第十六章 指针和数组

指针

- 一个存储对象（如，变量）的地址
- 使用指针，可以间接访问对象
  - 如，修改调用者的局部变量的函数

交换两个参数值的Swap函数

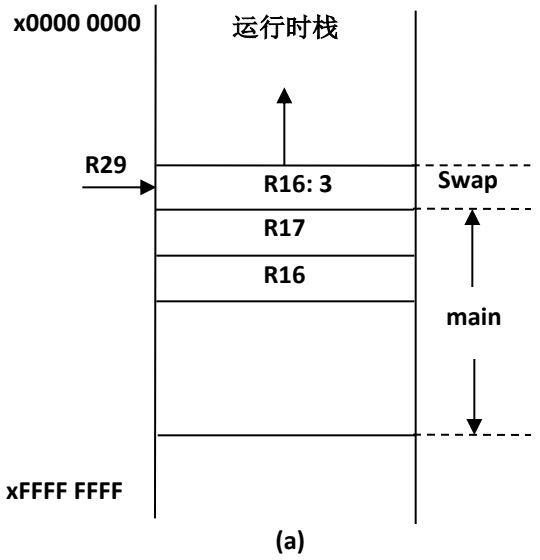
```
#include <stdio.h>

void Swap (int firstVal, int secondVal);

int main()
{
        int valueA = 3;              // R16
        int valueB = 4;              // R17

        Swap (valueA, valueB);

        printf ("valueA = %d and valueB = %d\n", valueA, valueB);
}

void Swap (int firstVal, int secondVal)     // R4: firstVal, R5: secondVal
{
        int tempVal;                 //R16

        tempVal = firstVal;
        firstVal = secondVal;
        secondVal = tempVal;
}
```

运行时栈

- 注：
  - 如无必要
    - 1、可以不用保存R31
    - 2、可以不使用R30

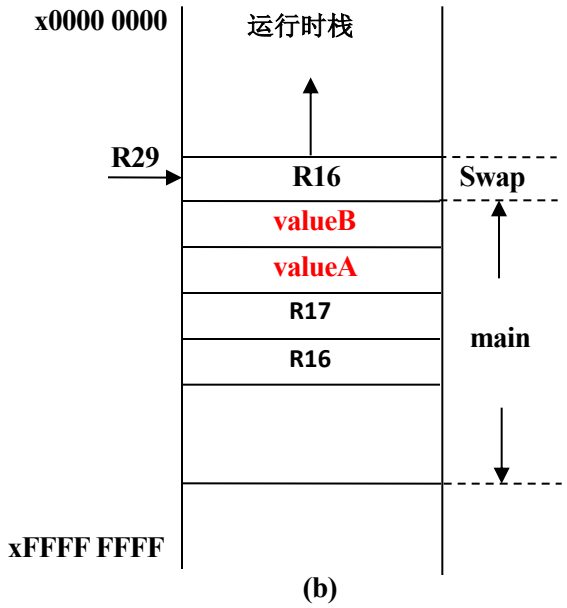
```
int main()
{
                int valueA = 3;                 // R16
                int valueB = 4;                 // R17
                Swap (valueA, valueB);
.....
}
void Swap (int firstVal, int secondVal)         // R4: firstVal, R5: secondVal
{
                int tempVal;                    //R16
                tempVal = firstVal;
                firstVal = secondVal;
                secondVal = tempVal;
}
```

```
01 Swap:    subi    r29, r29, #4
02          sw      0(r29), r16     ; 压入R16（寄存器的保存）
03
04          addi    r16, r4, #0     ; tempVal = firstVal;
05          addi    r4, r5, #0      ; firstVal = secondVal;
06          addi    r5, r16, #0     ; secondVal = tempVal;
07
08          lw      r16, 0(r29)     ; R16出栈
09          addi    r29, r29, #4
0A          jr      r31
```



指针

- 变元以值的形式从调用函数传递到被调用函数
- 要修改变元，必须在调用函数的活动记录中为变元分配空间
  - 通过访问存储它们的单元，修改变元
  - 指针及其相关运算



声明指针变量

- 指针变量
  - 包含一个存储对象的地址
  - 所指对象的类型
    - 整数指针变量，指向整数变量
- 声明
  - `type *pointer;`
    - 如

```
int *ptr;
char *cp;
double *dp;
```

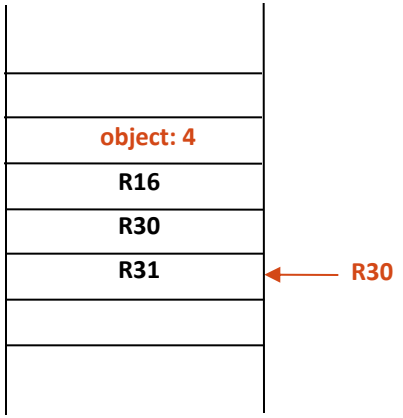
指针运算符

- 地址运算符“&”
- 间接运算符“*”

地址运算符

- “&”
  - 生成操作数的存储地址

```
int object;
int *ptr;

object = 4;
ptr = &object;
```

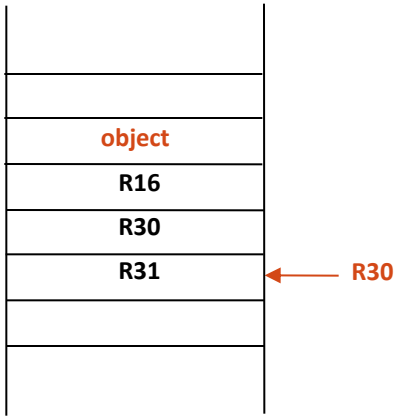



```
addi    r8, r0, #4      ; R8 = 4
sw      -12(r30), r8    ; object = 4;
subi    r16, r30, #12   ; ptr = &object;
```

间接运算符

- 解除参照运算符
  - “*”
  - 间接操作存储对象里的值
- type *pointer;
  - *pointer ，被指针变量pointer所指的值

间接运算符




```
int object;
int *ptr;

object = 4;
ptr = &object;
*ptr = *ptr + 1;
//object = object +1;
//object = *ptr + 1;
```

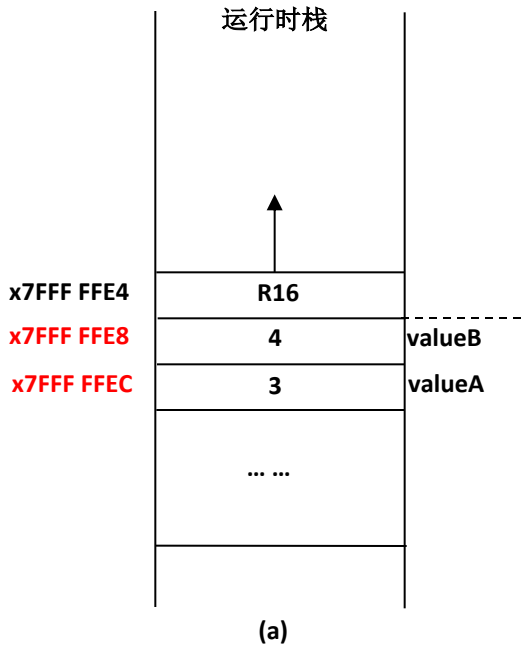
- *ptr
  - 赋值运算符的右边
    - 出现在单元中的值
  - 赋值运算符的左边
    - 要作修改的单元

```
lw      r8, 0(r16) ; R8←*ptr
addi    r8, r8, #1 ; *ptr + 1
sw      0(r16), r8 ; *ptr = *ptr + 1;
```

```
lw      r8, 0(r16)     ; R8←*ptr
addi    r8, r8, #1     ; *ptr + 1
sw      -12(r30), r8   ; object = *ptr + 1;
```

