CS2310 Computer Programming

LT04: Control Flow - Loop

Computer Science, City University of Hong Kong

Semester A 2023-24

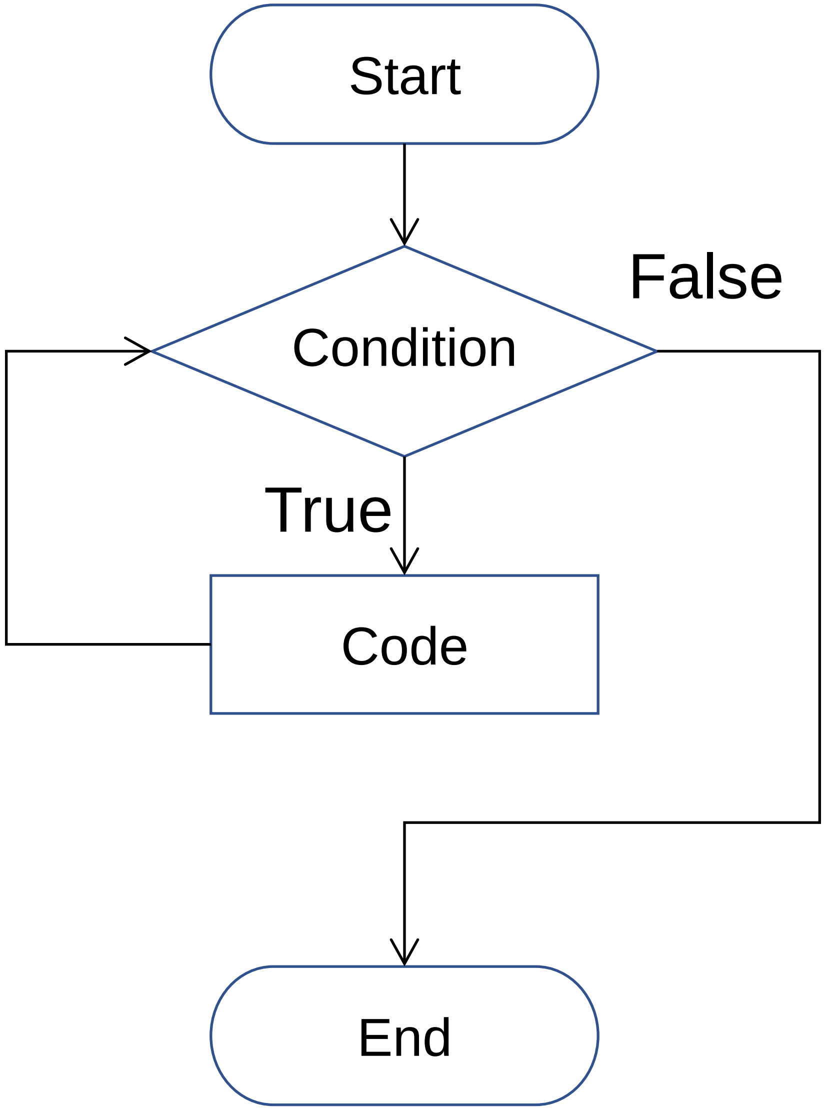
Today's Outline

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

Loop

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

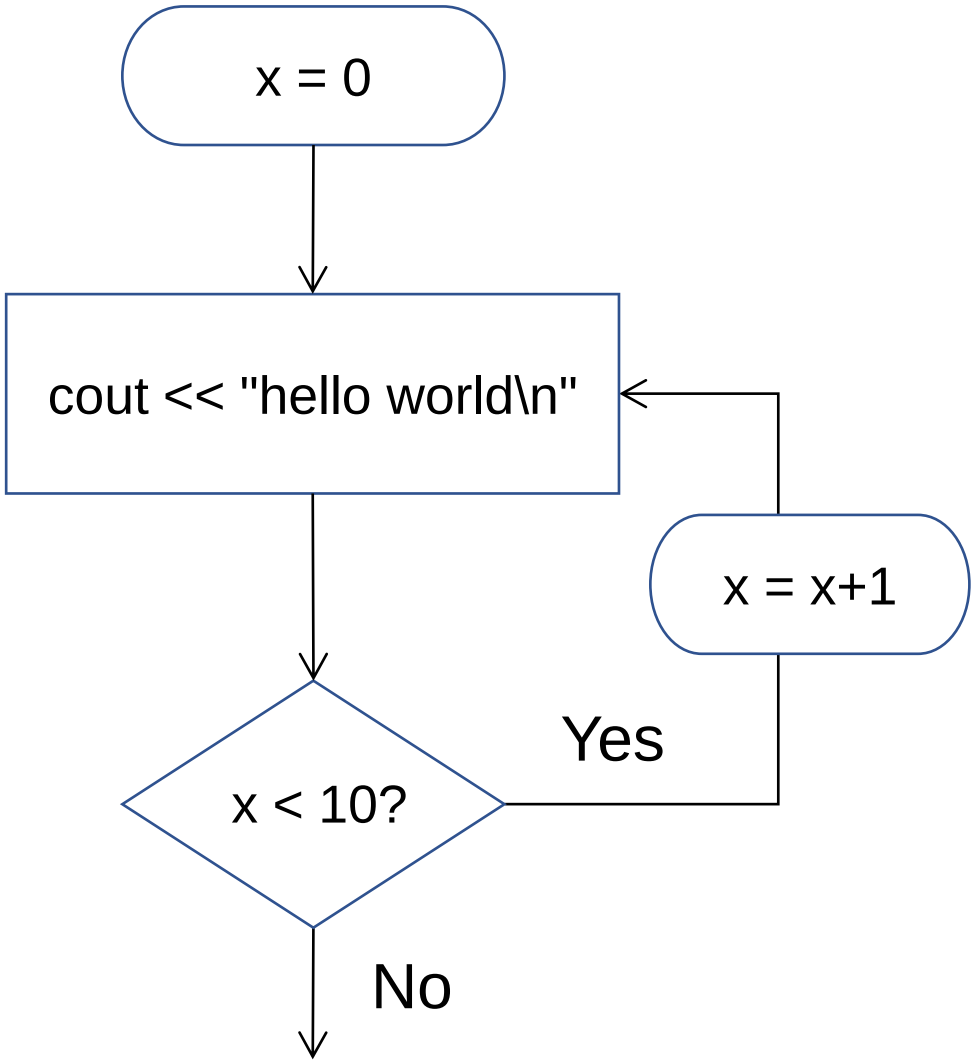
- Beside sequential and branch execution loop is another common control flow



