Loop

Session Objective

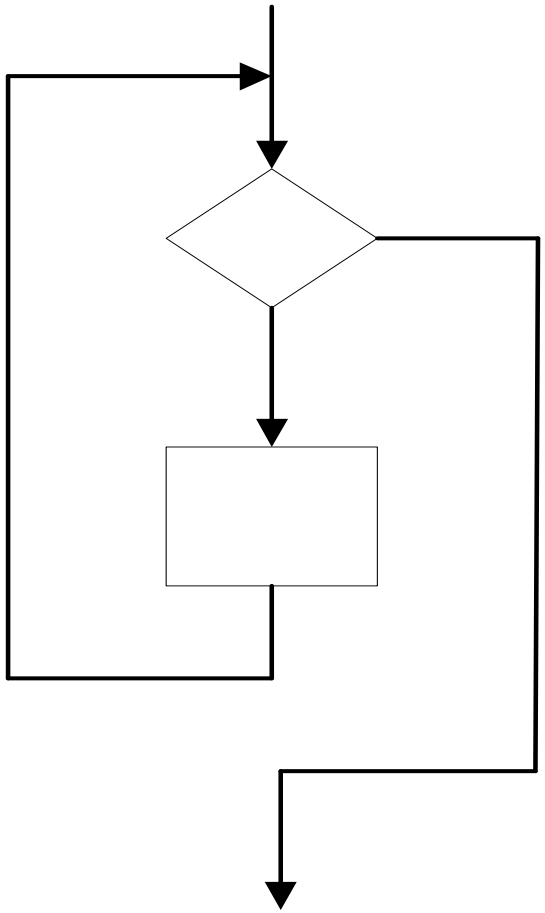
- To learn about different types of Control Statements

Session Topics

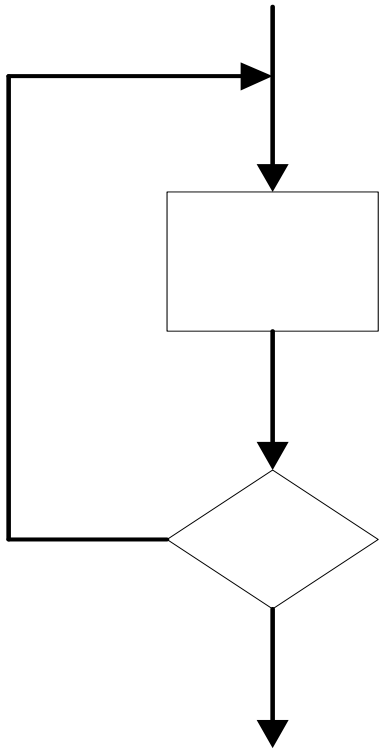
- Repetition or Iteration structure - for statement, continue statement, nested loop, while loop

C Control Structure Looping

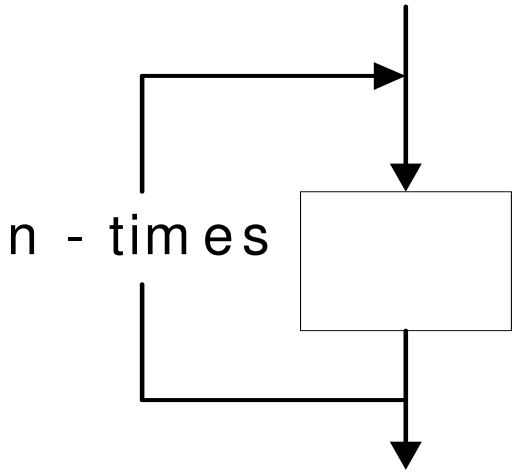
while



do-while



for



Operators (contd…)

- A unary operator has one operand
  - Post and Pre increment and decrement
    - a++,b--,++a,--b
      - » a = a++;
- A binary operator has two operands
    - a = b+c
- A ternary operator has three operands
    - ***boolean-expr*** ? ***expression-1*** : ***expression-2***

The ternary operator

- ***boolean-expr*** `?` ***expression-1*** `:` ***expression-2***
- This is like `if-then-else` for values rather than for statements
- If the ***boolean-expr*** evaluates to `true`, the result is ***expression-1***, else it is ***expression-2***
- Example: `max = a > b ? a : b ;` sets the variable `max` to the larger of `a` and `b`
- ***expression-1*** and ***expression-2*** need not be the same type, but either result must be useable (not a "void" function)
- *The ternary operator is right associative!*
  - To avoid confusion, use parentheses if your expression has more than one ternary operator

Increment and Decrement Operators

- AKA unary operators
- The increment operator ++ adds 1 to its operand, and the decrement operator - - subtracts 1.
- If either is used as a prefix operator, the expression increments or decrements the operand before its value is used.
- If either is used as a postfix operator, the increment and decrement operation will be performed after its value has been used.

```
main( )
{
int a=5;
printf("a=%d”,a++);
}
```

```
main()
{
int a=5;
printf("a=%d",++a);
}
```

```
main()
{
int a=5;
printf("a=%d“,a--);
}
```

```
main()
{
int a=5;
printf("a=%d”,--a);
}
```

```
main()
{
int a=5;
printf("a=%d a=%d",a,++a,a++);
}
```

```
main( )
{
int a=5;
printf("a=%d",a--);
printf("a=%d“,--a);
printf("a=%d",a--);
}
```

```
main()
{
int a=5;
printf("a=%d",a++);
printf("a=%d",++a);
printf("a=%d",a++);
}
```

The while Statement

- **Structure**

while(expression)
statement;

or

while(expression)
{
statement_1;
statement_2;
}

The while Statement Example

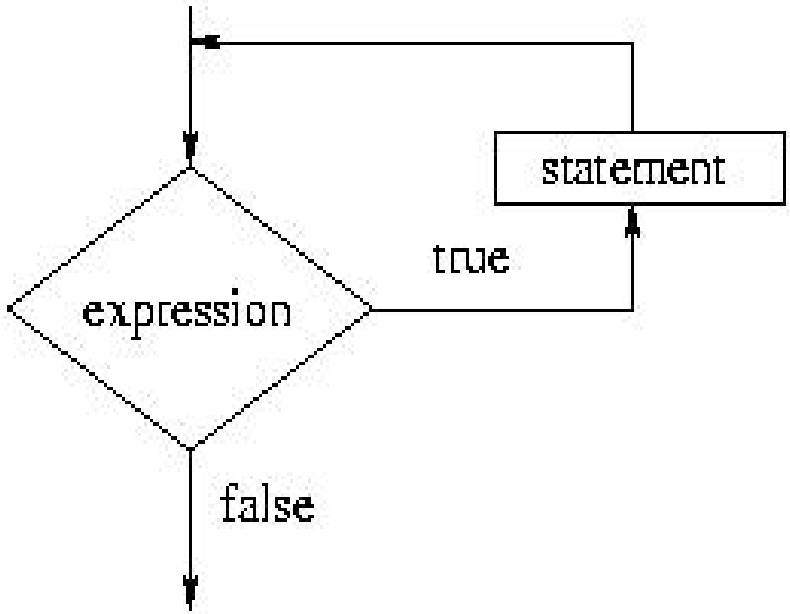
```
while (a < b)
{
printf("%d\n",
a);
a = a + 1;
}
```

