The University Of Michigan

EECS402 Lecture 01

Andrew M. Morgan

Savitch Ch. 2
C++ Basics
Flow Of Control

Identifiers

- Names of variables, constants, user-defined functions, etc
- Valid identifiers
  - Must start with letter or underscore
  - Contains only letters, digits, or underscore
  - Can not be C/C++ reserved word
- Note: C/C++ identifiers are *case sensitive*
- Valid identifier examples
  - i, val, Val, VAL, _internal, my_var, myVar, twoNums, x54
- Invalid identifier examples
  - 2nums, my-var, class, file.name

Variables

- All variables must have a name and a type
- C++ is a strongly-typed language
- Variable names are any valid C++ identifier
- The type of a variable describes what kind of data it holds
- Values of variables can change throughout a program
- Following are some of the C++ data types
  - int: Integer data (-6, 0, 741, -1024)
  - float/double: Floating point data (6.5, 8.0, -97.21204, 0.0081)
  - char: Character data ('a', 'q', '5', '\n')
  - bool: Boolean values (true, false)

Using Variables

- Before any variable can be used, it must be declared
- Gives the variable a type and sets aside memory
  - `int counter; //Declares an integer variable called counter`
  - `float average; //Declares a float called average`
  - `char grade; //Declares a character to represent a grade`
- Assignment – setting a variables value
  - `counter = 10;`
  - `average = 88.25;`
  - `grade = 'B';`
- Initialization can be done during declaration
  - `char modif = '+'; //Modifer to be appended to grade`
  - `int sumOfValues = 0; //Some of input values`
  - `float initialBudget = 135.50; //Initial budget for week`
- If not initialized, the value of the variable is undefined
  - Note: It will most likely NOT be 0
- Style: Variable names in lower case, except first letters of non-first words

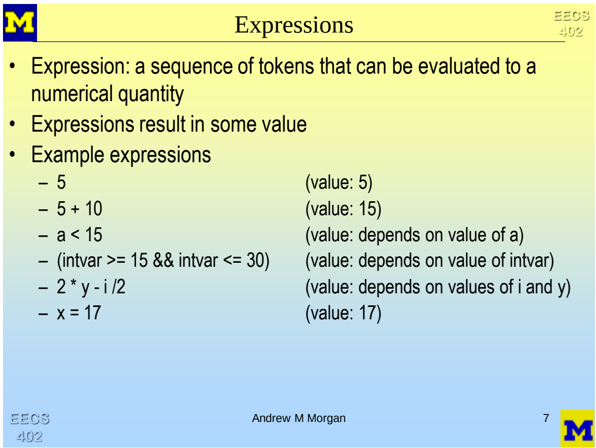
Declaring Constants

- Constants must have a name and a type
- The value of a constant *must* be initialized at declaration time
- The value is not allowed to change during program execution
- Used to avoid "magic numbers" – literal values in a program
  - Seeing the value 12 in a program is not very meaningful – it could represent the number of quiz scores, the number of hours in a half-day…
- Begin declaration with C++ keyword "const"
  - `const float PI = 3.1415;`
  - `const int NUM_SCORES = 12;`
  - `const char BEST_GRADE = 'A';`
- Style: Constant names in ALL CAPS to differentiate from variables

Some C++ Operators

- Operators are used on variables and/or literals to compute a new value.

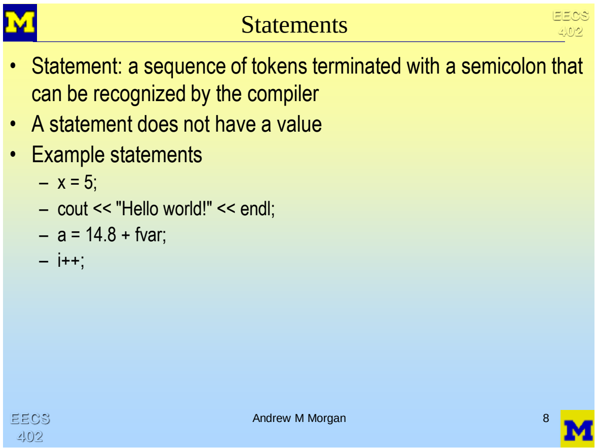
<table>
<tr><td>=</td><td>Assignment operator (not equality)</td></tr>
<tr><td>+, -, *, /</td><td>Add, subtract, multiply, divide</td></tr>
<tr><td>%</td><td>Modulus (remainder)</td></tr>
<tr><td>==, !=</td><td>Equality, inequality</td></tr>
<tr><td>++, --</td><td>Increment, decrement</td></tr>
<tr><td>+=, -=, *=, /=</td><td>Add/assign, etc</td></tr>
<tr><td colspan="2">i -= 4 is equivalent to i = i – 4</td></tr>
<tr><td>&gt;, &lt;, &gt;=, &lt;=</td><td>Greater than, less than, greater or equal, less or equal</td></tr>
<tr><td>&amp;&amp;</td><td>Logical AND, returns true when both operands are true</td></tr>
<tr><td>||</td><td>Logical OR, returns true when >= 1 operand is true</td></tr>
<tr><td>!</td><td>Local NOT, returns true when operand is false</td></tr>
</table>



