Class 5 Control Flow (2)

Loops in C Program

Some Problems

if (month == 1 || 3 ||5 ||7 ||8 || 10 || 12) ERROR

if (month == 1 || month == 3 || month == 5 || CORRECT
month ==7 || month ==8 || month ==10 || month ==12)

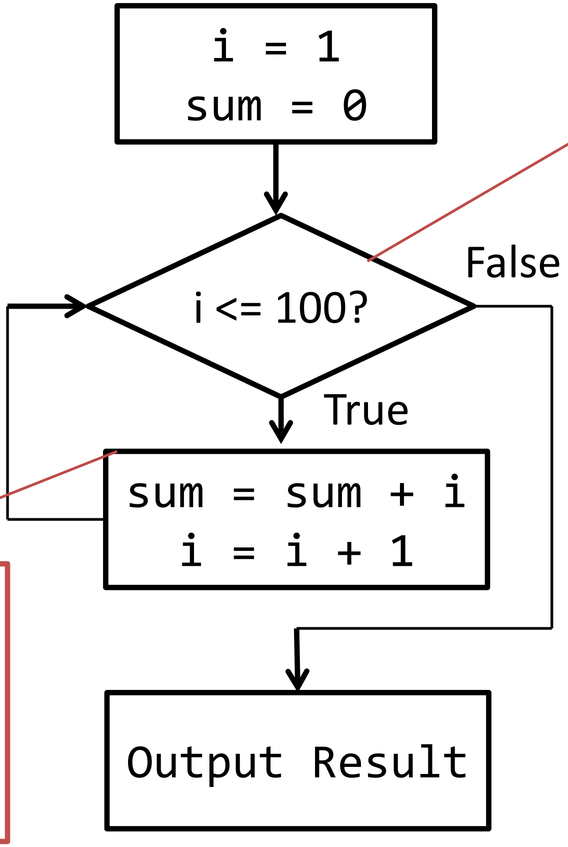
```
float s, p;
scanf(“%f”, &s);

switch( (int)(s/100) )
{
    case 0 :  p = 30.0f; break;
    case 1 :  p = 27.5f; break;
    case 2 :  p = 25.0f; break;
    case 3 :  p = 22.5f; break;
    default:  p = 20.0f;
}
```

How does a programmer control the C program?

- A programmer can control the execution of a C program by using three kinds of control structures
  - Sequence Structure (顺序结构)
  - Selection Structure (选择结构)
    - if statement
    - if-else statement
    - if-elseif-else statement
    - switch statement
  - Repetition Structure (循环结构)
    - Make the computer to do labor-intensive jobs by using a loop

An Example of A Loop

- Calculate the summation of integers that are from 1 to 100



i = 1
sum = 0

i <= 100?

False

True

sum = sum + i
i = i + 1

Output Result

Test Condition:

The loop circle is continue until the condition is not satisfied

Body of the loop

Do some labor-intensive jobs repeatedly

The Loop

- Repeat to do something when some conditions are satisfied
- Two Important Aspects to Design a loop
  - Conditions
  - Repeated Actions
- Three Looping Statements in C Language
  - while statement
  - do-while statement
  - for statement

while statement

Testing Condition:

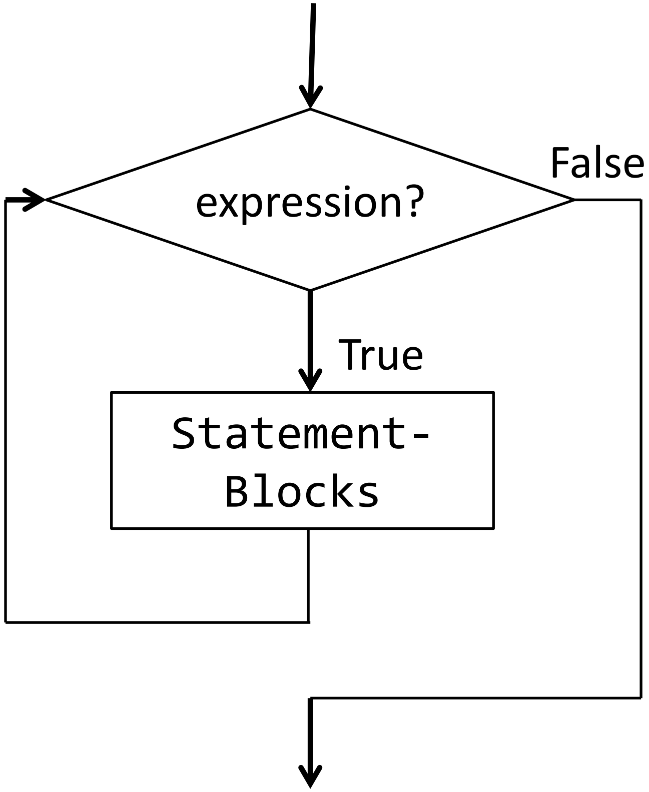
The loop circle is continued
if the expression gives TRUE;
otherwise the loop is finished.

```
while (expression)
{
        Statement-Block;
}
```

