Unit 1d – Conditional (if..else) Statements

Conditional ('if'…'else') Statements

Outline

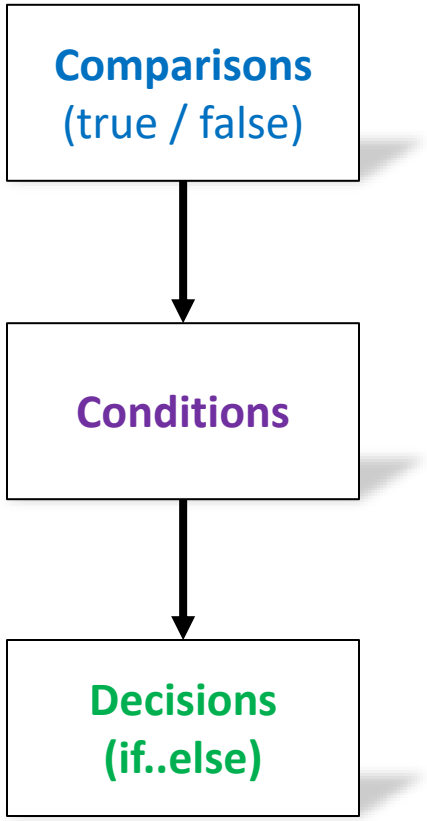
- Comparison operators
- Conditional statements (3 forms)
  - `if` statements
  - `if..else` statements
  - `if..else if..else` statements
- Compound conditional statements
  - AND, OR, and NOT (&&, ||, !) operators

Control Structures

- We need ways of making **decisions** in our program
  - To repeat code until we want it to stop
  - To only execute certain code if a condition is true
  - To execute one segment of code or another
- Language constructs that allow us to make decisions are referred to as **control structures**
- The common ones are:
  - if statements
  - switch statements – Not covered nor necessary in this class
  - while loops (Unit 2)
  - for loops (Unit 2)

Making Decisions

- **Comparison** of values results in 'true' or 'false' results (e.g. `x > 0`)
- Using comparisons we can develop simple or compound **conditions**
  - If `usc_wins >= 12` we will play for the championship
  - If `x > 0 AND y > 0`, take some action
- Using **conditions** we can make **decisions** about what code to execute
  - `if(x > 0 AND y > 0)`



Comparison Operators

- To perform comparison of variables, constants, or expressions in C/C++ we can use the basic 6 comparison operators

Operator(s)	Meaning	Example
==	Equality	`if(x == y)`
!=	Inequality	`if(x != 7)`
<	Less-than	`if(x < 0)`
>	Greater-than	`if(y > x)`
<=	Less-than OR equal to	`if(x <= -3)`
>=	Greater-than OR equal to	`if(y >= 2)`

Conditional Execution – 'if'

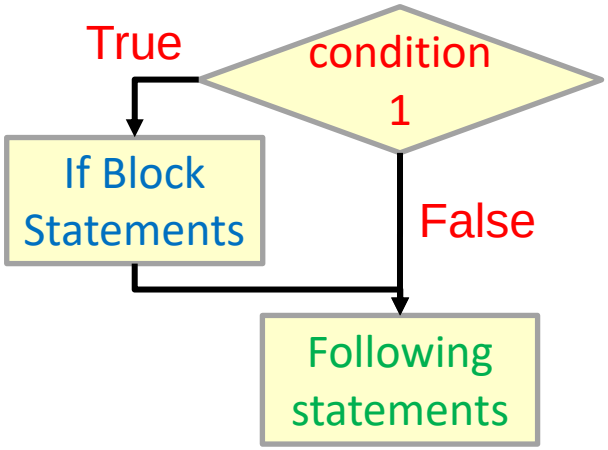
- `if` statements are the primary structures we use to execute a block of code only if a certain condition is met (true).
- Skips code inside { } if condition is false

(T / F) = Condition

```
if (condition1)
{
  // executed if condition1
  // is true
}
// following statements
x = x + 1;
```

