CS 103 Lecture 3 Slides

Control Structures

Mark Redekopp

Announcements

- Lab 2 – Due Friday
- HW 2 – Due next Thursday

Review

- Write a program to ask the user to enter two integers representing hours then minutes. Output the equivalent number of seconds.
- To get started…
  - Go to http://bits.usc.edu/cs103/in-class-exercises
    - printseconds
  - We've started the program for you…look at the
    - General template for a program with the #includes, using namespace std; and int main() function which returns 0
  - We've declared variables where you can store the input and computation results
  - Now you add code to
    - Get input from the user
    - And compute the answer and place it in the 'sec' variable

CONTROL STRUCTURES

Comparison/Logical Operators

- Loops & conditional statements require a condition to be evaluated resulting in a True or False determination.
- In C/C++…
  - 0 means False
  - Non-Zero means True
  - `bool` type available in C++ => ‘`true`’ and ‘`false`’ keywords can be used but internally ‘true’ = non-zero (usually 1) and ‘false’ = 0
- Example 1

```
int x = 100;
while(x)
  { x--;}
```

- Usually conditions results from comparisons

```
==, !=, >, <, >=, <=
```

Logical AND, OR, NOT

- Often want to combine several conditions to make a decision
- Logical AND => expr_a && expr_b
- Logical OR => expr_a || expr_b
- Logical NOT => ! expr_a
- Precedence (order of ops.) => ! then && then ||
  - !x || y && !z
  - ((!x) || (y && (!z)))
- Write a condition that eats a sandwich if it has neither tomato nor lettuce
  - if (!tomtato && !lettuce) { eat_sandwich(); }
  - if (!(tomato || lettuce)) { eat_sandwich(); }

A	B	AND
False	False	False
False	True	False
True	False	False
True	True	True

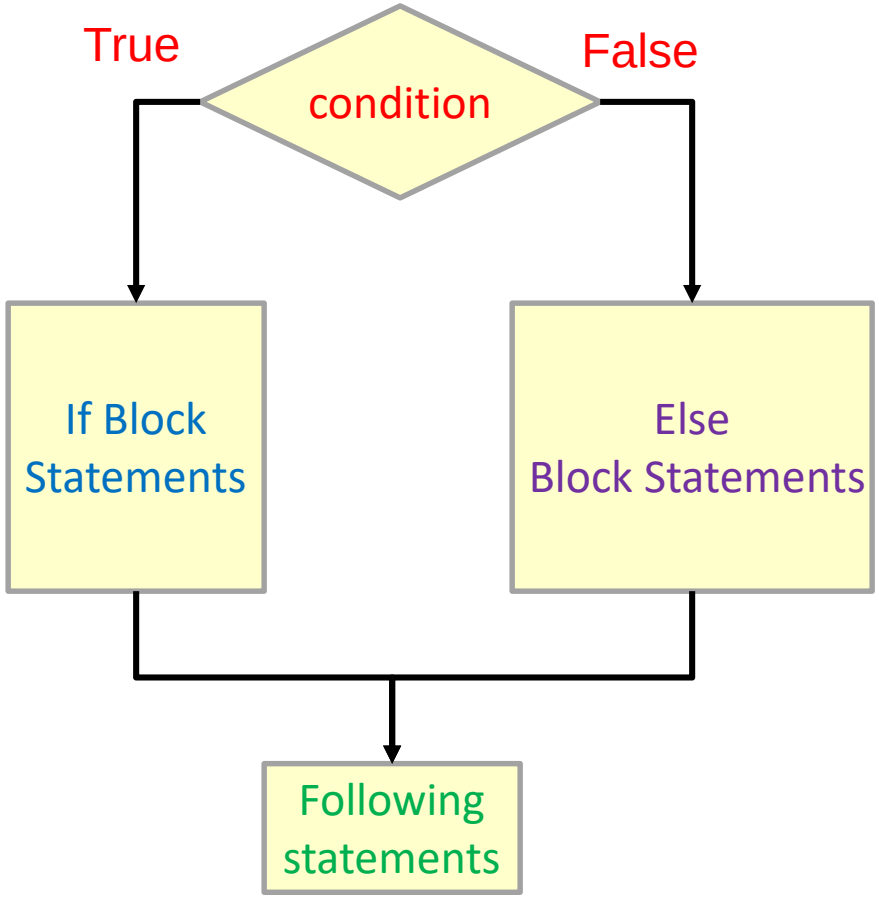
A	B	OR
False	False	False
False	True	True
True	False	True
True	True	True

A	NOT
False	True
True	False

Exercise

- Which of the following is NOT a condition to check if the integer x is in the range [-1 to 5]
  - `x >= -1 && x <= 5`
  - `-1 <= x <= 5`
  - `! ( x < -1 || x > 5)`
  - `x > -2 && x < 6`
- Consider `( !x || (y && !z ) )`
  If x=100, y= -3, z=0 then this expression is…
  - true
  - false

