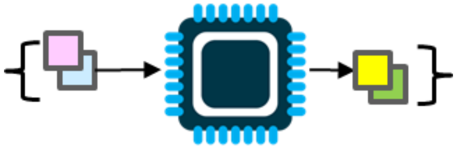
Unit 2a – Loop Syntax and Semantics

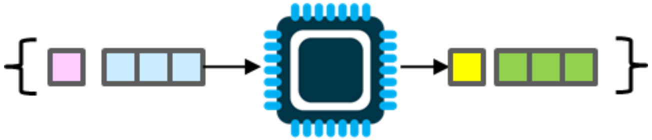
Mark Redekopp

Unit 2

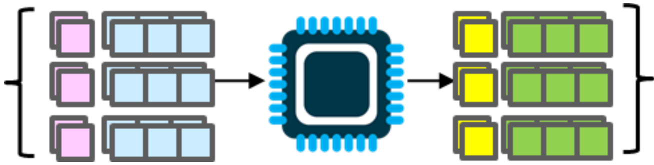
- **Unit 1**: Scalar processing
  - aka IPO=Input-Process-Output Programs



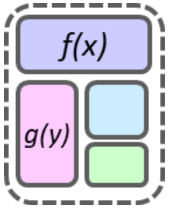
- **Unit 2**: Linear (1D) Processing



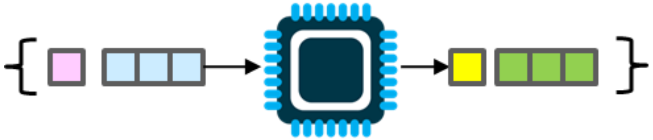
- **Unit 3**: Multidimensional Processing



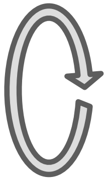
- **Unit 4**: Divide & Conquer (Functional Decomposition)