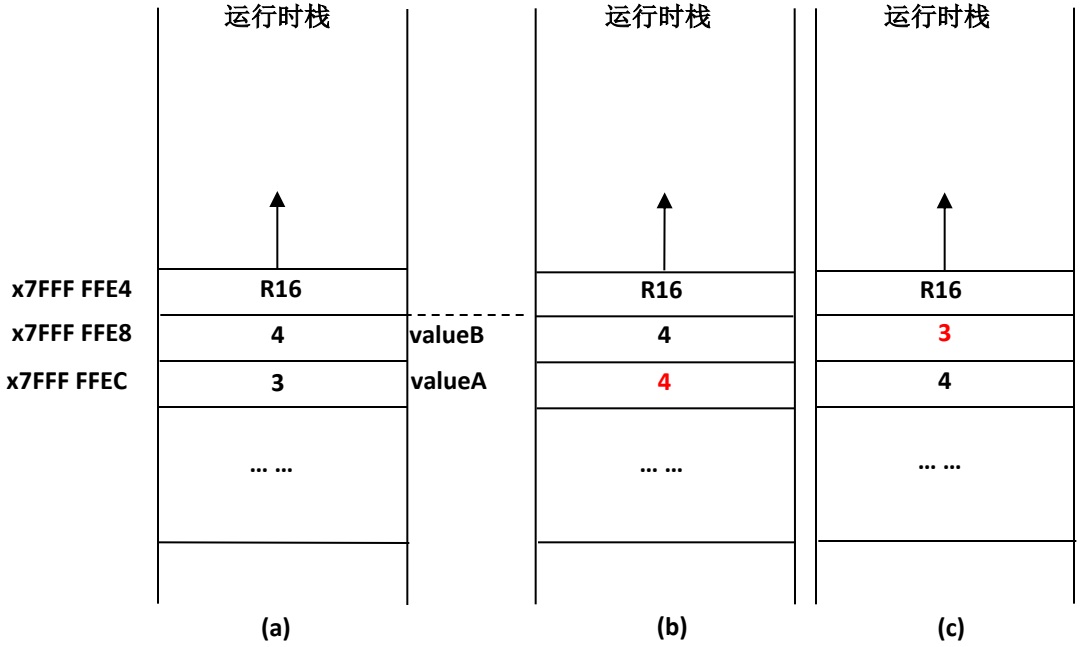
交换两个变量的NewSwap函数

```
#include <stdio.h>

void NewSwap (int *firstVal, int *secondVal);

int main()
{
    int valueA = 3;                          // x7FFF FFEC
    int valueB = 4;                          // x7FFF FFE8
    NewSwap (&valueA, &valueB);
    printf ("valueA = %d and valueB = %d\n", valueA, valueB);
}
void NewSwap (int *firstVal, int *secondVal) // R4: firstVal, R5: secondVal
{
    int tempVal;                             // R16
   tempVal = *firstVal;
    *firstVal = *secondVal;
    *secondVal = tempVal;
}
```



NewSwap

```
01   NewSwap:    subi    r29, r29, #4
02               sw      0(r29), r16     ; 压入R16（寄存器的保存）
03
04               lw      r8, 0(r4)       ; *firstVal
05               addi    r16, r8, #0     ; tempVal = *firstVal;
06               lw      r9, 0(r5)       ; *secondVal
07               sw      0(r4), r9       ; *firstVal = *secondVal;
08               sw      0(r5), r16      ; *secondVal = tempVal;
09
0A               lw      r16, 0(r29)     ; R16出栈
0B               addi    r29, r29, #4
0C               jr      r31
```



按引用传递变元

- 在NewSwap中，通过使用地址运算符“&”构建了一个对于变元的按引用的调用
- 当按引用传递变元时，其地址被传给被调用函数——为了使其有效，变元必须是变量或其他存储对象（即，必须有个地址）
- 被调用函数就能使用间接运算符“*”来访问（以及修改）其对象的原来的值

IA-32

- Swap.ppt
- NewSwap.ppt

问题求解：计算商和余数

- 给定一个除数与一个被除数，计算商与余数
- 难点：
  - 通过一个函数计算商和余数

```
return dividend/divisor; /*只能计算商*/
```

- 给调用函数提供多个值
  - 使用指针

IntDivide

```c
#include <stdio.h>

int IntDivide (int x, int y, int *quoPtr, int *remPtr);

int main()
{
    int dividend;              /*被除数*/
    int divisor;               /*除数*/
    int quotient;              /*商*/
    int remainder;             /*余数*/
    int error;

    printf ("Input dividend: ");
    scanf ("%d", &dividend);
    printf ("Input divisor: ");
    scanf ("%d", &divisor);

    error = IntDivide (dividend, divisor, &quotient, &remainder);

    if (!error)
        printf ("Answer: %d remainder %d\n", quotient, remainder);
    else
        printf ("IntDivide failed.\n");
}
```

```c
int IntDivide (int x, int y, int *quoPtr, int *remPtr)
{
    if (y != 0) {
            *quoPtr = x / y;              /*修改商*/
            *remPtr = x % y;              /*修改余数*/
            return 0;
    }
    else
            return -1;                    /*防御性编程实践*/
}
```

scanf

- 变元：变量地址

  scanf ("%d", &input);

- “%d”
  - 从输入流的下一个非空白字符开始，找到一个十进制数的ASCII序列，转化为整数，并存储在变量input中
  - 省略“&”，产生运行时错误
    - 程序试图修改一个它不能访问的存储单元

空指针

- 不指向任何东西的指针

```
int *ptr;
ptr = NULL;
```

  - NULL，特别定义的预处理宏
    - 如 0

指针函数

- 函数的返回值也可以被声明为指针类型

```
int *CmpSwap(int *firstVal, int *secondVal) {
   int tempVal;
   tempVal = *firstVal;
   *firstVal = *secondVal;
   *secondVal = tempVal;

   if (*firstVal >= *secondVal)
        return firstVal;
   else
        return secondVal;
}
```

数组

- 在存储器中连续排列的一列数据
  - 字符序列，字符数组
  - 向量、矩阵、列表等

数组

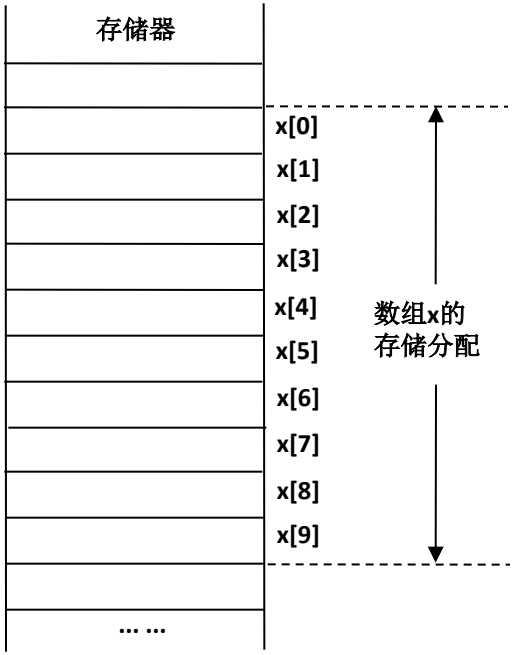
- 问题：一次课程考试中50个学生的期末成绩
  - 声明一个单独的对象，如examScore
  - 所有的元素是同一类型
- 一个连续的数值序列

数组的声明

- 类型

  int x[10];

