

CS162

Introduction to Computer Science

Welcome

Welcome to CS162!

- Today we will cover:

1. Syllabus
2. Demonstrate D2L
3. Discuss what Assignments will be like
4. Due dates
 1. Go to the Course Outline
 2. Algorithms – can't be late!!
 3. Programs – on time and late turn in dates available
5. Contracts for lecture and lab

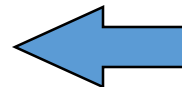
- Important:

1. Visit d2l.pdx.edu
2. Attendance is required!
3. Watch class lectures!!!
4. Participate with D2L discussions
5. Attend all labs
6. Keep up with the readings!
7. Always practice with small programs for each new concept.

Staying Informed

- Visit Office hours. They are posted on D2L's home page and the course syllabus.
- Weekly, check D2L's home page for any current announcements!
- Read your PSU email (pdx.edu) weekly for important information and reminders.
- Read discussion forums on D2L to stay informed.
- Seek assistance before it becomes too late
 - Email your Professor or Instructor
 - Visit Office Hours
 - Join Technical Assistant Office hours Monday through Friday
 - Be involved in class!

Programming



- First, get a CS or MCECS account
 - You will need this for the first lab
- Next, learn how to login
- Learn the fundamentals of working at the linux shell prompt
 - Make a directory (`mkdir directory_name`)
 - Change directories (`cd directory_name`)
 - Find out what directory you are in (`pwd`)
 - Go back one directory (`cd ..`)
- Your CS or MCECS account is where you will work for...
 - Labs (using `quizzor1.cs.pdx.edu` as the host)
 - Programming assignments (using `linux.cs.pdx.edu` as the host)
 - And for proficiency demonstrations
- Memorize your password. You will frequently need this!

Using ssh

- To login to our linux systems, you will need a secure shell program
- These are automatically available on Mac OS and linux systems, using “terminal” applications
- For PC users running Windows, you will need to find a free program called “ssh” or “putty” to download
 - Make sure to find a free site from which to download it.
 - It is a secure shell program and allows your computer to act as a ‘dumb terminal’ for logging in remotely.
 - You start up ssh by double clicking on it.
 - Then, connect to: `linux.cs.pdx.edu` as the host name
 - Use your CS or MCECS login name and password

Entering in Programs

- To enter in a program on linux, you will need to use an editor
- The available editors are:
 - pico, nano - these are good to start with as they provide menus
 - vi, vim, emacs – these are more advanced and require the use of single letter keys to navigate. By CS163 we will need to become proficient using one of these editors!
 - To use pico to type in a program, type at the unix prompt:
pico prog1.cpp <enter>

When done hit **control o** at the same time to write it out and then **control x** to exit.

- **Enter in a Program.** Start with something simple just to make sure it all works right!

Compiling Programs on Linux

- **Compile** your C++ source code file. The command to do this is:

g++ prog1.cpp

- If your program successfully compiles, it will produce a file named 'a.out' in your directory. Otherwise, you will need to correct syntax errors before continuing - by using pico again: **pico prog1.cpp**
- One common mistake is to make typos!
- **Run** your program by typing:

./a.out

CS162

Introduction to Computer Science

Learning C++ Syntax

Introduction to C++

- C++ is based on the C language
 - It started as a superset of C
 - to retain the power and flexibility of C
 - to deal effectively with hardware/software interfaces and low level programming
- Developed in early 1980's
 - by Bjarne Stroustrup to simplify complex event driven simulations
- A major design goal was to keep C++ compatible with C
 - C programmers need to “unlearn” some of the techniques used in programming
 - Improvements were made which keep C++ from being 100% compatible
 - C++ inherits the basic language mechanisms of C (operations, statements, loops, pointers, arrays) but then adds features to overcome the problems in C
 - But, if you are a C programmer, you still need to learn and use C++'s constructs!

Introduction to C++

as we move from CS162 to CS163

- The most significant impact of C++
 - is that it is flexible in adapting to many different programming styles:
 - procedural abstraction
 - modular abstraction
 - data abstraction (defining new data types)
 - object-oriented programming
- Realize that when we use C++ we may (or may not) be performing OOP
 - it is not a “pure” OOP language (hybrid)
- As we think about solving problems with abstractions
 - using data abstraction or OOP
 - we must develop standard reusable objects
 - which requires thinking about problems from a broader perspective
- **First, this requires building a solid foundation in the language fundamentals...**
 - Which is our goal in CS162!!

