Unit 8

'for' Loops

Side Topic: Pre-/Post- Increment/Decrement

- Recall the increment and decrement operators: ++ and --
  - If ++ comes before a variable it is call pre-increment; if after, it is called post-increment
  - `x++; // If x was 2 it will be updated to 3 (x = x + 1)`
  - `++x; // Same as above (no difference when not in a larger expression)`
  - `x--; // If x was 2 it will be updated to 1 (x = x – 1)`
  - `--x; // Same as above (no difference when not in a larger expression)`
- Difference between pre- and post- is only evident when used in a larger expression
- Meaning:
  - Pre: Update (inc./dec.) the variable before using it in the expression
  - Post: Use the old value of the variable in the expression then update (inc./dec.) it
- Examples [suppose we start each example with: `int y; int x = 3;`]
  - `y = x++ + 5;  // Post-inc.; Use x=3 in expr. then inc. [y=8, x=4]`
  - `y = ++x + 5;  // Pre-inc.; Inc. x=4 first, then use in expr. [y=9, x=4]`
  - `y = x-- + 5;  // Post-dec.; Use x=3 in expr. then dec. [y=8, x=2]`
  - `y = --x + 5;  // Pre-dec.; Dec. x=2 first, then use in expr. [y=7, x=2]`

Control Structures

- We need ways of making **decisions** in our program
  - To repeat code until we want it to stop
  - To only execute certain code if a condition is true
  - To execute one segment of code or another
- Language constructs that allow us to make decisions are referred to as **control structures**
- The common ones are:
  - if statements
  - switch statements
  - while loops
  - for loops

for Loops

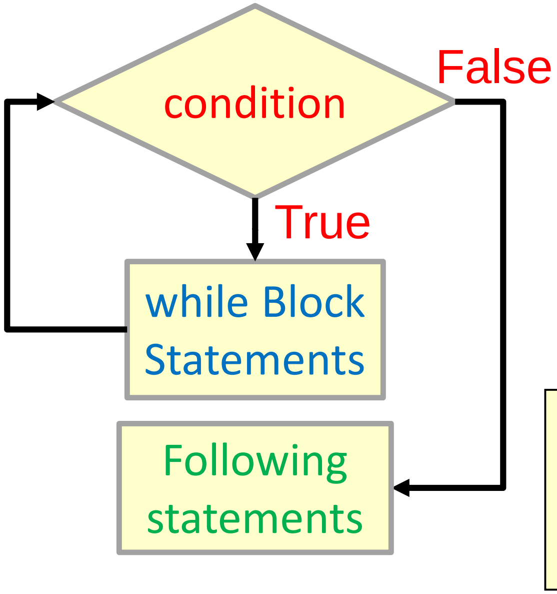
- 'for' loops provide additional syntax for initialization and an update after each iteration

```
while (condition)
{
  // executed if condition is true
} // go to top, eval cond. again

// following statements
// only gets here when cond. is false
```

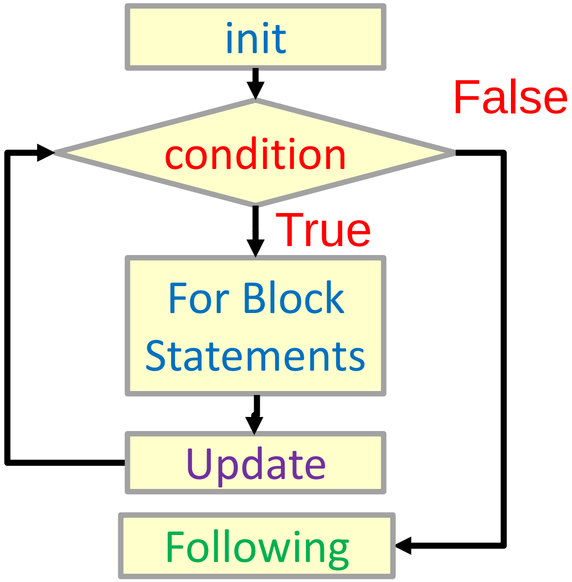
```
for( init; condition; update)
{
  // executed if condition is true
} // go to top, do update, eval cond. again

// following statements
// only gets here when cond. is false
```