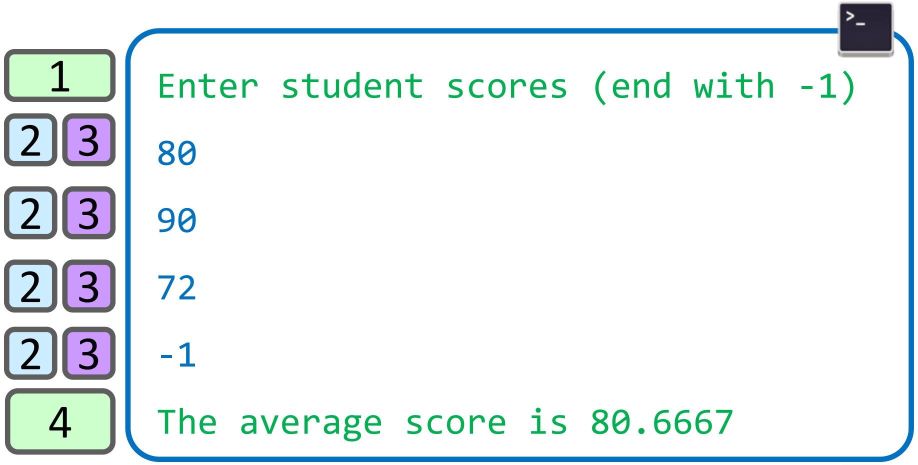


Linear (1D) Processing Programs

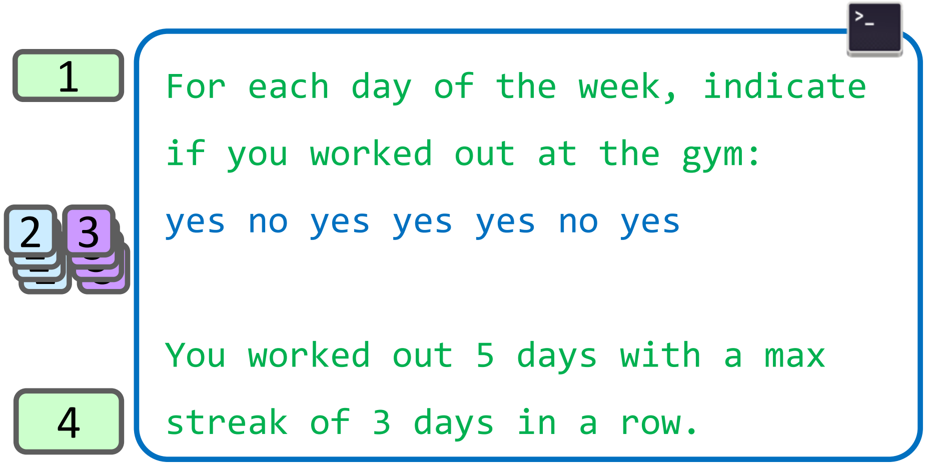


- Process an arbitrary length (or large fixed-length) sequence or set of data
- The distinguishing feature is the use of a LOOP to perform the same/similar processing repetitively on each data item
- We will likely still keep our general structure but with some sequence of those operations be repeated via the loop:
  - Prompt
  - Input
  - Process
  - Output





```
Enter student scores (end with -1)
80
90
72
-1
The average score is 80.6667
```



```
For each day of the week, indicate
if you worked out at the gym:
yes no yes yes yes no yes

You worked out 5 days with a max
streak of 3 days in a row.
```

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

Loops

- Loops are structures of code that may be repeated some number of times
- Examples:
  - Sum each student's grades (for all students in the class)
  - Search through a sequence of numbers for a particular value
  - Attend lecture ☺
- We need some condition to tell us when to stop looping, otherwise we'll repeat our code forever and never stop (a.k.a. an infinite loop)
- Several kinds of loops: 'while', 'do..while', and 'for'

Generalizing and repeating code

MOTIVATION FOR LOOPS

Motivation for Loops

- Take a simple task such as outputting the first 1000 positive integers
  - We could write 1000 cout statements
  - Yikes! We could do it but it would be painful!
- Or we could use a loop

```
#include <iostream>
using namespace std;
int main()
{
  cout << 1 << endl;
  cout << 2 << endl;
  cout << 3 << endl;
  // hundreds more cout statements

  cout << 999 << endl;
  cout << 1000 << endl;

  return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
  for(int i=1; i <= 1000; i+=1 )
  {
     cout << i << endl;
  }
  return 0;
}
```

Why We Need Loops (1)

- Suppose we are writing a program for a simple turn-based guessing game where the user must guess a secret number
- If they guess incorrectly what should we do?

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  if(guess == secretNum) {
    cout << "You got it!" << endl;
  }
  else {
    /* What should we do here? */




  }
  return 0;
}
```

Why We Need Loops (2)

- What if they guess wrong a second time? What should we do?

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  if(guess == secretNum) {
    cout << "You got it!" << endl;
  }
  else {
    cin >> guess;
    if(guess == secretNum) {
      cout << "You got it!" << endl;
    }
    else {
      /* What should we do here? */


    }
  }
  return 0;
}
```

Why We Need Loops (3)

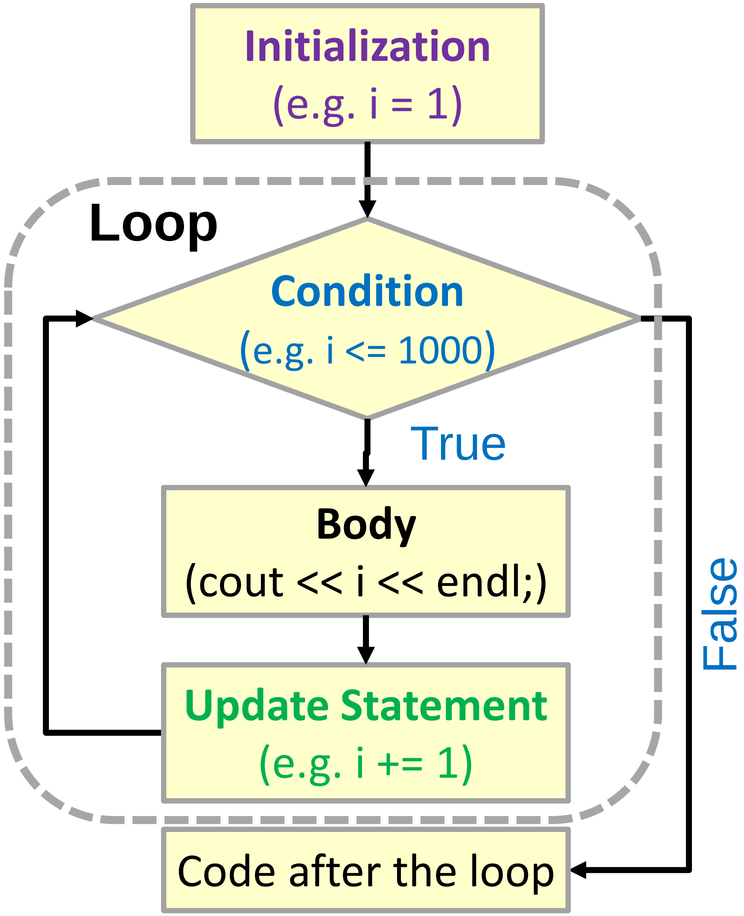
- We can never write enough `if` statements because someone might always use one more turn than we have `if` statements
- But we see there is a repetitive structure in this code
- Let's use a loop

```
#include <iostream>
using namespace std;
int main()
{
 int guess;
 int secretNum = /* some code */
 cin >> guess;
 if(guess == secretNum) {
  cout << "You got it!" << endl;
 }
 else {
  cin >> guess;
  if(guess == secretNum) {
   cout << "You got it!" << endl;
  }
  else {
   cin >> guess;
   if(guess == secretNum) {
    cout << "You got it!" << endl;
   }
   else {
    /* What should we do here? */
   }
 } }
 return 0;
}
```

