中南大學
CENTRAL SOUTH UNIVERSITY

第十七讲

指针的高级应用

动 态 存 储 分 配

内存分配函数

```
#include <stdlib.h>
```

malloc函数：分配内存块,不对内存块进行初始化

```
void *malloc(size_t  size);
```

calloc函数：分配内存块,并且对内存块进行清零

```
void *calloc(size_t  nmenb, size_t  size);
```

realloc函数：调整先前分配的内存块大小

```
void *realloc(void *ptr, size_t  size);
```

free函数：释放不需要的内存

```
void free(void *ptr);
```

返回 **void *** 类型的值

空指针：**NULL**

```
p = malloc(10000);
if (p == NULL) {
    /* allocation failed;*/
}
```

```
if (p == NULL) …
if (p != NULL) …
```

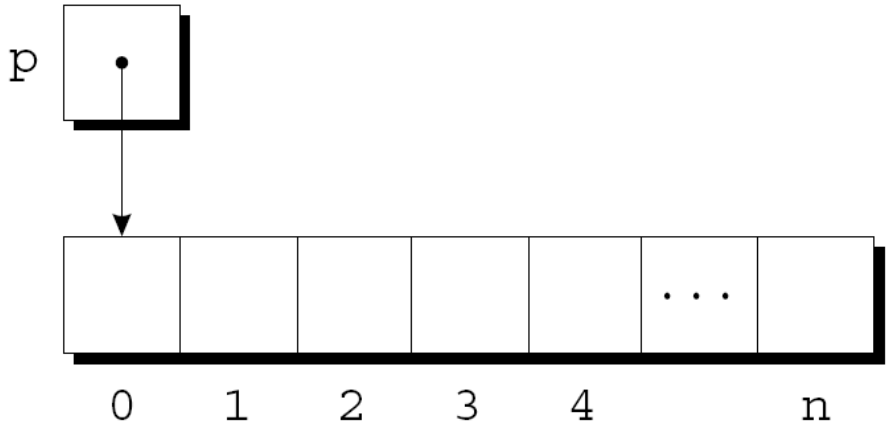
```
if (!p) …
if (p) …
```

动 态 分 配 字 符 串

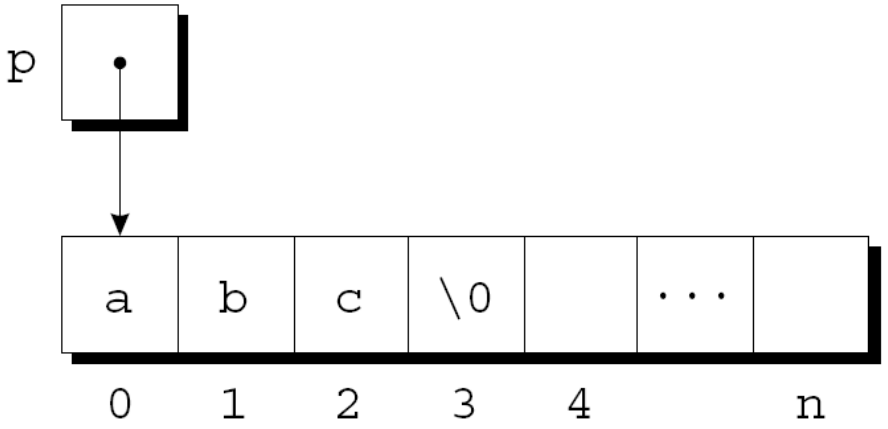
使用malloc为字符串分配内存

```
char *p;
p = malloc(n + 1);
```

```
p = (char *) malloc(n + 1);
```




```
strcpy(p, "abc");
```



动 态 分 配 字 符 串

连接两个字符串的函数concat但不改变其中任何一个字符串

```
char *concat(const char *s1, const char *s2)
{
  char *result;

  result = malloc(strlen(s1) + strlen(s2) + 1);
  if (result == NULL) {
    printf("Error: malloc failed in concat\n");
    exit(EXIT_FAILURE);
  }
  strcpy(result, s1);
  strcat(result, s2);
  return result;
}
```

