Array

- do-while loop
- nested loops
- concolusion of loop
- array
- basic operation on array
- function, revised

do-while loop

Count the number of digits

- Given a non-negative integer, print out the number of digits

```
int x;
scanf("%d", &x);
while (1) {
    if ( x==0 ) {
        break;
    }
    n++;
    x = x/10;
}
```

- This code does not deal with 0 as input

```
while (1) {
    n++;
    x = x/10;
    if ( x==0 ) {
        break;
    }
}
```

- Put the break condition at the end of the loop can deal with 0 naturally

- 当循环退出的条件是循环体内最后一句时，可以改造成do-while循环

```
while (1) {
    n++;
    x = x/10;
    if ( x==0 ) {
        break;
    }
}
```

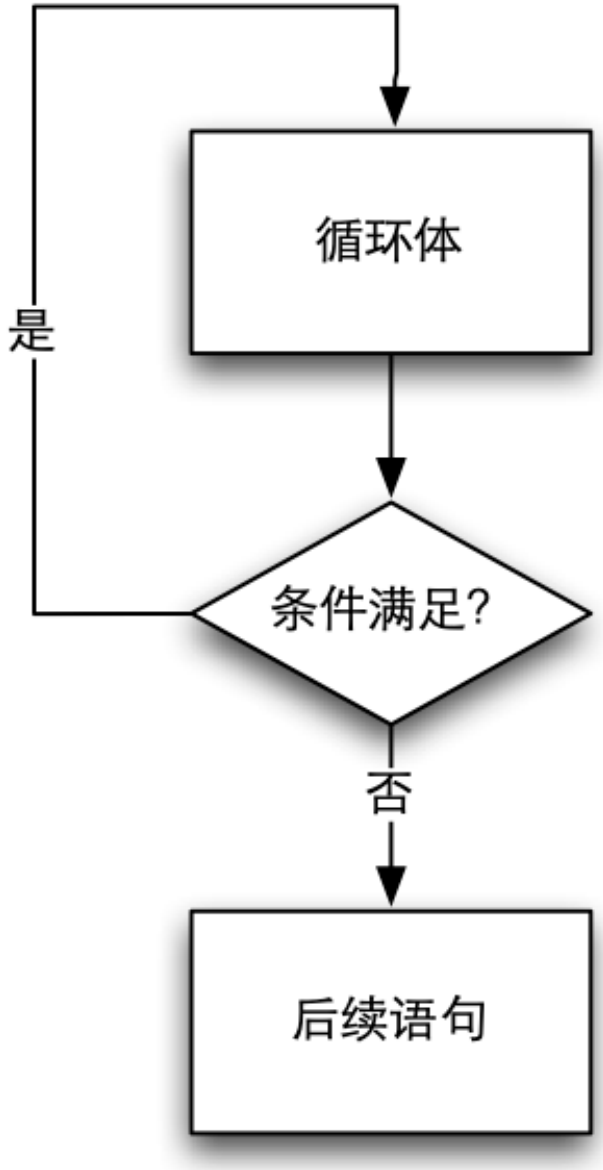
```
do {
    x /= 10;
    n ++;
} while ( x != 0 );
printf("%d", n);
```