4 Necessary Parts of a Loop

- Loops involve writing a task to be repeated
- Regardless of that task, there must be **4 parts** to a make a loop work
- **Initialization**
  - Initialization of the variable(s) that will control how many iterations (repetitions) the loop will executed
- **Condition**
  - Condition to decide whether to repeat the task or stop the loop
- **Body**
  - Code to repeat for each iteration
- **Update**
  - Modify the variable(s) related to the condition (without the update, the condition could be TRUE forever leading to an "infinite loop")

```
for(int i=1; i <= 1000; i+=1 ) {
    cout << i << endl;
}
```



Initialization
(e.g. i = 1)

Loop

Condition
(e.g. i <= 1000)

True

Body
(cout << i << endl;)

Update Statement
(e.g. i += 1)

False

Code after the loop

Types of Loops

- There are 2 (and a half) kinds of loops
- for loops and while (do..while) loops

```
int i;
for (i = 1; i <= 1000; i++)
{
  cout << i << endl;
}
// following statements
```

4 parts:

- Initialization
- Condition
- Body
- Update

```
int i = 1;
while (i <= 1000)
{
  // repetitive task
  cout << i << endl;
  i++;  // update
}
// following statements
```

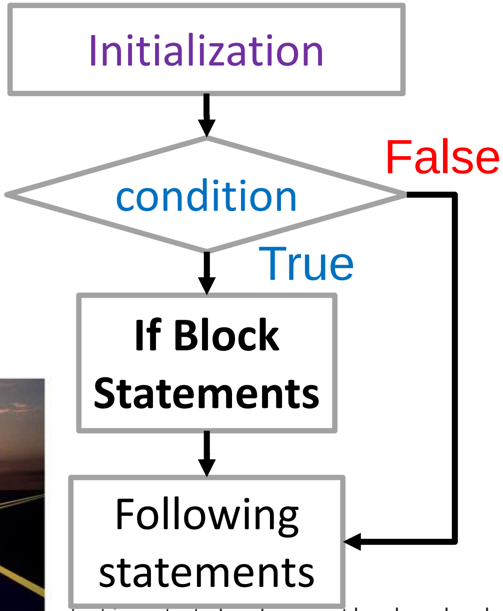
There is a variant of the while loop which is the do..while loop which we'll cover later.

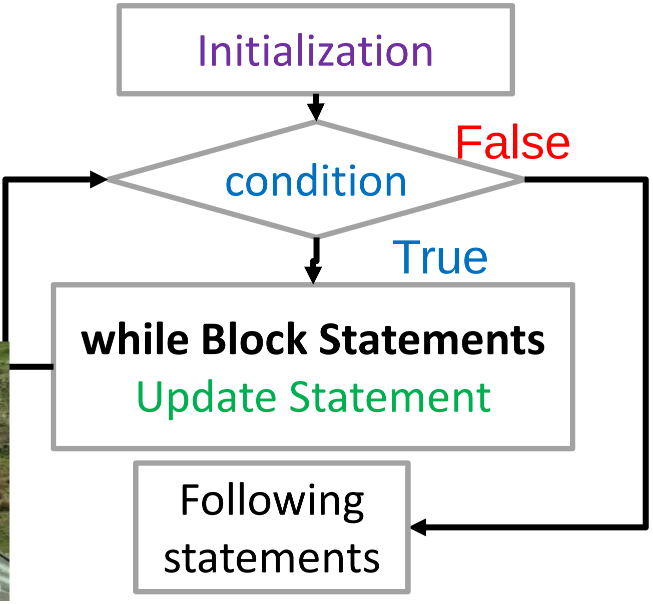
Type 1: while Loops

- A while loop is essentially a repeating 'if' statement

```
// initialization
if (condition)
{
  // executed if condition1 is true
}
// following statements
```