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)

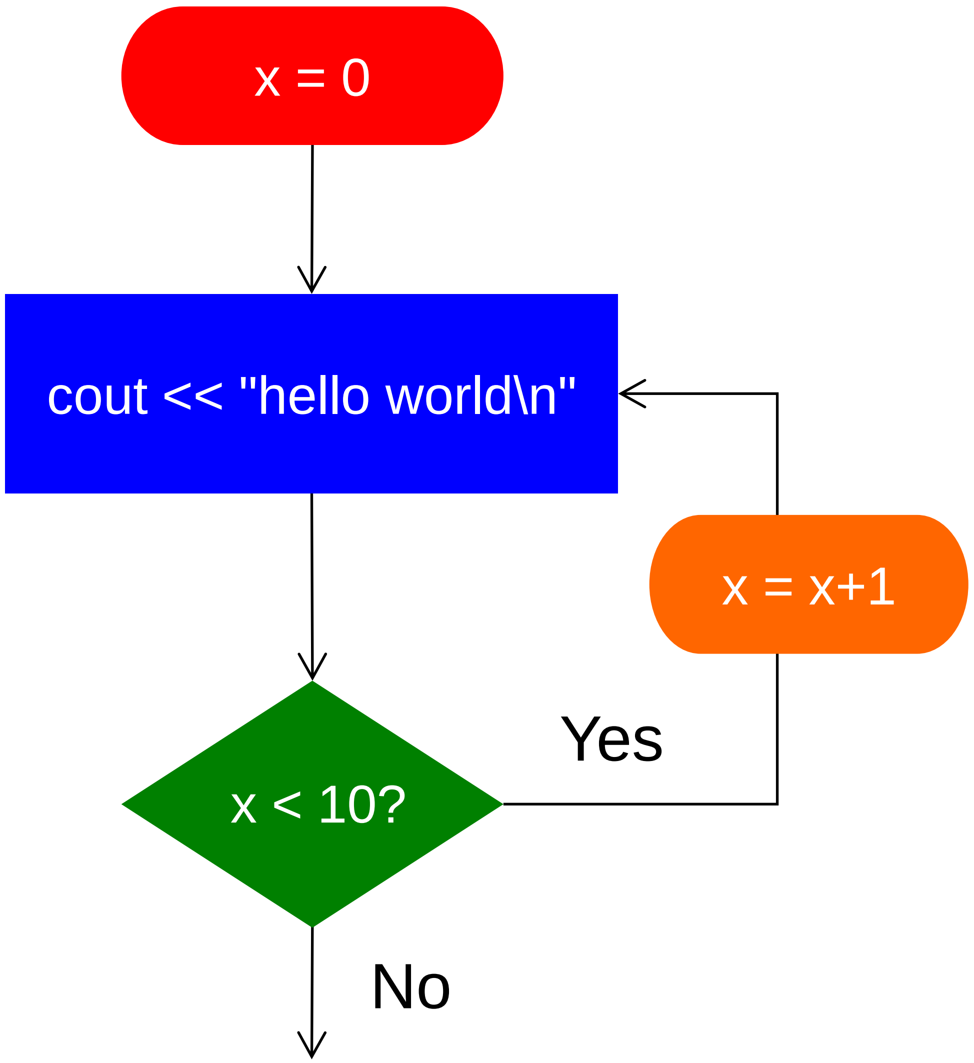
- In general, a loop consists of four parts

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

```cpp
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;
}
```

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);
```

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;
}
```

