Queens College, CUNY, Department of Computer Science

Object Oriented Programming in C++

CSCI 211 / 611

Summer 2018

Instructor: Dr. Sateesh Mane

July 20, 2018

Break and continue

- In this lecture we shall study the **break** and **continue** statements.
- The **break** and **continue** statements are used in loops.

1 Break and continue

- Suppose a program is executing a loop.
- It could be a `for` loop or a `while` loop, it does not matter.
- The **break** statement means exit the loop immediately.
  1. Do not execute the rest of the code in the loop, below the `break` statement.
  2. Do not perform the end of loop test, or increment the loop counter, etc.

```
for (int i = 0; i < n; ++i) {
  ...
  if (i > 3) break;    // exit the loop immediately if i > 3
                       // do not increment value of i, do not test end of loop condition
  ...                  // do not execute the code below the "break" statement
}

while (true) {
  ...
  if (condition == true) break; // if the condition is true, exit the loop immediately
  ...                           // do not execute the code below the "break" statement
}
```

- The **continue** statement means go to the next pass through the loop immediately.
  1. Do not execute the rest of the code in the loop, below the `continue` statement.
  2. **Increment the loop counter and perform the end of loop test.**
  3. **If the test passes, begin execution of the next pass through the loop.**

```
for (int i = 0; i < n; ++i) {
  ...
  if (i % 2 == 0) continue;  // if i is even, go to the next pass immediately
                             // increment value of i, test for end of loop
  ...                        // do not execute the code below the "continue" statement
}

while (true) {
  ...
  if (condition == true) continue; // if "condition" is true, begin next pass immediately
  ...                              // do not execute the code below the "continue" statement
}
```

2 Break: example

- Let us loop through a set of odd integers, say i.
  1. For each value of i, we test if i is prime by dividing by odd integers j such that $j \leq 3$ and $j * j \leq i$.
  2. In the loop over j, if the remainder $i \% j$ is zero, then i is not prime.
  3. Hence we break out of the loop over j.
  4. If i is prime we print a line of output.

```
#include <iostream>

using namespace std;

bool divisible(int i, int n)
{
  return ((i % n) == 0);
}

int main()
{
  for (int i = 3; i <= 100; i += 2) {
    bool b = true;
    for (int j = 3; j*j <= i; j += 2) {
      if (true == divisible(i,j)) {
        b = false;
        break;                          // break statement, exit the loop immediately
      }
    }
    if (b == true) {
      cout << "number is prime: " << i << endl;
    }
  }

  return 0;
}
```

3 Continue: example

- Let us loop through a set of integers, say i.
    1. If i is divisible by 2, we print a line to say i is divisible by 2.
    2. If i is also divisible by 3, we print a line to say i is divisible by 2 and 3.
    3. If i is also divisible by 5, we print a line to say i is divisible by 2 and 3 and 5.
- We can code this using a set of nested loops. This leads to a lot of nested loops.
- However, we can also code it without nested loops.
    1. If i is *not* divisible by 2, skip this value and go to the next value of i.
    2. Else print a line to say i is divisible by 2.
    3. If i is *not* divisible by 3, skip this value and go to the next value of i.
    4. Else print a line to say i is divisible by 2 and 3.
    5. If i is *not* divisible by 5, skip this value and go to the next value of i.
    6. Else print a line to say i is divisible by 2 and 3 and 5.

- **See next page.**

```
#include <iostream>

using namespace std;

bool divisible(int i, int n)
{
  return ((i % n) == 0);
}

int main()
{
  for (int i = 6; i <= 30; i += 4) {                          // nested loops
    if (true == divisible(i,2)) {
      cout << "divisible by 2: " << i << endl;
      if (true == divisible(i,3)) {
        cout << "divisible by 2 and 3: " << i << endl;
        if (true == divisible(i,5)) {
          cout << "divisible by 2 and 3 and 5: " << i << endl;
        }
      }
    }
  }

  cout << endl;
  for (int i = 6; i <= 30; i += 4) {                          // no nested loops
    if (false == divisible(i,2)) continue;                    // continue statement
    cout << "divisible by 2: " << i << endl;
    if (false == divisible(i,3)) continue;
    cout << "divisible by 2 and 3: " << i << endl;
    if (false == divisible(i,5)) continue;
    cout << "divisible by 2 and 3 and 5: " << i << endl;
  }

  return 0;
}
```

4 Break and continue: example with for and while loops

- We are given an input array a of length n, and the array is sorted in ascending order.
- We are also given two numbers b and c and we wish to sum all the a_i such that $b \leq a_i \leq c$.
- Therefore we loop through the array elements from $i = 0$ to $i = n - 1$.
- If $a_i < b$, we continue to the next value of i.
- If $a_i > c$, we break out of the loop, because the array is sorted, hence all subsequent values of a_i are greater than c, hence there is no need to test them.
- The main program contains a while loop with simple `continue` and `break` statements, just to demonstrate the use of `continue` and `break` statements in a `while` loop.

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

int sum_abc(int n, const int a[], int b, int c)
{
  int sum = 0;
  for (int i = 0; i < n; ++i) {
    if (a[i] < b) continue;                          // continue
    if (a[i] > c) break;                             // break
    sum = sum + a[i];
  }
  return sum;
}

int main()
{
  const int n = 100;
  int a[n];
  for (int i = 0; i < n; ++i)
    a[i] = i*i;

  int b = 1000;
  int c = 9999;
  while (true) {
    if (b >= c) break;                           // break out of while loop
    b += 700;
    if (b % 3 == 0) continue;                    // continue statement in while loop
    c -= 500;
    cout << b << "   " << c << "   " << sum_abc(n, a, b, c) << endl;
  }
  return 0;
}
```