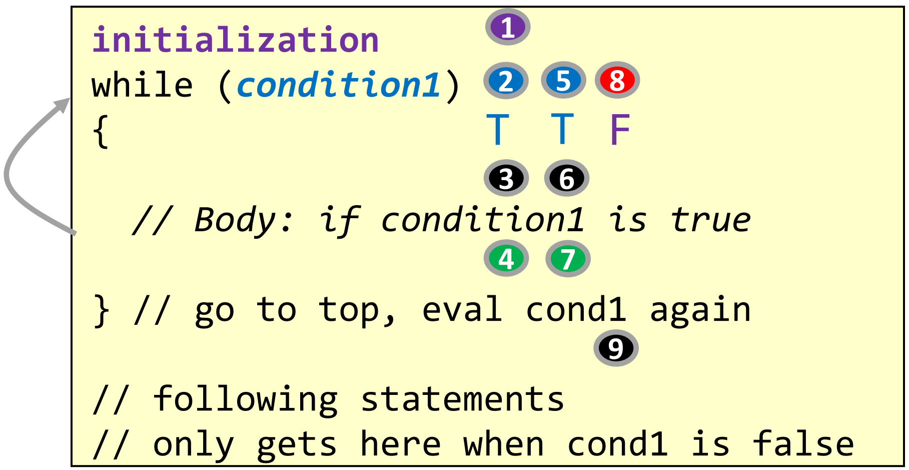
```
// initialization
while (condition)
{
  // executed if condition1 is true
  // update statement
} // go to top, eval cond1 again

// following statements
```



