1 指针

- 地址常量
- 指针数据类型
- 指针类型的变量

- 2' 传值调用，效率不高
- **2 指针典型用法一：指针作为函数的参数-传址调用，提高效率**
  - **2.1 传址调用产生的副作用及其避免**
    - 2.1.1 const的限制作用
  - 2.2' 传值调用不能通过参数返回数据，return不能返回多个数据
  - **2.2 传址调用产生的副作用及用指针型参数来“返回”多个数据**
- 3' 用指针操纵数组—不重要，但有助于学习动态数组
- **3 指针典型用法二：操纵动态变量或动态数组**
  - 3.1 动态变量的撤销
  - 3.2 内存泄露与悬浮指针

- **4 指针类型返回值：一般用来返回一组数据**
  - 链表
  - 字符串
    - 常用字符串处理库函数
    - 字符地址的输出
- **5 指针数组**
  - 5.1 带形参的 main 函数*
- **6 结构数组**

7 链表

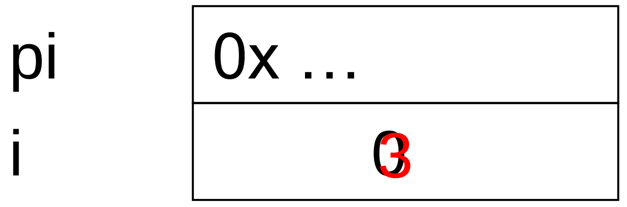
- 7.1 头插创建
  - 头插一个节点
  - 尾部追加一个节点
- 7.2 第i个节点后插入一个节点
- 7.3 第i个节点前插入一个节点
- 7.4 关键值节点之前插入一个节点
- 7.5 集合问题的完整程序
  - 含交、差、并集求解函数
  - 含尾部追加建立链表、输出链表、撤销链表函数
  - 含main函数
- 7.6 引用及其运用
  - 指针型参数的传值调用（没有取值操作）
  - 指针型参数的传址调用（二级指针）
  - 引用型参数（其实是传址调用）

1 指针

类型关键字

- 定义一个指针变量，并初始化

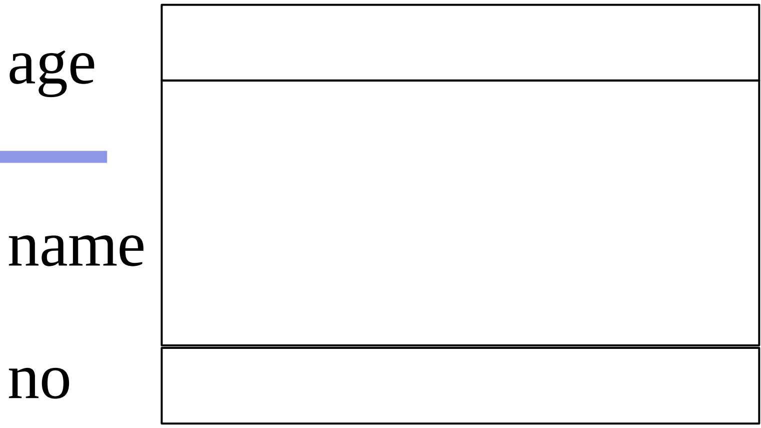
```
int i = 0;
int  *pi = &i;
*pi = 3;
```



2' 结构变量作为函数的参数-传值调用，效率不高

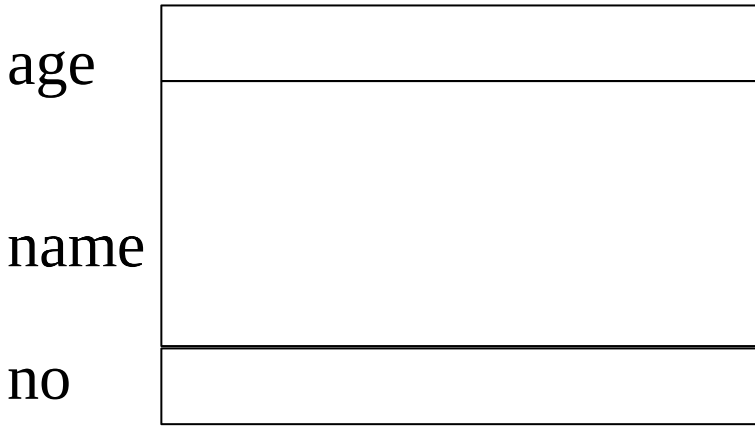
```
struct Stu
{     int no;
      char name[20];
      int age;
};
```

```
int main()
{
     Stu s;
     cin >> s.no >> s.name >> s.age;
     f(s);
     return 0;
}
```



```
Stu t = s;
```

```
void f(Stu t)
{
     cout << t.no;
     cout << t.name;
     cout << t.age;
}
```



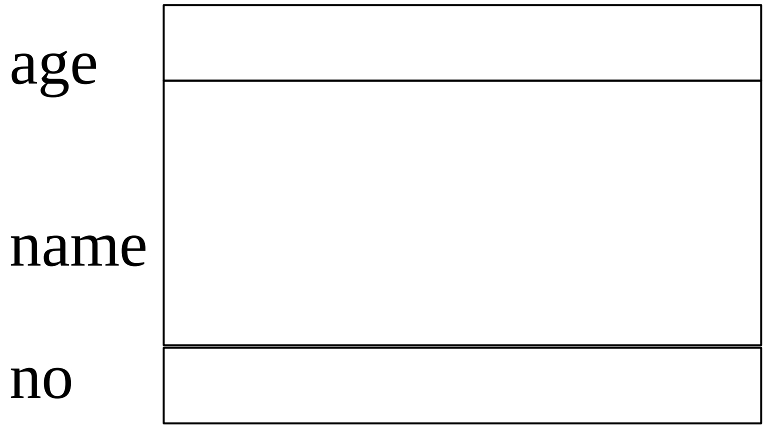
2 指针典型用法一：指针作为函数的参数-传址调用，提高效率

```
struct Stu
{       int no;
        char name[20];
        int age;
};
```

```
char *pc;
int *pi;
double *pd;
```

```
int main()
{
        Stu s;
        cin >> s.no >> s.name >> s.age;
        f(&s);
        return 0;
}
```

```
Stu *p = &s;
```



```
void f(Stu *p)
{
        cout << p -> no;   //(*p).no
        cout << p -> name;//(*p).name
        cout << p -> age; //(*p).age
}
```