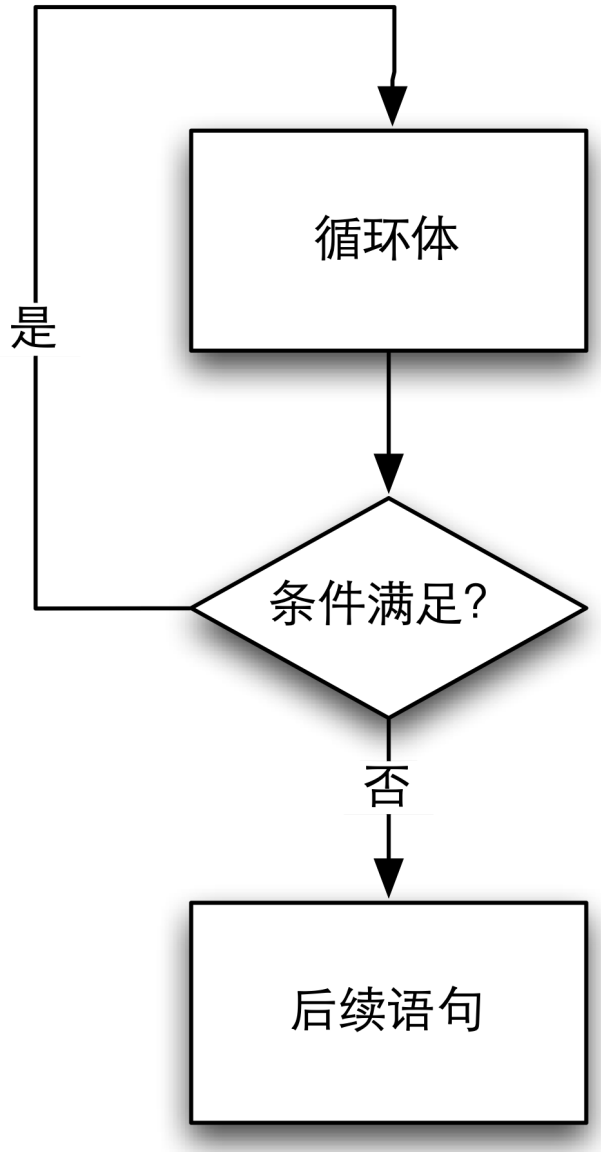
- 将条件取反作为do-while的条件

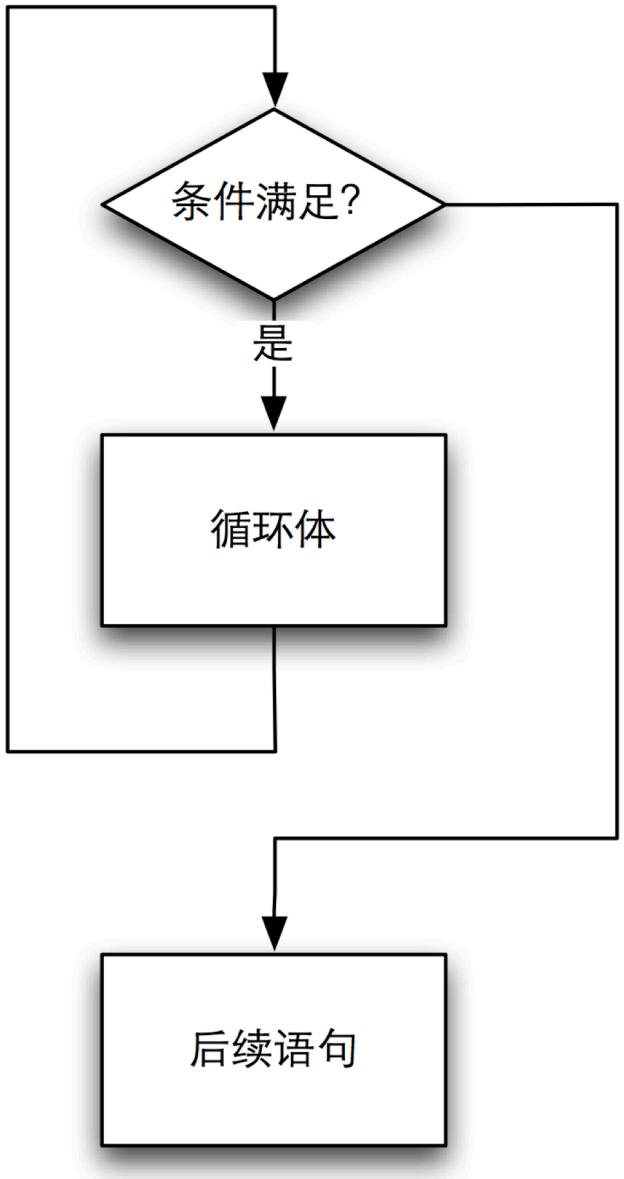
do-while loop

- Like a `while(1)` with a `break` at the bottom, it does not check the condition at the entering, but check it at the end of each iteration.

```
do {
    <loop body>
} while ( <condition> );
```







Validating input using do-while statement

```c
int get_int (int n_min, int n_max) {
    int in_val, /* input - number entered by user */
        status; /* status value returned by scanf */
    char skip_ch; /* character to skip */
    int error; /* error flag for bad input */
    /* Get data from user until in_val is in the range. */
    do {
        /* No errors detected yet. */
        error = 0;
        /* Get a number from the user. */
        printf("Enter an integer in the range from %d ", n_min);
        printf("to %d inclusive> ", n_max);
        status = scanf("%d", &in_val);
        /* Validate the number. */
        if (status < 1) { /* in_val didn't get a number */
            error = 1;
            scanf("%c", &skip_ch);
            printf("Invalid character >>%c<<. ", skip_ch);
            printf("Skipping rest of line.\n");
        } else if (in_val < n_min || in_val > n_max) {
            error = 1;
            printf("Number %d is not in range.\n", in_val);
        /* Skip rest of data line. */
        do
            scanf("%c", &skip_ch);
        while (skip_ch != '\n');
    } while (error);
    return (in_val);
}
```

两种循环

- do-while循环和while循环很像，区别是在循环体执行结束的时候才来判断条件。也就是说，无论如何，循环都会执行至少一遍，然后再来判断条件
- 与while循环相同的是，条件满足时执行循环，条件不满足时结束循环

