CS2311 Computer Programming

LT04: Control Flow - Loop

Computer Science, City University of Hong Kong

Semester B 2022-23

Quick Review: Control Flow - Condition

- Logical data type, operators and expressions
- If statement
    - Simple
    - Nested
- Switch statement

Quick Review: Logical Data Type: *bool*

- Takes only two values: *true* and *false*
- Numeric values are *1* (true) and *0* (false)
- the *lowest-ranked* data type
- Length: 1 byte

```
bool x = false, y = true;
cout << sizeof(bool) << endl; // 1
cout << x << " " << y << endl; // 0 1
cout << x + 6 << " " << y + 3.14; // 6 4.14
```

9. long double
8. double
7. float

6. long long
5. long
4. int
3. short

2. char

1. bool

Quick Review: Logical Data Type: *bool*

- when a *higher-ranked* type is casted to *bool*, only 0 is converted to false, all non-zero values are converted to true

```
bool x = 0, y = 3.14, z = 0x1100;
cout << x << " " << y << " " << z << endl; // 0 1 1
```

- different from demoted conversion of numeric types, which is *direct cut*

```
short a = 0xab0011;
cout << a; // 17
```

9. long double
8. double
7. float

6. long long
5. long
4. int
3. short

2. char

1. bool

Quick Review: Comparative Operators

- Binary operators which accept two operands and compare them

relational operators	syntax	example
Less than	`<`	`x < y`
Greater than	`>`	`z > 1`
Not greater than	`<=`	`b <= 1`
Not less than	`>=`	`c >= 2`

equality operators	syntax	example
Equal to	`==`	`a == b`
Not equal to	`!=`	`B != 3`

Quick Review: Logical Operators: AND (&&), OR (||), NOT (!)

- Used for *combining* two logical values and *create* a new logical values
- Logical AND (&&)
  - return true if both operands are true
  - otherwise return false
  - example: 18 < age && age < 60
- Logical OR (||)
  - return false if both operands are false
  - otherwise return true
- Logical-NOT (!) is a *unary* operator that *takes one operand* and *inverts its value*

```
! (A && B) is the same as (!A) || (!B)
! (A || B) is the same as (!A) && (!B)
```

Quick Review: && VS &

- What are the outputs?

```
int x = 0, y = 3, z = 2;

cout << (x&&y) << endl;
cout << (y&z) << endl;
```

x && y is (false) && (true), which is false = 0
y & z is (11_2) & (10_2), which is $10_2 = 2$

- & operator is a bitwise operator, while && operator is a logical operator.

Quick Review: Logical Expressions

- Expressions that take comparative or logical operators
  - `x == 3`
  - `y == x`
  - `x >= 2`
  - `x != y`
- The value of a logical expression is bool, i.e., can be true or false only
- DO NOT MIX: x=y VS x==y
- DO NOT MIX: a<x<b VS a<x && x<b

Quick Review: What are the outputs?

```
int x = 0, y = 3, z = 3;

cout << (x=y) << endl;
cout << (y==z) << endl;
```

```
// First will print 3, because:
// x=y is an assignment expression.
// It copies the value of y to x.
// The value of an assignment expression equals to
// the value of the right operand, i.e., 3

// Second will print 1 because:
// y==z is a logical expression!
// and y does equal to z
```

```
double a = 1;

cout << (0<a && a<1) << endl;
cout << (0<a<1) << endl;
```

```
// First will print 0, because:
// the value of 0 < a is true (0<1)
// the value of a < 1 is false (1<1)
// the operator && combines the two values
// and creates a new value
// which is false and printed as 0.

// Second will print 0, because:
// 0 < a < 1 is equivalent to (0 < a) < 1
// in this example, it’s equivalent to (0<1) < 1
// i.e., true < 1, which equals to false
// and printed as 0
```

Quick Review: Short-circuit evaluation

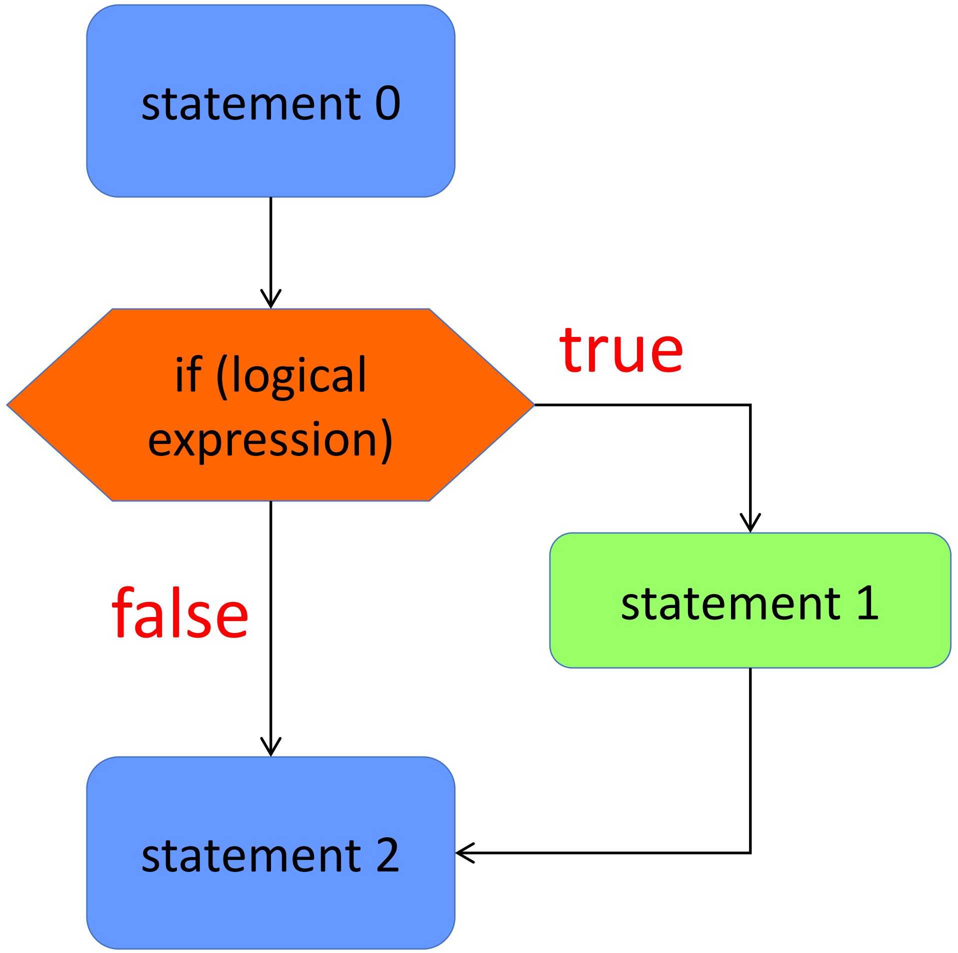
- Short-circuit evaluation can improve program efficiency
- Short-circuit evaluation exists in some other programming languages, e.g., C and Java
- Evaluation of logical expressions containing && and || stops as soon as the outcome true or false is known
- For &&: the value of x&&y is false as long as x is false in this case, the value of y doesn't matter and is NOT evaluated
- For ||: the value of x||y is true as long as x is true in this case, the value of y doesn't matter and is NOT evaluated

Quick Review: if Statement - Basic Syntax

```
statement 0;
if (logical expression)
    statement 1;
statement 2;
```

- statement 1 will be executed if logical expression is evaluated to true, for example

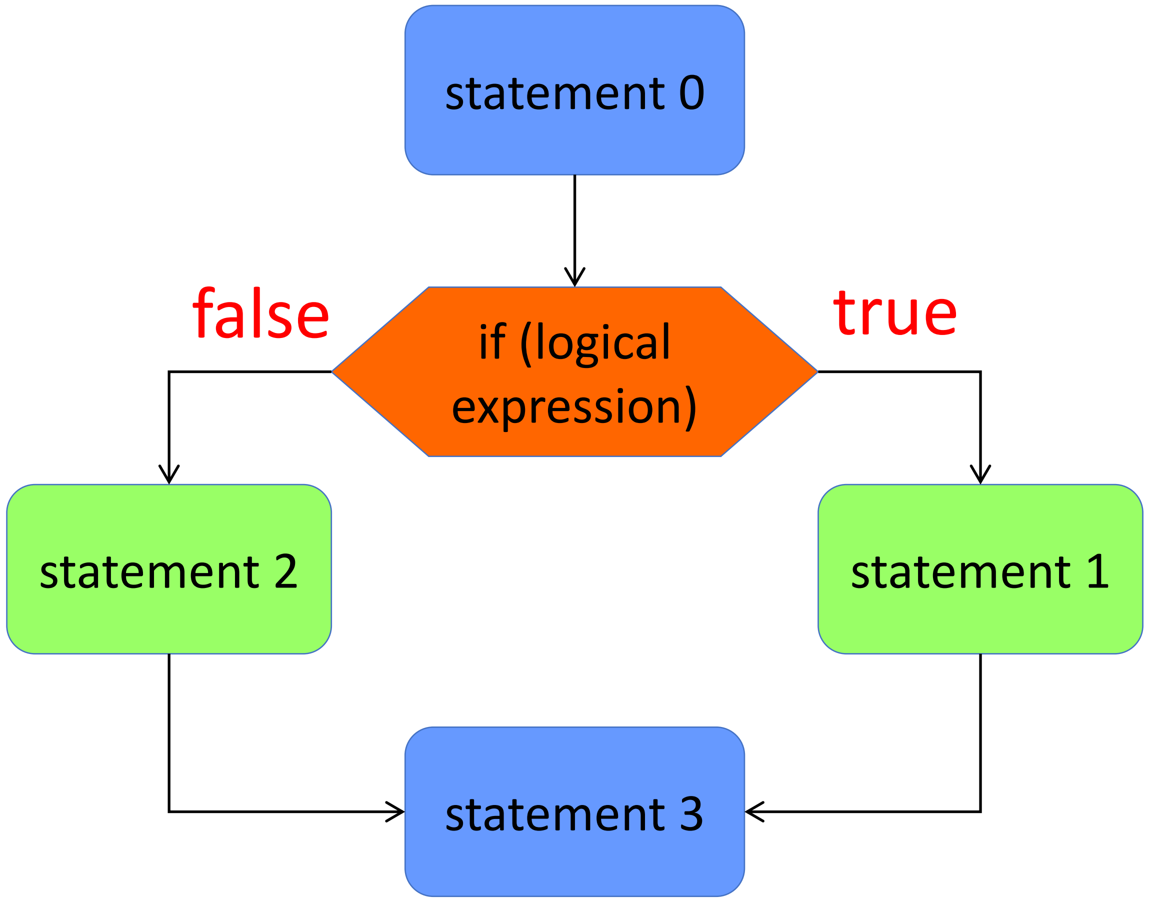
```
cin >> x;
if (x < 0)
    x = -x;
cout << x;
```