Example

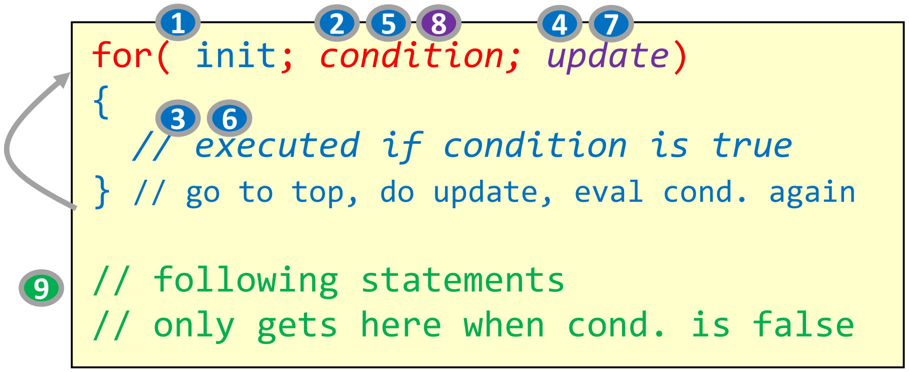
```
for( int i=0; i < 5; i++)
{
  cout << i << endl;
}
```



'for' Loop Sequencing

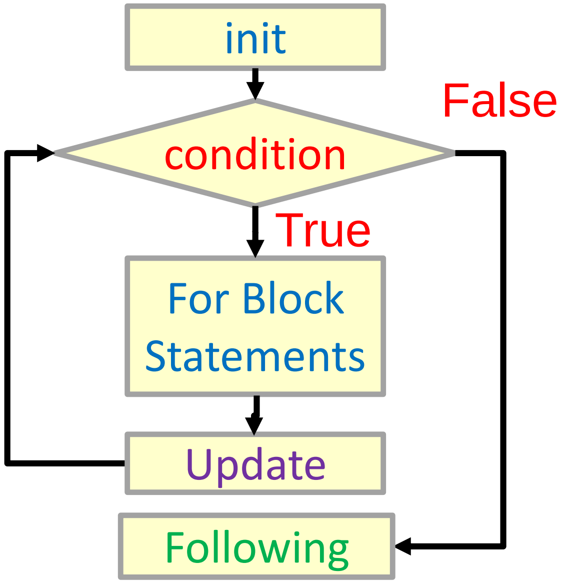
- 'for' loop
  - performs init statement once
  - checks the condition each iteration before deciding to execute the body or end the loop
  - performs the update statement after each execution of the body

Condition: T T F



```
for( init; condition; update)
{
  // executed if condition is true
} // go to top, do update, eval cond. again

// following statements
// only gets here when cond. is false
```



Some Examples

```
#include <iostream>
using namespace std;
int main()
{
  int i;
  for(i=0; i < 5; i++)
  {
    cout << i << endl;
  }
  return 0;
}
```

Program Output:

```
0
1
2
3
4
```

```
#include <iostream>
using namespace std;
int main()
{
  int i;
  for(i=8; i > 0; i=i/2 )
  {
    cout << i << endl;
  }
  return 0;
}
```

Program Output:

```
8
4
2
1
```

The initial value, condition, and update statement can be any valid expression!

Sets and For Loops

- For loops can often be used to generate or iterate over all the elements of a set
- For loops will usually utilize some variable to track/count how many iterations have elapsed
  - This is often known as the **inductive** or **control** variable
- If we want to iterate n times, the common idiom is to start at 0 and iterate through n-1, stopping on n
  - This is not a requirement; we can start where we like

Generate the first 10 multiples of 3:

$$\boldsymbol{S = \{3i \mid i \in \mathbb{N}, 1 \le i \le 10\}}$$

```
int i;
// print first 10 multiples of 3
for(i=1; i <= 10; i++)
{
  cout << 3*i << endl;
}
// What is i when the loop ends?
```

Generate the first 20 positive odd #s

$$\boldsymbol{S = \{2i + 1 \mid i \in \mathbb{N}, 0 \le i \le 19\}}$$

```
int i;
// print first 20 pos. odd #s
for(i=0; i < 20; i++)
{
  cout << 2*i+1 << endl;
}
// What is i when the loop ends?
```

Tangent: Scope

- A tangent that will be relative in our discussion of for loops is the idea of scope
- Scope refers to the **lifetime** and **visibility** of a variable
  - Recall variables are just memory slots in the computer
  - The program will reclaim those memory spots when a variable "dies"
- In C/C++, a variable's scope is the curly braces {} it is declared within
- **Main Point**: A variable dies at the end of the {…} it was declared in

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

  if(i > 0){
     int temp = 2*i;
     cout << temp << endl;
  }  // temp died here
  temp = i++; // won't compile
  cout << temp << endl;

  return 0;
} // i dies here
```

Declaring the Inductive Variable

- The initialization statement can be used to declare a control/inductive variable but its scope is considered to be the for loop (even though it is not technically declared in the {..} of the for loop
  - **Just realize that variable will die at the end of the loop**
- However, because it dies after the first loop you can use that same variable name in a subsequent loop

```
#include <iostream>
using namespace std;
int main()
{
  int n;
  cin >> n;
  for(int i=0; i < n; i++){
     cout << 3*i << endl;
  } // i dies here