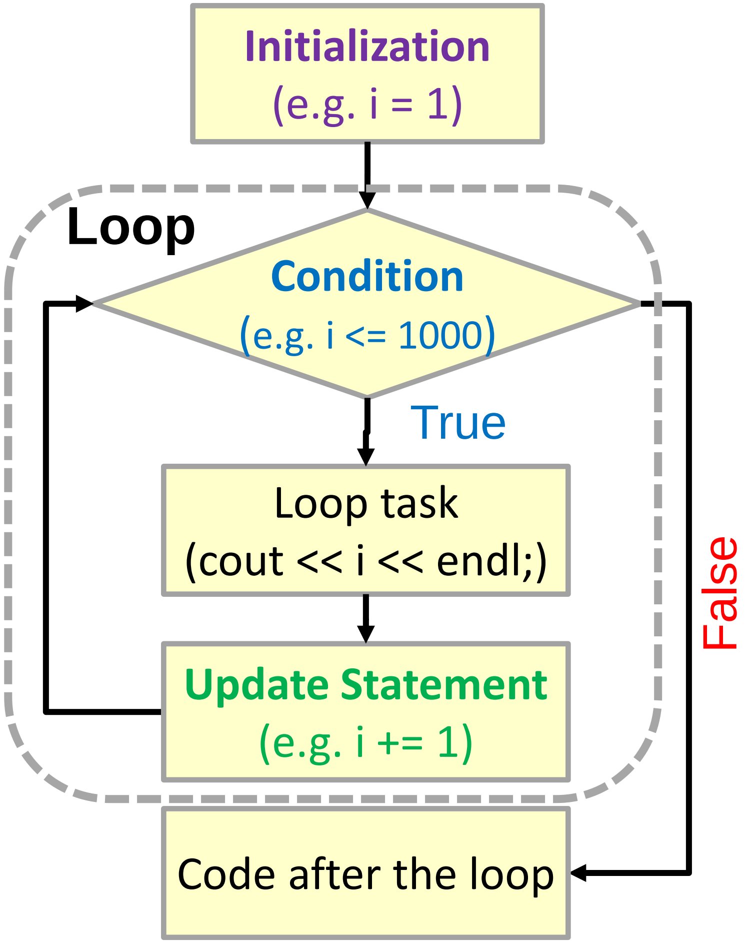


Type 1: while Loops

- A while loop is essentially a repeating 'if' statement

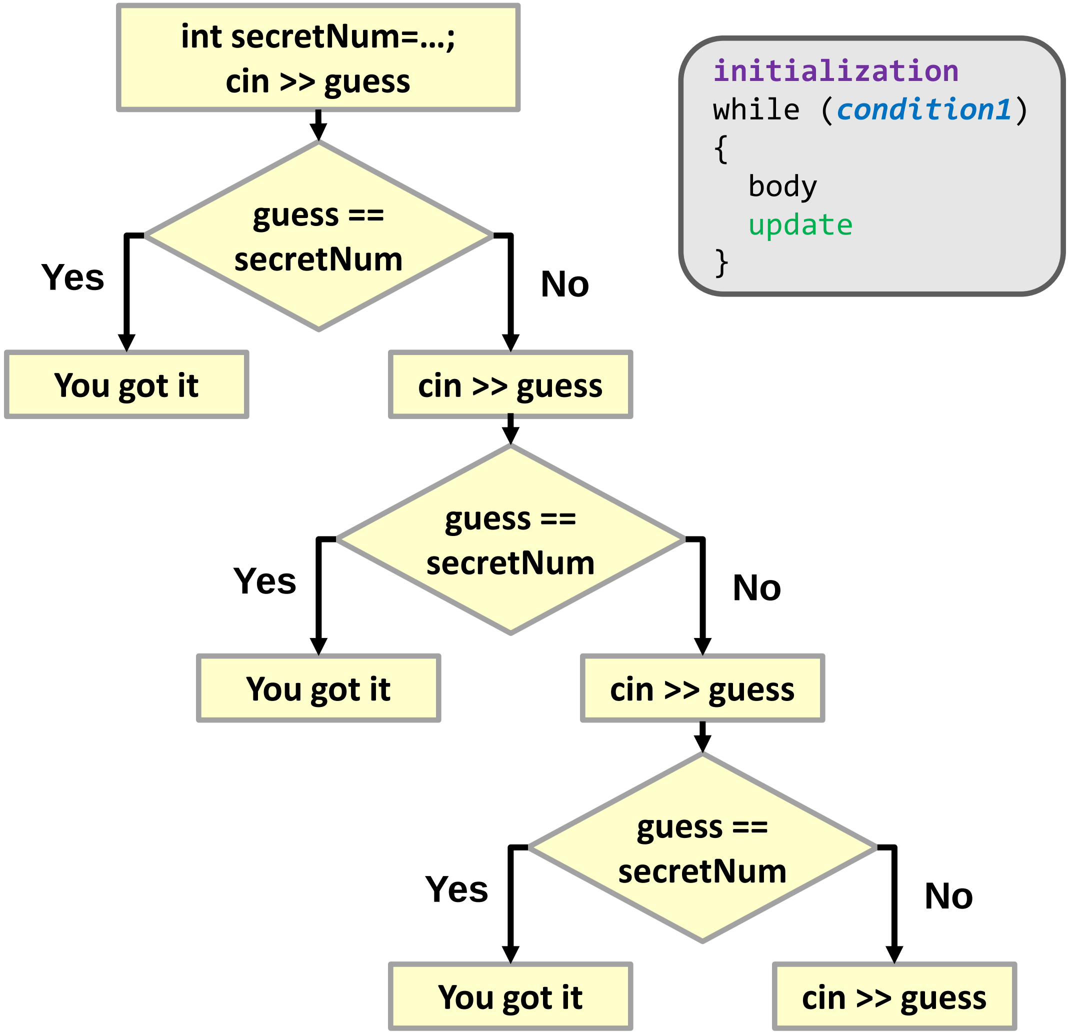


```
initialization                ①
while (condition1)            ② ⑤ ⑧
{                             T  T  F
                              ③ ⑥
   // Body: if condition1 is true
                              ④ ⑦
} // go to top, eval cond1 again
                                    ⑨
// following statements
// only gets here when cond1 is false
```



Deriving the Loop

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  if(guess == secretNum) {
    cout << "You got it!" << endl;
  }
  else {
    cin >> guess;
    if(guess == secretNum) {
      cout << "You got it!" << endl;
    }
    else {
      cin >> guess;
      if(guess == secretNum) {
        cout << "You got it!" << endl;
      }
      else {
        /* What should we do here? */
      }
  } }
  return 0;
}
```



```
initialization
while (condition1)
{
  body
  update
}
```

Applying the 4 Parts

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  cin >> guess;
  if(guess == secretNum) {
    cout << "You got it!" << endl;
  }
  else {
    cin >> guess;
    if(guess == secretNum) {
      cout << "You got it!" << endl;
    }
    else {
      cin >> guess;
      if(guess == secretNum) {
        cout << "You got it!" << endl;
      }
      else {
        /* What should we do here? */
      }
  } }
  return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */

  cin >> guess;
  while(guess != secretNum)
  {
    cout << "Wrong, guess again: " << endl;
    cin >> guess;
  }

  cout << "You got it!" << endl;
  return 0;
}
```

Always make sure you have the **4 parts**
(it's easy to forget initialization and/or update)

What Goes In a Loop Body

- What do we put in a `while` or `for` loop body?
- ANYTHING!
  - Expressions & variable assignment
  - Function calls
  - if..else statements
  - Even other loops!

