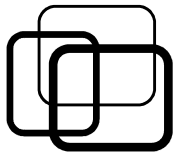
Control Flow Statements

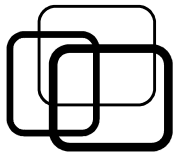
Inst. Nguyễn Minh Huy

Contents



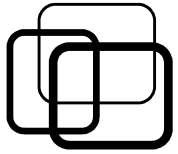
- Branch statements.
- Loop statements.

Contents



- **Branch statements.**
- Loop statements.

Branch statements

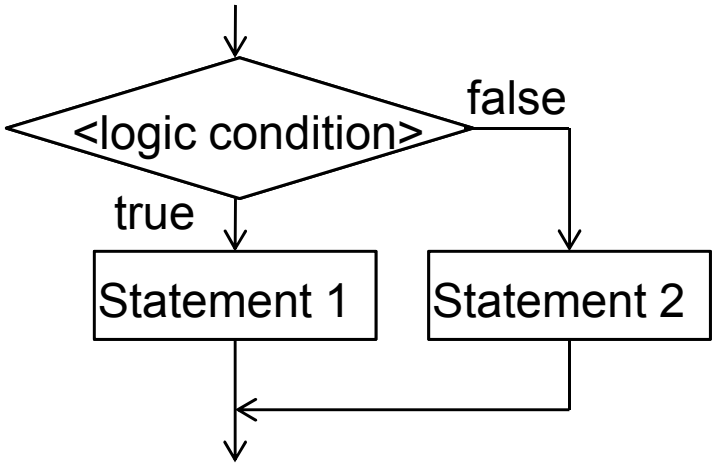


- if-else statement:
  - Syntax:

```
if (<logic condition>)
        <Statement 1>;
 [else
        <Statement 2>;]
```

Pseudo-code:

```
If <logic condition>
        Statement 1
 [Else
        Statement 2]
```



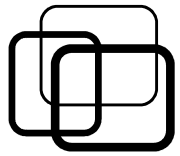
  - Examples:

```
// Full if-else
if (n > 0)
        a = a * 2;
else
        a = a / 2;
```

```
// No else
if (n > 0)
        a = a * 2;
```

```
// Compound statement
if (n > 0)
{
        a = a * 2;
        b = b + 1;
}
```

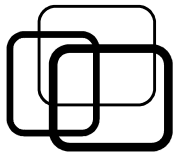
Branch statements



- if-else statement:
  - Notes:
    - Logic condition encloses ( ).
      - ➔ Value 1: true.
      - ➔ Value 0: false.
    - **if-else** compound statement.
      - ➔ No **;** after **if** or **else**.
    - **if-else** can be nested.
      - ➔ **else** goes with nearest **if**.

```
if  n > 0              // Wrong
        a = a * 2;

if (1)                 // Always true
        a = a * 2;

if (n > 0) ;           // Wrong
        a = a * 2;
else ;
        a = a / 2;

if (n > 0)         // Nested if-else
        if (a > b)
                c = c + 1;
        else
                c = c – 1;
```

Branch statements

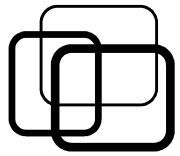


- if-else statement:
  - Nested if-else, logic condition on the same variable:

```
if (gpa >= 8.5)
        rank = “Excellent”;
else
        if (gpa >= 7.0)
                rank = “Good”;
        else
                if (gpa >= 5.0)
                        rank = “Fair”;
                else
                        rank = “Failed”;
```

```
if (gpa >= 8.5)
        rank = “Excellent”;
else if (gpa >= 7.0)
        rank = “Good”;
else if (gpa >= 5.0)
        rank = “Fair”;
else
        rank = “Failed”;
```

Branch statements

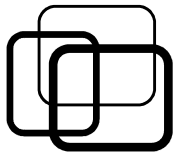


- switch-case statement:
  - Syntax:

```
switch (<expression>)
{
        [case <value 1>:
                <Statement 1>;
                break;
        case <value 2>:
                <Statement 2>;
                break;
        ....]
        [default:
                <Statement N>;]
}
```

```
// Equivalent if-else statement.
if (<expression> == <value 1>)
        <Statement 1>;
else if (<expression> == <value 2>)
        <Statement 2>;
…
else
        <Statement N>;
```

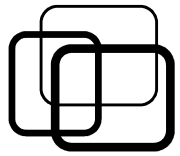
Branch statements



- switch-case statement:

```
switch (day_of_week)
{
        case 1:
                printf(“Sunday”); break;
        case 2:
                printf(“Monday”); break;
        case 3:
                printf(“Tuesday”); break;
        case 4:
                printf(“Wednesday”); break;
        case 5:
                printf(“Thursday”); break;
        case 6:
                printf(“Friday”); break;
        case 7:
                printf(“Saturday”); break;
}
```