Expressions

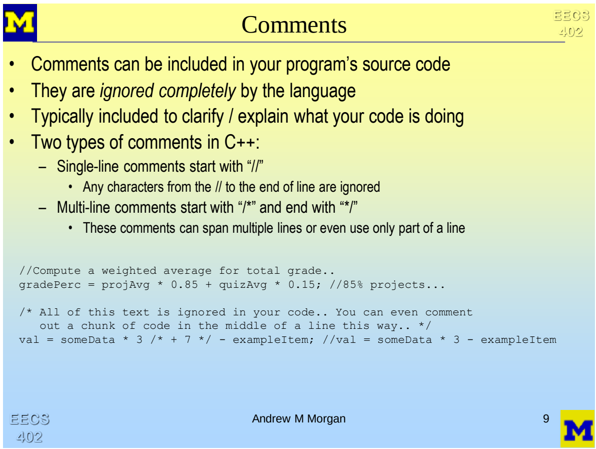
- Expression: a sequence of tokens that can be evaluated to a numerical quantity
- Expressions result in some value
- Example expressions

– 5	(value: 5)
– 5 + 10	(value: 15)
– a < 15	(value: depends on value of a)
– (intvar >= 15 && intvar <= 30)	(value: depends on value of intvar)
– 2 * y - i /2	(value: depends on values of i and y)
– x = 17	(value: 17)



Statements

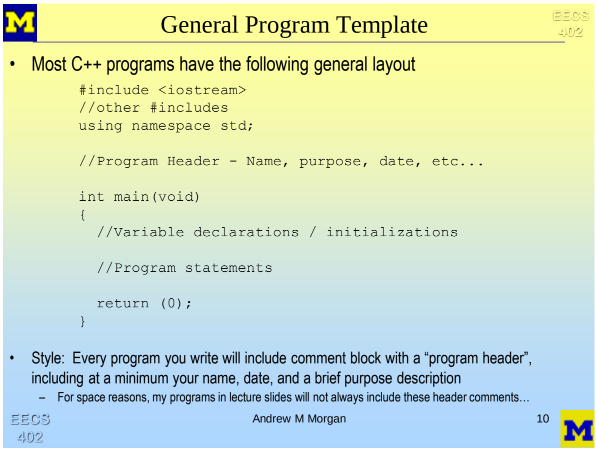
- Statement: a sequence of tokens terminated with a semicolon that can be recognized by the compiler
- A statement does not have a value
- Example statements
  - x = 5;
  - cout << "Hello world!" << endl;
  - a = 14.8 + fvar;
  - i++;



Comments

- Comments can be included in your program’s source code
- They are *ignored completely* by the language
- Typically included to clarify / explain what your code is doing
- Two types of comments in C++:
  - Single-line comments start with “//”
    - Any characters from the // to the end of line are ignored
  - Multi-line comments start with “/*” and end with “*/”
    - These comments can span multiple lines or even use only part of a line

```
//Compute a weighted average for total grade..
gradePerc = projAvg * 0.85 + quizAvg * 0.15; //85% projects...

/* All of this text is ignored in your code.. You can even comment
   out a chunk of code in the middle of a line this way.. */
val = someData * 3 /* + 7 */ - exampleItem; //val = someData * 3 - exampleItem
```

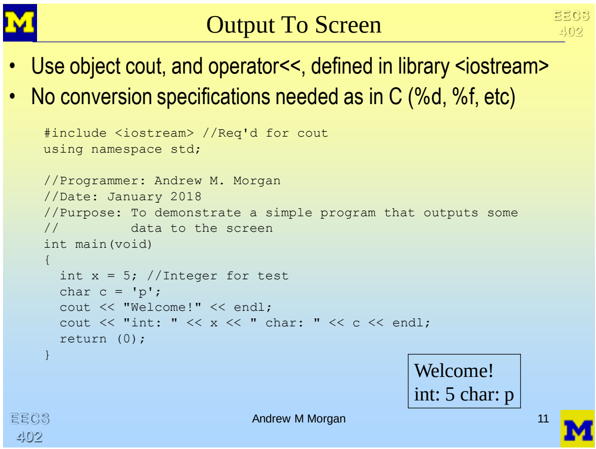


General Program Template

- Most C++ programs have the following general layout

```
#include <iostream>
//other #includes
using namespace std;

//Program Header - Name, purpose, date, etc...

int main(void)
{
  //Variable declarations / initializations

  //Program statements

  return (0);
}
```

- Style: Every program you write will include comment block with a “program header”, including at a minimum your name, date, and a brief purpose description
  - For space reasons, my programs in lecture slides will not always include these header comments…

