Ch07-Iteration

October 8, 2020

1 Iterations or Loops

1.1 Topics

- increment and decrement operators
- iteration and types of iterations
- iteration applications
- iterations inside functions

1.2 Increment and decrement operators

- in programming adding and subtracting an integer value by 1 is done frequently
- loops use them quite a bit
- C++ provies increment and decrement operators to make our life easier
- there are two types of increment or decrement
    1. post
    2. pre

1.2.1 post increment and post decrement

- syntax:

```
varName++;
varName--;
```

- value of variable varName is used first in the current operation
- value of varialbe varName is then increased or decreased by 1 for the next operation
    - value is incremented or decremented after its usage
    - hence: post increment or post decrement

[1]:
```
// post increment example
#include <iostream>
using namespace std;

int x;
```

[2]:
```
// store 10 in x
x = 10;
```

[3]:
```
// use the current value of x and then increment it
cout << x++ << endl;
```

```
10
```

[4]:
```
// value of x should be incremented by 1
cout << x;
```

```
11
```

[5]:
```
// post decrement
x--
```

[5]:
```
11
```

[6]:
```
x
```

[6]:
```
10
```

1.2.2 pre increment or decrement

- syntax:

```
++varName;
--varName;
```

- value of varialbe varName is first increased or decreased by 1
- new value of variable varName is used in the same operation
  - value is incremented or decremented before its usage
  - hence: pre increment or pre decrement

[7]:
```
// pre increment and decrement examples
x = 10;
```

[8]:
```
--x
```

[8]:
```
9
```

[9]:
```
++x
```

[9]:
```
10
```

1.3 Iteration or loop

- iteration is commonly called loop in programming
- real life is full of loops
  - routine works one does day after day
  - e.g. wake up, get ready, eat breakfast, commute to school/work, eat lunch, commute back home, eat dinner, sleep; repeat!

- computer is really good at automatically doing repeative tasks (millions and billions of reptitions)
  - repeating identical or similar tasks without errors or boredom is something computers do well and people do poorly
  - computers can also do those tasks many times faster than humans
- iteration strats at a starting point and repeats or loops from the same starting point
  - a block of code can be repeateadly executed using just a one or two lines of loop structures
- just like in real-life, loop must end/exit at some point; otherwise you'll get into infinite loop

1.4 Types of C++ loops

- there are 4 commonly used types of loop in C++
  1. for loop
  2. range-based for loop
  3. while loop
  4. do while loop

1.5 for loop

- very common reptition control structure
- normally executes a specific number of times
- syntax

```
for(initialization; condition; updatation) {
    // body of the loop
}
```

- interpreting for loop:
  1. initialization: initialize loop variables
  2. condition - check condition
  3. exit or execute loop body 3.a if condition is true, execute code in body of the loop 3.b exit the loop otherwise
  4. updation: update the loop variables
  5. repeat from step 2
- the following figure depicts the execution flow of **for loop**