2.1 传址调用产生的副作用及其避免

```
struct Stu
{     int no;
      char name[20];
      int age;
};
```

```
int main()
{
      Stu s;
      cin >> s.no >> s.name >> s.age;
      f(&s);
      return 0;
}
```

```
 f(s);
void f(Stu t)
{
      cout << t.no;
      cout << t.name;
      cout << t.age;
      t.no++;
}
```

```
 void f(const Stu *p)
 void f(Stu *p)
 {
      cout << p -> no;   //(*p).no
      cout << p -> name;//(*p).name
      cout << p -> age; //(*p).age
```
~~p -> no++;~~
```
 }
```

2.1.1 const的限制作用

必须初始化

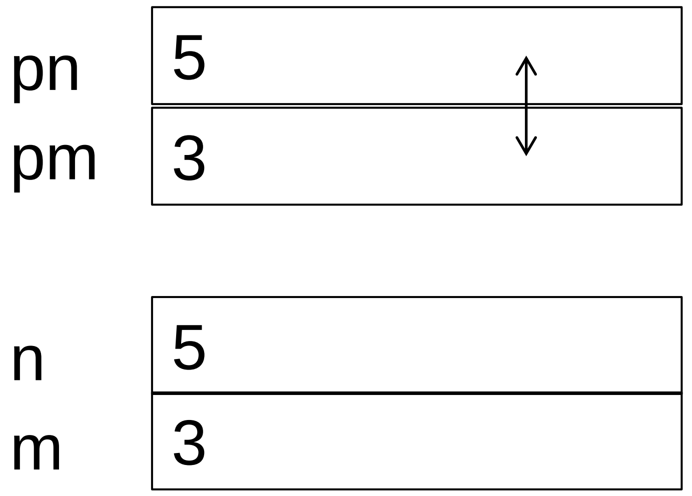
```
int n;
 const int M = 0;       //M 是常量
 const int *p1;         //*p1 是常量， 不过 p1 = &n 也可以


 int * const P2 = &n; // P2是常量

 const int * const P3 = &M;
  //*P3， P3是常量， 不过 const int * const P3 = &n; 也可以
```

2.2' 传值调用不能通过参数返回数据，**return**不能返回多个数据

无法交换m、n

```
void MySwap(int pm, int pn)
{
      int temp = pm;
      pm = pn;
      pn = temp;
}
```

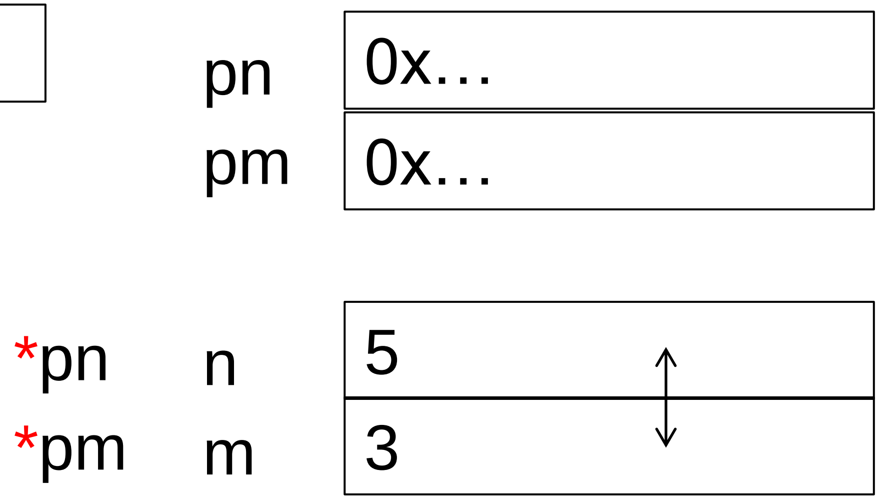
```
int m = 3;
int n = 5;
MySwap(m, n);
```



2.2 传址调用产生的副作用及用指针型参数来“返回”多个数据

可以交换m、n

```
void MySwap(int *pm, int *pn)
{
      int temp = *pm;
      *pm = *pn;
      *pn = temp;
}
```



```
int m = 3;
int n = 5;
MySwap(&m, &n);
```

3′ 用指针操纵数组—不重要，但有助于学习动态数组

用独立函数实现

```
#define N 4
int main( )
{     int a[N];
      for (int i=0; i<N; ++i)
            cin >> a[i];

      BubbleSort(a, N);

      …
}
```

```
void BubbleSort(int pa[ ], int count)
 {
     for(int i = 0; i < count-1; ++i)
           for(int j = 0; j < count-1-i; ++j)
                 if(pa[j] > pa[j+1])
                 {
                       int temp = pa[j];
                       pa[j] = pa[j+1];
                       pa[j+1] = temp;
                 }
}
```

第九次作业-用指针实现

```
#define N 4
int main( )
{     int a[N];
      for (int i=0; i<N; ++i)
            cin >> a[i];

      BubbleSort(a, N);

      …
}
```

一般不必用指针操纵数组

```
int *pa = a;
```

```
void BubbleSort(int *pa, int count)
 {
     for(int i = 0; i < count-1; ++i)
          for(int j = 0; j < count-1-i; ++j)
               if(pa[j] > pa[j+1])
               {
                    int temp = pa[j];
                    pa[j] = pa[j+1];
                    pa[j+1] = temp;
               }
}
```