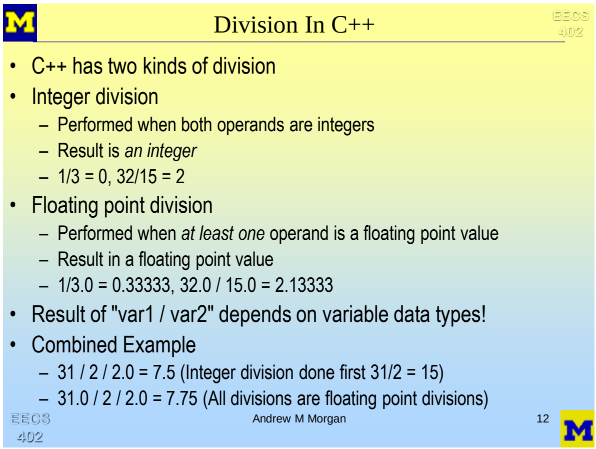


Output To Screen

- Use object cout, and operator<<, defined in library <iostream>
- No conversion specifications needed as in C (%d, %f, etc)

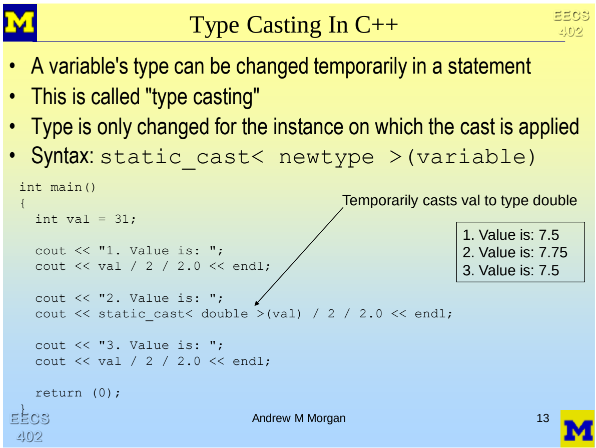
```
#include <iostream> //Req'd for cout
using namespace std;

//Programmer: Andrew M. Morgan
//Date: January 2018
//Purpose: To demonstrate a simple program that outputs some
//         data to the screen
int main(void)
{
  int x = 5; //Integer for test
  char c = 'p';
  cout << "Welcome!" << endl;
  cout << "int: " << x << " char: " << c << endl;
  return (0);
}
```

```
Welcome!
int: 5 char: p
```



Division In C++

- C++ has two kinds of division
- Integer division
  - Performed when both operands are integers
  - Result is *an integer*
  - 1/3 = 0, 32/15 = 2
- Floating point division
  - Performed when *at least one* operand is a floating point value
  - Result in a floating point value
  - 1/3.0 = 0.33333, 32.0 / 15.0 = 2.13333
- Result of "var1 / var2" depends on variable data types!
- Combined Example
  - 31 / 2 / 2.0 = 7.5 (Integer division done first 31/2 = 15)
  - 31.0 / 2 / 2.0 = 7.75 (All divisions are floating point divisions)



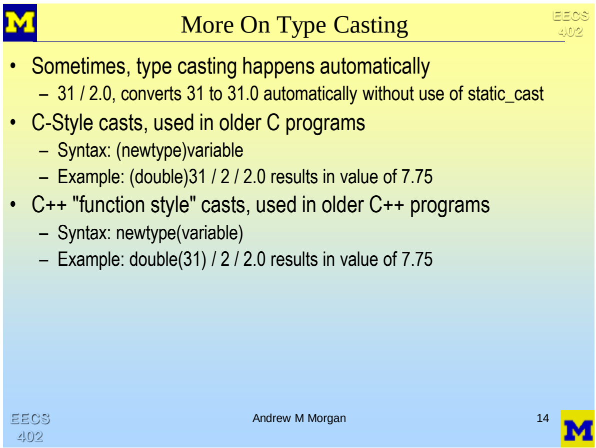
Type Casting In C++

- A variable's type can be changed temporarily in a statement
- This is called "type casting"
- Type is only changed for the instance on which the cast is applied
- Syntax: `static_cast< newtype >(variable)`

```
int main()
{
  int val = 31;

  cout << "1. Value is: ";
  cout << val / 2 / 2.0 << endl;

  cout << "2. Value is: ";
  cout << static_cast< double >(val) / 2 / 2.0 << endl;

  cout << "3. Value is: ";
  cout << val / 2 / 2.0 << endl;

  return (0);
}
```

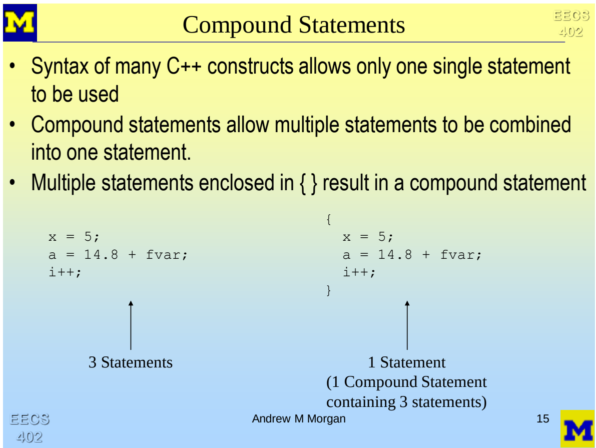
Temporarily casts val to type double

```
1. Value is: 7.5
2. Value is: 7.75
3. Value is: 7.5
```