  // won't compile
  cout << i << endl;

  // okay to reuse i
  for(int i=0; i < n; i++){
     cout << 4*i << endl;
  } // reincarnated i dies again

  return 0;
} // n dies here
```

Hand Tracing (1)

- For the first program, trace through the code and show all changes to i for:
  - n = 2;
- For the second program, trace through the code and show the output for:
  - t = PI/2, T = 2*PI

```
int main()
{
  int n;
  cin >> n;
  for(int i = -n; i <= n; i++)
  {
     cout << i << endl;
  }
  return 0;
}
```

```
int main()
{
  double t, T;
  cin >> t >> T;
  for( double th = 0 ; th < T; th += t)
  {
     cout << sin(th) << endl;
  }
  return 0;
}
```

Hand Tracing (2)

- For the first program, trace through the code and show all changes to i and y for:
  – x = 10
  – y = 2

```
int main()
{
  int x, y;
  cin >> x >> y;
  for(int i=1; i <= x; i=i+y)
  {
     cout << i << endl;
     y++;
  }
  return 0;
}
```

- For the second program, trace through the code and show all changes to i and y for:
  – x = 4
  – y = 11

```
int main()
{
  int x, y;
  cin >> x >> y;
  for(   ; x < y; x++)
  {
     cout << x << " " << y << endl;
     y--;
  }
  return 0;
}
```

Exercises 1

- Write a for loop to generate all the elements of the specified sets

$$\boldsymbol{S} = \{\mathbf{3, 7, 11, 15, 19, 23, 27, 31}\}$$

```
for(int i=0; i < 8; i++)
{
   cout << _________ << endl;
}
```

$$\boldsymbol{T} = \{\mathbf{5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5}\}$$

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

Exercises 2

- cpp/for/blastoff
- cpp/for/interest
- cpp/for/sum-mult-2-5
- cpp/for/bottles-wall

'while' or 'for'

While Loops

- Usually used to repeat code until some condition is false

```
int i=0;
/* how many iterations required */
while( i != -1 )
{
  cin >> i;
  cout << i << endl;
}
```

For Loops

- Usually used to repeat code some known amount of time
- Very useful to access **arrays** (which we will learn in a few weeks)

```
/* how many iterations required */
for(int i=0; i < 5; i++)
{
  cin >> i;
  cout << i << endl;
}
```

Common Loop Mistakes

- Updating the inductive variable in the wrong direction

```
int i=0, n=10;
for (i=n; i>0; i++) // oops, meant i--
{
  cout << "Iteration " << i << endl;
}
```

- Off by one error

```
// Print "Hello" 5 times
for (i=0; i<=5; i++) // oops, meant <
{
  cout << "Hello" << endl;
}
```

- Missing the exit condition

```
// Print "0", "2", and "4"
for (i=0; i!=5; i+=2) // oops, infinite
{
  cout << i << endl;
}
```

Converting while to for Loops

```
for(int i=0; i < 5; i++)
{
   cout << i << endl;
}
```

```
int i=0;
while(i < 5)
{
   cout << i << endl;
   i++;
}
```

```
cin >> guess;
while (guess != secretnum)
{
   cout << "Try again!" << endl;
   cin >> guess;
}
cout << "You got it!" << endl;
```

```
for( cin >> guess;
     guess != secretnum;
     cin >> guess)
{
   cout << "Try again!" << endl;
}
cout << "You got it!" << endl;
```

break Statement

- Sometimes we will want to iterate some number of times under normal circumstances, but stop iterating immediately if a certain condition is true (i.e. halt the loop)
- The break keyword will immediately cause the current loop to exit if it is executed
  - Note: break should always be in some kind of conditional (if or else) as otherwise the loop would only iterate once

```
/* Give the user 10 turns
    but stop if guess right */

int i, guess, secretNum = /* ... */
for(i=0; i < 10; i++)
{
  cin >> guess;
  if(guess == secretNum){
    break;
  }
}
if( i == 10 ){
  cout << "You lose!" << endl;
}
else {
  cout << "You win!" << endl;
}
```

Exercises 3

- cpp/for/rps-bestof3

Exercise 1 Solutions

- Write a for loop to generate all the elements of the specified sets

$$\boldsymbol{S} = \{\mathbf{3, 7, 11, 15, 19, 23, 27, 31}\}$$

```
for(int i=0; i < 8; i++)
{
   cout << 4*i+3 << endl;
}
//or
for(int i=3; i <=31; i+=4)
{
   cout << i << endl;
}
```

$$\boldsymbol{T} = \{\mathbf{5, 4, 3, 2, 1, 0, 1, 2, 3, 4, 5}\}$$

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