Variables, cin
Problem Solving

CSE100
Sections 2.6, 3.1, 1.5, 1.6

From last time...

Data Types in C++

- Data Type: set of values together with a set of operations is called a data type

- Two broad categories of simple data types
  - numeric
    - Integral: integers (numbers without a decimal)
    - Floating-point: decimal numbers
  - character

int Data Type

- Can take any integral value between -2147483648 and 2147483647
- Stored in 4 bytes of memory
- No commas are used within an integer
- int is a keyword

Variables

- A *variable* is a named location in main memory.
  - Called variable because the value stored at that location may change

- Variables consist of two parts:
  - A data type, so the computer knows how to store the data and the operations that can be performed on it.
  - An *identifier*, the name we give the variable

Variables

- *Variable Definition* - Assigning the variable a specific place in main memory.
  - Must be done before the variable can be used.
  - Done with code: <data type> <identifier>;
    - Example: int number;

- *Variable Initialization* - Assigning a value to the variable for the first time.
  - int number = 5;

Forming Identifiers

- Cannot be keywords
- Must begin with a letter (a-z or A-Z) or underscore (_)
- Can be followed by letters, digits (0-9), and the underscore character (_)

- C++ is case sensitive
  - Name and name are different identifiers
  - payRate, payrate and PayRate are all different

= Operator

- Called the *Assignment Operator*
- Evaluates the expression on the right and stores it in the variable on the left
  - Always need to have a variable on the left
  - The expression on the right, must evaluate to a value of the variable's data type
  - Any previous value at that location is replaced

- Example:

```
1. int num;
2. num = 5;
3. num = num + 12;
4. int num2 = 8;
5. num = num2 + 5;
6. num = num + num2 * 2;
```

Operators

Operator	Associativity	Description
()	left to right	Group operations
+, - (unary)	right to left	negate a number
*,/,%	left to right	arithmetic
+, - (binary)	left to right	arithmetic
<<	left to right	stream insertion
=	right to left	assignment
,	left to right	list separator

= Operator

- Example of WRONG use:
  - 4 + num2 = num;

- Can chain together assignments
  - Always operates from right to left
- Example:
  - int num1, num2;
  - num1 = num2 = 5*3;

Variable Output

- To output variables of simple data types, insert the variable name into the stream
  - Note the variable name does not go in quotes
    - What would happen if it did?
  - With variables number1, number2 and sum

```
cout << "\nThe sum of " << number1 << "
  and " << number2 << " is  " << sum <<
  endl;
```

- Be careful to use spaces in the strings around the insertion of the variables for readability.

Arithmetic Operators

- Work the same way as for int constants
- Need to always put the operator in between constants and variables
- Can use directly in output stream

Examples:

```
int x=2, y=8, z=5;
cout << 2*x+y;  // not 2x+y
cout << (x+y) / (z+2);  //What happens if missing ()
cout << x*y+z;   // not xy + z
```

Tips for Variables

- A variable MUST be defined before it's used
- Only define a variable once
- Names should be meaningful
  - If the variable is going to hold the length, should call it *length*, rather than l or x
  - Makes the program self-documenting
- By convention, variable names should start with a lowercase letter
- If the name is more than one word
  - Start the second word with a capital letter (called camelCase)
    - int payRate;
  - OR separate the words with an underscore
    - int pay_rate;

Program Example: Variables

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


int main()
{
   int number1, number2 = 7;
  number1 = 5;

  int sum = number1 + number2;

  cout << "\nThe sum of " << number1 << " and " << number2
  << " is " << sum << endl;

   return 0;
}
```

Variable Problem

Write a program that uses variables to find the perimeter of a 4 by 3 rectangle. Use variables for the length, width and perimeter.

cin Object (Basics)

The `cin` Object

- The cin object allows the user to get data from the keyboard (stands for **c**onsole **in**)
- To use cin, you need to include the iostream library and use namespace std
- User input goes from the keyboard to the *input buffer*, where it is stored as characters
- **`cin`** tries to convert the data to the type that matches the variable

```
int number1;
cout << "Enter the first number: ";
cin  >> number1;
```

- >> is called the ***stream-extraction operator***
  - must be followed by a defined variable

The `cin` Object

One should always ***prompt*** the user when using cin so that the user knows what the expected input is