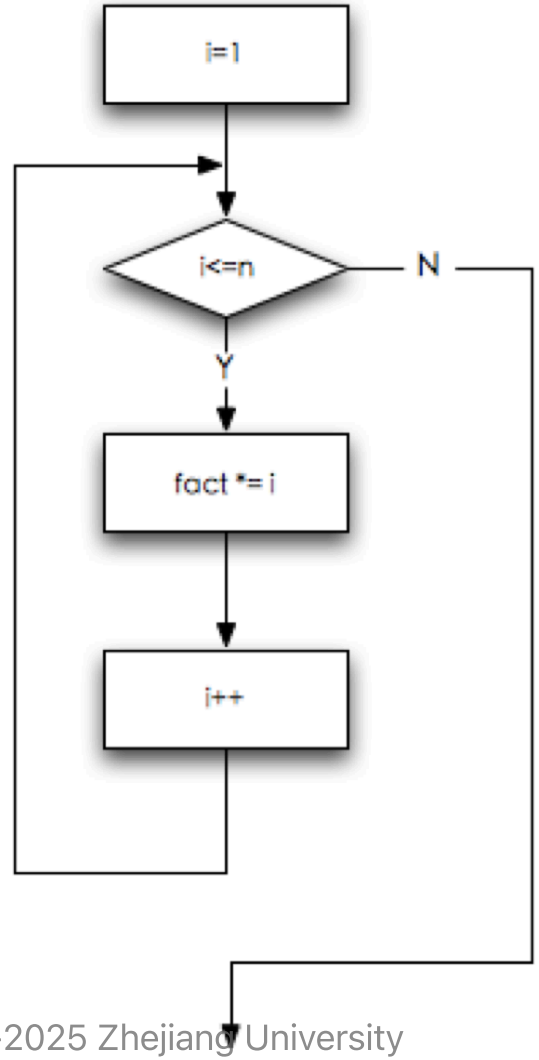
Quick check I

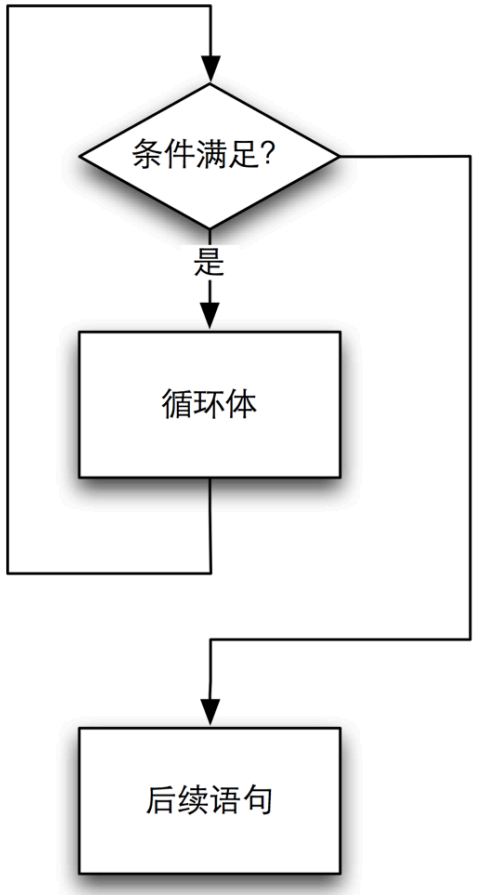
Which of the following code segments is a better way to implement a sentinel-controlled loop? Why?

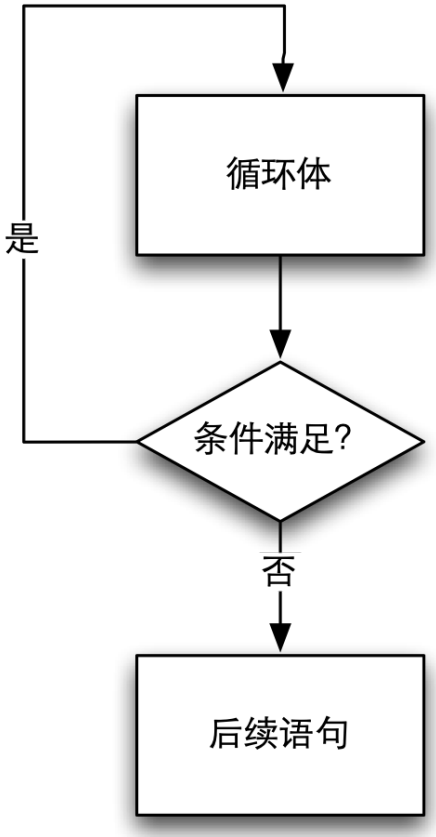
```
scanf("%d", &num);
while (num != SENT) {
    /* process num */
    scanf("%d", &num);
}
```

```
do {
    scanf("%d", &num);
    if (num != SENT) {
        /* process num */
    }
} while (num != SENT );
```

Three loops







Quick check II

Rewrite the following code using a do-while statement with no decisions in the loop body:

```
sum = 0;
for (odd = 1; odd < n; odd = odd + 2) {
    sum = sum + odd;
}
printf("Sum of the positive odd numbers less than %d is %d\n", n, sum);
```

In what situations will the rewritten code print an incorrect sum?

如果`if...break`在中间

```
while (1) {
    int x;
    scanf("%d", &x);
    if ( x<0 ) {
        break;
    }
    s += x;
    cnt += 1;
}
```

- 将循环体重复一遍（循环展开）

```
int x;
while (1) {
    scanf("%d", &x);
    if ( x<0 ) {
        break;
    }
    s += x;
    cnt += 1;
    scanf("%d", &x);
    if ( x<0 ) {
        break;
    }
    s += x;
    cnt += 1;
}
```

* `int x` 要拿到循环前面去

- 将第一句 `if` 之前的语句移到循环前面去
- 将第二句 `if` 开始的语句删除

```
int x;
scanf("%d", &x);
while (1) {
    if ( x<0 ) {
        break;
    }
    s += x;
    cnt += 1;
    scanf("%d", &x);
}
```

- 这是一个 `if` 是第一句的 `while`

```
int x;
scanf("%d", &x);
while (x>=0) {
    s += x;
    cnt += 1;
    scanf("%d", &x);
}
```

Quick check III-1

What is displayed by the following program segments, assuming `m` is 3 and `n` is 5?

```
for (i = 1; i <= n; ++i) {
    for (j = 0; j < i; ++j) {
        printf("*");
    }
    printf("\n");
}
```

Nested Loops

Loops may be nested just like other control structures. Nested loops consist of an outer loop with one or more inner loops. Each time the outer loop is repeated, the inner loops are reentered, their loop control expressions are reevaluated, and all required iterations are performed.