* **Operation: *expression* is evaluated and if *TRUE* then *statement* (or *statement_1 and statement_2)* is executed. The evaluation and executions sequence is repeated until the expression evaluates to be *FALSE*. If the expression is initially *FALSE* then *statement* is not executed at all.**

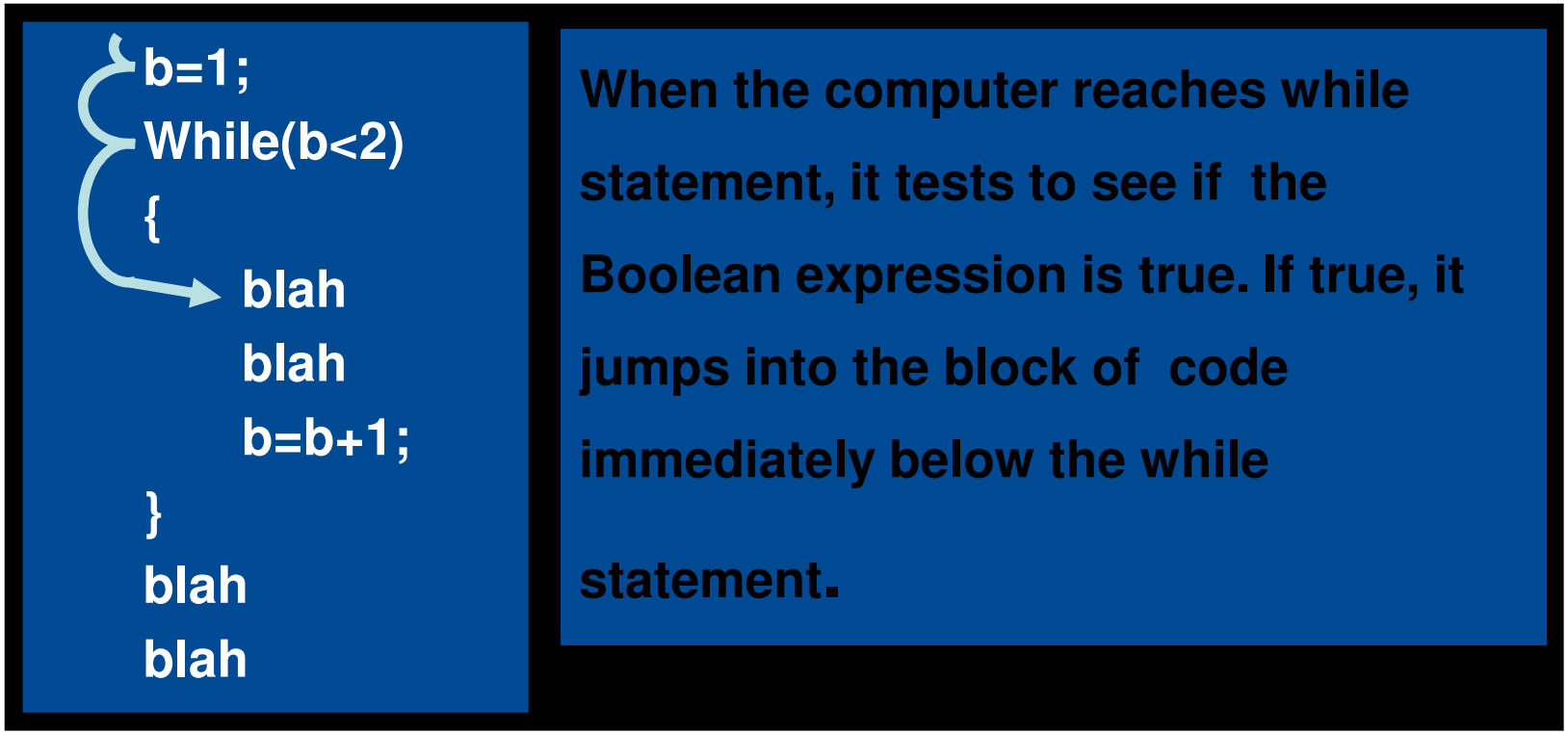
Flowchart of a while Loop

- The syntax of a while loop is as follows:

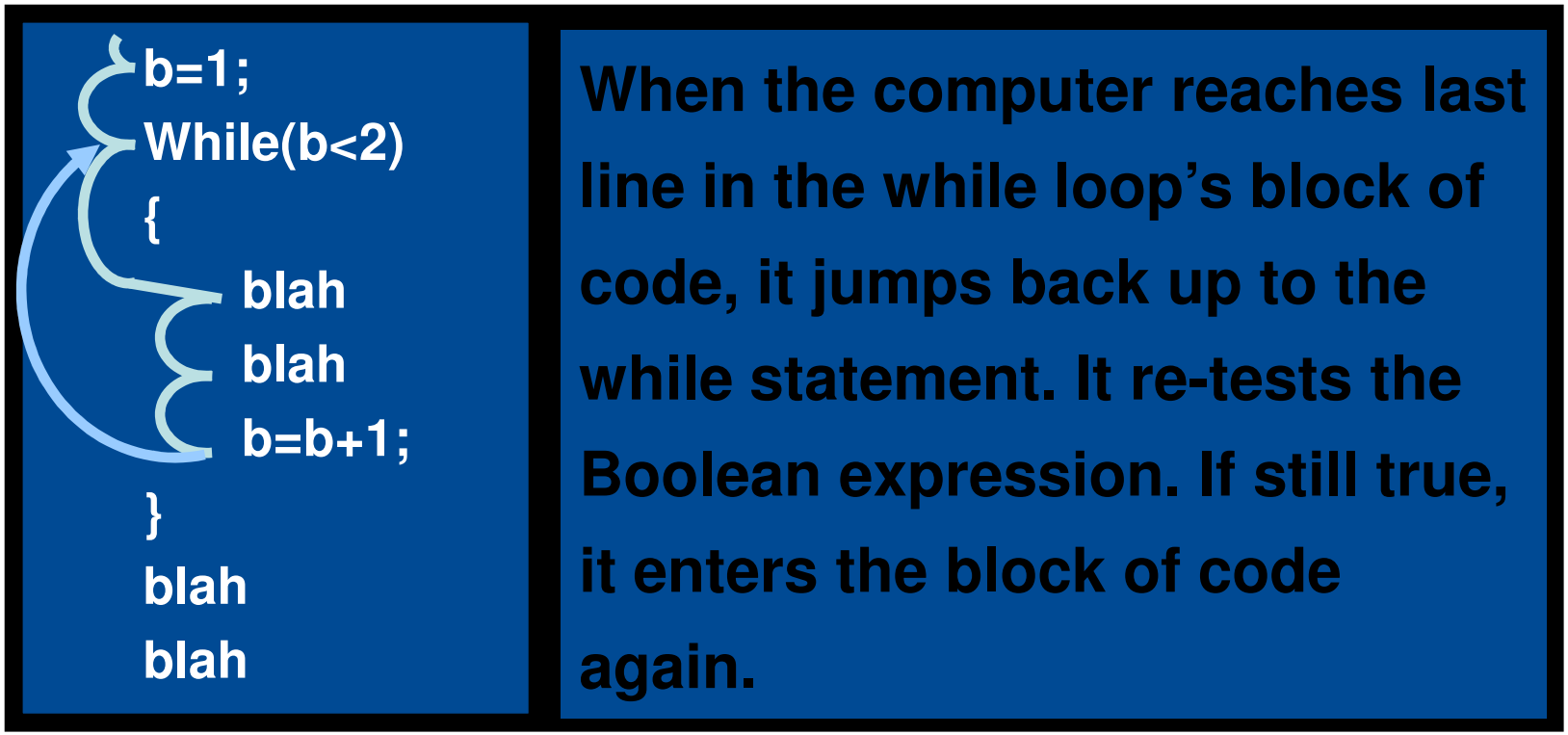
```
while(expression)
   statement
```



Understanding a while loop

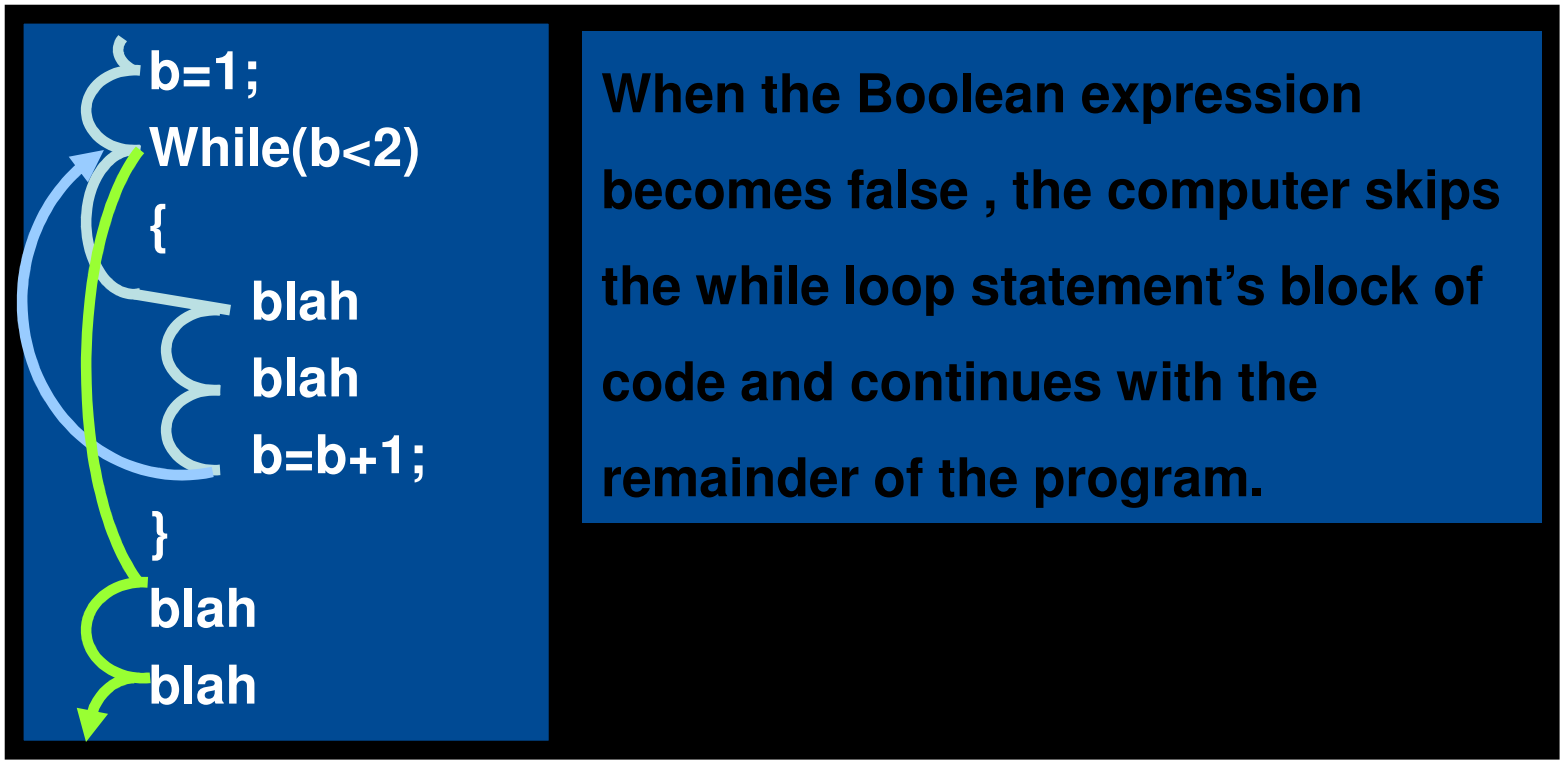


Understanding a while loop



When the computer reaches last line in the while loop's block of code, it jumps back up to the while statement. It re-tests the Boolean expression. If still true, it enters the block of code again.