If..Else Flow Chart



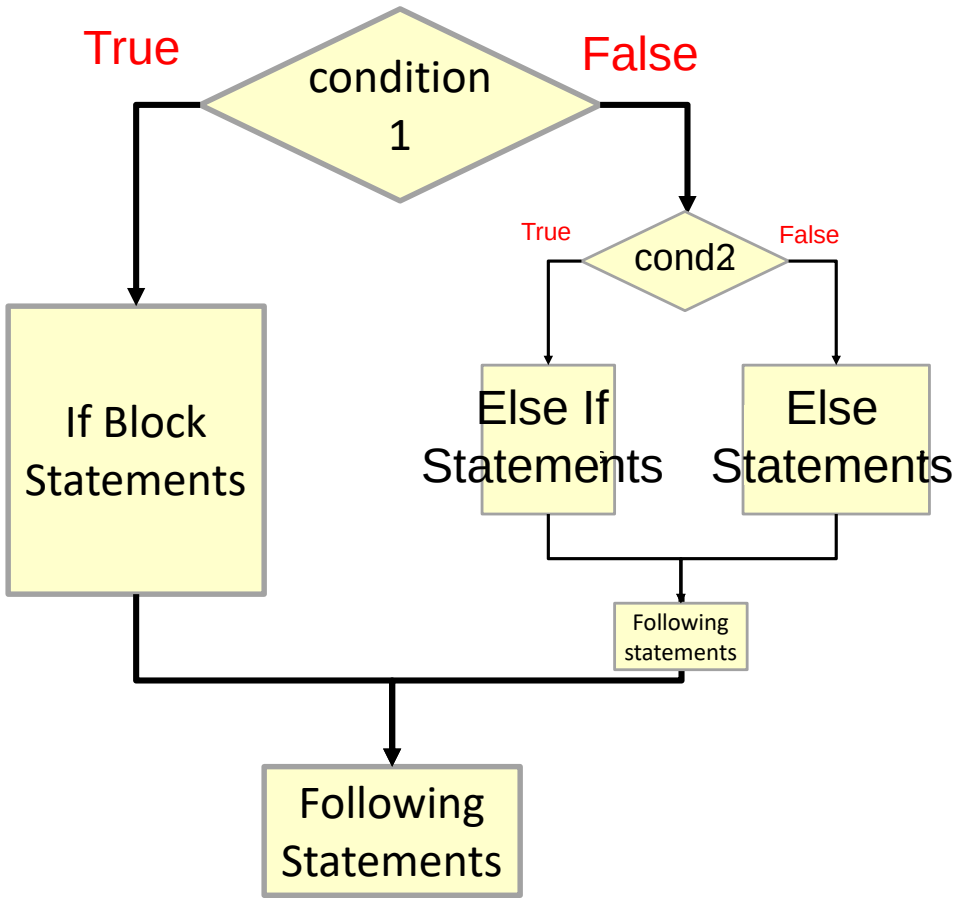

```
if (condition1)
{
  // executed if condition1 is true
}
else
{
  // executed if condition1
  // above is false
}

// following statements
```

If…Else If…Else

- Use to execute only certain portions of code
- Else If is ***optional***
  - Can have any number of else if statements
- Else is ***optional***
- { … } indicate code associated with the if, else if, else block

```
if (condition1)
{
  // executed if condition1 is true
}
else if (condition2)
{
  // executed if condition2 is true
  //  but condition1 was false
}
else if (condition3)
{
  // executed if condition3 is true
  //  but condition1 and condition2
  //  were false
}
else
{
  // executed if neither condition
  // above is true
}
```

Flow Chart



These 2 are equivalent

```
if (condition1)
{
  // executed if condition1 is True
}
else if (condition2)
{
  // executed if condition2 is True
  //  but condition1 was False
}
else
{
  // executed if neither condition
  // above is True
}
```

```
if (condition1)
{
  // executed if condition1 is True
}
else
{
  if (condition2){
    // executed if condition2 is True
    //  but condition1 was False
  }
  else
  {
    // executed if neither condition
    // above is True
  }
}
```

Single Statement Bodies

- An if or else construct with a single statement body does not require { … }
- Another if counts as a single statement

```
if (x == 5)
  y += 2;
else
  y -= 3;


if (x == 5)
  y += 2;
else
  if(x < 5)
    y = 6;
  else
    y = 0;
```

The Right Style

- Is there a difference between the following two code snippets
- Both are equivalent but the bottom is preferred because it makes clear to other programmers that only one or the other case will execute

```
int x;
cin >> x;

if( x >= 0 ) { cout << "Positive"; }
if( x < 0  ) { cout << "Negative"; }
```

```
int x;
cin >> x;

if( x >= 0 ) { cout << "Positive"; }
else         { cout << "Negative"; }
```

