

# **CS100**

# **Introduction to Programming**

## **Lecture 4. Control Flow**

# Relational Operators

Used for **comparison** between **two values**.

Return **Boolean** result: **true** or **false**.

## Relational Operators:

operator	example	meaning
<b>==</b>	ch == 'a'	<b>equal to</b>
<b>!=</b>	f != 0.0	<b>not equal to</b>
<b>&lt;</b>	num < 10	<b>less than</b>
<b>&lt;=</b>	num <=10	<b>less than or equal to</b>
<b>&gt;</b>	f > -5.0	<b>greater than</b>
<b>&gt;=</b>	f >= 0.0	<b>greater than or equal to</b>

# Logical Operators

- Work on one or more relational expressions to yield a logical value: **true** or **false**.
- Allow testing and combining the results of comparison expressions.

## Logical Operators:

operator	example	meaning
!	!(num < 0)	not
&&	(num1 > num2) && (num2 > num3)	and
	(ch == '\t')    (ch == ' ')	or

	A is true	A is false
!A	false	true

A    B	A is true	A is false
B is true	true	true
B is false	true	false

A && B	A is true	A is false
B is true	true	false
B is false	false	false

# Precedence of operators

- List of operators of **decreasing precedence**:

!	not
* /	multiply and divide
+ -	add and subtract
< <= > >=	less, less or equal, greater, greater or equal
== !=	equal, not equal
&&	logical and
	logical or

- Example:** The expression **!(5 >= 3) || (7 > 3)** is **true**, where the **logical or operator ||** is executed in the end.

# Boolean Result

- The **result** of evaluating an expression involving relational and/or logical operators is
  - either **true** or **false**
  - either **1** or **0**
  - When the result is **true**, it is **1**. Otherwise, it is **0**. That is, the C language uses 0 to represent a false condition.
- In general, **any integer expression whose value is non-zero is considered true**; otherwise it is **false**.
- Examples:

3	is true
0	is false
1    0	is true
!(5 >= 3)    0	is false

# The if Statement

**if (expression)**

**statement;**

/\* simple or compound statement  
enclosed with brackets \*/

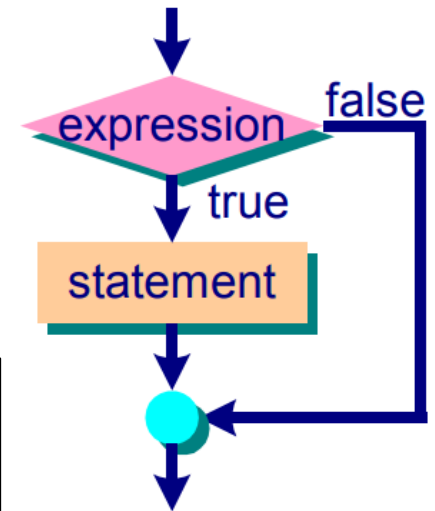
```
#include <stdio.h>
int main(void)
{
    int num; /* value supplied by user. */
    printf("Give an integer from 1 to 10: ");
    scanf("%d", &num);
    if (num > 5)
        printf("Your number is larger than 5.\n");
    printf("%d is the number you entered.\n", num);
    return 0;
}
```

## Output 1:

Give an integer from 1 to 10: **3**  
3 is the number you entered.

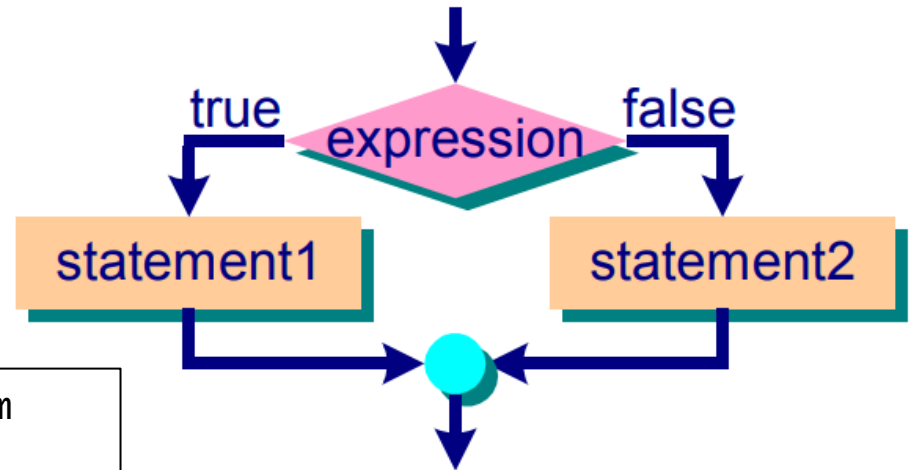
## Output 2:

Give an integer from 1 to 10: **7**  
Your number is larger than 5.  
7 is the number you entered.