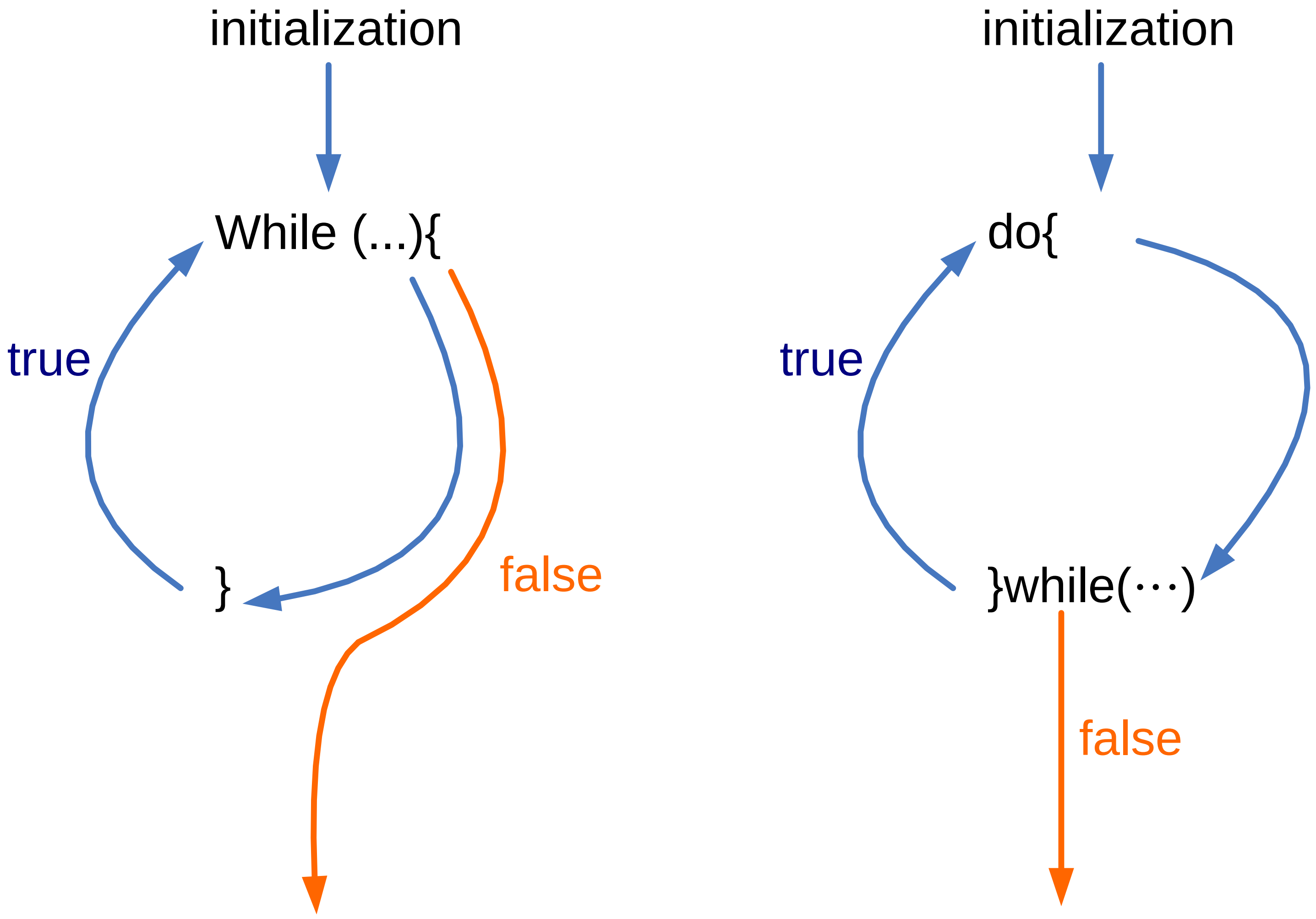
```
int x, max;
max = 0;



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



initialization

While (...){

true

}

false

initialization

do{

true

}while(⋯)

false

for: Syntax

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

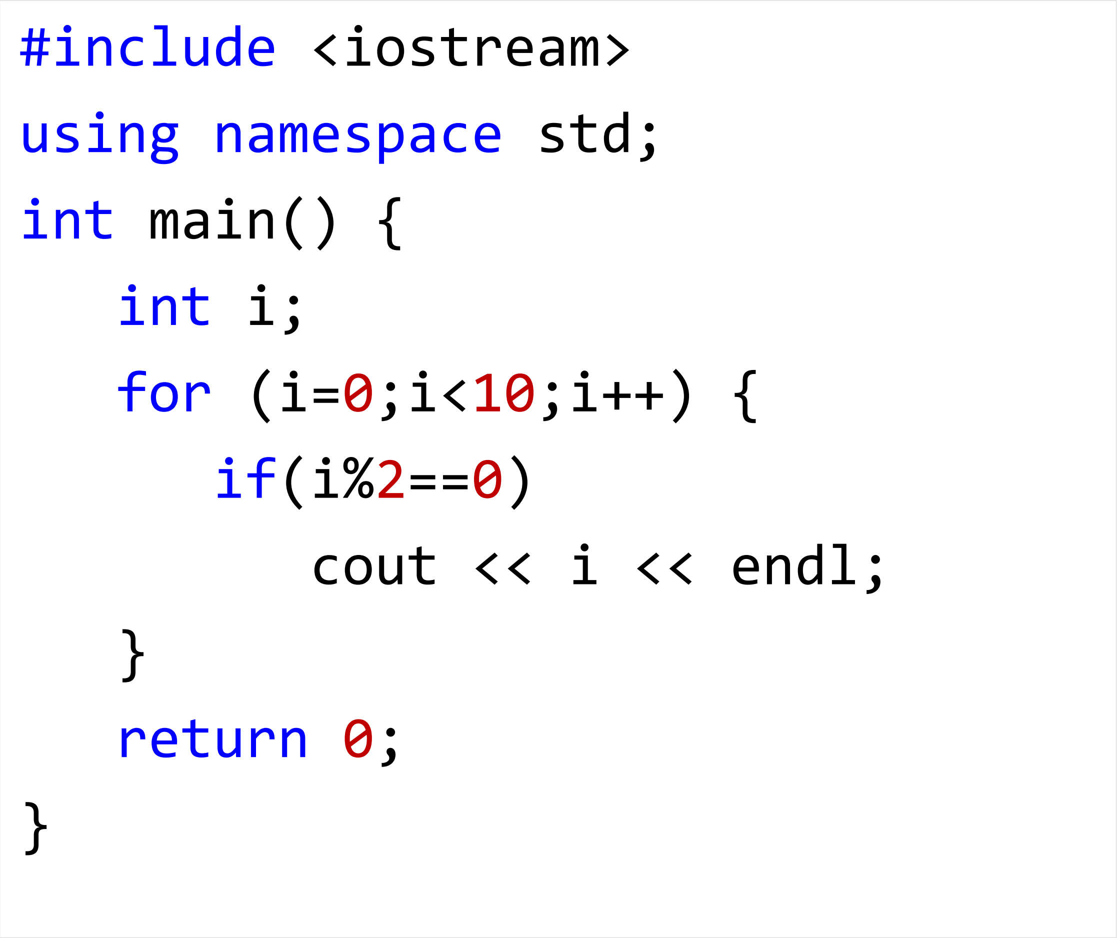
- Semantics

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