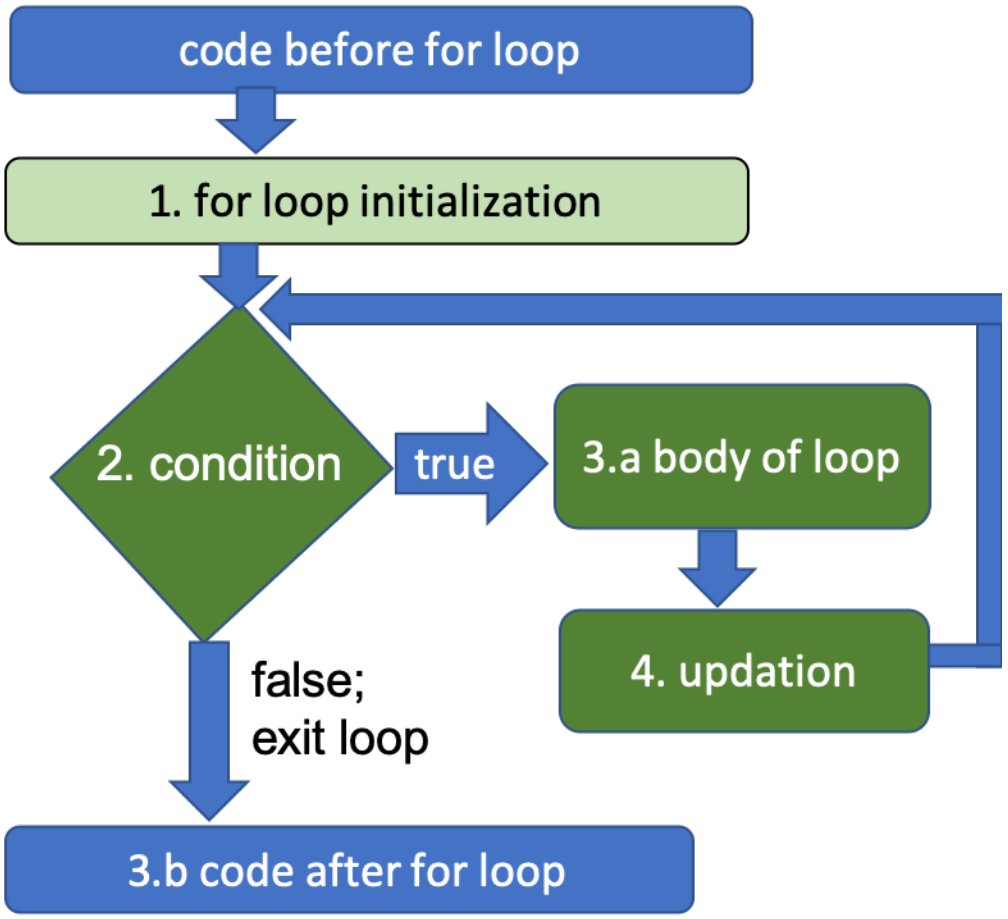

Fig. C++ For Loop Flow Chart

[10]:
```
// example 1 - the hard way of repeating code!
// write a program that counts "Mississippi!" 10 times
// if you didn't know loop, one could still do it, rather painfully!
// typing one statement at a time for 10 times!

#include <iostream>

using namespace std;

cout << "1. Mississippi!\n";
cout << "2. Mississippi!\n";
cout << "3. Mississippi!\n";
cout << "4. Mississippi!\n";
cout << "5. Mississippi!\n";
cout << "6. Mississippi!\n";
cout << "7. Mississippi!\n";
cout << "8. Mississippi!\n";
cout << "9. Mississippi!\n";
cout << "10. Mississippi!\n";
// phew... gets worse, when you need to do it for 100 or 1000 or more times...␣
↪Yikes!!
// you might just quit programming right now!
```

```
1. Mississippi!
2. Mississippi!
3. Mississippi!
4. Mississippi!
5. Mississippi!
6. Mississippi!
7. Mississippi!
8. Mississippi!
9. Mississippi!
10. Mississippi!
```

[10]: @0x10597fed0

[11]:
```
// Let's make our life a little easier!
// using for loop, let's tell the computer to repeatedly print "Mississippi!"␣
↪10 times
// so we don't have to type 10 different statements!
for(int i=1; i<=10; i++) {
    // it's common practice that i, j, k are used as loop counter variables
    // you can use any identifier
    cout <<  i << ". Mississippi!\n";
}
// how about counting "Mississippi!" 100 times or even 1000 and more?
```

```
1. Mississippi!
2. Mississippi!
3. Mississippi!
4. Mississippi!
5. Mississippi!
6. Mississippi!
7. Mississippi!
8. Mississippi!
9. Mississippi!
10. Mississippi!
```

1.5.1 visualize for loop execution in pythontutor.com

1.5.2 intitialization, condtion and updation statements can be 0 or more

- the intitialization, condtion and updation expressions in the **for loop** statment are all optional
- these can also have multiple statments separated by comma

1.5.3 infinite loop

- a common mistake a programmer can make while constructing a loop
- happens when you forget to update the loop variable or use condition that is never false

[ ]:
```
// infinite loop
// if you run this, computer will not stop executing!
// you've to manually interrupt the Kernel in Jupyter notebook
// Kernel -> Inerrrput
for( ; ; ) { // infinite loop; no condition that stops the for loop
    cout << "Hello World!" << endl;
}
```

[1]:
```
// if the kernel restarts; must include all the libraries again
#include <iostream>
using namespace std;
```