Find the bug

- What's the problem in this code…

```
// What’s the problem below
int x;
cin << x;
if (x = 1)
  { cout << "X is 1" << endl;}
else
  { cout << "X is not 1" << endl; }
```

Find the bug

- Common mistake is to use assignment '=' rather than equality comparison '==' operator
- Assignment puts 1 into x and then uses that value of x as the "condition"
  - 1 = true so we will always execute the if portion

```
// What's the problem below
int x;
cin << x;
if (x = 1) // should be (x == 1)
  { cout << "X is 1" << endl;}
else
  { cout << "X is not 1" << endl; }
```

Exercises

- Conditionals In-Class Exercises
  - Discount
  - Weekday
  - N-th

Switch (Study on own)

- Again used to execute only certain blocks of code
- Best used to select an action when an expression could be 1 of a set of values
- { … } around entire set of cases and not individual case
- Computer will execute code until a break statement is encountered
  - Allows multiple cases to be combined
- Default statement is like an else statement

```
switch(expr) // expr must eval to an int
{
 case 0:
   // code executed when expr == 0
   break;
 case 1:
   // code executed when expr == 1
   break;
 case 2:
 case 3:
 case 4:
   // code executed when expr is
   // 2, 3, or 4
   break;
 default:
   // code executed when no other
   // case is executed
   break;
}
```

Switch (Study on own)

- What if a break is forgotten?
  - All code underneath will be executed until another break is encountered

```
switch(expr) // expr must eval to an int
{
 case 0:
   // code executed when expr == 0
   break;
 case 1:
   // code executed when expr == 1
   // what if break was commented
   // break;
 case 2:
 case 3:
 case 4:
   // code executed when expr is
   // 3, 4 or 5
   break;
 default:
   // code executed when no other
   // case is executed
   break;
}
```

? Operator

- A simple if..else statement for assignment
  - `int x = (y > z) ? 2 : 1;`
  - Same as:

```
if(y > z)  x = 2;
else x = 1;
```

- Syntax: (*condition*) ? *expr_if_true* : *expr_if_false*;
- Meaning: the expression will result/return *expr_if_true* if *condition* evaluates to true or *expr_if_false* if *condition* evaluates to false

Performing repetitive operations

LOOPS

Need for Repetition

- We often want to repeat a task but do so in a concise way
  - Print out all numbers 1-100
  - Keep taking turns until a game is over
    - Imagine the game of 'war'…it never ends!!
- We could achieve these without loops, but…

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

int main()
{
  cout << 1 << endl;
  cout << 2 << endl;
  ...
  cout << 100 << endl;
  return 0;
}
```

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

int main()
{
  bool gameOver;
  gameOver = take_turn();
  if( ! gameOver ){
    gameOver = take_turn();
    if( ! gameOver ) {
        ...
    {
  }
}
```

Assume this produces a true/false result indicating if the game is over after performing a turn

while Loop

- While
  - Cond is evaluated first
  - Body only executed if cond. is true (maybe 0 times)
- Do..while
  - Body is executed at least once
  - Cond is evaluated
  - Body is repeated if cond is true

```
// While Type 1:
while(condition)
{
  // code to be repeated
  // (should update condition)
}



// While Type 2:
do {
  // code to be repeated
  // (should update condition)
} while(condition);
```

while Loop

- One way to think of a while loop is as a **repeating 'if' statement**
- When you describe a problem/solution you use the words '**until some condition is true**' that is the same as saying ***'while*** **some condition is** ***not*** **true**'