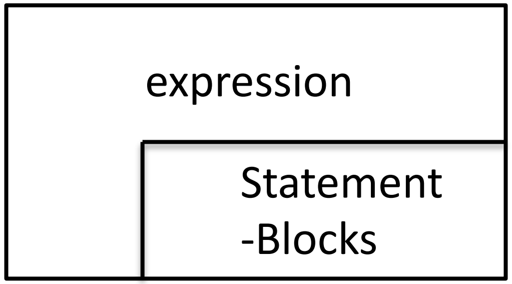
Body of the Loop:

The statement-blocks are repeated
until the expression gives FALSE

Flow Chart of while statement

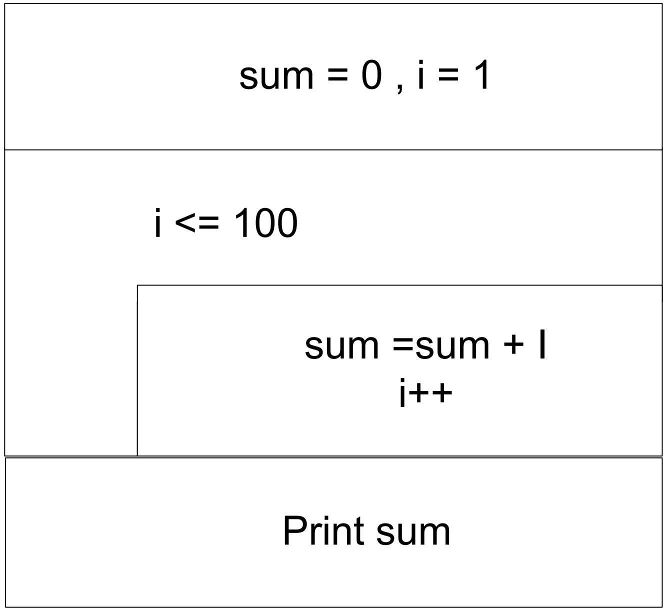


Flow Chart of while statement



Example of while statement

```c
#include <stdio.h>

int main()
{
    int sum = 0, i = 1;

    while ( i <= 100 )
    {
        sum = sum + i;
        i++;
    }

    printf(“sum = %d\n”, sum)

}
```



sum = 0 , i = 1

i <= 100

sum =sum + I
i++

Print sum

the loop body can also be written as follows

```c
sum += i;
i++;
```

OR

```c
sum += i++;
```

do-while statement

Body of the Loop:

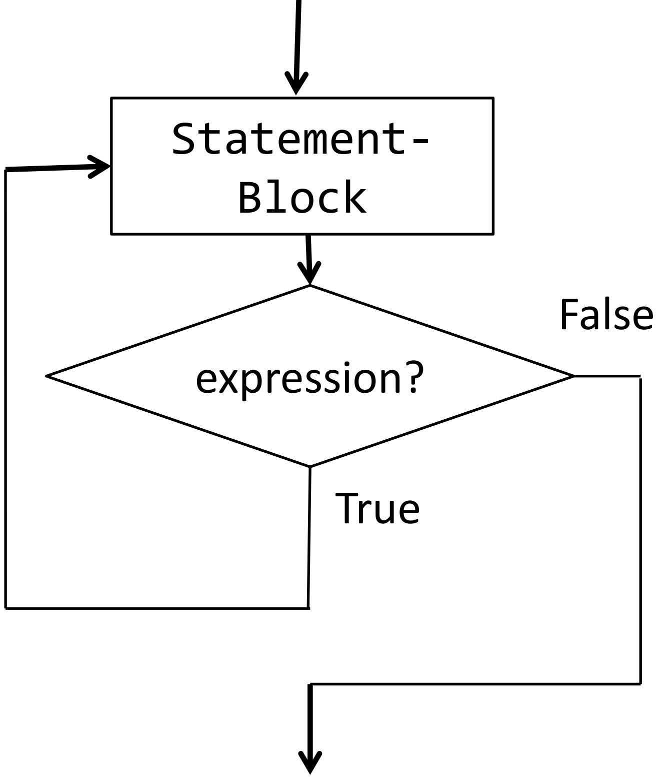
The statement-blocks are repeated until the expression gives FALSE

```
do
{
      Statement-Block;
} while (expression);
```

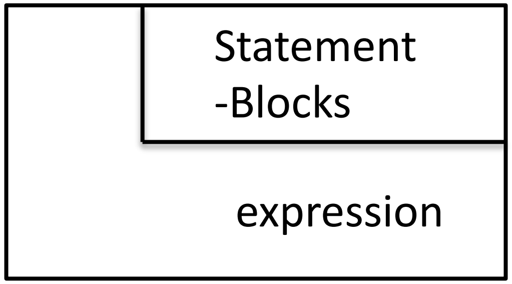
Testing Condition:

The loop circle is continued
if the expression gives TRUE;
otherwise the loop is finished.

