

Assignment and Interactive Input

CS 1: Problem Solving & Program Design Using C++

Objectives

- Describe assignment operators
- Functionally look at mathematical library functions
- Get proactive with interactive keyboard input
- Wrap ourselves around symbolic constraints
- Discover common programming errors

Assignment Operators

- The assignment operator assigns values of the correct type to variables
- Basic Assignment Operator
 - Format: variable = expression;
 - The equals sign in an assignment statement
 - First computes value of expression on right of = sign
 - Then assigns the value computed in Step 1 to the variable on left side of = sign (it stores the value in the variable location)

```
int length
```

```
length = 25;
```

“length is assigned the value 25”

Assignment Operators (2)

- If not initialized in a declaration statement, a variable should be assigned a value before used in any computation
 - Think of the flow of control in the main function of the program
 - Statements are executed one at a time, the computer has no knowledge of what the next statement will be when it is executing the current statement
 - It cannot “look ahead” for a variable assignment when using a variable in a calculation
 - If you attempt to use a variable that was not initialized or assigned a value in a computation the compiler will attempt to put whatever value happened to be in that memory slot into your variable producing unwanted results
 - For the same reason you cannot use a variable before it has been declared
- Variables can only store one value at a time
 - Subsequent assignment statements will overwrite previously assigned values

Assignment Operators (3)

- Operand to right of = sign can be:
 - A constant
 - A variable
 - A valid C++ expression
- Operand to left of = sign must be a variable
 - `amount + 1000 = 15-2; //is invalid`
 - The expression on the right side of the assignment operator evaluates to 13, which then to complete the operation needs to be stored in a variable
 - Since `amount + 1000` is not a valid variable name the computer does not know where to store the calculated value of 13
- If operand on right side of the = is an expression:
 - All variables in expression must have a value to get a valid result from the assignment

Assignment Operators (4)

- **EXPRESSION:** any combination of constants and variables that can be evaluated to yield a result
 - Regular precedence and associativity rules apply when evaluating the left side of an assignment operation
 - Examples of valid assignment expressions with valid expressions on the right side of the = sign

`sum = 3 + 7;`

`diff = 15 - 6;`

`product = .05 * 14.6;`

`tally = count + 1;`

`newTotal = 18.3 + total;`

`average = sum / items;`

`slope = (y2 - y1) / (x2 - x1);`

Assignment Operators (5)

- CONCEPT: In C++ it is important to realize that the equals sign used in the assignment statement is an operator (called the assignment operator)
- The assignment operator = has a lower precedence than any other arithmetic operator and this is why the value of the expression to the right of the equals sign is always evaluated first

Assignment Operators (6)

- Assignment statements themselves produce a value

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

```
int main()
{
    int a = 5;
```

```
    cout << "The value of the expression a=5 is " << (a = 5) <<
endl;
```

```
    return 0;
```

```
}
```

- The output is: The value of the expression is 5
- The significance of this will become more apparent when we look at relational operators and conditions for if/else statements

Assignment Operators (7)

- The assignment operator has right to left associativity
 - The statement: `a = b = 5;`
 - First assigns the value of 5 to the variable b and then assigns the value of b (which is 5) to a

Assignment Operators Example

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    double length;
    double width;
    double area;

    length = 27.2;
    width = 13.6;
    area = length * width;

    cout << fixed << setprecision(2);
    cout << "The length of the rectangle is " << setw(6) << length << endl;
    cout << "The width of the rectangle is " << setw(6) << width << endl;
    cout << "The area of the rectangle is  " << setw(6) << area << endl;

    return 0;
}
```

Assignment Operators Example (2)

- Shows an example of what would happen if values are not assigned to length and width before they are used for a calculation
- Note the pattern I used for this program, this has a basic logic you will repeat for a lot of your programs
 - Declare variables needed in program: width, length, area and set initial values
 - Get user input and assign values to variables (we'll learn this in a minute)
 - Perform calculations on the input
 - Display results (output)

Coercion

- The value on right side of a C++ assignment expression is converted to data type of variable on the left side
 - COERCION: converting data to the type of the variable it is assigned to (when possible)
 - You need to always know what DATA type your data is...keep track
 - Notice this is different from the strict typing rules of Java
 - C++ allows the programmer to make data type conversions that may result in a loss of data

Coercion Example

- If temp is an integer variable, the assignment