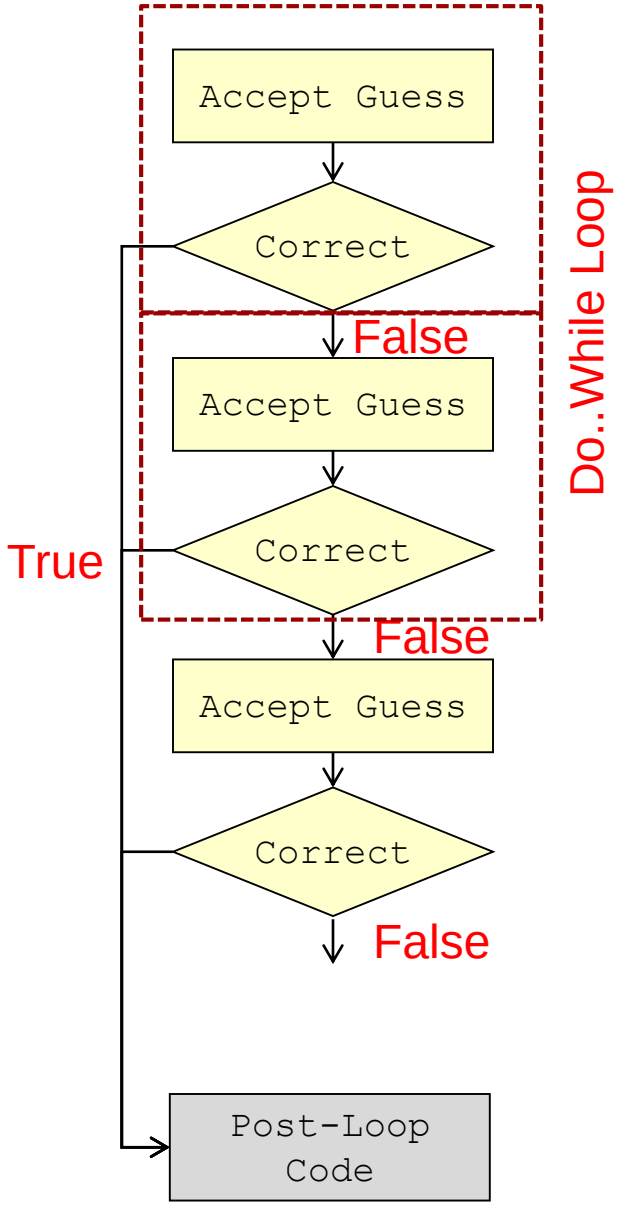
```
// guessing game
bool guessedCorrect = false;
if( !guessedCorrect )
{
   guessedCorrect = guessAgain();
}
// want to repeat if cond. check again
if( !guessedCorrect )
{
   guessedCorrect = guessAgain();
} // want to repeat if cond. check again
```

An if-statement will only execute once

```
// guessing game
bool guessedCorrect = false;

while( !guessedCorrect )
{
   guessedCorrect = guessAgain();
}
```

A 'while' loop acts as a repeating 'if' statement

Finding the 'while' Structure



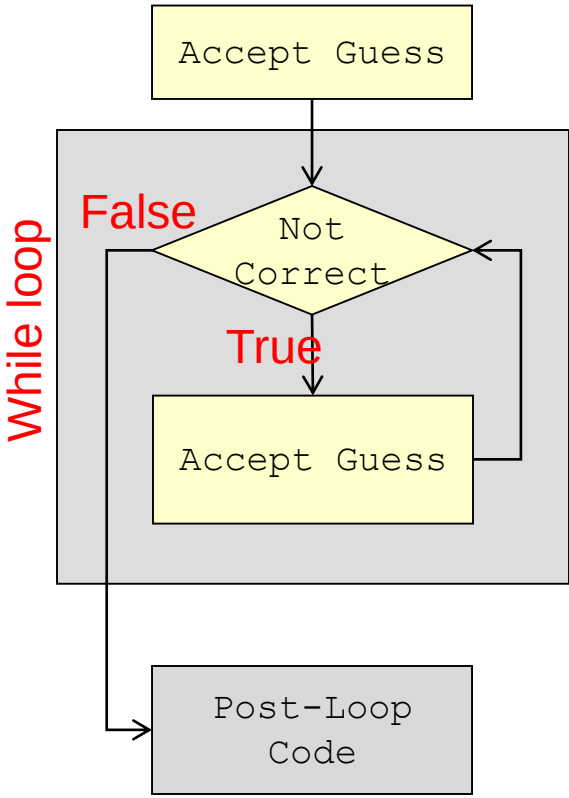
Draw out a flow chart of the desired sequence and look for the repetitive sequence

Here we check at the end to see if we should repeat…perfect for a do..while loop

```
do
  { accept_guess }
while ( ! correct )
```

But a while loop checks at the beginning of the loop, so we must accept one guess before starting:

```
accept_guess
while( ! correct )
  { accept_guess }
```



While Loop Exercise

- In-Class Exercises
  - countodd

for Loop

- Init stmt executed first
- Cond is evaluated next
- Body only executed if cond. is true
- Update stmt executed
- Cond is re-evaluated and execution continues until it is false
- Multiple statements can be in the init and update statements

```
for(init stmt; cond; update stmt)
{
  // body of loop
}

// Outputs 0 1 2 3 4 (on separate lines)
for(i=0; i < 5; i++){
  cout << i << endl;
}

// Outputs 0 5 10 15 … 95 (on sep. lines)
for(i=0; i < 20; i++){
  cout << 5*i << " is a multiple of 5";
  cout << endl;
}
// Same output as previous for loop
for(i=0; i < 100; i++){
  if(i % 5 == 0){
    cout << i << " is a multiple of 5";
    cout << endl;
  }
}

// compound init and update stmts.
for(i=0, j=0; i < 20; i++,j+=5){
  cout << j << " is a multiple of 5";
  cout << endl;
}
```

for vs. while Loop

- **'while' Rule of thumb**: Use when exact number of iterations is unknown when loop is started (i.e. condition updating inside the loop body)
- **'for' Rule of thumb**: Use when number of iterations is known when loop is started (independent of loop body)
- Both can be converted to the other…try it on the right

```
// guessing game
bool guessedCorrect = false;
while( !guessedCorrect )
{
   guessedCorrect = guessAgain();
}
```