Flow Chart of while statement

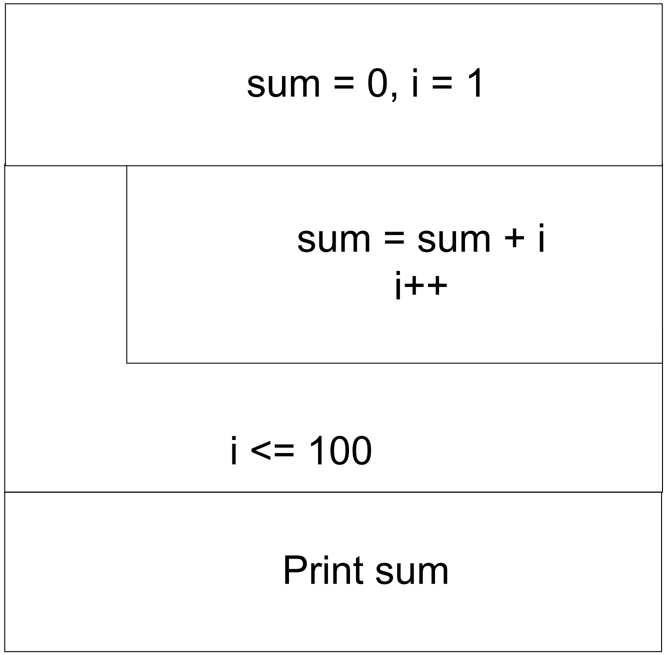


Flow Chart of while statement



Example of do-while statement

```
#include <stdio.h>

int main()
{
    int sum = 0, i = 1;

    do
    {
        sum = sum + i;
        i++;
    }while ( i <= 100 );

    printf(“sum = %d\n”, sum)

}
```



sum = 0, i = 1

sum = sum + i
i++

i <= 100

Print sum

the loop body can also be written as follows

```
sum += i;
i++;
```

OR

```
sum += i++;
```

Notes of while and do-while statement

- while statement
  - pre-test control: judge the condition before execute the loop body
  - it is possible that the loop body has never been executed
  - DO NOT ADD the semicolon after while statement

```
while ( i <= 100 ); // ERROR
{
    sum = sum + i;
    i++;
}
```

if this semicolon is added
the loop circle will not end

Notes of while and do-while statement

- do-while statement
  - post-test control : execute the loop body before judge the condition
  - the loop body is executed at least one time
  - Do NOT MISS the semicolon in the do-while statement

```
do
{
    sum = sum + i;
    i++;
}while ( i <= 100 );
```

if this semicolon is missed compiling error occurs

for statement

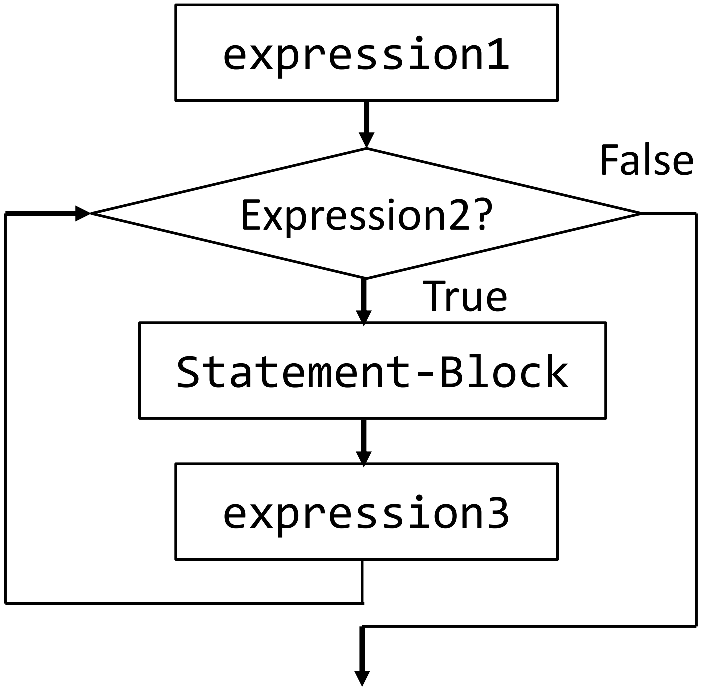
initialization

test condition

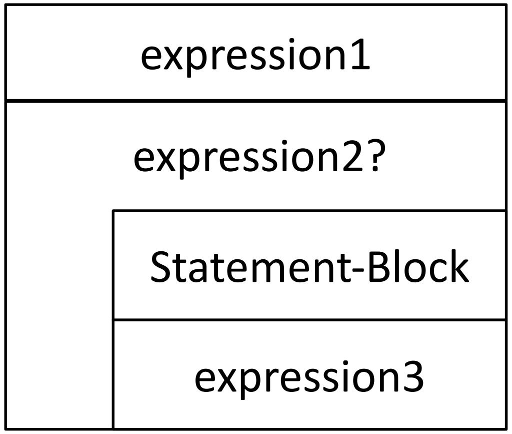
Increment/decrement

```
for (expression1 ; expression2 ; expression3)
{
      Statement-Block;
}
```

Flow-Chart of For Statement



Flow-Chart of For Statement



Example of For Statement

```
int sum =0, i;
for (i=1; i<＝100; i++)
      sum += i;
```

/*Initialization*/

/*Testing condition*/

/*Actions*/

/*Updating*/

```
int sum =1, i;
for (i=2; i<＝100; i++)
      sum += i;
```

```
int sum =0, i;
for (i=1; i<＝100;sum +=  i++) ;
```

Notes of the for statement

- Any of the three expression within the parentheses of the for statement can be omitted.