# The if-else Statement

```
if (expression)
    statement1;
else
    statement2;
```



```
/* This program computes the maximum
value of num1 and num2 */
#include <stdio.h>
int main(void)
{
    int num1, num2, max;
    printf("Please enter two integers:");
    scanf("%d %d", &num1, &num2);
    if (num1 > num2)
        max = num1;
    else
        max = num2;
    printf("The maximum of the two \
        is %d\n", max);
    return 0;
}
```

## Output:

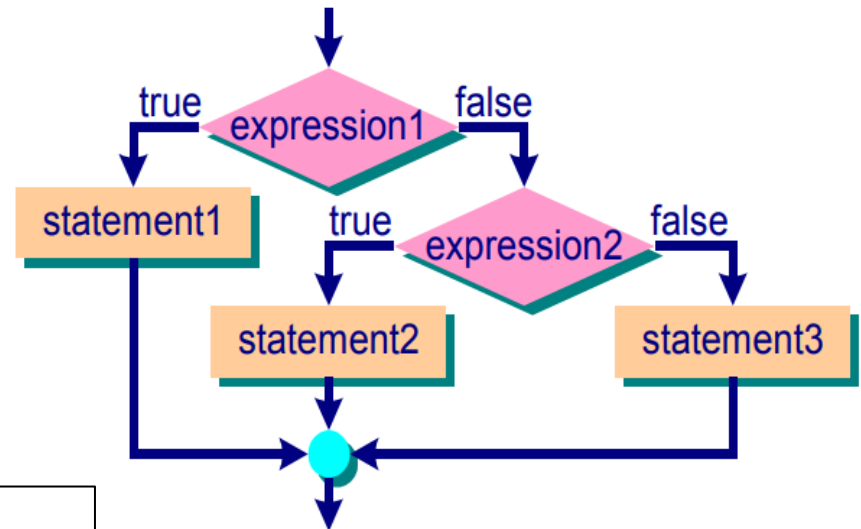
Please enter two integers: 9 4  
The maximum of the two is 9

Please enter two integers: -2 0  
The maximum of the two is 0

# The if...else if...else Statement

```
if (expression1)
    statement1;
else if (expression2)
    statement2;
else
    statement3;
```

```
#include <stdio.h>
int main(void)
{
    float temp; // temperature reading.
    printf("Temperature reading:");
    scanf("%f", &temp);
    if (temp >= 100.00 && temp <= 120.0)
        printf("Temperature OK.\n");
    else if (temp < 100.0)
        printf("Temperature too low.\n");
    else
        printf("Temperature too high.\n");
    return 0;
}
```



## Output:

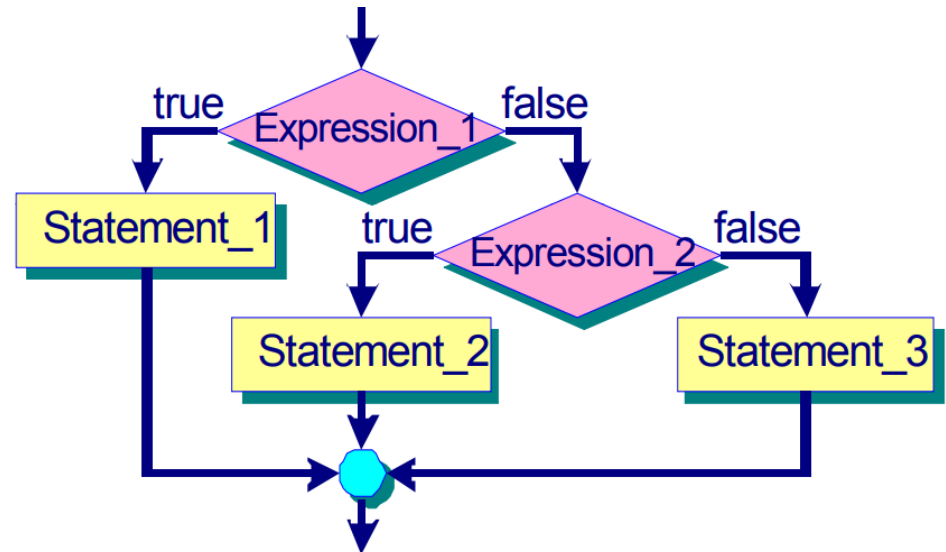
Temperature reading: 105.0  
Temperature OK.

Temperature reading: 130.0  
Temperature too high.