[2]:
```
// for loop with multiple statements for initialization; condition and updation
for(int i=1, j=10; i<=10 && j>=1; i++, j--) {
    cout << i << " + " << j << " = " << i+j << endl;
}
```

```
1 + 10 = 11
2 + 9 = 11
3 + 8 = 11
4 + 7 = 11
5 + 6 = 11
6 + 5 = 11
7 + 4 = 11
8 + 3 = 11
9 + 2 = 11
10 + 1 = 11
```

1.6 break and continue

- two commonly used statments inside the loop body
- **break** statement breaks/exits the loop immidiately when executed
  - any code after **break** inside the loop body will be ignored
- **continue** statement makes next loop or iteration to execute immidiately
  - any code after **continue** inside the loop body will be skipped
- break and continue are typically used based on some condition
  - with the reason to exit the loop body or continue with next iteration

[2]:
```
// example of break and continue
// comment and uncomment break and continue to see how each works
for(int i=1; i<=10; i++) {
    cout <<  i << ". Mississippi!\n";
    break;
    cout << i << ". Hello World!\n";
    // continue;
}
```

```
1. Mississippi!
```

[3]:
```
// break example
cout << "before loop\n";
for(int i=1; i<=10; i++) {
    if (i == 5)
        break;
    cout << i << ". Hello World!" << endl;
}
cout << "after loop... all done!\n";
```

```
before loop
1. Hello World!
2. Hello World!
3. Hello World!
4. Hello World!
after loop… all done!
```

[6]:
```
// continue example
// print odd numbers from 1 to 20
cout << "before loop\n";
for(int i=1; i<=20; i++) {
    if (i%2 == 0) // skip every even i
        continue;
    cout << i << " ";
}
cout << "\nafter loop... all done!\n";
```

```
before loop
1 3 5 7 9 11 13 15 17 19
after loop… all done!
```

[2]:
```
// breaking infinite loop!
int i=1, j=10;
for( ; ; ) {
    if (i<=10 && j>=1)
        cout << i << " + " << j << " = " << i+j << endl;
    else
        break;

    i++, j--;
}
```

```
1 + 10 = 11
2 + 9 = 11
```

```
3 + 8 = 11
4 + 7 = 11
5 + 6 = 11
6 + 5 = 11
7 + 4 = 11
8 + 3 = 11
9 + 2 = 11
10 + 1 = 11
```

[5]:
```
// Countdown to Blast-off using for loop
#include <iostream>
#include <unistd.h>

using namespace std;
```

[3]:
```
for(int i=10; i>=0; i--) {
    if (i == 0)
        cout << "Blast Off!!!" << endl;
    else {
        cout << i << endl;
        usleep(1000000); // sleep for 10^6 microseconds = 1 second
    }
}
```

```
10
9
8
7
6
5
4
3
2
1
Blast Off!!!
```

1.6.1 Countdown demo program - demo_programs/Ch07/countdown/countdown.cpp

1.7 Range-based for loop

- executes a for loop over a range of values
- typically used with container types such as array, vector, set, etc.
  - more readable alternative to the for loop operating over all elements in a container
- syntax:

```
for (range_declaration : range_expression) {
    // loop body
}
```

- range declaration:

    - declares a range variable, whose type is the type of the element in the sequence in range_expression
- range expression:
    - represents a suitalbe sequence of elements such as array or container with begin and end member functions

[3]:
```
// example of range-based for loop
cout << "before range-based loop...\n";
for(int num: {1, 2, 4, 5, 6, 8, 9, 10}) {
    cout << num << ". Mississippi!\n";
}
cout << "after range-based loop... all done!";
```

```
before range-based loop…
1. Mississippi!
2. Mississippi!
4. Mississippi!
5. Mississippi!
6. Mississippi!
8. Mississippi!
9. Mississippi!
10. Mississippi!
after range-based loop… all done!
```

[7]:
```
// example 2: iterate over each character in string using range-based loop
string text="This is a sentence!";
```

