
void insert(list_pointer *ptr, list_pointer node)
{  list_pointer temp;
   temp = (list_pointer) malloc(sizeof(list_node));
   if (IS_FULL(temp))
    {
       fprintf(stderr, “The memory is full\n”);
       exit (1);
      }
   temp->data = 50;
   if (*ptr) { // nonempty list
       temp->link =node ->link;
       node->link = temp;
  }
  else {    //empty list
      temp->link = NULL;
      *ptr =temp;
   }
}
```

لیست های تک پیوندی

❖ تابع حذف گره node از لیست پیوندی با نام ptr

```
void delete(list_pointer *ptr, list_pointer trail,  list_pointer node)
{
/* delete node from the list, trail is the preceding node
   ptr is the head of the list */
    if (trail)
       trail->link = node->link;
    else
       *ptr = (*ptr) ->link;
     free(node);
}
```



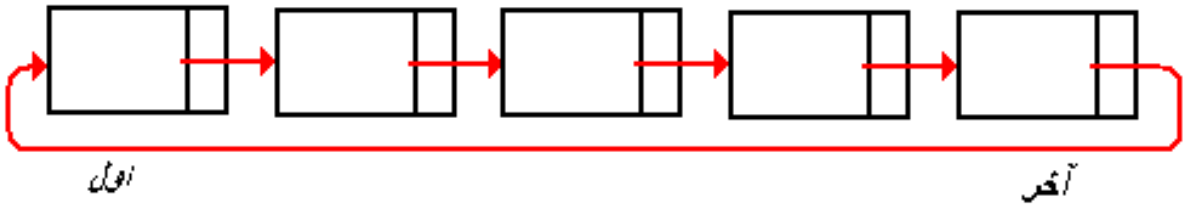
لیست های تک پیوندی

❖ تابع چاپ عناصر یک لیست تک پیوندی با نام ptr

```
Void  print_ list (list_pointer   ptr)
{
        print f (" The list contains: " );
        for  ( ; ptr ; ptr = ptr -> link)
                print f (" %4d "  ،ptr ->data);
        print f (" \ n " );
}
```

لیست های تک پیوندی حلقوی

- اگر اشاره گر خارج شده از آخرین گره به جای Null به گره ابتدایی اشاره کند ساختار جدیدی به نام لیست پیوندی حلقوی پدید می آید.



- در لیست های پیوندی حلقوی به خاطر بسته بودن لیست می توان از هر نقطه لیست به نقاط دیگر دسترسی پیدا کرد.

لیست های تک پیوندی حلقوی

❖ تابع اضافه کردن گره node به ابتدای لیست تک پیوندی حلقوی با نام ptr

```
void insert(list_pointer *ptr,list_pointer node)
{ list-pointer temp;
  if(*ptr)
    { temp=*ptr;
      while(temp->link!=*ptr)
              temp=temp->link;
      temp->link=*ptr;
      *ptr= node;
    }
 else
    {
     node->link=node;
      *ptr=node;
     }
}
```

لیست های تک پیوندی حلقوی

❖ تابع حذف یک گره از ابتدای لیست تک پیوندی حلقوی با نام ptr

```
void delete(list_pointer *ptr)
{ list-pointer temp,node;
  temp=node=*ptr;
  while(temp->link!=*ptr)
            temp=temp->link;
  temp->link=ptr->link;
  *ptr=(*ptr)->link;
  }
```

لیست های دو پیوندی

❖ لیست های تک پیوندی تنها بعضی از مشکلات را حل می کند چرا که فقط به راحتی می توانیم در جهت پیوندها حرکت کنیم.

❖ **مثال:**

فرض کنید که به یک گره مشخص مانند ptr ، اشاره کنیم و بخواهیم گره قبل از ptr را پیدا کنیم ، تنها راه یافتن گره ماقبل ptr پیمایش از ابتدای لیست می باشد.

لیست های دو پیوندی

❖ یک گره در یک لیست پیوندی دوگانه حداقل دارای سه فیلد می باشد :

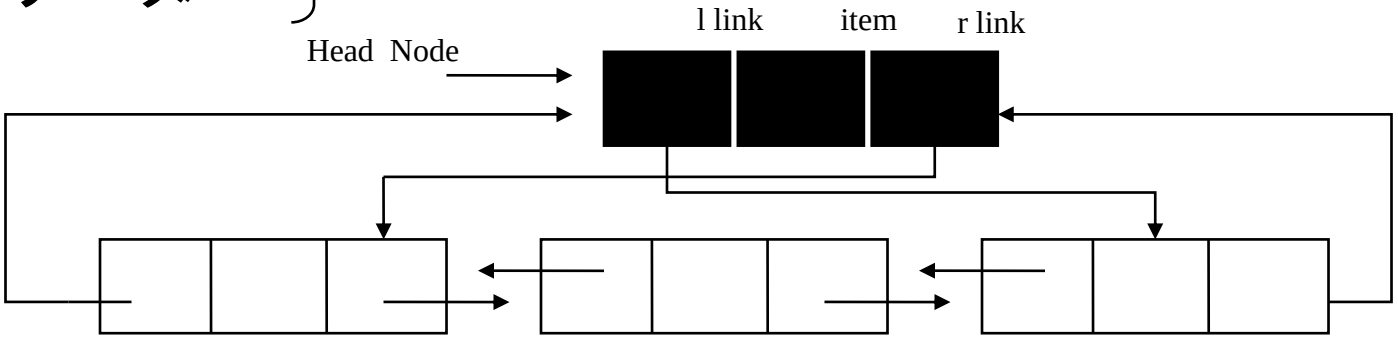
- فیلد data
- فیلد llink (اشاره گر به چپ)
- فیلد rlink (اشاره گر به راست)

```
typedef struct node *node_pointer;
typedef struct node {
   node_pointer llink;
   element item;
   node_pointer rlink;
}
```

لیست های دو پیوندی

یک لیست پیوندی دوگانه به دو صورت است :

- حلقوی
- غیر حلقوی



لیست پیوندی دوگانه حلقوی با گره head

درج و حذف از یک لیست پیوندی دوگانه بسیار ساده می باشد.

لیست های دو پیوندی

❖ تابع اضافه کردن گره newnode بعد از گره node در لیست دو پیوندی با نام ptr

```
Void  dinsert ( node_pointer  node , node_pointer  newnode )
{
/* insert  newnode  to  the  right  of  node */
        newnode-> llink = node ;
        newnode->rlink = node-> r link ;
        node -> r link -> l link = newnode;
        node -> r link = newnode ;
}
```

لیست های دو پیوندی

❖ تابع حذف گره node از لیست دو پیوندی با نام ptr

```
Void  delete ( node_pointer  ptr ،node_pointer  node)
{
/* delete from  the  doubly  linked  list  */
    if  (ptr= = node)
        print f (" Deletion  of  head  node  not  permitted  .\ n " )
    else {
        node-> l link  - > r link = node- > r link ;
        node-> r link  - > l link = node- > l link ;
        free ( node) ;
    }
}
```