```cpp
#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

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

```cpp
#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;
}
```

```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;                  ) {


    }
    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

```
#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;         ) {
        cout << "Input must be positive." << endl;
        cout << "Enter number: ";
    }
    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++)
    …
```

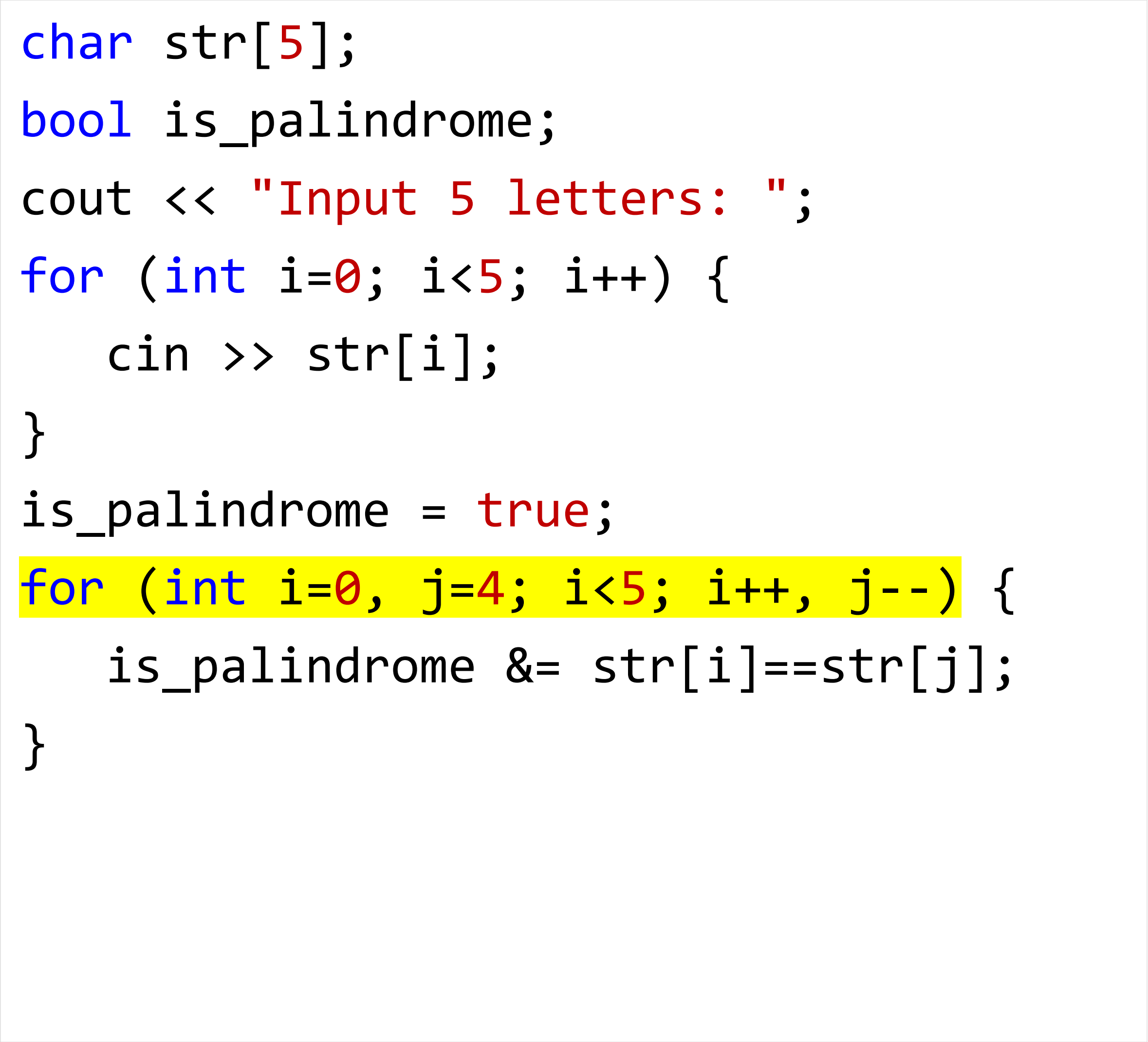
for: Examples (cont'd)

- *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];
}
```