Quick check III-2

What is displayed by the following program segments, assuming `m` is 3 and `n` is 5?

```
for (i = n; i > 0; --i) {
    for (j = m; j > 0; --j) {
        printf("*");
    }
    printf("\n");
}
```

Coins

- 如何用1角、2角和5角的硬币凑出10元以下的金额呢?

```
for ( int one = 1; one < x*10; one+=1 ) {
    for ( int two = 1; two < x*10/2; two+=1 ) {
        for ( int five = 1; five < x*10/5; five += 1 ) {
            if ( one + two*2 + five*5 == x* 10 ) {
                printf("%d %d %d\n", one, two, five);
                break;
            }
        }
    }
}
```

- Break is for the loop where it is

break by break

```
int exit=0;
for ( int one = 1; one < x*10; one+=1 ) {
    for ( int two = 1; two < x*10/2; two+=1 ) {
        for ( int five = 1; five < x*10/5; five += 1 ) {
            if ( one + two*2 + five*5 == x* 10 ) {
                printf("%d %d %d\n", one, two, five);
                exit=1;
                break;
            }
        }
        if ( exit ) {
            break;
        }
    }
    if ( exit ) {
        break;
    }
}
```

goto

```
for ( int one = 1; one < x*10; one+=1 ) {
    for ( int two = 1; two < x*10/2; two+=1 ) {
        for ( int five = 1; five < x*10/5; five += 1 ) {
            if ( one + two*2 + five*5 == x* 10 ) {
                printf("%d %d %d\n", one, two, five);
                goto END;
            }
        }
    }
}
END:
```

Rules to `goto`

- In two situation `goto` is good:
  - To jump out of multiple loops
  - To jump to the end of a function

A word about `continue`

对循环的小结

- 确定次数的循环 —> `for`
  - 以及对数组和链表的遍历用 `for`
- 在某种条件下结束的循环 —> `while` or `do-while`
  - 至少执行一次 -> `do-while`
- `while` 和 `do-while` 语句里的条件是循环继续的条件
- `break` 的条件才是循环结束的条件
- 先从 `while(1)...if...break` 开始写循环，再根据情况改造成带条件的循环

Array

- How do we write a program to calculate the average of all the numbers user inputs?
- It is not needed to record every number that user inputs.

- 方差是衡量随机变量或一组数据时离散程度的度量，它描述随机变量对于数学期望的偏离程度，是各个样本数据和平均数之差的平方和的平均数

$$\sigma^2 = \frac{\sum(X-\mu)^2}{n}$$

- 这里，μ是均值
- 显然，要计算方差，需要先计算出均值，然后用之前读到的每一个值去减均值，所以，需要记录读到的每一个值

- How do we write a program to calculate the average of all the numbers user inputs, and all the numbers that are greater than the average?
- Every number inputed has to be recorded and be compared with the average after the end of the input.

- How to record many numbers?

```
int num1,num2,num3.....?
```

- An array is a collection of two or more adjacent memory cells, called array elements, that are associated with a particular symbolic name. To set up an array in memory, we must declare both the name of the array and the number of cells associated with it.

```
int number[100];
scanf("%d", &number[i]);
```

- Each element of array `number` may contain a single type `int` value, so a total of 100 such numbers may be stored and referenced using the array name `number`.

Iteration of an array

```c
int x;
double sum = 0;
int cnt = 0;
int number[100];
scanf("%d", &x);
while ( x!= -1 ) {
    number[cnt] = x;
    sum += x;
    cnt ++;
    scanf("%d", &x);
}
if ( cnt >0 ) {
    int i;
    double average = sum / cnt;
    for ( i=0; i<cnt; i++ ) {
        if ( number[i] > average ) {
            printf("%d ", number[i]);
        }
    }
}
```

define an array

assignment for element

This program is not safe. Why?

to use element

iteration of the array

Defining an array

```
<type> name[number of elements];
int grades[100];
double weight[20];
```

- Number of elements has to be an integer
- Before C99: the number has to be a literal

Array(数组)

- 生活中有各种各样的容器，它们都可以往里面放东西，也能从里面取出来
- 在程序中也有这样存放多个数据的手段，数组里面可以放多个元素
- 数组的大小（元素的个数）是确定的

序列容器

- 这是一个7天药盒，每天要吃的药分别放在不同的格子里，这也是一种容器。这种容器里面的东西有确定的顺序，它会保持着你放进去的顺序

Array

- is a kind of container (thing that hold things):
  - All elements are of the same type;
  - Once created, it is not allowed to change the size(capacity)
  - 数组中的元素在内存中是连续依次排列的

```
double x[8];
```

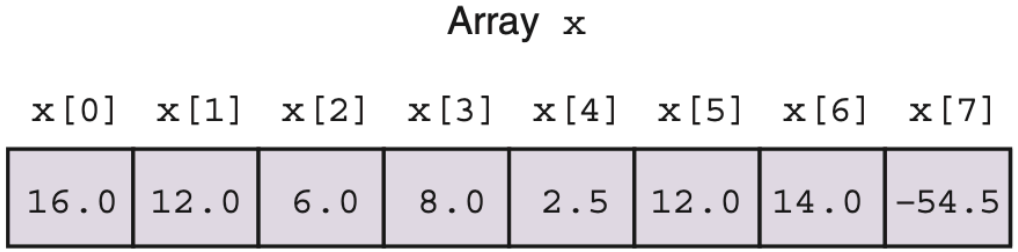
Array `x`

x[0]	x[1]	x[2]	x[3]	x[4]	x[5]	x[6]	x[7]
16.0	12.0	6.0	8.0	2.5	12.0	14.0	-54.5

`double x[8]`

- An array of `double`
- 8 elements: `x[0],x[1],...,x[7]`

```
double x[8];
```

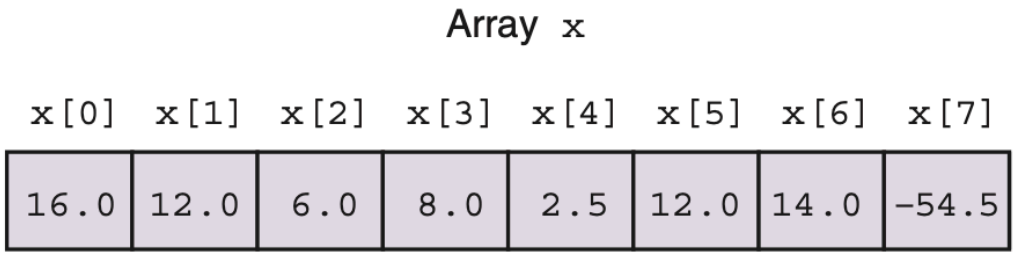


- Each element is a variable of int
- Can be used at the left side or right side of an assignment operator

```
x[2] = x[1]+6;
```

