Unit 7b

Using Loops to Solve Problems
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
CHOOSING THE TYPE OF LOOP
When Do I Use a While Loop (1)

- When you don't know in advance how many times something should repeat?
  - How many guesses will the user need before they get it right?

```cpp
#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;
}
```
When Do I Use a While Loop (2)

- Whenever you see, hear, or use the word 'until' in a description
- Important Tip:
  - "until" = "while not"
  - Saying "keep guessing until you are correct" is the same as "keep guessing while you are not correct"

```cpp
#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;
}
```
When Do I Use a For Loop (1)

• When you know **in advance (before the loop starts)** how many times to iterate
  – Usually a constant or variable that has been calculated or input from the user

```cpp
// Program to output numbers // 1 through n
#include <iostream>
using namespace std;
int main()
{
    int n;
    cin >> n;
    for(int i=1; i < n; i++)
    {
        cout << i << endl;
    }
    return 0;
}
```
Exercises 1

• Warm up with the following exercises:

• cpp/while/sum50
• cpp/while/dowhilen
PROBLEMS SOLVING IDIOMS

Map, Reduce, Selection
Map Idiom

- **Name**: Map
- **Description**: Convert (map) each value in a collection to another value
- **Structure**: Use a loop to process a series of input value and convert to the desired output value
  - Usually with a n-to-n input-output relationship
- **Example(s)**:
  - See examples on the right

```java
for(/* loop thru each input */) {
    // Get next input, x
    // Produce next output, f(x)
}
```

**Structure**

**Output the first $n$ odd integers**

| Input: 0, 1, 2, ..., n-1 | Output: 1, 3, 5, ..., 2(n-1)+1 |

**Given a threshold of 70, indicate if students have passed a quiz**

| Input: 78, 61, 85, 93, 54 | Output: T, F, T, T, F |

**Take the absolute value of each input**

| Input: -18, -13, 36, 2, -21 | Output: 18, 13, 36, 2, 21 |
Map Idiom Examples (2)

Output the first $n$ odd integers
Input:  $0, 1, 2, \ldots, n-1$
Output:  $1, 3, 5, \ldots, 2(n-1)+1$

Given a threshold of 70, indicate if students have passed a quiz
Input:  $78, 61, 85, 93, 54, -1$
Output:  $P, \ NP, \ P, \ P, \ NP$

Take the absolute value of each input
Input:  $-18, -13, 36, 2, -21$
Output:  $18, 13, 36, 2, 21$
More Map Examples

• Write a loop to generate the first $n$ positive, odd numbers
  – Odd numbers: 1, 3, 5, 7, 9
• We could use two separate variables
  – An inductive/control variable to count to $n$ and control how many repetitions
  – Another to produce the odd values
• It is more common to put the desired value in terms of the inductive/control variable
• If $i$ ranges from 0 to $n-1$, then the first $n$ odd numbers are generated by:
  – $2*i + 1$
• Tip: Write a table of $i$ and the desired value and try to see if a simply line ($y = mx+b$) can fit the data

```cpp
int n;
cin >> n;
int odd = 1;
for(int i=0; i < n; i++)
{
    cout << odd << endl;
    odd += 2;
}
```

Method 1: Generate the first $n$ positive, odd numbers

```cpp
int n;
cin >> n;
for(int i=0; i < n; i++)
{
    cout << __________ << endl;
}
```

Method 2: Generate the first $n$ positive, odd numbers
Exercise 2a

• Write a for loop to output all the elements of the specified sequences
  – Try to put your expressions in terms of the inductive variable, i

\begin{align*}
\{3, 7, 11, 15, 19, 23, 27, 31\} \\
&\text{for(int } i=0; i < 8; i++) \\
&\quad \{ \\
&\quad \quad \text{cout } \ll \text{______ } \ll \text{ endl; } \\
&\quad \} \\
\{1, 9, 2, 8, 3, 7, 4, 6, 5, 5\} \\
&\text{for(int } i=\_\_\_; i \leq \_\_\_; i++) \\
&\quad \{ \\
&\quad \quad \text{cout } \ll i \ll \text{ endl; } \\
&\quad \quad \text{cout } \ll \text{_______ } \ll \text{ endl;} \\
&\quad \} \\
\end{align*}
Exercise 2b