[11]:
```
#include <cstring> // touppper

// recall: auto can be used to autmaitcally determine type based on value
for(auto ch: text) {
    cout << ch << " -> ASCII: " << int(ch) << " UPPER: " << char(toupper(ch))␣
 ↪<< endl;
}
```

```
T -> ASCII: 84 UPPER: T
h -> ASCII: 104 UPPER: H
i -> ASCII: 105 UPPER: I
s -> ASCII: 115 UPPER: S
  -> ASCII: 32 UPPER:
i -> ASCII: 105 UPPER: I
s -> ASCII: 115 UPPER: S
  -> ASCII: 32 UPPER:
a -> ASCII: 97 UPPER: A
  -> ASCII: 32 UPPER:
s -> ASCII: 115 UPPER: S
e -> ASCII: 101 UPPER: E
n -> ASCII: 110 UPPER: N
```

```
t -> ASCII: 116 UPPER: T
e -> ASCII: 101 UPPER: E
n -> ASCII: 110 UPPER: N
c -> ASCII: 99 UPPER: C
e -> ASCII: 101 UPPER: E
! -> ASCII: 33 UPPER: !
```

[13]:
```
// convert a text into all uppercase
string upper_text = ""; // variable to collect uppercase chacaters
for(auto ch: text) {
    upper_text += char(toupper(ch));
}
cout << text << " -> " << upper_text << endl;
```

```
This is a sentence! -> THIS IS A SENTENCE!
```

1.8 while loop

- **while** keyword is used to create **while** statment
  - a loop that iterates 0 or more times
- if you're not sure how many times the loop should iterate, you use **while** loop
- while statement can be read as if it were Englisth
  - e.g. while you're not tired, keep running the track.
  - as opposed to **for loop** that says, run the track 10 times.
- syntax:

```
while(condition) {
    // body of loop
}
```

- execute the body of loop as long as the condition is true
- the following figure depicts while loop execution

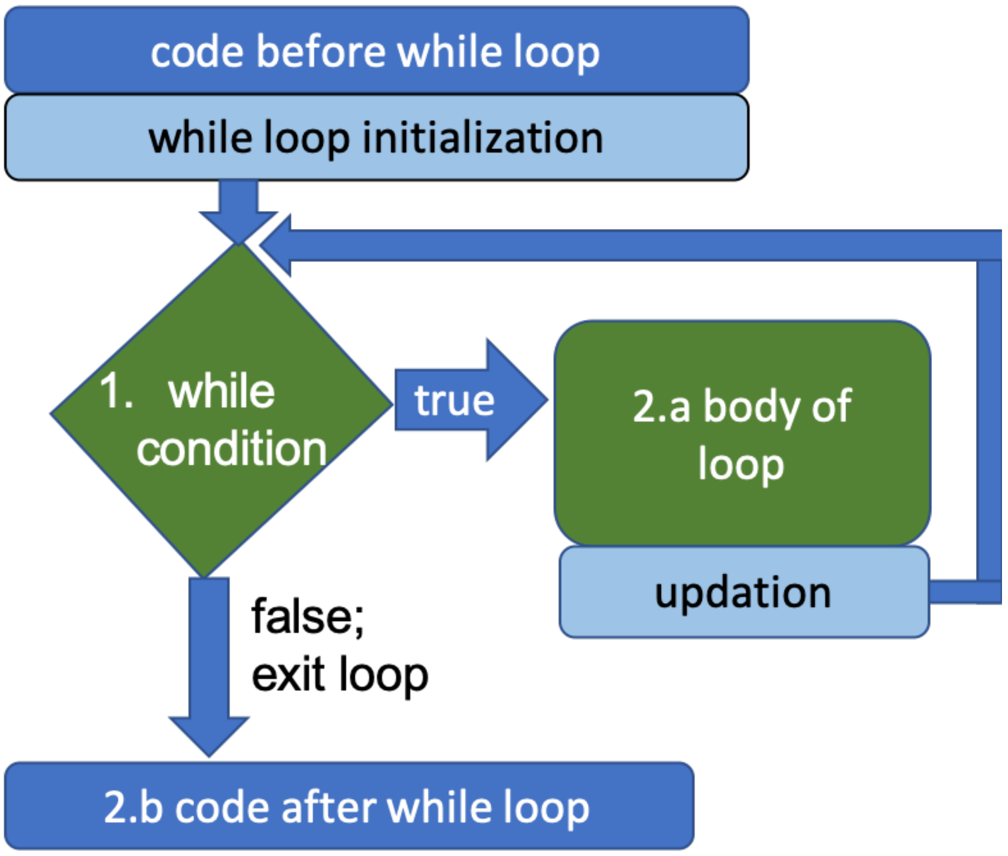

Fig. C++ While Loop Flow Chart

[1]:
```
// example 1 - print a log table from 1 to 10
#include <iostream>
#include <cmath> // log, log2, log10

using namespace std;

double x; // while loop initialization
```