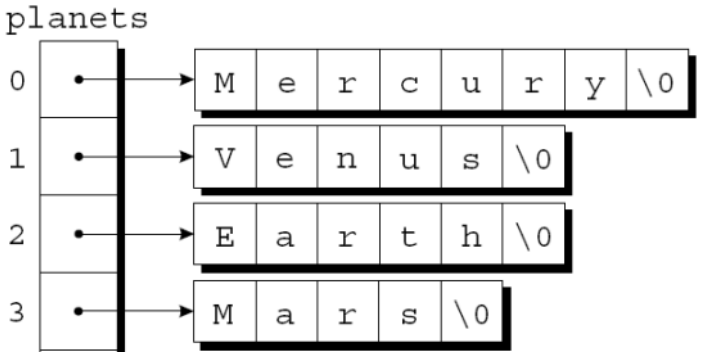
```
p = concat("abc", "def");
```

动 态 分 配 字 符 串

二维数组的行如果固定长度，可能浪费空间

	0	1	2	3	4	5	6	7
0	M	e	r	c	u	r	y	\0
1	V	e	n	u	s	\0	\0	\0
2	E	a	r	t	h	\0	\0	\0
3	M	a	r	s	\0	\0	\0	\0

指针数组



显示一个月的提醒列表　　01-remind2.c

动 态 分 配 数 组

为数组分配空间

```
int *a;

a = malloc(n * sizeof(int));
```

```
for (i = 0; i < n; i++)
  a[i] = 0;
```

void *calloc(size_t nmemb, size_t size);

void *calloc(n, size)分配n个单元，每个单元占size个字节的内存块

```
a = calloc(n, sizeof(int));

struct point { int x, y; } *p;

p = calloc(1, sizeof(struct point));
```

动 态 分 配 数 组

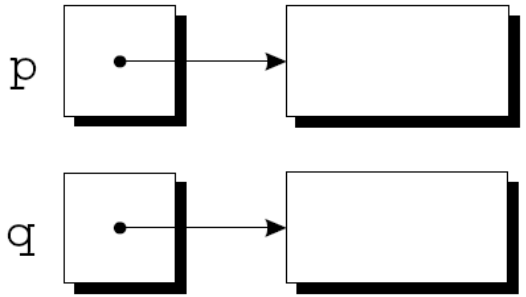
调整空间

```
void *realloc(void *ptr, size_t size);
```

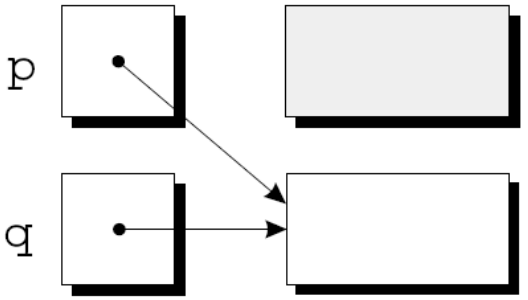
- ptr来自于先前malloc,calloc和realloc函数调用。
- realloc函数不会初始化新添加的内存块。
- realloc函数返回空指针时不会改变原有内存块数据。
- realloc(NULL,size)等价于malloc(size)。
- realloc(ptr,0)→会释放掉ptr指向的内存块。
- realloc函数返回后建议更新指向内存块的所有指针。

释 放 存 储 空 间

```
p = malloc(…);
q = malloc(…);
```




```
p = q;
```




```
void free(void *ptr);
```

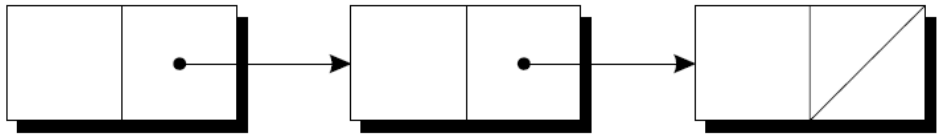
```
p = malloc(…);
q = malloc(…);
free(p);
p = q;
```

```
char *p = malloc(4);
…
free(p);
…
strcpy(p, "abc");    /*** WRONG ***/
```

悬空指针

链　表

链表　　一串结点组成　　每个结点包含指向下一个结点的指针



插入、删除容易　　不可随机访问

```
struct node {
  int value;           /* data stored in the node  */
  struct node *next;   /* pointer to the next node */
};
```

```
struct node *first = NULL;
```