第九次作业-用指针实现

一般不必用指针操纵数组

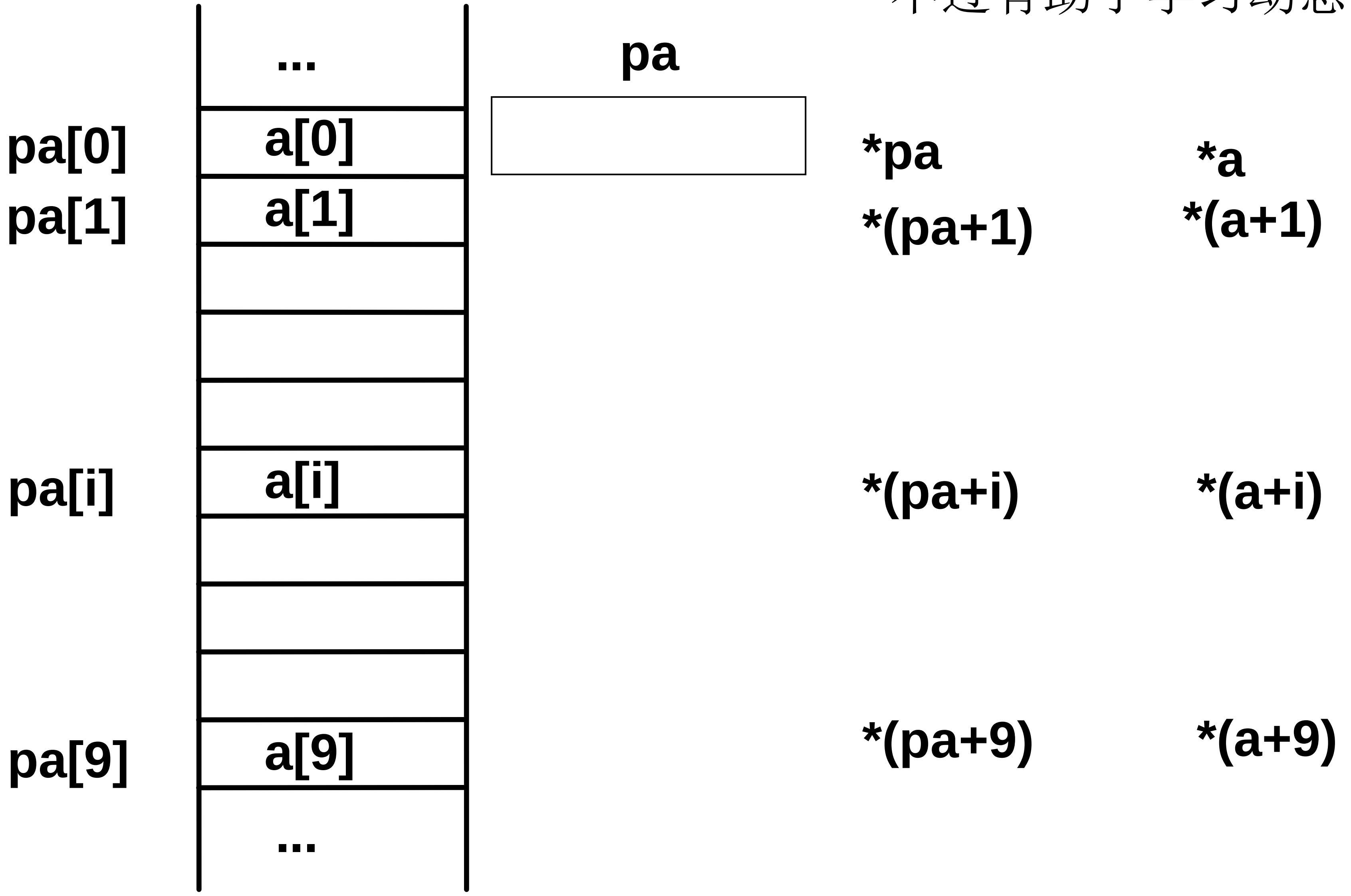
```
#define N 4
int main( )
{     int a[N];
      for (int i=0; i<N; ++i)
            cin >> a[i];

      BubbleSort(a, N);

      …
}
```

```
int *pa = a; //int *pa = &a[0];
```

```
void BubbleSort(int *pa, int count)
 {
     for(int i = 0; i < count-1; ++i)
          for(int j = 0; j < count-1-i; ++j)
               if(*(pa+j) > *(pa+j+1))
               {
                    int temp = *(pa+j);
                    *(pa+j) = *(pa+j+1);
                    *(pa+j+1) = temp;
               }
 }
```

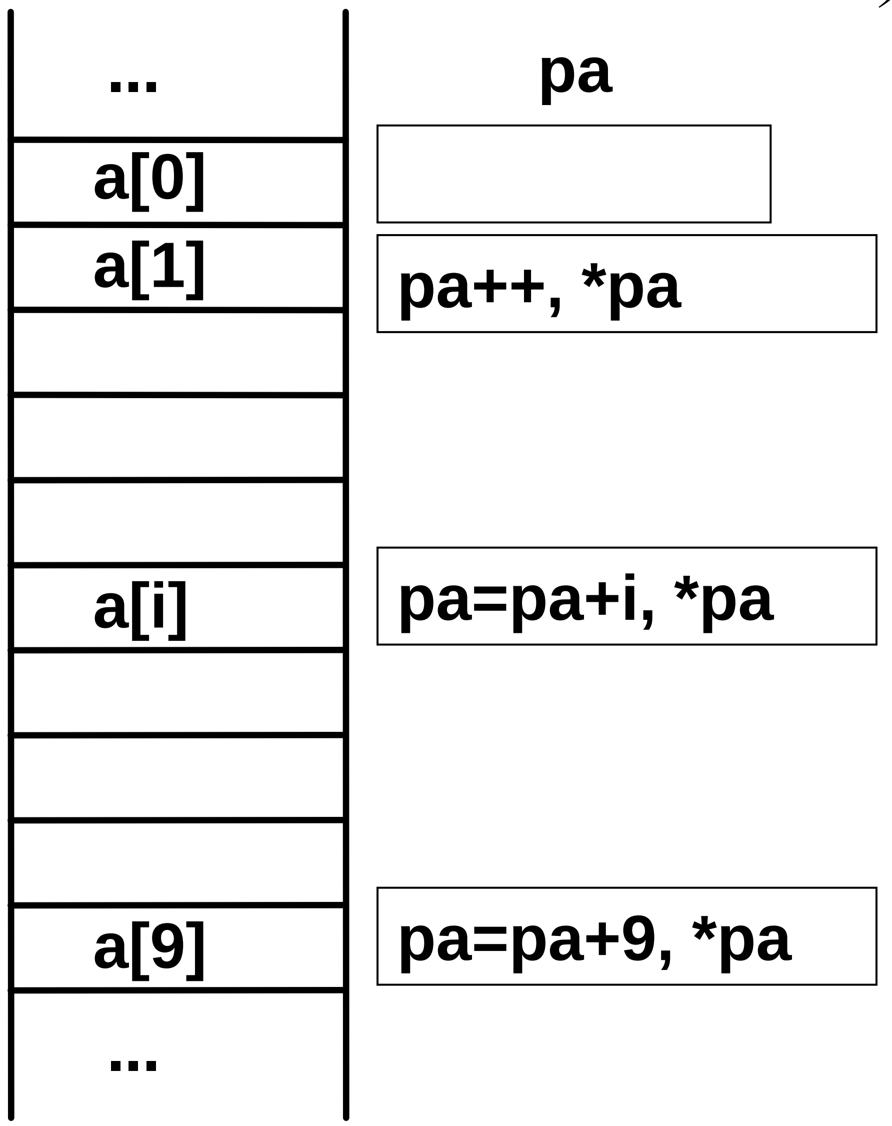
a

不过有助于学习动态数组



a

不过有助于学习动态数组



二维数组的指针*

不过有助于学习二维动态数组

```
int b[5][10];
int *p;
p = &b[0][0];//或“p = b[0];”
```

第一行某个元素 p[j]