Understanding a while loop



Looping: A Real Example

- **Let's say that you would like to create a program that prints a Fahrenheit-to-Celsius conversion table. This is easily accomplished with a for loop or a while loop:**

```
main()
{
    int a;
    a = 0;
    while (a <= 100)
    {
        printf("%d degrees F = %d degrees C\n", a, (a - 32)
          * 5 / 9);
        a = a + 10;
    }
}
```

Example:

```
int i = 0;
while(i < 5){
    printf("%d ",i);
    i++;
}
```

Output:

```
0 1 2 3 4
```

```
main( )
{
int i = 1,sum=0;
while(i <=100 ){
     sum=sum+i;
printf("Sum=%d \t &
   i=%d\n",sum,i++);
}}
```

```
main( )
{
int i = 100,sum=0;
while(i >=1 ){
     sum=sum+i;
printf("Sum=%d \t &
   i=%d\n",sum,i--);
}}
```

```
main( )
{
int i = 1,sum=0;
while(i <=100 ){
     sum=sum+i;
printf("Sum=%d \t &
   i=%d\n",sum,i=i+5
   );
}}
```

```
main( )
{
int i = 100,sum=0;
while(i >=1 ){
     sum=sum+i;
printf("Sum=%d \t &
   i=%d\n",sum,i=i-
   5);
}}
```

```
main( )
{
int i = 1,sum=0;
while(i <=100 ){
if(i%2==0)
{
sum=sum+i;
}
printf("Sum=%d \t &
   i=%d\n",sum,i++);
}
}
```

```
main( )
{
int i = 1,sum=0;
while(i <=100 ){
if(i%2==1)
{
sum=sum+i;
}
printf("Sum=%d \t &
   i=%d\n",sum,i++);
}
}
```

Solve same problem using decrement(- -)operator

```
main( )
{
int i = 1,sum=0;
while(i <=100 ){
if(i%2==0 &&
   i%5==0)
{
sum=sum+i;
}
printf(“Sum=%d \t &
   i=%d\n”,sum,i++);
}
}
```

```
main( )
{
int i = 100,sum=0;
while(i >=1){
   if(i%2==0 && i%5==0 &&
   i%10==0)
   {
   sum=sum+i;
   printf("Sum=%d \t & i=%d\n",sum,i);
}//bracket for if
   i=i-1;
}//bracket for while
}//bracket for main
```

Example:

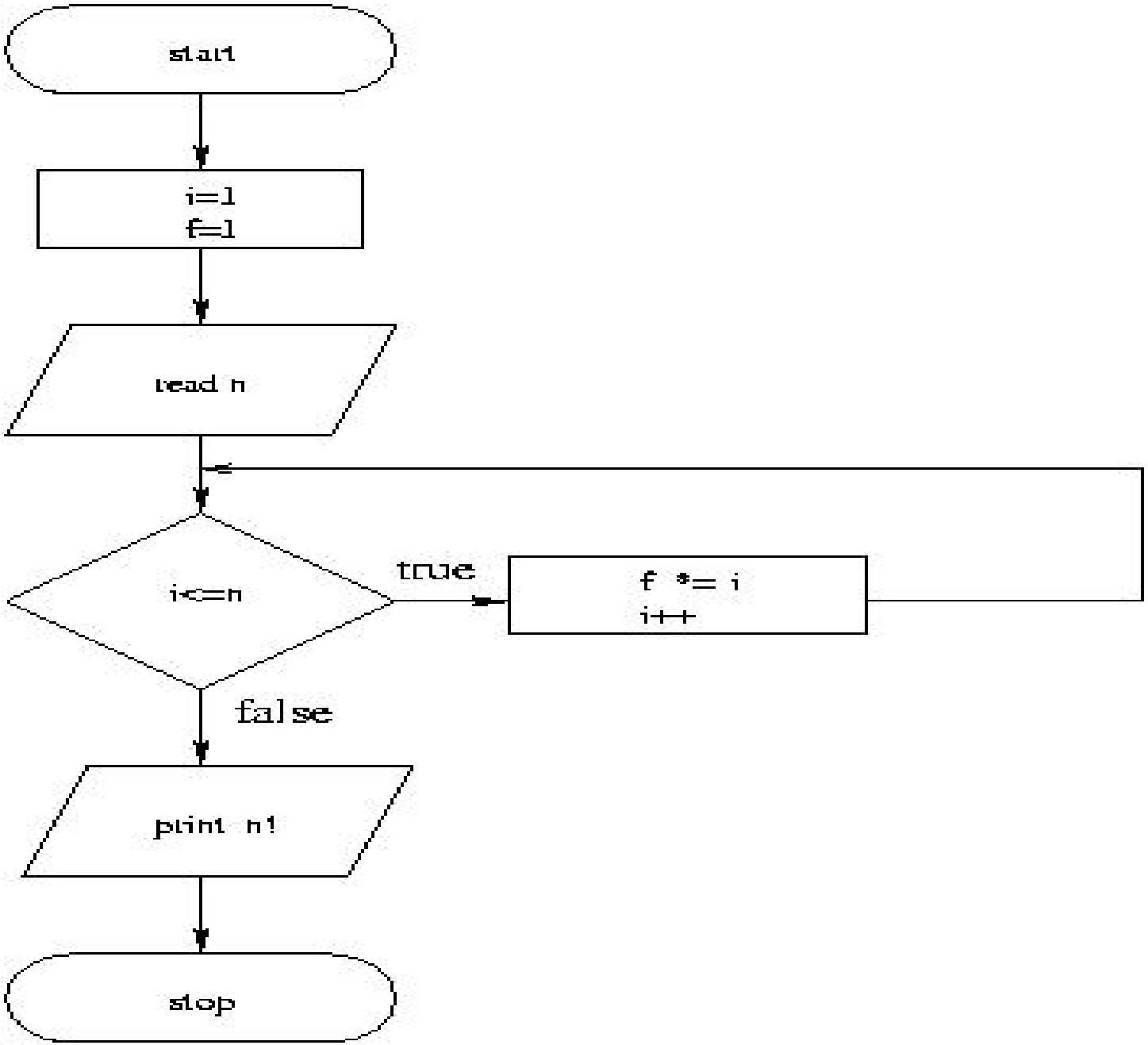
Calculating a factorial 5!. The factorial n! is defined as n(n-1)!*

```
main() {
    /* declaration */
       int i, f, n;
    /* initialization */
    i = 1;
    f = 1;

    /* processing */
    printf("Please input a number\n");
    scanf("%d", &n);
    while (i <= n) {
       f *= i;
       i++;
    }

    /* termination */
    printf("factorial %d! = %d\n", n, f);
}
```

```
5
factorial 5! = 120
```