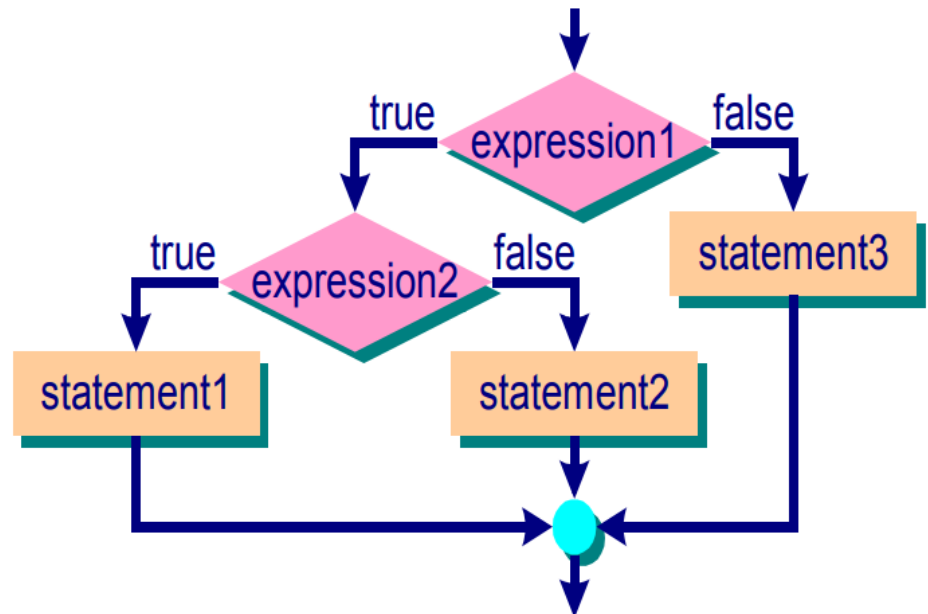


# Nested-if

```
if (expression 1)  
    statement1;  
else  
    if (expression2)  
        statement2;  
    else  
        statement3;
```



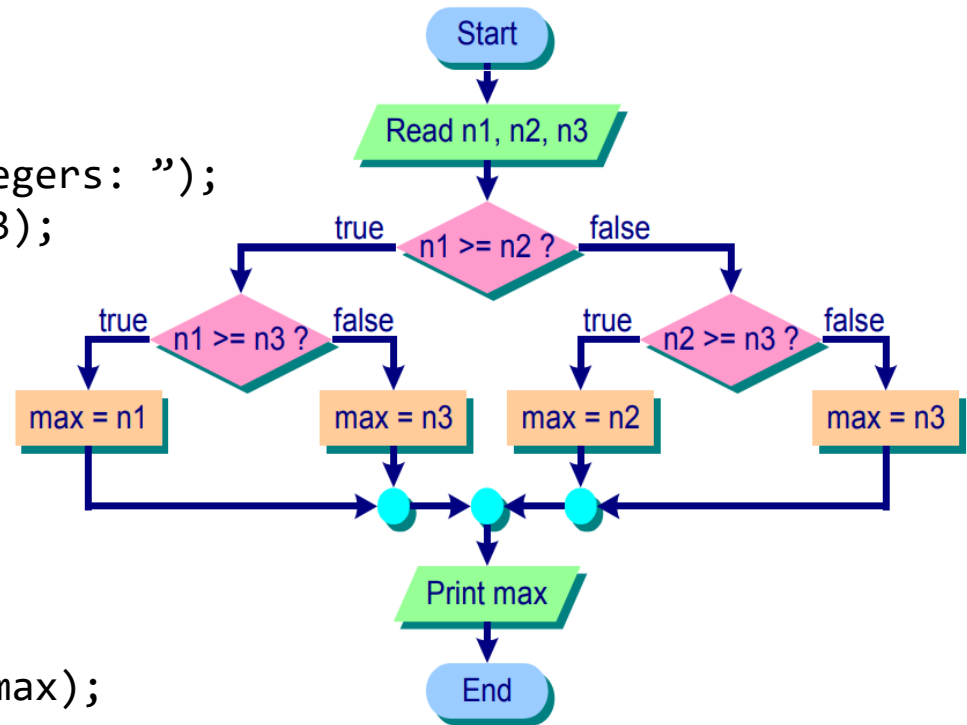
```
if (expression 1)  
    if (expression2)  
        statement1;  
    else  
        statement2;  
else  
    statement3;
```