指向元素

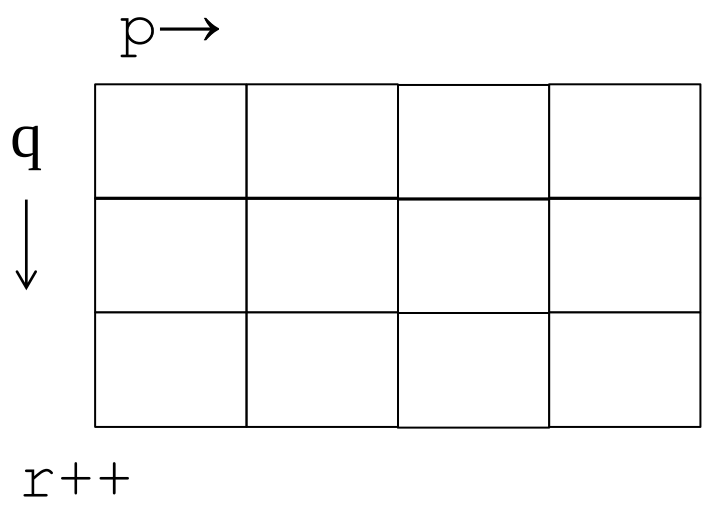
```
int (*q)[10];
q = &b[0];   //或“q = b;”
```

某个元素 q[i][j]

指向行

```
int (*r)[5][10];
r = &b;
```

指向片



3 指针典型用法二：操纵动态变量或动态数组

```
int *pd = new int;
*pd = 3;
cout << endl << *pd << endl;

int *pda = new int[5];
for(int i=0; i < 5; ++i, ++pda)
     cin >> *pda;

pda -= 5;
for(int i=0; i < 5; ++i, ++pda)
     cout << *pda << ", ";
```

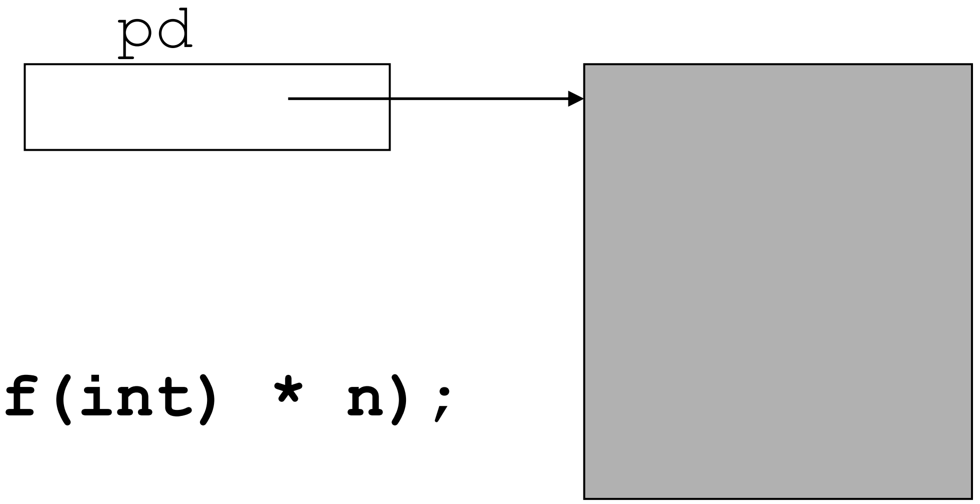
```
for(int i=0; i < 5; ++i)
    printf("%d, ", *pda++);
```

```
int (*pdaa)[10] = new int[n][10];
```

二维动态数组：个别答疑

3.1 动态变量的撤销

```
int *pd = new int;
delete pd;

int *pd = new int[n];
delete []pd;



int *pd = (int *)malloc(sizeof(int) * n);
free(pd);
```



3.2 内存泄露与悬浮指针*

```
int *pda;
int m;
pda = new int[n];
……

pda = &m;
```

pda所指向的动态空间没有释放，但无法访问，泄漏了

pda所指向的动态空间释放了，不知道会分配给谁，
但pda 里存储的还是该动态空间的首地址

```
int *pda;
pda = new int[n];
……
delete []pda;
```

4 指针类型返回值：一般用来返回一组数据

```
Node *InsCreate( )
{
        Node *head = NULL;

        for(int i = 0; i < N; i++)
        {       Node *p = new Node;
                p -> data = i;
                p -> next = head;
                head = p;
        }
        return head;
}
```

```
int main( )
{
        Node *h = InsCreate( );
        PrintList(h);
        ……
```

```
int main( )
{
        Node *h = NULL;
        h = InsCreate( );
        PrintList(h);
        ……
```

4 指针类型返回值：一般用来返回一组数据

```
char *strCpy(char *dst, const char *src)
{
	int i;
	for (i = 0; src[i] != '\0'; ++i)
		dst[i] = src[i];
	 dst[i] = '\0';
	return dst;
}
```

```
strCpy(str, "NJU");
cout << str;

cout << strCpy(str, "NJU");
```

不要返回局部变量的地址：个别答疑

函数指针：个别答疑

4.2.1 常用字符串库函数

```
unsigned int strlen(const char *s);
// int len = strlen(str);

char *strncpy(char *s1, const char *s2, int n);
// char *str = strncpy(str, "nju", 2);

char *strcat(char * s1, const char * s2);
char *strncat(char * s1, const char * s2, int n);
int strcmp(const char *s1, const char *s2);

int strncmp(const char *s1, const char *s2, int n);
//if( strncmp(str, "nju", 2) == 0 )说明 str 前两个字符为 nj
```