(1)
```
int  i =1 ;    //Taking out initialization
for ( ;  i <= n ;  i ++  )
{  sum = sum  + i ; }
```

(2)
```
for ( i =1;  ;  i ++ )
   {  sum = sum  + i ;
      if ( i > n)  break ;
   }
for (i=1; 1; i++)
```

(3)
```
for ( i =1;  i <= n ; )
 {  sum = sum  + i ;
       i + + ;
 }    //Taking out updating
```

(4)
```
i = 1;
for  (  ;  ;  )
{  sum + = i + + ;
   if  ( i > n ) break ;
}   //all are absent
```

(5)
```
for ( i =1;
        sum += i ++ ,  i <= n ; )  ;
        //comma operator
```

Semicolons appear all the time.

Comparison of while, do-while and for statement

- Initialization
  - variable should be initialized before the while/do-while part
  - variable can be initialized in expression1 of for statement
- Body of the Loop
  - the statement that can terminate the loop should be within the body of the loop for while/do-while statement
  - expression3 in for statement contain the part that is able to terminate the loop
- Type of the Loop
  - Entry Control : test condition before the repeated action
    - while, for statement
  - Exit Control : do the repeated action before test the condition
    - do-while statement

Exercise

- Write a program to evaluate the equation $y=x^n$ when n is a non-negative integer using for, while and do-while independently.

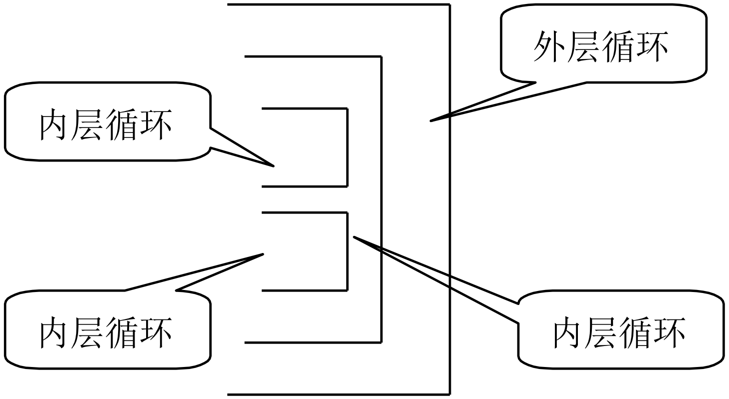
```
#include <stdio.h>

int main()
{
    int i, n;
    float x, y;
    scanf(“%d”, &n);
    scanf(“%f”, &x);

    i = 0; y = 1.0f;
    while( i < n )
    {
        y *= x;
        i++;
    }

    printf(“%f\n”, y);
    return 0;
}
```

```
#include <stdio.h>

int main()
{
    int i, n;
    float x, y;
    scanf(“%d”, &n);
    scanf(“%f”, &x);

    i = 0; y = 1.0f;
    do
    {
        y *= x;
        i++;
    }while( i < n );

    printf(“%f\n”, y);
    return 0;
}
```

```
#include <stdio.h>

int main()
{
    int i, n;
    float x, y;
    scanf(“%d”, &n);
    scanf(“%f”, &x);

    for (i=0, y=1.0f;
         i < n;
         i++)
    {
        y *= x;
    }

    printf(“%f\n”, y);
    return 0;
}
```

Nest of loops

- General form



合法的嵌套循环

- An example of nest of for loops

```
for ( i = 0 ; i < 10 ; i++ )
{
    ......
    for ( j = 0 ; j < 5 ; j++ )
    {
        ......
    }
    ......
}
```

Outer Loop

Inner Loop

Examples of nested loops

```
(1) while( )
    {…
      while( )
      {…}
    }
```

```
(2) do
    {…
      do
      {… } while( );
    } while( );
```

```
(3) for( ; ; )
    {
      for( ; ;)
      {… }
    }
```

```
(4) while( )
    {…
      do{…}
      while( ) ;
    }
```

```
(5) for(;;)
    {  …
      while( )
      {  }
    }
```

```
(6) do
    {  …
      for(;;)
      {  … }
    } while( );
```

An example

- ➢ Write a program to print the multiplication table from 1×1 to 12×10 as shown below is given:

```
 1   2   3   4   5   6   7   8   9  10
 2   4   6   8  10  12  14  16  18  20
 3   6   9  12  15  18  21  24  27  30
 4   8  12  16  20  24  28  32  36  40
 5  10  15  20  25  30  35  40  45  50
 6  12  18  24  30  36  42  48  54  60
 7  14  21  28  35  42  49  56  63  70
 8  16  24  32  40  48  56  64  72  80
 9  18  27  36  45  54  63  72  81  90
10  20  30  40  50  60  70  80  90 100
11  22  33  44  55  66  77  88  99 110
12  24  36  48  60  72  84  96 108 120
```

