Problem Solving

CSE100
Sections 1.5, 1.6

cin Problem

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";
```

cin Problem 2

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.

Solution 2

```
#include <iostream>
using namespace std;
int main()
{
    int length, width;

   // get input from user
    cout << "Enter the length of the rectangle: ";
    cin >> length;
    cout << "Enter the width of the rectangle: ";
    cin >> width;

    // calculate perimeter and area
    int perimeter = 2 * (length + width);
    int area = length * width;

     // display results
    cout << "\nThe perimeter of the rectangle is " << perimeter << endl;
    cout << "\nThe area of the rectangle is " << area << endl;
    return 0;
}
```

cin Problem 3

Write a program to find how many minutes have passed since midnight and how many minutes are remaining in the day.

Assume the user will enter the time using the 24-hour clock format with no symbol between the hour and minutes (e.g. 3:25pm will be entered as 1525).

Computational Problem Solving

Computational Thinking

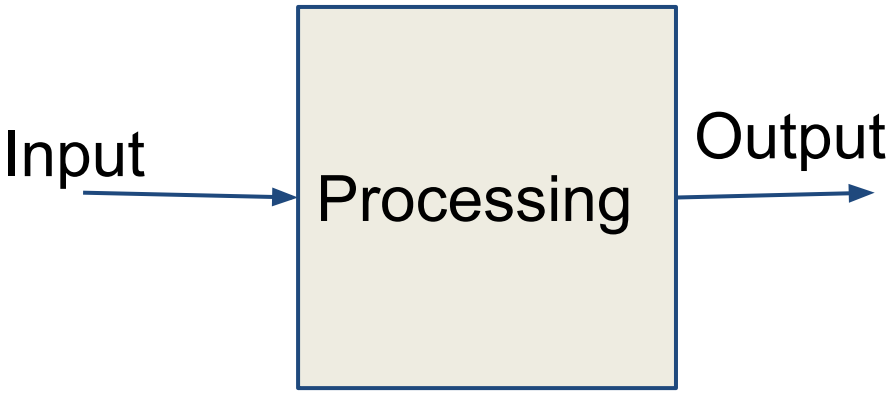
Computational Thinking = Critical Thinking + Computational Power

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

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

Example Cont…

- Minutes Algorithm:
  - Get time from the user
  - Separate time into hours and minutes
  - Find how many hours have passed since midnight

```
hours since midnight = 60 * hours + minutes
```

  - Find how many hours are left in the day

```
hours left in day = hours in day - hours since
  midnight
```

-Display the results

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
- *Syntax Errors* - Errors in the program that violate the rules of the language. Found at compile time
- Fix any compilation errors

```
#include <iostream>
using namespace std;
int main()
{
    // get the input
    int time;
    cout << "Enter the current time using 24-hour time "
                << "\nwith no space between the hours and minutes: ";
    cin >> time;

    // separate the hours and minutes of the time
    int hours = time / 100;
    int minutes = time % 100;

    // find how many minutes have passed since midnight
    int minutesSinceMidnight = 60 * hours + minutes;

    // find how much time is left in the day
    int minutesLeftInDay = 60*24 - minutesSinceMidnight;

    // print out the results to the user
    cout << "\nIt has been " << minutesSinceMidnight << " minutes since midnight."
<<endl;
    cout <<"There are " << minutesLeftInDay << " minutes remaining in the day." << endl;
     return 0;
}
```

Testing

- It is important to test your executable code once it compiles
- Finds *Logical Errors* - errors in the logic of the program
  - Ex. multiplying instead of adding
- Also finds *Runtime Errors* - Errors that occur while the program in running that causes the program to crash
  - Ex: accessing an invalid memory location

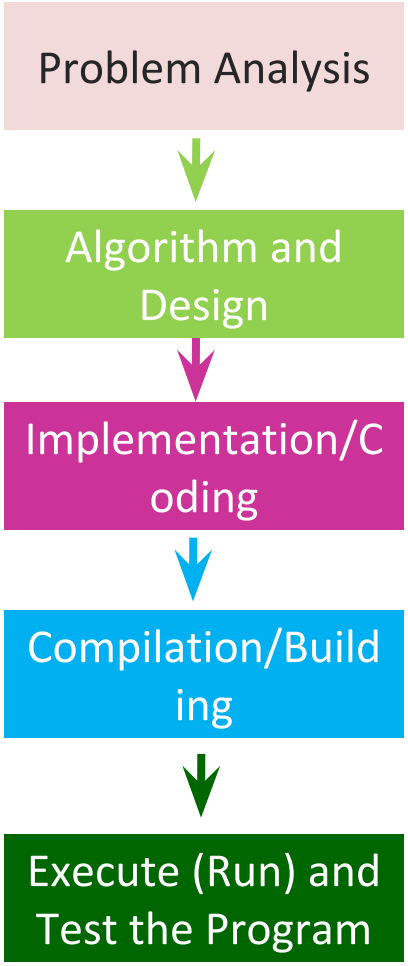
Testing

- Unit testing - you should test each solution to a subproblem separately before integrating it into the final program
  - Test as you 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 4

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”.

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