Quick Review: if Statement - Two-way Condition

```
statement 0;
if (logical expression)
    statement 1;
else
    statement 2;
statement 3;
```

- if logical expression is true, statement 1 will be executed
- If logical expression is false, statement 2 will be exected



Quick Review: if Statement - Inline Ternary

- Also known as *inline* if-then-else constructs

- Syntax
  - expr1 ? expr2 : expr3 ;

- Semantics
  - expr1 is evaluated as the condition
  - if the value of expr1 is non-zero/true, then execute expr2;
  - else execute expr3

```
int a, b, c;
cin >> a;
cin >> b;
a>=b ? c=a : c=b;
cout << c;
```

Quick Review: Compound if

- Group multiple statements into one block using {} to be executed for one branch

```
if (logical expression) {
        statement 1;
        …
        statement n;
} else {
        statement n+1;
        …
        statement n+m;
}
```

We may group **multiple statements** to form a **compound statement** using a pair of braces {}

```
if (j!=3){
   b++;
   cout << b;
}
else
   cout << j;
```

Compound statements are treated as if it were a **single statement**

```
if (j!=5&&d==2) {
   j++;
   d--;
   cout<<j<<d;
} else {
   j--;
   d++;
   cout<<j<<d;
}
```

Quick Review: if Statement - Multi-way Condition

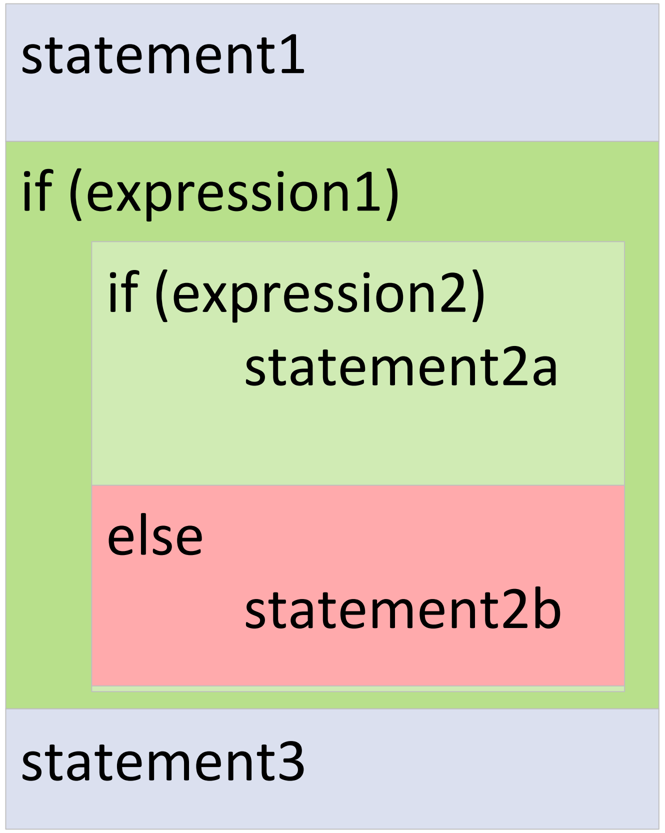
- In C++, multi-way condition can be constructed as,

```
if (logical expression 1) {
   statements when expression 1 is true
}
else if (logical expression 2) {
   statements when expression 1 is false and expression 2 is true
}
else {
   statements when both logical expression 1 and 2 are false
}
```

- Don’t forget to use { } to enclose the statements

Quick Review: Nested if

- An if-else statement is included within another if or else statement



```
statement1

if (expression1)
    if (expression2)
        statement2a

    else
        statement2b

statement3
```

```
if (mark>=90 && mark<=100) {
  // divide A into can be A-, A or A+
  if (mark>97)
      cout << "You get grade A+\n";
  else if (mark>93)
      cout << "You get grade A \n";
  else
      cout << "You get grade A-\n";
}
```

- An *else* is attached to the nearest *if*

Quick Review: switch - Syntax and Semantic

- Syntax

```
    switch (expression) {
        case constant-expr1:
          statements
          break;     // optional
        ...
        case constant-exprN:
           statements
           break;    // optional
        default:     // optional
           statements
           break;    // optional
    }
```

- Semantic
  - Evaluate the expression which results in an integer type (`int`, `long`, `short`, `char`)
  - Go to the case label having a constant value that matches the value of expression;
  - when a break statement is encountered, terminate the switch
  - If there is no break statement, execution falls through to the next statement
  - if a match is not found, go to the default label;
  - if default label does not exist, terminate the switch

Quick Review: Summary

- Boolean logic has two values only; true or false.

- Conditional statements are the statements that only execute under certain conditions.

- In C++, there are two approaches to construct conditional statement
  - if (…){…}else{…}
  - switch(…){case:break…}

Quick Review: Exercise 1

- Determine the outputs, assuming we provide inputs for `count` as 0 and `limit` as 10

```
int count, limit;
cin >> count >> limit;
if ((count=1) && (limit < 0))
  cout << limit + count << endl;
else if ((count == 1) || (limit = 0))
  cout << count - limit << endl;
else
  cout << limit / (count*1.0) << endl;
```

- *limit* is 10,
- *count* is 1

* credit: Savitch, Walter J. Absolute C++. Pearson Education, 2006.

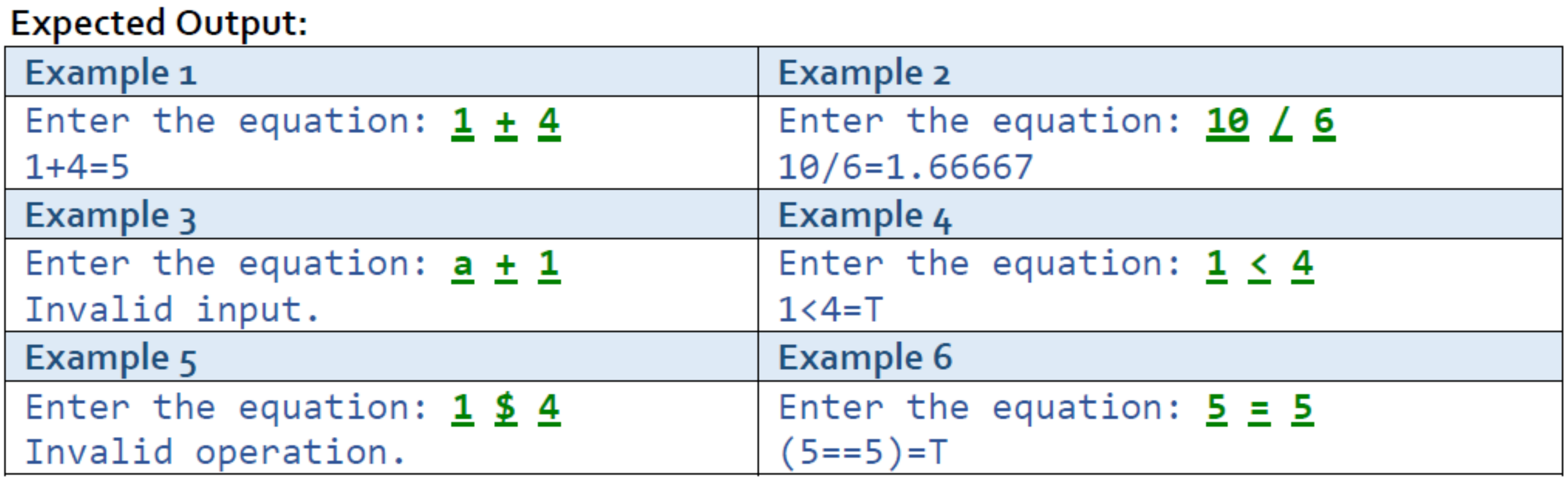
Quick Review: Exercise 2

```
// What happens there is no break ??
int day_of_week;
cin >> day_of_week;
switch (day_of_week) {
    case 1:  cout << "Monday\n";
    case 2:  cout << "Tuesday\n";
    case 3:  cout << "Wednesday\n"; break;
    case 4:  cout << "Thursday\n";
    case 5:  cout << "Friday\n";
    case 6:  cout << "Saturday\n";
    case 7:  cout << "Sunday\n";
    default: cout << "Invalid\n";
}  // end switch
```

- What are the outputs if we enter ‘2’?
- What are the outputs if we enter ‘5’?

Recap: Lab 4 Q1

- write a program that calculates the result of 'a', 'operator', 'b'



Expected Output:

Example 1	**Example 2**

`Enter the equation: 1 + 4`
`1+4=5`
	`Enter the equation: 10 / 6`
	`10/6=1.66667`

Example 3	**Example 4**

`Enter the equation: a + 1`
`Invalid input.`
	`Enter the equation: 1 < 4`
	`1<4=T`

Example 5	**Example 6**

`Enter the equation: 1 $ 4`
`Invalid operation.`
	`Enter the equation: 5 = 5`
	`(5==5)=T`

Recap: Lab 4 Q1

- To verify whether the input is a number

```
int x; cin >> x;
if (cin)
```

…

- OR

```
int x;
if (cin >> x)
    …
```

Recap: Lab 4 Q1

```
...
Declare variable a and b, and o;
Get a, o, b from input
Check whether input 'a' and 'b' are digits.
→YES, then:
  o is the case '<': then do sth;
  o is the case '>': then do sth;
  ...
  o is none of the above case: then do sth;
→NO, then:
  Output sth
```