The one at the left is called left value

```
double x[8];
```




```c
printf("%.1f", x[0]);
x[3] = 25.0;
sum= x[0] +x[1];
sum += x[2];
x[3] += 1.0;
x[2] = x[0] + x[1];
```

Element数组的单元

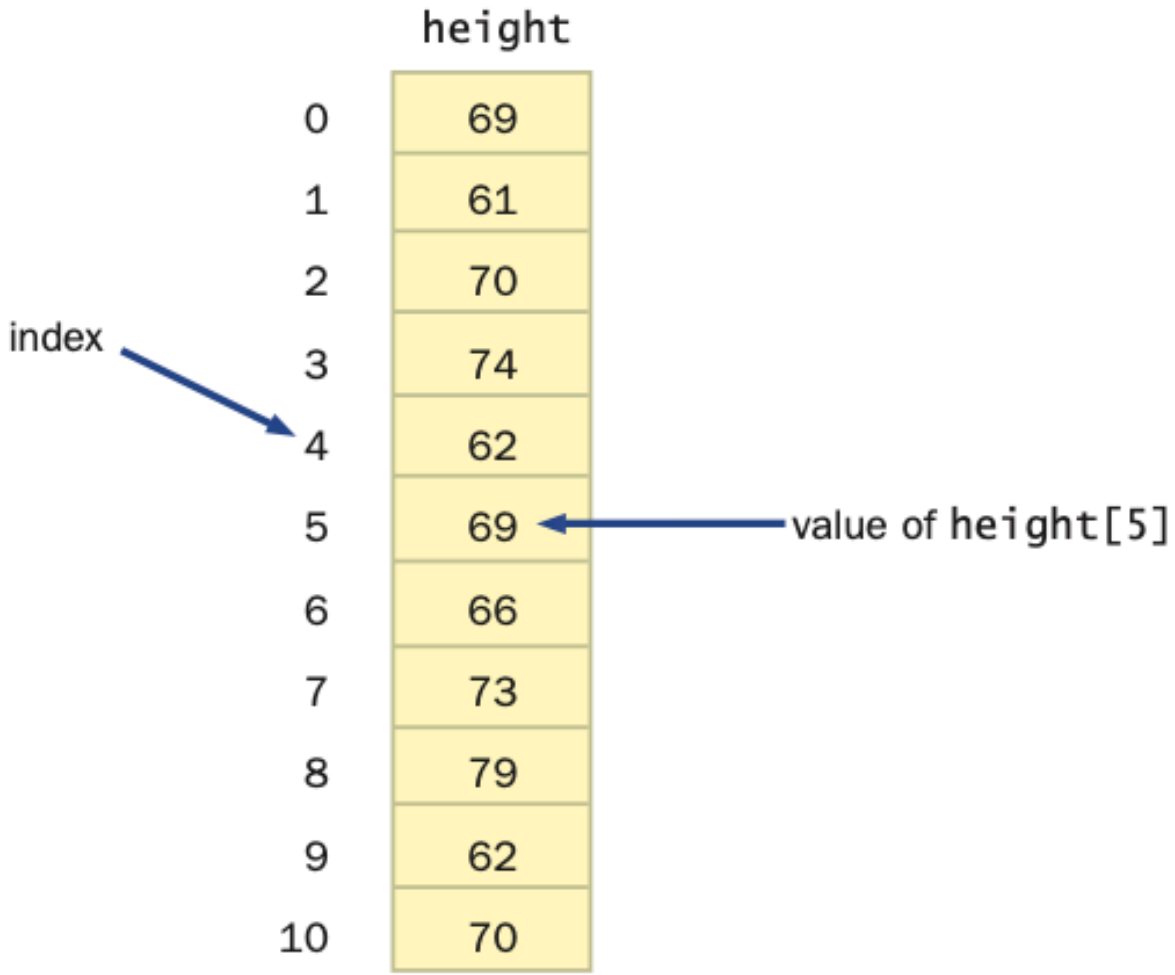
- Each element is a variable of the type of that array
- The number in `[]` is called 下标subscript or 索引index, that begins from 0:

```
grades[0]
grades[99]
average[5]
```

- each element is a variable of the type
- index or subscript starts from 0

```
grades[0]
grades[99]
average[5]
```