sqrt
fabs
pow
rand
srand

新标准下的常用字符串库函数

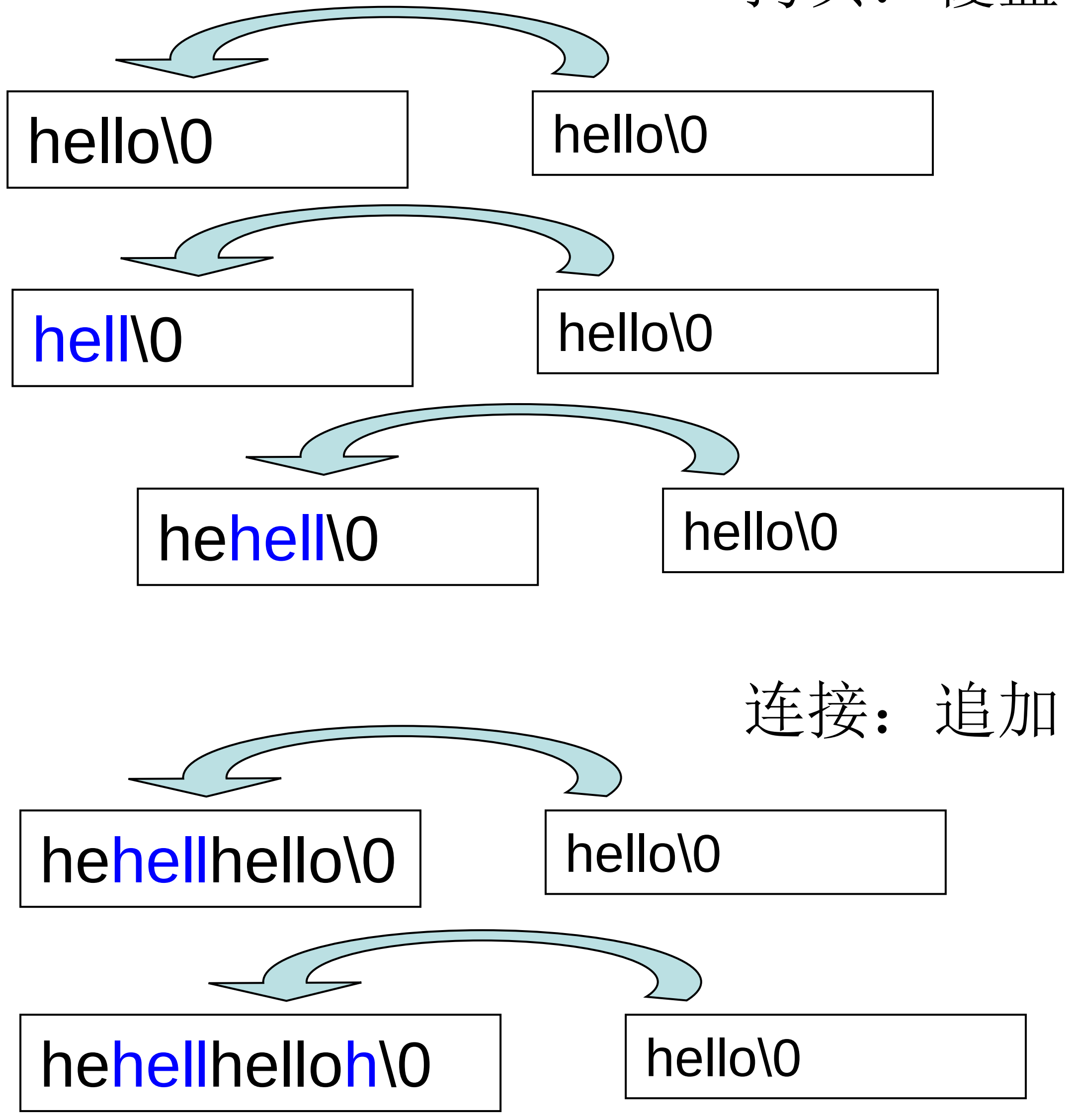
结果字节数（含'\0'）

拷贝：覆盖

```
strcpy_s(dstr, 6,  "hello")

strncpy_s(dstr, 5,  "hello", 4)

strncpy_s(dstr+2, 7,  "hello", 4)
```

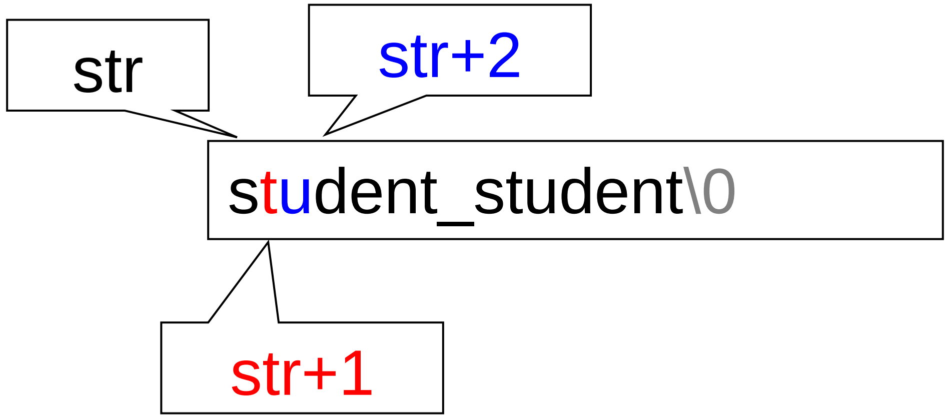
连接：追加

```
strcat_s(dstr, 12,  "hello")

strncat_s(dstr, 8,  "hello", 1)
```



4.2.2 字符型地址，默认输出字符串

```
char str[] = "student_student";
cout << str << endl;  //输出整个数组的字符，直到'\0'
cout << str+1 << endl;
cout << str+2 << endl;
```

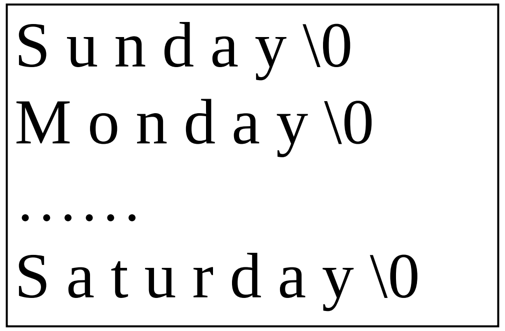


```
char ch = 'c';
char *pc = &ch;
cout << pc << endl; //输出以c开头的乱码字符串
```