  - 类型：int
  - 数组名：x
  - [ ]：数组
  - 10：包含10个顺序排列的整数
    - C89,不支持变量

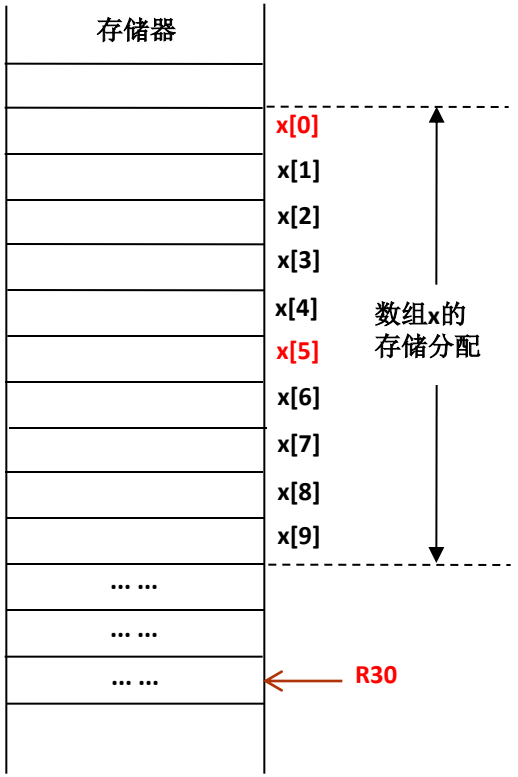


访问数组的元素

- 在[]中提供下标
  - 偏移量
  - 数组的第一个元素：编号为0

x[5] = x[0] + 1;

```
subi    r8, r30, #48    ; 将x[]的基址给R8
lw      r9, 0(r8)       ; R9←x[0]
addi    r9, r9, #1      ; R9←x[0] + 1
sw      20(r8), r9      ; x[5] = x[0] + 1;
```



基址+偏移量

- 数组的下标可以是任意的合法的C语言整数表达式

```
x[i + 5] = x[i] + 1;  //假设i被分配给R16
```

```
subi    r8, r30, #48      ; 将x的基址给R8
slli    r9, r16, #2       ; R9← i*4
add     r9, r8, r9        ; 计算x[i]的地址
lw      r10, 0(r9)        ; R10←x[i]
addi    r10, r10, #1      ; R10←x[i] + 1

addi    r9, r16, #5       ; R9←i + 5
slli    r9, r9, #2        ; R9← (i + 5)*4
add     r9, r8, r9        ; 计算x[i + 5]的地址
sw      0(r9), r10        ; x[i+5] = x[i] + 1;
```

示例1

- 每个学生的总评成绩=“exam1 × 10% + exam2 × 30%+ exam3 × 60%”
- 假设该门课程有50个学生修读

```c
#include <stdio.h>
#define NUM_STUDENTS 50

int main() {
    int i;
    int exam1[NUM_STUDENTS];
    int exam2[NUM_STUDENTS];
    int exam3[NUM_STUDENTS];
    int total[NUM_STUDENTS];

    /*输入平时成绩*/
    for (i = 0; i < NUM_STUDENTS; i++) {
            printf ("Input exam1 score for student %d : ", i);
            scanf ("%d", &exam1[i]);
    }
    printf ("\n");
```

```
/*输入期中考试成绩*/
for (i = 0; i < NUM_STUDENTS; i++) {
       printf ("Input exam2 score for student %d : ", i);
       scanf ("%d", &exam2[i]);
}
printf ("\n");

/*输入期末考试成绩*/
for (i = 0; i < NUM_STUDENTS; i++) {
       printf ("Input exam3 score for student %d : ", i);
       scanf ("%d", &exam3[i]);
}
printf ("\n");
```

```
    /*计算总评成绩*/
    for (i = 0; i < NUM_STUDENTS; i++) {
          total[i] = exam1[i]*0.1 + exam2[i]*0.3 + exam3[i]*0.6;
    }

    /*输出总评成绩*/
    for (i = 0; i < NUM_STUDENTS; i++) {
          printf ("Total score for student %d = %d\n", i, total[i]);
    }
}
```

编程风格

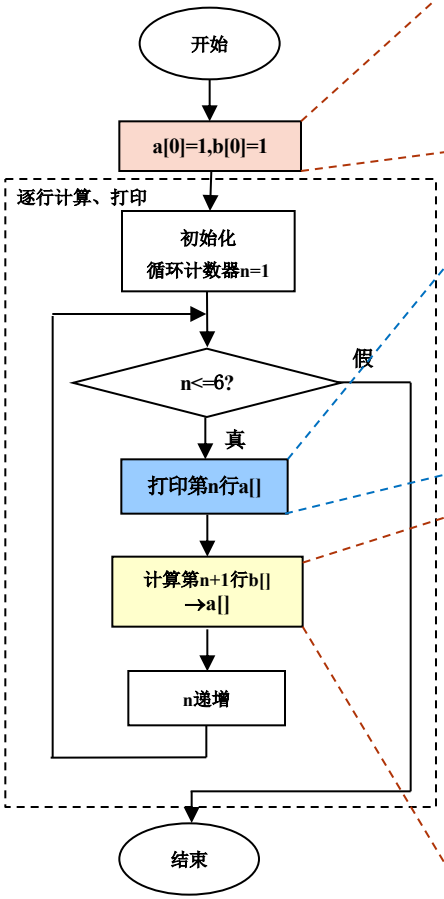
- 数组的大小，使用预处理宏
  - 增加数组的大小，只需改变宏的定义

打印杨辉三角形

- 杨辉三角形如：

```
                1
              1   1
            1   2   1
          1   3   3   1
        1   4   6   4   1
      1   5  10  10   5   1
... ...
```

NUMBER=6




```
#include <stdio.h>
#define NUMBER 6

int main()
{
    int i, n;
    int a[NUMBER];                          /*当前行的数*/
    int b[NUMBER];                          /*下一行的数*/

    a[0] = 1;                               /*当前行第一个数为1*/
    b[0] = 1;                               /*下一行第一个数为1*/
    for (n = 1; n <= NUMBER; n++)
    {
        for (i = 0; i < NUMBER - n; i++)    /*打印空白*/
            printf("  ");

        for (i = 0; i < n; i++)
            printf("%2d  ", a[i]);         /*打印a[i]*/
            /*“%2d”表示整数占两位，如果不足两位，则左边补空格*/
        printf("\n");

        if (n < NUMBER)
        {
            for (i = 1; i <= n; i++)        /*计算下一行*/
            {
                if (i == n)
                    b[i] = 1;               /*每行最后一个数为1*/
                else
                    b[i] = a[i - 1] + a[i]; /*上一行相邻两数之和*/
            }
            for (i = 0; i <= n; i++)
                a[i] = b[i];
        }
    }
}
```

数组初始化

```
int f[10]={0,1,2,3,4,5,6,7,8,9};
// f[0]=0,f[1]=1,……f[9]=9
int a[10]={0,1,2,3};
// a[0]=0,a[1]=1,……a[4]=0,a[5]=0,……a[9]=0
int b[]={0,1,2,3,4,5};
//数组长度为6
int c[5]={0,1,2,3,4,5};
//error : too many initializers
```