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
```

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.

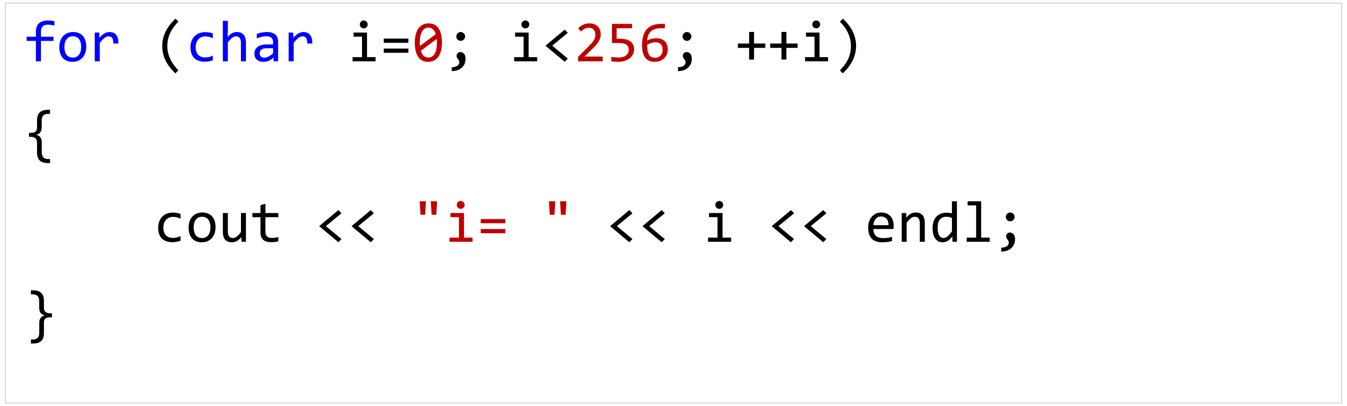
```
int sum = 0;
for (j=1; j<=10; j++)
    sum += j;
```

Is NOT the same as

```
int sum = 0;
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
```

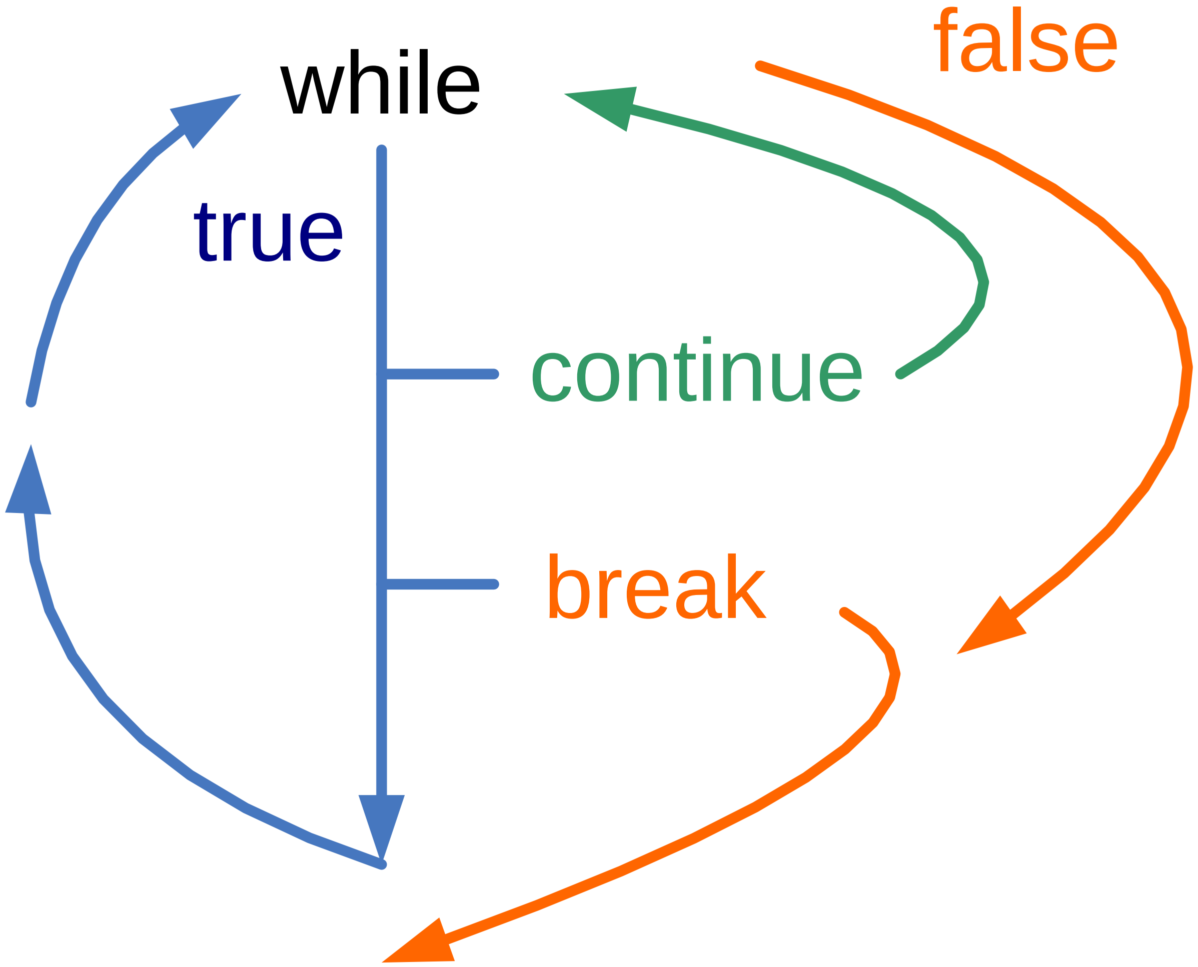
continue Statement

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;
}
```