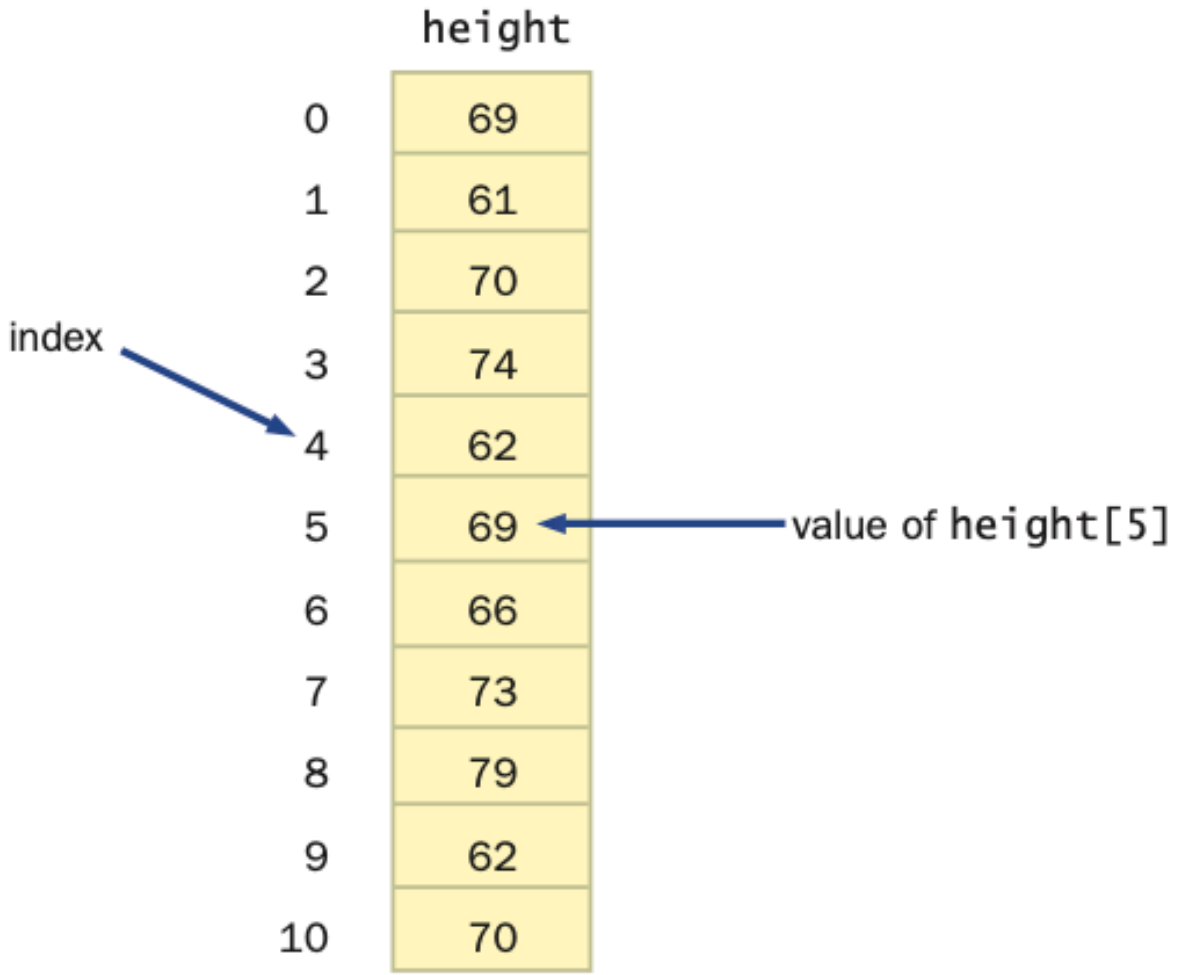
> Key Concept: An array of size N is indexed from 0 to N-1



Quick check IV

- Based on the array shown at right, what are each of the following?

1. value of `height[5]`
2. `height[2] + height[5]`
3. `height[2 + 5]`
4. the value stored at index 8
5. the fourth value



Valid range of index

- Either the compiler or the run-time is not going to check wether the index is out-of-range or not, whatever at reading from or writing to an element.
- In run-time, it is possible the out-of-range index causes severe program and makes the program crashed.
    - **segmentation fault**
- However it is possible nothing serious happens at good luck. However, it is still dangerous.
- A program can compile —> may not be a program can run.
- A program can run this time —> may not be a program runs every time.
- It is your responsibility to make sure your program uses index with the range: `[0, the size of the array -1]`

Average

- Using C99, it is possible that the use inputs the number of numbers first.

```
int x;
double sum = 0.0;
int n;
scanf("%d", n);
if ( n>0 ) {
    int number[n];
    for ( int i=0; i<n; i++ ) {
        scanf("%d", &number[i]);
    }
}
```

Lab 1

PTA 6-1 大于平均的身高

Aggregate initialization

```
int a[] = {2,4,6,8,10,12,14,16,18};
```

- Put all the init values in `{}`
- The size is not needed, because the compiler can count for you

```
int b[20] = {2};
```

- However if the size, but not all the init value provided, the rest of the elements will be init with 0
- It does not work for array sized with a variable

集成初始化时的定位

```
int a[10] = {
    [0] = 2, [2] = 3,6
}
```

- 用[n]在初始化数据中给出定位 没有定位的数据接在前面的位置后面
- 其他位置的值补零
- 也可以不给出数组大小，让编译器算
- 特别适合初始数据稀疏的数组

Size of an array

- `sizeof` returns the number of bytes the whole array occupies

```
sizeof(a)/sizeof(a[0])
```

- `sizeof(a[0])` returns the size of one single element, so a division gives the number of elements
- Code like this does not need to be modified when the init data for array is to be changed
- It is called "extensiblity"(可扩展性)

Assignment of arrays

```
int a[] = {1,2,3,4,5};
int b[] = a;    //  ERROR!
```

- Array variable can not be assigned.
- To assign all elements of one array to another, an iteration is a must

Basic Operation of Arrays

Iteration of an array

```
for ( int i=0; i<n; i++ ) {
    scanf("%d", &number[i]);
}
```

- The `for` loop is usually used to let `i` from `0` to less than the size of the array. in this way, the biggest `i` in the loop is the biggest valid index
- Common pitfalls:
  - The condition for the loop is : `<= size of the array`, or
  - `i` is to be used as an index after loop!

Max or min

- Use the first element as the seed for min or max

```
int findmin(int a[], int n) {
    int min=a[0];
    for ( int i=1; i<n; i++) {
        if ( a[i]<min ) {
            min = a[i];
        }
    }
    return min;
}
```

Index of the min

```
int findminidx(int a[], int n) {
    int minidx=0;
    for ( int i=1; i<n; i++) {
        if ( a[i]<a[minidx] ) {
            minidx = i;
        }
    }
    return minidx;
}
```

Lab 2

PTA 6-2 最小值下标

Insert and remove

- Array is a fixed-size container
- Inserting and removing data inside an array, is to move data in the elements

Book IDs

Tom has an array that stores the IDs of 5 books:

```
int books[10] = {101, 103, 104, 105, 108};  // aggregate initialization
int n = 5;   // current number of elements
```

Now he found a new book with ID `102`, and he wants to insert it **between `101` and `103`**.

