Unit 7a

'while' and 'for' Loop Syntax and Semantics
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

```cpp
#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;
}
```

```cpp
#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?

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

int main()
{
    int guess;
    int secretNum = /* some code */
    cin >> guess;
    if(guess != secretNum) {
        /* What should we do here? */
    }
    else {
        cout << "You got it!" << endl;
    }
    return 0;
}
```
Why We Need Loops (2)

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

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

- Loops involve writing a task to be repeated
- Regardless of that task, there must be **4 parts** to make a loop work
- **Initialization**
  - Initialization of the variable(s) that will control how many iterations (repetitions) the loop will execute
- **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
Types of Loops

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

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

```cpp
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

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

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

![Flowchart Diagram](image.png)
Type 1: while Loops

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

```
while (condition1) {
    // Body: if condition1 is true
    // go to top, eval cond1 again
    // following statements
    // only gets here when cond1 is false
```

Diagram:
- Initialization (e.g. i = 1)
- Condition (e.g. i <= 1000)
- True
  - Loop task (cout << i << endl;)
  - Update Statement (e.g. i += 1)
- False
Type 1: Applying the 4 Parts

- Examine the guessing game code
- When using a loop there are almost always FOUR parts:
  - Initialization
    - 1 or more variables that are part of the loop
  - Body
  - Condition
    - Condition for continuing
  - Update:
    - 1 or more variables involved in the condition (without the update, the condition could be TRUE forever leading to an "infinite loop")

```cpp
#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!
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

```cpp
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;
}
```
Hand Tracing (2)

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

```cpp
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;
}
```
Type 2: for Loops

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

```c++
// 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**

```c++
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

```
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

```cpp
#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
```

```cpp
#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

```cpp
#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

```cpp
#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
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
Exercises

- cpp/while/blastoff
- cpp/for/blastoff
- cpp/while/sum50
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

```c
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

```c
while (condition) {
    // executed if condition1 is true
    // go to top, eval cond1 again
}
// following statements
// only gets here when cond1 is false
```

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

Flowchart:
- **while** Block Statements
  - True: Repeat
  - False: Exit

- **do**
  - True: Repeat
  - False: Exit

Flowchart:
- **condition**
  - True: Repeat
  - False: Exit

Flowchart:
- **while Block Statements**
  - True: Repeat
  - False: Exit
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
```