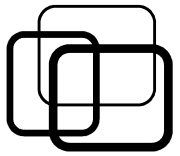
Branch statements



- switch-case statement:
  - Notes:
    - Expression encloses ( ).
    - Value of **case**:
      - ➔ Single value.
      - ➔ Not range value.
    - Statement **break**:
      - ➔ Exit the **case**.
      - ➔ Ignore to combine **case**.

```
switch a + b    // false
{
        …
}

switch (a + b)
{
        case > 5:     // false
                …
}
```

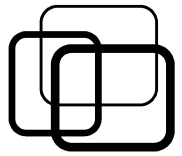
Branch statements



- switch-case statement:

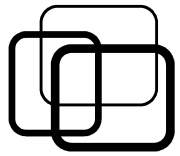
```
switch (day_of_week)
{
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
                printf("Working day"); break;
        case 1:
        case 7:
                printf("Weekend"); break;
        default:
                printf("Invalid day of week");
}
```

Contents



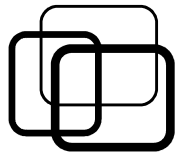
- Branch statements.
- **Loop statements.**

Loop statements



- Consider the program:
  - Print integers in range [1..10].
    - Write 10 output statements.
  - Print integers in range [1..100]?
    - Write 100 output statements?!
    - ➔ Use loop statement.

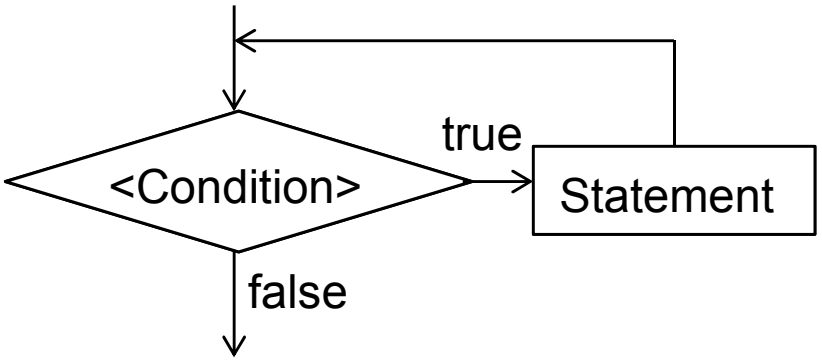
Loop statements



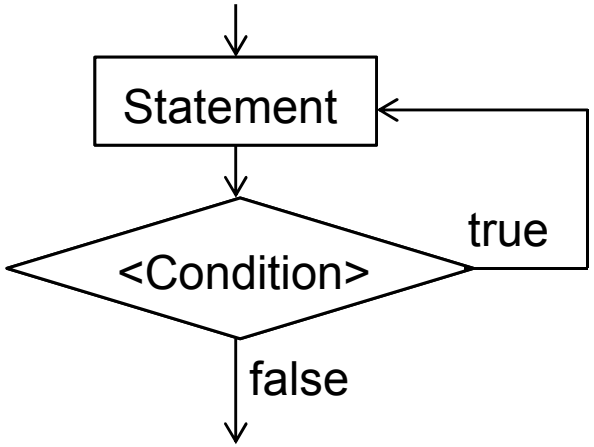
- while and do-while statements:
  - Syntax:

// while statement
while (<Condition>)
    <Statement>;

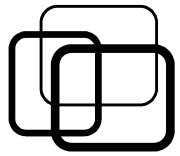
// Equivalent while for do-while
    <Statement>;
while (<Condition>)
    <Statement>;



// do-while statement
do
{
    <Statement>;
} while (<Condition>**);**



Loop statements

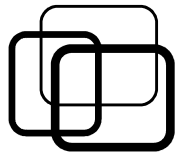


- while and do-while statements:
  - Examples:

```
// while statement.
printf("Enter n = ");
scanf("%d", &n);

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

```
// do-while statement.
printf("Enter n = ");
scanf("%d", &n);

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

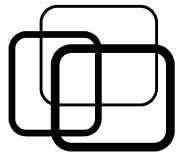
Loop statements



- while and do-while statements:
  - Notes:
    - Loop condition encloses ( ).
    - Loop often includes:
      - ➔ Step 1: Initialize counter.
      - ➔ Step 2: Check condition.
      - ➔ Step 3: Execute a loop.
      - ➔ Step 4: Update counter.

```
while  n > 0    // false
{
        …
}

k = 0;                  // Step 1
while (k < n)           // Step 2
{
        S = S * k;      // Step 3
        k = k + 1;      // Step 4
}
```