- Write a loop to generate and output this sequence:
  - 0,0,1,1,2,2,3,3,4,4
  - Trying doing so using only the inductive variable

```cpp
for( int i=___; ________; ____)
{
    cout << _________ << endl;
}
```
Reduce Idiom

- **Name**: Reduce / Combine / Aggregate
- **Description**: Combine/reduce all elements of a collection to a single value
- **Structure**: Use a "reduction" variable and a loop to process a series of input values, combining each of them to form a single (or constant number of) output value in the reduction variable
  - An n-to-1 input-output relationship
- **Example(s)**:
  - See example on the right

```
// Declare reduction variable, r
// Set r to identity value
for(/* loop thru each input */)
{
    // Get next input, x
    // Update r using x
}
```

**Average a series of 4 numbers**

Input: 2, 3, 1, 8
Average: 3.5

```cpp
double sum = 0;
double x;
for(int i=0; i < 4; i++)
{
    cin >> x;
    sum += x;
}

cout << sum / 4.0 << endl;
```
Map / Reduce Example: Series Approximations

• Many interesting **real-valued** functions or constants may be **approximated** as a rational number using a **series summation or product**

\[ e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots \]

• Series are best generated using loops where **each iteration generates one term** (i.e. **map**) and combines it with the **previous terms** (by adding or multiplying as necessary, i.e. **reduce**)

Reduce Example: Factorials

• Write a loop to compute n! (factorial)
  – \( n! = 1 \cdot 2 \cdot \cdots \cdot (n - 1) \cdot n = \prod_{i=1}^{n} i \)
  – 0! is defined to just be 1
    • We would not want to multiply by 0 since any further multiplication would result in 0 as well

```cpp
int n;
cin >> n;
int fact = ___;
for( int i=1; i <= n; i++)
{
    ________________;
}
```
Map + Reduce: Calculating $e^x$

- Write a loop to generate the first $n$ terms of the approximation of $e^x$
  \[ e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots \]

- Tips:
  - Generalize: Look at the pattern and write out the expression for the $i$-th term
  - Since $0!$ is a bit strange and just defined to be 1, pull out the first term and let the loop calculate the remaining terms
  - The first time around you can use the `pow(base, exp)` function; then try to see how you'd do it without using `pow()`
  - Keep a variable for $i!$ updating it each iteration to be ready for the next

```cpp
double x, e_x = ____;
int n, fact = 1;

cin >> x >> n;
for( int i=___; ______; ____)
{
    fact ______________;
    e_x ______________________;
}
```

**Attempt 1**

```cpp
double x, e_x = ____ , x_i = ____;
int n, fact = 1;

cin >> x >> n;
for( int i=___; ______; ____)
{
    x_i ______________________
    fact ______________;
    e_x ______________________;
}
```

**Attempt 2**
Selection Idiom

- **Name**: Selection
- **Description**: Select a subset (possibly one or none) of elements from a collection based on a particular property
- **Structure**: Loop through each element and check whether it meets the desired property. If so, perform a `map`, `reduce`, or other *other update* operation.
- **Example(s)**:
  - Count all *positive* integers inputs

```java
// declare/initialize any state variables
// needed to track the desired result

// loop through each instance
for( /* each input, i */ ) {
    // Check if input meets the property
    if(property is true for i) {
        // Update state (variables) as needed
    }
}

// Output the state variables
```

**Count Positive Integers**

- **Input**: 2, -3, -1, 8
- **Output**: 2
Selection Idiom Examples

- **Example 1:** Count how many negative numbers are input (stopping for input 0)

```cpp
int x, neg_cnt = 0;
cin >> x;
while(x != 0)
    { if(x < 0) { neg_cnt += 1; } 
        cin >> x;
    }
cout << neg_cnt << endl;
```

- **Example 2:** Find the largest number of 50 positive integer input values

```cpp
int x, max = -1, neg_cnt = 0;
for(int i=0; i < 50; i++)
    { cin >> x;
        if(x > max) { max = x; }
    }
cout << max << endl;
```
Exercises 3

• For each of the following exercises, think about the problem and identify which idioms can be used to solve the problem