5 指针数组

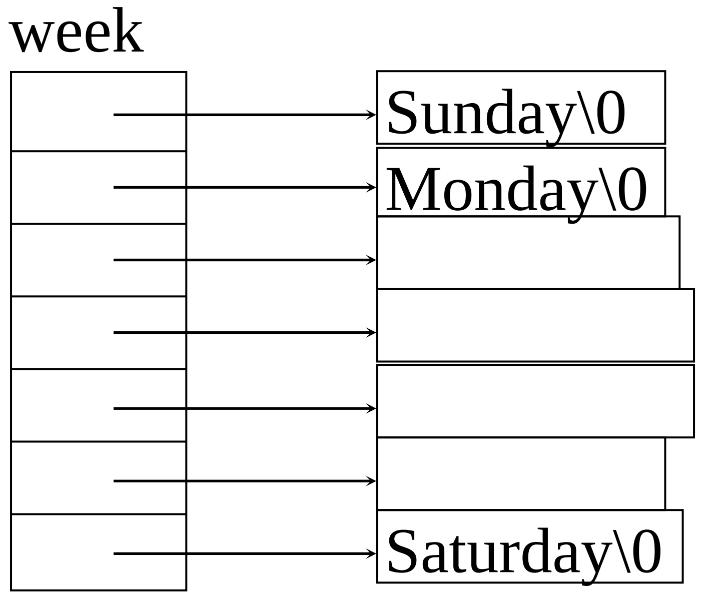
- 二维字符型数组

char weekday[7][10] = {"Sunday", "Monday", "…", "Saturday" };



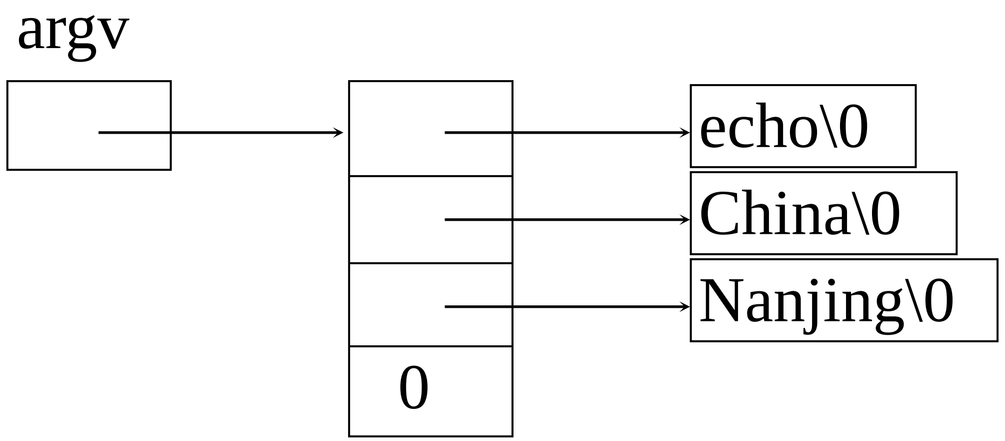
- 字符指针数组

char *week[7] = {"Sunday", "Monday", "…", "Saturday"};



5.1 带形参的 main 函数*

```
#include <stdio.h>
int main(int argc, char *argv[ ])
{
  while(argc > 1)
  {
      ++argv;
      printf("%s \n", *argv);
      --argc;
  }
  return 0;
}
```



6 结构数组

```
#define N 5
enum FeMale {F, M};
struct Stu
{
        int id;
        char name;
        FeMale s;
        int age;
        float score;
};
```

stu_array[5]

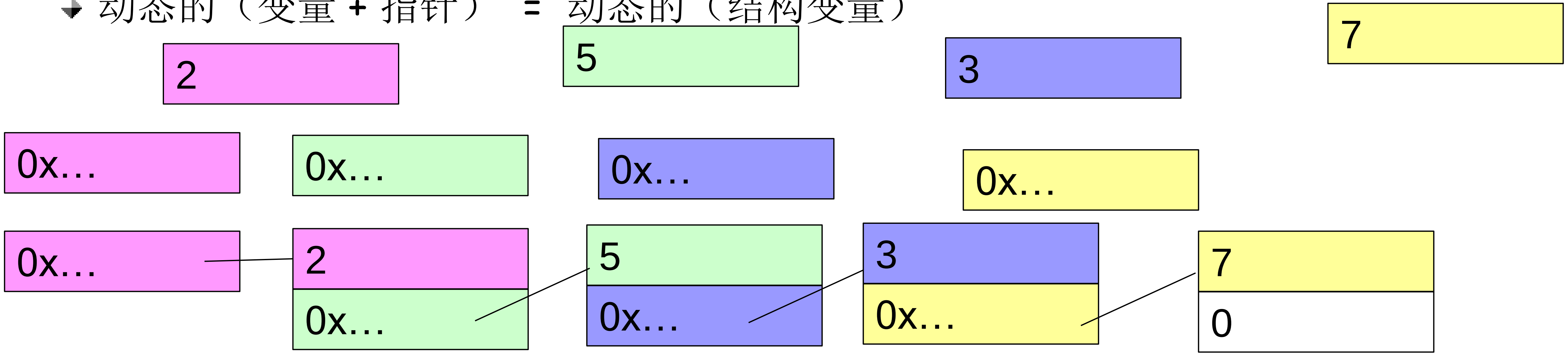
结构变量

一维数组

	num	name	s	age	score
s[0]	1001	T	M	20	90.0
s[1]	1002	K	F	19	89.0
s[2]	1003	M	M	19	95.5
s[3]	1004	J	M	18	100.0
s[4]	1005	L	F	18	81.0

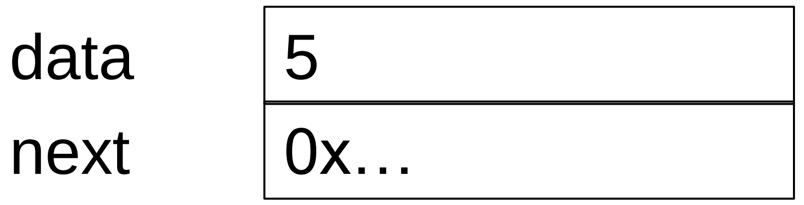
7 链表

- 内存
  - 栈区：存放程序中定义的基本类型变量、数组、指针变量、指针数组、结构变量、结构数组、形式参数…
  - 堆区（零星的空间）：存放程序中创建的单个动态变量、动态数组（多个关联的动态变量）
- 没有足够的连续存储空间时怎么办？
  - 用指针把若干个分散的动态变量链接起来
  - 动态的（变量 + 指针） = 动态的（结构变量）



```
struct Node
{
    int data;
    Node *next;
};
```

```
Node s;
```



```
Node *p;
```

检查:
空链表 (head==NULL)
只有一个节点
对第一个节点进行操作
对最后一个节点进行操作
最后一个节点的next指针应为NULL
操控链表的指针是否已经指向了链表末尾

7.1 创建：空链表头部插入N个结点

```
Node *InsCreate( )
{
    Node *head = NULL;
    for(int i = 0; i < N; ++i)
    {
        Node *p = new Node;
        cin >> p -> data;
        p -> next = head;
        head = p;
    }
    return head;
}
```