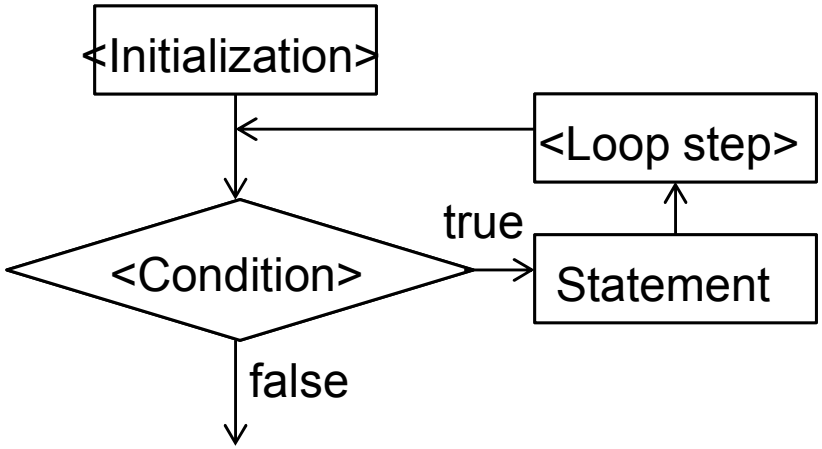
Loop statements



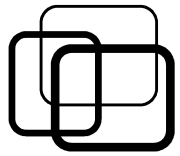
- for statement:
  - Syntax:

for ([<Initialization>] **;** [<Condition>] **;** [<Loop step>]**)**
    <Statement>;

// Equivalent while.
<Initialization>;
while (<Condition>**)**
{
    <Statement>;
    <Loop step>;
}



Loop statements



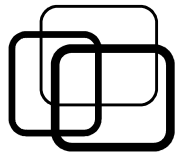
- for statement:
  - Examples:

```
printf("Enter n = ");
scanf("%d", &n);

// Full for.
for (i = 1; i <= n; i++)
        printf("%d", i);
```

```
printf("Enter n = ");
scanf("%d", &n);

// No initialization.
i = 1;
for ( ; i <= n; i++)
        printf("%d", i);
```

```
printf("Enter n = ");
scanf("%d", &n);

// No loop step.
i = 1;
for ( ; i <= n; )
{
        printf("%d", i);
        i++;
}
```

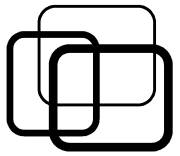
Loop statements



- break and continue statements:
  - break:
    - Exit the current loop.
    - Use with **if-else** for exit condition.
  - continue:
    - Skip the current loop one time.
    - Use with **if-else** for skip condition.

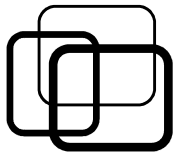
```
printf("Enter n = ");
scanf("%d", &n);

for (i = 1;  ; i++)
{
        if (i > n)
                break;
        if (i % 2 == 0)
                continue;

        printf("%d", i);
}
```

Summary



- Branch statements:
  - if-else: two branches.
  - switch-case: multiple branches.
- Loop statements:
  - while: condition is checked before the loop.
  - do-while: condition is checked after the loop.
  - for:
    - Initialize counter.
    - Check condition.
    - Execute loop.
    - Update counter.
  - break, continue: use with if-else in a loop.

Practice



Practice 3.1:

Write C/C++ program to simulate a calculator as follow:

- Enter two integers.
- Enter an operator (+, -, *, /, %).
- Perform the operator on two integers and print result.

Note: flush the standard input stream after each input.

- C: fgets, or while getchar, C++: cin.getline, or cin.ignore.

Input format:

Enter two integers = 7 5

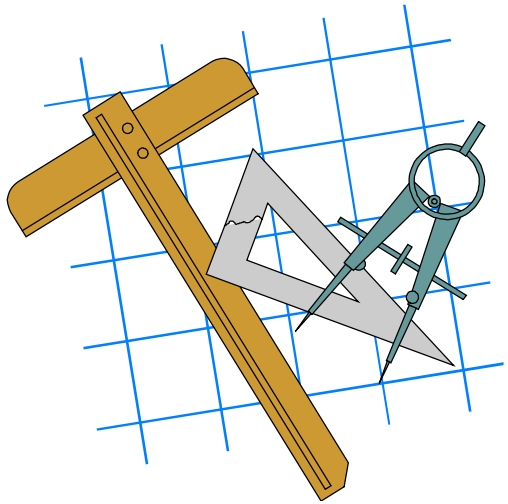
*Enter an opertor (+, -, *, /, %) = +*

Output format (no error):

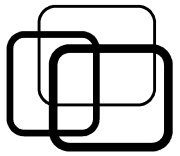
Result = 12

Output format (divided-by-zero error):

Error: divided by zero.