• cpp/while/goldilocks
• cpp/for/interest
• cpp/for/sum-mult-2-5
Exercises 4

• For each of the following exercise, talk about the problem and identify which idioms can be used to solve the problem

• cpp/for/polydeg
• cpp/while/turn360
**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.

```cpp
/* 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;
}
```
Side Topic: Pre-/Post- Increment/Decrement

- Recall the increment and decrement operators: ++ and --
  - If ++ comes before a variable it is called 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]
Common 'while' Loop Mistakes

• Failing to update the variables that affect the condition
• Assignment rather than equality check
• Off-by-one error
• Often leads to infinite loops
  – When you run your program it will not stop
  – Use Ctrl+c to force quit it

```cpp
int i=0, n=10;
while (i < n)
{
    cout << "Iteration " << i << endl;
    // Oops, forgot to change i
}
cout << "Done" << endl;
```

```cpp
int i=0, n=5;
while (i = n) // oops, meant i==n
{
    cin >> i;
}
cout << "Done" << endl;
```

```cpp
int i=0;
// want to print "Hi" 5 times
while (i <= 5) // oops, meant i < n
{
    cout << "Hi" << endl;
    i++;
}
```
Common 'for' Loop Mistakes

• Updating the inductive variable in the wrong direction

• Off by one error

• Missing the exit condition

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

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

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

• A Boolean flag
  – Two values: true or false
  – Pattern: Initialize to a value that will cause the while loop to be true the first time and then check for the ending condition in an if statement and update the flag
  – Up to you to determine the meaning of the flag (e.g. done or again)

```cpp
int guess, secretNum;
bool done = false;
while (!done)
{
    cin >> guess;
    if(guess == secretNum) {
        done = true;
    }
}
cout << "You got it!" << endl;

int guess, secretNum;
bool again = true;
while (again)
{
    cin >> guess;
    if(guess == secretNum) {
        again = false;
    }
}
cout << "You got it!" << endl;
```
What Goes In an while Condition

• What do we put in a `while` condition?

  – The compiler will interpret what is in the parentheses as a Boolean
    • 0 = false
    • Non-0 = true

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

  // Uses Boolean result of comparison
  while( x > 0 )    { /* code */ }

  // Uses value of bool variable.
  //   Executes if done == false.
  while( !done )     { /* code */ }

  // Interprets number as a bool
  //   Executes if val is non-zero
  while( val )      { /* code */ }

  // Interprets return value as bool
  //   Executes if the min is non-zero
  while( min(x,y) ) { /* code */ }

  return 0;
}
```
'while' or 'for'

While Loops

• Usually used to repeat code until some condition is false

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)

Output each input until -1 is entered

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

Sum 5 input values

```cpp
int sum = 0, val = 0;
/* how many iterations required */
for(int i=0; i < 5; i++)
{
    cin >> val;
    sum += val;
}
```
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;
```
When Should I Use do..while

• We generally prefer while loops
• We can use do..while loops when we know we want to execute the code at least one time (and then check at the end)
• Even then...
  – See next slide
Converting do..while to while Loops

We need to get one guess at least and then determine if we should repeat. This seems a natural fit for the do..while structure but we can easily mimic this behavior with a normal while loop.

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

cin >> guess;
while (guess != secretnum)
{
    cin >> guess;
} // go to top, eval cond1 again
cout << "You got it!" << endl;

guess = secretnum + 1;
while (guess != secretnum)
{
    cin >> guess;
} // go to top, eval cond1 again
cout << "You got it!" << endl;

We can duplicate the body of the loop once before we start the loop.

We can set our variables to ensure the while condition is true the first time.
Exercises 5

• cpp/for/rps-bestof3
• Write a for loop to generate all the elements of the specified sets

\[ S = \{3, 7, 11, 15, 19, 23, 27, 31\} \]

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

\[ \{1, 9, 2, 8, 3, 7, 4, 6, 5, 5\} \]

```cpp
for(int i=1; i <= 5; i++)
{
    cout << i << endl;
    cout << 10-i << endl;
}
```
Exercise 2b Solutions

- Write a loop to generate and output this sequence:
  - 0,0,1,1,2,2,3,3,4,4
  - Trying doing so using only the inductive variable

```cpp
for (int i=0; i < 10; i++)
{
    cout << i/2 << endl;
}
```