# Nested-if Example

```
/* This program computes the maximum value of
three numbers */
#include <stdio.h>
int main(void)
{
    int n1, n2, n3, max;
    printf("Please enter three integers: ");
    scanf("%d %d %d", &n1, &n2, &n3);
    if (n1 >= n2)
        if (n1 >= n3)
            max = n1;
        else max = n3;
    else if (n2 >= n3)
        max = n2;
    else max = n3;

    printf("The maximum is %d\n", max);
    return 0;
}
```



## Output:

Please enter three integers: 1 2 3  
The maximum of the three is 3

# The Conditional Operator

- The conditional operator is used in the following way:

**Expression\_1 ? Expression\_2 : Expression\_3**

The value of this expression depends on whether **Expression\_1** is true or false.

if **Expression\_1** is **true**

=> value of the expression is that of **Expression\_2**

**else**

=> value of the expression is that of **Expression\_3**

- Example:

```
max = (x > y) ? x : y;
```

<==>

```
if (x > y)
    max = x;
else
    max = y;
```

# Conditional Operator: Example

```
/* An example to show a conditional expression */
#include <stdio.h>
int main(void)
{
    int selection; /* User input selection */
    printf("Enter a 1 or a 0 => ");
    scanf("%d", &selection);

    selection ? printf("A one.\n") : printf("A zero.\n");
    return 0;
}
```

## Output:

Enter a 1 or a 0 => 1

A one.

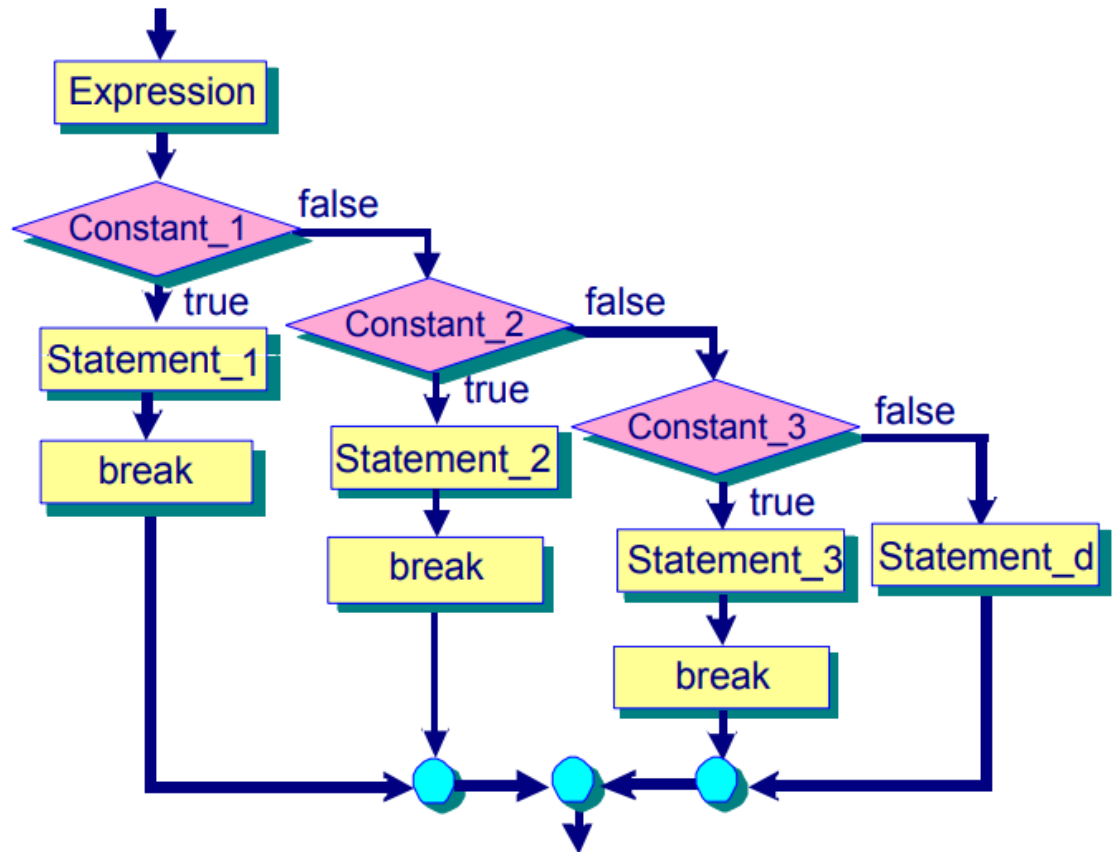
Enter a1 or a0 => 0

A zero.