[2]:
```
cout << "x\tlog(x)\tlog2(x)\tlog10(x)\n";
cout << fixed << setprecision(4) << endl;
x = 1.0;
while(x <= 10.0) {
    // natural log base e, base 2 and base 10
    cout << x << '\t' << log(x) << '\t' << log2(x) << '\t' << log10(x) << endl;
    x += 1.0; // update loop variable
}
```

```
x       log(x)  log2(x) log10(x)
```

```
1.0000  0.0000  0.0000  0.0000
2.0000  0.6931  1.0000  0.3010
3.0000  1.0986  1.5850  0.4771
4.0000  1.3863  2.0000  0.6021
5.0000  1.6094  2.3219  0.6990
6.0000  1.7918  2.5850  0.7782
7.0000  1.9459  2.8074  0.8451
8.0000  2.0794  3.0000  0.9031
9.0000  2.1972  3.1699  0.9542
10.0000 2.3026  3.3219  1.0000
```

[2]:
```
// example 2 - run the track until you're tired
int lapCount = 0;
string tired_response;
bool tired = false; // while loop initialization
```

[3]:
```
while(not tired) {
    lapCount += 1;
    cout << "lap count = " << lapCount << endl;
    cout << "Are you tired yet? [y|yes] or [n\no]: ";
    cin >> tired_response;
    if (tired_response == "y" or tired_response == "yes")
        tired = true; // update loop variable
}
```

```
lap count = 1
Are you tired yet? [y|yes] or [n
o]: n
lap count = 2
Are you tired yet? [y|yes] or [n
o]: no
lap count = 3
Are you tired yet? [y|yes] or [n
o]: yes
```

[5]:
```
// using break and continue statements in while loop
// NOTE: they don't have to be used together!
lapCount = 0;
while(true) {
    lapCount += 1;
    if (lapCount == 2) continue; // skip the rest of the code
    cout << "lap count = " << lapCount << endl;
    cout << "Are you tired yet? [y|yes] or [n\no]: ";
    cin >> tired_response;
    if (tired_response == "y" or tired_response == "yes")
        break;
}
```

```
lap count = 1
Are you tired yet? [y|yes] or [n
o]: n
lap count = 3
Are you tired yet? [y|yes] or [n
o]: no
lap count = 4
Are you tired yet? [y|yes] or [n
o]: yes
```

1.8.1 visualize while loop in pythontutor.com

1.9 do-while loop

- do while loop is an extension of while loop
- makes a block of code execute 1 or more times
- syntax:

```
do {
    // body of loop
} while (condition);
```

- notice the semi-colon after while statement
- interpreting do-while loop
    1. do execute the block of code at least once
    2. while the condition is true go to step 1
        - exit the loop otherwise
- the following figure depicts the execution flow of **do-while loop**