数组与指针之间的关系

- 数组的名字：数组的基址

```
int x[10];
int *ptr;


ptr = x;
```

- x：等价于&x[0]
- x[5] 或 *(ptr + 5)：访问同一元素

区别

```
int x[10];
int *ptr;

ptr = x;
```

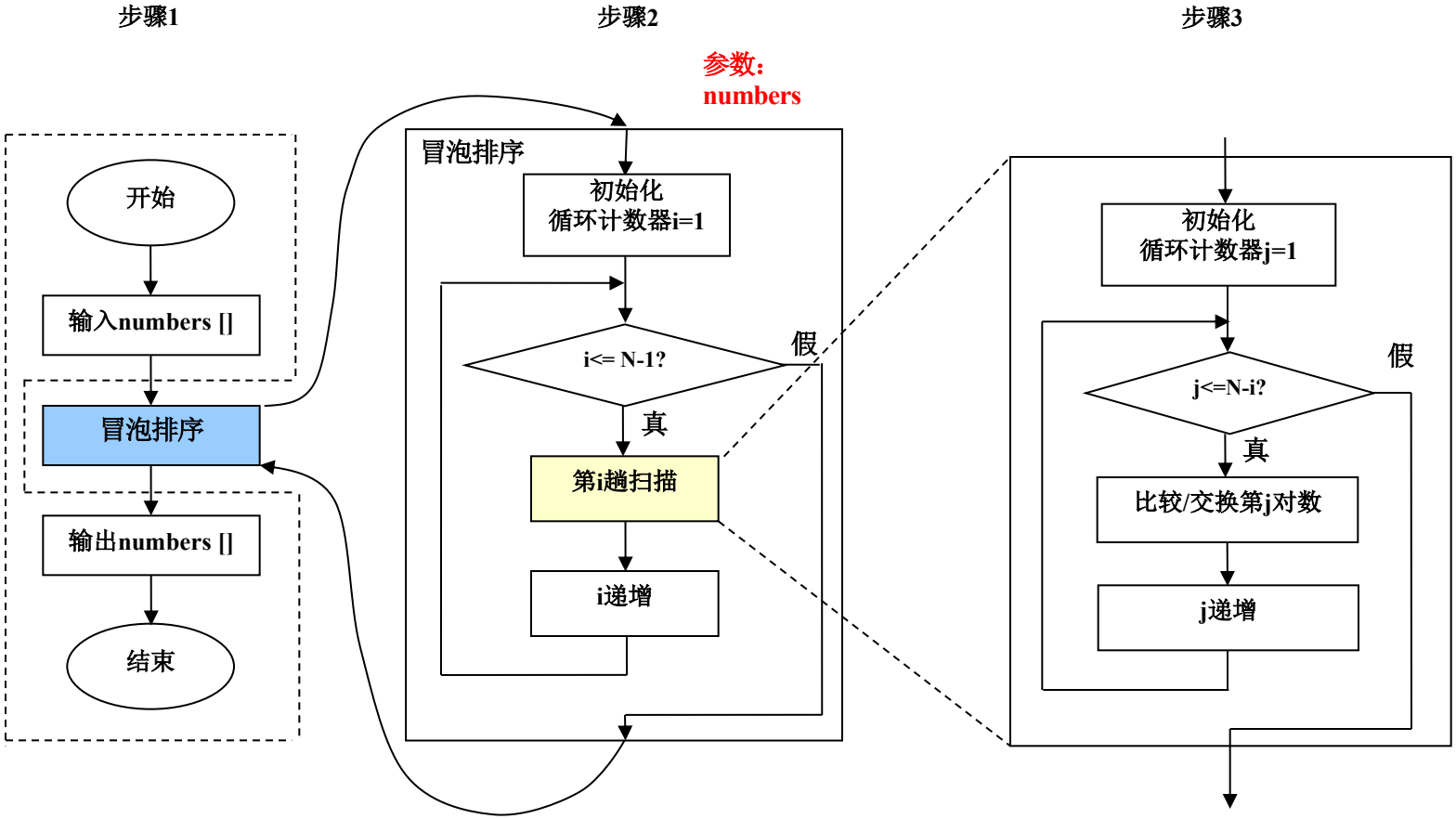
- ptr：变量，可以被重新赋值
- 数组名x：不能被重新赋值
  - x = ptr++; //非法！

数组作为参数

- 在函数之间传递数组
- 问题：把数组中的所有值从一个函数传递到另一个函数中，花费大！
- C语言：传递数组名（数组的基址）

问题求解：冒泡排序

- 将一个整数数组按升序排列
  - x[0]≤x[1]≤x[2]……≤x[N-1]
- 算法思想：
  - 第一趟：依次比较相邻的两个数，将小数放在前面，大数放在后面
    - 比较第1个和第2个数，将小数放在前面，大数放在后面；
    - 比较第2个数和第3个数……；
    - ……
    - 比较最后两个，……。
    - ——第一趟比较结束，最大的一个数被交换到最后！
  - 第二趟：
    - 从第一对数开始，……比较最大数前的一对相邻数。
    - ——第二趟结束，在倒数第二个数中得到一个新的最大数！
  - ……
  - 第N-1趟：
    - 比较第一个数和第二个数。
    - ——完成排序！


步骤1
步骤2
步骤3
参数:
numbers
开始
输入numbers []
冒泡排序
输出numbers []
结束
冒泡排序
初始化
循环计数器i=1
i<= N-1?
假
真
第i趟扫描
i递增
初始化
循环计数器j=1
j<=N-i?
假
真
比较/交换第j对数
j递增

函数声明

```
1 #include <stdio.h>
2 #define MAX_NUMS 10
3
4 void BubbleSort (int list[]);
5
```

- list，可省略
- “[ ]”
  - 参数是特定类型的数组的基址

  void BubbleSort (int *list);

函数调用

```
6 int main()
7 {
8           int index;
9           int numbers [MAX_NUMS];
10
11          /*获取输入*/
12          printf ("Enter %d numbers.\n", MAX_NUMS);
13          for (index = 0; index < MAX_NUMS; index++)
14          {
15                      printf ("Input number %d : ", index);
16                      scanf ("%d", &numbers[index]);
17          }
18
19          /*调用排序程序*/
20          BubbleSort (numbers);
21
22          /*输出已排序的数组*/
23          printf ("\nThe input set, in ascending order:\n");
24          for (index = 0; index < MAX_NUMS; index++)
25                      printf ("%d\n", numbers[index]);
26 }
27
```

- numbers
  - 数组的基址
  - 等价于 &numbers[0]
  - 类型 与int*类似

函数定义

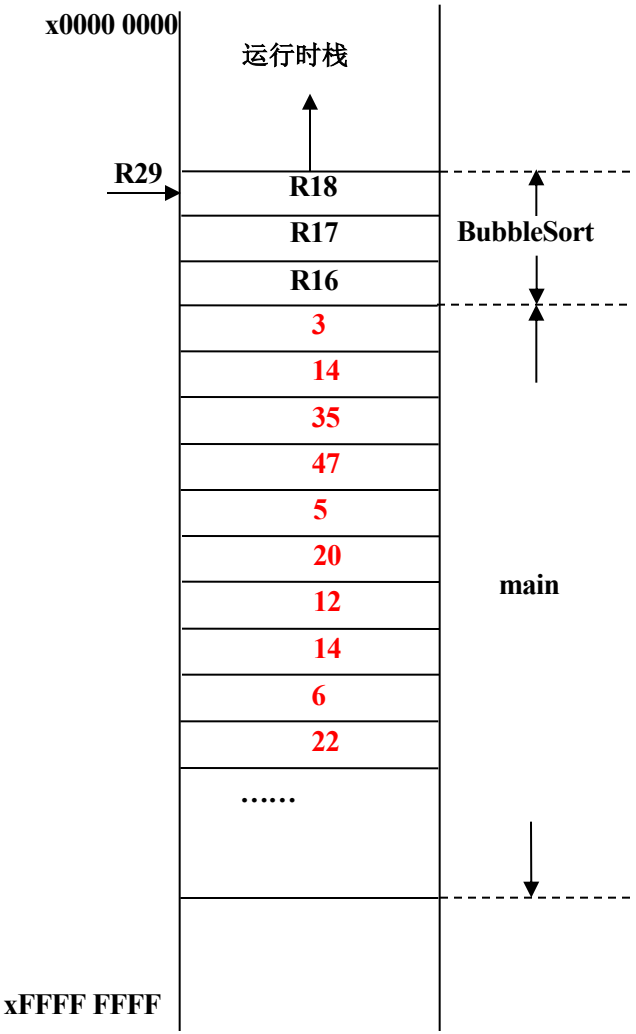
```
28          void BubbleSort (int list[])
29          {
30                      int i, j;
31                      int temp;
32                      for (i = 1; i <= MAX_NUMS-1; i++)
33                                  for (j = 1; j <= MAX_NUMS-i; j++)
34                                              if (list[j - 1] > list[j])
35                                              {
36                                                          temp = list[j-1];
37                                                          list[j - 1] = list[j];
38                                                          list[j] = temp;
39                                              }
40          }
```