Getting Started:

The format of a C++ program

- All C++ programs have the following “form”

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

```
/*   A Block of header comments   */
//   Your name, class, purpose of the program
```

```
int main()
{
    //program body goes here...
    return 0;
}
```

Sample Program: *notice the style!*

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

// *****
//   Karla S. Fant
//   Purpose of this program is to convert
//   inches entered in by the user into
//   millimeters and display the results
//   *****
int main()
{
    float inches, mm;        //variable definitions
    cout <<"Welcome! We will be converting"
         <<" inches to mm today" <<endl;

    //Get the input (prompt, read)
    cout <<"please enter the number of inches"
         <<" that you wish to convert: ";

    cin >> inches;           //read the # inches
```

Sample Program

```
//Echo what was entered
cout <<"You entered: " <<inches <<"in"
    <<endl;

//Convert inches to millimeters
mm = 25.4 * inches;

//Display the results
cout <<inches <<"in converts to " <<mm
    <<mm <<"mm" <<endl;

return 0;
}
```

Preprocessor Directives

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

- The `#include` is a preprocessor directive
- `#include` allows our programs to copy the contents of some other file (`iostream`, for example) and have it behave as if that file was typed at the beginning of our programs!
- `iostream` allows our programs to perform input from the keyboard (standard in) and output to the screen (standard out)
- All I/O should be done with the `iostream` library which means at the beginning of every program you will have:
 - `#include <iostream>`
 - `using namespace std;`

Preprocessor Directives

- You may have encountered `#include <iostream.h>`
- This is an older `iostream` library that placed all identifiers in the global namespace, making for global namespace “pollution”
- Without the `.h`, all identifiers are placed in a grouping, called the `std` namespace.
 - If you don’t say “`using namespace std`”, then all of the identifiers must be qualified with the `std::` grouping name and the scope resolution operator. Doing this will reduce the readability of your programs!
 - To output you would have to say: `std::cout << “Welcome CS162”;`
 - Or you can make the `iostream` identifiers global for your program, by saying `using namespace std`; This will bring everything in the `iostream` library and make it global for your program so that all functions can use those identifiers (like `cin` and `cout`)
 - Or you can say “`using namespace std`” locally in a function and only bring in those identifiers as local for that particular function avoiding the side effects of global variables!

Header Comments...

```
//      ****  
//  Karla S. Fant
```

- These are **in line** comments and provide for documentation in our programs
- Once encountered, the rest of the line is taken as documentation.
- If you want to surround them by asterisks -- MAKE SURE to place at least one space between the // and the asterisks.... otherwise, your entire program will be mistaken as a comment!?!?!

(Different Kind of Comment...)

```
/*      ****  
      Karla S. Fant  
      ****      */
```

- This type of comment is best used when writing a large block of comments. They begin with a `/*` and end when you type a `*/`
- If you forget the ending `*/` your entire program is taken as a comment!!!
- I recommend placing a space between the `/*` and any other asterisks....

Variable Definitions

```
//Define variables  
float inches;           //to save # inches  
float mm;               //to save the result
```

- Everyone should already know why we need variables
 - To store information needed by our program
- How are they defined?
data_type variable_name
- What is a data type and why is it important?
- What kind of variable names can we use?

Variable Definitions

- What are variables?
 - Allocate Memory to store data
- How are they defined?
data_type variable_name
- What data types are available?
 - Real numbers: float, double
 - Whole numbers: int, short, long
 - Single character data: char
 - Boolean (true/false): bool
- Notice “string” is not listed above
 - Because a string is a user defined data type and not built into the language.
 - We will use arrays of characters to represent string type data
- What kind of variable names can we use?
 - Variables must start with a letter, be any combination of letters, digits, or underscores.
 - Dollars signs may not be used as part of a variable name

Variable Definitions: Rules

- Variables must be defined before they can be used
- Local variables are garbage unless explicitly initialized

- `int count;`
- `int i, j;`
- `char initial = 'k';`
- `char response ('Y');`
- `int Total_Length;`
- `int total_length;`
- `int TotalLength;`
- `float cost = 0.0;`

- Variables can be defined using a comma separated list
- Notice there are two different ways to initialize variables
- Pick a consistent naming convention!
- Avoid unnecessary type conversions.

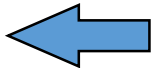
Conventions

- Be consistent with Naming Conventions!
- Avoid Global Variables
 - A global is one defined outside of a function
- Avoid defining Local Variables inside of inner blocks
 - We recommend defining all variables at the beginning of the function
 - This convention avoids unnecessary conflicts by redefining the same variable in multiple different blocks
- Global constants are Great!
`const int SIZE = 42;`
 - A constant is one that cannot change in value
 - Provides self documentation by specifying a meaningful name for a value
 - Convention has us use all upper case characters to easily detect when a constant is being used within a program!

Displaying Output

```
cout << "message" <<endl;
```

- Pronounced “see out”
- << is the **insertion operator**
- **endl** ← this ends in a lower case **L**
- Think of << as an arrow. The message you want to display is being sent to the OUTPUT device in the direction of the arrow:

output_device  message in double quotes

Displaying Output

```
cout << "message";  
cout << endl;
```

- The insertion operator will not add any extra spacing
- If you want a space, you need to have a space within the double quotes

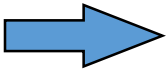
```
cout << endl << endl;
```

- This will output a blank line to the standard output device

Receiving Input

```
cin >> inches;
```

- Pronounced “see in”
- >> is the **extraction operator**
- Think of >> as an arrow. The data you want to receive is coming from the input device in the direction of the arrow and being saved in memory:

input_device  variable_name

The Extraction Operator

```
cin >> inches;
```

- The extraction operator skips leading whitespace, reads in the appropriate information that is next in the input stream (also known as the input buffer)
- If the wrong data type is in the input buffer, no input will happen and the data will need to be removed from the input buffer to proceed.
- In some situations, the wrong data type will cause an error state in the iostream library to be set which may need to be cleared by using: `cin.clear();`

Removing data from the Input Stream