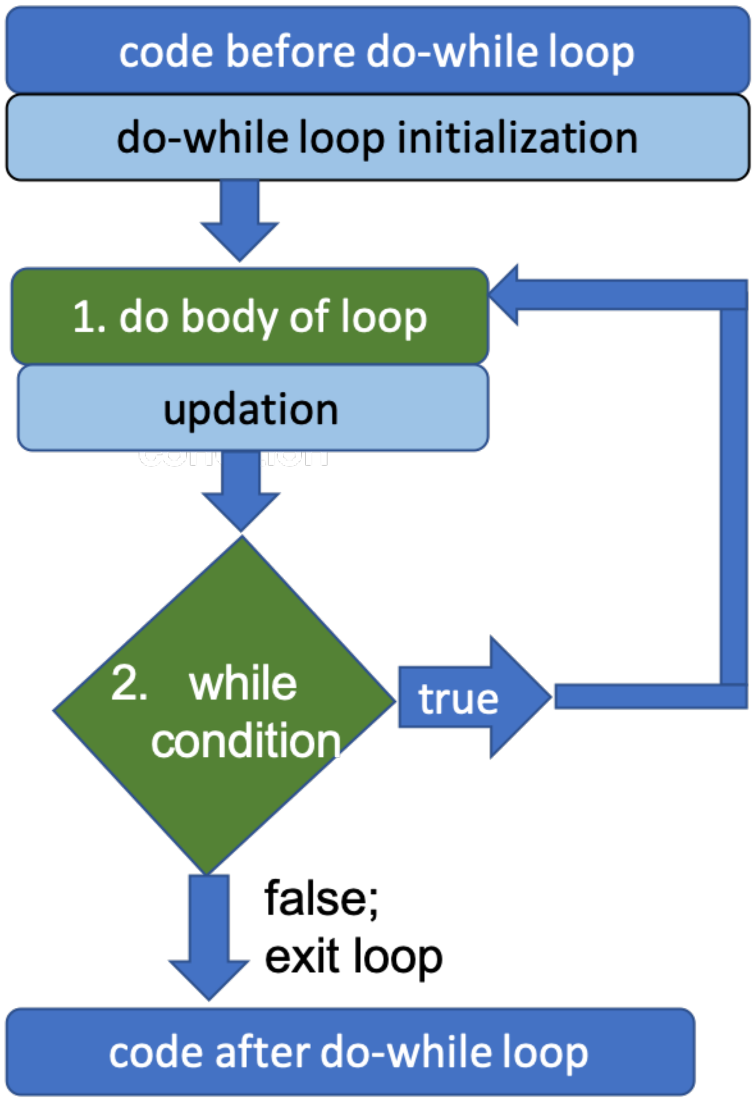

Fig. C++ Do-While Loop Flow Chart

[6]:
```
// example 1 - game play simulation
// initialize loop variables
int counter = 0; // keep track of no. of times game is played
string play_again; // player's response after each game
```

[7]:
```
// play game at least once
do {
    // call game() function or implement game here...
    counter++;
    cout << "played " << counter << " times.\n";
    cout << "want to play again? [y|n]: ";
    cin >> play_again;
    if (play_again != "y") break;
    else continue; // not necessary!
} while (true);
```

```
played 1 times.
want to play again? [y|n]: y
played 2 times.
want to play again? [y|n]: y
played 3 times.
want to play again? [y|n]: n
```

[1]:
```
// example 2 - input validation
int input; // variable to store user input
```

[6]:
```
do {
    cout << "Enter a whole number between 1 and 20: ";
    cin >> input;
    if (cin.fail()) { // somehow cin failed; wrong type is entered
        cin.clear(); // clear the error flag
        cin.ignore(INT_MAX, '\n'); // extract and discard upto INT_MAX␣
 ↪characters or upto '\n' in std input stream
        cout << "Invalid input. Try again!\n";
        continue;
    }
    else if (input < 1 || input > 20) {
        cout << "input must be a whole number between 1 and 20\n";
    }
    else break;
} while (true);
```

```
Enter a whole number between 1 and 20: -1
input must be a whole number between 1 and 20
Enter a whole number between 1 and 20: 21
input must be a whole number between 1 and 20
Enter a whole number between 1 and 20: asdf
Invalid input. Try again!
Enter a whole number between 1 and 20: sdfaf12
Invalid input. Try again!
Enter a whole number between 1 and 20: 15
```

[7]:
```
cout << "Great! You entered: " << input << endl;
```

```
Great! You entered: 15
```

1.9.1 see example 2 input validation as a function here demo_programs/Ch07/input_validate/input_validation.cpp

1.10 Loops in functions

- all the loop statements can be used inside a function
- in fact, any fundamental concepts (io, math, operations, conditionals, loops, etc.) can be used inside loop
- functions can be called inside loop body

1.10.1 write a function that prints a multiplication table from 1 to 10 as shown below

- use composition and incremental development

```
    1    2    3    4    5    6    7    8    9   10
    2    4    6    8   10   12   14   16   18   20
    3    6    9   12   15   18   21   24   27   30
    4    8   12   16   20   24   28   32   36   40
    5   10   15   20   25   30   35   40   45   50
    6   12   18   24   30   36   42   48   54   60
    7   14   21   28   35   42   49   56   63   70
    8   16   24   32   40   48   56   64   72   80
    9   18   27   36   45   54   63   72   81   90
   10   20   30   40   50   60   70   80   90  100
```