- 参数list：&numbers[0]
  - list[2]：numbers[2]
  - *(list + 2)：numbers[2]
- 赋值运算符右边
  - 取出该地址中的值
- 赋值运算符左边
  - 要作修改的地址

- 在被调用函数中对数组值的修改都是可见的
- R4：list，&numbers[0]

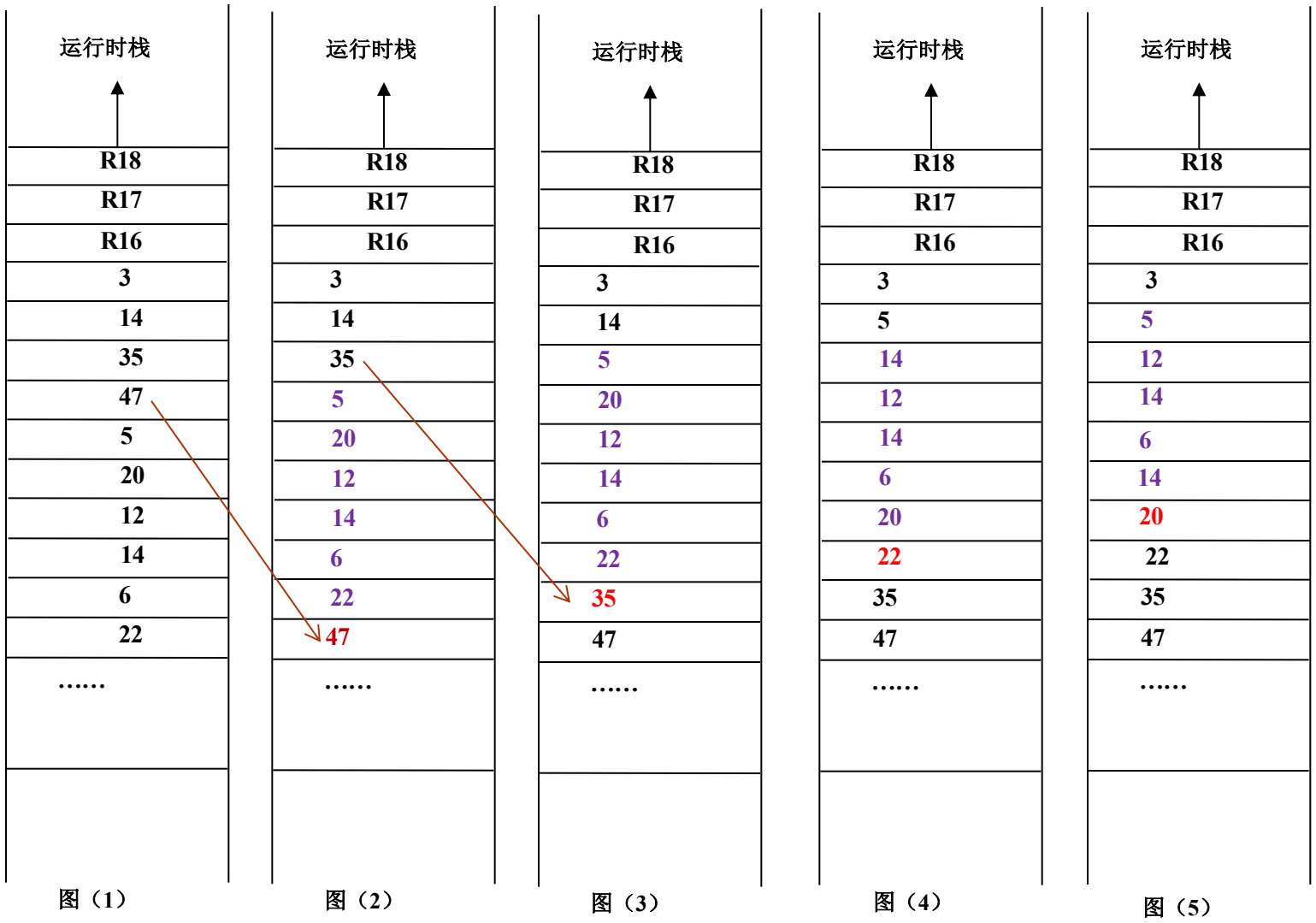
```
01  BubbleSort:  subi   r29, r29, #4
02                sw    0(r29), r16       ; 压入R16（寄存器的保存）
03                subi  r29, r29, #4
04                sw    0(r29), r17       ; 压入R17（寄存器的保存）
05                subi  r29, r29, #4
06                sw    0(r29), r18       ; 压入R18（寄存器的保存）
07 ;
08                addi  r16, r0, #1       ; i = 1, R16
09 OutLoop:       slei  r8, r16, #9
0A                beqz  r8, exit_1        ; i <= MAX_NUMS-1
0B                addi  r17, r0, #1       ; j = 1, R17
0C InLoop:        addi  r9, r0, #10
0D                sub   r9, r9, r16       ; MAX_NUMS-i
0E                sle   r8, r17, r9
0F                beqz  r8, exit_2        ; j <= MAX_NUMS-i
10 ;
11                subi  r8, r17, #1
12                slli  r8, r8, #2        ; (j-1)*4
13                add   r8, r4 , r8       ; &list[j - 1], R8
14                lw    r10, 0(r8)        ; list[j - 1], R10
```

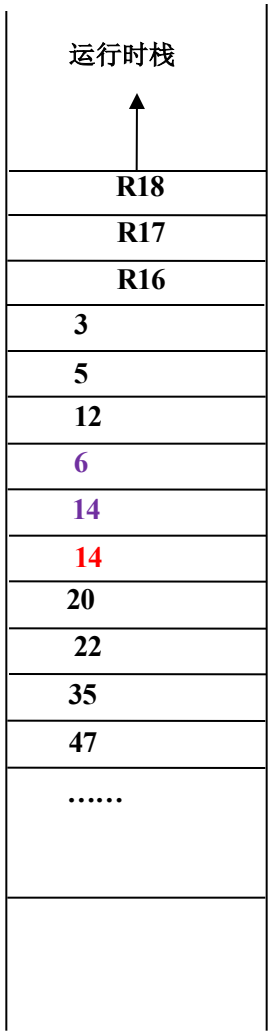
```
15                slli  r9, r17, #2       ; j*4
16                add   r9, r4, r9        ; &list[j], R9
17                lw    r11, 0(r9)        ; list[j], R11
18                slt   r12, r11, r10     ;if( list[j - 1] > list[j])
19                beqz  r12, exit_3
1A                addi  r18, r10, #0      ; temp = list[j-1]; R18
1B                sw    0(r8), r11        ; list[j - 1] = list[j];
1C                sw    0(r9), r18        ; list[j] = temp;
1D exit_3:        addi  r17, r17, #1      ;j++
1E                j     InLoop
1F exit_2:        addi  r16, r16, #1      ;i++
20                j     OutLoop
21 ;
22 exit_1:        lw    r18, 0(r29)       ; R18出栈
23                addi  r29, r29, #4
24                lw    r17, 0(r29)       ; R17出栈
25                addi  r29, r29, #4
26                lw    r16, 0(r29)       ; R16出栈
27                addi  r29, r29, #4
28                jr    r31
```