1×1 1×2 1×3 … 1×10
2×1 2×2 2×3 … 2×10
3×1 3×2 3×3 … 3×10

12×1 12×2 12×3 … 12×10

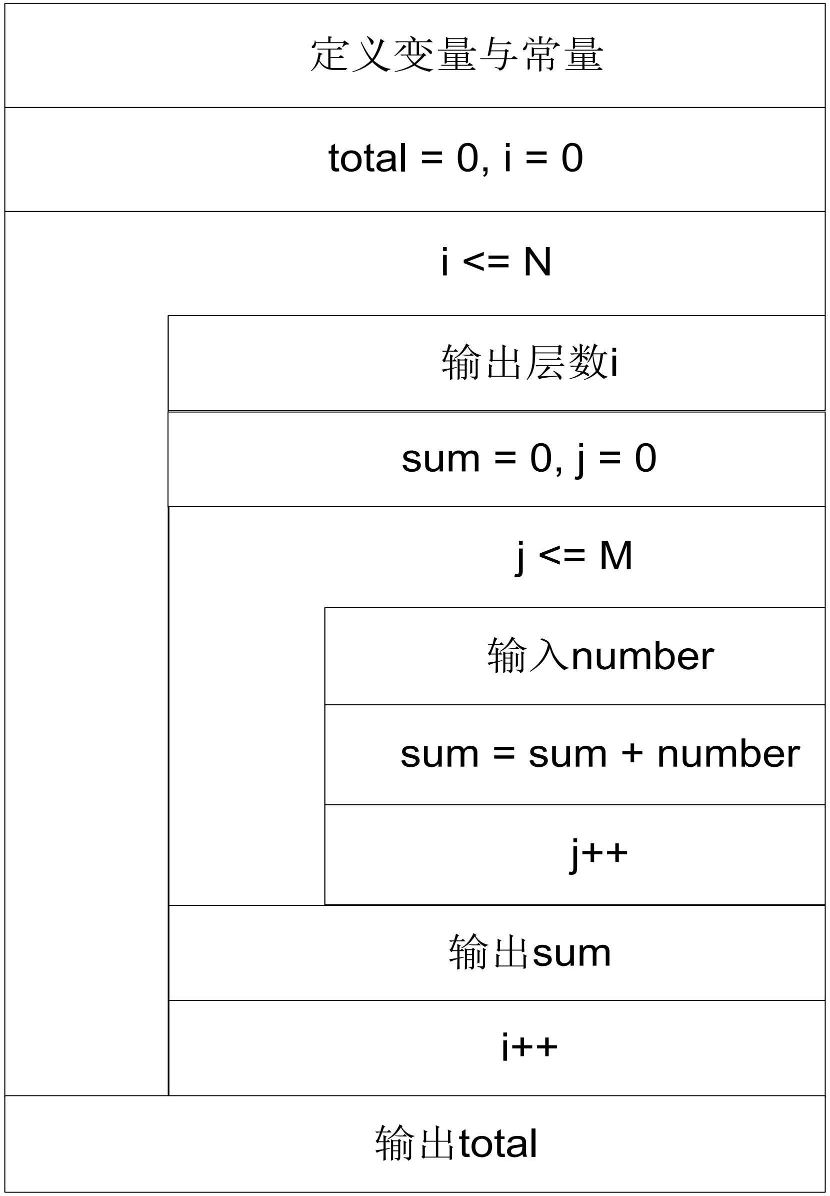
We may calculate row by row → a loop
Each row is composed of repeating multiplications → a loop

```
#define ROWMAX 12
#define COLMAX 10

int main()
{
    int row, column, y;

    for( row = 1 ; row <= ROWMAX ; row++ )
    {
        for( column = 1 ; column <= COLMAX ; column++ )
        {
            y = row * column;
            printf( “%4d” , y);
        }

        printf( “\n” );
    }

    return 0;
}
```

Exercise

```
#define N 3
#define M 6
```

- Residents live in a building with 3 floors, each of which has 6 rooms. Input the number of persons in each room from the keyboard, count the number of persons in each floor and the total number in this building.
  - Inner loop for ??
  - Outer loop for ??

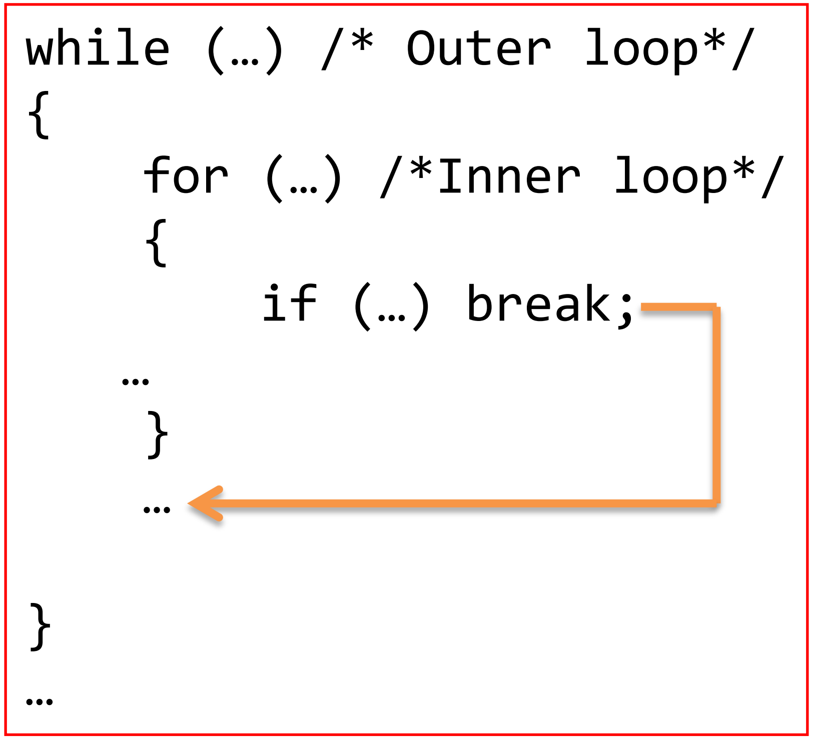


```
定义变量与常量
total = 0, i = 0
i <= N
    输出层数i
    sum = 0, j = 0
    j <= M
        输入number
        sum = sum + number
        j++
    输出sum
    i++
输出total
```

```
#include <stdio.h>
#define N 3
#define M 6

int main()
{
    int i, j, sum, total = 0, number;
    for ( i = 1 ;  i <= N ; i++ ) /* Outer loop*/
    {
        printf( “The %dth floow”:\n" , i);
        sum = 0 ;
        for ( j = 1 ; j <= M ; j++ ) /* Inner loop*/
         {
             printf(“How many people in the %dth room?:",j);
             scanf("%d",&number);
             sum += number;
         }
        printf( ”There are %d people in this floor\n",sum );
        total += sum;
    }
    printf( ”There are %d people in this building.\n",total);
    return 0;
}
```