```
cin.ignore();
```

- Removes one character

```
cin.ignore(100, '\n');
```

- Removes up to 100 characters or until a newline is removed

```
cin.ignore(100);
```

- DO NOT USE
- Ignores 100 characters or until end-of-file (control d on linux) occurs. Essentially it will ignore 100 characters!!

Other interesting Input Stream functions

```
cin.get();      or char_variable =  
cin.get();
```

- Reads in a single character
- Returns it as well
- Reads and returns whitespace (doesn't skip it)

```
char_variable = cin.peek();
```

- Peeks into the input buffer and returns the next character
- It does not modify the input buffer
- Reads and returns whitespace (doesn't skip it)

Assignment Operation

```
//Step #3 Convert inches to millimeters  
mm = 25.4 * inches;
```

- Multiplication requires the asterisk
 - can't leave it out like we do in math
 - 3x would be written 3 * x in C++
- = is the assignment operation
 - takes the value on the right and saves it in the memory for the variable on the left

Arithmetic Operations

- Assignment Operator:
 - =
- Arithmetic Operators:
 - + - * / % (mod operator)
- Compound Arithmetic Ops:
 - += -= *= /= %=
- Integer Division:
 - Quotient: $a = b/c$;
 - Remainder: $a = b\%c$;
- Increment: ++
 - Adds one
- Decrement: --
 - Subtracts one
- Form:
 - Variable = mathematical expression
- Examples:
 - $a = a + 10$; or $a += 10$;
 - $a = a + b * 10$; or $a += (b * 10)$;
- ILLEGAL:
 - $a + 10 = b$;
 - $a + 10$ is an “rvalue”; the result is stored in a temporary and cannot be on the left hand side of an assignment operation

Arithmetic Expressions

- Let's take a look on operations that can be performed on real data:

result = +realvariable <== No change

result = -realvariable <== Takes the negative

result = a+b <== Takes the sum

result = a-b <== Takes the difference

result = a*b <== Takes the product

result = a/b <== Performs division

Arithmetic Expressions

- Other Interesting Operators for...
 - Compound Assignment

$\ast =$ $/ =$ $+ =$ $- =$

result $+ =$ 10

result = result+10

result $\ast =$ x+y

result = result \ast (x+y)

result $/ =$ x+y

(x+y) result



result = result/(x+y)

Arithmetic Expressions

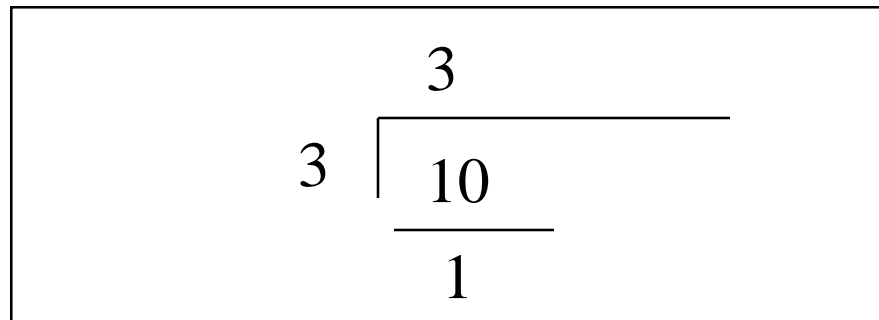
- One more operator...
 - Integer division

/ % (remainder)

```
int number;
```

```
number = 10/3;            //answer is 3
```

```
number = 10%3;           //answer is 1
```



A hand-drawn diagram illustrating the integer division of 10 by 3. It is enclosed in a rectangular box. On the left, the number 3 is written. To its right is a vertical line that forms part of a division symbol. To the right of this vertical line is a horizontal line, and below it is another horizontal line. Between these two horizontal lines is the number 10. Below the bottom horizontal line is the number 1. Above the top horizontal line is the number 3. This represents the calculation 10 divided by 3, with a quotient of 3 and a remainder of 1.

Operator Precedence

- One operator cannot follow another
- $(2.5 + -3.6)$ is illegal....you can do this by using parentheses:
 $(2.5 + (-3.6))$
- With parentheses, the operations within parens is performed first. When parens are nested...the innermost set of parens is performed first:

$$2.0 + (3.0 * (4.0 - 1.0))$$

is the same as

$$2.0 + (3.0 * 3.0)$$

which is 11.0

Operator Precedence

- Watch out for ambiguous expressions when parens are not used:
 - What does $3.0+1.0*4.0$ mean?

7.0? Or, 16.0?

- Since no parens are given, we go by the order of precedence of operators. $*$, $/$, $\%$ have a higher precedence....than $+$ and $-$.
- So, the answer to above is 7.0!

Operator Precedence

- What about, $3.0 * 2.0 - 7.0 / 2.0$?
1. First take the highest priority ($*$ and $/$)...and go left to right.
 2. So, first multiply $3.0 * 2.0 \Rightarrow 6.0$
 3. Then divide $7.0 / 2.0 \Rightarrow 3.5$
 4. So, we have so far $6.0 - 3.5$
 5. Which is 2.5

Increment/Decrement Ops

- There are two more operators that add or subtract 1

`++i` means `i = i + 1`

`--i` means `i = i - 1`

- These are used in their prefix form
- Use the prefix form unless the problem needs the postfix behavior
- They can also be used in a postfix form:

`i++`

`i--`

First fetch the value of the variable and store it in a temporary. Then add 1 to the variable. Finally, the residual value is the value of the temporary which is the “pre incremented” value of the variable

Postfix Increment/Decrement

`i++` means:

- 1) Residual value is the current value of the variable
- 2) Then, increment the variable by 1

3) For example: `int i = 100;`
`cout << i++;`

Displays 100 not 101!

But the value of `i` is 101

- 4) We use postfix when we need to progress the variable to the next but use the current value of the variable (pre-incremented value) at the same time.
- 5) If all you need to do is to add 1 or subtract 1, then use the prefix form of these operators.

Increment/Decrement

- More examples:

```
int i;  
cin >> i;  
i++;  
++i;  
cout << i;
```

input	output
50	52
100	102

Increment/Decrement

- More examples:

```
int i, j;  
cin >> i;  
j = i++;  
cout <<j <<" ,";  
j = ++i;  
cout <<j;
```

input	output
50	50, 52
100	100, 102

Introduction to C++

I/O Formatting

Next, to Format our Output

- We must learn about **precision**
- By default, real numbers are displayed with no more than 6 digits, plus the decimal point
- This means that 6 significant digits are displayed in addition to the decimal point and a sign if the number is negative

Default Precision -- Examples

```
float test;  
cout << "Please enter a real number";  
cin >> test;  
cout << test;
```

<u>Input</u>	<u>Resulting Output</u>
1.23456789	1.23457
10.23456789	10.2346
100.23456789	100.235
1000.23456789	1000.23
100000.23456789	100000

To Change Precision

```
float test;  
cout << "Please enter a real number";  
cout.precision(3); //3 instead of 6!!  
cin >> test;           cout << test;
```

<u>Input</u>	<u>Resulting Output</u>
1.23456789	1.23
10.23456789	10.2
100.23456789	100
10000.23456789	1e+04
	(Exponential notation)

Another way to do this...

```
#include <iomanip.h>  
float test;  
cout << "Please enter a real number";  
cin >> test;  
cout << setprecision(3) << test;
```

- setprecision is a manipulator
- To use it, we must include the `iomanip.h` header file
- There is no difference between
 `cout.precision(3)` and `cout << setprecision(3)`

What is “width”?

- The width of a field can be set with:

```
cout.width(size);
```

- If what you are displaying cannot fit, a larger width is used
 - to prevent the loss of information
- Important
 - Width is only in effect for the next output

How does width work...

```
float test;  
cout.precision(4); cout.width(10);  
cin >>test;      cout << test;  
                  cout <<endl <<test;
```

Input

1.23456789

Resulting Output

1	.	2	3	5
---	---	---	---	---

1.235

Another way to do this...

```
#include <iomanip.h>
float test;
cout.precision(4);
cin >> test;
cout <<setw(10) << test;
cout <<endl <<test;
```

Input

1.23456789

Resulting Output

1	.	2	3	5
---	---	---	---	---

1.235

Trailing Zeros

- For real numbers, trailing zeros are discarded when displayed

<u>Input</u>	<u>Resulting Output</u>
• To display trailing zeros we use: <code>cout.setf(ios::showpoint);</code>	1.23
(for an precision of 3 or greater)	

Displaying Trailing Zeros

```
float test;  
cout.precision(4);  
cout.setf(ios::showpoint);  
cin >> test;    cout << test << endl;  
cout.unsetf(ios::showpos); //reset...  
cout << test;
```

<u>Input</u>	<u>Resulting Output</u>
1.2300	1.230 1.23

Displaying Dollars and Cents!

- There is another meaning to precision...

- if we put in our programs:

```
cout.setf(ios::fixed, ios::floatfield) ;
```

- then, subsequent precision applies to the number of digits after the decimal point!

```
cout.precision(2) ;    cout << test ;
```

1.2300

1.23

¹
²⁰
Input

¹
²
Resulting Output

Displaying Dollars and Cents!

- Since we ALSO want trailing zero displayed...do all three:

```
cout.setf(ios::fixed, ios::floatfield) ;  
cout.precision(2) ;  
cout.setf(ios::showpoint) ;  
cout << test;
```

1.2300

1.23

Input

Resulting Output

Introduction to C++

Selective Execution: Conditionals

Selective Execution

- Selective Execution:
 - if (conditional expression)
- Relational Operators:
 - < > <= >=
- Equality Operators:
 - == !=
- Logical Operators:
 - && - and
 - || - or
 - ! - not
- The keyword “if” must be lower case
- Parentheses are required.
- Put the conditional expression inside the parens
- Be careful to use the == to compare two items. The = is the assignment statement and will NOT compare!

if Statements have the form...

1) One alternative:

```
if (conditional expression)
    single C++ statement;
```

```
char selection;
cout <<"Is this correct? Y or N ";
cin >> selection;
if ('Y' == selection)
    cout <<"Your selection was correct"
        <<endl;
```

if Statements have the form...

2) Two alternatives:

```
if (conditional expression)
    single C++ statement;
else
    single C++ statement;
```