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

  int secretNum = /* some code */
  cin >> guess;
  while(guess != secretNum)
  {
    cout << "Enter guess: " << endl;
    cin >> guess;
  }

  cout << "You got it!" << endl;
  return 0;
}
```

Hand Tracing (1)

- Ensure you understand the meaning (semantics) of a while loop by tracing through the code to the right
- Show all changes to x and y for:
  - x = 24
  - y = 18

```
int main()
{
  int x, y;
  cin >> x;
  while( (x % 2) == 0){
    x = x/2;
  }

  cin >> y;
  while(y > 0){
    if( y >= 10 ){
      y -= 5;
    }
    else if( y >= 5 ){
      y -= 3;
    }
    else {
      y -= 1;
    }
    cout << y << endl;
  }
  return 0;
}
```

Hand Tracing (2)

- Trace through the code and show all changes to x and y for:
  - x = 27
  - y = 6

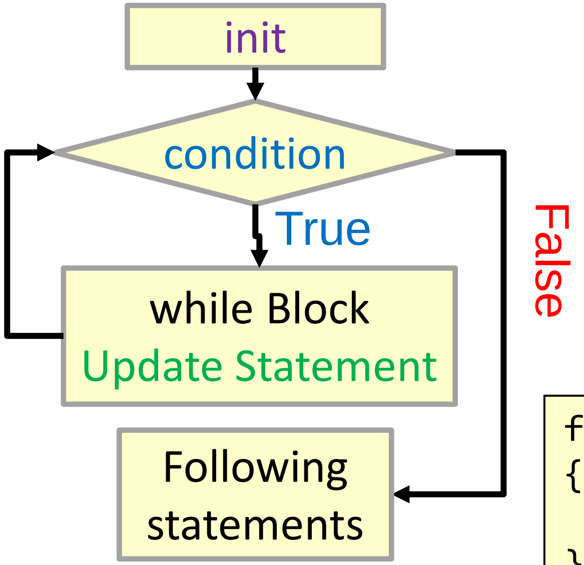
```
int main()
{
  int x, y;
  cin >> x;
  while( (x % 2) == 0){
    x = x/2;
  }

  cin >> y;
  while(y > 0){
    if( y >= 10 ){
      y -= 5;
    }
    else if( y >= 5 ){
      y -= 3;
    }
    else {
      y -= 1;
    }
    cout << y << endl;
  }
  return 0;
}
```

Type 2: for Loops

- 'for' loops have the same ability as a 'while' loop but make the 4 parts of a loop EXPLICIT

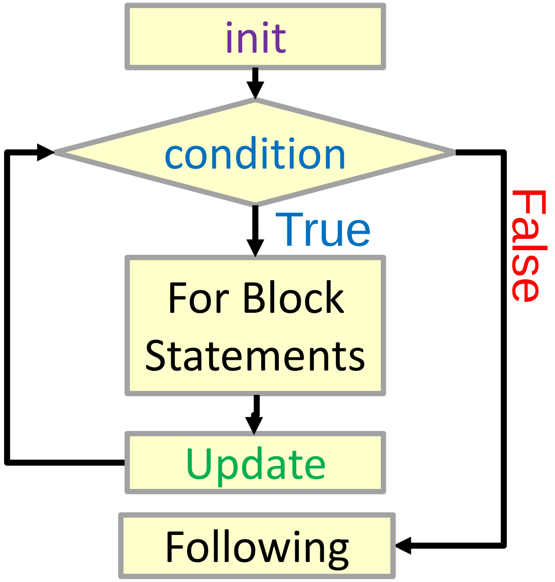
```
// initialization
while (condition)
{
  // executed if condition is true
  // Update statement
}
// following statements
```

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

// following statements
```