```
cout <<"Enter the first number: ";
```

- You can't mix input and output
  - cout << "Enter a number: " >> number1 << endl;
  - This will cause an error

The `cin` Object

- Can be used to input multiple values

```
cin >> number1 >> number2;
```

- Multiple values from keyboard must be separated by spaces or [Enter]
  - cin will ignore the whitespace between int values
- Must press [Enter] after typing last value
- Order is important; first value entered is stored in first variable, etc.
- Multiple values need not all be of the same type

cin Problem 1

Write a program to find the sum of two integers.

cin Example

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


int main()
{
   int number1, number2;
   cout<<"Enter two integers for which you want to find the
   sum: ";
   cin >> number1 >> number2;

   int sum = number1 + number2;

   cout << "\nThe sum of " << number1 << " and " << number2
   << " is  "<< sum << endl;

   return 0;
}
```

cin Problem

Write a program to find the perimeter and area of a rectangle. Be sure to prompt the user for any input they may need to provide.

Computational Thinking

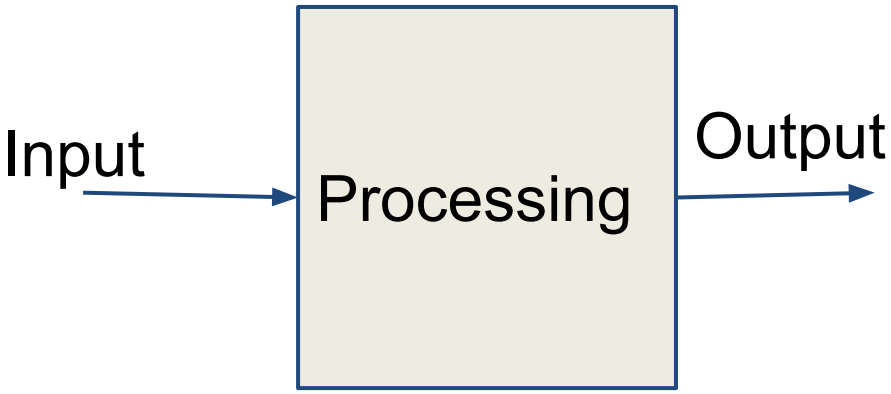
Computational Thinking = Critical Thinking + Computational Power

Computational Thinking is the thought processes involved in formulating problems and their solutions so that the solutions are represented in a form that can be effectively carried out by an information-processing agent [CunySnyderWing10]

Problem Analysis

- What are the requirements of the problem statement. Are they clear? Do they make sense?
- Abstraction

  Being able hide details and focus on main artifact of the solution in more generalized way. In other words, answer the question **what but not how**.
- What is the input required?
- What processes need to be performed?
- What is the expected output?



Data Representation

- First step of answering how
- How should we represent our input/output?
- Are the expected values integral, decimal, or strings (words)?
- Should we abstract to other data types?
- What should the program look like to the user?

Decomposition

- Divide a larger problem into smaller (hopefully easier) subproblems
- Solve those problems separately
- Combine the solutions of the smaller subproblems to solve the larger problem
- Also known as 'Divide and Conquer', *Modular Programming*, or *top-down design*
- Uses a *hierarchy chart* to help design
- We will be studying functions and Object Oriented ideas this semester.

Algorithms

- *Algorithm* - a clearly defined (unambiguous) finite set of steps (each doing a finite amount of work) with a terminating condition to solve a particular problem.

- When we are writing algorithms, we use
  - Sequence - do the steps in order
  - Selection - if a condition is true do something, otherwise do something else
  - Repetition - repeat a set of steps until a condition is false
- Every algorithm can be written as a combination of those three components

Algorithms (continued)

- Algorithms can be expressed in various different ways such as *flowcharts* and *pseudocode*
  - *flowcharts* are a graphical representation that shows the logical flow of the program and the order each instruction is performed in
  - *pseudocode* is a cross between human language and programming languages
    - Easy to write as there are no special rules to be followed
    - Close enough to programming languages to easily be implemented

Example 1 Cont…

- Rectangle Algorithm:
  - Get length of the rectangle
  - Get width of the rectangle
  - Find the perimeter using the following equation:

```
perimeter = 2 * (length + width)
```