start
i=1
f=1
read n
i<=n
true
f *= i
i++
false
print n!
stop

Control of Repetition

- Counter-controlled repetition
- Sentinel-controlled repetition

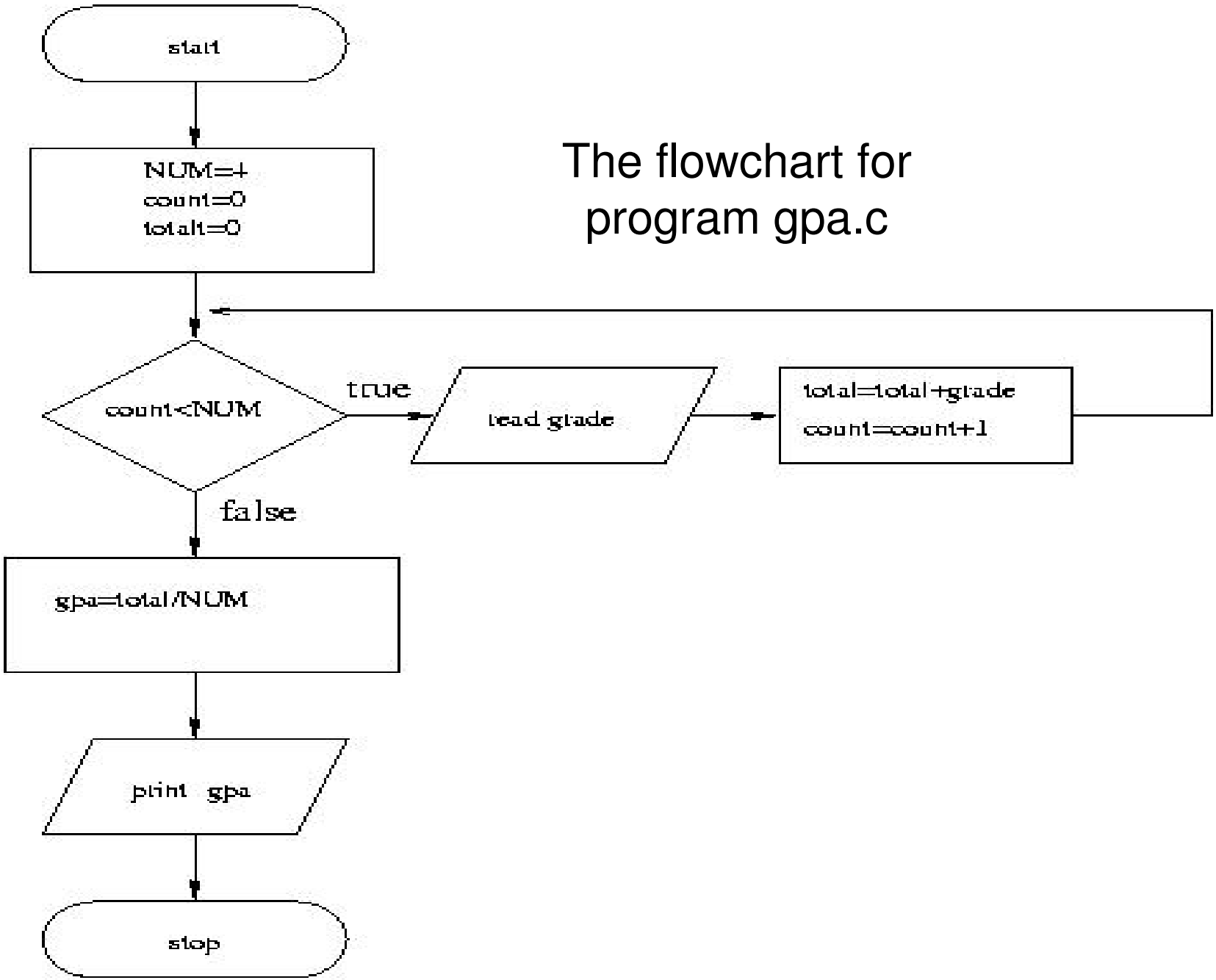
Counter-controlled repetition

- **Loop repeated until counter reaches a certain value.**
- **Definite repetition: number of repetitions is known**

Example:

A student takes four courses in a quarter.
Each course will be assigned a grade with the score from 0 to 4.
Develop a C program to calculate the grade point average (GPA) for the quarter.

The flowchart for program gpa.c



```
#define NUM 4
main() {
    int count;
    double grade, total, gpa;
    count = 0;
    total = 0;
    /* processing */
    while(count < NUM) {
       printf("Enter a grade: ");
       scanf("%lf", &grade);
       total += grade;
       count++;
    }
    /* termination */
    gpa = total/NUM;
    printf("The GPA is: %f\n",
gpa);
}
```

Macro

NUM ≡ 4

Execution and Output:

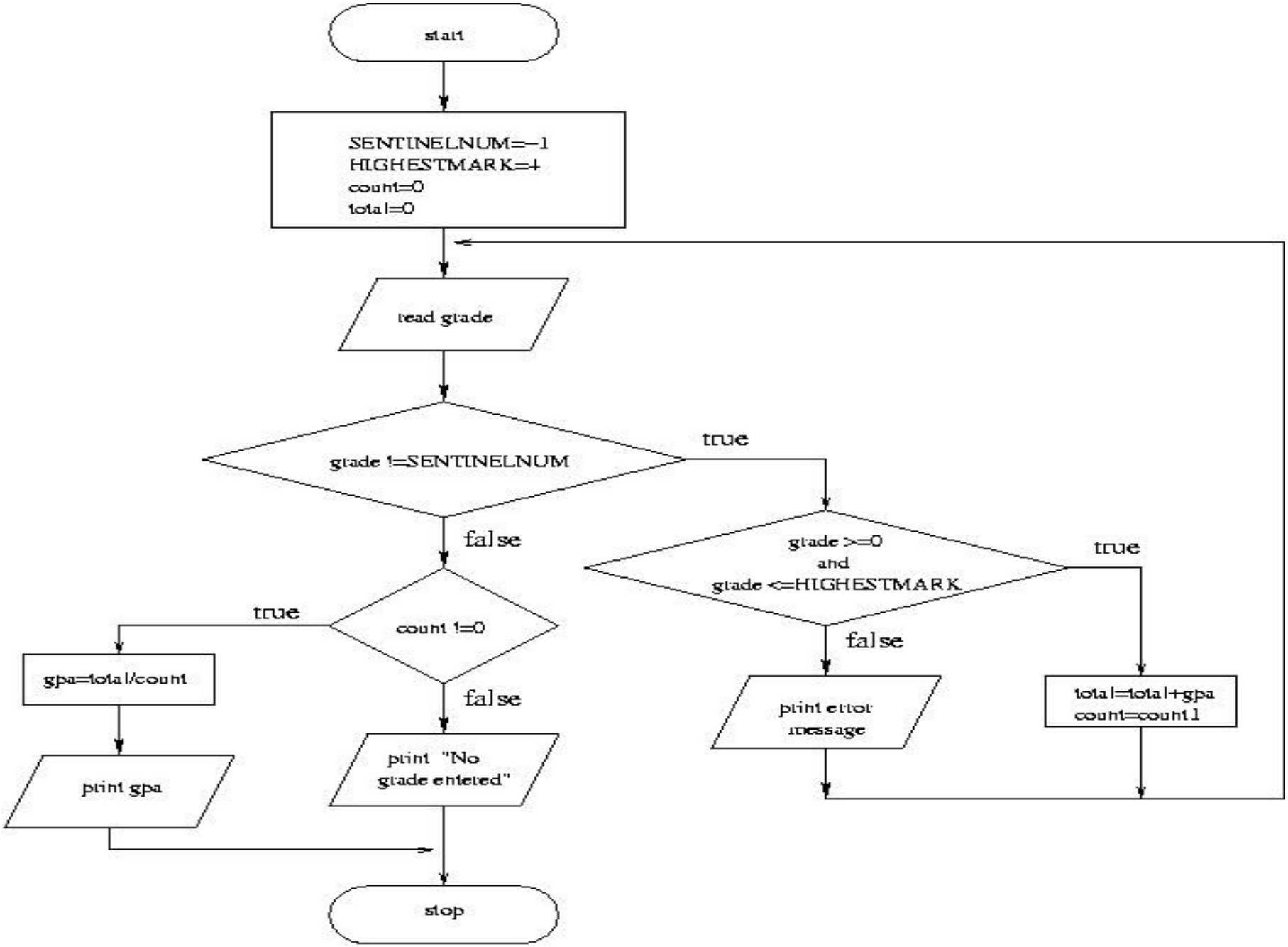
```
Enter a grade: 4
Enter a grade: 3.7
Enter a grade: 3.3
Enter a grade: 4
The GPA is:
  3.750000
```

Sentinel-controlled repetition

- **Loop repeated until the sentinel (signal) value is entered.**
- **Indefinite repetition: number of repetitions is unknown when the loop begins execution.**

Example:

Develop a GPA calculation program that will process grades with scores in the range of [0, 4] for an arbitrary number of courses.


start
SENTINELNUM=-1
HIGHESTMARK=+
count=0
total=0
read grade
grade !=SENTINELNUM
true
false
grade >=0
and
grade <=HIGHESTMARK
true
false
count !=0
true
false
gpa=total/count
print gpa
print "No
grade entered"
print error
message
total=total+gpa
count=count1
stop

```
#define SENTINELNUM -1
#define HIGHESTMARK 4
main() {
    int count;
    double grade, total, gpa;
    /* initialization */
    count = 0;
    total = 0;
    printf("Enter a grade [0, %d] or %d to end: ",
           HIGHESTMARK, SENTINELNUM);
    scanf("%lf", &grade);
    while((int)grade != SENTINELNUM) {
       if(0 <= grade && grade <= HIGHESTMARK) {
         total += grade;
         count++;
       }
       else {
         printf("Invalid grade %c\n", '\a');
       }
       printf("Enter a grade [0, %d] or %d to end: ",
              HIGHESTMARK, SENTINELNUM);
       scanf("%lf", &grade);
    }
```

```
    /* termination */
     if(count != 0) {
        gpa = total/count;
        printf("The GPA is: %f\n",
gpa);
     }
     else
       printf("No grade entered.\n");

     }
```

Execution and Output:

```
Enter a grade [0, 4] or -1 to end: 4
Enter a grade [0, 4] or -1 to end: 3.7
Enter a grade [0, 4] or -1 to end: 3.3
Enter a grade [0, 4] or -1 to end: 4
Enter a grade [0, 4] or -1 to end: 10
Invalid grade
Enter a grade [0, 4] or -1 to end: 3.7
Enter a grade [0, 4] or -1 to end: -1
The GPA is: 3.740000
```

- **do-while Loop**
  - The syntax of a do-while statement is as follows:

```
do
   statement
while(expression);
```

  - The evaluation of the controlling expression takes place after each execution of the loop body.
  - The loop body is executed repeatedly until the return value of the controlling expression is equal to 0.

The do-while Statement

- **Structure**

do
{
statement;
} while(expression);

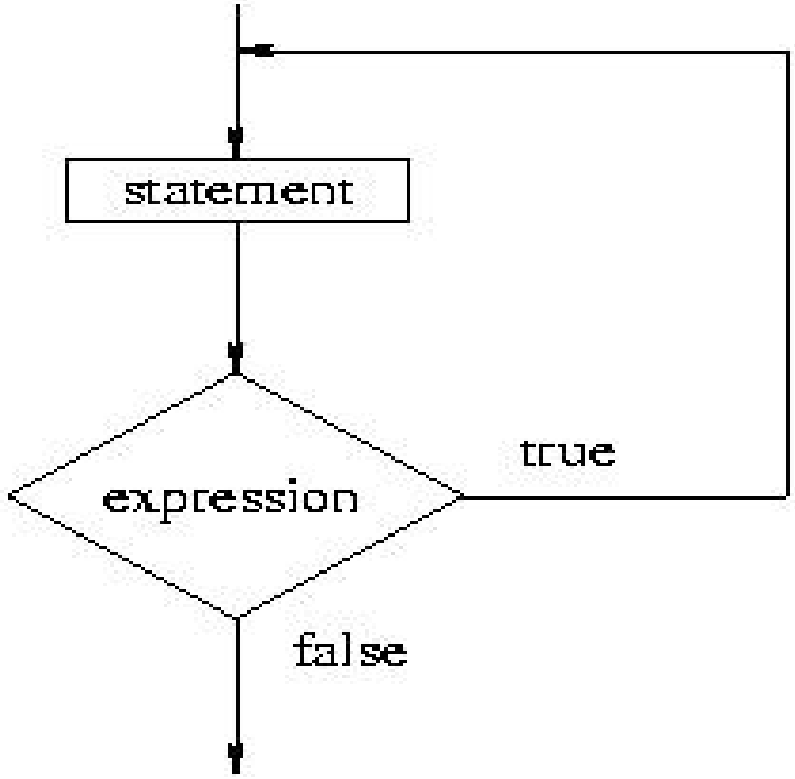
```
do {
 printf("%d\n", a);
a = a + 1;
} while (a < b);
```

- **Operation: Similar to the *while* control except that *statement* is executed before the *expression* is evaluated. This guarantees that *statement* is always executed at least one time even if *expression* is *FALSE*.**

Flowchart of a do-while loop

- The syntax of a do-while statement is as follows:

```
do
    statement
while(expression);
```



Example:

```
int i = 0;
do {
    printf("%d ", i);
    i++;
} while(i < 5);
```

Output:

```
0 1 2 3 4
```

– What is the output of the following example?