[3]:
```
#include <iostream>
#include <iomanip>

using namespace std;
```

[1]:
```
// function that multiplies two numbers
int multiply(int n1, int n2) {
    return n1*n2;
}
```

[4]:
```
// function prints multiples of N from 1 to 10
void print_multiples(int N) {
    for(int i=1; i<=10; i++)
        cout << setw(5) << multiply(N, i);
    cout << endl;
}
```

[5]:
```
print_multiples(1);
```

```
    1    2    3    4    5    6    7    8    9   10
```

[6]:
```
print_multiples(2);
```

```
    2    4    6    8   10   12   14   16   18   20
```

[7]:
```
// now print_mutiples need to be called 10 times
void printMultipleTable() {
    for(int i=1; i<=10; i++)
        print_multiples(i);
}
```

[8]:
```
printMultipleTable();
```

```
    1    2    3    4    5    6    7    8    9   10
    2    4    6    8   10   12   14   16   18   20
    3    6    9   12   15   18   21   24   27   30
    4    8   12   16   20   24   28   32   36   40
    5   10   15   20   25   30   35   40   45   50
    6   12   18   24   30   36   42   48   54   60
    7   14   21   28   35   42   49   56   63   70
    8   16   24   32   40   48   56   64   72   80
    9   18   27   36   45   54   63   72   81   90
   10   20   30   40   50   60   70   80   90  100
```

1.11 Nested loops

- a loop can be nested inside another
- outer loop reapeats everyting inside inner loop
- a lot of advanced algorithms and problems require nested double and even tripple loops

[9]:
```
// function prints multiplication table using nested loop
// print_multiples function's code is used as an inner loop
void multiplicationTable() {
    for(int i=1; i<=10; i++) { // for every i... (row)
        for(int j=1; j<=10; j++) // column
            cout << setw(5) << multiply(i, j);
        cout << endl;
    }
}
```

[10]:
```
multiplicationTable();
```

```
    1    2    3    4    5    6    7    8    9   10
    2    4    6    8   10   12   14   16   18   20
    3    6    9   12   15   18   21   24   27   30
    4    8   12   16   20   24   28   32   36   40
    5   10   15   20   25   30   35   40   45   50
    6   12   18   24   30   36   42   48   54   60
    7   14   21   28   35   42   49   56   63   70
    8   16   24   32   40   48   56   64   72   80
    9   18   27   36   45   54   63   72   81   90
   10   20   30   40   50   60   70   80   90  100
```

1.11.1 define a function that prints a right-traingle shape with some symbol such as * of given height N

- e.g. the following is a righ-right triangle of height 5 printed with *

```
*
* *
* * *
* * * *
* * * * *
```

[15]:
```
// solution
void printTriangle(char ch, int height) {
    for(int i=1; i<=height; i++) {
        for(int j=1; j<=i; j++)
            cout << ch << " ";
        cout << endl;
    }
}
```

[16]:
```
printTriangle('*', 5);
```

```
*
* *
* * *
* * * *
* * * * *
```

1.11.2 Rectanlge - demo program

- Write a complete C++ that computes area and perimeter of a rectangle given length and width.
- write at least 3 test cases for each function
- program must calculate area and perimeter of as many rentangles as the user wants
- see sample solution here: demo_programs/Ch07/rectangle/

1.12 Exercises

1. Write a function that prints multiplication table from 1 to some value N.
   - program only prints the lower half of the table ignoring all the redundant upper half values
2. Write a C++ program including algorithm steps that calculates area and circumference of a circle.
   - must write functions to compute area and perimeter and automatically test each function with atleast 3 test cases
   - **program finds area and perimeter of as many circle as the user wishes**
3. Write a C++ program including algorithm steps that calculates Body Mass Index (BMI) of a person.

