Loops
for loops
while & do while loops

Dr. Anwar Shah, PhD, MBA(HR)
Assistant Professor in Computer Science
FAST National University of Computer and Emerging Sciences CFD

Looping

Some typical use-cases

Execute a loop:

- a specific number of times
- for each element in a collection
- while a specific condition remains true
- until a specific condition becomes false
- until we reach the end of some input stream
- forever
- many, many more

C++ Looping Constructs

- `for` loop
    - iterate a specific number of times
- `Range-based for` loop
    - one iteration for each element in a range or collection
- `while` loop
    - iterate while a condition remains true
    - stop when the condition becomes false
    - check the condition at the beginning of every iteration
- `do-while` loop
    - iterate while a condition remains true
    - stop when the condition becomes false
    - check the condition at the end of every iteration

`for` Loop

```
for (initialization ; condition ; increment)
   statement;



for (initialization ; condition ; increment)  {
   statement(s);
}
```

`for` Loop

```
int i {0};

for (i = 1 ; i <= 5 ; ++i)
   cout << i << endl;

1
2
3
4
5
```

for Loop

```
for (int i {1} ; i <= 5 ; ++i)
   cout << i << endl;


for (int i = 1 ; i <= 5 ; ++i)
   cout << i << endl;

i = 100; // ERROR i only visible in the loop
```

for Loop

display even numbers

```
for (int i {1} ; i <= 10 ; ++i) {
    if (i % 2 == 0)
        cout << i << endl;


2
4
6
8
10
```

for Loop

array example

```
int scores [] {100,90,87};

for (int i {0} ; i < 3 ; ++i) {
        cout << scores[i] << endl;
}

for (int i {0} ; i <= 2 ; ++i) {
        cout << scores[i] << endl;
}
100
90
87
```

`for` Loop

comma operator

```
for (int i {1}, j {5} ; i <= 5 ; ++i, ++j) {
   cout << i << " * " << j << " : " << (i * j) << endl;
}

1 * 5 : 5
2 * 6 : 12
3 * 7 : 21
4 * 8 : 32
5 * 9 : 45
```

`for` Loop

some other details…

- The basic for loop is very clear and concise
- Since the for loop’s expressions are all optional, it is possible to have
    - no initialization
    - no test
    - no increment

```
for (;;)
   cout << "Endless loop" << endl;
```

Range-based `for` Loop

Introduced in C++11

```
for (var_type var_name: sequence)
   statement; // can use var_name
```

```
for (var_type var_name: sequence) {
   statements; // can use var_name
}
```

Range-based `for` Loop

```
int scores [] {100, 90, 97};

for (int score : scores)
   cout << score << endl;

100
90
97
```

Range-based `for` Loop

auto

```
int scores [] {100, 90, 97};

for (auto score : scores)
   cout << score << endl;

100
90
97
```

Range-based `for` Loop

vector

```
vector<double> temps {87.2, 77.1, 80.0, 72.5};

double average_temp {};
double running_sum {};

for (auto temp: temps)
   running_sum += temp;

average_temp = running_sum / temps.size();
```

Range-based `for` Loop

initializer list

```
double average_temp {};
double running_sum {};
int size {0};

for (auto temp: {60.2, 80.1, 90.0, 78.2} ) {
    running_sum += temp;
   ++size;
}
average_temp = running_sum / size;
```

Range-based `for` Loop

string

```
 for (auto c: "Frank")
     cout << c << endl;

F
r
a
n
k
```

`while` Loop

```
while (expression)
   statement;


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

`while` Loop

```
int i {1};

while (i <= 5) {
   cout << i << endl;
   ++i;  // important!
}

1
2
3
4
5
```

`while` Loop

even numbers

```
int i {1};

while (i <= 10) {
   if (i % 2 == 0)
      cout << i << endl;
   ++i;
}

2
4
6
8
10
```

`while` Loop

array example

```
int scores [] {100,90,87};
int i {0};

while (i < 3){
   cout << scores[i] << endl;
   ++i;
}

100
90
87
```

`while` Loop

input validation

```
int number {};

cout << "Enter an integer less than 100: ";
cin >> number;

while (number >= 100){  // !(number < 100)
   cout << "Enter an integer less than 100";
   cin >> number;
}

cout << "Thanks" << endl;
```

`while` Loop

input validation

```
int number {};

cout << "Enter an integer between 1 and 5: ";
cin >> number;

while (number <= 1 || number >= 5){
   cout << "Enter an integer between 1 and 5: ";
   cin >> number;
}

cout << "Thanks" << endl;
```

`while` Loop

input validation – boolean flag

```
bool done {false};
int number {0};

while (!done) {
    cout << "Enter an integer between 1 and 5: ";
    cin >> number;
    if (number <=1 || number >=5)
        cout << "Out of range, try again" << endl;
    else {
        cout << "Thanks!" << endl;
        done = true;
    }
}
```

`do-while` Loop