Example:

```
int i = 10;
do {
   printf("%d ", i);
   i++;
} while(i < 5);
```

Output:

```
10
```

The for Statement

- **Structure:** ***for(expr1; expr2; expr3)***

```
{
        statement;
}
```

- **Operation: The for loop in C is simply a shorthand way of expressing a while statement:**

```
 expr1;
while(expr2)
{
        statement;
         expr3
}
```

- **The comma operator lets you separate several different statements in the initialization and increment sections of the for loop (but not in the test section).**

C Errors to Avoid

- **Putting = when you mean == in an if or while statement**
- **Forgetting to increment the counter inside the while loop - If you forget to increment the counter, you get an infinite loop (the loop never ends).**
- **Accidentally putting a ; at the end of a for loop or if statement so that the statement has no effect - For example:**

```
for (x=1; x<10; x++);
printf("%d\n",x);
```

only prints out one value because the semicolon after the for statement acts as the one line the for loop executes.

- **For Loop**
  - The syntax of a **`for`** statement is as follows:

```
for(expression1; expression2; expression3)
   statement
```

  - The expression `expression1` is evaluated as a void expression before the first evaluation of the controlling expression.
  - The expression `expression2` is the controlling expression that is evaluated before each execution of the loop body.
  - The expression `expression3` is evaluated as a void expression after each execution of the loop body.
  - Both `expression1` and `expression3` can be omitted. An omitted `expression2` is replaced by a nonzero constant.

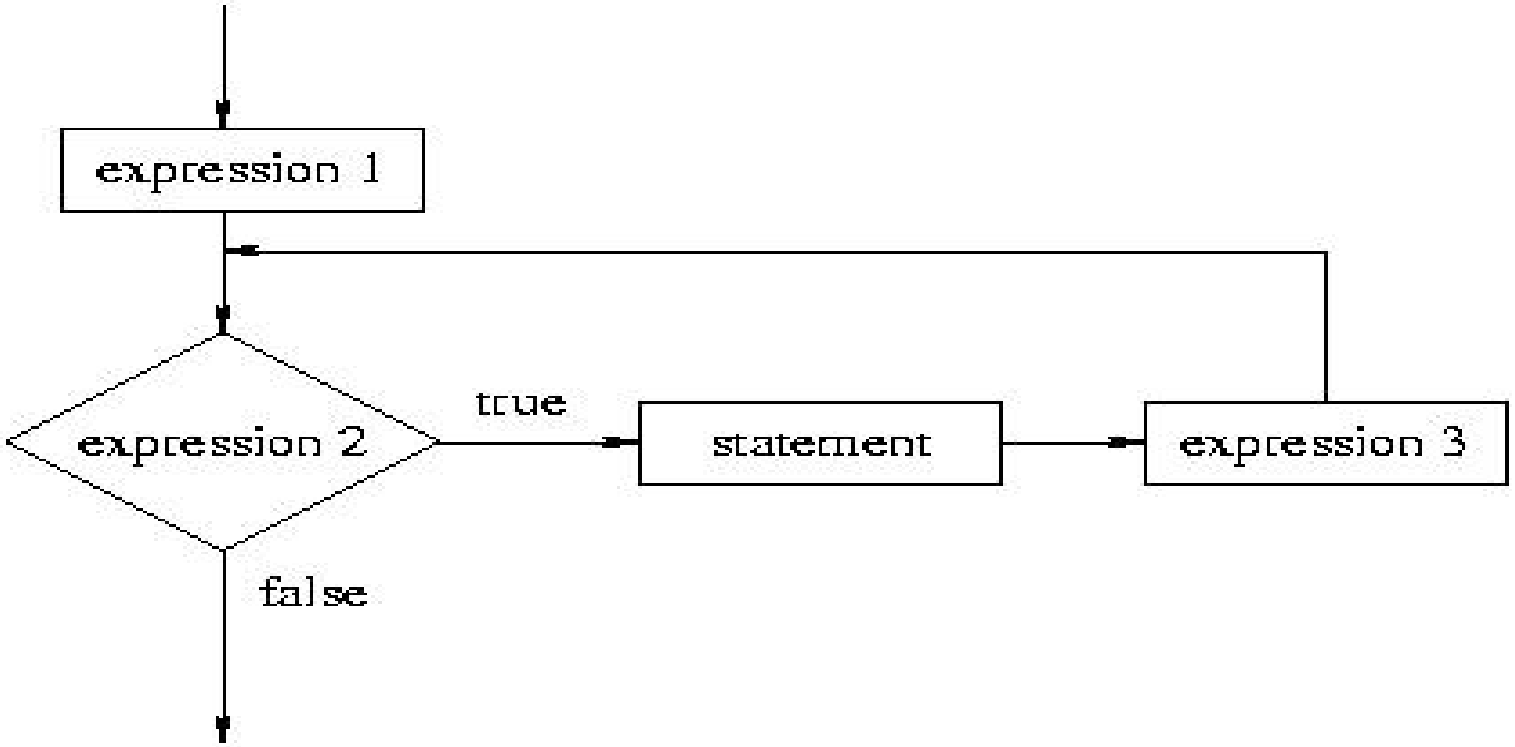
– The for-loop is semantically equivalent to the following while-loop:

```
expression1;
while(expression2)
 {
   statement
   expression3;
}
```

