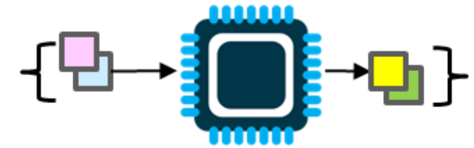
Unit 2b – Coding with Loops and Loop Idioms

Mark Redekopp

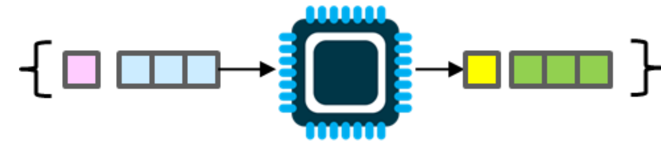
© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Unit 2

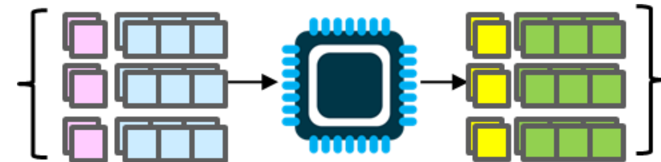
- **Unit 1**: Scalar processing
  - aka IPO=Input-Process-Output Programs



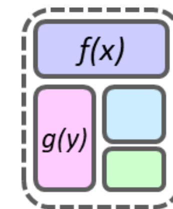
- **Unit 2**: Linear (1D) Processing



- **Unit 3**: Multidimensional Processing



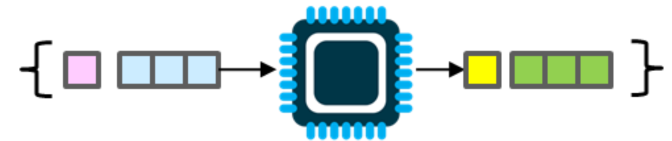
- **Unit 4**: Divide & Conquer (Functional Decomposition)



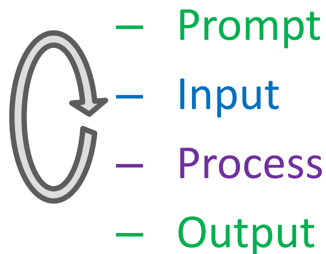
© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

This Photo by Unknown Author is licensed under CC BY-NC

Linear (1D) Processing Programs



- Process an arbitrary length sequence or set of data (rather than a fixed amount)
- 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



1

2 3

2 3

2 3

2 3

4

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

1

2 3

4

```
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.
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

CHOOSING THE TYPE OF LOOP

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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?

```
#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;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

When Do I Use a While Loop (2)

- Whenever you see, hear, or use the word 'until' in a description
- Important Tip:
  - "until x" = "while not x"
    - until(x)⇔while(!x)
  - Ex: "Keep guessing **until** you are correct" is the same as "keep guessing **while** you are **NOT** correct"

```
#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;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Practice: Until to While Not

- Rephrase the following statements using while
  - I run `until` I'm tired.
  - I work `until` 5 p.m. or I'm done.
  - I study `until` I get a good grade and understand the material.

Note: In logic, DeMorgan's Theorem tell us:

- `!(x || y)` ⇔ `!x && !y`
- `!(x && y)` ⇔ `!x || !y`

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

When Do I Use a For Loop (1)

- When you **DO 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

```
// 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;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

for Loop Example

- Suppose we change our guessing game to limit the user to 10 guesses.
- A for loop to repeat the process 10 times seems appropriate
- But do we always want to iterate 10 time?
- Under what conditions do we want to print "You lose!"

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  for(i=0; i < 10; i++)
  {
    cout << "Enter guess: " << endl;
    cin >> guess;

    if(guess == secretNum){

      cout << "You win!" << endl;
      // what should we do now?
    }

    // Should we print "You lose!" here?
  }
  // Or here? And under what condition?
  cout << "You lose!" << endl;

  return 0;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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); otherwise the loop would only iterate once

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  for(i=0; i < 10; i++)
  {
    cout << "Enter guess: " << endl;
    cin >> guess;
    if(guess == secretNum){
      cout << "You win!" << endl;
      break;
    }
  }
  // Should we always print this?
  cout << "You lose!" << endl;

  return 0;
}
```

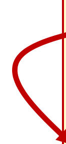
© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Multiple Ways to Exit

- When we break we immediately leave the loop and resume execution at the code AFTER the loop.
- But sometimes we need to know WHY the loop terminated…
  - Was it because we executed a break?
  - Or was it because the loop reached its terminating condition?
- Need to use some variable (a bool often can be useful here) to record how we left the loop