Solution

```
int books[10] = {101, 103, 104, 105, 108};
int n = 5;  // current number of elements
int newBook = 102;
int pos = 1;  // insert at index 1 (before 103)

// Shift elements to the right
for (int i = n; i > pos; i--) {
    books[i] = books[i - 1];
}

// Place the new value
books[pos] = newBook;
n++;
```

Insertion in an array

- Insertion requires **shifting the elements** to make space
- The larger the array, the more shifting is needed
  - insertion is not always efficient
- There have to be enough space to hold the new element

Another books ID story

Tom has an array that stores the IDs of 6 books:

```
int books[10] = {101, 102, 103, 104, 105, 108}; // aggregate initialization
int n = 6;   // current number of elements
```

Now he decides to remove the book with ID `103`.

Solution

```
int books[10] = {101, 102, 103, 104, 105, 108};
int n = 6;  // current number of elements
int pos = 2; // index of element to delete (103 is at index 2)

// Shift elements to the left
for (int i = pos; i < n - 1; i++) {
    books[i] = books[i + 1];
}

n--;  // one element is removed
```

Deletion in an array

- Insertion = shift elements **to the right**.
- Deletion = shift elements **to the left**.

Lab 3

PTA 6-3 删除数据

Prime numbers, again

- 从2到x-1测试是否可以整除

```
int isprime = 1;
for ( int i=2; i<x; i++ ) {
    if ( x%i==0 ) {
        isprime = 0;
        break;
    }
}
```

- 对于n要循环n-1遍 当n很大时就是n遍

- 无须到`x-1`，到`sqrt(x)`就够了

```
#include <math.h>

...

int isprime = 1;
for ( int i=2; i<sqrt(x)+1; i++ ) {
    if ( x%i==0 ) {
        isprime = 0;
        break;
    }
}
```

- $\sqrt{x} << x$

判断是否能被已知的且$<x$的素数整除

- 所有的偶数都不是素数
- 如果一个数能够被3整除，那么一定能被6、9、12这些3的倍数整除
- 所以，不需要判断$[2,\sqrt{x}]$的数能否整除，只需要判断$[2,x)$之间所有的素数能否整除
- 这种方法适合构造n以内的素数表，不适合判断单个自然数是否是素数

```c
int n;
scanf("%d", &n);
int prime[n]; //  known primes
int nprime = 0; //  number of known primes
for ( int x=2; i<n; i++ ) {
    if ( isPrime(x,prime,nprime) ) {
        prime[nprime++] = x;
    }
}
```

```c
int isPrime(int x, int prime[], int nprime)
{
    int isprime = 1;
    for ( int i = 0; i<nprime; i++ ) {
        if ( x%prime[i] == 0 ) {
            isprime = 0;
            break;
        }
    }
    return isprime;
}
```

- `nprime` 起到了两个作用：
  i. 已经放进去了几个素数；
  ii. 下一个素数放哪里

产生素数表

- 合数都可以被分解为素数的乘积

> 14=2*7
>
> 39=3*13
>
> 65=5*13

- 因此素数的任意倍数一定都不是素数

To construct a prime table under n

1. let $x = 2$
2. mark $2x$, $3x$, $4x$ until $ax < n$ as non-prime
3. make x the next non-marked number, go back to 2, until all the numbers have beed tried

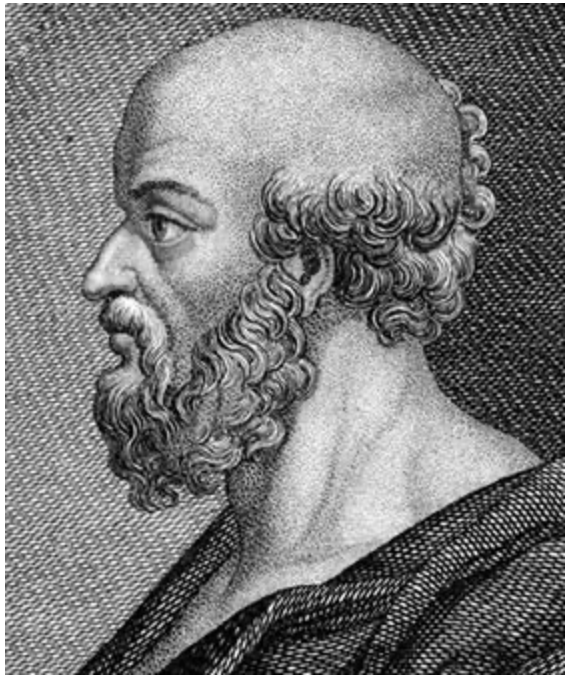
构造素数表

- 欲构造n以内(不含)的素数表

1. 开辟`prime[n]`，初始化其所有元素为1，`prime[x]`为1表示`x`是素数
2. 令`x=2`
3. 如果`x`是素数，则对于`(i=2;x*i<n;i++)`令`prime[i*x]=0`
4. 令`x+=1`，如果`x<n`，重复3，否则结束

> 算法不一定和人的思考方式相同

Sieve of Eratosthenes



- Sieve of Eratosthenes is an ancient algorithm for finding all prime numbers up to any given limit

