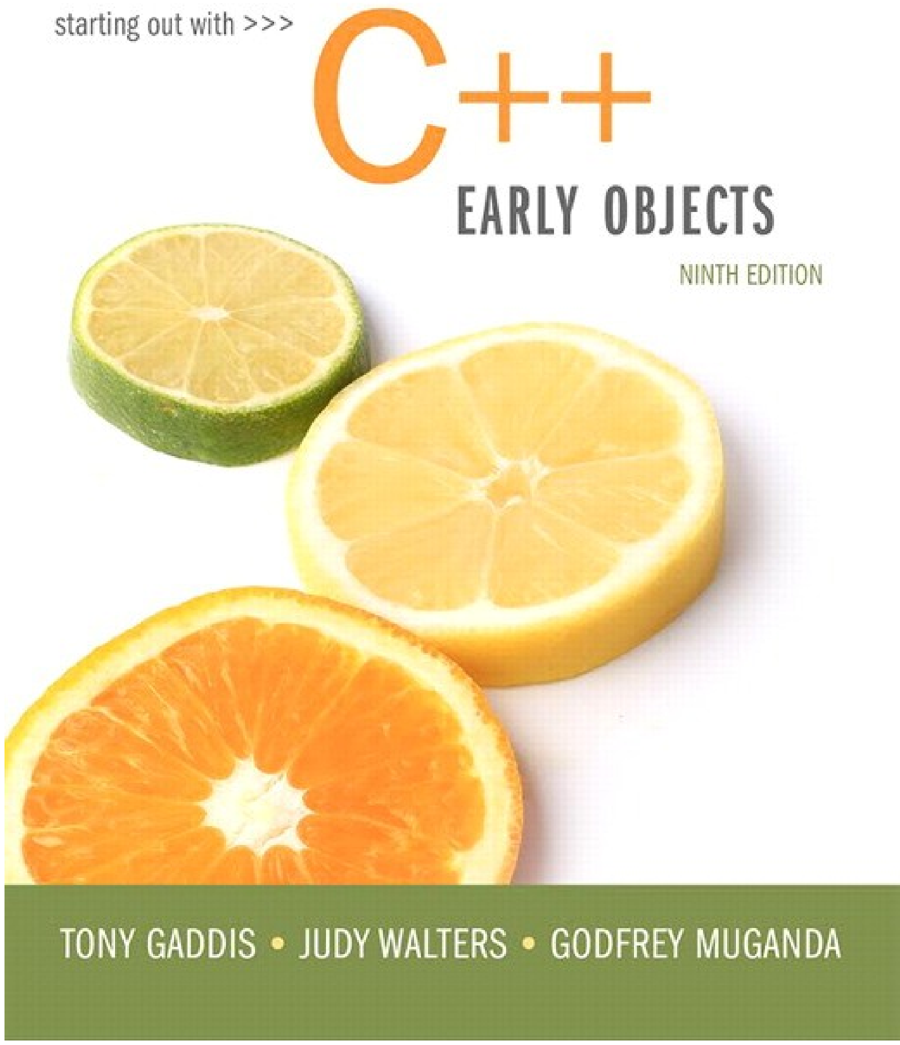
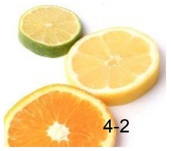
Week #04 Slides:

Making

Decisions



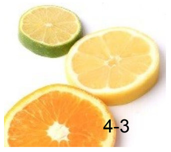
Topics



4.1 Relational Operators

- Are used to compare numeric and **`char`** values to determine relative order
- Operators:

`>`	Greater than
`<`	Less than
`>=`	Greater than or equal to
`<=`	Less than or equal to
`==`	Equal to
`!=`	Not equal to



Relational Expressions

- Relational expressions are Boolean (*i.e.*, evaluate to **`true`** or **`false`**)
- Examples:

  `12 > 5` is **`true`**

  `7 <= 5` is **`false`**

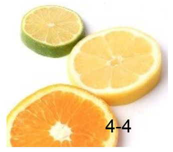
  if **`x`** is 10, then

  `x == 10` is **`true`**,

  `x <= 8` is **`false`**,

  `x != 8` is **`true`**, and

  `x == 8` is **`false`**

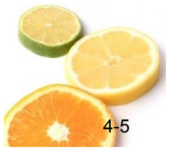


Relational Expressions

- The value can be assigned to a variable

```
bool result = (x <= y);
```

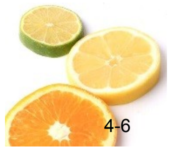
- Assigns **`0`** for **`false`**, **`1`** for **`true`**
- Do not confuse **`=`** (assignment) and **`==`** (equal to)



Hierarchy of Relational Operators

Operator	Precedence
`>   >=   <   <=`	Highest
`==    !=`	Lowest

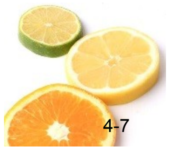
Use this when evaluating an expression that contains multiple relational operators



4.2 The `if` Statement

- Supports the use of a decision structure, giving a program more than one path of execution
- Allows statements to be conditionally executed or skipped over
- It models the way we evaluate real-life situations

  "If it is cold outside,
  wear a coat and wear a hat."



Format of the `if` Statement

```
if (condition)
{
   statement1;
   statement2;
        …
   statementn;
}
```

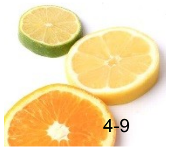
No ; goes here

they go here

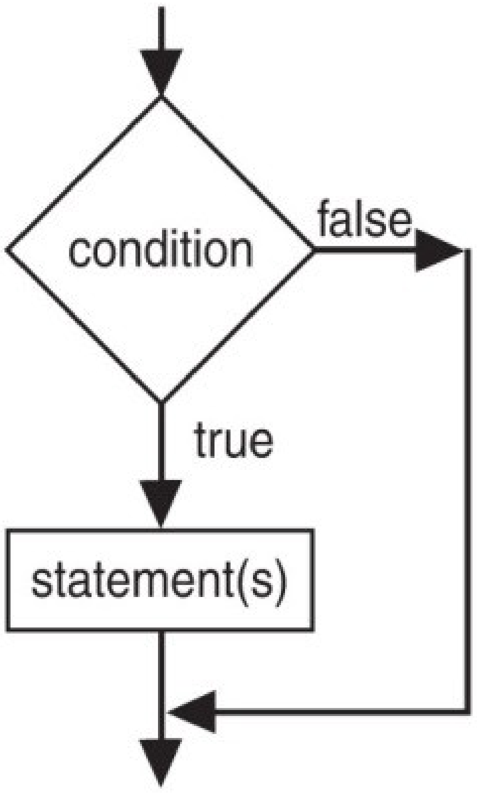
The block of statements inside the braces is called the <u>body</u> of the `if` statement. If there is only 1 statement in the body, the `{ }` may be omitted.

How the `if` Statement Works

- If **(*condition*)** is **`true`**, then the ***statement(s)*** in the body are executed.
- If **(*condition*)** is **`false`**, then the ***statement(s)*** are skipped.



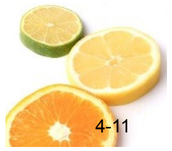
`if` Statement Flow of Control



Example `if` Statements

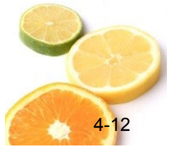
```
if (score >= 60)
   cout << "You passed." << endl;

if (score >= 90)
{
   grade = 'A';
   cout << "Wonderful job!" << endl;
}
```



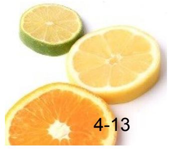
`if` Statement Notes

- `if` is a keyword. It must be lowercase
- `(`***`condition`***`)` must be in ( )
- Do not place `;` after `(`***`condition`***`)`
- Don't forget the { } around a multi-statement body
- Don't confuse = (assignment) with == (comparison)



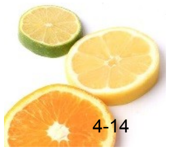
`if` Statement Style Recommendations

- Place each ***`statement;`*** on a separate line after **`(`*****`condition`*****`)`**
- Indent each statement in the body
- When using { and } around the body, put { and } on lines by themselves



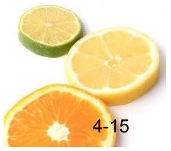
What is `true` and what is `false`?

