



EECS 183

Week 4
Diana Gage

[www-personal.umich.edu/
~drgage](http://www-personal.umich.edu/~drgage)

slides by Leah Baron-Simmons

So far in EECS183...

- Variables and data types
- Input/output (cout, cin)
- Operators and operations
 - Important for conditionals
- Project 1 (yay!)
- Functions
- Scope

- Any general questions??

Upcoming Deadlines

- **Assignment 2** will be due this Friday (10/2)
 - Debugging Exercise, Zyante, Codelab
- **Project 2** will be due next Friday (10/9)
 - Get started early once it is released!
- Office Hours might be updating location from the UGLi – Stay tuned!

Why use a function?

- Helps reduce duplicated code
 - Call more than once with new parameters to do the same work, but with new values
- **One approach:** If you know all the details about a function (RME) you can implement it...
 - assuming everything else, including main, already works (even if it doesn't yet)

Why use a function?

- **Another approach:** plan out the logic of your `main()` first
 - you can assume all the functions already do what they are supposed to do...
 - even though you haven't implemented them yet!
- Helps organize your project and your code
 - This is good for you, and also for anyone that reads or uses your code

Function Signature

- what is the function signature of our add function?

```
int add (int a, int b){  
    return a + b;  
}
```

Function Signature

- what is the function signature of our add function?

```
int add (int a, int b){  
    return a + b;  
}
```

int add(int, int) or int add(int a, int b)

Function Signature

int add(int, int) or int add(int a, int b)

- The signature of a function is the combination of the unique/defining elements of the function
 - Return type
 - Name
 - Parameter types

Practice with Functions

- Using 183study!
- Practice exam questions
- Fall 2014 Exam 1, questions 3, and 16

Last week's challenge problem

- ◉ Working with square roots and conditionals
- ◉ It was meant to be tricky, and look ahead
- ◉ It should make more sense now after this week's lectures!
- ◉ Source code and possible solution online
- ◉ Understanding it will be extremely helpful for breaking down Project 2

Project 2 - overview

- **Birthday Calculator**

- User puts in a date, and the program prints out what day of the week it is/was/will be on
- All about functions!
- Functions can and should call other functions!
- Main will execute many function calls, but some functions will only be called within other functions (not directly from main)
- The functions have been given to you in the source code – your job is to implement each function, and implement the `main()` function

Project 2 – plan ahead

- Plan out which functions are called by main, and which function serve only to help other functions
- Plan out the order in which main will call functions
- Be very comfortable with the details in the spec
- Read and understand the RME's given to you above the function declarations

Review: Scope

- What is an example of Local scope?
- What is the difference between a global variable, and a const global variable?
- Which do we NEVER use?

Review: Scope

- What is the difference between a global variable, and a const global variable?
- Which do we NEVER use?
 - We NEVER use **global** variables
 - We do use **const global** variables
- If you are going to use a variable with a global scope, it MUST be declared const, and never be changed throughout the program

Review: Operations and Operators

+	-	/	%
&&		!	
<	>	<=	>=
		!=	

Review:

Operations and Operators

+ addition

- subtraction

/ division

% remainder (modulus)

Review:

Operations and Operators

&& 'and'

|| 'or'

! 'not' (negation)

= assignment operator

== comparison operator

Review:

Operations and Operators

< less than

> greater than

<= less than or equal to

>= greater than or equal to

!= not equal to

Assignment Operator

- The assignment operator is:

=

- This operator assigns whatever is on the left to whatever is on the right

```
int k = 25;
```

Comparison Operator

- The comparison operator is:

==

- This operator compares whatever is on the left to whatever is on the right
- This comparison will evaluate to true or false
- Either the two sides are the same, or not
- **Be careful with the difference between = and ==**

Boolean Operators

Shown Within Operator Precedence

Order	Operator	Meaning	Associativity
1	()	Group or cast	Left to right
2	!x +x -x	Not, negate	Right to left
3	* / %	Multiply, divide, modulo	Left to right
4	+ -	Add, subtract	Left to right
5	< <= >= >	Greater/less than (or equal to)	Left to right
6	<< >>	Input/output	Left to right
7	== !=	(Not) equal	Left to right
8	&&	Logical and	Left to right
9	 	Logical or	Left to right

New Material: Conditionals

- What is a conditional?

New Material: Conditionals

- What is a conditional?
 - **A statement with a condition**
 - **E.g. an 'if' statement**

New Material: Conditionals

- What is a conditional?
 - **A statement with a condition**
- If something evaluates to true, we want to do one thing
- If that thing evaluates to false, we want to do something else
- \wedge general idea

Example: if statement

```
int num = 23;
```

```
if (num >= 18){  
    cout << "You can get a tattoo!";  
    cout << endl;  
}
```

if, else if, else statements

- Often conditionals are a group of statements comprised of:
 - if (...) {...}
 - else if (...) {...} *(there can be many else ifs)*
 - else {...}
- These are called **branches**

Branching

- Often conditionals are a group of statements comprised of:
 - if (...) {...}
 - else if (...) {...} Can have as many 'else if's as you want
 - else {...}
- **Notice the else statement does not have parentheses, only brackets**
- **The else is a “catch all”, so it doesn't have a condition**

Branching

- These statements are **dependent on each other** in the **order** the code is written
- `if (...) {...}`
 - This always goes first, and it will either execute (if true) or not (if false)
- `else if (...) {...}`
 - This runs only if the previous if statement did not execute
- **`else {...}`**
 - This runs only if neither of the previous statements executes
 - This will **ALWAYS** execute if the neither of the previous statements did

Example of Branching – pseudo-code*

```
if (sunny && above60){  
    cout << "Let's play outside!" << endl;  
}  
else if (rainy){  
    cout << "Let's watch a movie!";  
    cout << endl;  
}  
else{  
    cout << "I can't decide!" << endl;  
}
```

Branching

- ◉ There can be as many else ifs as you want
- ◉ But only **one** 'if', and only **one** 'else' in a branch
- ◉ This doesn't mean you can't have many 'if' statements in a row
 - ◉ Each each will execute always, and regardless of the previous one
 - ◉ Sometimes this is what you want!