More On Type Casting

- Sometimes, type casting happens automatically
  - 31 / 2.0, converts 31 to 31.0 automatically without use of static_cast
- C-Style casts, used in older C programs
  - Syntax: (newtype)variable
  - Example: (double)31 / 2 / 2.0 results in value of 7.75
- C++ "function style" casts, used in older C++ programs
  - Syntax: newtype(variable)
  - Example: double(31) / 2 / 2.0 results in value of 7.75



Compound Statements

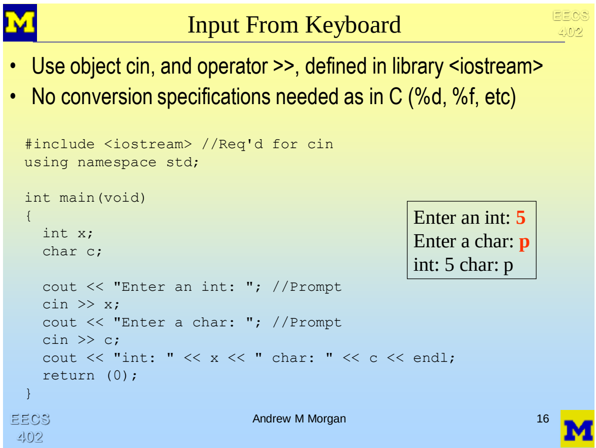
- Syntax of many C++ constructs allows only one single statement to be used
- Compound statements allow multiple statements to be combined into one statement.
- Multiple statements enclosed in { } result in a compound statement

```
x = 5;
a = 14.8 + fvar;
i++;
```

3 Statements

```
{
  x = 5;
  a = 14.8 + fvar;
  i++;
}
```

1 Statement
(1 Compound Statement
containing 3 statements)

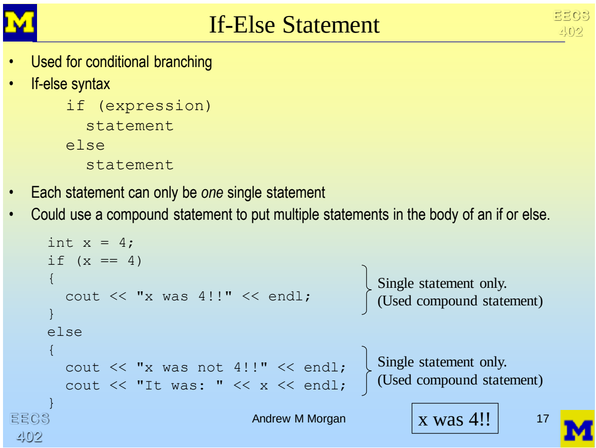


Input From Keyboard

- Use object cin, and operator >>, defined in library <iostream>
- No conversion specifications needed as in C (%d, %f, etc)

```
#include <iostream> //Req'd for cin
using namespace std;

int main(void)
{
  int x;
  char c;

  cout << "Enter an int: "; //Prompt
  cin >> x;
  cout << "Enter a char: "; //Prompt
  cin >> c;
  cout << "int: " << x << " char: " << c << endl;
  return (0);
}
```

```
Enter an int: 5
Enter a char: p
int: 5 char: p
```



If-Else Statement

- Used for conditional branching
- If-else syntax

```
if (expression)
  statement
else
  statement
```

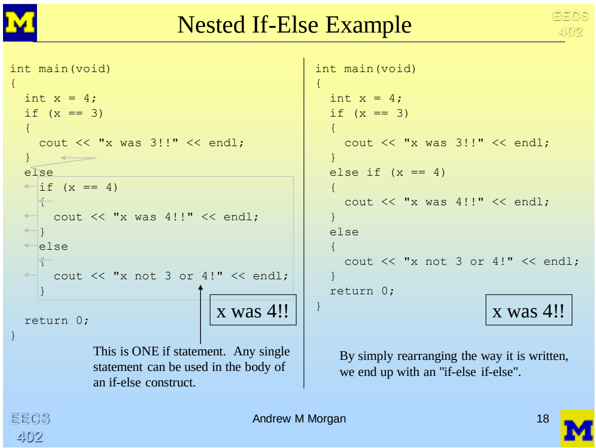
- Each statement can only be *one* single statement
- Could use a compound statement to put multiple statements in the body of an if or else.

```
int x = 4;
if (x == 4)
{
  cout << "x was 4!!" << endl;
}
else
{
  cout << "x was not 4!!" << endl;
  cout << "It was: " << x << endl;
}
```

Single statement only.
(Used compound statement)

Single statement only.
(Used compound statement)

```
x was 4!!
```



Nested If-Else Example

```
int main(void)
{
  int x = 4;
  if (x == 3)
  {
    cout << "x was 3!!" << endl;
  }
  else
  if (x == 4)
  {
    cout << "x was 4!!" << endl;
  }
  else
  {
    cout << "x not 3 or 4!" << endl;
  }

  return 0;
}
```