Practice



- **Practice 3.2:**

  Write C/C++ program to solve quadratic equation: $ax^2 + bx + c = 0$.

  Input format:

  Enter coefficients a, b, c = 2 -5 3

  Output format (2 solutions):

  Solution 1 = 1

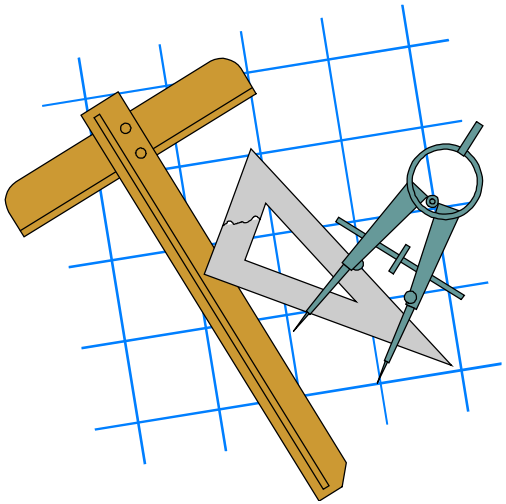
  Solution 2 = 1.5

  Output format (1 solution):

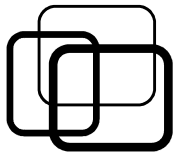
  Solution 1 = <result>

  Output format (no solution):

  No solution!



Practice



- ## Practice 3.3:

Write C/C++ program to count the number of days in a month:

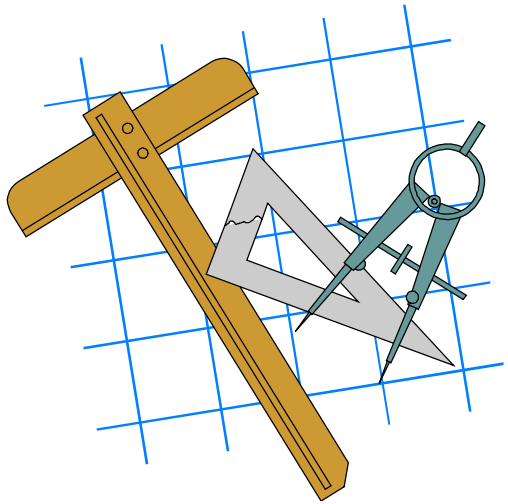
- Enter month and year.
- Count the number of days in the month and print result.

Input format:

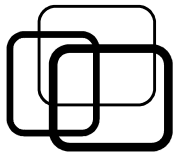
Enter month and year = 6 2012

Output format:

*Month **6** in year **2012** has **30** days.*



Practice



- ## Practice 3.4:

Write C/C++ program as follow:

- Enter a positive integer N.
- Compute and print results:
  - a) N! = 1 * 2 * … * N.
  - b) ln(2) = 1 - 1/2 + 1/3 - … +/- 1/N.
  - c) PI = 4 (1 – 1/3 + 1/5 – … +/- 1/(2*N + 1)).
  - d) S = $a_1 + a_2 + \ldots a_k$ ({ a_i } are all square numbers <= N).

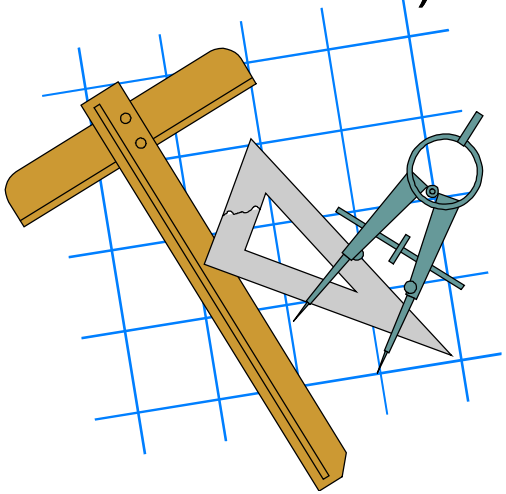
Output format:

N! = <a result>

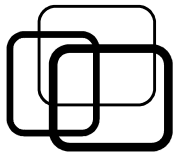
ln(2) = <b result>

PI = <c result>

S = <d result>



Practice



- **Practice 3.5:**

  Write C/C++ program to find all numbers satisfying:

  - A 3-digit positive integer.
  - Tens digit = Hundreds digit + Ones digit.

  Note: how to delay output?

  - C on Windows: Sleep(<millisecond>) (<windows.h>).
  - C on Linux: usleep(<microsecond>) (<unistd.h>).
  - C++: std::this_thread::sleep_for(std::chrono::milliseconds(<value>)).

  Output format:

  1: <the 1st satisfied number>

  2: <the 2nd satisfied number>

  …

  N: <the Nth satisfied number>

  There are N satisfied numbers.