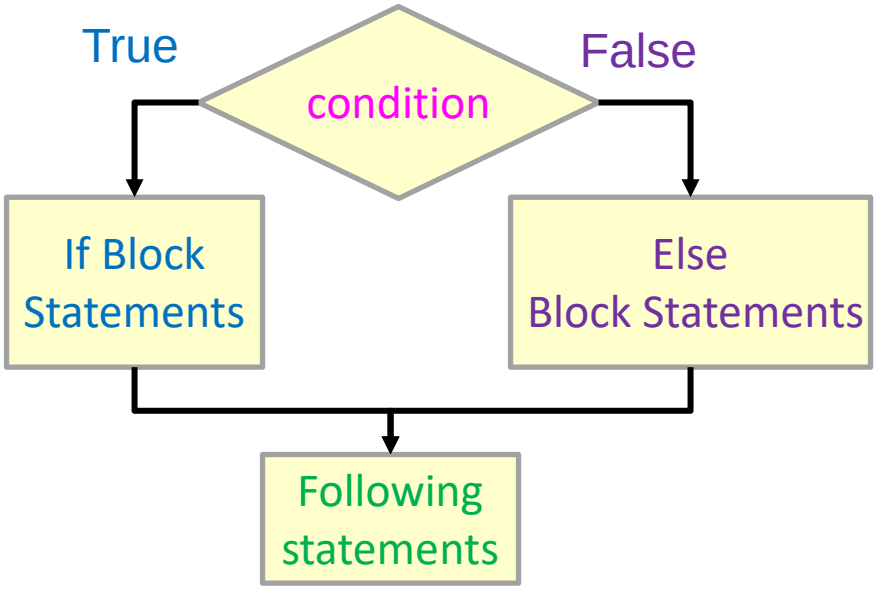
Example

```
int x;
cin >> x;
if (x < 0)
{
  x = -x;
}
cout << x << endl;
```



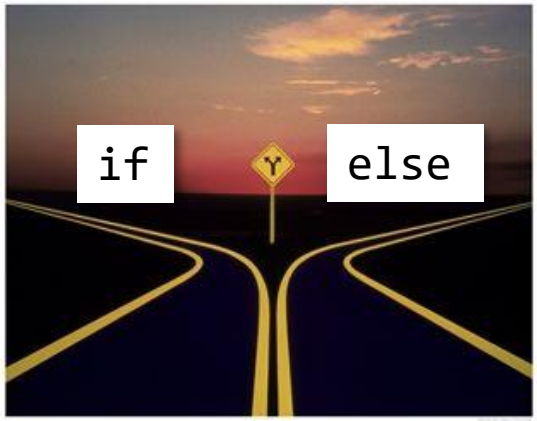
Conditional Execution – 'if..else'

- `else` statements are always optional and will execute when `if` conditions are false
- Create a mutually exclusive (one or the other) code execution structure



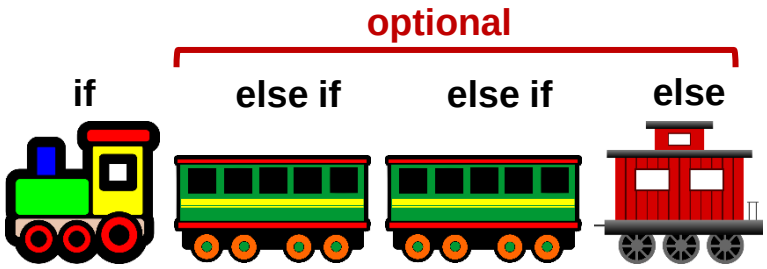
(T / F) = Condition

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



If…Else If…Else

- Got more than 2 cases, use `'else if'`
- Can have any number of `else if` statements
  - `else if` is ***optional***
- Reminder: `else` is ***optional***
- { … } indicate code associated with the if, else if, else block

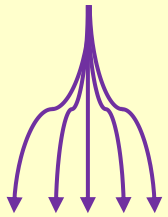



```
if (condition1)
{
  // executed if condition1 is true
}
else if (condition2)
{
  // executed if condition2 is true
  //  but condition1 was false
}
else if (condition3)
{
  // executed if condition3 is true
  //  but condition1 and condition2
  //  were false
}
else
{
  // executed if neither condition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

Pick the Right Structure

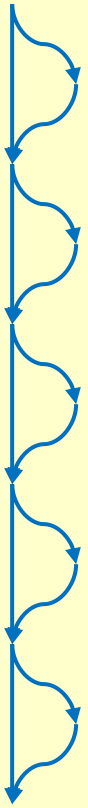
- What will each implementation print if 'grade' is 95?