Function, revised

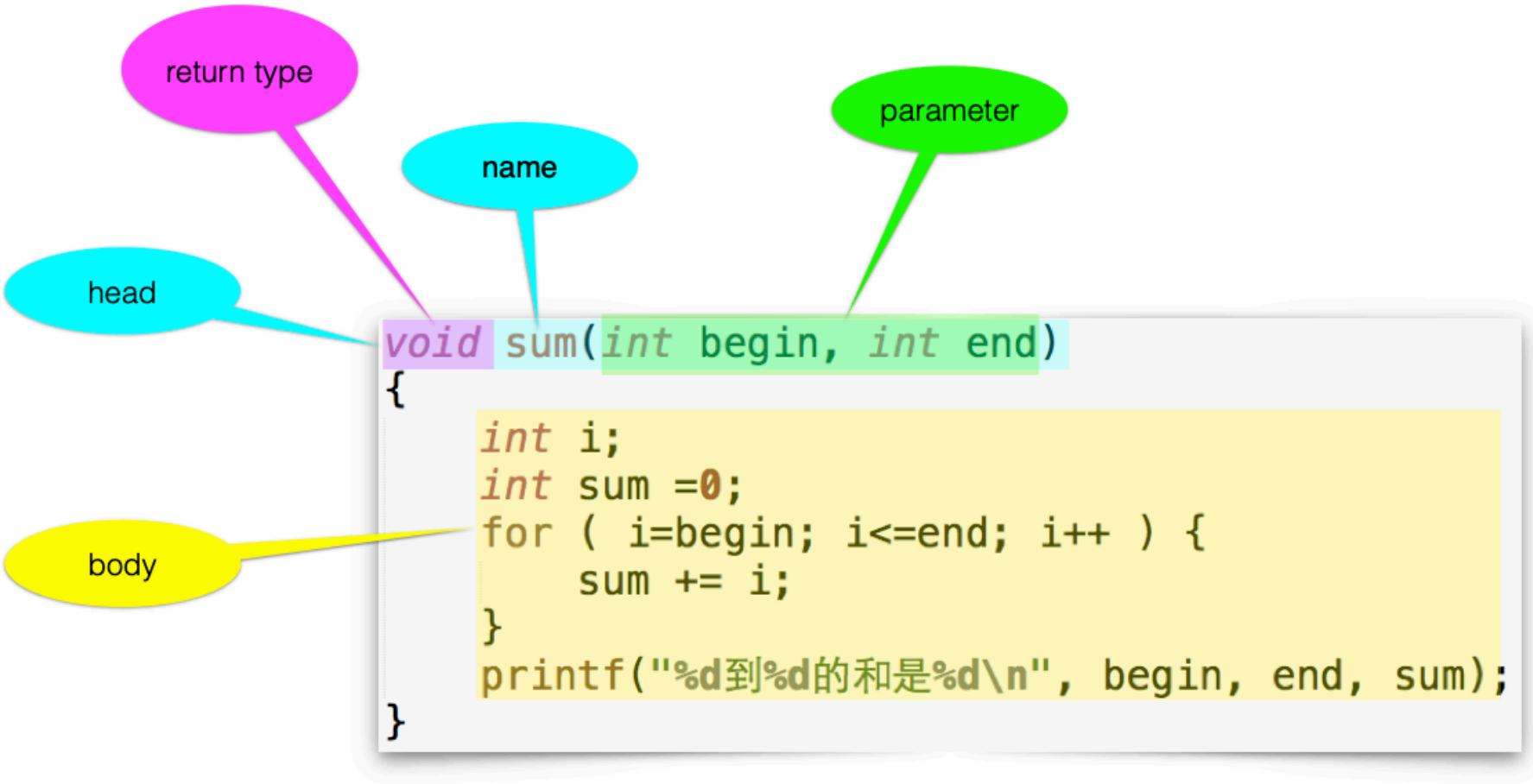
Function

- Function is a piece of code, which takes zero or many parameters, does one thing and returns zero or one value
- It is something like the function:

$$y = f(x)$$

- It represents a value

Definition



Calling a function

```
function_name(parameter values);
```

- `()` is the evidence of a function call
  `()` is needed when there is no parameter at all
- Values should be provided in correct order and number
- These values will be used to initialize the parameters respectively

to return from a function

- The function knows the place it is called, and returns to the right place to

Return a value back

```
int max(int a, int b) {
    int ret;
    if ( a>b ) {
        ret = a;
    } else {
        ret = b;
    }
    return ret;
}
```

- `return` ceases the execution of the function, and sends back a value

  `return;`

  `return <expression>;`
- There may be more than one `return` in a function

The value returned

```
int a=5, b=6, c;
c = max(10,12);
c = max(a, b);
c = max(c, 23);
c = max(max(c, a), 5);
printf("%d", max(a, b));
max(12,13);
```

- Can be assigned to a variable
- Can be passed to a function
- Can be dropped off
- Sometimes it is the side-effect that we want from the function

Function without return value

```
void function_name(parameter list)
```

- There is nothing to be returned
    - No `return` is possible
- It does NOT have a value but does have side-effect
- Can not be *right value* at calling
- A return with value must be used when the function has a type.

Value and side-effect of a function

- A function may or may not return a value
- A function may or may not have side-effect
  - A function returns a value may have side-effect as well
  - A function returns nothing usually has side-effect
- Functions have no side-effect are reenterable functions that
  - do not call any non-reenterable functions, and
  - do not access any outside resource
- 可重入函数的值与外界环境无关，也不会改变外界环境

Functions we've learned

- Function that takes one argument and returns one value
- Function takes more than one argument
- Function returns nothing

What we've learned today?

- 语言
  - do-while loop
  - array
  - `sizeof()`
  - function, the `void`
- 算法
  - min/max
  - sieve of Eratosthenes