链　表

逐个创建节点

分配内存单元　　**存储数据**　　**插入链表**

```
struct node *new_node;
new_node = malloc(sizeof(struct node));
```

```
(*new_node).value = 10;
```

-> 运算符（右箭头选择）

```
new_node->value = 10;
```

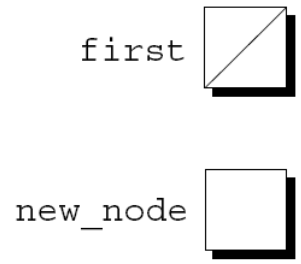
```
scanf("%d", &new_node->value);
```

链　表

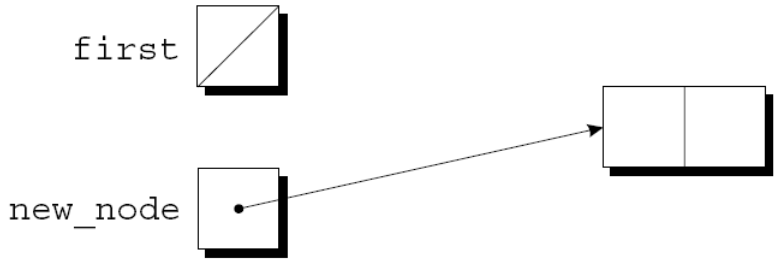
插入成为首结点

```
new_node->next = first;
first = new_node;
```

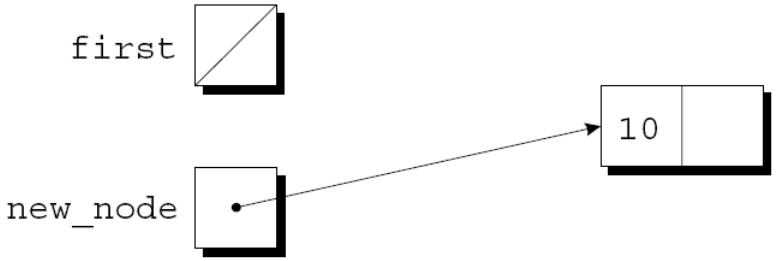
```
first = NULL;
```



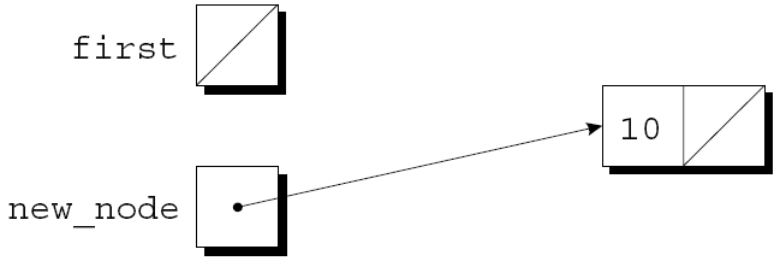

```
new_node =malloc(sizeof(struct node));
```