```
if (grade >= 90)
{
  cout << "A range" << endl;
}
else if (grade >= 80)
{
  cout << "B range" << endl;
}
else if (grade >= 70)
{
  cout << "C range" << endl;
}
else if (grade >= 60)
{
  cout << "D range" << endl;
}
else
{
  cout << "Not gonna happen!" << endl;
}
```



Only one block can execute

```
if (grade >= 90)
{
  cout << "A range" << endl;
}
if (grade >= 80)
{
  cout << "B range" << endl;
}
if (grade >= 70)
{
  cout << "C range" << endl;
}
if (grade >= 60)
{
  cout << "D range" << endl;
}
else
{
  cout << "Not gonna happen!" << endl;
}
```



Any number of 'if`s can execute

Common Style Mistake

- Using multiple `if`s when `if..else if..else` is appropriate
  - It's not functionally wrong but is less readable and error-prone
- Guideline:
  - If various blocks of code are **mutually exclusive** then put them in an `if..else if..else` structure and not many individual `if..` `if..` statements

```
// BAD STYLE!
if (x < 0) {
  cout << "negative" << endl;
}
if (x >= 0) {
  cout << "positive" << endl;
}

// GOOD STYLE!
if (x < 0) {
  cout << "negative" << endl;
}
else {
  cout << "positive" << endl;
}
```

Rule/Exception Idiom

- A common use of a single `if` statement is to deal with an exceptional case.
- Perform a default action before the `if` and then us the `if` to correct any exceptional cases

```
// Default action

if( /* Exceptional Case */ )
{
   // Code to apply to
   // exceptional case
}
```

Structure

```
bool primeMember = /* set somehow */;

double shippingFee = 7.99;
if( primeMember == true )
{
   shippingFee = 0;
}
```

Example 1

```
double salary = /* some value */

double bonus = 0.05 * salary;
if( manager == true )
{
   bonus += 1000;
}
```

Example 2

Rule Exception Idiom (2)

- **Connections:** Often equivalent to use the 'else' for one of the cases

```
bool primeMember = /* set somehow */;

double shippingFee = 7.99;
if( primeMember == true )
{
   shippingFee = 0;
}
```

```
bool primeMember = /* set somehow */;

if( primeMember == true )
{
   shippingFee = 0;
}
else
{
   shippingFee = 7.99;
}
```

What Goes In an `if` / `else` Block

- What do we put in an `if` or an `else` statement?
- ANYTHING!
  - Expressions & variable assignment
  - Function calls
  - **Even other `if..else` statements**

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

int main()
{
  double val;
  cout << "Enter a pos. #" << endl;
  cin >> val;

  if( val >= 0) {
    double res = sqrt(val);
    cout << "Result=" << res << endl;
  }
  else {
    cout << "Error!" << endl;
  }

  return 0;
}
```

Decision Tree (Subcase) Idiom

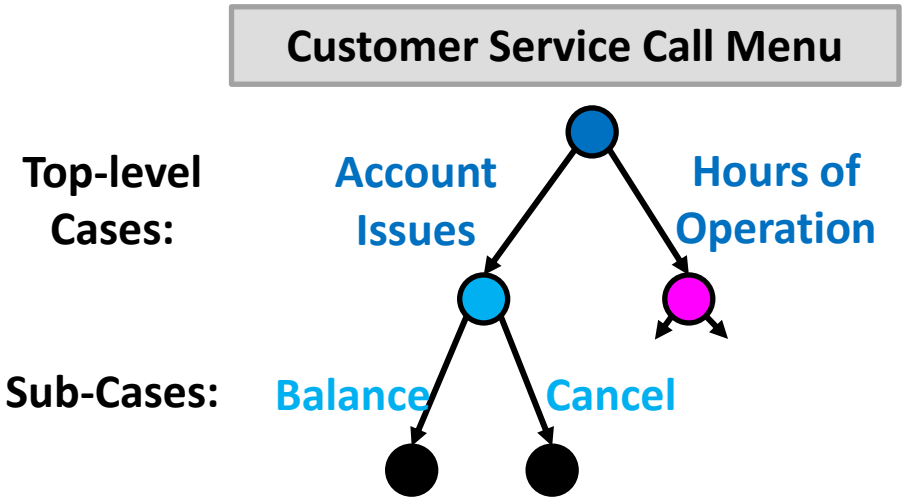
- **Name**: Subcase
- **Description**: Further divide one case into one or more subcases
- **Structure**: Nested 'if' statements

```
if( /* Condition 1 */ )
{
   // Case 1 code

   if( /* Subcondition 1a */ ) {
      // Subcase 1a code
   }
   else {
      // Subcase 1b code
   }

}
else if( /* Condition 2 */ )
{
   // Case 2 code

   if( /* Subcondition 2a */ ) {
      // Subcase 2a code
   }
}
```