```
#include <iostream>
using namespace std;
int main()
{
  int guess;
  int secretNum = /* some code */
  bool won = false;
  for(i=0; i < 10; i++)
  {
    cout << "Enter guess: " << endl;
    cin >> guess;
    if(guess == secretNum){
      cout << "You win!" << endl;
      won = true;
      break;
    }
  }
  if(won == false) // same as if(!won)
   {  cout << "You lose!" << endl; }

  return 0;
}
```



© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

We Can Use A While Loop

- We can always interchange `while` and `for` loops
- Neither type is more powerful, but sometimes one is more intuitive than the other.
- Take some time and trace this code for yourself to understand how it works

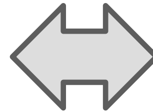
```
#include <iostream>
using namespace std;
int main()
{
  int secretNum = /* some code */
  int guess = secretNum-1, i = 0;
  while(guess != secretNum && i < 10)
  {
    cout << "Enter guess: " << endl;
    cin >> guess;
    i++;
  }
  if(guess == secretNum) {
    cout << "You win!" << endl;
  }
  else {
    cout << "You lose!" << endl;
  }
  return 0;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Converting while to for Loops

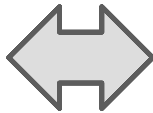
- While and for loops are **EQUALLY** expressive (i.e. what you can do with one, you can ALWAYS achieve with the other).
- Simply pick whichever makes the most sense to you!

```
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;
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

'while' or 'for'

While Loops

- Usually used to repeat code until some condition is false

UNTIL ⇔ WHILE not

For Loops

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

Output each input until -1 is entered

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

Sum 5 input values

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

© 20[illegible] by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Map, Reduce, Selection

PROBLEMS SOLVING IDIOMS

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

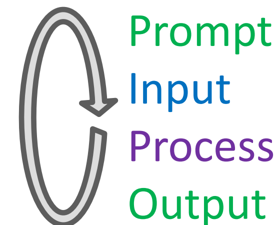
Map Idiom

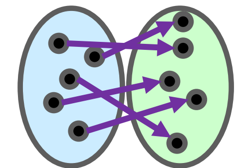
- **Name**: Map
  - Defines a **many-to-many** input-output relationship
- **Description**: Process / transform / convert (aka map) each value in a collection to another value
- **Structure**: Use a loop to process a series of input values and convert each to the desired output value
- **Example(s)**:
  - See example on the right

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

```
Input:   78, 61, 85, 93, 54
Output:  P,  NP,  P,  P, NP
```

```
for(/* loop N times */)
{
   // Get next input, x
   // Transform to f(x)
   // Output f(x)
}
```





Structure: (Prompt), Input, Process, Output are repeated each iteration

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Map Idiom Examples (2)

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
```

```
int score = 0;
cin >> score;
while ( score != -1) {
   if(score >= 70) {
     cout << "P" << endl;
   }
   else { cout << "NP" << endl; }
   cin >> score;
}
```

Output the first *n* odd integers

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

```
for( int i=0; i < n; i++ ) {
   // i itself is the input
   cout << 2*i + 1 << endl;
}
```

Take the absolute value of each input

```
Input:  -18, -13, 36, 2, -21
Output:  18,  13, 36, 2,  21
```

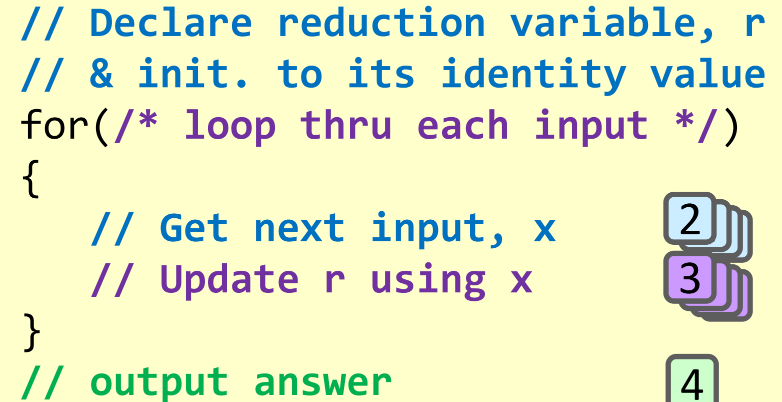
```
int val;
for( int i=0; i < n; i++ ) {
   cin >> val;
   if(val < 0) {
     val = -val;
   }
   cout << val << endl;
}
```

Note: In example 2 and 3, assume n is initialized earlier in the code.