```
temp = 25.89;
```

- causes integer value 25 to be stored in integer variable temp
 - The assignment here truncates (doesn't round) the double value 25.89 when it is converted to an int

- If temp is a double the assignment

```
temp = 25;
```

- causes float value 25.0 to be stored in the float variable temp
 - The .0 added to 25 indicates the conversion from an int to a double
- NOTE: Remember that even though, in this example, 25.0 is converted to a double, `cout` formats it as 25 rather than 25.0 when it outputs to the screen
 - To get `cout` to display the .0 you need to use `setprecision`

Coercion Example #2

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    int x;
    double y;

    x = 25.75;
    y = 5;

    cout << "The value 25.75 was coerced to the integer value " << x <<
endl;
    cout << fixed << setprecision(2)
        << "The value 5 was coerced to the double value " << y << endl;

    return 0;
}
```

Assignment Variations/ Shortcut Operators

- Assignment expressions such as:

```
sum = sum + 25;
```

- This expression is evaluated in two steps (remember the two steps for assignment)
 - The first step is to calculate the value `sum + 25`
 - The second step is to store the computed value in the `sum` variable
- This type of expression can be written by using following shortcut operators:

```
+= -= *= /= %=
```

Assignment Variations/ Shortcut Operators

- Example:

`sum = sum + 10;`

- can be written as

`sum += 10;`

- It's important to note that the variable to the left of the assignment operator is applied to the complete expression on the right

`sum *= a + 10;`

IS: `sum = sum * (a + 10);`

NOT: `sum = sum * a + 10;`

Mathematical Library Functions

- Standard preprogrammed functions that can be included in a program
 - EXAMPLE: `sqrt(number)` calculates the square root of number
 - The number passed into the square root function is called the argument of the function and constitutes the input data of the function
 - Once the `sqrt` function receives the input data it performs mathematical operations on it and returns the number resulting from those mathematical operations to the calling function
 - Notice that the input to the `sqrt` function must be a real number

Mathematical Library Functions

- Notice that the input to the `sqrt` function must be a real number
 - The real number may be of type `float`, `double` or `long double`
 - This is called function overloading
 - Function overloading permits the same function to be defined for different argument types
 - In this case, there are three functions defined for `sqrt`, one that takes a `float` argument, one that takes a `double` argument, and one that takes a `long double` argument
 - Function overloading will become an important concept to understand as we progress in learning C++

Mathematical Library Functions Example

```
#include <iostream>
#include <cmath>

using namespace std;

int main()
{
    double x;

    x = sqrt(4.2);

    cout << "The output is " << x << "\n";

    return 0;
}
```

Mathematical Library Functions Example (2)

- Notice the header `#include <cmath>` is used in this program
 - The `sqrt` function won't work without it because the `sqrt` function is defined in the `cmath` library
- What happens if you sent an `int` to the `sqrt` function instead of a real?
 - The compiler automatically promotes the integer (`int`) to a data type the function accepts
 - This may differ by compiler

More on Mathematical Library Functions

- Before using a C++ mathematical function the programmer must know
 - Name of the desired mathematical function
 - What the function does
 - Type of data required by the function (input)
 - Data type of the result returned by the function (output)
- Remember what functions are and what their point is
 - A function is good if it can be reused by many programs like the `sqrt` function can
- In a couple of lectures, you will be writing your own functions and people using your functions will need to know the same information to use them properly

Commonly Used Mathematical Library Functions

- To access these functions in a program, the header file `cmath` must be used in your program
 - Format: `#include <cmath>` <- no semicolon

Function Name	Description	Returned Value
<code>abs(x)</code>	Absolute value	Same data type as argument
<code>pow(x1, x2)</code>	x_1 raised to the x_2 power	Data type argument of x_1
<code>sqrt(x)</code>	Square root of x	double
<code>sin(x)</code>	Sine of x (x in radians)	double
<code>cos(x)</code>	Cosine of x (x in radians)	double
<code>tan(x)</code>	Tangent of x (x in radians)	double
<code>log(x)</code>	Natural logarithm of x	double
<code>log10(x)</code>	Common log (base 10) of x	double
<code>exp(x)</code>	e raised to the x power	double