# The switch Statement

The **switch** is for multi-way selection. The syntax is:

```
switch (Expression) {  
    case Constant_1:  
        Statement_1;  
        break;  
    case Constant_2:  
        Statement_2;  
        break;  
    case Constant_3;  
        Statement_3;  
        break;  
    default:  
        Statement_d;  
}
```



# The switch Statement

- *switch*, *case*, *break* and *default* are reserved words.
- The result of *Expression* in ( ) must be an **integral type**.
- *Constant\_1*, *Constant\_2*, ... are called **labels**. Each must be an **integer constant**, a **character constant** or an **integer constant expression**, e.g. 3, 'A', 4+'b', 5+7, etc.
- Each of the labels *Constant\_1*, *Constant\_2*, ... must deliver a **unique integer value**. Duplicates are not allowed.
- We may also have **multiple labels** for a statement, for example, to allow both the **lower** and **upper** case selection.
- If we **do not use break** after some statements in the switch statement, execution will **continue** with the statements for the subsequent labels until a break statement or the end of the switch statement. This is called the **fall through** situation.

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

```
main(void) {
```

```
    char choice;
```

```
    int num1, num2, result;
```

```
    printf("Enter your choice (A, S or M)=> ");
```

```
    scanf("%c", &choice);
```

```
    printf("Enter two numbers:");
```

```
    scanf("%d %d", &num1, &num2);
```

```
    switch (choice) {
```

```
        case 'a':
```

```
        case 'A': result = num1 + num2;
```

```
            printf("num1 + num2 = %d\n", result);
```

```
            break;
```

```
        case 's':
```

```
        case 'S': result = num1 - num2;
```

```
            printf("num1 - num2 = %d\n", result);
```

```
            break;
```

```
        case 'm':
```

```
        case 'M': result = num1 * num2;
```

```
            printf("num1 * num2 = %d\n", result);
```

```
            break;
```

```
        default: printf("Not a proper choice.\n");
```

```
    }
```

```
    return 0;
```

```
}
```

# switch: Example

## Output:

Enter your choice (A, S or M) => S

Enter two numbers: 9 5

9 - 5 = 4

# A switch Example: Converting Score to Grade

Weighted Average Score S	Grade
$90 \leq S$	A
$80 \leq S < 90$	B
$70 \leq S < 80$	C
$60 \leq S < 70$	D
$50 \leq S < 60$	E
$S < 50$	F

```
switch ((int)averageScore/10) {  
    case 10: case 9:  
        grade = 'A'; break;  
    case 8:  
        grade = 'B'; break;  
    case 7:  
        grade = 'C'; break;  
    case 6:  
        grade = 'D'; break;  
    case 5:  
        grade = 'E'; break;  
    default: grade = 'F';  
}
```



# Looping

# Repetition: Loops

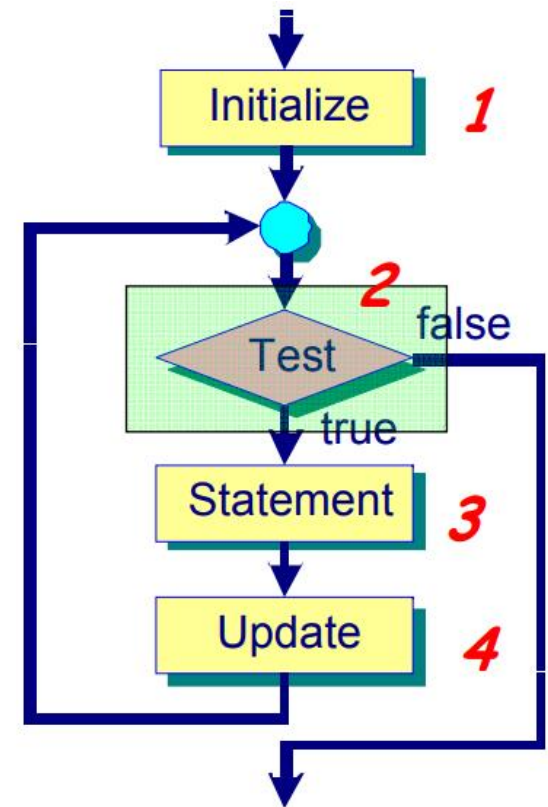
Some sections of statements are executed **repeatedly**. These repetitions are referred to as **loops**. There are two types of loops:

- **Counter-controlled loops**: The loop body is repeated for a number of times, and the number of repetitions is known before the loop starts execution.
- **Sentinel-controlled loops**: The number of repetitions is not known before the loop starts execution. Usually, a **sentinel value** (such as  $-1$ , different from regular data) is used to determine whether to execute the loop body.