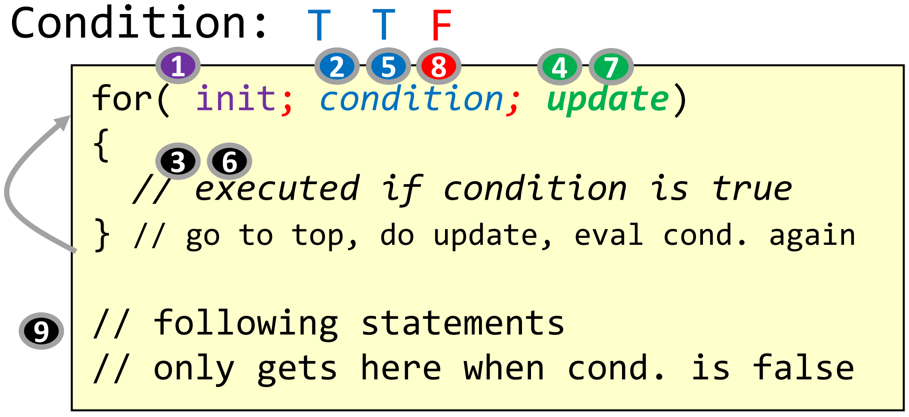
Example

```
for( int i=1; i < 1000; i++)
{
  cout << i << endl;
}
```



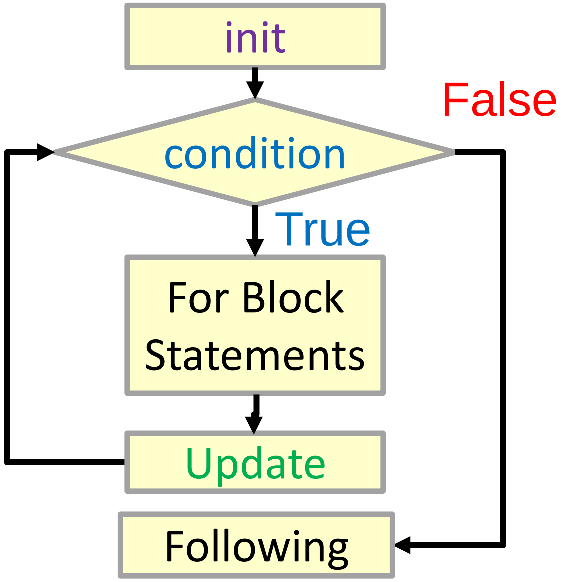
Type 2: 'for' Loop Sequencing

- 'for' loop
  - performs initialization 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!

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

void f1()
{
  // is i visible here?
  cout << i << endl;
}
```

A Last Note on Variables: Scope

- "Scope" of a variable refers to the
  - **Visibility** (who can access it) and
  - **Lifetime** of a variable (how long is the memory reserved
- For now, there are 2 scopes we will learn
  - **Global:** Variables are declared ***outside*** of any function and are visible to ***all*** the code/functions in the program
    - For various reasons, it is "bad" practice to use global variables. You MAY NOT use them in CS 102.
  - **Local:** Variables are declared ***inside*** of a function and are ***only*** visible in that function and ***die*** when the function ends

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

// Global Variable
int x=1;

int add_x()
{
  int n; // n is a "local" variable
  cin >> n;
  // y and z NOT visible (in scope) here
  // but x is since it is global
  return (n + x);
} // n dies here
int main()
{
  // y and z are "local" variables
  int y=0, z;

  z = add_x();
  y += z / x;  // n is NOT visible
  cout << x << " " << y << endl;
  return 0;
} // y and z die 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
- 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(int i=1; i <= x; i=i+y)
  {
     cout << i << endl;
     y++;
  }
  return 0;
}
```

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

Exercises

- cpp/while/blastoff
- cpp/for/blastoff

do..while Loops (1)

- while loops have a sibling known as do..while loops
- do..while loops
  - Start with keyword do
  - Followed by the body of code to be executed repeatedly in brackets { }
  - Ends with `while` condition and semicolon (;)
- do..while loops will execute the body at least once