Notice we cannot predict how many times this will run.

```
int x;
cin >> x;
for(i=0; i < x; i++){
  cout << 5*i << " ";
}
cout << endl;
```

Though we don't know x we can say the loop will run exactly x times.

```
for(init stmt; cond; update stmt)
{
  // body of loop
}
// Equivalent while structure
```

On your own time, practice tracing the following loops

TRACING EXECUTION 1

Tracing Exercises (Individually)

- To understand a loop's execution make a table of relevant variable values and show their values at the time the condition is checked
- If the condition is true perform the body code on your own (i.e. perform specified actions), do the update statement, & repeat

i (at condition check)	Actions of body
0	"0 "
1	"1 "
2	"2 "
3	"3 "
4	"4 "
5	-
Done	"0 1 2 3 4 \n"

```
int i;
cout << "For 1: " << endl;
for(i=0; i < 5; i++){
  cout << i << " ";
}
cout << i << endl;
```

Tracing Exercises (for 2-4)

- Perform hand tracing on the following loops to find what will be printed:

```
int i;

cout << "For 2: " << endl;
for(i=0; i < 5; i++){
  cout << 2*i+1 << " ";
}
cout << endl;


int j=1;
cout << "For 3: " << endl;
for(i=0; i < 20; i+=j){
  cout << i << " ";
  j++;
}
cout << endl;


j = 1;
cout << "For 4: " << endl;
for(i=10; i > 0; i--){
  cout << i+j << " ";
  i = i/2; j = j*2;
}
cout << endl;
```

Answers at end of slide packet

Tracing Exercises (for 5-6)

- Perform hand tracing on the following loops to find what will be printed:

```
int i;

char c = 'a';
i = 3;
cout << "For 5: " << endl;
for( ; c <= 'j';  c+=i ){
  cout << c << " ";
}
cout << endl;


double T = 8;
cout << "For 6: " << endl;
for(i=0; i <= T; i++){
  // Force rounding to 3 decimal places
  cout << fixed << setprecision(3);
  // Now print the number
  cout << sin(2*M_PI*i/T) << endl;
}
```

Answers at end of slide packet

Tracing Exercises (while 1-2)

- Perform hand tracing on the following loops to find what will be printed:

```
int i=15, j=4;
cout << "While loop 1: " << endl;
while( i > 5 && j >= 1){
  cout << i << " " << j << endl;
  i = i-j;
  j--;
}

i=1; j=1;
cout << "While loop 2: " << endl;
while( i || j ){
  if(i && j){
    j = !j;
  }
  else if( !j ){
    i = !i;
  }
  cout << i << " " << j << endl;
}
```

Answers at end of slide packet

Tracing Exercises (while 3)

- Perform hand tracing on the following loops to find what will be printed:

```
cout << "While loop 3: " << endl;
bool found = false;
int x = 7;
while( !found ){
  if( (x%4 == 3) &&
      (x%3 == 2) &&
      (x%2 == 1) )
  {
    found = true;
  }
  else {
    x++;
  }
}
cout << "Found x = " << x << endl;
```

Answers at end of slide packet

IN-CLASS CODING EXAMPLES

Loop Practice

- Write a for loop to compute the first 10 terms of the Liebniz approximation of π/4:
  - π/4 = 1/1 – 1/3 + 1/5 – 1/7 + 1/9 …
  - Tip: write a table of the loop counter variable vs. desired value and then derive the general formula
- http://bits.usc.edu/websheets/?folder=cpp/control
  - liebnizapprox

Counter (i)	Desired	Pattern	Counter (i)	Desired	Pattern
0	+1/1	for(i=0;i<10;i++) Fraction:	1	+1/1	for(i=1; i<=19; i+=2) Fraction:
1	-1/3		3	-1/3	
2	+1/5		5	+1/5	
…	…	+/- =>	…	…	+/- =>
9	-1/19		19	-1/19	

Loop Practice

- Write a for loop to compute the first 10 terms of the Liebniz approximation of π/4:
  - π/4 = 1/1 – 1/3 + 1/5 – 1/7 + 1/9 …
  - Tip: write a table of the loop counter variable vs. desired value and then derive the general formula

Counter (i)	Desired	Pattern	Counter (i)	Desired	Pattern

0	+1/1	for(i=0; I <10; i++)
		Fraction:
		1/(2*i+1)
		
		+/- =>
		pow(-1,i)
		if(i is odd)
		neg.
			1	+1/1	for(i=1; i <=19; i+=2)
					Fraction:
					1/i
					
					+/- =>
					if(i%4==3)
					neg.