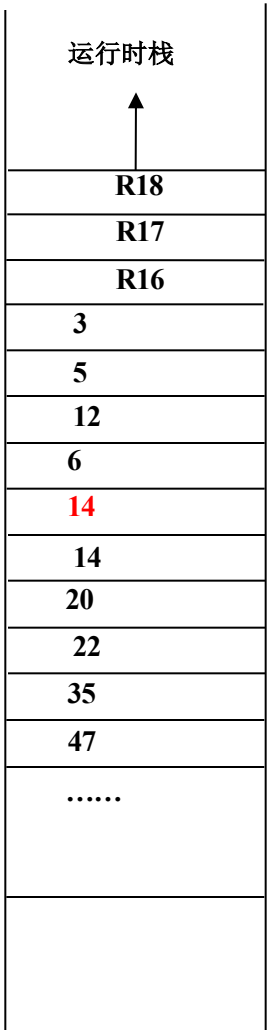
图（1）

```
28 void BubbleSort (int list[])
29 {
30            int i, j;
31            int temp;
32            for (i = 1; i <= MAX_NUMS-1; i++)
33                        for (j = 1; j <= MAX_NUMS-i; j++)
34                                    if (list[j - 1] > list[j])
35                                    {
36                                                temp = list[j-1];
37                                                list[j - 1] = list[j];
38                                                list[j] = temp;
39                                    }
40 }
```

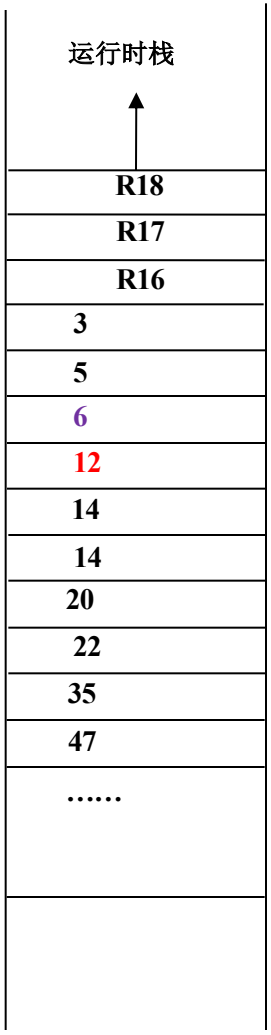

运行时栈
R18
R17
R16
3
14
35
47
5
20
12
14
22
……
图（1）
运行时栈
R18
R17
R16
3
14
35
5
20
12
14
6
22
47
……
图（2）
运行时栈
R18
R17
R16
3
14
5
20
12
14
6
22
35
47
……
图（3）
运行时栈
R18
R17
R16
3
5
14
12
14
6
20
22
35
47
……
图（4）
运行时栈
R18
R17
R16
3
5
12
14
6
14
20
22
35
47
……
图（5）



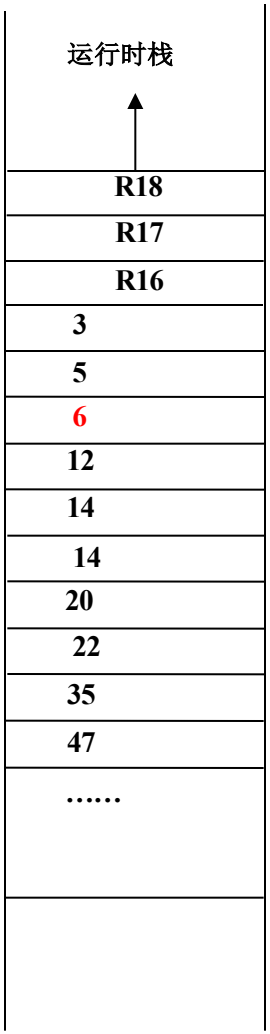
图（6）



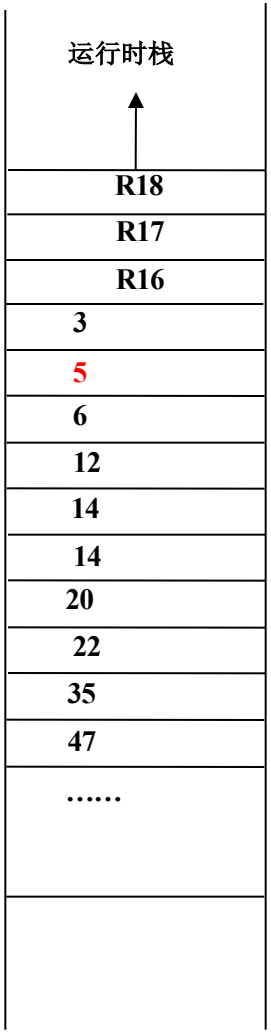
图（7）



图（8）

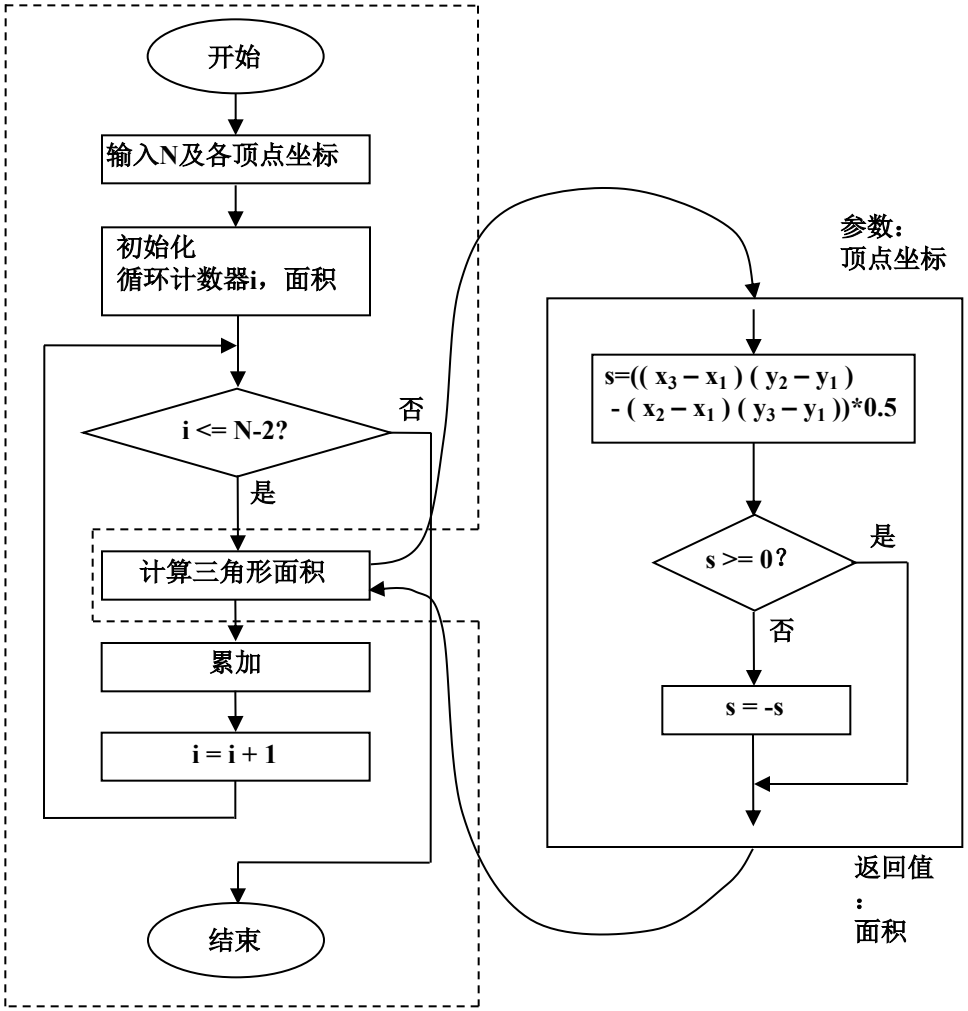


图（9）



图（10）

计算凸多边形面积


开始
输入N及各顶点坐标
初始化
循环计数器i，面积
i <= N-2?
否
是
计算三角形面积
累加
i = i + 1
结束
参数：
顶点坐标
s=((x3 − x1) (y2 − y1)
- (x2 − x1) (y3 − y1))*0.5
s >= 0?
是
否
s = -s
返回值
：
面积