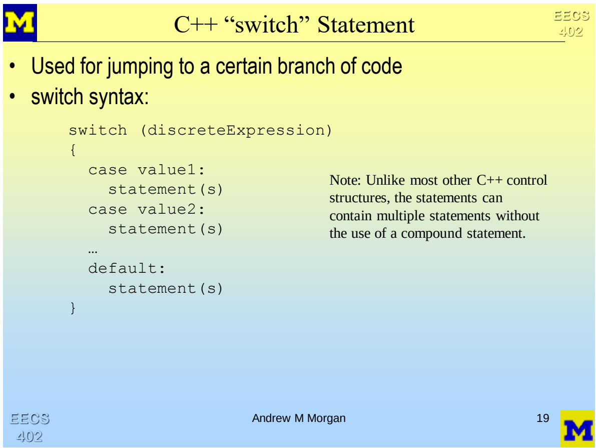
```
x was 4!!
```

This is ONE if statement. Any single statement can be used in the body of an if-else construct.

```
int main(void)
{
  int x = 4;
  if (x == 3)
  {
    cout << "x was 3!!" << endl;
  }
  else if (x == 4)
  {
    cout << "x was 4!!" << endl;
  }
  else
  {
    cout << "x not 3 or 4!" << endl;
  }
  return 0;
}
```

```
x was 4!!
```

By simply rearranging the way it is written, we end up with an "if-else if-else".

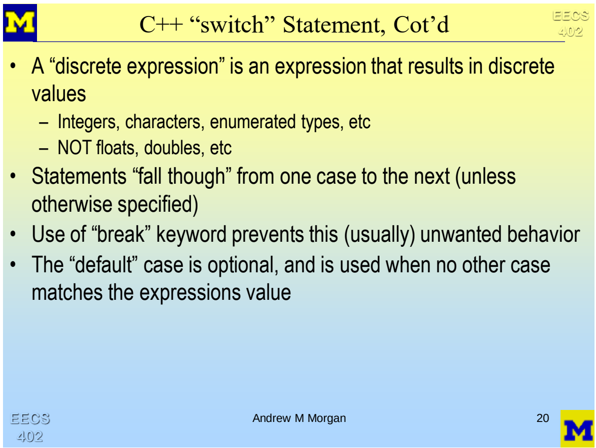


C++ “switch” Statement

- Used for jumping to a certain branch of code
- switch syntax:

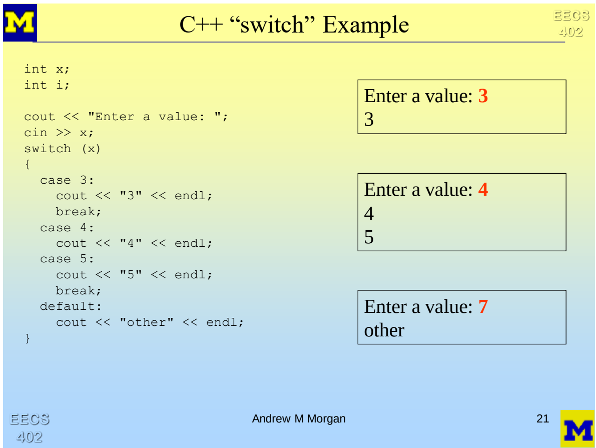
```
switch (discreteExpression)
{
  case value1:
    statement(s)
  case value2:
    statement(s)
  …
  default:
    statement(s)
}
```

Note: Unlike most other C++ control structures, the statements can contain multiple statements without the use of a compound statement.



C++ “switch” Statement, Cot’d

- A “discrete expression” is an expression that results in discrete values
  - Integers, characters, enumerated types, etc
  - NOT floats, doubles, etc
- Statements “fall though” from one case to the next (unless otherwise specified)
- Use of “break” keyword prevents this (usually) unwanted behavior
- The “default” case is optional, and is used when no other case matches the expressions value



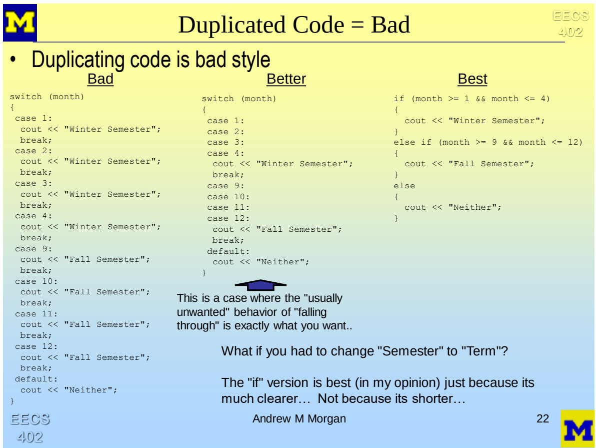
C++ “switch” Example

```
int x;
int i;

cout << "Enter a value: ";
cin >> x;
switch (x)
{
  case 3:
    cout << "3" << endl;
    break;
  case 4:
    cout << "4" << endl;
  case 5:
    cout << "5" << endl;
    break;
  default:
    cout << "other" << endl;
}
```

```
Enter a value: 3
3
```

```
Enter a value: 4
4
5
```

```
Enter a value: 7
other
```