```
if ('Y' == selection)
    cout << "Great job!";
else
    cout << "The choice was not correct";
```

if Statements have the form...

- This means that either the first statement is executed when running your program OR the second statement is executed. BOTH sets of statements are NEVER used.
 - ONE OR THE OTHER!
- If the comparison is true - the first set is used;
- If the comparison is false - the second set is used;

if Statements have the form...

- When an if is encountered, the conditional expression is TRUE if it is **non zero**. In this case, the statement following the expression is executed.
- Otherwise, if the conditional expression evaluates to **zero** it means it is FALSE. In this case, if there is an else the statement following the else is executed.
- If there is no else then nothing is done if the conditional expression evaluates to **zero** (FALSE).

if Statements have the form...

3) Two or more alternatives:

```
if (conditional expression)
    single C++ statement;
else if (conditional expression)
    single C++ statement;
```

```
if ('Y' == selection)
    cout << "Great Job!";
else if ('N' == selection)
    cout << "Next time it will be better!";
```

Compound if statements...

- 4) You might want more than a single statement to be executed given an alternative...so instead of a single statement, you can use a **compound statement**

```
if (conditional expression)
{
    Many C++ statements;
}
```

```
else //optional
```

Example of if Statements

```
if (selection == 'm')
{
    cout <<"Enter the # inches: ";
    cin >>inches;
    mm = 25.4*inches;
    cout <<inches <<"in converts to "
         <<mm <<" millimeters" <<endl;
}
else //selection is not an 'm'
{
    cout <<"Enter the # millimeters: ";
    cin >>mm;
    inches = mm/25.4;
    cout <<mm <<"mm converts to "
         <<inches <<" inches" <<endl;
}
```

Conditional Expressions

- The comparison operators may be:
 - **Relational Operators:**
 - > for greater than
 - < for less than
 - >= for greater than or equal
 - <= for less than or equal
 - **Equality Operators:**
 - == for equal to
 - != for not equal to

Logical Operators

- There are 3 logical (boolean) operators:
 - && And (operates on two operands)
 - || Or (operates on two operands)
 - ! Not (operates on a single operand)
- && evaluates to true if both of its operands are true;
 - otherwise it is false.

Logical Operators

- `||` evaluates to true if one or the other of its operands are true;
 - it evaluates to false only if both of its operands are false.
- `!` gives the boolean complement of the operand.
 - If the operand was true, it results in false.

AND Truth Table

- **op1 && op2** results in:

op1	op2	residual value	
true	true	true	1
true	false	false	0
false	true	false	0
false	false	false	0

OR Truth Table

- **op1 || op2** results in:

op1	op2	residual value	
true	true	true	1
true	false	true	1
false	true	true	1
false	false	false	0

NOT Truth Table

- **!op1** results in:

op1	residual value	
true	false	0
false	true	1

Logicals in `if` Statements

- Now let's apply this to the `if` statements.
- For example, to check if our input is only an 'm' or an 'i'

```
char selection;  
cin >>selection  
if (selection != 'm' &&  
    selection != 'i')  
    cout <<"Error! Try again";
```

Logicals in `if` Statements

- Why would the following be incorrect?

```
if (selection != 'm' ||  
    selection != 'i')  
    cout << "Error! Try again";
```

- Because no matter what you type in (m, i, p, q) it will never be both an m and an i!
- If an m is entered, it won't be an i!!!!

Logicals in `if` Statements

- Let's change this to check if they entered in either an m or an i: (this is correct)

```
if (selection == 'm' ||  
    selection == 'i')  
    cout << "Correct!";  
else  
    cout << "Error. Try Again!";
```

Logicals in `if` Statements

- Now, let's slightly change this....

```
if (! (selection == 'm' ||  
      selection == 'i' ))  
    cout << "Error. Try Again!";
```

- Notice the parens...you must have a set of parens around the conditional expression

Common Errors

- | | |
|--|---|
| <ul style="list-style-type: none">• Assignment vs Equality<ul style="list-style-type: none">• <code>if (response = 'Y')</code>• Correct:
<code>if (response == 'Y')</code>• There are no short cuts!<ul style="list-style-type: none">• <code>if (response == 'y' 'Y')</code>• Correct:
<code>if (response == 'y' response == 'Y')</code>• Really, no short cuts!<ul style="list-style-type: none">• <code>if (10 < a < 100)</code>• Correct:
<code>if (10 < a && a < 100)</code> | <ul style="list-style-type: none">• This assigns the value of 'Y' to the variable response. Then, it takes the ASCII value of a 'Y' (which is NOT a zero) and checks to see if it is true or false. It will always be TRUE (100% of the time) because the ASCII value of a 'Y' is never zero.• <code>Response == 'y'</code> could be true or false<ul style="list-style-type: none">• But, 'Y' is NEVER zero. It is always true.• Resulting in a true answer always• This will result in an always true situation! |
|--|---|

Quick Review of Logicals

- `||` (or) is true when either operand is true (or both)
- `&&` (and) is true ONLY when both operands are true
- Usually `||` is used when we compare for equality (`==`)
- Usually `&&` is used when we compare if two items are not equal (`!=`)

Switch Statements

- Another C++ control statement is called the switch statement
- It allows you to pick the statements you want to execute from a list of possible statements, instead of just two different alternatives (as is available with an if/else) or a set of nested if/elses!
- It allows for multi-way decisions.

Switch Statements