```
do {
   statements;
} while (expression);
```

`do-while` Loop

input validation

```
int number {};
do {

   cout << "Enter an integer between 1 and 5: ";
   cin >> number;

} while (number <= 1 || number >= 5);

cout << "Thanks" << endl;
```

`do-while` Loop

area calculation with calculate another

```
char selection {};

    do {
        double width {}, height {};
        cout << "Enter width and height separated by a space :";
        cin >> width >> height;

        double area {width * height };
        cout << "The area is " << area << endl;

        cout << "Calculate another? (Y/N) : ";
        cin >> selection;
    } while (selection == 'Y' || selection == 'y');
    cout << "Thanks!" << endl;
```

`continue` and `break` statements

- `continue`
  - no further statements in the body of the loop are executed
  - control immediately goes directly to the beginning of the loop for the next iteration

- `break`
  - no further statements in the body of the loop are executed
  - loop is immediately terminated
  - Control immediately goes to the statement following the loop construct

`continue` and `break` statements

```
vector<int> values {1,2,-1,3,-1,-99,7,8,10};
for (auto val: values) {
  if (val == -99)
     break;
  else if (val == -1)
     continue;
  else
     cout << val << endl;
}
```

```
1
2
3
```

Infinite Loops

- Loops whose condition expression always evaluate to true
- Usually this is unintended and a programmer error
- Sometimes programmers use infinite loops and include and break statements in the body to control them
- Sometimes infinite loops are exactly what we need
  - Event loop in an event-driven program
  - Operating system

Infinite `for` Loops

```
for (;;)
   cout << "This will print forever" << endl;
```

Infinite `while` Loops

```
while (true)
   cout << "This will print forever" << endl;
```

Infinite `do-while` Loops

```
do {
   cout << "This will print forever" << endl;
} while (true);
```

Infinite `while` Loops

example

```
while (true) {
   char again {};
   cout << "Do you want to loop again? (Y/N): ";
   cin >> again;

   if (again == 'N' || again == 'n')
       break;
}
```

Nested Loops

- Loop nested within another loop
- Can be many as many levels deep as the program needs
- Very useful with multi-dimensional data structures
- Outer loop vs. Inner loop

Nested Loops

```
for (outer_val {1}; outer_val <= 2 ; ++outer_val)
   for (inner_val {1}; inner_val <= 3; ++inner_val)
     cout << outer_val << ", " << inner_val << endl;

1, 1
1, 2
1, 3             outer_val, inner_val
2, 1             Note: inner loop loops "faster"
2, 2
2, 3
```

Nested Loops

Multiplication Table

```
for (int num1 {1}; num1 <=10 ; ++num1) {    // outer
   for (int num2 {1}; num2 <=10; ++num2) { // inner
      cout << num1 << " * " << num2
           << " = " << num1 * num2 << endl;
   }
   cout << "------------" << endl;
}

Displays 10 x 10 Multiplication Table
```

Nested Loops

2D Arrays – set all elements to 1000

```
int grid[5][3] {};

for (int row {0}; row < 5; ++row ) {
   for (int col {0}; col < 3; ++col ) {
     grid[row][col] = 1000;
   }
}
```

Nested Loops

2D Arrays – display elements

```
for (int row {0}; row < 5; ++row ) {
    for (int col {0}; col < 3; ++col ) {
        cout << grid[row][col] << " ";
    }
    cout << endl;
}
```

Nested Loops

2D Vector – display elements

```
vector<vector<int>> vector_2d
{
   {1, 2, 3},
   {10, 20, 30, 40},
   {100, 200, 300, 400, 500}
};

for (auto vec: vector_2d) {
   for (auto val: vec) {
      cout << val << " ";
   }
   cout << endl;
}
```

Output

```
1 2 3
10 20 30 40
100 200 300 400 500
```