1	-1/3		3	-1/3	
2	+1/5		5	+1/5	
…	…		…	…	
9	-1/19		19	-1/19	

Loop Practice

- Write for loops to compute the first 10 terms of the following approximations:
  - e^x: $1 + x + x^2/2! + x^3/3! + x^4/4! \ldots$
    - Assume 1 is the 1st term and assume functions
      - fact(int n) // returns n!
      - pow(double x, double n) // returns x^n
  - Wallis:
    - $\pi/2 = 2/1 * 2/3 * 4/3 * 4/5 * 6/5 * 6/7 * 8/7 \ldots$
    - http://bits.usc.edu/websheets/?folder=cpp/control
      - wallisapprox

The Loops That Keep On Giving

- There's a problem with the loop below
- We all write "infinite" loops at one time or another
- Infinite loops never quit
- When you do write such a program, just type "Ctrl-C" at the terminal to halt the program

```
#include <iostream>
using namespace std;
int main()
{ int val;
  bool again = true;
  while(again = true){
    cout << "Enter an int or -1 to quit";
    cin >> val;
    if( val == -1 ) {
       again = false;
    }
  }
  return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
  int i=0;
  while( i < 10 ) {
     cout << i << endl;
     i + 1;
  }
  return 0;
}
```

http://blog.codinghorror.com/rubber-duck-problem-solving/

The Loops That Keep On Giving

- There's a problem with the loop below
- We all write "infinite" loops at one time or another
- Infinite loops never quit
- When you do write such a program, just type "Ctrl-C" at the terminal to halt the program

```
#include <iostream>
using namespace std;
int main()
{ int val;
  bool again = true;
  while(again == true){
    cout << "Enter an int or -1to quit";
    cin >> val;
    if( val == -1 ) {
       again = false;
    }
  }
  return 0;
}
```

```
#include <iostream>
using namespace std;
int main()
{
  int i=0;
  while( i < 10 ) {
     cout << i << endl;
     i = i + 1;
  }
  return 0;
}
```

http://blog.codinghorror.com/rubber-duck-problem-solving/

Getting All The Inputs

- Notice another way to receive all the numbers entered by a user

```
while(cin >> val)
 { // do stuff  }
```

- In this approach cin does two things
  - It does receive input into the variable 'val'
  - It returns 'true' if it successfully got input, 'false' otherwise
- Keeps grabbing values one at a time until the user types Ctrl-D

```
#include <iostream>
using namespace std;
int main()
{ int val;
  // reads until user hits Ctrl-D
  // which is known as End-of-File(EOF)
 cout << "Enter an int or Ctrl-D ";
 cout << " to quit: " << endl;

 while(cin >> val){
    cout << "Enter an int or Ctrl-D "
    cout << " to quit" << endl;
    if(val % 2 == 1){
      cout << val << " is odd!" << endl;
    }
    else {
      cout << val << " is even!" << endl;
    }
  }
  return 0;
}
```

Single Statement Bodies

- An if, while, or for construct with a single statement body does not require { … }
- Another if, while, or for counts as a single statement

```
if (x == 5)
  y += 2;
else
  y -= 3;


for(i = 0; i < 5; i++)
  sum += i;


while(sum > 0)
  sum = sum/2;


for(i = 1 ; i <= 5; i++)
  if(i % 2 == 0)
     j++;
```