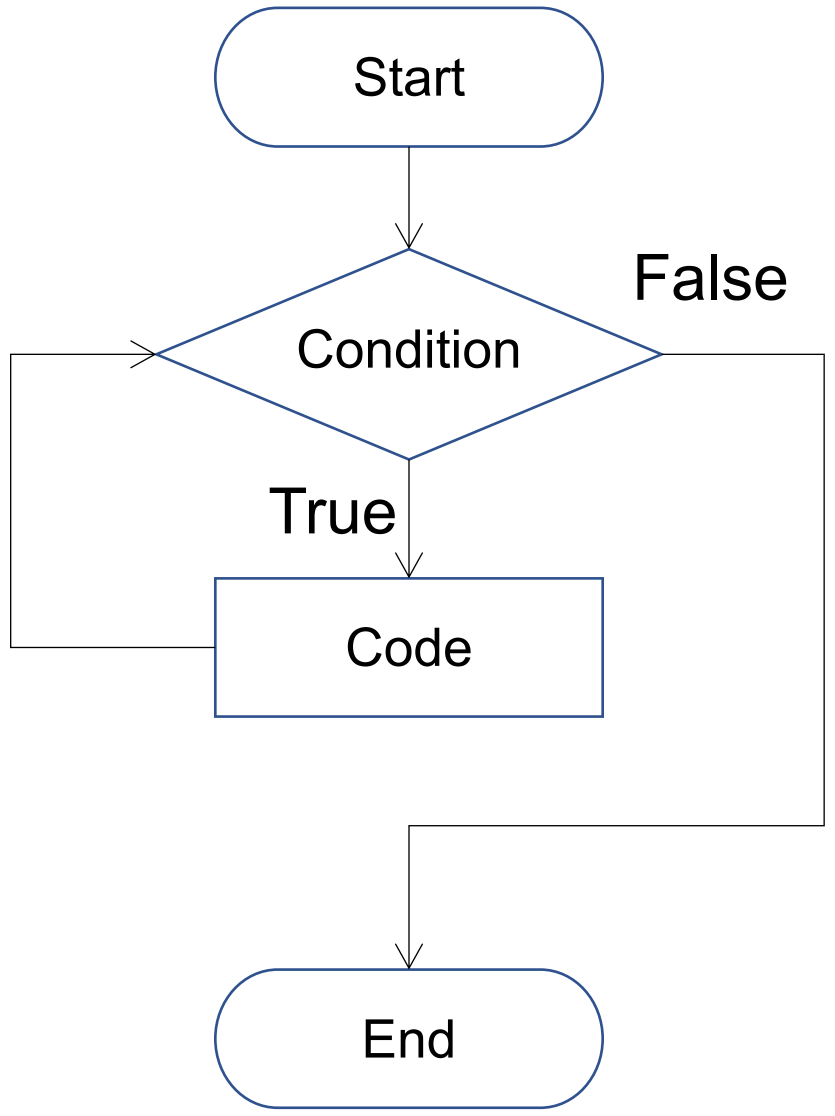
Today's Outline

- Loop
  - while
  - do-while
  - for
- Programming styles for control flow

Loop

- When the execution enters a loop, it executes a block of code repeatedly as long as a loop condition is met
    - ➢ Loop body
    - ➢ Iteration

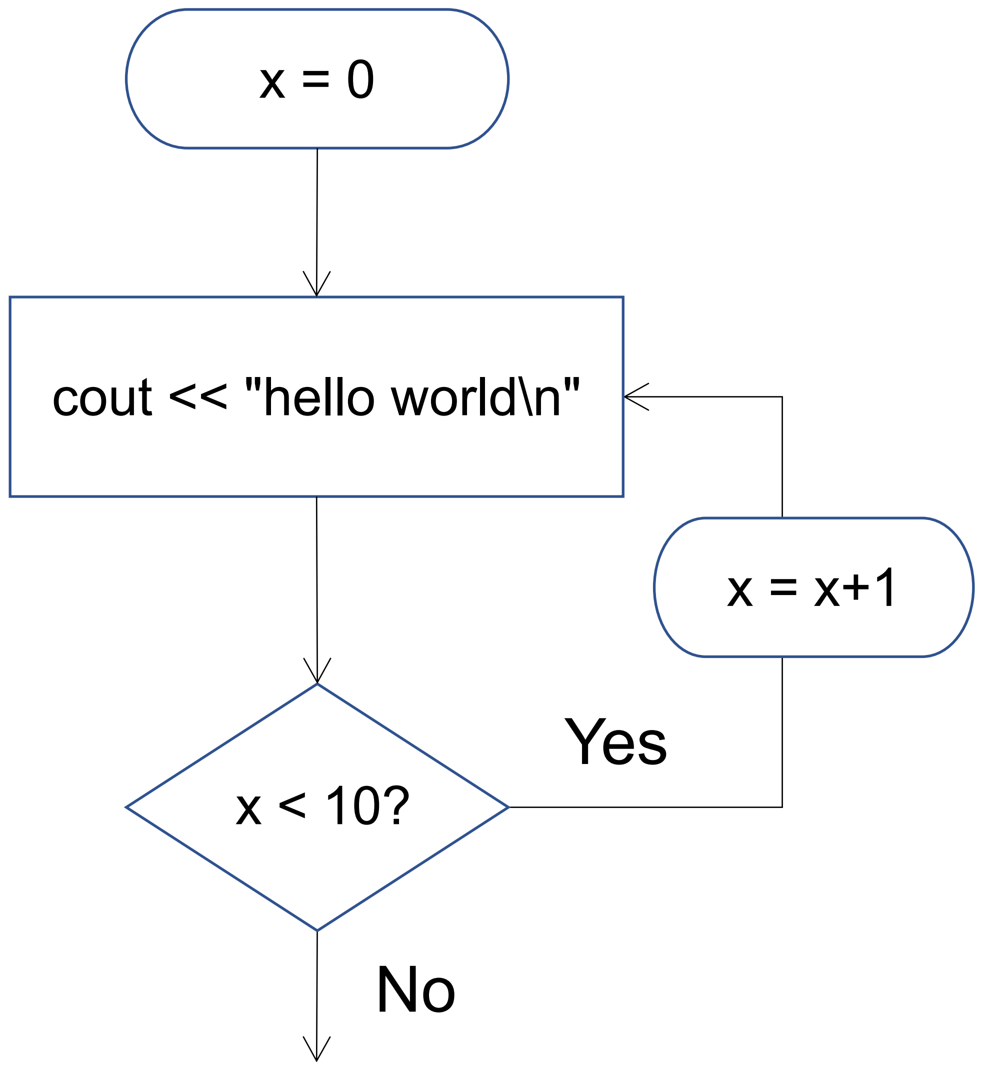
- Beside sequential and branch execution loop is another common control flow



Loop (cont'd)

- Print "hello world" 10 times

1. Set x=0;
2. cout << "hello world\n"
3. if (x < 10) then add 1 to x and loop back
4. Else exit the loop



Loop (cont'd)

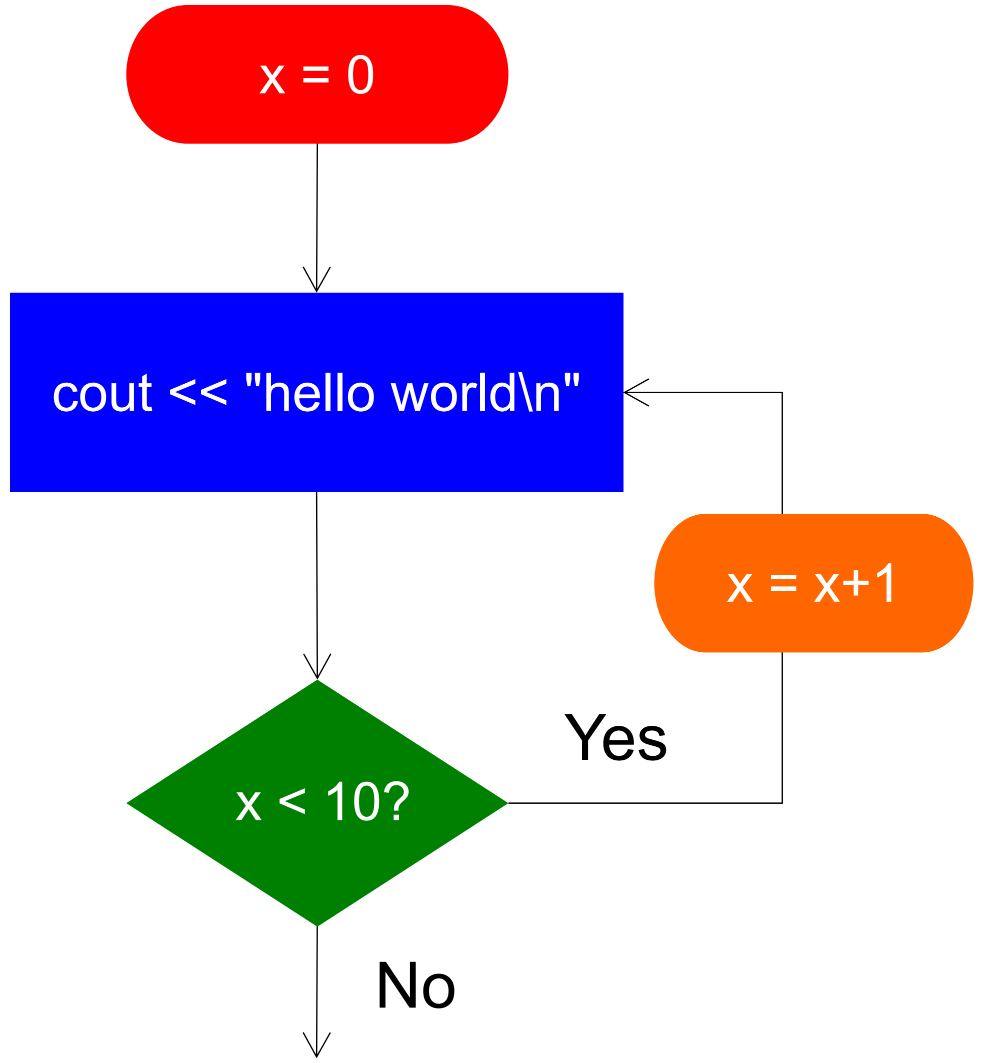
- In general, a loop consists of four parts

initialization statements

body

loop condition

post loop statements (stepping forward to exit loop condition)