- An expression whose value is 0 is considered `false`.
- An expression whose value is non-zero is considered `true`.
- An expression need not be a comparison – it can be a single variable or a mathematical expression.



Flag

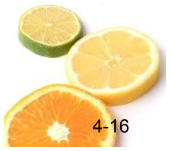
- A flag is a variable that signals a condition
- It is usually implemented as a **`bool`**
- Meaning:
  - **`true`**: the condition exists
  - **`false`**: the condition does not exist
- The flag value can be both set and tested with **`if`** statements



Flag Example

Example:

```
    bool validMonths = true;
          …
    if (months < 0)
        validMonths = false;
          …
    if (validMonths)
        monthlyPayment = total /
    months;
```



Integer Flags

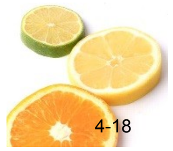
- Integer variables can be used as flags
- Remember that 0 means false, any other value means true

```
 int allDone = 0;  // set to false
          …
  if (count > MAX_STUDENTS)
       allDone = 1;  // set to true
          …
  if (allDone)
       cout << "Task finished";
```

4.3 The `if/else` Statement

- Allows a choice between statements depending on whether `(`*`condition`*`)` is **`true`** or **`false`**
- Format:

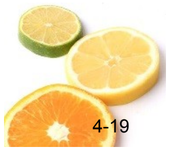
```
if (condition)
{
    statement set 1;
}
else
{
    statement set 2;
}
```



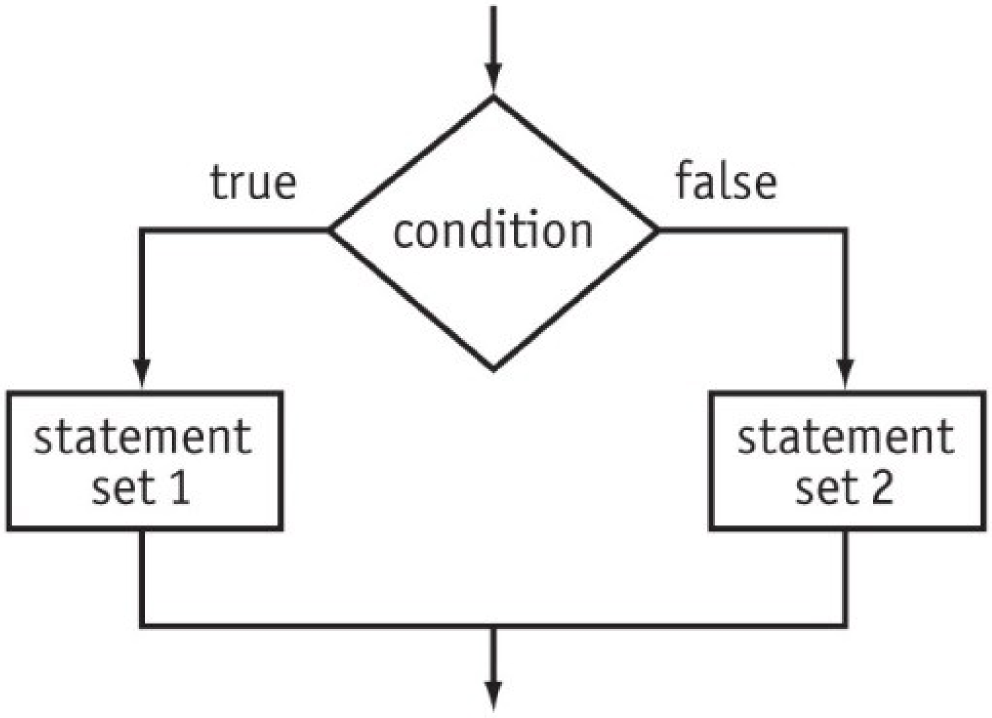
How the `if/else` Works

- If `(`***`condition`***`)` is **`true`**, ***`statement set 1`*** is executed and ***`statement set 2`*** is skipped.

- If `(`***`condition`***`)` is **`false`**, ***`statement set 1`*** is skipped and ***`statement set 2`*** is executed.