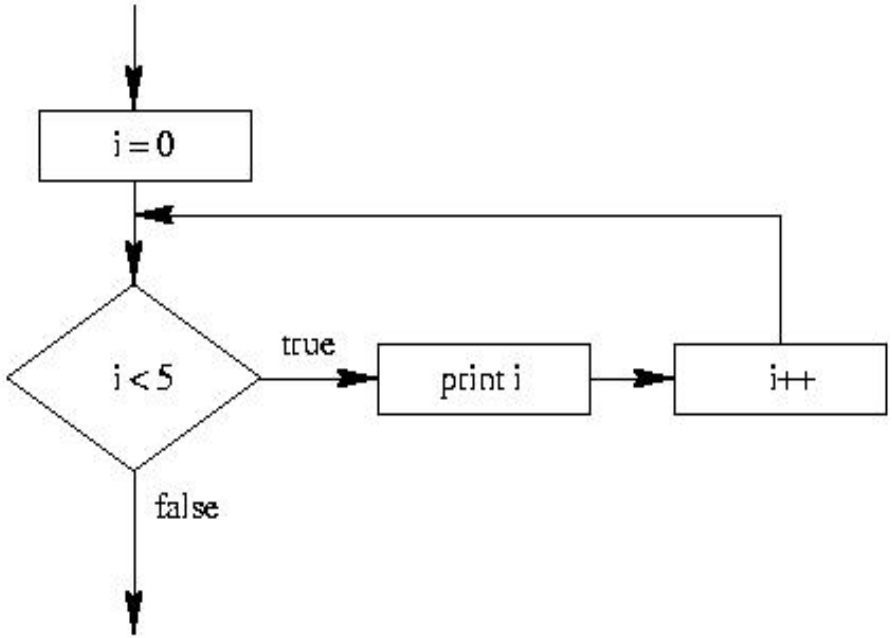
Flowchart of a for Loop

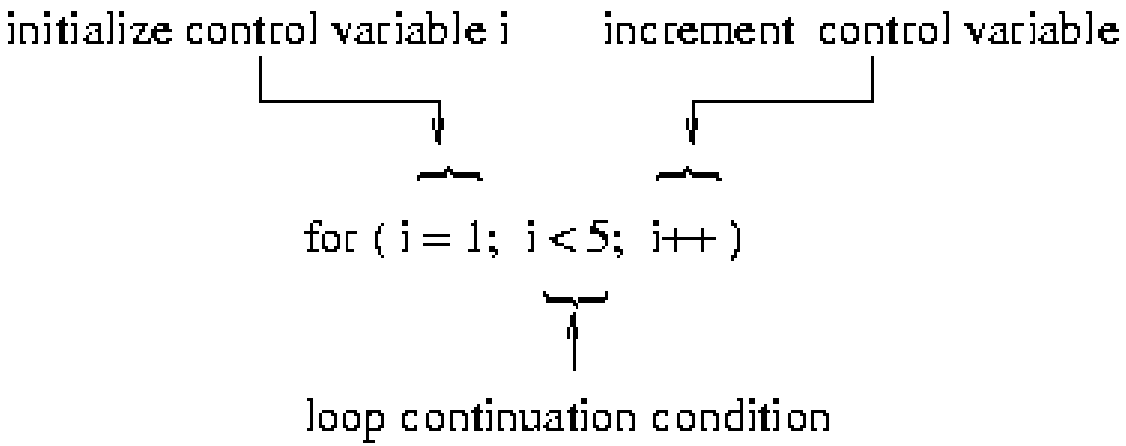
The syntax of a for loop is as follows:

```
for(expression1; expression2; expression3)
  statement
```



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





Output:

```
0 1 2 3 4
```

Example:

Calculating a factorial 5!. The factorial n! is defined as n*(n-1)!

```
main() {
     int i, f, n;
     printf("Please input a number\n");
     scanf("%d", &n);
     for(i=1, f=1; i<=n; i++)
     {
        f = f*i;
     }
     printf("factorial %d! = %d\n", n, f);
}
```

Execution and Output:

```
5
factorial 5! = 120
```

Jump Statements

- **Break Statements**
  - The break statement provides an early exit from the for, while, do-while, and for each loops as well as switch statement.
  - A break causes the innermost enclosing loop or switch to be exited immediately.

Example:

```
int i;
   for(i=0; i<5; i++)
  {
     if(i == 3)
        {
        break;
        }
        printf("%d", i);
   }
```

Output :0 1 2

- **Continue Statements**
  - The `continue` statement causes the next iteration of the enclosing `for`, `while` and `do-while` loop to begin.
  - A continue statement should only appear in a loop body.

```
int i;
          for(i=0; i<5; i++)
         {
            if(i == 3)
            {
              continue;
            }
            printf("%d", i);
          }
```

Output : 0 1 2 4

Nested Loop

Nested loop = loop inside loop

Program flow

The inner loops must be
finished before the outer loop resumes iteration.

Write a program to print a multiplication table.

```
    1   2   3   4   5   6   7   8   9  10
---------------------------------------------
 1|   1
 2|   2   4
 3|   3   6   9
 4|   4   8  12  16
 5|   5  10  15  20  25
 6|   6  12  18  24  30  36
 7|   7  14  21  28  35  42  49
 8|   8  16  24  32  40  48  56  64
 9|   9  18  27  36  45  54  63  72  81
10|  10  20  30  40  50  60  70  80  90 100
---------------------------------------------
```

Example: A program to print a multiplication table.

```
main()
{
int i, j;
printf("1   2   3   4   5   6   7   8   9 10\n");
printf("-------------------------------------\n");
for(i=1; i<= 10; i++)
        { /* outer loop */
            printf("%d  ", i);
                  for(j=1; j<=i; j++)
                  { /* inner loop */
                        printf("%d  ", i*j);
                  }
            printf("\n");
     }
printf("-------------------------------------\n");
}
```

Solve all problem using while,do – while and for

- WAP to find sum of ten numbers.
- WAP to display all numbers between 1 to 1000 that are perfectly divisible by 10.
- WAP to display all numbers between 2000 to 100 that are perfectly divisible by 13 and 15 and 17 and 19.
- WAP to find sum of all numbers between 500 to 1500 that are perfectly divisible by 3 and 5 and 15 and 45.
- WAP to find sum of all numbers between 10000 to 1000 that are perfectly divisible by 3 and 7 and 9 and 42.
- WAP to find factorial of given number
  - N! = N * (N-1)!
- WAP to check whether a given number is prime or not.
  - A number is prime if it is divisible by 1 and the number itself only
- WAP to generate all prime numbers between 100 to 1.

- WAP to generate multiplication table of any given number
- WAP to generate multiplication table of number 1 to 5.
- WAP to check whether a given program in Armstrong or not.
  - 153 is a Armstrong number because $1^3+5^3+3^3=153$
- WAP to check whether a number is strong or not.
  - 145 is a strong number because 1! + 4! +5 ! = 145
- WAP to check whether a number is perfect or not.
  - 28 is a perfect number because 1+2+4+7+14 = 28
- WAP to find GCD (HCF) of given two numbers.
- WAP to find LCM of given numbers.