Duplicated Code = Bad

- Duplicating code is bad style

Bad

```
switch (month)
{
 case 1:
  cout << "Winter Semester";
  break;
 case 2:
  cout << "Winter Semester";
  break;
 case 3:
  cout << "Winter Semester";
  break;
 case 4:
  cout << "Winter Semester";
  break;
 case 9:
  cout << "Fall Semester";
  break;
 case 10:
  cout << "Fall Semester";
  break;
 case 11:
  cout << "Fall Semester";
  break;
 case 12:
  cout << "Fall Semester";
  break;
 default:
  cout << "Neither";
}
```

Better

```
switch (month)
{
 case 1:
 case 2:
 case 3:
 case 4:
  cout << "Winter Semester";
  break;
 case 9:
 case 10:
 case 11:
 case 12:
  cout << "Fall Semester";
  break;
 default:
  cout << "Neither";
}
```

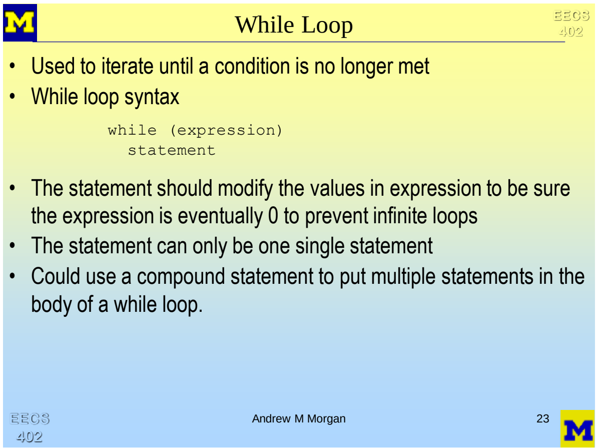
This is a case where the "usually unwanted" behavior of "falling through" is exactly what you want..

Best

```
if (month >= 1 && month <= 4)
{
  cout << "Winter Semester";
}
else if (month >= 9 && month <= 12)
{
  cout << "Fall Semester";
}
else
{
  cout << "Neither";
}
```

What if you had to change "Semester" to "Term"?

The "if" version is best (in my opinion) just because its much clearer… Not because its shorter…

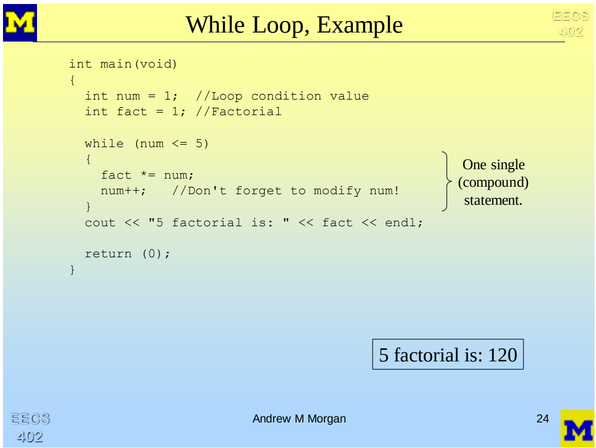


While Loop

- Used to iterate until a condition is no longer met
- While loop syntax

```
while (expression)
  statement
```

- The statement should modify the values in expression to be sure the expression is eventually 0 to prevent infinite loops
- The statement can only be one single statement
- Could use a compound statement to put multiple statements in the body of a while loop.

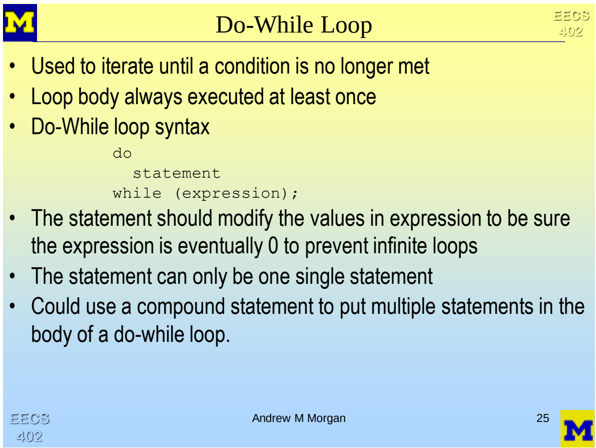


While Loop, Example

```
int main(void)
{
  int num = 1;  //Loop condition value
  int fact = 1; //Factorial

  while (num <= 5)
  {
    fact *= num;
    num++;   //Don't forget to modify num!
  }
  cout << "5 factorial is: " << fact << endl;

  return (0);
}
```

