EECS 183

Week 4
Diana Gage

www-personal.umich.edu/~drgage

Upcoming Deadlines

- Assignment 2 due September 30th (this Friday)
- Project 2 will be due October 7th (a week from Friday)
  - Get started early!

Scope

- A variable can either have a local scope or a global scope
- Local scope
  - Exists only within current function
- Global scope
  - Exists for all functions in the program

Full program && Scope

```
void say_hello (string name_in); //function declaration

int main(void){
        string name = "Jimmy";
        say_hello(name);
}

void  say_hello(string name_in) {
        cout << "Hello " << name_in << endl;
}
```

What is the scope of the variable name?

What is the scope of the variable name_in?

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

  ^ general idea

New Operations and Operators

&& ‘and’

|| ‘or’

! ‘not’ (negation)

= assignment operator

== comparison operator

New Operations and Operators

<	less than
>	greater than
<=	less than or equal to
>=	greater than or equal to
!=	not equal to

Example: if statement

```
int num = 23;

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

&& -- The 'and' operator

- **Both** pieces of the conditional must evaluate to true
- Sample code:

```
if (sunny && above_60){
        cout << "Let's throw a disc outside!";
        cout << endl;
}
```

- Both must be true

|| – The 'or' operator

- **One or more** pieces of the conditional must evaluate to true
- Sample code:

```
if (sunny || above_60){
        cout << "I guess we can throw a disc outside";
        cout << endl;
}
```

- One, or the other, **or both**, must be true

! – The ‘not’ operator

- This is the negation operator
- It takes the truth value of the variable or expression in question, and negates it

```
bool has_green_hair = true;
cout << !has_green_hair << endl;
```

What does this print?

! – The ‘not’ operator

- This is the negation operator
- It takes the truth value of the variable or expression in question, and negates it

```
bool has_green_hair = true;
cout << !has_green_hair << endl;
```

What does this print? **0**

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 (…) {…}
  - else {…}

Can have as many 'else if's as you want

- ***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
- The *else if* follows directly from the *if*, and the *else* follows from both the *if* and *else if*

Branching

- 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 neither of the previous statements did**

Example of Branching:

```
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!

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 ==**

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!

Short-circuit evaluation

```
bool print_hello(){
    cout << "Hello" << endl;
    return true;
}

int main(){
    bool sunny = true;
    if (sunny && print_hello()){
        cout << "It's sunny and we said hello!" << endl;
    }
    else if (sunny || print_hello()){
        cout << "its either sunny, or we printed hello, or both!" << endl;
    }
    else{
        cout << "It's not sunny and we didn't say hello." << endl;
}
```

What will happen? What will print?

Short-circuit evaluation

```
bool print_hello(){
    cout << "Hello" << endl;
    return true;
}

int main(){
    bool sunny = true;
    if (sunny && print_hello()){
        cout << "It's sunny and we said hello!" << endl;
    }
    else if (sunny || print_hello()){
        cout << "its either sunny, or we printed hello, or both!" << endl;
    }
    else{
        cout << "It's not sunny and we didn't say hello." << endl;
}
```

What will happen? What will print? Because of short-circuit evaluation, if the first element in an or statement conditional evaluates to true, the rest of the conditional won't even be checked

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 <= age && age <= 22

true (1) false (0)

false

VS

18 <= age <= 22

true (1)

true

ALWAYS True!!

Class Exercise: Nested Conditionals

- Take 5 minutes to try to write out the following scenario in pseudocode – we will turn it into full code as a class

Scenario:

- It costs 10000 dollars to go on a trip to Tuscany, and 7000 dollars to go on a trip to London
- Your user will run your program and you will read in how much money the user has
- There is a function called doYouHaveVacationTime() that will return true or false whether the user has vacation time (ask the user)
- You can't travel anywhere unless you have vacation time – you will watch Netflix at home instead
- If you can go to Tuscany you will; London is your second choice; if you can't afford either you will go somewhere in the US

Comparing doubles

- You have to be careful when comparing doubles, since their values are not exact
- Use: if (fabs(x - y) < .0001) instead
  - This is taking the absolute value (in float/ double form) of the difference between x and y
  - This is essentially the same as comparing the two for equality, but you have to check that the result is LESS THAN .0001

Comparing strings

- You can use the comparison operator with strings (==)
- Make sure you understand that **case matters**
  - "Apple" will not be evaluated as equal to "apple"
- The comparison does a character-by-character comparison of each ascii value
- Capital "A" is **less than** lowercase "a" in the ascii table, therefore "Apple" < "apple"
- A longer string will evaluate as **greater than** a shorter string

New Concept: Switch Statements

- Switch statements are a simpler way to organize code if all the *else if* statements have the same kind of condition
- For instance:
  - All the *else if* are checking if x == y, where x and y are variable (can change)
  - Each case is like an *else if* statement, but simpler and with less code to type
  - There is a default case, which works like an *else* statement

Switch example (from lecture)

```
int n = 1; // or char
switch (n) {
        case 0:
                cout << “n is 0” << endl;
                break;
        case 1:
                cout << “n is 1” << endl;
                break;
        default:
                cout << “n is something else… << endl;
                break;
}
```

Switch example (from lecture)

```
int n = 1;
switch (n) {
        case 0:
                cout << "n is 0" << endl;
                break;
        case 1:
                cout << "n is 1" << endl;
                break;
        default:
                cout << "n is something else… << endl;
                break;
}
```

OUTPUT:

n is 1

Switch Statements: details

- A switch statement can only have an integer or char value as the condition
- If you are comparing strings, a switch statement is not the way to go – use branching instead
- The 'break' tells the program that the work for that case is done, and the switch statement should be exited
- What happens if the break isn't there?

Switch Statements: details

- A switch statement can only have an integer value as the condition
- If you are comparing strings, a switch statement is not the way to go – use branching instead
- The 'break' tells the program that the work for that case is done, and the switch statement should be exited
- What happens if the break isn't there?
  - The program execution will fall through to the next case of the switch statement

Switch example (from lecture)

What if we take out the break?

```
int n = 1;
switch (n) {
        case 0:
                cout << "n is 0" << endl;
                break;
        case 1:
                cout << "n is 1" << endl;
                break;
        default:
                cout << "n is something else… << endl;
                break;
}
```

Switch example (from lecture)

What if we take out the break?

```
int n = 1;
switch (n) {
        case 0:
                cout << "n is 0" << endl;
                break;
        case 1:
                cout << "n is 1" << endl;
                break;
        default:
                cout << "n is something else… << endl;
                break; // do we need this one?
}
```

OUTPUT:

n is 1

n is something else

Switch vs. if-else if-else

- Efficiency
  - switch typically runs faster
  - Not a big difference with modern compilers
- Generality
  - switch syntax limits applicability
    - (many things that **if** can do that **switch** can't)
    - can't switch on doubles, ranges, strings, …
- Readability
  - switch table format very clear
- Errors
  - Less likely to create subtle bugs with switch

Feel free to ask any questions after class!