# Looping

To construct loops, we need:

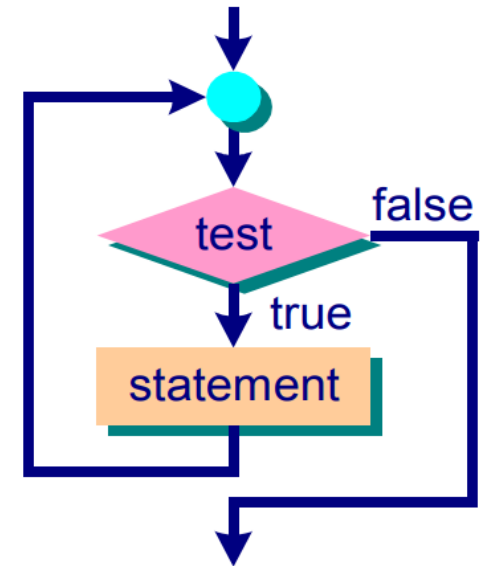
1. **Initialization** – initialize the loop control variable.
2. **Test condition** – evaluate the test condition (involve loop control variable).
3. **Loop body** – if test is true, the loop body is executed.
4. **Update** – typically, loop control variable is **modified** through the execution of the loop body. It can then go through the **test** condition again.



# The while Loop

```
while (test)  
    statement
```

```
/* sum up a list of integers. The list of  
integers is terminated by -1. */  
#include <stdio.h>  
int main(void)  
{  
    int sum, item;  
    printf("Enter the list of integers:\n");  
    scanf("%d", &item);  
    while (item != -1) {  
        /* Sentinel-controlled */  
        sum += item;  
        scanf("%d", &item);  
    }  
    printf("The sum is %d\n", sum);  
    return 0;  
}
```



## Output:

Enter the list of integers:

1 8 11 24 36 48 67 -1

The sum is 195

Enter the list of integers:

-1

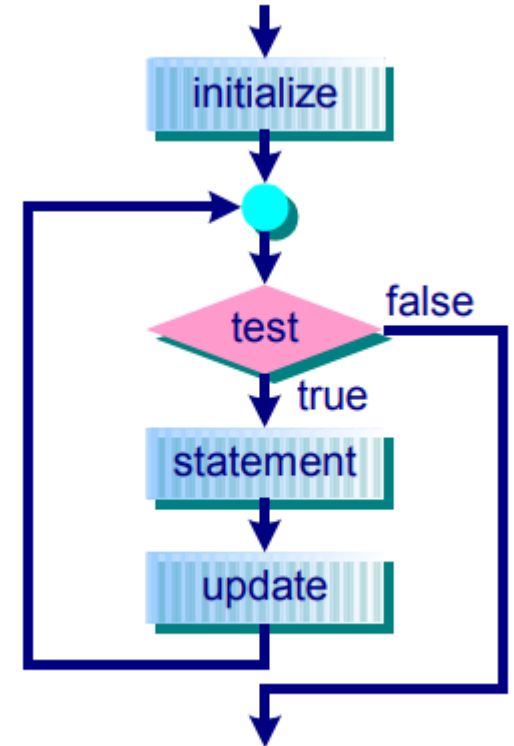
The sum is 0

# The for Loop

```
for (initialize; test; update)  
    statement;
```

- Normally, *test* is a relational expression to control iterations.
- *update* is frequently used to update some loop control variable before repeating the loop.
- Any or all of the 3 expressions may be omitted. In case test is missing, it becomes an *infinite loop*, i.e. all statements inside the loop will be executed again and again. For example:

```
for (;;) {    /* an infinite loop */  
    statement1;  
    ...  
}
```



# for Loop: Example 0



# for Loop: Example 1

```
/* Display the distance a body falls in
feet/sec for the first n seconds; n is input
by user. */
#include <stdio.h>
#define ACCELERATION 32.0
main()
{
    int timeLimit, t;
    /* Distance by a falling body */
    int distance;

    printf("Enter the time limit(seconds): ");
    scanf("%d", &timeLimit);
    for (t = 1; t <= timeLimit; t++) {
        distance = 0.5 * ACCELERATION * t * t;
        printf("Dist after %d seconds is %d\
            feet.\n", t, distance);
    }
    return 0;
}
```

## Output:

Enter the time limit(seconds): 5

Dist after 1 seconds is 16 feet.

Dist after 2 seconds is 64 feet.

Dist after 3 seconds is 144 feet.

Dist after 4 seconds is 256 feet.

Dist after 5 seconds is 400 feet.

