EECS 183

Week 5
Diana Gage

www-personal.umich.edu/~drgage

slides by Leah Bar-On Simmons

Main Concepts so far...

- Main concepts so far:
- Conditionals and Nested conditionals
- Functions
- Scope
- Operators and operations

- Any questions??

Today:

- In Class Exercise on loops
- Function review (with conditionals)
- Loops: while and for
- Strings
- Nested Loops

Conditional with a function

```
bool hello (string name);
int main (){
        int x = 1;
        if (hello("Jimmy") && (x == 1){
                cout << "We said hello to you!";
        }
   return 0;
}
bool hello (string name){
        cout << "Hello " << name << "!" << endl;
        return true;
}
```

Conditional with a function

```
bool hello (string name);
int main (){
        int x = 1;
        if (hello("Jimmy") && (x == 1){
                cout << "We said hello to you!";
        }
   return 0;
}
bool hello (string name){
        cout << "Hello " << name << "!" << endl;
        return true;
}
```

What is the output?

Conditional with a function

```
bool hello (string name);
Int main (){
        int x = 1;
        if (hello("Jimmy") && (x == 1){
                cout << "We said hello to you!";
        }
   return 0;
}
bool hello (string name){
        cout << "Hello " << name << "!" << endl;
        return true;
}
```

What is the output?

Hello Jimmy!

We said hello to you!

New topic: LOOPS

- What is a loop?

Loops

- What is a loop?
- What are its components?
- Two types of loops:
  - While loop
  - For loop

Definition

- A loop is…
  - a block of code that is executed **repeatedly** while a certain condition is true → stops when the condition becomes false!
  - Like an if statement that is executed **more than once**

3 Components

1. Initialization
2. Condition
3. Update

- All are important!
  - No update? →

Compound Assignment Operators

+= **-=** **/=**

***=** **%=**

Variable = variable + (expression)

~~ is the same as~~

Variable += expression

Compound Assignment

```
int  x;
x = 2;
x += 5;
x -= 2;
x *= 3;
```


```
int  x;
x = 2;
x = x + 5;
x = x - 2;
x = x * 3;
```

%= is tricky

x %= 20 - y

~~Is the same as~~

x = x % (20 – y) ← the entire expression

NOT x = x % 20 - y

%= is tricky

x %= 20 - y

~~*Is the same as*~~

x = x % (20 – y) ← the entire expression

NOT ~~x = x % 20 - y~~

++i and i++

- These are called the increment operators
- They increment i by one

- i++ happens after i does its job
  - This is called the post-increment operator

- ++i happens before i does its job
  - This is called the pre-increment operator

--i and i--

- These are called the decrement operators
- They decrement i by one

- i-- happens after i does its job
  - This is called the post-decrement operator

- --i happens before i does its job
  - This is called the pre-increment operator

Different Kinds of Loops

- Count-controlled
  - Ends after a certain **number** of iterations
  - Usually use a **for**, can use a **while**
- Event-Controlled
  - Ends after an **event** occurs that makes condition no longer true
  - Use a while – **DON'T** use a for
- You can **always** use a while loop, and you **must** use one in an event-controlled situation

Both kinds of loops

- **While** and **for** loops that sum numbers 0-4

```
int x   = 0;
int sum = 0;

while (x < 5) {
      sum += x;
      ++x;
}
```

```
int sum = 0;

for (int x = 0; x < 5; ++x) {
      sum += x;
```

initialization, condition, update

Event-controlled loops

- Don't know how long something will be true? → **event-controlled**:
  - While it is not raining, stay outside
    - As soon as it is raining (not raining = false)… go inside!
  - Good for checking user input
    - Prompt user to enter info. until valid

While loop for checking input

```
string answer;
cin >> answer;
```

initialization

condition

```
while (answer != “yes” && answer!= “no”) {
      cout << “Please type ‘yes’ or ‘no’.”;
      cin >> answer;
}
```

update

While loops with cin

```
int count = 0;
int num = 0;
cout << “Enter numbers!” <<

while (cin >> num){
        ++count;
}
cout << “You entered “ << count << “valid
numbers.” << endl;
```

A non-number will put **cin** into a "fail" state.