Initial Exercises

- extracredit
- stoplight
- stoplight3
- nestedec

COMPOUND CONDITIONS AND LOGICAL OPERATORS

Logical Operators

- We can create compound conditions by using the logical AND, OR, and NOT operator

Operator(s)	Meaning	Example
&&	AND	`if( (x==0) && (y==0) )`
||	OR	`if( (x < 0) || (y < 0) )`
!	NOT	`if( !x )`

Logical AND, OR, NOT

- The following tables show how the logical operations are evaluated under any set of values
- AND:
  - All inputs must be true for resulting expression to be true
  - If even one is false, the condition is fails (false)
- OR:
  - If any input is true the condition evaluates to true

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

Order of Evaluation (Precedence)

- Precedence (Order of Operations)
  - Highest = ! (NOT)
  - Next = && (AND)
  - Lowest = || (OR)
- Better strategy:
  - Explicitly parenthesize everything

```
int main()
{
  int a, b, c;
  /* Some code that sets a, b, c */

  if( a>0 && !(b<5) || c==0)
  {
    /* Some code */
  }
}
```

```
int main()
{
  int a, b, c;
  /* Some code that sets a, b, c */

  if( (a>0 && (!(b<5))) || c==0)
  {
    /* Some code */
  }
}
```

Digging Deeper: Structural-OR and AND

```
if( level == "freshman" ) {
   cout << "Underclassmen";
}
else if( level == "sophomore" ) {
   cout << "Underclassmen";
}
else if( level == "junior" )  {
   cout << "Upperclassmen";
}
else if( level == "senior" ) {
   cout << "Upperclassmen";
}
```

```
if( level == "freshman" ||
    level == "sophomore" )
{
  cout << "Underclassmen";
}
```

- **if..else if** performing the same action could be replaced with an **OR**
  - Underclassmen if freshman OR sophomore

```
if( category == "manager" )  {
   bonus = .1*salary; // 10% bonus
   if( title == "ceo" ) {
      bonus += 1000; // 1000 extra
   }
}
else {
{
   bonus = .05*salary; // 5% bonus
}
```

```
if( category == "manager" &&
    title == "ceo" )
{
    bonus = 0.1 * salary + 1000;
}
```

- Nested **if** statements form a kind of **AND** relationship.

Common Mistakes 1

- Using assignment operator (=) rather than equality check operator (==)
  - If you accidentally use '=', it will convert the assigned value to a Boolean
- Using multiple `if` statements rather than `if..else` or `if..else if` statements
  - Without looking at the conditions, two 'if' statements imply both could be true while 'if..else' implies one or the other

```
int main()
{
  int x, y;
  cin >> x >> y;

  // Wrong!
  if( x = 0 )  { /* some code */ }
  // Right!
  if( x == 0 ) { /* some code */ }


  // Wrong!
  if(x != y) { x = 5;  }
  if(x == y) { y = 7; }

  // Right
  if(x != y) { x = 5;  }
  else       { y = 7; }
  return 0;
}
```

Common Mistakes 2

- All conditions must be formulated as a combination of comparisons of two values at a time

```
int main()
{
  int x, y;
  cin >> x >> y;

  // Wrong!
  if( 0 <= x <= 9 )
    { /* some code */ }
  // Right!
  if( (0 <= x) && (x <= 9) )
    { /* some code */ }


 // Wrong!
  if( x == 0 || 1 )
    { /* some code */ }
  // Right!
  if( (x == 0) || (x == 1) )
    { /* some code */ }
  return 0;
}
```

`if` statement Tips

- https://sourcemaking.com/refactoring/simplifying-conditional-expressions

What Goes In an `if` Condition

- What do we put in an `if` condition?
- ANYTHING.
  - The compiler will interpret what is in the parentheses as a Boolean
    - 0 = false
    - Non-0 = true

```
int main()
{
  int x, y, val;
  bool quit;
  cin >> x >> y >> val >> quit;
  // Uses Boolean result of comparison
  if( x > 0 )    { /* code */ }

  // Uses value of bool variable.
  //  Executes if quit == true.
  if( quit )     { /* code */ }

  // Interprets number as a bool
  //  Executes if val is non-zero
  if( val )      { /* code */ }

  // Interprets return value as bool
  //  Executes if the min is non-zero
  if( min(x,y) ) { /* code */ }

  return 0;
}
```