Casts

- CAST: forces conversion of a value to another data type
 - Two versions: compile-time and run-time
- COMPILE-TIME CAST: unary operator with syntax
`dataType(expression)`
 - expression converted to data type of `dataType`
 - Old C-style compile time casting may also be used
`(dataType)(expression)`
- RUN-TIME CAST: requested conversion checked at runtime, applied if valid
 - SYNTAX: `staticCast<dataType>(expression)`
 - expression converted to data type of `dataType`

Casts Example

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

int main()
{
    double x;

    x = 4.2;

    cout << "When x is cast to an int the value is " << (int)x << '\n';

    return 0;
}
```


Interactive Keyboard Input

- A good program needs to be able to get input from the user
- cin object: used to enter data while a program is executing
 - Example: `cin >> num1;`
 - Statement stops program execution and accepts data from the keyboard
- The cin object allows the user to enter a value at the terminal (keyboard)
 - The value the user enters is then directly stored in a variable

Interactive Keyboard Input Example

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

int main()
{
    double num1;
    double num2;
    double product;

    //prompt the user for information
    cout << "Please input a number: ";
    cin >> num1;
    cout << "Please input another number: ";
    cin >> num2;

    //use input received from the user for the computation of the product
    product = num1 * num2;

    //output results
    cout << num1 << " times " << num2 << " is " << product << endl;

    return 0;
}
```

About the Interactive Keyboard Input Example

- First `cout` statement in Example program prints a string
 - Tells the person at the terminal what to type
 - A string used in this manner is called a prompt
- Next statement, `cin`, pauses computer (what `cin` does)
 - Waits for user to type a value
 - User signals the end of data entry by pressing Enter key
 - Entered value stored in variable to right of extraction symbol
 - `>>` is called an extraction symbol, as it extracts a stream of data from the keyboard and puts the data into a variable
- Computer comes out of pause and goes to next `cout` statement which in this program is another prompt

Continuing on Interactive Keyboard Input

- In general basic programs will follow this sort of pattern:
 - Declare variables
 - Prompt user for input
 - Get user input
 - Perform some sort of computations on user input
 - Output the results
- The `cin` statement can also be used to enter and store as many values as there are extraction symbols `>>`
 - Example: `cin >> num1 >> num2;`
 - This results in two values being read from the terminal and into the variables `num1` and `num2`
 - When the numbers are being entered into the keyboard at least one space must be put between them in order for them to be read into the two variables
 - Inserting more than one space has no effect on `cin`
 - As long as no space is entered the program stays in pause mode waiting for the second number to be input

Continuing on Interactive Keyboard Input (2)

- When invalid input is entered into `cin` the operator can do some simple conversions but these conversions can cause results that are not desired
- For example, if `num1` above was an `int` and `num2` was a `double` and the data input into the keyboard was:

22.83 1

- The computer would stop reading after the decimal point in the 22 assuming the decimal point indicated the end of the integer
 - 22 would be stored in `num1`
 - .83 would be stored into `num2`
 - 1 would be considered extra input and would be ignored

Interactive Keyboard Input Example #2

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

int main()
{
    double num1;
    double num2;
    double sum;

    //prompt the user for information
    cout << "Please input two numbers: ";
    cin >> num1 >> num2;

    //use input received from the user for the computation of the sum
    sum = num1 + num2;

    //output results
    cout << num1 << " plus " << num2 << " is " << sum << endl;

    return 0;
}
```

A First Look at User-Input Validation

- A well-constructed program should validate all user input
 - Ensures that program does not crash or produce nonsensical output
- ROBUST PROGRAMS: programs that detect and respond effectively to unexpected user input
 - Also known as bullet-proof programs
- USER-INPUT VALIDATION: validating entered data and providing user with a way to re-enter invalid data
- A simple way to check input in your programs is to use a cout statement to print the value in the variables you read in
 - So if your program is not working properly this is one way you can quickly check to see if you have a data input problem

Symbolic Constants

- CONCEPT: Constants may be given names that symbolically represent them in a program
- MAGIC NUMBERS: literal data used in a program over and over
 - Some have general meaning in context of program
 - Tax rate in a program to calculate taxes
 - Others have general meaning beyond the context of the program
 - $\pi = 3.1416$
 - Euler's number = 2.71828
- Constants can be assigned symbolic names