One single (compound) statement.

```
5 factorial is: 120
```

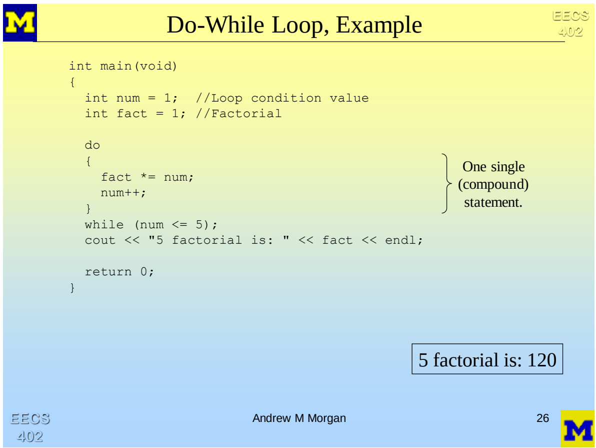


Do-While Loop

- Used to iterate until a condition is no longer met
- Loop body always executed at least once
- Do-While loop syntax

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

- The statement should modify the values in expression to be sure the expression is eventually 0 to prevent infinite loops
- The statement can only be one single statement
- Could use a compound statement to put multiple statements in the body of a do-while loop.

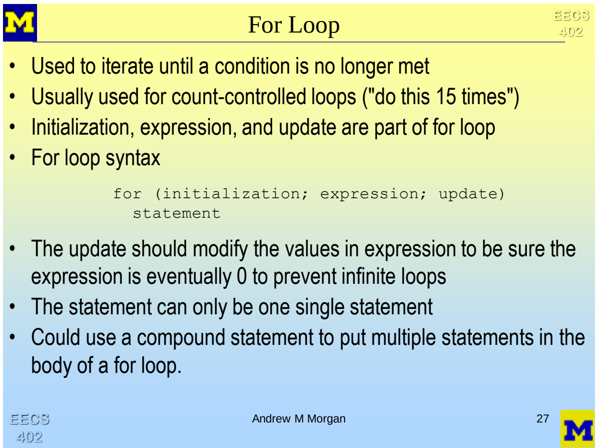


Do-While Loop, Example

```
int main(void)
{
  int num = 1;  //Loop condition value
  int fact = 1; //Factorial

  do
  {
    fact *= num;
    num++;
  }
  while (num <= 5);
  cout << "5 factorial is: " << fact << endl;

  return 0;
}
```

One single (compound) statement.

```
5 factorial is: 120
```

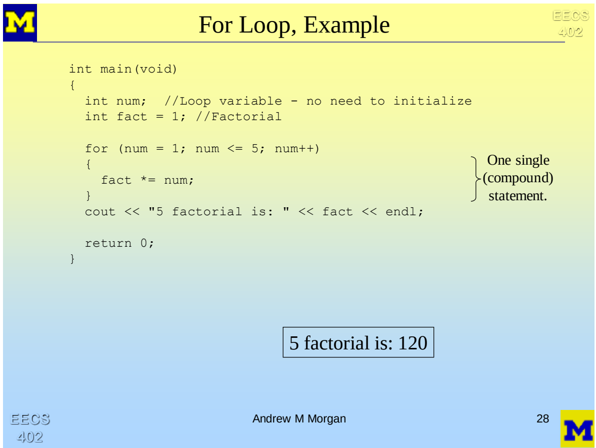


For Loop

- Used to iterate until a condition is no longer met
- Usually used for count-controlled loops ("do this 15 times")
- Initialization, expression, and update are part of for loop
- For loop syntax

```
for (initialization; expression; update)
  statement
```

- The update should modify the values in expression to be sure the expression is eventually 0 to prevent infinite loops
- The statement can only be one single statement
- Could use a compound statement to put multiple statements in the body of a for loop.



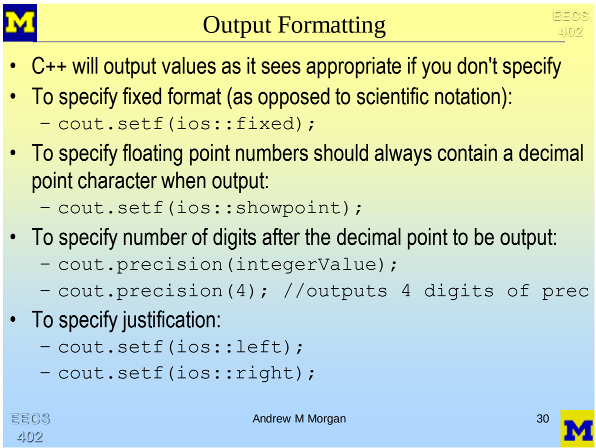
For Loop, Example

```
int main(void)
{
  int num;  //Loop variable - no need to initialize
  int fact = 1; //Factorial

  for (num = 1; num <= 5; num++)
  {
    fact *= num;
  }
  cout << "5 factorial is: " << fact << endl;

  return 0;
}
```

One single (compound) statement.