IMPORTANT difference:

= VS. ==

- Be careful not to use the assignment operator **within a conditional**
- Instead of checking whether they are the same (true) or different (false)...
 - The line of code will set the left equal to whatever was on the right
 - This is probably not what you want

Common error: = vs. ==

//let's say the following code is in a function

```
int a = 30;
```

```
int b = 25;
```

```
if (a == b){  
    cout << "equal" << endl;  
    return true;  
}
```

```
if (a = b){  
    cout << "equal" << endl;  
    return true;
```


Common error: = vs. ==

//let's say the following code is in a function

```
int a = 30;
```

```
int b = 25;
```

```
if (a == b){  
    cout << "equal" << endl;  
    return true;  
}
```

```
if (a = b){  
    cout << "equal" << endl;  
    return true;  
}
```

The if statement won't execute because a is not equal to b

Common error: = vs. ==

//let's say the following code is in a function

```
int a = 30;
```

```
int b = 25;
```

```
if (a == b){  
    cout << "equal" << endl;  
    return true;  
}
```

```
if (a = b){  
    cout << "equal" << endl;  
    return true;  
}
```

Common error: = vs. ==

//let's say the following code is in a function

```
int a = 30;
```

```
int b = 25;
```

```
if (a == b){  
    cout << "equal" << endl;  
    return true;  
}
```

```
if (a = b){  
    cout << "equal" << endl;  
    return true;  
}
```

- a will be set to 25
- the value 25 is true
- the statements inside will execute
- but that's incorrect!

Style with Conditionals

- **Do not** do

if (true) or if (a == false)

- Do not compare doubles
- Do not compare things of different types
 - ie string and doubles
- Use consistent brackets
- Don't over-complicate expressions

If (!(a == b)) vs. if (a != b)

Important!

age = 25

$18 \leq \text{age} \ \&\& \ \text{age} \leq 22$

true (1)

false (0)

false

VS

$18 \leq \text{age} \leq 22$

true (1)

true

ALWAYS 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;  
}
```

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!

Nested Conditionals

- You can have **conditionals exist within other conditionals**

```
int money_for_tuscany = 10000;
int money_for_london = 7000;

if (has_money && has_vacation_time){
    if (has_money > money_for_tuscany){
        cout << "Let's travel to Tuscany!" << endl;
    }
    else if (has_money > money_for_london){
        cout << "Let's go to London!" << endl;
    }
    else{
        cout << "Let's go somewhere in the US" << endl;
    }
}
else{
    cout << "Netflix?" << endl;
}
```

New Concept: Short Circuits

- What evaluates first if you have a conditional such as:

```
if (a == b || a > b){  
    cout << "a is not less than B" << endl;  
}
```

Short Circuits

- What evaluates first if you have a conditional such as:

```
if (a == b || a > b){  
    cout << "a is not less than b)" << endl;  
}
```

- The program will first check if `a == b` is true
- Only one of the conditions must be true for an 'or' statement to be true...
 - if `a==b` is true, the program won't even check the `a > b` condition

Purpose of Short Circuit

- To save time, chains of && and || are not evaluated after the first false or true
- To make use of this trick, put the thing most likely to be true or false **first** in the conditional

Loops

- What is a loop?
- What are its components?
- Two types of loops
 - we've only discussed **while** loops so far!
 - more on the other type (for loops) next week!

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? →



While Loop

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

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

initialization, condition, update

When to use while?

- You can **always** use a while loop (keep this in mind when you learn for loops)
- You **must** use a while in event-controlled situations
 - Event-controlled: ends after an **event** occurs that makes condition no longer true
 - Count-controlled: ends after a certain **number** of iterations
 - For loops will help with this later too

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
```

```
while (answer != "yes" && answer !=      condition  
"no") {
```

```
    cout << "Please type 'yes' or 'no'.";
```

```
    cin >> answer;      update
```

```
}
```

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 loop for printing 3 stars (each on own line)

DESIRED OUTPUT:



```
*
```

```
*
```

```
*
```

```
int count = 0;  
while (count < 4) {  
    cout << "*" << endl;  
}
```

**What's missing from
this loop?**

What's the problem?

While loop for printing 3 stars (each on own line)

DESIRED OUTPUT:



```
*  
*  
*
```

```
int count = 0;  
while (count < 4) {  
    cout << "*" << endl;  
    ++count;  
}
```

**What's missing from
this loop? **update****

What's the problem?

We enter an infinite loop if we
never change count

Open Discussion for Questions!

- Conditionals
- Functions
- While Loops
- Codelab
- Projects!
- EECS183