```
char grade;
cout <<"Enter the grade..." <<endl;
cin >>grade;
switch (grade) {
    case 'A':  cout <<"Excellent" <<endl;
               cout <<"Keep up the good work!";
               break;
    case 'B':  cout <<"Very Good"; break;
    case 'C':  cout <<"Passing";   break;
    case 'D':  case 'F': cout <<"Too Bad";
               break;
    default :
               cout <<"No match was found...try again";
               break;
}
```

Switch Statements

- C++ provides a "default" clause so that if there isn't a match something is done. If the default is left off...and there is no match...no action takes place at all.
- When a case statement is executed, the value of Grade is checked and then depending on which of the cases it matches -- the statement following the colon for that case will be executed.

Switch Statements

- To exit from a switch statement...use break.
- Unlike Pascal, with C++ once you have a match...
- It will fall thru (ignoring any additional case or default labels that are encountered and continue executing code until a break is encountered.

Switch Statements

- The rule of thumb is that you can use these to switch on integers and characters.
- It is not permitted to use the switch with floating point types or a string of characters.
- The type of the expression following a switch keyword must be the same as the expressions following each case keyword....and no two expressions following the case keywords can be the same.

What Fall Thru means...

```
int count;
cout <<"Please enter the number of asterisks:";
cin >>count;
switch (count) {    //these { } are mandatory!
    case 1: cout <<"*";
    case 2: cout <<"**";
    case 3: cout <<"***";
    case 4: cout <<"****";
    default: cout <<"!";
}
cout <<endl;
```

The CORRECT version....

```
int count;
cout <<"Please enter the number of asterisks:";
cin >>count;
switch (count) {    //these { } are mandatory!
    case 1: cout <<"*";      break;
    case 2: cout <<"**";break;
    case 3: cout <<"***";    break;
    case 4: cout <<"****";   break;
    default: cout <<"!";break;
}
cout <<endl;
```

Introduction to C++

Repetition

Three types of Loops

- There are three ways to repeat a set of code using loops:
 - while loop
 - do while loop
 - for loop
- Each of these can perform the same operations...
 - it is all in how you think about it!

....let's see....

Using a While Loop

- Let's give the user a 2nd (and 3rd, 4th, 5th...) chance to enter their data using a while loop.
- While loops have the form: (notice semicolons!)

```
while (conditional expression)
    single statement;
```

```
while (conditional expression)
{
    many statements;
}
```

Using a While Loop

- The while statement means that while an expression is true, the body of the while loop will be executed.
- Once it is no longer true, the body will be bypassed.
- The first thing that happens is that the expression is checked, before the while loop is executed.

THIS ORDER IS IMPORTANT TO REMEMBER!

Using a While Loop

- The Syntax of the While Loop is:

```
while (loop repetition condition)  
    <body>
```

- Where, the <body> is either one statement followed by a semicolon or a compound statement surrounded by {}.
- Remember the body is only executed when the condition is true.
- Then, after the body is executed, the condition is tested again...

Using a While Loop

- Notice, you must remember to initialize the loop control variable before you enter the while loop.
- Then, you must have some way of updating that variable inside of the body of the loop so that it can change the condition from true to false at some desired time.
- If this last step is missing, the loop will execute "forever" ... this is called an infinite loop.

Using a While Loop

- We will need a loop control variable to be used to determine when the loop is done...

```
char response = '\n';
while ('\n' == response)
{
    cout << "Please enter ... ";
    cin >> data;
    cout << "We received: " << data
        << "\nIs this correct? (y/n)";
    cin >> response;
}
```

Using a While Loop

- What is a drawback of the previous loop?
 - The user may have entered a lower or upper case response!
- One way to fix this:
 - Change the conditional expression to list all of the legal responses

```
while (response == 'n' || response == 'N') {  
    . . .  
}
```

Using a While Loop

- **Yet another** way to fix this:

- To loop, assuming that they want to continually try again until they enter a Y or a y!
- Notice the use of AND versus OR!

```
while (response != 'y' && response != 'Y') {  
    . . .  
}
```


Using a While Loop

- **Another** way to fix this:
 - Use the tolower function in the cctype library:

```
#include <cctype>
```

```
while (tolower(response) != 'y') {  
    ...  
}
```

Using a While Loop

- **Another** way to fix this:
 - Use the toupper function in the ctype.h library:

```
#include <ctype.h>
```

```
while (toupper(response) != 'Y')  
{  
    . . .  
}
```

Using a do while Loop

- This same loop could have been rewritten using a do while loop instead
- do while loops have the form: (notice semicolons!)

```
do  
    single statement;  
while (conditional expression);
```

```
do  
{  
    many statements;  
} while (conditional expression);
```

Using a do while Loop

- Things to notice about a do while statement:
 - (1) The body of a do while statement can be one statement or a compound statement surrounded by {}
 - (2) Each statement in the do while loop is separated by a semicolon
 - (3) Notice the body is always executed once! Even if the conditional expression is false the first time!

Using a do while Loop

- Don't use a do while unless you are sure that the body of the loop should be executed at least once!