Order of Evaluation (Precedence)

- Precedence (Order of Operations)
  - Highest = ! (NOT)
  - Next = && (AND)
  - Lowest = || (OR)
- Better strategy:
  - Explicitly parenthesize everything

```
int main()
{
  bool a, b, c;
  /* Some code that sets a, b, c */

  if( a && !b || c)
  {
    /* Some code */
  }
}
```

```
int main()
{
  bool a, b, c;
  /* Some code that sets a, b, c */

  if( (a && (!b)) || c)
  {
    /* Some code */
  }
}
```

Exercises

- taxbrackets
- instock
- bill2law
- rps

Comparing Floating Point (Doubles) Numbers

- doubles effectively represent a number in binary with a technique similar to scientific notation ($+7.54*10^5$)
- However performing operations on doubles often leads to slight rounding errors
  - One result yields $+7.541*10^5$ while another yields $+7.539*10^5$
- This makes comparison difficult
  - Thus we rarely check for exact equality
  - Instead, we take the different of the two numbers and take the absolute value (`abs()` in `<cmath>`) and see if it is within a small epsilon

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

int main()
{
  double x, y;
  x = 1.0 / 10;
  y = 0.1;

  // Wrong!
  if( x == y )
    { /* some code */ }
  // Right!
  if( abs(x-y) < 1e-6 )
    { /* some code */ }

  return 0;
}
```

PROBLEM SOLVING IDIOMS

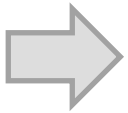
Look-up Table Idiom

- Consider any data that can be broken into **mutually exclusive cases**, such as
  - A table where each row maps some input to an output
- When cases are **mutually exclusive** we can use an '`if..else if..else`' statement to code each case separately

```
if( /* Condition 1 */ )
{
  // Case 1 code
}
else if( /* Condition 2 */ )
{
   // Case 2 code
}
else if( /* Condition 3 */ )
{
   // Case 3 code
}
else {  /* Default */
   // Default code
}
```

Look-up Table Structure

Weather	Dress
Hot	T-shirt
Mild	Long Sleeves
Cold	Sweater



```
if( weather == "hot" ) {
  clothing = "t-shirt";
}
else if( weather == "mild" ) {
  clothing = "long sleeves";
}
else {  /* Default */
  clothing = "sweater";
}
```

Look-up Table Idiom Examples (2)

- **Example(s)**

```
if( level == "freshman" )
{

}
else if( level == "sophomore" )
{

}
else if( level == "junior" )
{

}
else if( level == "senior" )
{

}
else {  /* Others/Errors */

}
```

```
int p1Score, p2Score;

/* Set variables */

if( p1Score > p2Score) {
  cout << "Player 1 wins";
}
else if(p1Score < p2Score){
  cout << "Player 2 wins";
}
else {
  cout << "Tie";
}
```

Decision Tree Exercise

- Write a program that asks a user to secretly pick one of the (former ☹) Pac-12 schools and then asks questions (via cout and cin) to determine ***whether the school is in the Pac-12 North or South***.
  - You may not ask directly what school they chose but must use indirect questions. Try to minimize the number of questions you need to ask in the WORST CASE (i.e. don't just ask, "Did you pick Buffalos [y/n]?", "Did you pick the Cougars [y/n]?"). Questions are not limited to y/n style.

Pac12-North
Wash. Huskies
Cougars (WSU)
Oregon Ducks
Beavers (OSU)
Stanford Cardinal
Cal Bears

Pac12-South
Utah Utes
Colorado Buffalos
Arizona Wildcats
Sun Devils (ASU)
USC Trojans
UCLA Bruins

Portfolio 1

- Write a program that uses if statements to determine the property of some secretly chosen item by the user from some limited set of items
- **Requirements**: Your program must include
  - Asks multiple input questions of the user
  - An if statement that requires at least three (we'd prefer 4 but will settle for 3) branches (i.e. if..else if..else)
  - Have at least 2 (we'd prefer 3 but will settle for 2) levels of nested if statements
- Style of program that may work
  - A Decision Tree (Identification of a certain class of people, animals, numbers, USC majors, music groups, etc.)
  - Some kind of escape room / choose your own adventure
  - Escape room type of game where you would (1) print out a description of each room, (2) choose from a numbered list of which door to enter, (3) have options to battle characters in the room or possibly lose the game
  - Some kind of Pokemon battle