```
const float PI = 3.1416f;
```

```
const double SALESTAX = 0.05;
```


Symbolic Constants (2)

- `const`: qualifier specifies that the declared variable and the value assigned to it cannot be changed
 - Once you have specified the identifier that is tied to the literal it cannot be changed in the program by any type of operation
- A `const` identifier can be used in any C++ statement in place of number it represents

```
circum = 2 * PI * radius;
```

```
amount = SALESTAX * purchase;
```

Symbolic Constants (3)

- `const` identifiers commonly referred to as symbolic constants
- Advantages of using a symbolic constant
 - Named constants
 - If it is used several times in the program and its value changes (like a sales tax rate) you only have to change it once
 - This prevents mistakes in trying to change it several times and possibly missing a case
 - This also saves time

Symbolic Constants Example

```
#include <iostream>
#include <iomanip>

using namespace std;

int main()
{
    const double SALES_TAX = .05;

    double amount;
    double taxes;
    double total;

    //prompt the user for information
    cout << "Enter the amount of your purchase before sales tax $:";
    cin >> amount;
```

Symbolic Constants Example (2)

```
//use input received from the user for the computation of the total
price
    taxes = amount * SALES_TAX;
    total = amount + taxes;

//output results
cout << fixed << setprecision(2);

    cout << "\nThe taxes at a sales tax rate of " << SALES_TAX << " is "
<< taxes;

    cout << "\nThe total price including taxes is $" << total << endl;

    return 0;
}
```

Placement of Statements

- A variable or symbolic constant must be declared before it is used
- C++ permits preprocessor directives and variable declaration statements to be placed anywhere in program
 - Doing so results in very poor program structure
 - It's always best to structure your program with preprocessor directives at the top and symbolic constants and variables at the top of the function they belong to

Placement of Statements (2)

- As a matter of good programming practice, the order of statements should be:

preprocessor directives

int main()

{

 symbolic constants

 variable declarations

 other executable statements

 return value

}

Common Programming Errors

- Forgetting to assign or initialize values for all variables before they are used in an expression
- Forgetting to separate all variables passed to cin with an extraction symbol, >>

Summary

- **EXPRESSION:** sequence of operands separated by operators
- Expressions are evaluated according to precedence and associativity of its operands
- The assignment symbol, `=`, is an operator
 - Assigns a value to variable
 - Multiple assignments allowed in one statement
- C++ provides library functions for various mathematical functions
 - These functions operate on their arguments to calculate a single value
 - Arguments, separated by commas, included within parentheses following function's name
- Functions may be included within larger expressions

Summary (2)

- `cin` object used for data input
- `cin` temporarily suspends statement execution until data entered for variables in `cin` function
- Good programming practice: prior to a `cin` statement, display message alerting user to type and number of data items to be entered
 - Message called a prompt
- Values can be equated to a single constant by using the `const` keyword

Summary (3) – Key Terms

- **ARGUMENTS:** items that are passed to a function through parentheses
 - Think of arguments as the input to a function
 - This input will then have calculations performed on it and some sort of result will be passed back to the program using the function.
- **ASSIGNMENT STATEMENT:** a C++ statement that tells the computer to determine the value of the operand to the right of the equals sign and then store (or assign) that value in the locations associated with the variable to the left of the equals sign
- **CAST:** the operator used to force the conversion of a value to another data type
- **COERCION:** a conversion of the value of the expression on the right side of the assignment to the data type of the variable to the left of the assignment operator

Summary (4) – Key Terms

- **EXPRESSION:** any combination of constants and variables that can be evaluated to yield a result
- **LITERAL:** a data element within a program that explicitly identifies itself
- **OVERLOADING:** a property that permits the same function to be defined for different argument data types
- **ROBUST PROGRAM:** a program that detects and responds effectively to unexpected user input
- **SYMBOLIC CONSTANT:** an identifier that has been equated to a constant in a declaration statement
- **TRUNCATION:** discarding or losing the fractional part of a value