Types of Loop

- while loop
- do-while loop
- for loop

`while`

- Syntax

```
while (expression)
{
   loop statement(s);
}
```

- Semantics
  - If the value of `expression` is non-zero (true), `loop statements` will be executed, otherwise, the loop terminates
  - After loop `statements` are executed, the `expression` will be tested again

while

- An Example:
- Ask user to input positive integers
- Stop when user enters '0'
- Print the maximum of entered positive integers before quit

```
int main() {
  int x, max;
  max = 0;










  return 0;
}
```

while

- An Example:
- Ask user to input positive integers
- Stop when user enters '0'
- Print the maximum of entered positive integers before quit

```
int main() {
  int x, max;
  max = 0;
  cout << "Enter a positive integer. ";
  cout << "Type 0 to quit.\n";
  cin >> x;

















  return 0;
}
```

while

- An Example:
- Ask user to input positive integers
- Stop when user enters '0'
- Print the maximum of entered positive integers before quit

```
int main() {
  int x, max;
  max = 0;
  cout << "Enter a positive integer. ";
  cout << "Type 0 to quit.\n";
  cin >> x;
  while (x != 0) {






  }






  return 0;
}
```

while

- An Example:
- Ask user to input positive integers
- Stop when user enters '0'
- Print the maximum of entered positive integers before quit

```
int main() {
  int x, max;
  max = 0;
  cout << "Enter a positive integer. ";
  cout << "Type 0 to quit.\n";
  cin >> x;
  while (x != 0) {
    if (x > max)
      max = x;





  }








  return 0;
}
```

while

- An Example:
- Ask user to input positive integers
- Stop when user enters '0'
- Print the maximum of entered positive integers before quit

```
int main() {
  int x, max;
  max = 0;
  cout << "Enter a positive integer. ";
  cout << "Type 0 to quit.\n";
  cin >> x;
  while (x != 0) {
    if (x > max)
      max = x;
    cout << "Enter a positive integer. ";
    cout << "Type 0 to quit.\n";
    cin >> x;
  }







  return 0;
}
```

while

- An Example:
- Ask user to input positive integers
- Stop when user enters '0'
- Print the maximum of entered positive integers before quit

```
int main() {
  int x, max;
  max = 0;
  cout << "Enter a positive integer. ";
  cout << "Type 0 to quit.\n";
  cin >> x;
  while (x != 0) {
    if (x > max)
      max = x;
    cout << "Enter a positive integer. ";
    cout << "Type 0 to quit.\n";
    cin >> x;
  }
  if (max == 0) {
    cout << "You didn't enter any positive integer\n";
  } else {
    cout << "The maximum integer you entered is ";
    cout << max << endl;
  }
  return 0;
}
```

do-while

- Syntax

```
do {
   loop statement(s);
}
while (expression);
```

- Semantics
  - `loop statements` are executed first; thus the loop body will be executed for at least once
  - If the value of `expression` is non-zero (true), the loop repeats; otherwise, the loop terminates

do-while

- An Example:
- Ask user to input positive integers
- Stop when user enters '0'
- Print the maximum of entered positive integers before quit

```
int main() {
  int x, max;
  max = 0;

  do {
    cout << "Enter a positive integer. ";
    cout << "Type 0 to quit.\n";
    cin >> x;
    if (x > max)
      max = x;
  } while (x != 0);

  if (max == 0) {
    cout << "You didn't enter any positive integer\n";
  } else {
    cout << "The maximum integer you entered is ";
    cout << max << endl;
  }

  return 0;
}
```

while vs do-while

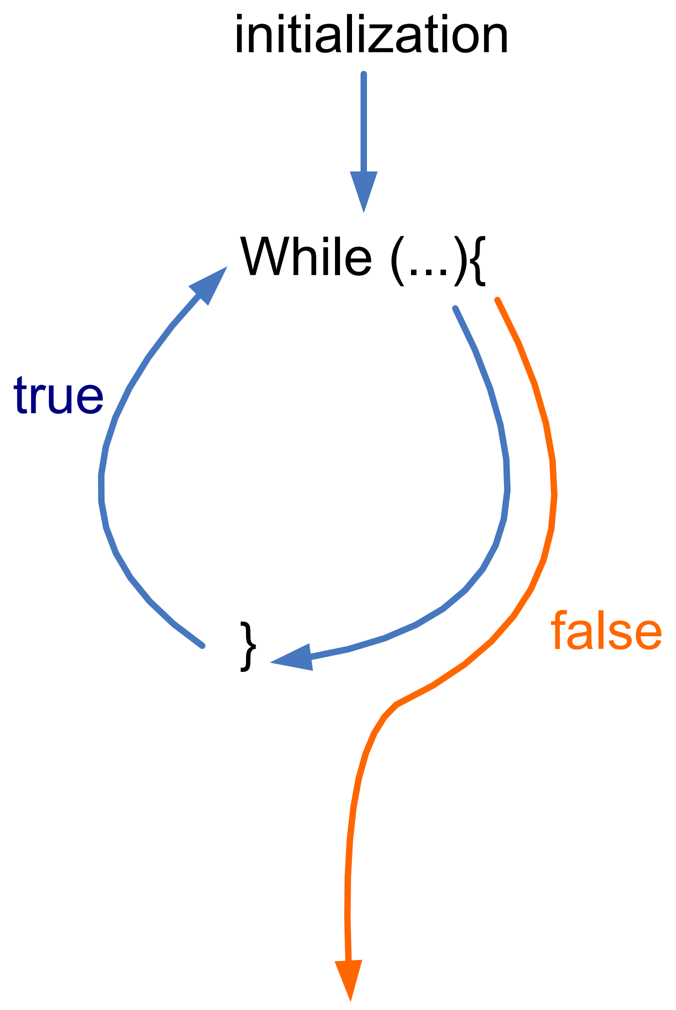
```
int x, max;
max = 0;
cout << "Enter a positive integer. ";
cout << "Type 0 to quit.\n";
cin >> x;
while (x != 0) {
  if (x > max)
    max = x;
  cout << "Enter a positive integer. ";
  cout << "Type 0 to quit.\n";
  cin >> x;
}
```

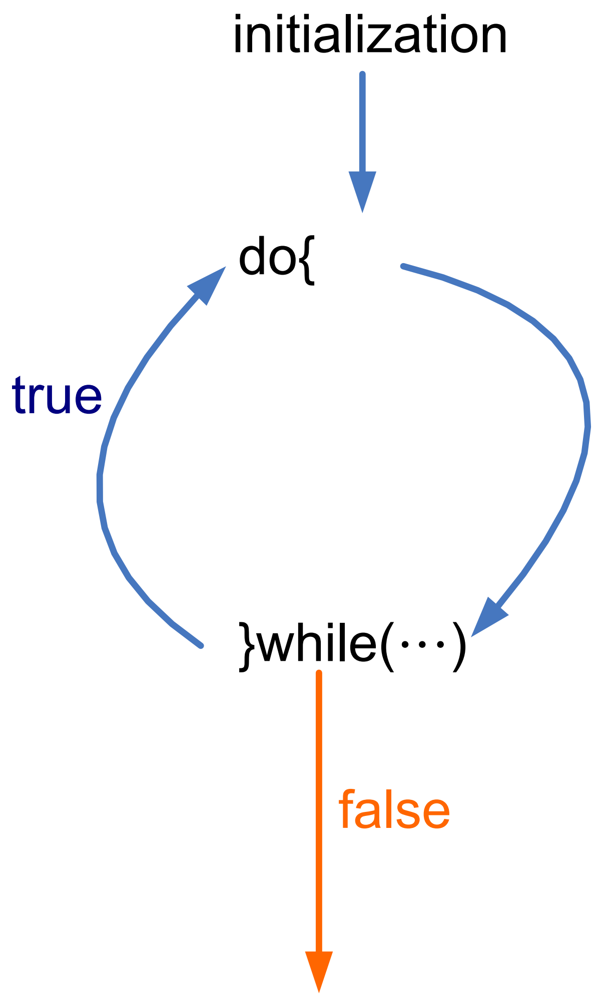
```
int x, max;
max = 0;


do {
  cout << "Enter a positive integer. ";
  cout << "Type 0 to quit.\n";
  cin >> x;
  if (x > max)
    max = x;
} while (x != 0);
```

- do-while is better suited for loops that require at least one iteration

while vs do-while





Exercise:

What are the outputs?

```
int x = 10;
do
{
      cout << x << endl;
      x = x - 3;
} while (x > 0);
```

for: Syntax

```
for (expr1; expr2; expr3) {
    loop statements;
}
```

- Semantics

Loop statements are repeatedly executed as long as `expr2` is non-zero (true). Otherwise, the loop ends.

`expr1`: executed before entering the loop body. Often used for initializing a loop counter or loop status.

`expr3`: executed after each iteration of the loop body. Often used to update the loop counter or loop status.

for: Examples

```
#include <iostream>
using namespace std;
int main() {
   int i;
   for (i=0;i<10;i++) {
      if(i%2==0)
         cout << i << endl;
   }
   return 0;
}
```

for: Examples

```
#include <iostream>
using namespace std;
int main() {
   int i;
   for (i=0;i<10;i++) {
      if(i%2==0)
         cout << i << endl;
   }
   return 0;
}
```

```
#include <iostream>
using namespace std;
int main() {

   for(int i=0;i<10;i++) {
      if(i%2==0)
         cout << i << endl;
   }
   return 0;
}
```

```cpp
#include <iostream>
using namespace std;
// get input from user until a positive integer is entered
int main() {
    int x;
    cout << "Enter a number: ";
    cin >> x;
    while (x <= 0) {
        cout << "Input must be positive." << endl;
        cout << "Enter number: ";
        cin >> x;
    }
    return 0;
}
```

```
#include <iostream>
using namespace std;
// get input from user until a positive integer is entered
int main() {
    int x;
    cout << "Enter a number: ";
    cin >> x;
    while (x <= 0) {
        cout << "Input must be positive." << endl;
        cout << "Enter number: ";
        cin >> x;
    }
    // for-loop equivalent to the above while-loop
    for (cin >> x;                    ) {



    }
    return 0;
}
```

🟨 initialization

🟦 loop condition

⬜ body

🟥 post loop statements

```
#include <iostream>
using namespace std;
// get input from user until a positive integer is entered
int main() {
    int x;
    cout << "Enter a number: ";
    cin >> x;
    while (x <= 0) {
        cout << "Input must be positive." << endl;
        cout << "Enter number: ";
        cin >> x;
    }
    // for-loop equivalent to the above while-loop
    for (cin >> x; x <= 0;          ) {


    }
    return 0;
}
```