Enter the time limit(seconds): 0

# for Loop: Example 2

```
/* Display the distance a body falls every
5 seconds for the first n seconds; n is input
by user. */
#include <stdio.h>
#define ACCELERATION 32.0

main()
{
    int timeLimit, t;
    /* Distance by a falling body */
    int distance;

    printf("Enter the time limit(seconds): ");
    scanf("%d", &timeLimit);
    for (t = 5; t <= timeLimit; t += 5) {
        distance = 0.5 * ACCELERATION * t * t;
        printf("Dist after %d seconds is %d\
            feet.\n", t, distance);
    }
    return 0;
}
```

## Output:

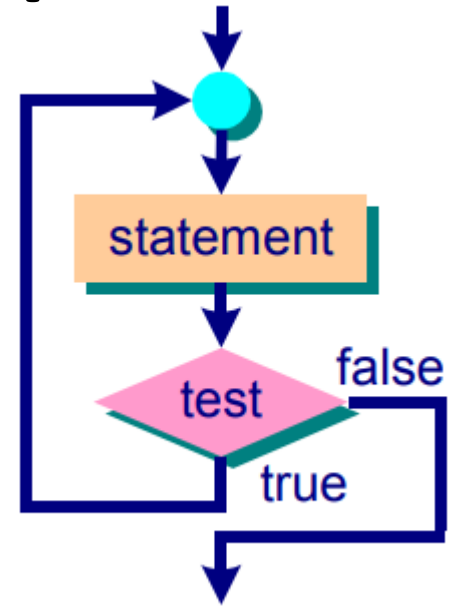
Enter the time limit(seconds): 15  
Dist after 5 seconds is 400 feet.  
Dist after 10 seconds is 1600 feet.  
Dist after 15 seconds is 3600 feet.



# The do-while Loop

```
do {  
    statement;  
} while (test);
```

```
/* Menu-Based User Selection */  
#include <stdio.h>  
int main()  
{  
    int input; /* User input number. */  
    do {  
        printf("Input a number >= 1 and <=5: ");  
        scanf("%d", &input);  
        if (input > 5 || input < 1)  
            print("%d is out of range.\n", input);  
    } while (input > 5 || input < 1);  
    printf("Input = %d\n", input);  
    return 0;  
}
```



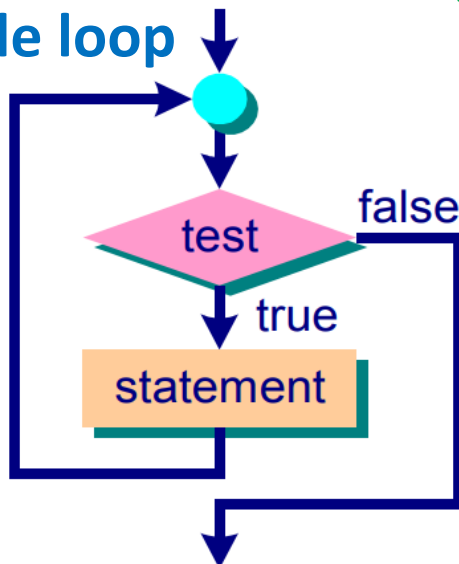
Output:

Input a number >= 1 and <=5: 6  
6 is out of range.  
Input a number >= 1 and <= 5: 5  
Input = 5

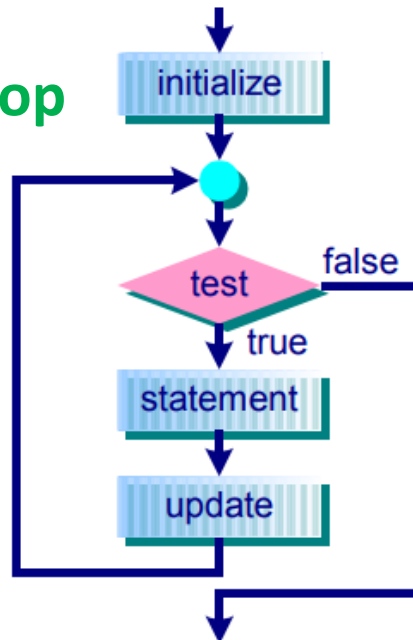
# The do-while Loop

- Different from the *while* and *for* statements:
  - The condition *test* is performed **after** executing the statement every time.
  - This means the loop will be executed **at least once**.
  - Note: the *while* or *for* loop might **not** be executed **even once**.