```
char response;
do
{
    cout <<"Please enter ... ";
    cin >> data;
    cout <<"We received: " <<data
        <<"\nIs this correct? (y/n)";
    cin >>response;
} while (response != 'y' && response != 'Y');
```

Using a for loop

- The for loop is commonly used to loop a certain number of times. For example, you can use it to print out the integers 1 thru 9:

```
int i;
```

```
for (i=1; i <= 9; ++i)
```

```
    cout <<i <<endl;
```

Using a for loop

- *i* is called the loop control variable.
- It is most common to use variables *i*, *j*, and *k* for control variables.
- But, mnemonic names are better!

```
for (initialize; conditional exp; increment)
    <body>
```

- The body of the for loop is either one statement followed by a semicolon or a compound statement surrounded by {}.

Using a for loop

- The for statement will first
 - (1) INITIALIZE VARIABLE *i* to 1;
 - (2) Check the conditional expression to see if it is True or False;
 - (3) if it is True the body of the loop is executed and it INCREMENTs VARIABLE *i* by 1;or, if it is False the loop is terminated and the statement following the body of the loop is executed.

Using a for Loop

- **In C++**

```
for (i=0; i < 10; ++i)
    j+=i ;//remember this is j = j+1;
```

- **is the same as:**

```
i = 0;
while (i < 10) {
    j += i;
    ++i;
}
```

Using a for Loop

- We can also use a for loop to do the same loop that we have been talking about today:

```
for (char response = '\n';
     response != 'y' && response != 'Y';
     cin >> response)
{
    cout << "Please enter ... ";
    cin >> data;
    cout << "We received: " << data
         << "\nIs this correct? (y/n)";
}
```

Using a for Loop – NULL BODY

- Remember to use semicolons after each statement; however, a semicolon right after the parentheses will cause there to be a null body (i.e., nothing will be executed as long as you are inside the loop!):

```
for (i=1; i <= 10; ++i) ; //null body
    cout <<"hello";      //this happens ONLY
                        //after i is > 10.
```

Using a do while Loop

- When using loops, desk check for the following conditions:
 - (1) Has the loop iterated one too many times? Or, one too few times?
 - (2) Have you properly initialized the variables used in your while or do-while conditional expressions?
 - (3) Are you decrementing or incrementing those variables within the loop?
 - (4) Is there an infinite loop?

Introduction to C++

Arrays

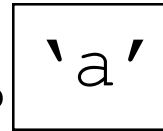
Introduction to Arrays

- Strings are represented in C++ by **arrays of characters**
- **Or**, they are represented as a User Defined Type (called a Class) ... but first let's learn about arrays
 - “Strings” are actually not built into the language
- We all know what a character is (a single byte), so what's an array of characters?
 - a sequence of character stored sequentially in memory
- The size must be specified as a constant or a literal
 - *Just because it compiles doesn't mean it is correct!*

How do I define an Array of Characters?

- We know how to define a single character:

```
char ch = 'a';
```



- But what about an array of characters?

```
char str[5];
```



- Since these are

just characters stored sequentially in memory, we use a special character to indicate the end of a string: `'\0'`

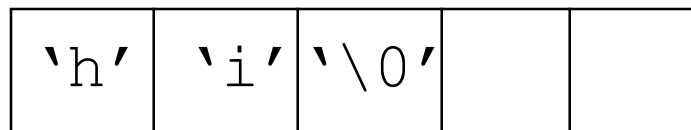
- We must allow for ONE extra element to hold the `'\0'` which is the terminating NUL character, indicating the end of the used part of the array versus the unused.
- This array can hold 4 characters for the name and one character for the `'\0'`

How do I read in a string?

- There are two ways to read in strings
- If the string is a sequence of characters without any whitespace (like your first name), then you can say:

```
cin >>str;
```

- If I enter “hi”, this is what is stored:



- This skips leading whitespace and reads until whitespace is encountered but not read.

What does `cin >> array_of_characters` do?

```
char str[5];  
cin >>str;
```

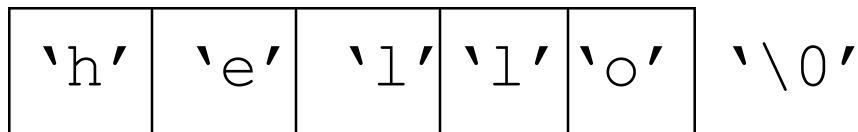
- When reading in an array of characters, `cin` and the extraction operator (`>>`) skip leading whitespace and read characters until a whitespace character is encountered.
- Then, it automatically stores a `'\0'` after the last character read in.

What do we need to be careful about?

- We need to be careful when working with arrays of characters...
- If we have an array of size 5
 - that means there are 5 bytes of memory allocated for our variable sequentially in memory
- This means that we can store four characters at most, since one spot needs to be reserved for the terminating nul

So, What could happen???

- Using `cin >>str;`
- If I enter “hello”, this is what is stored:



- Notice we ended up storing the ‘\0’ in memory that is not allocated for our variable
 - this is extremely dangerous and can cause our programs to bomb! (segmentation fault or core dump when running...)

What do we need to be careful about?

- What this means is that C++ does not check to make sure we stay within the bounds of our arrays
- C++ assumes that we know what we are doing!
- It is a powerful language...one that can even be used to design operating systems
- Therefore, if there is a chance that the user may type in too many characters, we need to read in our strings using a different approach

How do I read in a string safely?

- Or, we can use this function to read in a string using 2 or 3 arguments:

```
char str[5];  
cin.get(str, 5, '\n');  
// same as: cin.get(str, 5);
```

- this reads in the next sequence of characters up until (size-1) characters are read or the delimiting character is encountered ('\n' by default) but not read.
- We need to follow this by a `cin.ignore` so that the newline is removed from the input buffer.

How do I read in a string safely?

- The three argument version of `cin.get` has the following form:

```
cin.get(array_name, max_size, delimiting_character);
```

- A side benefit of this function is that it will allow us to read in lines, sentences, our entire first/last name, a paragraph, etc. This is because the delimiting character need not be white space!

How do I read in a string safely?

- There is one “gotcha” with this function.
- While the three argument version of `cin.get` won’t read in too many character (so it will never store characters outside your array bounds),
 - it will not read in the delimiting character!
- Therefore, we must always “eat” the delimiting character, using either:

`cin.get();` or `while(cin.get() != '\n');`

- *this must be done after all input preceding a 2 or 3 argument `cin.get` use*

Let's read another string, using `cin.get`:

- Using `cin.get(str, 5);`
- If I enter “hi !”, this is what is stored:

'h'	'i'	' '	'!'	'\0'
-----	-----	-----	-----	------

- Notice that room is left to store the `'\0'` at the end of the array, and there is no danger of writing outside of our array bounds.
- But, what is left in the input buffer? `'\n'`
- How do we “flush” this? `cin.get();`

Let's read another string, using `cin.get`:

- Using `cin.get(str, 5);`
- If I enter “hello”, this is what is stored:

'h'	'e'	'l'	'l'	'\0'
-----	-----	-----	-----	------

- Notice that room is left to store the `'\0'` at the end of the array, and there is no danger of writing outside of our array bounds.
- But, what is left in the input buffer? `'o\n'`
- How do we “flush” this? `while(cin.get() != '\n');`

How do I display a string?

- Luckily, displaying strings isn't as complicated.

```
cout <<str;
```

- Simply by using cout followed by the insertion operator (<<), we can display as many characters as have been stored in the array until the terminating nul ('\0') is encountered.
- Notice, the '\0' is important so that we don't display "garbage" characters (i.e., memory that has not been set or used yet!)

Operations on Strings

- There are very few operations that can be performed on array of characters (i.e., strings)
- For example, we cannot compare two strings by saying:

```
char str1[10], str2[10];  
if (str1 == str2)
```

- This is because an array is really *the address of the first element in a sequentially allocated set of memory*.
- So, the == or != operators would simply be comparing the memory locations! Oops!

Comparing Strings:

- Instead, to compare two strings we can include another library: `string.h`

- And, call the string compare function:

```
strcmp(first_array, second_array);
```

- The `strcmp` function returns:

- 0 if `first_array` is equal to `second_array`
 - <0 if `first_array` is less than `second_array`
 - >0 if `first_array` is greater than `second_array`

Copying Strings

- We also cannot copy strings using the assignment operator:

```
char str1[10], str2[10];  
str1 = str2;
```

- This is illegal because an array is really *the address of the first element in a sequentially allocated set of memory*.
- Instead, we call strcpy from string.h

```
strcpy(str1, str2); //str1=str2;
```

For example:

- Let's now put this to use by writing a function to read in two strings and displaying them in alphabetical order
- First, write the algorithm:
 - Get two strings (prompt, input, echo)
 - If the first string is less than the second
 - display the first string followed by the second
 - If the first string is greater or equal to the second
 - display the second string followed by the first

Working with arrays, character at a time:

- We can also work with strings an element at a time,
 - by indexing through the array
 - we begin by using subscripts that start at zero and then progress until the array size-1
- For example, we can read in a string by:
 - Read a character
 - If that character is not a carriage return
 - save the character in the array

Reading a Character at a time:

```
char str[20]; char ch;  
int index = 0;  
  
ch = cin.get();  
while (ch != '\n') {  
    str[index] = ch; //str[index] is a char  
    ++index;  
    ch = cin.get();  
}  
str[index] = '\0'; //why is this important?
```

- But, what if they type in too many characters?

A Better Approach?

```
const int MAX = 20;
char str[MAX];      char ch;
int index = 0;

ch = cin.get();
while (index < MAX-1 && ch != '\n') {
    str[index] = ch; //str[index] is a char
    ++index;
    ch = cin.get();
}
str[index] = '\0';    //why is this important?
```

The Same Thing...Just Condensed:

```
const int MAX = 20;
char str[MAX];
int index = 0;

while (index < MAX-1 && (ch= cin.get()) != '\n'))
    str[index++] = ch;    //Remember postfix????

str[index] = '\0';    //Still important
```

Or, going to an extreme!

```
const int MAX = 20;
char str[MAX];
int index = 0;

while (index < MAX-1 &&
      (str[index++] = cin.get()) != '\\n');

str[index] = '\\0';
```

Introduction to C++