计算三角形的面积

```
/*计算三角形的面积*/
double AreaOfTriangle (double coX[], double coY[]) {
   double s;
   s = ((coX[2] - coX[0]) * (coY[1] - coY[0]) -
   (coX[1] - coX[0]) * (coY[2] - coY[0])) * 0.5;
   if (s >= 0)
        return s;
   else
        return -s;
}
```

main

```
int main() {
   int n;                                        /*N*/
   double xa[MAX_NUMS];   /*多边形各顶点x坐标*/
   double ya[MAX_NUMS];   /*多边形各顶点y坐标*/
   double x[3];                    /*三角形各顶点x坐标*/
   double y[3];                    /*三角形各顶点y坐标*/
   double area = 0.0;              /*面积*/
   int i, j;

/*输入N及各顶点坐标*/
   printf ("Enter the sides number of polygon(<=%d): ",
   MAX_NUMS);
   scanf ("%d", &n);
```

```
    for (i = 0; i < n; i++) {
            printf ("Enter the coordinates of vertex A%d: ", i + 1);
            scanf ("%lf %lf", &xa[i], &ya[i]);
    }

    x[0] = xa[0];
    y[0] = ya[0];
    for (i = 0; i < n-2; i++) {
            for (j = 1; j < 3; j++) {
                    x[j] = xa[i + j];
                    y[j] = ya[i + j];
            }
            area += AreaOfTriangle (x, y);
    }
.....
}
```

计算平均值

- 按引用传递

```
main() {
   int numbers[MAX_NUMS];
   …
   mean = Average(numbers);
//mean = Average(&numbers[0]);
   …
}


int Average(int inputValues[MAX_NUMS]) {
   …
   for (index = 0; index < MAX_NUMS; index++)
       sum = sum + inputValues[index];
   return (sum / MAX_NUMS);
}
```

This must be a constant, e.g.,
`#define MAX_NUMS 10`

或者

```
int Average(int *inputValues) {
      …
        for (index = 0; index < MAX_NUMS; index++)
                sum = sum + *(inputValues+index);
        return (sum / MAX_NUMS);
}
或
int Average(int inputValues[]) {
      …
        for (index = 0; index < MAX_NUMS; index++)
                sum = sum + *(inputValues+index);
//sum = sum + inputValues[index];
        return (sum / MAX_NUMS);
}
```

数组的名字

- 在C语言中，一个数组的名字指的是数组的基址
  - AreaOfTriangle(x, y);
    - 名字x等价于&x[0]
    - x的类型与double*类似，它是包含了一个浮点数的存储单元的地址

传递变元

- 使用x和y作为传递给函数AreaOfTriangle的变元，就是将数组x和y的地址传递给函数AreaOfTriangle
- 在函数AreaOfTriangle内部，参数coX就被赋值为数组x的首地址，coY被赋值为数组y的首地址
- 在AreaOfTriangle中，可以用标准的数组符号来访问原数组中的元素

通过引用传递

- 任何在被调用函数中对数组值的修改都是可见的
  - 如：X[i]=……
  - 赋值运算的左边

字符串

- 表示文本的字符序列
- 字符数组
  - char word[10];
  - 可以存储10个字符的数组

赋值

```
char word[10];
word[0] = 'H';
word[1] = 'e';
word[2] = 'l';
word[3] = 'l';
word[4] = 'o';
word[5] = '\0';
```

- 字符串结尾
- ASCII码值为0的空字符
- 字符串"Hello"

声明时初始化

```
char word[10] = "Hello";
```

- 注意：
  - 一，使用双引号
    - 与单引号区分（如'A'）
  - 二，在字符串末尾自动加一个空字符

在声明时初始化

```
char c[10]={'I',' ','a','m',' ',
'h','a','p','p','y'};
// c[0]='I',c[1]=' ',……c[9]='y'
```

- 输出：

```
for(i=0;i<10;i++)
   printf("%c", c[i]);
```

%s

```
printf ("%s", word);
```

- “%s”，从参数（word）所指的字符开始，以'\0'结尾，打印字符串

scanf

scanf ("%s", word);

- 从输入流中扫描一个字符串
- 从一个非空白符开始，到下一个空白字符之间的所有字符
- 存储于以地址word开始的存储单元中
  - '\0'被自动添加

%s

scanf ("%s", word);

- 用户输入：

Let's go.

- word ← "Let's"
- 余下的保留在输入流中
- 再执行一个

scanf ("%s", word);

- word ← "go"

问题

- 如果单词长度超过9个字符，会发生什么？
- scanf并不检验数组word的大小
  - 把字符依次存储进地址word开头的单元中
  - 数组word[]后面的单元内容会被改写

数组的常见错误

- C语言，在访问数组时不进行范围检测
- 编译器
  - 为“x[i]”生成代码
  - i可能超出数组尾部

防御性编程

- 使用数组时
  - 在程序中加入范围检查
    - 数组的大小是否足够

I/O with Strings

```
char c[10]={'I',' ','a','m',' ',
'h','a','p','p','y'};
printf("%s", c);
```

•输出错误!

```
char c[10]= "I am happy";
printf("%s", c);
```

•输出错误!

```
char c[ ]= "I am happy";
printf("%s", c);
```

•正确!

```
char c[11]={'I',' ','a','m',' ',
'h','a','p','p','y','\0'};
printf("%s", c);
```

•正确!

scanf格式说明

- “%s”
  - 从输入流中扫描一个字符串，从一个非空白符开始（抛弃掉之前的空白 ） ，到下一个空白字符（不抛弃，仍在输入流中）之间
  - 一个'\0'字符被自动的添加进来
  - “%s%c”输入流中的空白也作为字符
    - 输入xy a，%c对应的为“空格”
  - “%s　　%c”（空格>0）输入流中的空白不作为字符
    - 输入xy a、xy　　a结果相同， %c对应的为a

计算字符串长度

```
#include <stdio.h>
#define MAX_STRING 20

int StringLength (char string[]);

int main()
{
    char input[MAX_STRING];
    int length = 0;

    printf ("Input a word (less than 20 characters) : ");
    scanf ("%s", input);

    length = StringLength (input);
    printf ("The word contains %d characters\n", length);
}
```

```
int StringLength (char string[])
{
   int index = 0;

   while (string[index] != '\0')
          index = index + 1;

   return index;
}
```

字符串比较

```
#include <stdio.h>
#define MAX_STRING 20

int StrCmp (char *firstStr, char *secondStr);

int main()
{
     char input1[MAX_STRING];
     char input2[MAX_STRING];

     printf ("Input the first word (less than 20 characters) : ");
     scanf ("%s", input1);

     printf ("Input the second word (less than 20 characters) : ");
     scanf ("%s", input2);

     printf ("The result is %d.\n", StrCmp (input1, input2));
}
```

```
int StrCmp (char *firstStr, char *secondStr)
{
     int i = 0;
     int res;

     while (!(res = firstStr[i] - secondStr[i]) && secondStr[i])
     {
          i++;  /*如果二字符相同且第二个字符串未到末尾，继续比较*/
     }
      if (res < 0)        /*res保存字符比较的结果*/
          res = -1 ;
      else if (res > 0)
         res = 1 ;
      return res;
}
```

标准库函数

- 字符数组
  - 复制字符串
  - 结合字符串
  - 比较字符串
  - 计算字符串长度
  - ……
- 头文件<string.h>

C standard library

- <string.h>
- strcat(str1,str2); //str1=str1@str2 ×
- strcpy(str1,str2); //str1=str2 ×
- strcmp(str1,str2); //str1==str2 ×
- strlen(str1);
- strlwr(str1); //lowercase
- strupr(str1); //uppercase