Jumps in Loops

- break
  - Jump out of a loop



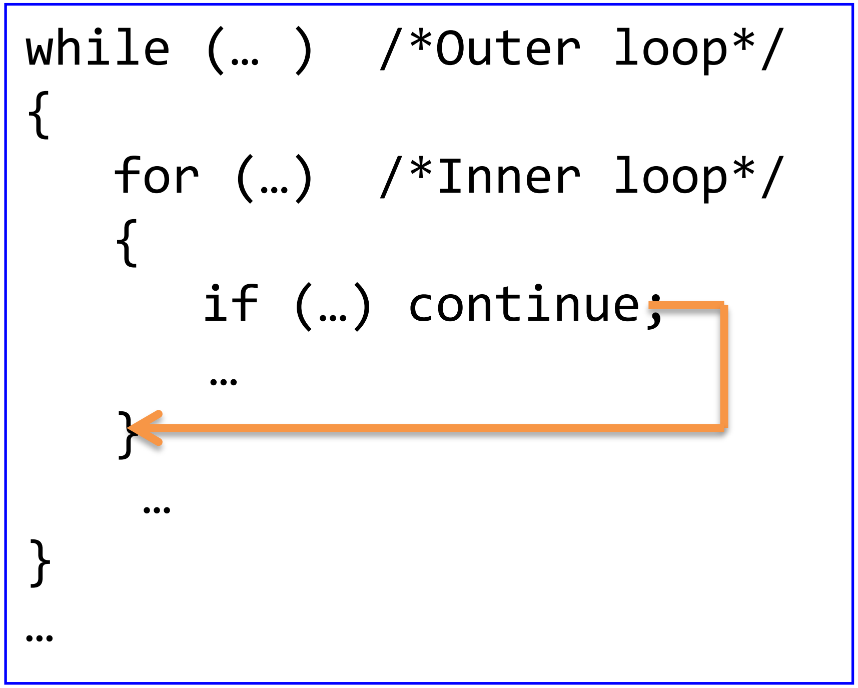
```
while (…) /* Outer loop*/
{
    for (…) /*Inner loop*/
    {
        if (…) break;
   …
    }
    …

}
…
```

An Example of Break

```
// quit system
while(1)
{      …
       printf( “quit? (y / n): ”);
       scanf(“%c”, &ch);
       if ( ch==‘y’ ) break;
}
```

Jumps in Loops

- continue
  - Skip the current iteration of a loop



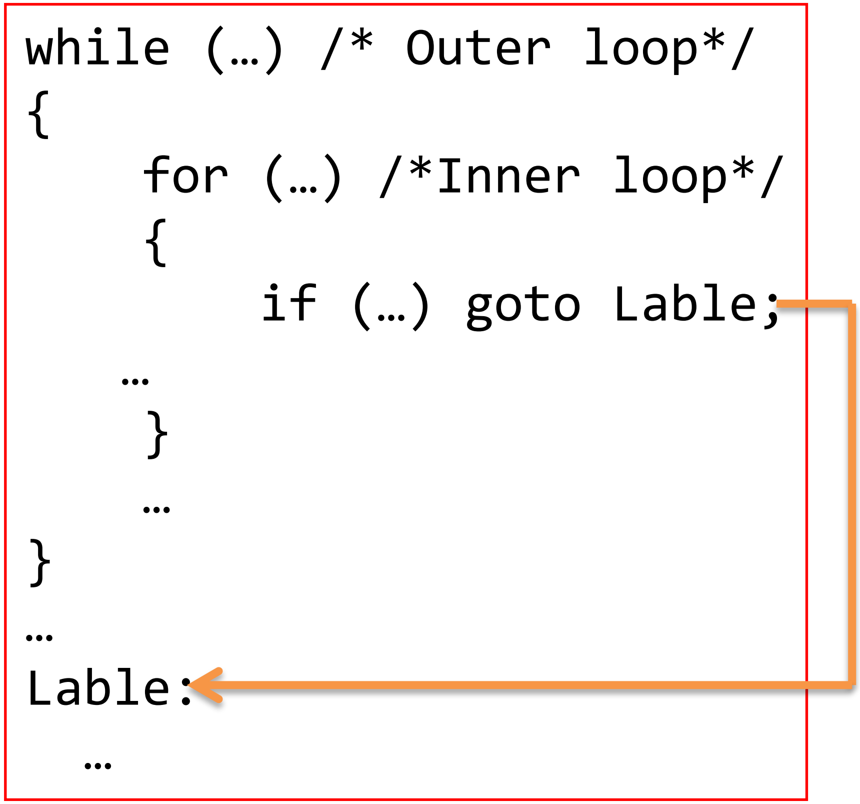
```
while (… )  /*Outer loop*/
{
   for (…)  /*Inner loop*/
   {
      if (…) continue;
      …
   }
    …
}
…
```