When **cin** is in the fail state, it will evaluate to 0 as a condition.

While loops with cin

```
int count = 0;
int num = 0;
cout << "Enter numbers!" <<

while (cin >> num){
        ++count;
}
cout << "You entered " << count << "valid
numbers." << endl;
```

A non-number will put **cin** into a "fail" state.

When **cin** is in the fail state, it will evaluate to 0 as a condition.

You'll need to clear the **cin** fail state before any further input takes place.

```
cin.clear();
```

Count-controlled Loops

- Want to do something a certain number of times? → **count-controlled:**
  - Hitting snooze on alarm to get more sleep
    - You can hit it **3** times and be on time → loop executes 3 times, then stops
  - Printing a certain number of stars

While loops

```
int x = 0; //initialize x

while (x < 5){
        cout << x << endl;
        ++x;
}
```

While loops

```
while (x < 5){
        cout << x << endl;
        ++x;
}
```

- **The (x < 5) is the…**

While loops

```
while (x < 5){
        cout << x << endl;
        ++x;
}
```

- **The (x < 5) is the…** condition

While loops

```
while (x < 5){
        cout << x << endl;
        ++x;
}
```

- The (x < 5) is the… condition
- **The cout << x << endl is the…**

While loops

```
while (x < 5){
        cout << x << endl;
        ++x;
}
```

- The (x < 5) is the… condition
- **The cout << x << endl is the…** loop body

While loops

```
while (x < 5){
        cout << x << endl;
        ++x;
}
```

- The (x < 5) is the… condition
- The cout << x << endl is the… loop body
- **The ++x is the…**

While loops

```
while (x < 5){
        cout << x << endl;
        ++x;
}
```

- The (x < 5) is the… condition
- The cout << x << endl is the… loop body
- **The ++x is the…** update

While loops

```
int x = 0; //initialize x

while (x < 5){
        cout << ++x << endl;
        ++x;
}
```

What would happen if the increment happened as above?

While loops

```
int x = 0; //initialize x

while (x < 5){
        cout << ++x << endl;
        ++x;
}
```

What would happen if the increment happened as above?

increment x, and then print this new value of x

While loops

```
int x = 0; //initialize x

while (x < 5){
        cout << x++ << endl;
        ++x;
}
```

What would happen if the increment happened this way instead?

print x is it is, and then **increment** x

For loops

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

What are the differences here?

For loops

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

What are the differences here?

- Everything happens inside parentheses
  - Initialization (int i = 0)
  - Condition (i < 5)
  - Update (++i)