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Reduce Idiom

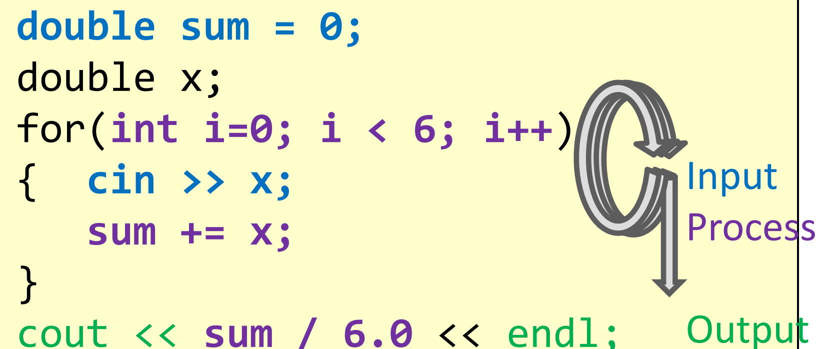
- **Name**: Reduce / Combine / Aggregate
  - A many-to-1 input-output relationship
- **Description**: Combine/reduce all inputs of a collection to a single value
- **Structure**: Use a "reduction" variable and a loop to process a series of input values, combining each of them to form a single (or constant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



Structure

Average a series of 6 numbers

Input: 2, 3, 1, 8, 4, 3
Average: 3.5



© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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

```
// 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
```

Structure

Count Positive Integers

```
Input:   5, -3, -1, 8
Output:  2
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Selection Idiom Examples

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

```
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

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

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Exercise Set 1

- For each of the following exercises, think about the problem and identify which idioms can be used to solve the problem
  - goldilocks
  - Interest
  - sum50
  - sum-mult-2-5

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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]
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

MORE MAP AND REDUCE EXAMPLES (GENERALIZING PATTERNS)

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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, i
- *Tip: Write a table of i and the desired value and try to see if a simple line (y = mx+b) can fit the data*

```
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

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

Method 2: Generate the first n positive, odd numbers

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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

$\{3, 7, 11, 15, 19, 23, 27, 31\}$

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

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

```
for(int i=___; i <= ___; i++)
{
   cout << i << endl;

   cout << _______ << endl;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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

```
for( int i=___; _________; _____)
{
  cout << _________ << endl;

}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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.g. π, e^x, etc.)**

  - $e^x = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots$

- 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**)

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Reduce Exercise 3a: Factorials

- Write a loop to compute n! (factorial)
  - $n! = 1 * 2 * \cdots * (n-1) * n = \prod_{i=1}^{n} i$
  - 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

```
int n;
cin >> n;
int fact = ____;
for( int i=1; i <= n; i++)
{
   ________________________;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Exercise 3b: Calculating e^x

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

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

cin >> x >> n;
for( int i=___; _________; _____)
{
   fact _________________;
   e_x __________________________;
}
```

Attempt 1

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

cin >> x >> n;
for( int i=___; _________; _____)
{
   x_i _________________________
   fact _________________;
   e_x __________________________;
}
```

Attempt 2

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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**

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

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

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

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Common 'for' 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

```
// Goal: 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;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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)

```
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;
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Exercises 4

- For each of the following exercise, talk about the problem and identify which idioms can be used to solve the problem
  - polydeg
  - turn360

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Non-Comparison Conditions

- If the expression in the `if`, `while`, or `for` loop does not result in a Boolean, it will try to convert the expression to a Boolean
  - 0 = false
  - Non-0 = true

```
int main()
{
  int x, y, val;
  bool done;
  cin >> x >> y >> val >> 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;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

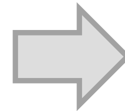
When Should I Use do..while

- 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

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Converting do..while to while Loops

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



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.

```
cin >> guess;
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.

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

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

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Exercises 5

- cpp/for/rps-bestof3

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Exercise 2a Solutions

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

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

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

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

```
for(int i=1; i <= 5; i++)
{
  cout << i << endl;

  cout << 10-i << endl;
}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

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

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

}
```

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.

Exercise 3b: Calculating e^x

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

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

cin >> x >> n;
for( int i=1; i < n; i++)
{
   fact *= i;
   e_x += pow(x,i)/fact;
}
```

Attempt 1

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

cin >> x >> n;
for( int i=1; i < n; i++)
{
   x_i *= x;
   fact *= i;
   e_x += x_i / fact;
}
```

Attempt 2

© 2023 by Mark Redekopp. This content is protected and may not be shared, uploaded, or distributed.