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, types of variables can not
- 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
```

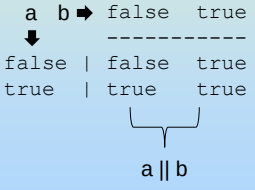
- If not initialized, the value of the variable is undefined
  - Note: It will most likely NOT be 0
- Style: Variable names in lower case, except first letters of non-first words

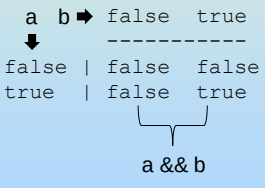
Declaring Constants

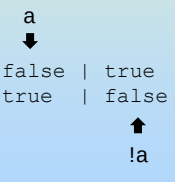
- Constants must have a name and a type
- The value of a constant *must* be initialized at declaration time
- The value is not allowed to change during program execution
- Used to avoid "magic numbers" – literal values in a program
  - Seeing the value 12 in a program is not very meaningful – it could represent the number of quiz scores, the number of hours in a half-day…
- Begin declaration with C++ keyword "const"
  - `const double PI = 3.141592654;`
  - `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.
  - `=` Assignment operator (not equality)
  - `+, -, *, /` Add, subtract, multiply, divide
  - `==, !=` Equality, inequality
  - `++, --` Increment, decrement
  - `+=, -=, *=, /=` Add/assign, etc (`i -= 4` is equivalent to `i = i - 4`, etc)
  - `>, <, >=, <=` Greater than, less than, greater or equal, less or equal
  - `&&` Logical AND, returns true when both operands are true
  - `||` Logical OR, returns true when >= 1 operand is true
  - `!` Local NOT, returns true when operand is false
  - `%` Modulus (integer remainder)







Expressions

- Expression: a sequence of tokens that can be evaluated to a value
- Expressions result in some value
- Example expressions
  - 5 (value: 5)
  - 5 + 10 (value: 15)
  - averageVal < 15 (value: depends on value of averageVal)
  - (intVar >= 15 && intVar <= 30) (value: depends on value of intVar)
  - 2 * stdDev - i /2 (value: depends on values of i and stdDev)

Statements

- Statement: a sequence of tokens (often terminated with a semicolon) that can be recognized by the compiler as a complete instruction
- A statement does not have a value
- Example statements
  - numItems = 5;
  - success = wasInitialized && didFirstStepWork;
  - scaledValue = BIAS_AMOUNT + initialValue * SCALE_AMOUNT;
  - 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
  - Comments should *add value* to the code from a reader's perspective
- 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()
{
  //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 cout and << as defined in library <iostream>
  - endl means to print a newline character

```
#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()
{
  int ageYears = 5; //assuming 5 years old for now
  char firstInit = 'P';

  cout << "Welcome!" << endl;
  cout << "Age: " << ageYears << " years, first initial: " << firstInit << endl;
  cout << "In 10 years, age will be: " << (ageYears + 10) << endl;

  return 0;
}
```

```
Welcome!
Age: 5 years, first initial: P
In 10 years, age will be: 15
```


Input From Keyboard

- Use cin and >>, as defined in library <iostream>

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

int main()
{
  int ageYears;
  char firstInit;

  cout << "Enter your age: "; //Prompt
  cin >> ageYears;

  cout << "Enter your first initial: "; //Prompt
  cin >> firstInit;

  cout << "Age: " << ageYears << " years, first initial: " << firstInit << endl;

  return 0;
}
```

```
Enter your age: 25
Enter your first initial: P
Age: 25 years, first initial: 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)`

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

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 many 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

```
ageYears = 5;
olderAge = ageYears + NUM_YEARS;
i++;
```

3 Statements

```
{
  ageYears = 5;
  olderAge = ageYears + NUM_YEARS;
  i++;
}
```

1 Statement
(1 Compound Statement containing 3 statements)

- Style: All code within a set of curly braces must be indented using a consistent number of spaces (not tabs)!

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.
- The "else" part is optional - if you don't need to do anything when the expression is false, just leave it off completely

```
int ageYears = 25;
if (ageYears == 25)
{
  cout << "age was 25!!" << endl;
}
else
{
  cout << "age was not 25!!" << endl;
  cout << "It was: " << ageYears << endl;
}
```

Single statement only. (Used compound statement)

Single statement only. (Used compound statement)

age was 25!!

If Statement (without else)

- The "else" part is optional - if you don't need to do anything when the expression is false, just leave it off completely
- Used for conditional branching
- If syntax

```
if (expression)
  statement
```

- Statement can only be *one* single statement
- Could use a compound statement to put multiple statements there

```
int ageYears = 35;
if (ageYears == 25)
{
  cout << "age was 25!!" << endl;
}
```

Single statement only. (Used compound statement)

<no output generated>

If-"Else If"-Else Example


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

int main()
{
  int ageYears = 30;

  if (ageYears < 21)
  {
    cout << "age was under the limit!!" << endl;
  }
  else if (ageYears == 30)
  {
    cout << "age was 30!!" << endl;
  }
  else
  {
    cout << "age was over the limit, but not 30!!" << endl;
    cout << "It was: " << ageYears << endl;
  }

  return 0;
}
```

age was 30!!

Exactly 1 of the branches will be executed in this case, always.

Without an "else", exactly 0 or 1 of the branched would be executed.

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


```
char sizeLetter;

cout << "Enter size (s, m, l): ";
cin >> sizeLetter;
switch (sizeLetter)
{
  case 's':
    cout << "Small" << endl;
    break;
  case 'm':
    cout << "Medium" << endl;
  case 'l':
    cout << "Large" << endl;
    break;
  default:
    cout << "Invalid size!" << endl;
}
```

```
Enter size (s, m, l): s
Small
```

```
Enter size (s, m, l): m
Medium
Large
```

```
Enter size (s, m, l): r
Invalid size!
```

Duplicated Code = Bad

- Style: Avoid code duplication

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()
{
  int curNum = 1;  //Loop condition value
  int factVal = 1; //Factorial

  while (curNum <= 5)
  {
    factVal *= curNum;
    curNum++;   //Don't forget to modify num!
  }
  cout << "5 factorial is: " << factVal << 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()
{
  int curNum = 1;  //Loop condition value
  int factVal = 1; //Factorial

  do
  {
    factVal *= curNum;
    curNum++;
  }
  while (curNum <= 5);

  cout << "5 factorial is: " << factVal << 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()
{
  int curNum;  //Loop variable - no need to initialize
  int factVal = 1; //Factorial

  for (curNum = 1; curNum <= 5; curNum++)
  {
    factVal *= curNum;
  }

  cout << "5 factorial is: " << factVal << endl;

  return 0;
}
```

Everything you need to know about how the loop works is right here!

One single (compound) statement.

5 factorial is: 120

Short Circuiting with || and &&

- C++ uses something called “short circuiting”
- When the result of a logical expression is known, it stops checking the rest of the conditions
- We’ll see why this matters later on…

```
int value = 10;

if (value > 4 || value < -50)
{
  //...
}

if (value > 100 && value < 200)
{
  //...
}
```

Since the “logical or” results in true if *either* of its operands is true, the right hand side expression isn’t even evaluated! Once it sees “value > 4” is true, it already knows the whole expression is going to be true.

Since the “logical and” results in false if *either* of its operands is false, the right hand side expression isn’t even evaluated! Once it sees “value > 100” is false, it already knows the whole expression is going to be true.

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