  - **But its all still there!**

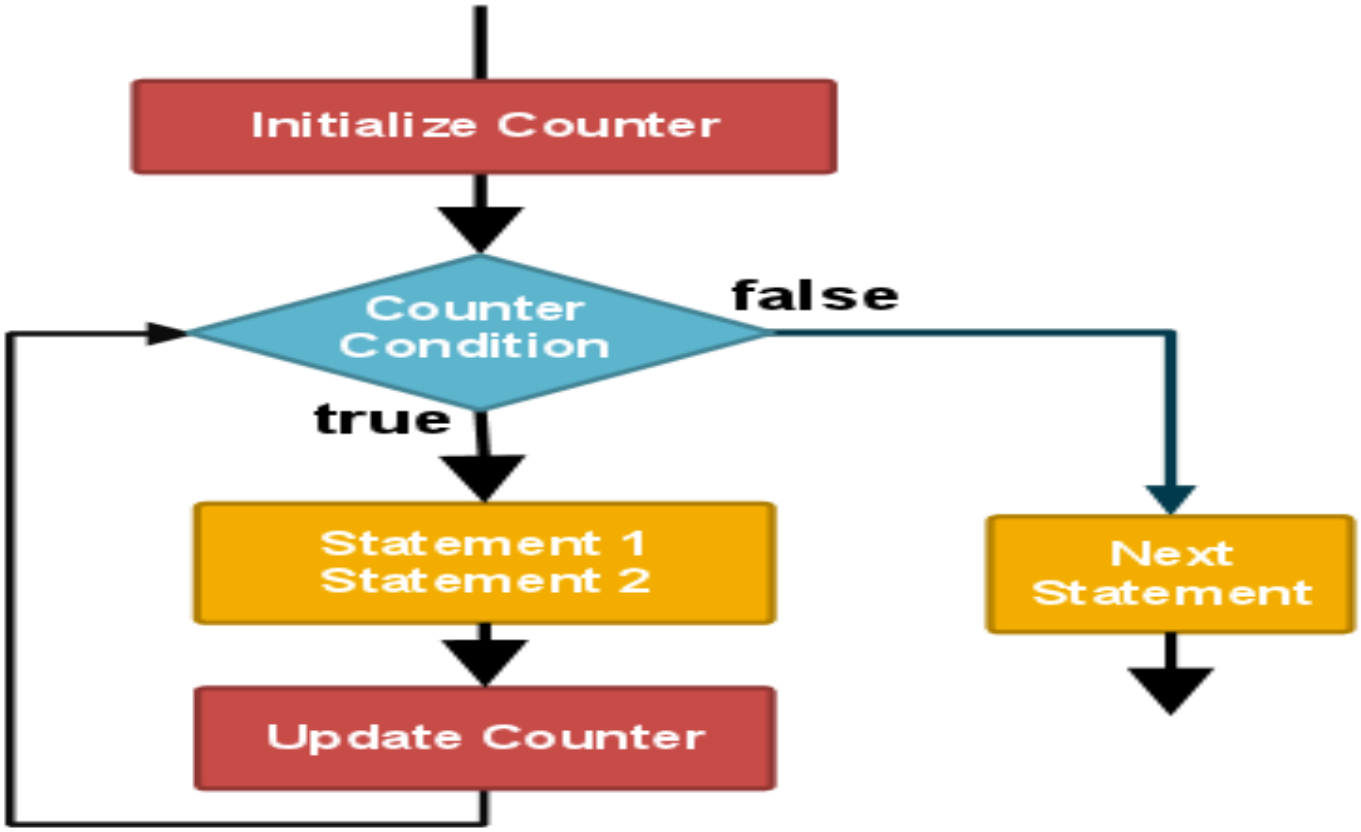
For loops

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

Establishing order of a 'for' loop:

1) **Initialize the counter variable i**
   - This only happens once
2) **Evaluate the condition**
   - Happens every time the loop runs again
3) **Update the counter variable**
   - Happens at the end of each iteration of the loop body

Logic for count-controlled loops



Difference in scope

```
int x = 0;
while (x < 5){
        cout << x << endl;
        ++x;
}
```

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

- **What is the scope of x?**

Difference in scope

```
int x = 0;
while (x < 5){
        cout << x << endl;
        ++x;
}
```

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

- **What is the scope of x?**
  - whatever function the while loop is in (could be main)

Difference in scope

```
int x = 0;
while (x < 5){
        cout << x << endl;
        ++x;
}
```

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

- What is the scope of x?
  - whatever function the while loop is in (could be main)
- **What is the scope of i?**

Difference in scope

```
int x = 0;
while (x < 5){
        cout << x << endl;
        ++x;
}
```

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

- What is the scope of x?
  - whatever function the while loop is in (could be main)
- **What is the scope of i?**
  - Only exists inside the for loop

Difference in scope

```
int x = 0;
while (x < 5){
        cout << x << endl;
        ++x;
}
```