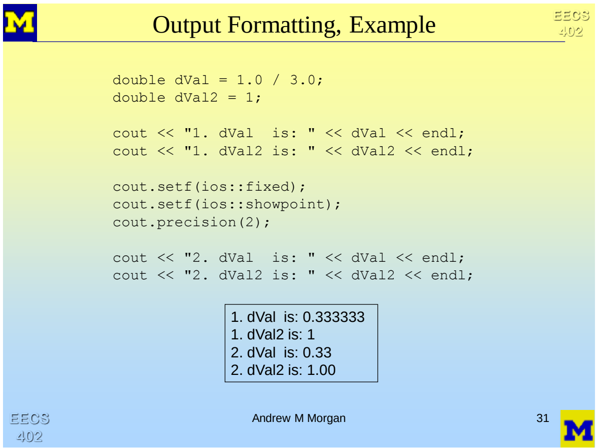
```
5 factorial is: 120
```

Additional Reference Material



Output Formatting

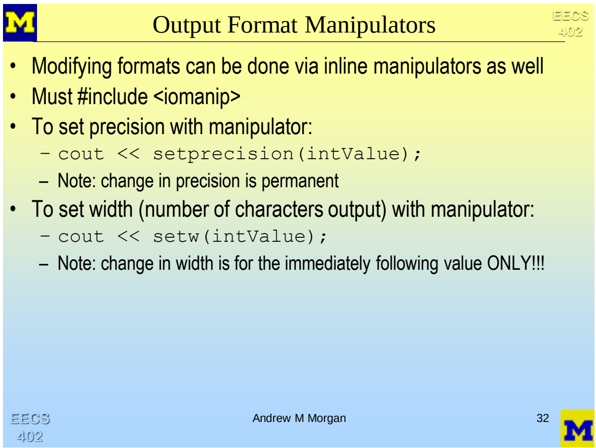
- C++ will output values as it sees appropriate if you don't specify
- To specify fixed format (as opposed to scientific notation):
  - `cout.setf(ios::fixed);`
- To specify floating point numbers should always contain a decimal point character when output:
  - `cout.setf(ios::showpoint);`
- To specify number of digits after the decimal point to be output:
  - `cout.precision(integerValue);`
  - `cout.precision(4); //outputs 4 digits of prec`
- To specify justification:
  - `cout.setf(ios::left);`
  - `cout.setf(ios::right);`



Output Formatting, Example

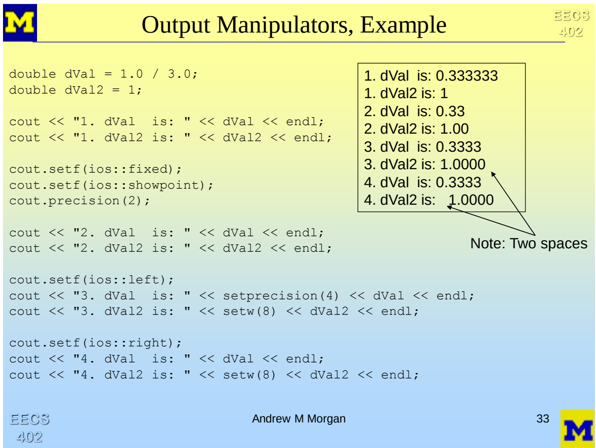
```
double dVal = 1.0 / 3.0;
double dVal2 = 1;

cout << "1. dVal  is: " << dVal << endl;
cout << "1. dVal2 is: " << dVal2 << endl;

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);

cout << "2. dVal  is: " << dVal << endl;
cout << "2. dVal2 is: " << dVal2 << endl;
```

```
1. dVal  is: 0.333333
1. dVal2 is: 1
2. dVal  is: 0.33
2. dVal2 is: 1.00
```



Output Format Manipulators

- Modifying formats can be done via inline manipulators as well
- Must #include <iomanip>
- To set precision with manipulator:
  - `cout << setprecision(intValue);`
  - Note: change in precision is permanent
- To set width (number of characters output) with manipulator:
  - `cout << setw(intValue);`
  - Note: change in width is for the immediately following value ONLY!!!



Output Manipulators, Example

```
double dVal = 1.0 / 3.0;
double dVal2 = 1;

cout << "1. dVal  is: " << dVal << endl;
cout << "1. dVal2 is: " << dVal2 << endl;

cout.setf(ios::fixed);
cout.setf(ios::showpoint);
cout.precision(2);

cout << "2. dVal  is: " << dVal << endl;
cout << "2. dVal2 is: " << dVal2 << endl;

cout.setf(ios::left);
cout << "3. dVal  is: " << setprecision(4) << dVal << endl;
cout << "3. dVal2 is: " << setw(8) << dVal2 << endl;

cout.setf(ios::right);
cout << "4. dVal  is: " << dVal << endl;
cout << "4. dVal2 is: " << setw(8) << dVal2 << endl;
```

```
1. dVal  is: 0.333333
1. dVal2 is: 1
2. dVal  is: 0.33
2. dVal2 is: 1.00
3. dVal  is: 0.3333
3. dVal2 is: 1.0000
4. dVal  is: 0.3333
4. dVal2 is:   1.0000
```

Note: Two spaces