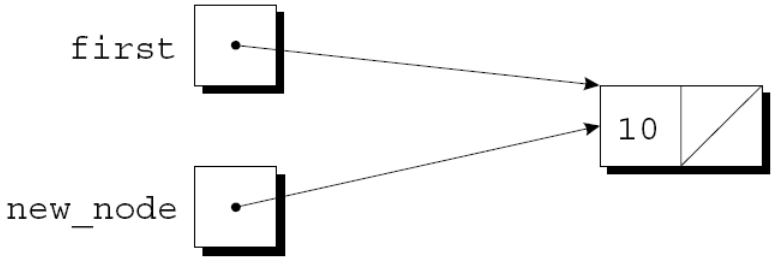

```
new_node->value = 10;
```



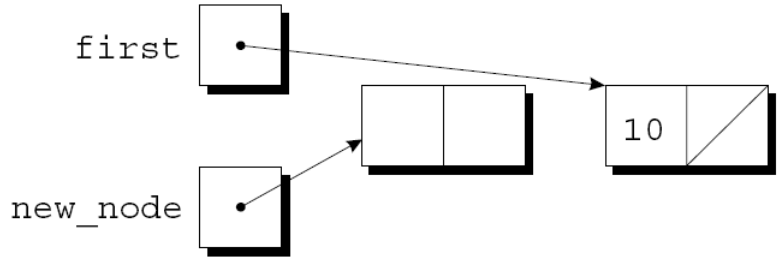

```
new_node->next = first;
```



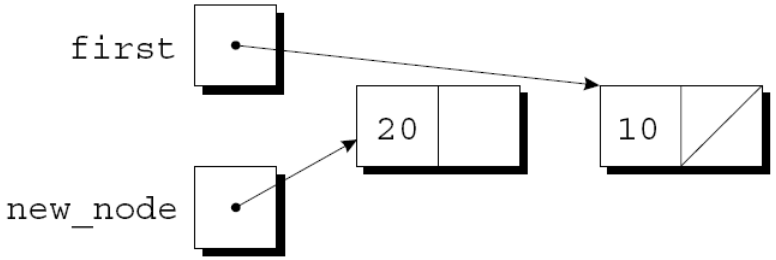

```
first = new_node;
```



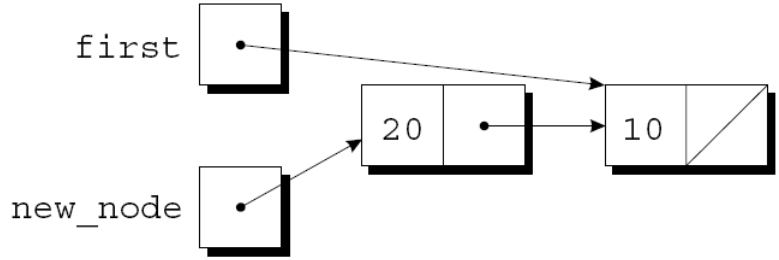

```
new_node =malloc(sizeof(struct node))
```



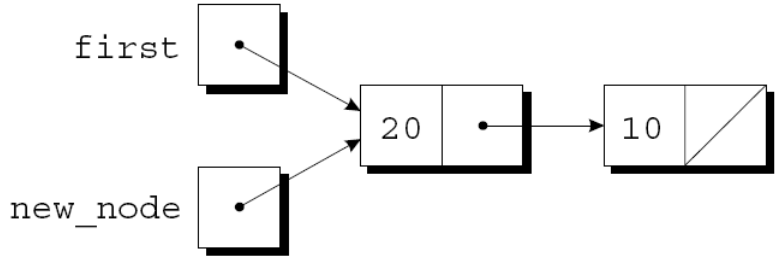

```
new_node->value = 20;
```




```
new_node->next = first;
```




```
first = new_node;
```



插入结点

```
struct node *add_to_list(struct node *list, int n)
{
  struct node *new_node;

  new_node = malloc(sizeof(struct node));
  if (new_node == NULL) {
    printf("Error: malloc failed in add_to_list\n");
    exit(EXIT_FAILURE);
  }
  new_node->value = n;
  new_node->next = list;
  return new_node;
}
```

```
first = add_to_list(first, 10);
first = add_to_list(first, 20);
```

插入结点

```
struct node *read_numbers(void)
{
  struct node *first = NULL;
  int n;

  printf("Enter a series of integers (0 to terminate): ");
  for (;;) {
    scanf("%d", &n);
    if (n == 0)
      return first;
    first = add_to_list(first, n);
  }
}
```

搜索链表

```
for (p = first; p != NULL; p = p->next)
  …
```

```
struct node *search_list(struct node *list, int n)
{
  struct node *p;

  for (p = list; p != NULL; p = p->next)
    if (p->value == n)
      return p;
  return NULL;
}
```

删除结点

定位　　改变前一结点指针，使其绕过删除结点　　释放内存空间

```
struct node *delete_from_list(struct node *list, int n)
 {
   struct node *cur, *prev;

   for (cur = list, prev = NULL;
        cur != NULL && cur->value != n;
        prev = cur, cur = cur->next)
     ;
   if (cur == NULL)
     return list;              /* n was not found */
   if (prev == NULL)
     list = list->next;        /* n is in the first node */
   else
     prev->next = cur->next;   /* n is in some other node */
   free(cur);
   return list;
 }
```

有序链表

维护零件库　　02-inventory2.c