  - Find the area using the following equation:

```
area = length * width
```

-Display the result

Tips for getting started with Algorithms

- If you are stuck trying to design an algorithm for a problem
  - Try to work a few easy sample problems by hand
  - *Pattern Recognition* - Look for patterns in the solving process
  - *Generalize* - Try to replace the numbers with variables in the process
  - Test your generalized solution for logic errors with a more complex sample

Implement and Compile

- Once you have decided on an algorithm and checked to ensure that the logic is correct, you are ready to implement it in a programming language
- When you are finished compile the source code and check for any errors
- Fix any compilation errors

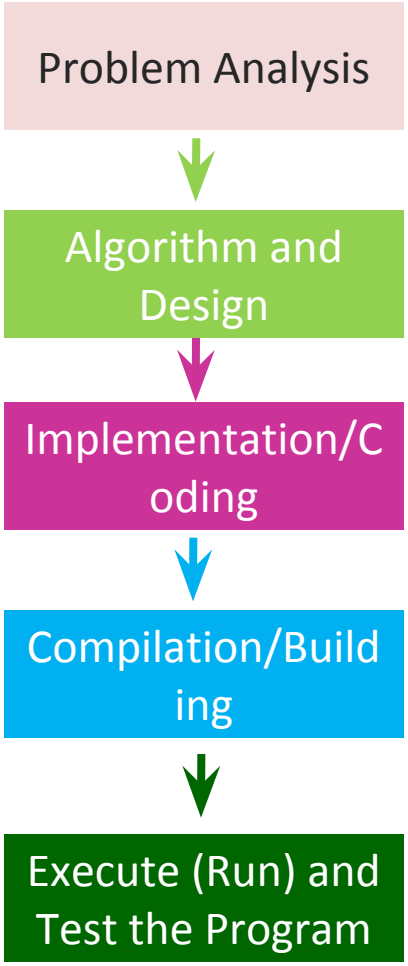
Testing

- It is important to test your executable code once it compiles
- Unit testing - you should test each solution to a subproblem separately before integrating it into the final program
- Make an appropriate set of test cases where you have the input and expected output
- Validate that your program meets all the problem requirements

Software Engineering

- The engineering field that encompasses the complete process of crafting computer software from determining project requirements to testing and maintenance
- Software Engineers often work in large teams and use many tools to design and implement efficient and safe designs and algorithms

Problem Solving Process Summary



Analysis: inputs, outputs, and the data processing requirements

Algorithms/Design: pseudo code, flowchart. Consider alternative solutions and safety and security issues with your design.

Implementation: use a suitable programming language to implement your solution. Make sure to test it.

Compilation: make the executable code

Execute the program: error free execution gives the correct result. Otherwise go back to implementation, design, or problem analysis and solve the problem.

Formatting Source Code

The standard conventions should be followed when writing source code to help *readability* (the ease with which text can be read and understood):

- Use blank lines to separate logical parts of the program
- Use comments to describe the major parts of the program
- Use consistent indentation inside braces
- Follow variable naming conventions

cin Problem 3

Write a program that will take as input a height in inches and output the height in feet and inches.

For example, if the user enters 71 inches, then the output would be 5’11”.

cin Problem 4

Find and fix all mistakes in the following section of code which should find the quotient and remainder from dividing two integers.

```
int numerator, denominator;
cin << denominator, numerator;
cout >> "Enter two ints to perform division on (e.g. 5/3 enter 5 3): "
quotient =  numerator/denominator;
numerator%denominator;
cout >> "/nnumerator" >> " divided by " >> denominator >> "has quotient " >> Quotient >> and remainder >> "remainder" >> "endl";
```

Next Time

- Other integral data types (2.7)
- Floating Point Data Types (2.8)
- type conversions: implicit and explicit (3.3)

Book Problems

The following problems are for extra practice, but are not due.

Checkpoint questions on pages 41, 48 (2.10, 2.11, 2.14, 2.15 only), and 90 (3.9-3.11 only). (answers in Appendix C)

Review Questions page 69-72 : 9, 13, 16, 25A,B, 26B

Programming Challenges page 73: 1, 14