🟨 initialization

🟦 loop condition

⬜ body

🟥 post loop statements

```cpp
#include <iostream>
using namespace std;
// get input from user until a positive integer is entered
int main() {
    int x;
    cout << "Enter a number: ";
    cin >> x;
    while (x <= 0) {
        cout << "Input must be positive." << endl;
        cout << "Enter number: ";
        cin >> x;
    }
    // for-loop equivalent to the above while-loop
    for (cin >> x; x <= 0; cin >> x) {
        cout << "Input must be positive." << endl;
        cout << "Enter number: ";
    }
    return 0;
}
```

🟨 initialization

🟦 loop condition

⬜ body

🟥 post loop statements

`for`: Examples

- Aside from using `int` as loop counters, we can also use other integral types

```
for (char ch='a'; ch<='z'; ch++)
{
      cout << ch << endl;
}
```

for: Syntax (cont'd)

```
for (expr1; expr2; expr3) {
   loop statements;
}
```

- expr1 and expr3 can contain multiple statements. Each statement is separated by a comma ','
- Example

```
for (int i=0, j=0; i<10; i++, j++)
    …
```

for: Examples (cont'd)

- *Palindrome string*: a string is palindrome if the reverse of that string is the same as the original (e.g., abcba)
- Check if a string consisting of 5 characters is palindrome or not

```
char str[5];
bool is_palindrome;
cout << "Input 5 letters: ";
for (int i=0; i<5; i++) {
   cin >> str[i];
}
```

for: Examples (cont'd)

- *Palindrome string*: a string is palindrome if the reverse of that string is the same as the original (e.g., abcba)
- Check if a string consisting of 5 characters is palindrome or not

```
char str[5];
bool is_palindrome;
cout << "Input 5 letters: ";
for (int i=0; i<5; i++) {
   cin >> str[i];
}
is_palindrome = true;
for (int i=0, j=4; i<5; i++, j--) {
   is_palindrome &= (str[i]==str[j]);
}
cout << "It's";
cout << (is_palindrome ? " ":" NOT ");
cout << "palindrome\n";
```

for: Nested Loop

```
int i, j;
for (i=0; i<3; i++) {
  cout << "Outer for: \n";
  for (j=0; j<2; j++) {
    cout << "Inner for: ";
    cout << "i=" << i << ", j=" << j << endl;
  } // end of inner loop
  cout << endl;
} // end of outer loop
```

```
Outer for:
Inner for:i=0, j=0
Inner for:i=0, j=1

Outer for:
Inner for:i=1, j=0
Inner for:i=1, j=1

Outer for:
Inner for:i=2, j=0
Inner for:i=2, j=1
```

- The outer loop is executed 3 times. In each iteration of the outer loop, the inner loop is executed 2 times

for: Nested Loop (Example)

- Write a program to generate a *n* x *n* diagonal matrix (*n* is input by the user), where the element at the *i*-th row is *i*. Assume *n* > 1 and *n* <= 9
- E.g., when *n* = 5, the following matrix is generated

```
1
    2
        3
            4
                5
```

for: Nested Loop (Example)

- Write a program to generate a *n* x *n* diagonal matrix (*n* is input by the user), where the element at the *i*-th row is *i*. Assume *n* > 1 and *n* <= 9

- Solution

```
int n;
cin >> n;
for (int row=1; row<=n; row++) {
   for (int col=1; col<=row-1; col++)
      cout << " ";
   cout << row << endl;
}
```

for: Nested Loop (Example)

- Write a program to generate a *n* x *n* diagonal matrix (*n* is input by the user), where the element at the *i*-th row is *i*. Assume *n* > 1 and *n* <= 9
- Solution

```
int n;
cin >> n;
for (int row=1; row<=n; row++) {
    for (int col=1; col<=row-1; col++)
        cout << " ";
    cout << row << endl;
}
```

```
1
```

```
// row-1=0
```

for: Nested Loop (Example)

- Write a program to generate a *n* x *n* diagonal matrix (*n* is input by the user), where the element at the *i*-th row is *i*. Assume *n* > 1 and *n* <= 9
- Solution

```
int n;
cin >> n;
for (int row=1; row<=n; row++) {
    for (int col=1; col<=row-1; col++)
        cout << " ";
    cout << row << endl;
}
```

```
1
 2
```

```
// row-1=1
```

for: Nested Loop (Example)

- Write a program to generate a *n* x *n* diagonal matrix (*n* is input by the user), where the element at the *i*-th row is *i*. Assume *n* > 1 and *n* <= 9

- Solution

```
int n;
cin >> n;
for (int row=1; row<=n; row++) {
    for (int col=1; col<=row-1; col++)
        cout << " ";
    cout << row << endl;
}
```

```
1
  2
    3
```

```
// row-1=2
```

for: Nested Loop (Example)

- Write a program to generate a *n* x *n* diagonal matrix (*n* is input by the user), where the element at the *i*-th row is *i*. Assume *n* > 1 and *n* <= 9
- Solution

```
int n;
cin >> n;
for (int row=1; row<=n; row++) {
   for (int col=1; col<=row-1; col++)
      cout << " ";
   cout << row << endl;
}
```

```
1
 2
  3
   4
```

// row-1=3

for: Nested Loop (Example)

- Write a program to generate a *n* x *n* diagonal matrix (*n* is input by the user), where the element at the *i*-th row is *i*. Assume *n* > 1 and *n* <= 9

- Solution

```
int n;
cin >> n;
for (int row=1; row<=n; row++) {
    for (int col=1; col<=row-1; col++)
        cout << " ";
    cout << row << endl;
}
```

```
1
 2
  3
   4
    5    // row-1=4
```

for: Common Errors

- Scope of loop counter declaration

```
for (int k=1; k<=8; k++)
    cout << "log(" << k << ") = " << log(1.0*k) << endl;
cout << k << endl;  // SYNTAX ERROR
```

for: Common Errors

The variable k is declared within the for-loop. It is not visible/accessible outside the for-loop.

- Scope of loop counter declaration

```
for (int k=1; k<=8; k++)
    cout << "log(" << k << ") = " << log(1.0*k) << endl;
cout << k << endl;  // SYNTAX ERROR

// Variable k can be declared before the for-loop
int k=0;
for (k=1; k<=8; k++)
    cout << "sqrt(" << k << ") = " << sqrt(k) << endl;
cout << k << endl;
```

for: Common Errors (cont'd)

- Unaware of extra semi-colons, e.g.

```
int sum = 0, j;
for (j=1; j<=10; j++)
        sum += j;
```

Is NOT the same as

If output variable *sum*, then?

```
int sum = 0 , j;
for (j=1; j<=10; j++) ;
        sum += j;
```

for: Common Errors (cont'd)

- Unreachable loop termination condition => unintended infinite loop
- Example I

```
for (char i=0; i<256; ++i)
{
    cout << "i= " << i << endl;
}
```

for: Common Errors (cont'd)

- Unreachable loop termination condition => unintended infinite loop
- Example II

```
for (unsigned int i=100; i>=0; --i)
{
    cout << "i= " << i << endl;
}
```

for: Common Errors (cont'd)

- Unreachable loop termination condition => unintended infinite loop
- Example III

```
int iter=0;
for (float i=0.0; i!=0.000001; i+=0.0000001)
  cout << "This is the  " << ++iter << " iteration\n";
```

for: Common Errors (cont'd)

- Unreachable loop termination condition => unintended infinite loop
- Example III

```
int iter=0;
for (float i=0.0; i!=0.000001; i+=0.0000001)
  cout << "This is the  " << ++iter << " iteration\n";
```

```
int iter=0;
for (float i=0.0; i<0.000001; i+=0.0000001)
  cout << "This is the  " << ++iter << " iteration\n";
```

for: Common Errors (cont'd)

- Unreachable loop termination condition => unintended infinite loop
- Example III

```
int iter=0;
for (float i=0.0; i!=0.000001; i+=0.0000001)
  cout << "This is the  " << ++iter << " iteration\n";
```

```
int iter=0;
for (float i=0.0; i<0.000001; i+=0.0000001)
  cout << "This is the  " << ++iter << " iteration\n";
```

- To control a loop, use a relational expression if possible, rather than an equality expression
- Don't use a variable of any floating point data type to control a loop because real numbers are represented in their approximate values internally

break Statement

- The `break` statement causes an exit from the innermost enclosing loop or `switch` statement

```
while (1) {
   cin >> n;
   if (n < 0)
      break;
   cout << n << endl;
}
// if break is run, jumps to here
```