An Example of Continue

```
for ( i = 1 ; i < 100 ; i++ )
{
    if ( i%3 != 0 )
        continue;
    printf(“i = %d\n”, i);
}
```

Jumps in Loops

- goto
  - It is able to break out the nested loop and make the program directly transferred to a certain place
  - not recommended



```
while (…) /* Outer loop*/
{
    for (…) /*Inner loop*/
    {
        if (…) goto Lable;
   …
    }
    …
}
…
Lable:
  …
```

Example

- Write a program to test if an input natural number is a prime number(素数)
  - Prime number: a natural number greater than 1 that has no positive divisors other than 1 and itself.
- The key idea
  - Try all natural numbers other than 1 and itself (m)
  - Test if the numbers are the divisors of m

```
#include <stdio.h>

int main()
{
    int m, i;
    scanf(“%d”, &m);

    for ( i = 2 ; i < m ; i++ )
        if ( m%i == 0 ) break;

    if ( i == m )
        printf(“%d is a prime.\n”, m);
    else
        printf(“%d is not a prime.\n”, m);

    return 0;
}
```

A slight improvement

- Mathematics

Suppose m has a divisor *i* and *j* $m = i \times j$

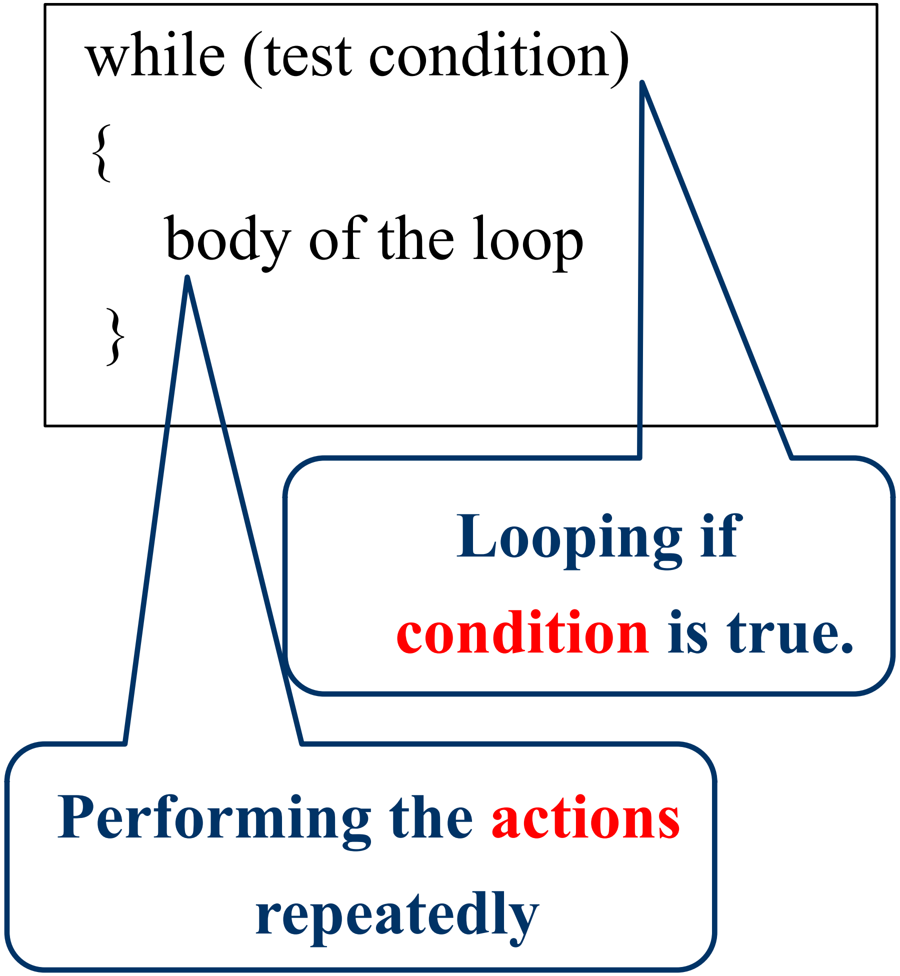
and assume $i \leq j$

so we get $i \leq m / i \longrightarrow i \leq \sqrt{m}$

We only need to search the range between 2 and the square root of m

```
#include <stdio.h>
#include <math.h>

int main()
{
    int m, i, k, flag;
    scanf(“%d”, &m);

    flag = 1;
    k = sqrt(m);
    for ( i = 2 ; i < k && flag ; i++ )
        if ( m%i == 0 ) flag = 0;

    if ( flag )
        printf(“%d is a prime.\n”, m);
    else
        printf(“%d is not a prime.\n”, m);

    return 0;
}
```

Issues to design a loop



```
while (test condition)
{
    body of the loop
 }
```

Looping if condition is true.