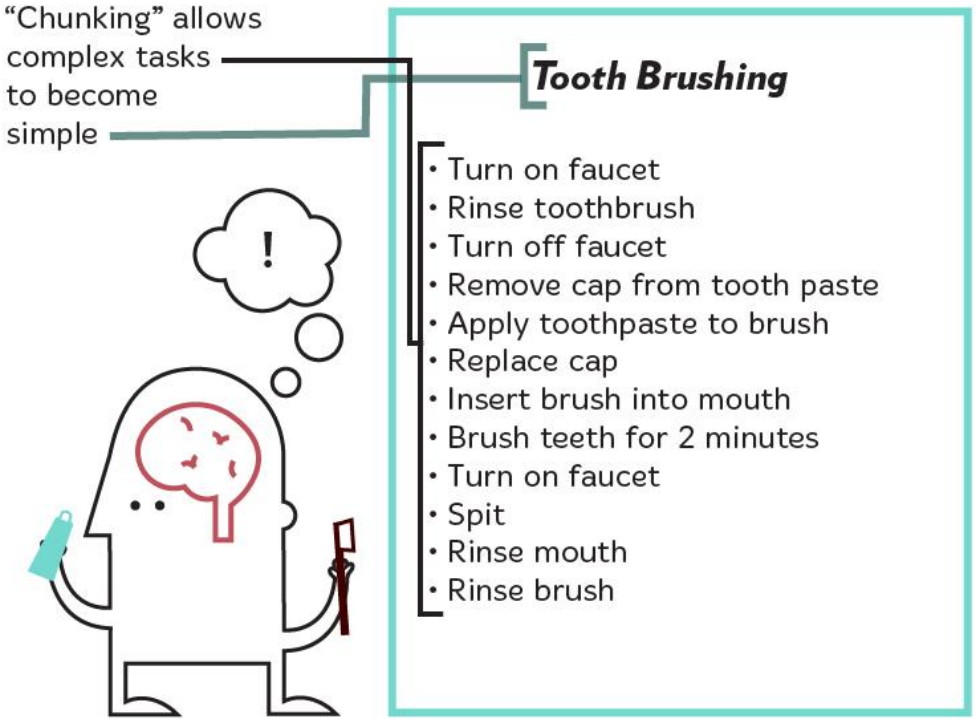
More Exercises

- Determine if a user-supplied positive integer > 1 is prime or not
  - How do we determine if a number is a factor of another?
  - What numbers could be factors?
  - How soon can we determine a number is not-prime?
- Reverse the digits of an integer[1]
  - User enters 123 => Output 321
  - User enters -5293 => -3925
  - In-class-exercises:
    - revdigits

[1]Taken from D.S. Malik, C++ Programming, 6th Ed.

20-second Timeout: Chunking

- Right now you may feel overwhelmed with all the little details (all the parts of a for loop, where do you need semicolons, etc.)
- As you practice these concepts they will start to "**chunk**" together where you can just hear "for loop" and will immediately know the syntax and meaning
- Chunking occurs where something more abstract takes the place of many smaller pieces



https://designbyben.wordpress.com/tag/chunking/

NESTED LOOPS

Nested Loops

- Inner loops execute fully (go through every iteration before the next iteration of the outer loop starts)

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

int main()
{
  for(int i=0; i < 2; i++){

    for(int j=0; j < 3; j++){
      // Do something based
      // on i and j
      cout << i << " " << j;
      cout << endl;

    }
  }
  return 0;
}
```

Output:

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

Nested Loops

- Write a program using nested loops to print a multiplication table of 1..12
- Tip: Decide what abstract "thing" your iterating through and "read" the for loop as "for each "thing" …
  - For each "row" …
    - For each column…
      print the product

	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9

```
#include <iostream>

using namespace std;

int main()
{
  for(int r=1; r <= 12; r++){
    for(int c=1; c <= 12; c++){
       cout << r*c;
    }
  }
  return 0;
}
```

This code will print some not so nice output:

123456789101112246810121416182022 24…

Nested Loops

- Tip: Decide what abstract "thing" your iterating through and "read" the for loop as "for each "thing" …
  - For each "row" …
    - For each column…
      print the product followed by a space
    - Print a newline

```
#include <iostream>

using namespace std;

int main()
{
  for(int r=1; r <= 12; r++){
    for(int c=1; c <= 12; c++){
       cout << " " << r*c;
    }
    cout << endl;
  }
  return 0;
}
```

	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9

This code will still print some not so nice output:

1 2 3 4 5 6 7 8 9 10 11 12
2 4 6 8 10 12 14 16 18 20 22 24

Nested Loops

- Tip: Decide what abstract "thing" your iterating through and "read" the for loop as "for each "thing" …
  - For each "row" …
    - For each column…
      print the product

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

int main()
{
  for(int r=1; r <= 12; r++){
    for(int c=1; c <= 12; c++){
       cout << setw(4) << r*c;
    }
    cout << endl;
  }
  return 0;
}
```

	1	2	3
1	1	2	3
2	2	4	6
3	3	6	9

Nested Loop Practice

- 5PerLine series
  - In-class-exercises:
    - 5perlineA
    - 5perlineB
    - 5perlineC
  - Each exercise wants you to print out the integers from 100 to 200, five per line, as in:

```
100 101 102 103 104
105 106 107 108 109
...
195 196 197 198 199
200
```

break and continue

- Break
  - Ends the current **loop** [not if statement] immediately and continues execution after its last statement
- Continue
  - Begins the next iteration of the nearest loop (performing the update statements if it is a for loop)
  - Can usually be accomplished with some kind of if..else structure
  - Can be useful when many nested if statements…

```
bool done = 0;
while ( !done ) {
  cout << "Enter guess: " << endl;
  cin >> guess;
  if( guess < 0 )
    break;
  }
  // ... Process guess

}

// Guess an int >= 0
while( !done ) {
  cin >> guess;
  if(guess < 0){
    continue;
  }
  // Can only be here if guess >= 0
}
// Equivalent w/o using continue
while( !done ) {
  cin >> guess;
  if(guess >= 0){
    // Process
  }
}
```