Today's Outline

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

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
```

Today's Outline

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

Exercises: Data Types 1

- What will be printed and why?

```
#include <iostream>
using namespace std;
int main() {
      char vChar1 = 'A';
      char 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: I/O

- write a program ConvertTemperature.cpp

  a) read a temperature in Celsius (data type: int) and display it in Fahrenheit (data type: double). Round the result to 2 decimal places.

  - Hint: *Fahrenheit = 9/5*Celsius + 32*

  b) convert calculated Fahrenheit (data type: double) into Kelvin (data type: double). Round the result to 2 decimal places.

  - Hint: *Kelvin* = (*Fahrenheit*+459.67) * 5/9

Expected Output:

```
Enter Temperature in
Centigrade:
30
Temperature in Fahrenheit
is:
86.00
Temperature in Kelvin is:
303.15
```

Exercises: I/O

```
int Celsius;
double Fahrenheit, Kelvin;
cout << "Enter Temperature in Centigrade:\n";
cin >> Celsius;
cout <<"Temperature in Fahrenheit is:\n";
Fahrenheit = 9.0 / 5 * Celsius + 32;
cout << fixed << setprecision(2);
cout << Fahrenheit << "\n";
cout << "Temperature in Kelvin is:\n";
Kelvin = (Fahrenheit + 459.67) * 5 / 9.0;
cout << Kelvin << "\n";
```

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
```

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

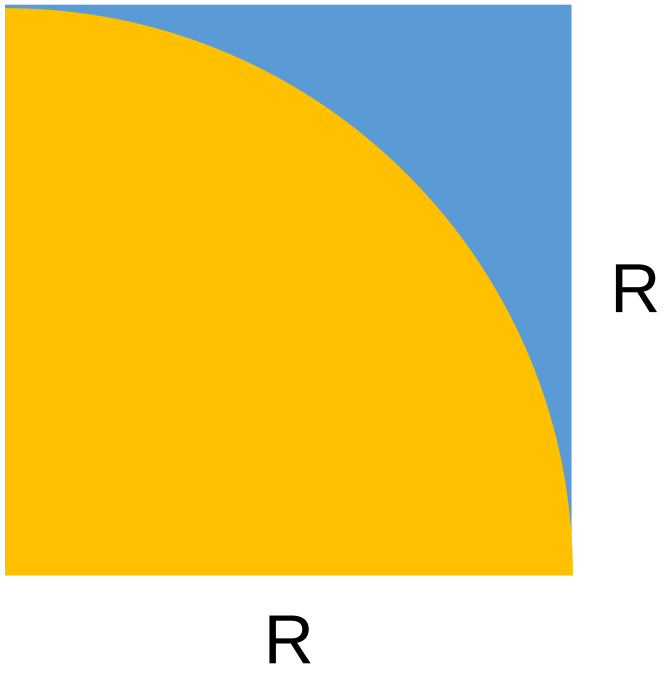
Example 1	Example 2
Enter the number of rows: 5 0 1 2 3 4 1 0 1 2 3 2 1 0 1 2 3 2 1 0 1 4 3 2 1 0	Enter the number of rows: 8 0 1 2 3 4 5 6 7 1 0 1 2 3 4 5 6 2 1 0 1 2 3 4 5 3 2 1 0 1 2 3 4 4 3 2 1 0 1 2 3 5 4 3 2 1 0 1 2 6 5 4 3 2 1 0 1 7 6 5 4 3 2 1 0
Example 3	**Example 4**
Enter the number of rows: 0 Please enter positive integer.	Enter the number of rows: 3 0 1 2 1 0 1 2 1 0

Exercises: Loop 2