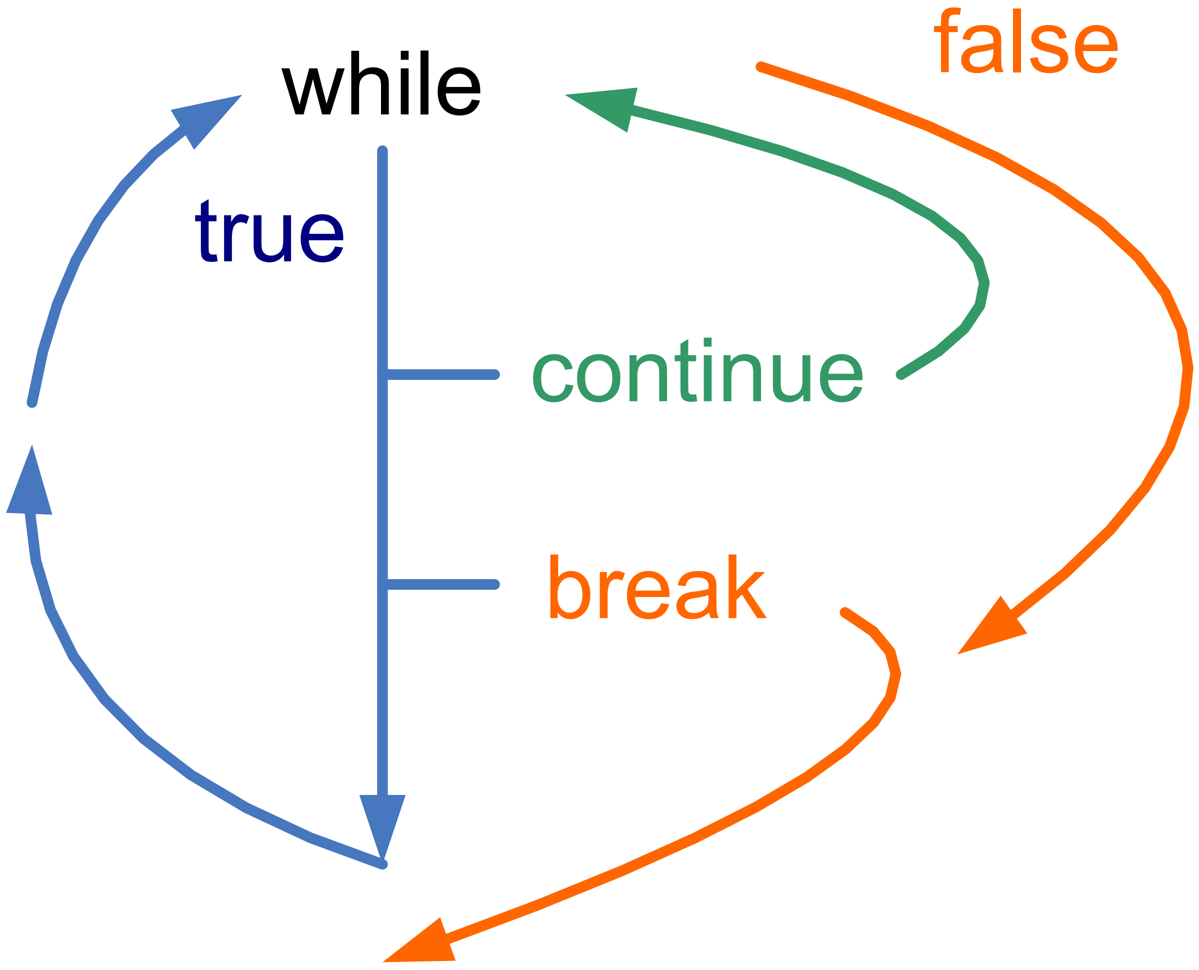
`continue` Statement

- `continue` statement causes the current iteration of a loop to stop and the next iteration to begin immediately

- It can be applied in a `while`, `do-while` or `for` statement

```
cnt=0;
while (cnt<10) {
   cin >> x;
   if (x > -0.01 && x < 0.01)
      continue; // discard small values
   ++cnt;
   sum += x;
}
```

continue, break



goto Statement

- goto statement transfers control to another statement specified by a label
- goto statement is considered a harmful construct and a bad programming practice
  - It makes the logic of the program complex and tangled
  - It can be replaced with the use of break and continue

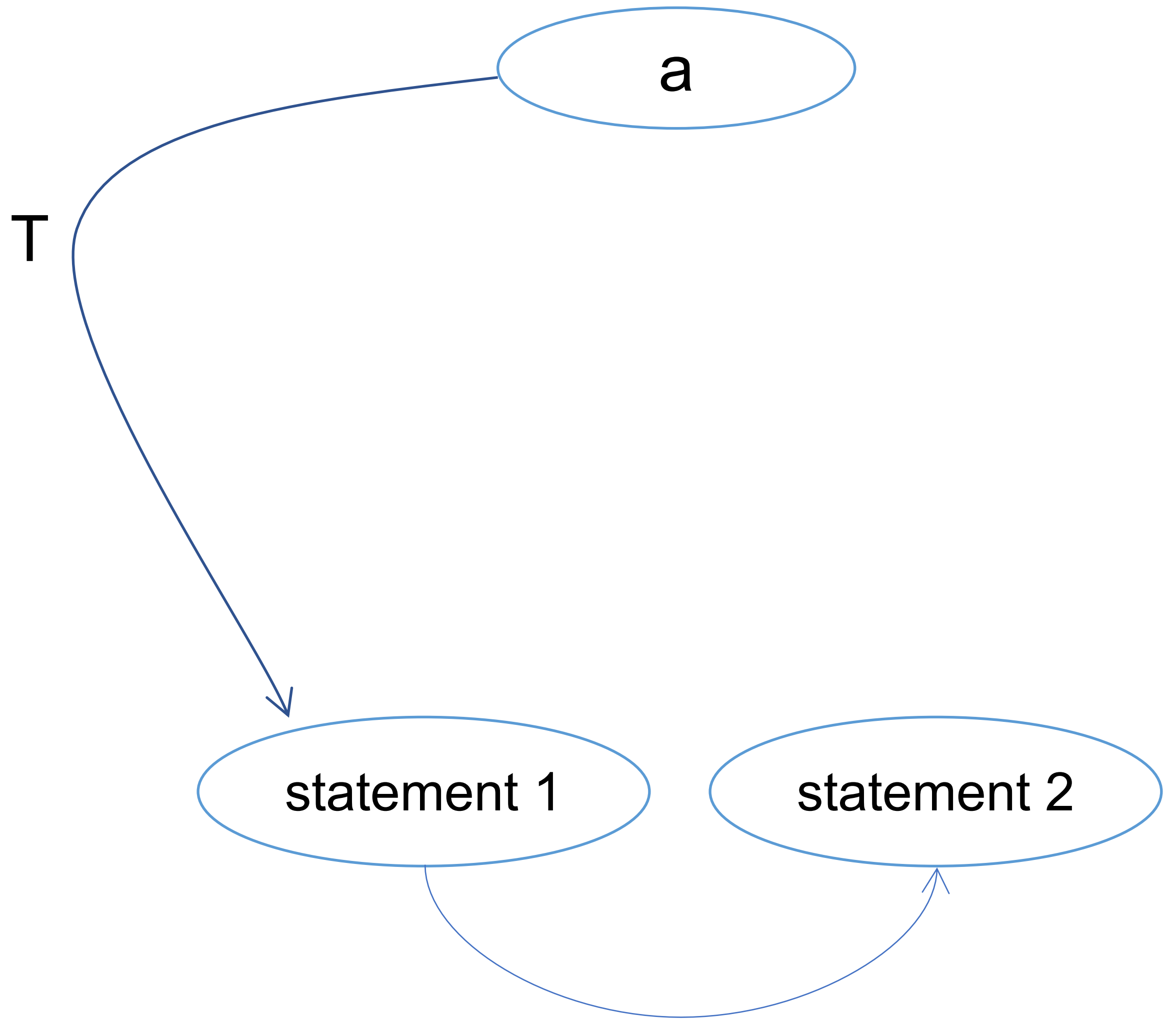
```
int main() {
   int num;
   cin >> num;
   if (num%2 == 0)
      goto even;
   else
      goto odd;
even:
   cout << num << " is even\n";
   return 0;
odd:
   cout << num << " is odd\n";
   return 0;
}
```

Short-Circuit Evaluation with goto

```
if (a || b || c)
   statement1;
statement2;
```

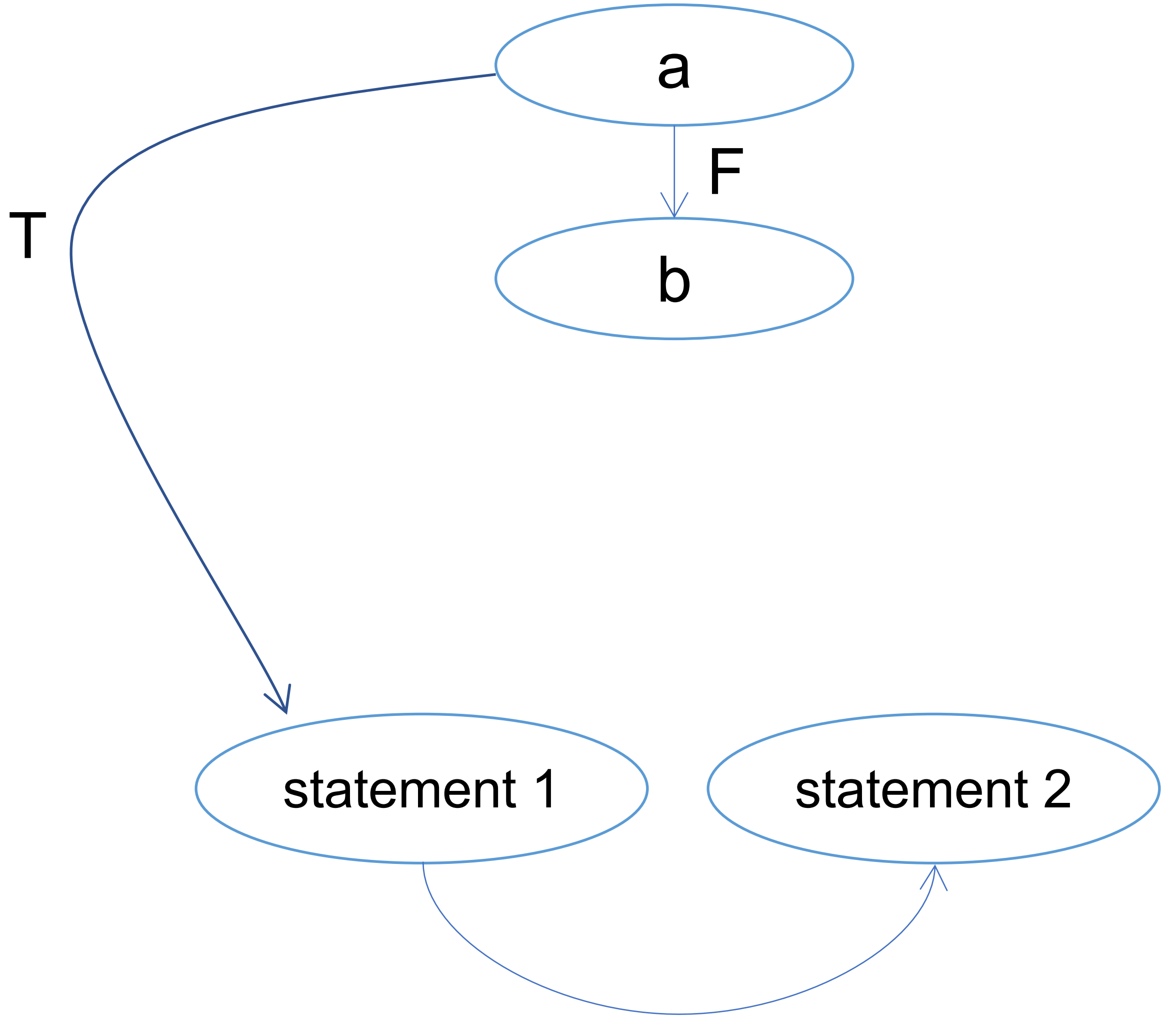
Short-Circuit Evaluation with goto

```
if (a || b || c)
   statement1;
statement2;
```