Performing the actions repeatedly

These two are the core of designing a loop.

Guides for Loop control

- Generally, there are two ways to control a loop
  - Counter control
    - We may use a counter when we know the number of iterations beforehand (*a priori)* .
    - We have to initialize the counter before a loop and update the counter in a loop, and its value is kept out of a loop.
    - Examples: summation of series with fixed length, power of N, and prime test etc.
  - Sentinel control
    - Use an indicator as the loop condition.
    - The value of the indicator may be ‘abnormal’.
    - Typical usage is to exit a loop by user interactions.

```
while ( (c = getchar()) != ‘\n’ )
{
     ......
}
```

    - Sometimes we may use both (counter and sentinel), e.g., many scientific computing algorithm.

Guides for selecting a loop

- Analyze the problem and see whether it required a pre-test or post-test loop.
  - If it requires a post-test loop, we can only use do-while.
  - If it requires a pre-test loop, we have two choices: for and while.

- Decide whether the loop termination requires counter-based control or sentinel-based control.
  - Use for loop if the counter-based control is necessary.
  - Use while loop if the sentinel-based control is required.
  - Note that both the counter-controlled and sentinel-controlled loops can be implemented by all the three control structures.

Examples to Use a Loop

- Exhausted search
  - Try all possibilities
  - No apparent updating patterns
- Examples
  - Solve the following equations (x, y and z are integers)

$$\begin{cases} x + y + z = 100 \\ 5x + 3y + z / 3 = 100 \end{cases}$$

subject to $x > 0, y > 0, z > 0, z\%3 = 0$

```c
#include <stdio.h>
int main()
{
    int x,y,z;
    for (x=1; x<100; ++x)
    {
        for (y=1; y<100; ++y)
        {
            for (z=3; z<100; z+=3)
            {
                if( 5*x+3*y+ z/3==100 && x+y+ z==100 )
                     printf(”x=%d,y=%d,z=%d\n", x, y, z);
            }
        }
    }
    return 0;
}
```

Use Less Loops to Improve Efficiency

```
#include <stdio.h>
int main()
{
    int x,y,z;
    for (x=1; x<20; ++x)
    {
        for (y=1; y<33; ++y)
        {
             z = 100 – x – y;
             if( 5*x+3*y+ z/3==100 && x+y+ z==100 && z%3 == 0)
                 printf(”x=%d,y=%d,z=%d\n", x, y, z);
        }
    }
    return 0;
}
```

Examples

- Calculate the greatest common divisor (GCD 最大公约数) of two positive integers *m* and *n*

- Idea of Exhausted Search
  - Find the maximum integer i between [1, min(m,n)] that satisfies (m%i == 0 && n%i == 0)

```
for (i = min(m,n) ; i >1 ; i--)
    if ( m % i == 0 && n % i == 0 )
        break;

GCD = i;
```

```c
#include <stdio.h>
int main()
{
    int m, n, temp, i;
    scanf("%d %d", &m, &n);

    if ( m > n )
    {
        temp = m; m = n; n = temp;
    }
    printf("m = %d, n = %d\n", m, n);

    for ( i = m ; i > 1 ; i-- )
        if ( m % i == 0 && n % i == 0 )
            break;
    if ( i == 1 )
        printf("No GCD\n");
    else
        printf("GCD = %d\n", i);
    return 0;
}
```

Examples to Use a Loop

- Iteration
  - Induction process → From n to n+1
  - We can clearly define the updating process
- Examples
  - Calculate sin(x) by using the following Taylor equation
    - Condition : the last term in Taylor equation should be less than 1e-6

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

Idea for Programming

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

$$a_n = (-1)^{n-1} \frac{x^{2n-1}}{(2n-1)!} = -a_{n-1} \frac{x^2}{(n+1)(n+2)}$$

```
term = x, sum = 0;
while ( |term| >= 1e-6 )
{
    sum += term;
    calculate the next term;
}
```

```
#include <stdio.h>
#include <math.h>
#define PI 3.1415926

int main()
{
    double degree, x, sum, term;
    int n = 1;

    scanf(“%lf”, &degree);
    x = degree * PI / 180.0;

    sum = x; term = x;
    while( fabs(term) >= 1e-6 )
    {
        term = -term * x * x / ((n+1) * (n+2)) ;
        sum += term;
        n += 2;
    }

    printf(“sin(%.2f) = %.5f\n”, x, sum);

    return 0;
}
```

Loops summary

- Designing a loop
  - Condition
  - Repeated actions
- Implementing a loop
  - Initialization
  - Condition
  - Updating
  - Action

- Repetition Structures in C Language
  - while statement
  - do-while statement
  - for statement

Tips for efficient programs

- Narrow the searching range (prime number, reverse, chicken/dollar)

- Increase the searching step (gcd, binary search, shell sorting, and many linear equation solvers)

- Use the values or space available previously (series summation, reverse and incremental algorithms)