```
int n;
cout << "Enter the number of rows: ";
cin >> n;
if (n <= 0) {
      cout << "Please enter positive integer.\n";
} else {
      for (int row=0; row<n; row++) {
            for (int col=0; col<n; col++)
                  cout << abs(col-row) << " ";
            cout << endl;
      }
}
```

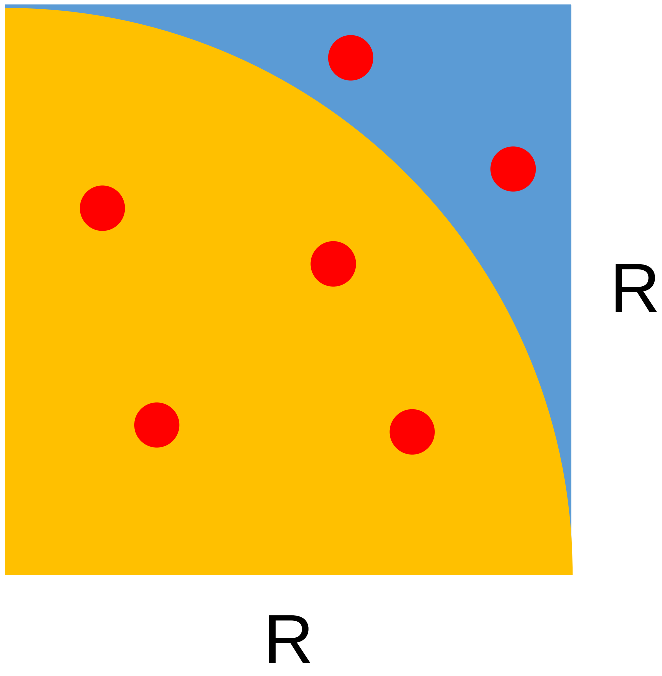
Exercises: Loop 3

- Monte Carlo estimation of Pi
  - circle_area = Pi*R*R/4
  - square_area = R*R
  - Pi = 4*circle_area/square_area
  - How to estimate circle_area/square_area?



Exercises: Loop 3

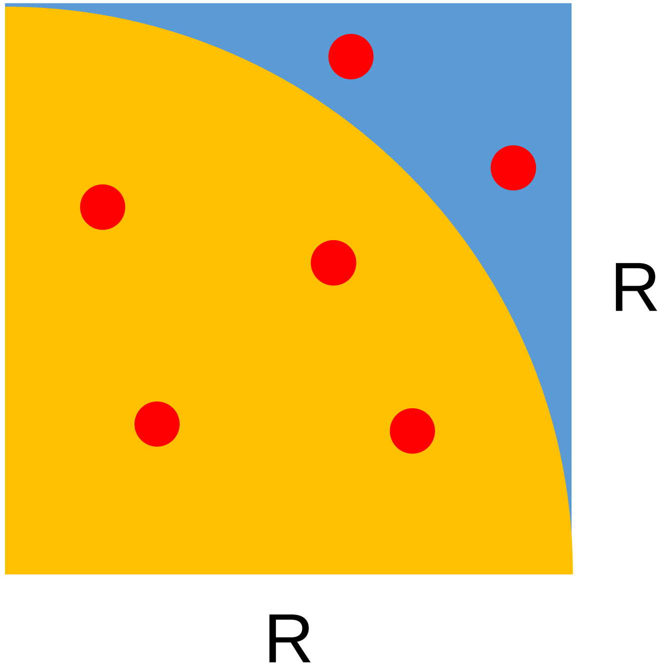
- Monte Carlo estimation of Pi
  - circle_area = Pi*R*R/4
  - square_area = R*R
  - Pi = 4*circle_area/square_area
  - How to estimate circle_area/square_area?
    - Randomly throw *N* points to the square
    - Let *M* be the number of points falling to the yellow area
    - circle_area/square_area $\approx \frac{M}{N}$



Exercises: Loop 3

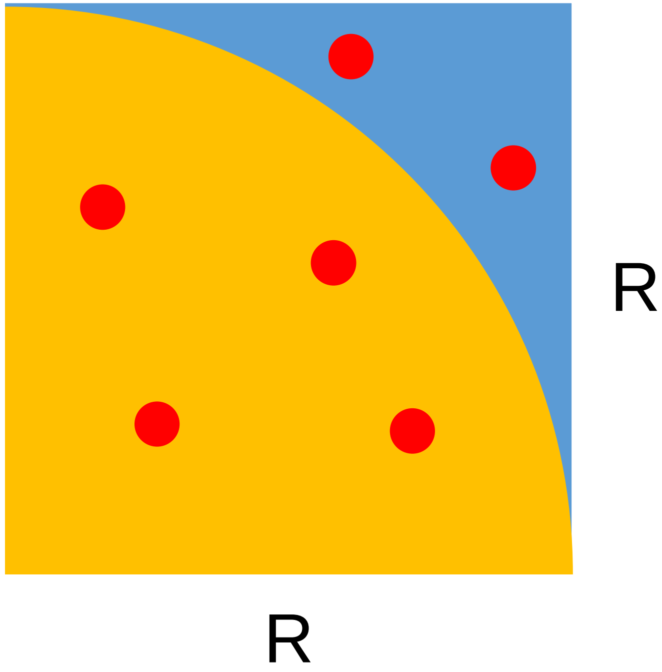
```
// Assume R=1
int M=0;