- must use as many functions as possible
- write at least 3 test cases for each function
- more info on BMI - https://www.nhlbi.nih.gov/health/educational/lose_wt/BMI/bmicalc.htm
- formula found here
- **program must calculate BMI of as many patients as user wants**
- a sample solution is provided at `exercises/Ch07/BMI/BMI_v4.cpp`

4. Write a C++ program including algorithm steps that computes area and perimeter of a triangle given three sides.
   - must write and use separate functions to calculate area and perimeter
   - write at least 3 test cases for each function
   - **program computes area and perimeter of as many triangles as user wishes**
   - Hint: use Heron's formula to find area with three sides
5. Write a C++ program that converts hours into seconds.
   - must write and use function(s) to computer answer(s)
   - must write at least 3 test cases for each function
   - e.g. given 2 hours, program should print 7200 as answer
   - **program continues to run converting as many hours into seconds as the user wishes without restaring it**
6. Write a C++ program that converts seconds into hours, minutes and seconds.
   - must define and use function(s)
   - write at least 3 test cases for each function
   - sample input: 3600 sample output: 1 hour, 0 minute and 0 second
   - sample input: 3661 sample output: 1 hour, 1 minute and 1 second
   - Hint: use series of division and module operations **program will continue to run converting multiple inputs**
7. Write a C++ program that counts a number of even digits in a given integer.
   - must write function and write atleast 3 test cases
   - **program must contine to run as many times as the user wishes**
8. Write a C++ program that converts decimal number into binary. See Chapter 2 and 3 for the algorithm and partial solution.
   - **program will continue to run converting as many decmial number as the user wishes**
9. Write a C++ program that converts binary number into decimal. See Chapter 2 and 3 for the algorithm.
10. Write a C++ program that determines if the given integer is prime.
11. Write a C++ program that does countdown for rocket launch. Must use for loop.
    - prints count down from 10 to 1 and finally prints "Blast Off!"
12. Write a C++ program that does countdown for rocket launch. Must use while loop.
    - prints count down from 10 to 1 and finally prints "Blast Off!"

13. Write a program that prints a right-traingle shape with some symbol such as * and given height N

    - e.g. the following is a righ-right triangle of height 5 printed with *

```
* * * * *
* * * *
* * *
* *
*
```

14. Write a program that prints a square shape with some symbol such as * and given height N
    - e.g. the following is a square of height 5 printed with *

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

1.13 Kattis Problems

- with all the fundamental concepts covered so far, one should be equipped to solve a lot more problems in Kattis.
- all most every Kattis problem needs loop to process large amount of data or test cases
- some of the Kattis problems that require loop (and of course other concepts that have been covered so far ar listed below)

1.13.1 solve each Kattis problem with at least 3 automated test cases

- test case should try to address the corner/edge cases
- use the data other than the ones provides by the problem

1. Oddities - https://open.kattis.com/problems/oddities

    - a sample solution can be found here:
    - https://github.com/rambasnet/KattisDemos/tree/master/oddities

2. Cold-puter Science - https://open.kattis.com/problems/cold

    - a sample solution can be found here:
    - https://github.com/rambasnet/KattisDemos/tree/master/cold

3. Help a PhD Candidate Out! - https://open.kattis.com/problems/helpaphd

    - a sample solution can be found here:
    - https://github.com/rambasnet/KattisDemos/blob/master/helpaphd/helpaphd.cpp

4. Egypt - https://github.com/rambasnet/KattisDemos/blob/master/egypt/egypt.cpp

    - a sample solution can be found here:
    - https://github.com/rambasnet/KattisDemos/blob/master/egypt/egypt.cpp

5. FizzBuzz - https://open.kattis.com/problems/fizzbuzz

6. Stuck In A Time Loop - https://open.kattis.com/problems/timeloop
7. Heart Rate - https://open.kattis.com/problems/heartrate
8. Reversed Binary Numbers - https://open.kattis.com/problems/reversebinary
9. Modulo - https://open.kattis.com/problems/modulo
10. Quality-Adjusted Life-Year - https://open.kattis.com/problems/qaly
11. Tarifa - https://open.kattis.com/problems/tarifa
12. Judging Moose - https://open.kattis.com/problems/judgingmoose
13. Tower Construction - https://open.kattis.com/problems/tornbygge

1.14 Summary

- learned another fundamental programming concept: iteration or loop
- learned that there are 4 types of loops (2 are for loops and 2 are while loops)
- leaned two import keywords break and continue that are used inside loops
- learned that functions can be called inside loop body and loops can be written inside functions
- learned about nested loop with some example applications
- exercise and example solutions

```
[ ]:
```