数组与指针之间的关系

```
char word[10];
char *cptr;
cptr = word;
```

- 可以通过使用“word[3]”或“*(cptr + 3)”访问字符串中的第四个字符

cptr	word	&word[0]
(cptr + n)	word + n	&word[n]
*cptr	*word	word[0]
*(cptr + n)	*(word + n)	word[n]

区别

- cptr是一个变量，从而可以被重新赋值
- 而另一个数组标识符word不能被重新赋值

C89

- 非法的C代码:

```
void SomeFunction (int num_elements) {
    int temp[num_elemments];            /*生成语法错误*/
}
```

- 为了处理这个限制，C程序员仔细分析代码将要被使用的情形，然后为数组分配足够的空间
- 另一个可选的方法是使用动态存储分配，在运行时为这个数组分配空间

二维数组

- 一个由3行、4列组成的二维数组的声明：

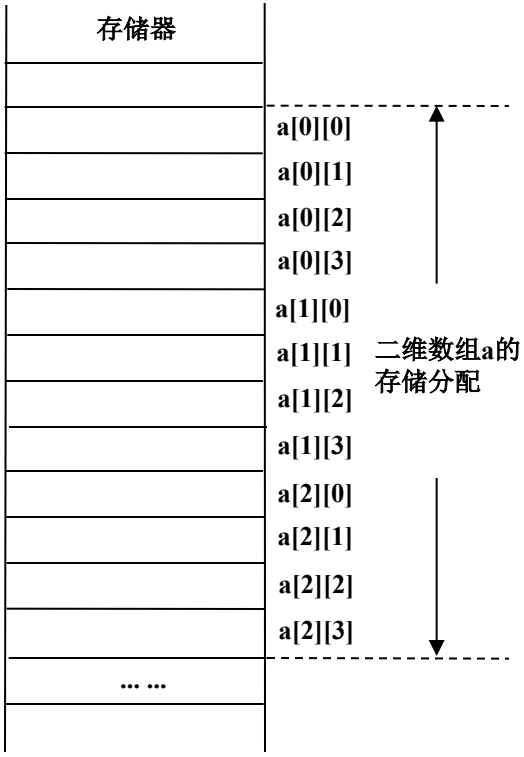
```
int a[3][4];
```

  - 关键词int表明声明的是整数类型的事物，a是数组的名称
  - 声明中出现了两个中括号，这就表明声明的是一个二维数组，第一个中括号表示行，第一个表示列
    - 3和4分别表示这个数组包含3行、4列整数

访问

对这12个整数的访问如下所示：

a[0][0] a[0][1] a[0][2] a[0][3]
a[1][0] a[1][1] a[1][2] a[1][3]
a[2][0] a[2][1] a[2][2] a[2][3]



初始化

```
int a[3][4]={{0,1,2,3},{4,5,6,7},{8,9,10,11}};
```

- a[0][0]=0, a[0][1]=1, a[0][2]=2, a[0][3]=3, a[1][0]=4, a[1][1]=5, a[1][2]=6, a[1][3]=7……

```
int b[3][4]={{1},{4,5},{8}};
```

- b[0][0]=1, b[0][1]=0, b[0][2]=0, b[0][3]=0, b[1][0]=4, b[1][1]=5, b[1][2]=0, b[1][3]=0……

```
int aa[][4]={0,1,2,3, 4,5,6,7, 8,9,10,11};
```

数组第一维长度为3

```
int bb[][4]={{0,1,2},{},{4,5}};
```

数组第一维长度为3

数组与指针的关系

- 数组名代表数组所占存储空间的首地址
- a代表该二维数组的首地址，等价于&a[0][0]
- 如何表示二维数组每一行的首地址呢？
  - 使用a[1]即可得到第1行的首地址，即a[1]等价于&a[1][0]

示例

- 对10个学生姓名进行排序：
- 首先，每个学生姓名都是一个字符数组，即字符串，10个学生的姓名可以声明如下：

```
char stuName[10][8];
```

  - 第一维的10表示有10名学生，第二维的8表示每个学生的姓名最多由7个字符构成

BubbleSort

```
void BubbleSort (char list[MAX_NUMS][8]) {
        int i, j;
        char temp[8];
        for (j = 1; j <= MAX_NUMS-1; j++)
                for (i = 0; i <= MAX_NUMS-j-1; i++)
                        if (strcmp (list[i], list[i + 1]) > 0) {
                                strcpy (temp, list[i]);
                                strcpy (list[i], list[i + 1]);
                                strcpy (list[i + 1], temp);
                        }
   }
```

main

```
int main() {
        int index;
        char stuName[MAX_NUMS][8];
        /*获取输入*/
        printf ("Enter names of %d students.\n", MAX_NUMS);
        for (index = 0; index < MAX_NUMS; index++) {
                printf ("Input name %d : ", index);
                scanf ("%s", stuName[index]);
        }
        /*调用排序程序*/
        BubbleSort (stuName);
        ……
  }
```

指针函数

- 函数的返回值类型是一个指针类型
  - 返回类型标识符　*函数名称（形式参数列表);

```
#include <stdio.h>
double *Find (double(*stu)[4], int n);
main() {
      double
  score[][4]={{60,70,80,90},{56,89,34,45},{34,23,56,45}};
      double*p;
      int i,m;
      printf("Enter the number to be found:");
      scanf("%d",&m);
      printf("the score of NO.%d are:\n",m);
      p=Find(score,m-1);
      for(i=0;i<4;i++)
            printf("%5.2f\t",*(p+i));
}
```

```
double *Find(double(*stu)[4],int n) {
   double *pt;
   pt=*(stu+n);
   return(pt);
}
```

- stu是指向指针的指针
  - double (*stu)[4] 等价于 double stu[][4];
  - *(stu+1)指向第1行，等价于stu[1]

二维数组指针

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
int (*p)[4]=a;
int **p1=a;  /*warning C4047: 'initializing' : 'int ** ' differs in levels of indirection from 'int (*)[4] ‘*/
int i=2;
int j=3;
```

p，按行访问

- a：&a[0][0]，如0x12FF18
- p：&a[0][0]，如0x12FF18
- *(p+i)：即p[i]，a[i]，如0x12FF38，第i行首地址
- *(p+i)+j：&a[i][j]，如0x12FF44，第i行第j个元素的地址
- *(*(p+i)+j)：a[i][j]，即12

p，指向指针的指针

- p: &a[0][0],如0x12FF18
  ;或&a[0],所以有*p: a[0],如0x12FF18
- p+i: a[i],如0x12FF38;
- *p+i: a[0]+i*sizeof(int),如0x12FF20 //一维数组
- (*p+i)+j: a[0]+i*sizeof(int)+j*sizeof(int),如0x12FF2C
- *((*p+i)+j): 即6

p1

- p1+i: a[0]+i*sizeof(int),如0x12FF20
- //一维数组
- *(p1+i):即3

函数指针

- int (*func) (int a); /* 声明一个函数指针 */
- 函数返回值类型 (*指针变量名)(形参列表);
- 函数指针不指向变量，而是指向函数
  - 函数名和数组名一样，代表了函数代码的首地址
- 用途：调用函数和做函数的参数

调用函数

```
int Func(int x);  /* 声明一个函数 */
int (*f) (int x);  /* 声明一个函数指针 */
f=Func; /* 将Func函数的首地址赋给指针f */
   (*f)和Func代表同一函数
(*f)(x);    /*x必须先赋值*/
```

习题

- 注：C-DLX编译
  - 寄存器分配规则
  - 如无必要
    - 1、可以不用保存R31
    - 2、可以不使用R30

- 上机
  - 16.3
  - 16.5
  - 16.6
    - 1)
    - 2)
  - 16.11
    - 2)
  - 16.13

- 书面作业
  - 15.7
  - 16.1
  - 16.2
  - 16.9
  - 16.10
  - 16.14