   // Randomly throw a point
   double x = (double)rand()/RAND_MAX;
   double y = (double)rand()/RAND_MAX;
   if (x*x+y*y < 1.0)
      M++; // Increment M if (x, y) is within the yellow area
```



Exercises: Loop 3

```
// Assume R=1
int M=0, N=10000;
for (int n=0; n<N; n++) {
   // Randomly throw a point
   double x = (double)rand()/RAND_MAX;
   double y = (double)rand()/RAND_MAX;
   if (x*x+y*y < 1.0)
      M++; // Increment M if (x, y) is within the yellow area
   double est = 4.0*M/n;
   cout << n << " " << est << endl;
}
```



Exercises: Loop 3

- Each round we throw 10000 pts
- Then observe if the estimated value of Pi has changed significantly
- If true, the estimation is not stable => continue throwing pts
- If false, the estimation is now stable, stop

Exercises: Loop 3

- Each round we throw 10000 pts
- Then observe if the estimated value of Pi has changed significantly
- If true, the estimation is not stable => continue throwing pts
- If false, the estimation is now stable, stop

```
int M=0, N=0;


   for (int n=0; n<10000; n++) {
      double x = (double)rand()/RAND_MAX;
      double y = (double)rand()/RAND_MAX;
      if (x*x+y*y < 1.0)
         M++;
      N++;
   } // end of for
```

Exercises: Loop 3

- Each round we throw 10000 pts
- Then observe if the estimated value of Pi has changed significantly
- If true, the estimation is not stable => continue throwing pts
- If false, the estimation is now stable, stop

```
double prev_est=0, curr_est=0;
int M=0, N=0;


   for (int n=0; n<10000; n++) {
      double x = (double)rand()/RAND_MAX;
      double y = (double)rand()/RAND_MAX;
      if (x*x+y*y < 1.0)
         M++;
      N++;
   } // end of for
```

Exercises: Loop 3

- Each round we throw 10000 pts
- Then observe if the estimated value of Pi has changed significantly
- If true, the estimation is not stable => continue throwing pts
- If false, the estimation is now stable, stop

```
double prev_est=0, curr_est=0;
int M=0, N=0;
while (true) {
   prev_est = curr_est;
   for (int n=0; n<10000; n++) {
      double x = (double)rand()/RAND_MAX;
      double y = (double)rand()/RAND_MAX;
      if (x*x+y*y < 1.0)
         M++;
      N++;
   } // end of for
   curr_est = 4.0*M/N;
   cout << curr_est << endl;
} // end of while
```

Exercises: Loop 3

- Each round we throw 10000 pts
- Then observe if the estimated value of Pi has changed significantly
- If true, the estimation is not stable => continue throwing pts
- If false, the estimation is now stable, stop

```
double precision = 1e-10;
double prev_est=0, curr_est=0;
int M=0, N=0;
while (true) {
   prev_est = curr_est;
   for (int n=0; n<10000; n++) {
      double x = (double)rand()/RAND_MAX;
      double y = (double)rand()/RAND_MAX;
      if (x*x+y*y < 1.0)
         M++;
      N++;
   } // end of for
   curr_est = 4.0*M/N;
   cout << curr_est << endl;
   if (abs(curr_est-prev_est) < precision)
      break;
} // end of while
```

Exercises: Loop 3

- Each round we throw 10000 pts
- Then observe if the estimated value of Pi has changed significantly
- If true, the estimation is not stable => continue throwing pts
- If false, the estimation is now stable, stop

```
double precision = 1e-10;
double prev_est=0, curr_est=0;
int M=0, N=0;
~~while (true)~~ do {
    prev_est = curr_est;
    for (int n=0; n<10000; n++) {
        double x = (double)rand()/RAND_MAX;
        double y = (double)rand()/RAND_MAX;
        if (x*x+y*y < 1.0)
            M++;
        N++;
    } // end of for
    curr_est = 4.0*M/N;
    cout << curr_est << endl;
    ~~if (abs(curr_est-prev_est) < precision)~~
        ~~break;~~
} while (abs(curr_est-prev_est)>precision);
```

Exercises: Conditional + Loop

- Guess an integer number in user's mind. Assume the number is positive and not greater than 100 (i.e., 0 < num <= 100)
- In each round, the program asks the user a question and the user answers with 'T' (true) or 'F' (false), until the program guesses the correct number
- Try to guess the number with as few rounds as possible
- Example:

  Is it 99? N

  Is it 98? N

  …

  Is it 16? Y