break and continue

- Break and continue apply only to the inner most loop (not all loops being nested)
  - Break ends the current (inner-most) loop immediately
  - Continue starts next iteration of inner-most loop immediately
- Consider problem of checking if a '!' exists anywhere in some lines of text
  - Use a while loop to iterate through each line
  - Use a for loop to iterate through each character on a particular line
  - Once we find first '!' we can stop

```
bool flag = false;
while( more_lines == true ){
  // get line of text from user
  length = get_line_length(...);

  for(j=0; j < length; j++){
   if(text[j] == '!'){
     flag = true;
     break; // only quits the for loop
   }
  }
}

bool flag = false;
while( more_lines == true && ! flag ){
  // get line of text from user
  length = get_line_length(...);

  for(j=0; j < length; j++){
   if(text[j] == '!'){
     flag = true;
     break; // only quits the for loop
   }
  }
}
```

C LIBRARIES & RAND()

Preprocessor & Directives

- Somewhat unique to C/C++
- Compiler will scan through C code looking for directives (e.g. #include, #define, anything else that starts with '#')
- Performs textual changes, substitutions, insertions, etc.
- **#include <filename> or #include "filename"**
  - Inserts the entire contents of "filename" into the given C text file
- **#define *find_pattern replace_pattern***
  - Replaces any occurrence of *find_pattern* with *replace_pattern*
  - #define PI 3.14159

    Now in your code:

    x = PI;

    is replaced by the preprocessor with

    x = 3.14159;

#include Directive

- Common usage: To include "header files" that allow us to access functions defined in a separate file or library
- For pure C compilers, we include a C header file with its filename: #include <stdlib.h>
- For C++ compilers, we include a C header file without the .h extension and prepend a 'c': #include <cstdlib>

C	Description	C++	Description
stdio.h **cstdio**	Old C Input/Output/File access	**iostream**	I/O and File streams
stdlib.h **cstdlib**	rand(), Memory allocation, etc.	**fstream**	File I/O
string.h **cstring**	C-string library functions that operate on character arrays	**string**	C++ string class that defines the 'string' object
math.h **cmath**	Math functions: sin(), pow(), etc.	**vector**	Array-like container class

rand() and RAND_MAX

- (Pseudo)random number generation in C is accomplished with the rand() function declared/prototyped in cstdlib
- rand() returns an integer between 0 and RAND_MAX
  - RAND_MAX is an integer constant defined in <cstdlib>
- How could you generate a flip of a coin [i.e. 0 or 1 w/ equal prob.]?

```
int r;
r = rand();
if(r < RAND_MAX/2){ cout << "Heads"; }
```

- How could you generate a decimal with uniform probability of being between [0,1]

```
double r;
r = staic_cast<double>(rand()) / RAND_MAX;
```

Seeding Random # Generator

- Re-running a program that calls rand() will generate the same sequence of random numbers (i.e. each run will be exactly the same)
- If we want each execution of the program to be different then we need to seed the RNG with a different value
- srand(int seed) is a function in <cstdlib> to seed the RNG with the value of seed
  - Unless seed changes from execution to execution, we'll still have the same problem
- Solution: Seed it with the day and time [returned by the time() function defined in ctime]
  - `srand( time(0) );  // only do this once at the start of the program`
  - `int r = rand();  // now call rand() as many times as you want`
  - `int r2 = rand();  // another random number`
  - `// sequence of random #'s will be different for each execution of program`

Only call srand() <u>ONCE</u> at the start of the program, not each time you want to call rand()!!!

Common Loop Tasks

- Aggregation / Reduction
  - Sum or combine information from many pieces to a single value
  - E.g. Sum first 10 positive integers
  - Declare aggregation variable and initialize it outside the loop and update it in each iteration
- Search for occurrence
  - Find a particular occurrence of some value or determine it does not exist
  - Declare a variable to save the desired occurrence or status, then on each iteration check for what you are looking for, and set the variable if you find it and break the loop

```
// aggregation example
int sum = 0;
for(int i=1; i <= 10; i++){
  sum += i;
}


// search for first perfect square
//  between m and n
int square = -1; // default
for(int i=m; i <= n; i++){
  if( sqrt(i)*sqrt(i) ==
                      (double)i){
    square = i;
    break;
  }
}
if( square != -1 ){
  // we have found such an int
}
```

Tracing Answers

```
For 1:
0 1 2 3 4 5

For 2:
1 3 5 7 9

For 3:
0 2 5 9 14

For 4:
11 6 5

For 5:
a d g j

For 6:
0.000
0.707
1.000
0.707
0.000
-0.707
-1.000
-0.707
-0.000
```

```
While loop 1:
15 4
11 3
8 2
6 1

While loop 2:
1 0
0 0

While loop 3:
Found x = 11
```