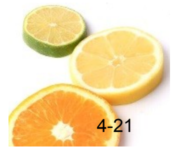
`if/else` Flow of Control



Example `if/else` Statements

```
if (score >= 60)
    cout << "You passed.\n";
else
    cout << "You did not pass.\n";

if (intRate > 0)
{   interest = loanAmt * intRate;
    cout << interest;
}
else
    cout << "You owe no interest.\n";
```



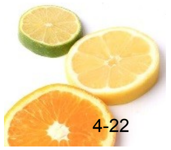
`if` vs. **`if/else`**

If there are two conditions and both of them can be true or both can be false, then use two **`if`** statements:

```
if (num > 0)
   cout << num << " is positive\n";
if (num %2 == 0)
   cout << num << " is even\n";
```

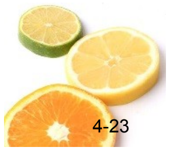
If the two conditions cannot both be true, then a single **`if/else`** statement can work:

```
if (num %2 == 0)
   cout << num << " is even\n";
else
   cout << num << " is odd\n";
```



Comparisons with floating-point numbers

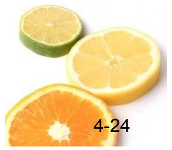
- It is difficult to test for equality when working with floating point numbers.
- It is better to use
  - greater-than or less-than tests, or
  - test to see if value is very close to a given value



4.4 The **`if/else if`** Statement

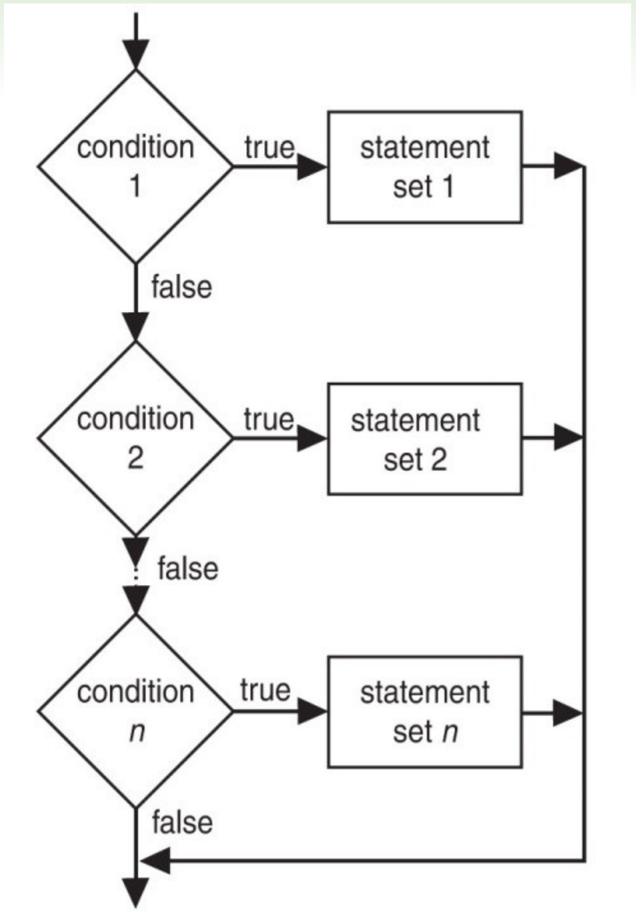
- This is a chain of **`if`** statements that test in order until one is found to be true
- This also models thought processes

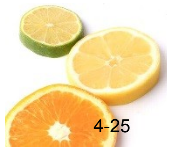
  "If it is raining, take an umbrella,
  else, if it is windy, take a hat,
  else, if it is sunny, take sunglasses."