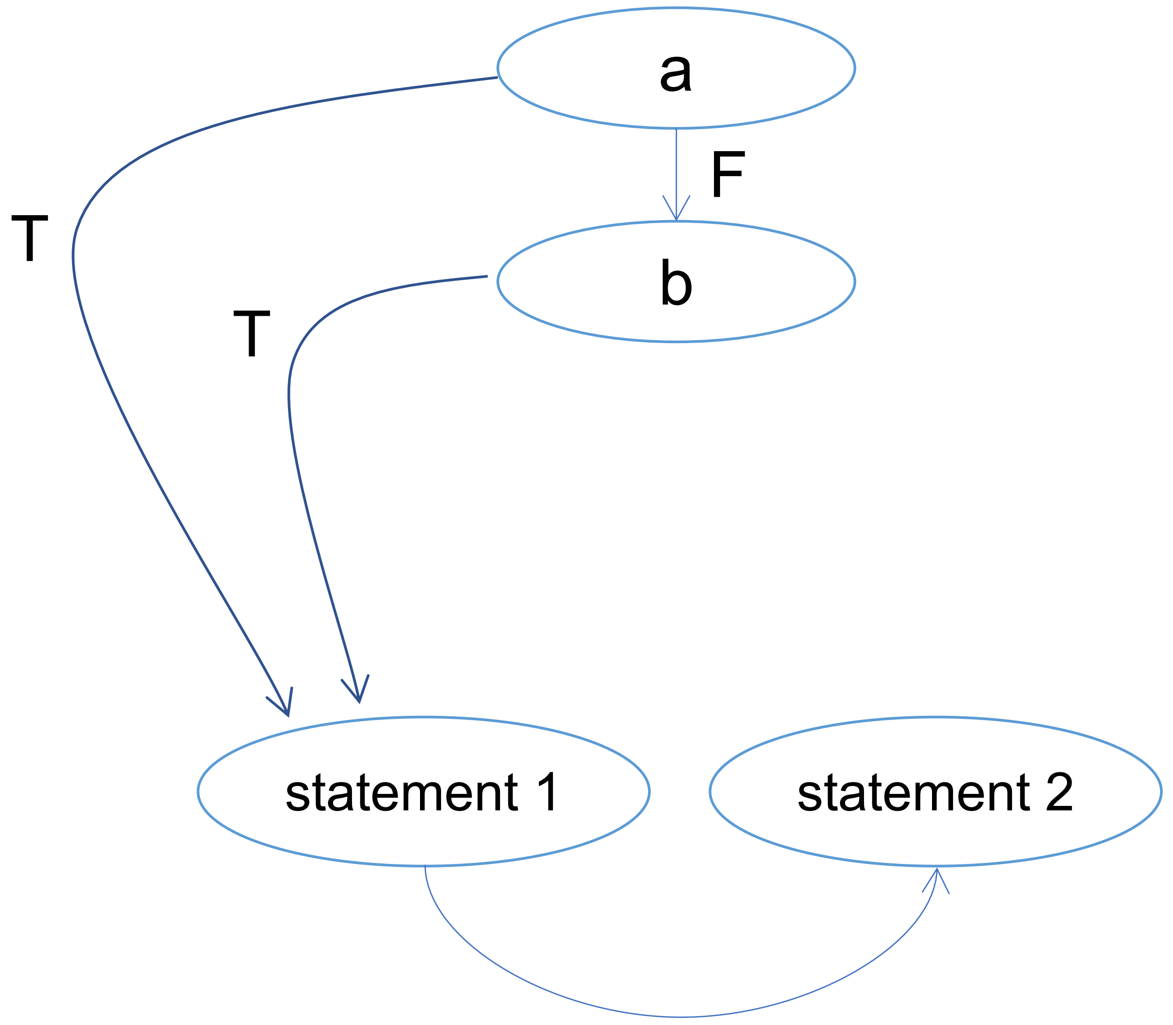
Short-Circuit Evaluation with goto

```
if (a || b || c)
   statement1;
statement2;
```



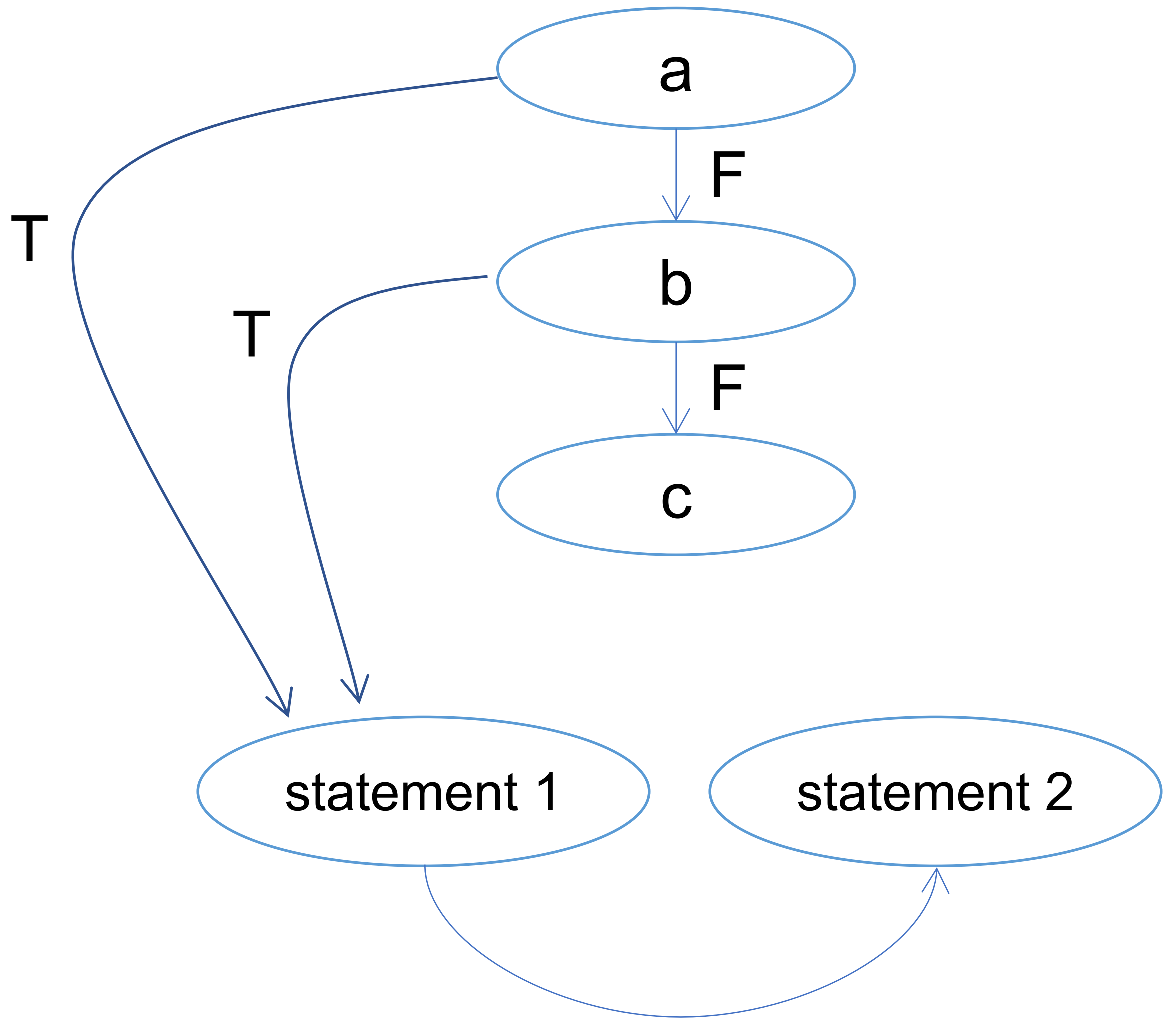
Short-Circuit Evaluation with goto

```
if (a || b || c)
   statement1;
statement2;
```



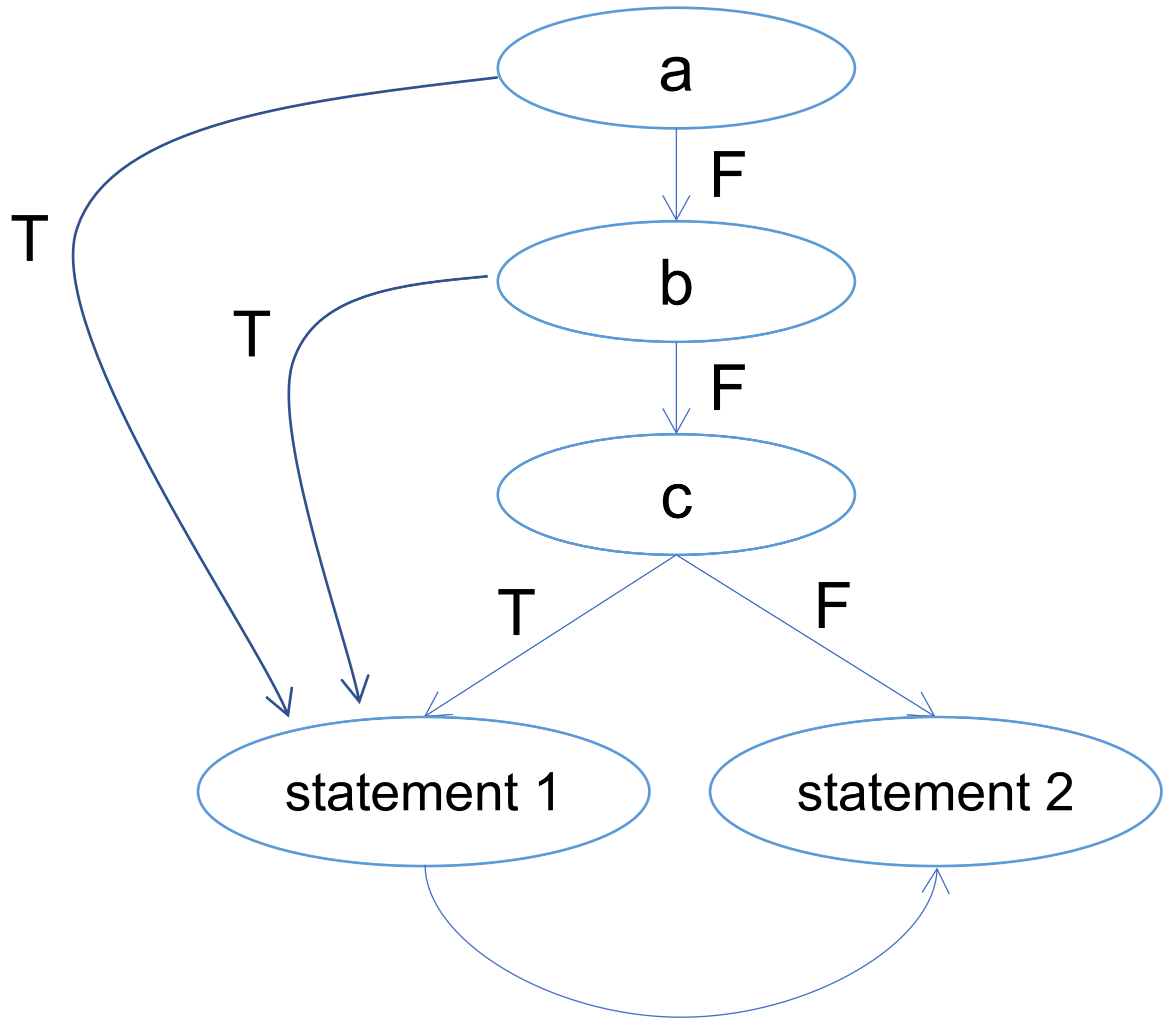
Short-Circuit Evaluation with goto

```
if (a || b || c)
   statement1;
statement2;
```