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

- What is the scope of x?
  - whatever function the while loop is in (could be main)
- What is the scope of i **now?**

Difference in scope

```
int x = 0;
while (x < 5){
        cout << x << endl;
        ++x;
}
```

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

- What is the scope of x?
  - whatever function the while loop is in (could be main)
- What is the scope of i **now?**
  - **Now i exists in whatever function the for loop is in**

Common Errors with Loops

- Incrementing one further than you wanted (off-by-one errors)
  - < versus <= in condition
- Forgetting to update the counter (in while loops) or double updating (for loops)
- Infinite loops!
  - Make sure your condition will fail at some point

New Material

- Loops: for and while
- **Char types (characters) and the ASCII table**
- **String literals**
- **Nested loops**

chars and ASCII

- Chars are actually integers, and they get converted from int to char via the ASCII table

ASCII Table

A	B	C	D	E	F	G	H	I	J	K	L	M
65	66	67	68	69	70	71	72	73	74	75	76	77
N	O	P	Q	R	S	T	U	V	W	X	Y	Z
78	79	80	81	82	83	84	85	86	87	88	89	90

Lowercase letters have different ASCII values

chars and addition

- You can do something like

  cout << ‘A’ + 6 << endl;

- **What does this print?**

chars and addition

- You can do something like

  cout << ‘A’ + 6 << endl;

- **What does this print? 71**

  (value of char ‘A’ = 65 in ASCII table + 6)

chars and addition

- What about:

```
char c = 'a';
c += 3; //adds the three integers
cout << c << endl;
```

- What would this print?

chars and addition

- What about:

```
char c = 'a';
c += 3;
cout << c << endl;
```

- What would this print? **d**
  - Because we are printing out a variable of type char, the conversion happens

String Literals

- Remember what a string is made up of?

String Literals

- Remember what a string is made up of?
  - **A collection of chars**

- We can access each one of these chars

```
string name = "Jim";
cout << name[0] << endl; //prints: J
```

Accessing characters of strings

```
string name = "Jim";
cout << name[0] << endl; //prints: J
```

Strings are what we call: **0 indexed**

- The first char of the string is at **index 0**
- The second char of the string is at **index 1**
- **And so on…**

The index is inside the brackets: [1]

Adding Characters to Strings

- This is called **concatenating**

```
string name = "Jim";
cout << name[0] << endl; //prints J

name += 'm'; //adds m to the end of the string
name += 'y'; //adds y to the end of the string

cout << name << endl; //prints Jimmy
```

Nested Loops

- Nested loops means one loop is inside another
- There is an outer loop, and then inner loops (can be many)
- Need to keep track of how the inner loops are controlled by the changes to the outer loop
- The inner loops will run many times, whereas the outer loop will run once through to completion
- Can you have a **for loop** inside a **while loop** and vice-versa?

Nested Loops

- Nested loops means one loop is inside another
- There is an outer loop, and then inner loops (can be many)
- Need to keep track of how the inner loops are controlled by the changes to the outer loop
- The inner loops will run many times, whereas the outer loop will run once through to completion
- Can you have a **for loop** inside a **while loop** and vice-versa? **YES**

Nested Loops

- Important for upcoming projects
- Let's practice together

- Let's print a 3x3 square of * (star characters)

- **First with three separate loops**

Nested Loops

- Important for upcoming projects
- Let's practice together

- Let's print a 3x3 square of * (star characters)

- **Now with a loop and a nested loop**

Let's code some loops!

- **Pick some of the following to try out, and write each using a *while* loop and a *for* loop**
- Sample source code file is on Ctools
- Printing only odd or only even numbers from 0 to 100
- Count to 1000 by twos, and only print the numbers divisible by 50
- Count to 50 by 2s, then finish to 100 by 5s (two loops needed here – not nested! Why?)
- Print each letter of a string on a separate line until the string is done (tricky – feel free to ask)
- Any other fun ones you can think of!