`if/else if` Format

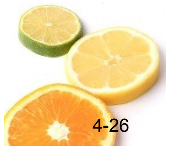
```
if (condition 1)
{    statement set 1;
}
else if (condition 2)
{    statement set 2;
}
            . . .
else if (condition n)
{    statement set n;
}
```





Using a Trailing `else`

- Is used with a set of `if/else if` statements
- It provides a default statement or action that is performed when none of the conditions is true
- It can be used to catch invalid values or handle other exceptional situations

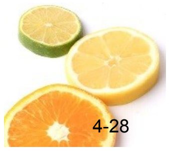


Example **`if/else if`** with Trailing **`else`**

```
if (age >= 21)
   cout << "Adult";
else if (age >= 13)
   cout << "Teen";
else if (age >= 2)
   cout << "Child";
else
   cout << "Baby";
```

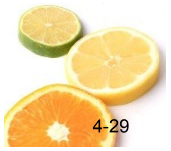
4.5 Menu-Driven Program

- Menu: list of choices presented to the user on the computer screen
- Menu-driven program: program execution is controlled by user selecting from a list of actions
- A menu-driven program can be written using **`if/else if`** statements



Menu-driven Program Organization

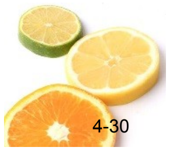
- Display a list of numbered or lettered choices for actions.
- Input user's selection of number or letter
- Test the user selection in `(condition)`
  - if a match, then execute code to carry out desired action
  - if not, then test with next `(condition)`



4.6 Nested `if` Statements

- An `if` statement that is part of the `if` or `else` part of another `if` statement
- This can be used to evaluate > 1 data item or to test > 1 condition

```
if (score < 100)
{
    if (score > 90)
        grade = 'A';
}
```

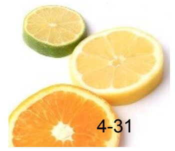


Notes on Coding Nested **`ifs`**

- An **`else`** matches the nearest previous **`if`** that does not have an **`else`**

```
if (score < 100)
   if (score > 90)
      grade = 'A';
   else ...   // goes with second if,
              // not first one
```

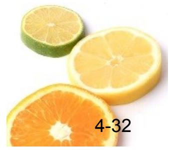
- Proper indentation aids understanding



4.7 Logical Operators

Are used to create relational expressions from other relational expressions

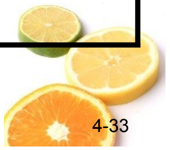
Operator	Meaning	Explanation
&&	**AND**	New relational expression is true if both expressions are true
||	**OR**	New relational expression is true if either expression is true
!	**NOT**	Reverses the value of an expression; true expression becomes false, false expression becomes true



Logical Operator Examples

```
int x = 12, y = 5, z = -4;
```

`(x > y) && (y > z)`	`true or 1`
`(x > y) && (z > y)`	`false or 0`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`



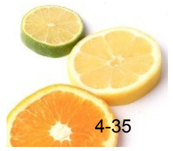
Logical Operator and `bool` Variables

- Logicial operators can be used with **`bool`** variables as well as expressions that evaluate to **`true`** or **`false`**.
- Ex:

```
bool done = false;
if ((!done) && (count < 6))
{
. . .
}
```

Short-Circuit Evaluation

- If an expression using the `&&` operator is being evaluated and the subexpression on the left side is **`false`**, then there is no reason to evaluate the subexpression on the right side. It is skipped.
- If an expression using the `||` operator is being evaluated and the subexpression on the left side is **`true`**, then there is no reason to evaluate the subexpression on the right side. It is skipped.



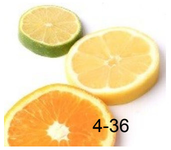
Logical Precedence

Highest	**!**
	&&
Lowest	**||**

Example:

$$\underbrace{\texttt{(2 < 3) || }\underbrace{\texttt{(5 > 6) \&\& (7 > 8)}}}$$

is true because AND is evaluated before OR



More on Precedence

Highest	arithmetic operators
↓	relational operators
Lowest	logical operators

Example:

$$\underbrace{\underbrace{8 < \underbrace{2 + 7}_{}}_{} \mathbin{||} \underbrace{5 == 6}_{}}_{} \qquad \text{is true}$$

Checking Numeric Ranges with Logical Operators

- Used to test if a value is within a range

```
if (grade >= 0 && grade <= 100)
    cout << "Valid grade";
```

- You can also test if a value lies outside a range

```
if (grade <= 0 || grade >= 100)
   cout << "Invalid grade";
```

- Note that you cannot use mathematical notation

```
if (0 <= grade <= 100) //Doesn't
                       //work!
```