7.1.1 已有链表头部插入一个结点

```
Node *InsOneNode(Node *head)
{
    Node *p = new Node;
    cin >> p -> data;
    p -> next = head;
    head = p;

    return head;
}
```

7.1.2 已有链表尾部追加一个结点

```
Node *q = head;
while(q -> next != NULL)
     q = q -> next;
```

```
Node *AppOneNode(Node *head)
{


     Node *p = new Node;
     cin >> p -> data;
     p -> next = NULL;

     q -> next = p;

     return head;
}
```

7.2 第 i 个节点之后插入一个节点

```
void InsertAfterNode(Node *head, int i)
{
  Node *current = head;
  int j = 1;
  while(j < i && current -> next != NULL)     //查找第i个节点
  {   current = current -> next;
      j++;
  }         //current指向第 i 个节点，或最后一个节点
  …


}
```

```
void InsertAfterNode(Node *head, int i)
{
   …
   if(j == i)
   {   Node *p = new Node;
       cin >> p -> data;
       p -> next = current ->next;
                 //让第i+1个节点链接在新节点之后
       current -> next = p; //让新节点链接在第i个节点之后
   }
   else     //链表中没有第i个节点
       cout << "没有节点:" << i << endl;
}
```

7.3 第 i 个节点之前插入一个节点

```
void InsertBeforeNode(Node *head, int i)
{
  Node *pre = NULL;
  Node *current = head;   // current指向第一个节点
  int j = 1;
  while(j < i && current -> next != NULL)     //查找第i个节点
  {  pre = current;
     current = current -> next;
     j++;
  }
  …

}
```

7.4 data为key的那个节点之前插入一个节点

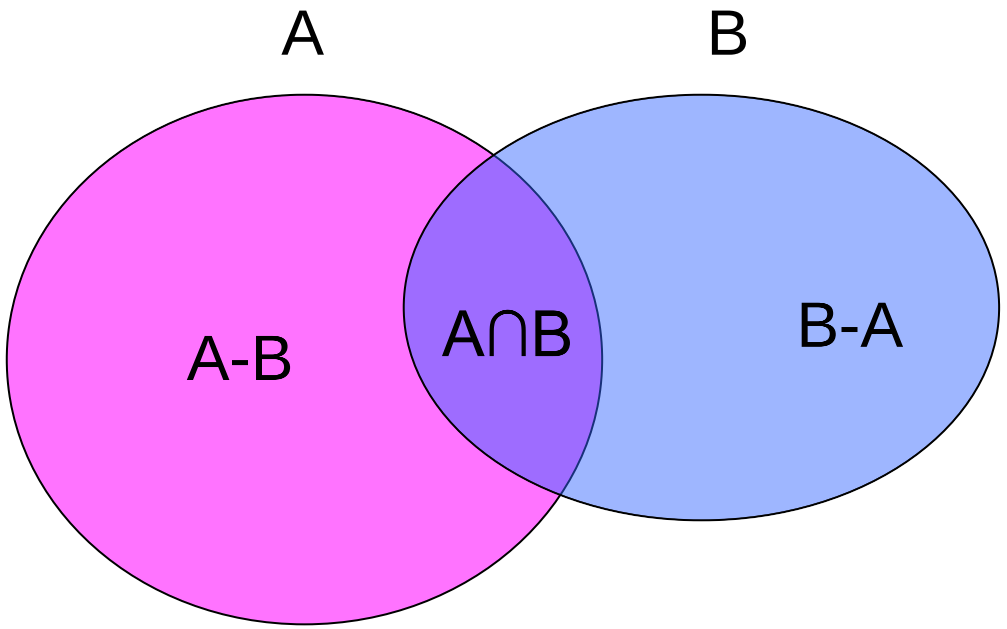
```
void InsertBeforeNode(Node *head, int key)
{
  Node *pre = NULL;
  Node *current = head;   // current指向第一个节点
  while(current ~~-> next~~ != NULL  && current -> data != key)
  {   pre = current;
      current = current -> next;
  }
  …


}
```

```
void InsertBeforeNode(Node *head, int key)
{
     ……
     if(current != NULL && pre != NULL)
     {    p -> next = current;
          pre -> next = p;
     }
     else if(current != NULL && previous == NULL)
     {    p -> next = current;
          head = p;
     }
     return head;
}
```

7.5 集合问题

- 设计*C/C++*程序，首先用链表建立两个集合（从键盘输入集合的元素），然后计算这两个集合的交集、并集以及一个差集，最后输出计算结果。

```
struct Node
{
        int content;
        Node *next;
};
```



```
Node * Sintersection (Node *head1, Node *head2)
{
     Node *head = NULL;
     for (Node *p=head1; p != NULL; p=p->next)
          for (Node *q=head2; q!=NULL; q=q->next)
               if (p->content == q->content)
               {    Node *r = new Node;
                    r->content = p->content;
                    r->next = head;
                    head = r;
               } //建链表，插入节点，A、B中都有的值
     return head;
}
```

```
Node * Sdifference (Node *head1, Node *head2) // A-B
{     Node *head=NULL;
      for (Node *p=head1; p != NULL; p=p->next)
      {     bool flag = true;
            for (Node *q=head2; q!=NULL; q=q->next)
                  if (p->content == q->content) //B中有该值
                        flag = false;
            if(flag == true)
            {     Node *r = new Node;
                  r->content = p->content;
                  r->next = head;
                  head = r;
            } //建链表，插入节点，B中没有、A中有的值
      }
      return head;
}
```

```
Node * Sunion (Node *head1, Node *head2)
{
     Node *head = NULL;
     head = Sdifference (head1, head2); //先求A-B
     for (Node *q=head2; q!=NULL; q=q->next)
     {    Node *r = new Node;
          r->content = q->content;
          r->next = head;
          head = r;
     } //头部插入B
     return head;
}
```

```
const int N = 5;
Node *AppCreate( )
{
     Node *head = NULL, *tail = NULL;
     for(int i = 0; i < N; ++i)
     {
          Node *p = new Node;
          cin >> p -> content;
          p -> next = NULL;
          if(head == NULL)
               head = p;
          else
               tail -> next = p;
          tail = p;
     }
     return head;
}
```

```
void Output(const Node *head)
{      while(head != NULL)
     {          cout << head -> content << " ";
                head = head->next;
     }
     cout << endl;
}

void DeleteList(Node *head)
{     while(head)
     {    Node *current = head;
          head = head -> next;
          delete current;
     }
}
```

```
int main()
{
     Node *list1 = AppCreate();
     Node *list2 = AppCreate();
     Node *list_I = Sintersection(list1, list2);
     Output(list_I);
     Node *list_D = Sdifference(list1, list2);
     Output(list_D);
     Node *list_U = Sunion(list1, list2);
     Output(list_U);
     DeleteList(list1);
     DeleteList(list2);
     DeleteList(list_I);
     DeleteList(list_D);
     //DeleteList(list_U);  //并集不是新建的链表，其中的节点已经被撤销
     return 0;
}
```