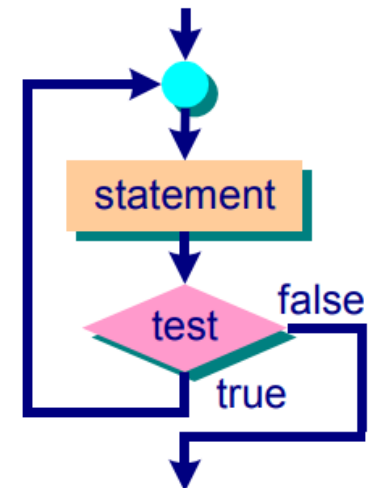
while loop



for loop



do-while loop



# The break Statement

- To alter flow of control inside loop (and inside the switch statement). Execution of **break** causes immediate termination of the inner most enclosing loop or switch statement.

```
/* use break to exit a loop */
#include <stdio.h>
int main(void)
{
    float length, width;
    printf("Enter rectangle length:\n");
    while (scanf("%f", &length) == 1) {
        printf("Enter its width:\n");
        if (scanf("%f", &width) != 1)
            break;
        printf("The area = %6.3f\n\n",
            length * width);
        printf("Enter rectangle length:\n");
    }
    return 0;
}
```

## Output:

Enter rectangle length:

2

Enter its width:

10

The area = 20.000

Enter rectangle length:

4

Enter its width:

a

# The continue Statement

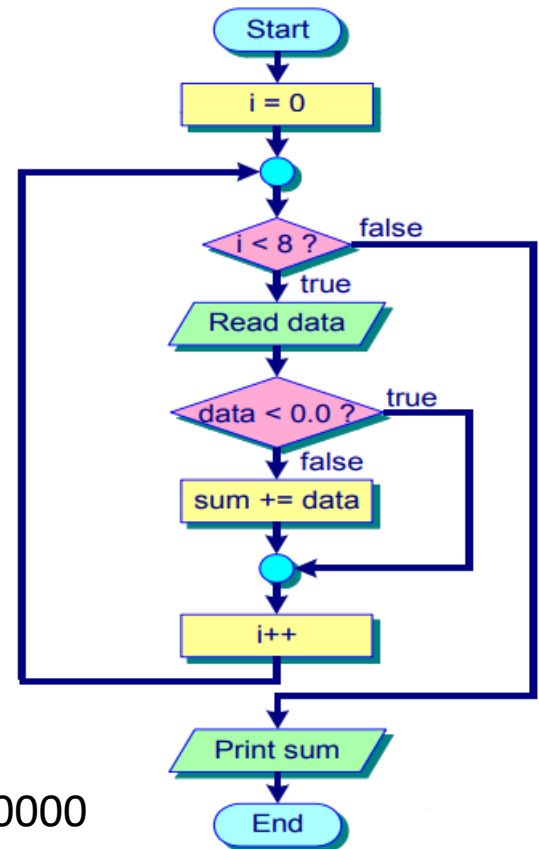
- The control immediately passed to the (update and) test condition of the nearest enclosing loop. All subsequent statements after the continue statement are not executed for this particular iteration.

```
/* summing up positive numbers
from a list of 8 numbers */
#include <stdio.h>
int main(void)
{
    int i;
    float data, sum = 0.0;
    /* read 8 numbers */
    for (i=0; i < 8; i++) {
        scanf("%f", &data);
        if (data < 0.0)
            continue;
        sum += data;
    }
    printf("The sum is %f\n", sum);
    return 0;
}
```

**Output:**

**3 7 -1 4 -5 8 3 1**

The sum is 26.000000



# Nested Loops

- A loop may appear inside another loop. This is called a **nested loop**. We can nest as **many levels** of loops as the hardware allows. And we can nest **different types** of loops.

```
/* count the number of different strings of a, b, c
*/
#include <stdio.h>
int main(void)
{
    char i, j;    /* for loop counters */
    int num = 0; /* overall loop counter */
    for (i = 'a'; i <= 'c'; i++) {
        for (j = 'a'; j <= 'c'; j++) {
            num++;
            printf("%c%c ", i, j);
        }
        printf("\n");
    }
    printf("%d different strings of letters.\n", num);
    return 0;
}
```

## Output:

```
aa ab ac
ba bb bc
ca cb cc
9 different strings of letters.
```

# Nested Loops: Example

```
#include <stdio.h>
int main(void)
{
    int a, b, height, lines;

    printf("Enter the height of pattern: ");
    scanf("%d", &height);
    for (lines=1; lines <= height; lines++) {
        for (a=1; a <= (height - lines); a++)
            putchar(' '); // print blank space
        for (b=1; b <= (2*lines - 1); b++)
            putchar('*'); // print asterisk
        putchar('\n');
    }
    return 0;
}
```

**Output:**

Enter the height of pattern: 5

*a, b* → \*  
*lines* ↓ \*\*\*  
\*\*\*\*\*  
\*\*\*\*\*  
\*\*\*\*\*

# Recap

- This lecture covers the following concepts:
  - Relational and logical operators
  - if, if ... else ..., if ... else if ... else statements
  - Nested if statements
  - switch statements
  - while, for and do-while loops
  - break and continue statements
  - Nested loops
- Next:
  - Functions