Short-Circuit Evaluation with goto

```
if (a || b || c)
   statement1;
statement2;
```

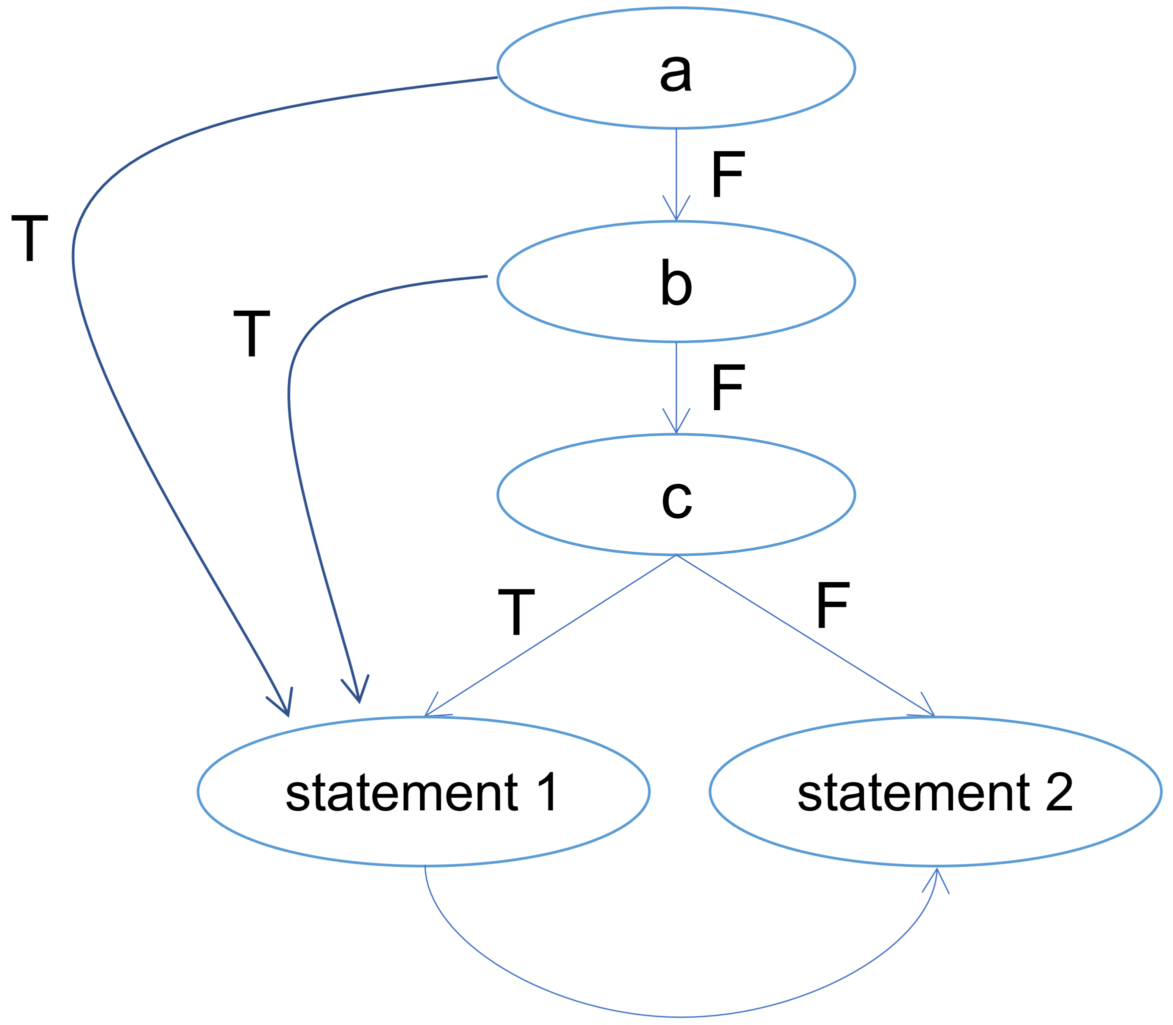


Short-Circuit Evaluation with goto

```
if (a || b || c)
   statement1;
statement2;
```

is equivalent to …

```
if (a) goto S1;
if (b) goto S1;
if (c) goto S1;
goto S2; // a||b||c == false
S1:
   statement1;
S2:
   statement2;
```



Short-Circuit Evaluation with goto

```
if (a && b && c)
   statement1;
statement2;
```

is equivalent to ? (using *if (x) goto label* construct)

```
if (!a) goto S2;
if (!b) goto S2;
if (!c) goto S2;
statement1; // a&&b&&c == true
S2:
   statement2;
```

Today's Outline

- Loop
  - while
  - do-while
  - for
- Programming styles for control flow

Indentation

```
int main() {
    int i;
    for (i=0; i<100; i++) {
        if (i>3)
            cout << i;
    }
    return 0;
}
```

⟷ 1st level (1 tab)

⟷ 2nd level (2 tabs)

⟷ 3rd level (3 tabs)

- Indent code in a consistent fashion to indicate the flow of control (use the tab key)
- Note the multiple levels of indentation

Formatting Programs

- Indent the code properly as you write the program to reflect the **structure** of the program.
  - Improve readability and increase the ease for debugging the program
  - In assignment, marks will be allocated for indentation.
- To indent in visual studio, you may press the **tab** button
- You may select multiple lines of statements and press tab to indent all of them
- To move back one level to the left, press **shift+tab**

if Condition Style

```
if(condition) {                   // Bad-space missing after if
if ( condition ) {                // Bad-space between the parentheses
                                     and the condition
if (condition){                   // Bad–space missing before {
if(condition){                    // Doubly bad
if (int a = f();a == 10) {        // Bad – space missing after the
                                    semicolon
if (conditionA && conditionB) {   // GOOD
```

Exercises: Basic Syntax

- What are the errors in the following codes, which aim to output the sum and product of the two input integers in two separate lines?

```
#include <iostream>
using namespace std;

int main() {
    int n, m;
    cout << 'Enter inputs: ' << endl;
    cin >> n >> m >> endl;
    cout << n+1;
    cout << m+1;
    return 0;
}
```

Expected Output Example

```
Enter inputs: 5 6
11
30
```

Exercises: Data Types 1

- What will be printed and why?

```
#include <iostream>
using namespace std;
int main() {
      char vChar1 = 'A’;
      int vChar2 = '0';
      cout << vChar1 << " " << vChar2 << endl;
      cout << ++vChar1 << endl;
      return 0;
}
```

Exercises: Data Types 2

- For integral operands, division operator yields algebraic quotient with any fractional part discarded (i.e., round towards zero)
  - If quotient `a/b` is representable in type of result, `(a/b)*b+a%b` is equal to `a`
  - So, assuming b is not zero and no overflow, `a%b` equals `a-(a/b)*b`
- What's the value of k at each step?

```
int m = 3, n = 2;
double k;
k = m / n;
k = m / double(n);
k = double(m) / n;
k = double(m/n);
k = m / 2;
k = m / 2.0;
```

Exercises: Loop 1

- write a program to generate a matrix of *n* rows and *m* column (*n* and *m* is input by the user), where the element at the *i*-th row and *j*-th colum is the multiplication of *i* and *j*. Assume *n* > 1 and *m* <= 9
- E.g., when *n* = 4, *m* = 3, the following matrix is generated

```
1    2    3
2    4    6
3    6    9
4    8    12
```

```
int main() {
   int n, m; // n: rows, m: columns
   cin >> n >> m;

   // Your codes ...

   return 0;
}
```

Exercises: Loop 1

```
int main() {
    int n, m; // n: rows, m: columns
    cin >> n >> m;
    for (int i=1; i<=n; i++) {
        for (int j=1; j<=m; j++) {
            // for the element at the i-th row and j-th column
            cout << i*j << "\t";
        }
        cout << endl;
    }
    return 0;
}
```

Exercises: Loop 2

- write a program to produce a *nxn* matrix (*n* is input by user) with 0's down the main diagonal, 1's in the entries just above and below the main diagonal, 2's above and below that, etc.
- *hint*: consider using nested for-loop, with the outer loop responsible for row and the inner loop for each column

Example 1

```
Enter the number of rows: 5
0 1 2 3 4
1 0 1 2 3
2 1 0 1 2
3 2 1 0 1
4 3 2 1 0
```

Example 2

```
Enter the number of rows: 8
0 1 2 3 4 5 6 7
1 0 1 2 3 4 5 6
2 1 0 1 2 3 4 5
3 2 1 0 1 2 3 4
4 3 2 1 0 1 2 3
5 4 3 2 1 0 1 2
6 5 4 3 2 1 0 1
7 6 5 4 3 2 1 0
```

Example 3

```
Enter the number of rows: 0
Please enter positive
integer.
```

Example 4

```
Enter the number of rows: 3
0 1 2
1 0 1
2 1 0
```

```
int n;
cout << "Enter the number of rows: ";
cin >> n;
if (n <= 0) {
          cout << "Please enter positive integer.\n";
} else {
          // Your codes ...
}
```