7.6 引用

```
int x = 0;
int &y = x;
```

```
void MySwap(int *pm, int *pn)
{
      int temp = *pm;
      *pm = *pn;
      *pn = temp;
}
```

```
void MySwap(int &pm, int &pn)
{
      int temp = pm;
      pm = pn;
      pn = temp;
}
```

```
int m = 3;
int n = 5;
MySwap(&m, &n);
```

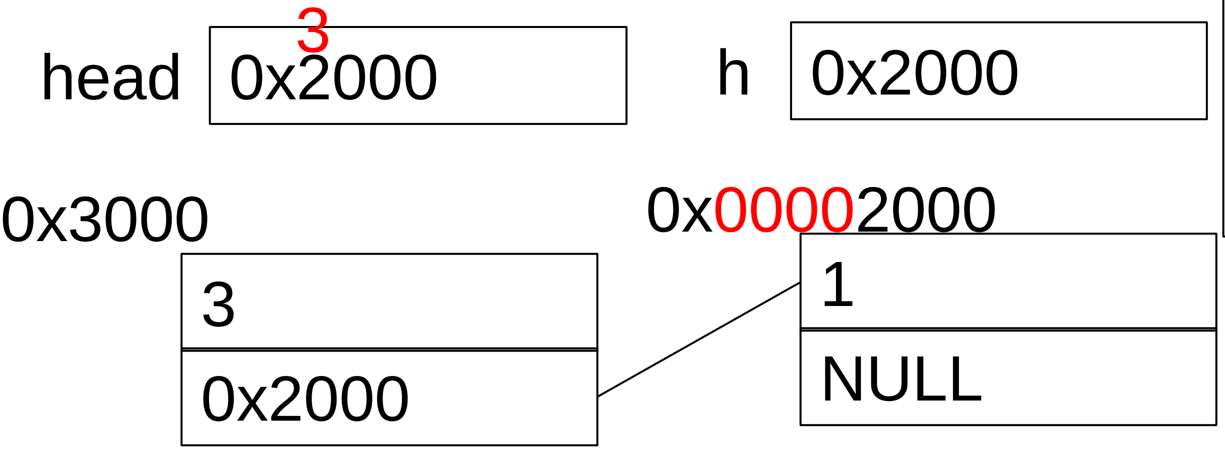
```
int m = 3;
int n = 5;
MySwap(m, n);
```

参数为指针的传值调用

```
int main()
{
   Node *h = new Node;
   h-> data = 1;
   h-> next = NULL;
   InsOneNode(h);
   …

   return 0;
}
```

已有链表头部插入1个结点?

```
void
~~Node *~~InsOneNode(Node *head)
{
       Node *p = new Node;
       cin >> p -> data;
       p -> next = head;//并未取值
       head = p;//并未取值

       ~~return head;~~
}
```

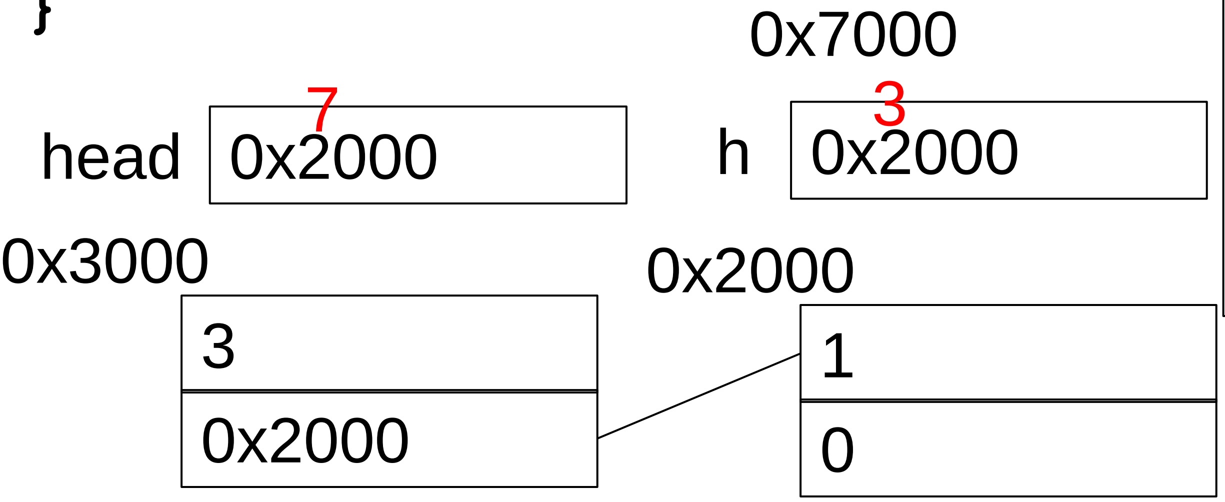


改为传址调用

```
int main()
{
   Node *h = new Node;
   h-> data = 1;
   h-> next = NULL;
   InsOneNode(&h);
   …

   return 0;
}
```

已有链表头部插入1个结点?

void
```
~~Node~~ *InsOneNode(Node **head)
{
       Node *p = new Node;
       cin >> p -> data;
       p -> next = *head;
       *head = p;

       ~~return head;~~
}
```



改为引用

```
int main()
{
  Node *h = new Node;
  h-> data = 1;
  h-> next = NULL;
  InsOneNode(h);
  …

  return 0;
}
```

已有链表头部插入1个结点?

```
void
~~Node *~~InsOneNode(Node *&head)
{
     Node *p = new Node;
     cin >> p -> data;
     p -> next = head;
     head = p;

     ~~return head;~~
}
```

删除头结点

```
Node *DeleteNode(Node *head)
{     Node *current = head;
      head = head->next;
      delete current;
      return head;
}

void DeleteNode(Node *&head)
{     Node *current = head;
      head = head->next;
      delete current;
      //return head;
}
```

祝大家期末考试取得好成绩！

Thanks！