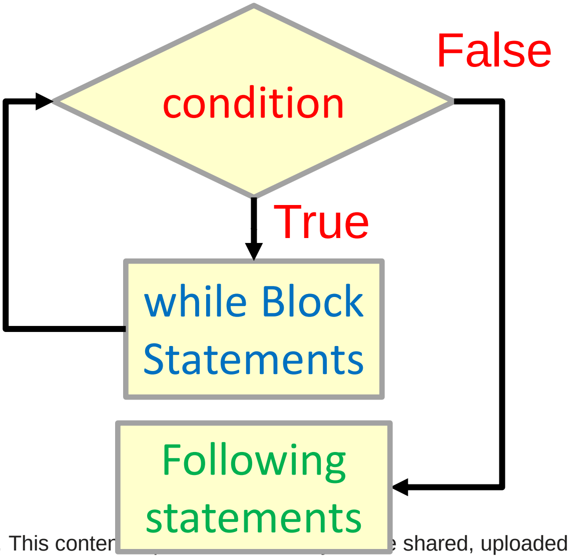
```
int main()
{
  int x, y, val;
  bool quit;

  // a while loop
  while( x < val )
  {
    /* body of code */
  }

  // a do..while loop
  do
  {
    /* body of code */
  } while( x < val );

  return 0;
}
```

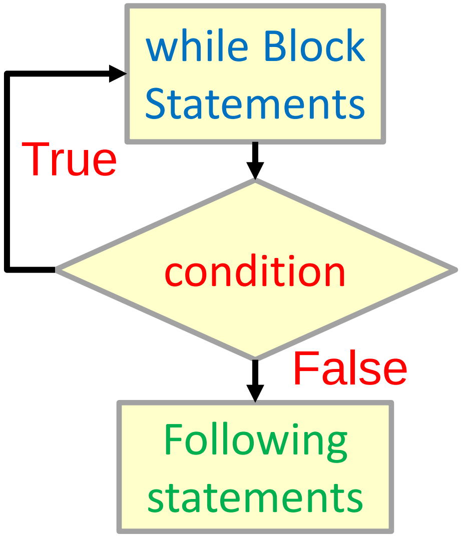
do..while Loops (2)

- do..while loops check the condition after executing at least once and repeat if the condition is true

```
while (condition)
{
  // executed if condition1 is true
} // go to top, eval cond1 again

// following statements
// only gets here when cond1 is false
```

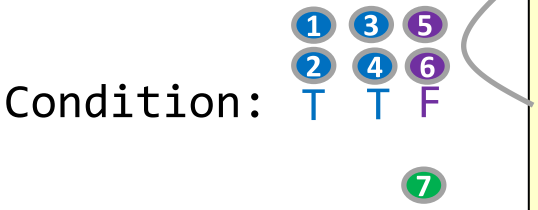
```
do
{
  // executed at least once
} while (condition);// go to 'do' (top)
                   //if cond1 evals to true
// following statements
// only gets here when cond1 is false
```



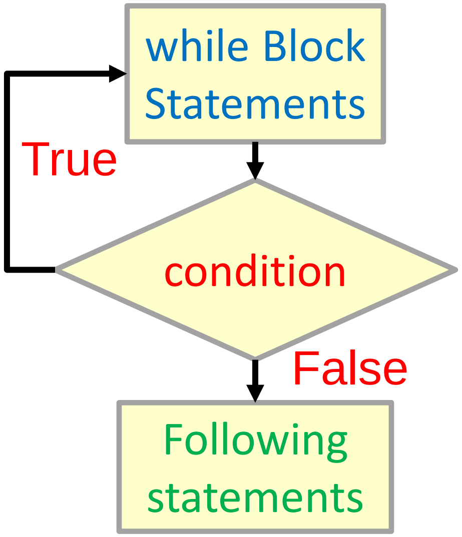


do..while Loops (3)

- do..while loops check the condition after executing at least once and repeat if the condition is true



```
do
{
  // executed at least once
} while (condition);// go to 'do' (top)
               //if cond1 evals to true
// following statements
// only gets here when cond1 is false
```



Solutions 0

```
int main()
{
  int x, y;
  cin >> x;
  while( (x % 2) == 0){
    x = x/2;
  }

  cin >> y;
  while(y > 0){
    if( y >= 10 ){
      y -= 5;
    }
    else if( y >= 5 ){
      y -= 3;
    }
    else {
      y -= 1;
    }
  }
  return 0;
}
```

Program Output for input of **24 18**:

```
X: 24, 12, 6, 3
Y: 18, 13, 8, 5, 2, 1, 0
```

Program Output for input of **27 6**:

```
X: 27
Y: 6, 3, 2, 1, 0
```

Solutions 1

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

Program Output for input of **2**:

```
-2
-1
0
1
2
```

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

Program Output for input π /2 and 2π:

```
0
1
0
-1
```

Solutions 2

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

Program Output for input of **10 2**:

```
1
4
8
```

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

Program Output for input **4 11**:

```
4 11
5 10
6 9
7 8
```