Designing a Program

- **Algorithm** is a series of steps that carry out a specific *task*
- This is the basis of all computer programs
- They are independent of any single programming language

Algorithms

- They are the logic on *how* to do something
  - how to compute the value of Pi
  - how to delete a file
  - how to load a webpage
- Examples:
  - tying your shoes
  - driving to Sac State
  - making a peanut butter and jelly sandwich

Designing a Program

- The process of designing a program consists of a series of steps
- These include the conceptualization of the problem, and creating the logic on how to address it
- It also includes the steps to make the program work correctly

Designing a Program

1. Design – flowcharts and pseudocode help with this process
2. Write the code
3. Cleared code of syntax errors
4. Checked for logic errors.
5. If logic errors exist, debug the program
Two Steps of Designing a Program

1. Understand the tasks that the program is to perform
2. Determine the steps require for the task
   • create an algorithm
   • use flowcharts and/or pseudocode to solve

Procedural Programming

- Traditional approach to programming
  - easy for humans to conceptualize
  - it is based on how we do things
- Programs
  - contain as a sequence of steps
  - steps run in order – from first to last
  - repetition is performed with “looping”

Flow Chart Overview

- Graphical representation
  - each statement is a shape
  - useful for conceptualizing an algorithm
  - easy to understand and visualize
- Used everywhere
  - programmers used to create charts first
  - used extensively in business Worldwide

Start / End

- Indicates the start and end of an algorithm
- Represented by a rectangle with rounded sides
- There are typically two:
  - one to start the flowchart
  - one to end the flowchart

Input / Output

- Indicates data being:
  - inputted into the computer
  - outputted to the user
- Represented by a parallelogram
- Flowcharts can have many
Processes

- Indicates data:
  - being processed
  - also called "calculations"
- Represented by a rectangle
- This tends to be the most common symbol

Decisions

- Indicates a logical branch
  - algorithm executes instructions if some criteria is met
  - can contain a positive and negative branch
- Represented by a diamond

Pseudocode

"Fake" code used as a model for programs
- No syntax rules
- Well written pseudocode can be easily translated to actual code

What is Pseudocode?

- Description of an algorithm's logic
  - not as formalized as English
  - it can "look" like code, but this varies
- It is often useful
  - lets you write down "something" to conceptualize the algorithm
  - not worry about the implementation issues
Area of a Circle Pseudocode

My area of a circle program

Input radius
Set Area = (pi x radius^2)
Display "The area is", area

Flowchart vs. Pseudocode

Display "What is the bill?"
Input bill
Set tip = bill * 0.15
Display "The tip is $", tip

Output, Input, and Variables

- Computer programs typically follow 3 steps
- They are:
  1. Input is received
  2. Some process is performed on the input
  3. Output is produced

Output, Input, and Variables

- **Input**
  - refers to data that is received from the user
  - allows the program to solve different sets of problems without change
- **Output**
  - refers to data that is displayed to the user
  - this includes prompts and the results of calculations

Output, Input, and Variables

- **Variables**
  - storage locations in memory for data
  - used to: perform calculations, store results, hold messages, etc...
  - they are vital to understand
The textbook, we are using in class, makes use of both flowcharts and pseudocode to describe program logic.

- The pseudocode, throughout the book, is consistent - the same words and structure is used in every example.

**Pseudocode in the Book**

- Display is the "keyword" to show output to the screen.
- When you see a line that starts with "display", you will known information will shown.

**Book Pseudocode: Display**

- In computer science, a series of characters is called a *string*.
- Display often uses string literals – a sequence of characters delimited by double quotes.

**Examples of String Literals**

- "Sac State"
- "Hornet"
- "Computer Science 10"
- "1850"
- "Tappa Kegga Bru"
- "Cuppa Kappa Chino"

**Output Example**

- In the following example, a string literal is output to the screen.
- The book's pseudocode uses Display.

```
Display "Hello!"
```

**Output Example**

- When the program gets to the display statement, it will output the string literal.

```
Hello!
```
Output Example 2

- You can output numbers and strings
- The following outputs the name and founding year of Sac State

```
Display "Sacramento State"
Display 1947
```

Output Example

- We will get into what the difference between the two pieces of data later

```
Sacramento State
1947
```

Book Pseudocode: Input

- **Input** is the keyword to read values from the user
- These values...
  - need to be put somewhere
  - we will put them in memory locations called variables

Variable Declarations & Data Types

- A **variable declaration** tells the programming language the name of a variable you want and what it will store
- Found in practically all programming languages

Explicit Variable Declaration

- **Explicit** variable declaration require the program to specify the name and type of each variable
- Prevents typo bugs (which are hard to find and fix)
Explicit Variable Declaration

- Commonly used in high-level & professional languages
- Examples:
  - Java
  - Visual Basic .NET
  - C#
  - C++

Implicit Variable Declaration

- *Implicit* variable declaration creates a variable the moment it is first "seen"
- Typically used by scripting languages – but few major languages

Implicit Variable Declaration

- Tends to be problematic since a simple typo will create new variables
- For example:
  - the programmer uses *value* and *valu* (no e)
  - it will create two variables even though the user meant *value* for both
- These can be hard to find and fix

Data Types

- A *data type* – defines the type of data you intend to store in a variable
- The names of data types and which ones are supported varies greatly between languages

Common Data Types

- Integer – stores only whole numbers
- Real – stores whole or decimal numbers
- String – any series of characters

Declaring Variables

- When you *declare* a variable
  - you allocate space for it in memory
  - and give this memory a unique name
- This unique name is called an *identifier*
  - every programming language has different rules for how the identifier should look
  - fortunately, the main rules are pretty consistent
Typical Naming Rules

1. Letter followed by zero or more letters or numbers
2. One word - no spaces
3. Generally, punctuation characters are avoided – some languages allow underscores, dashes, etc…

Multiple Word Identifiers

- Often variable names are more than one word
- Goal of the name is to make it “readable” – not short!
- There are a couple of ways programmers format the name

Proper Case

- In proper case (aka pascal case), the first letter of each word is capitalized
- Standard convention for Visual Basic.NET and C#
- Examples:
  - AverageScore
  - TaxRate

Camel Case

- In camel case, the first letter of the first word is in lower case and the rest start with a capital
- Standard convention for C, C++, and Java
- Examples:
  - averageScore
  - taxRate

Declaration Example

- The follow declares a variable called age that can store integers
- The second variable, price, can store reals

```
Declare Integer age
Declare Real price
```

What Happens?

Memory:

```
year
? 
```

Declare Integer year;
What Happens?

Memory

While a variable is first declared — and created — what value does it have?

- In many languages, the initial value is unknown
- So, to avoid possible errors, variables should be initialized to 0 (or some other value) before it is used later

Variable Initialization

- Variable assignment does not always have to come from user input
- It can also be set through an assignment statement

Variable Assignment

- The *Set* keyword designates an assignment statement
- It allows a value to be stored in a variable without reading from the user
- It is also used to store the result of a calculation

Book Pseudocode: Set

- Declare Integer age
- Set age = 18

Declaration Example

- The first line creates the variable
- The second line, sets it to 18

What Happens?

Memory:

- Declare Integer year
- Set year = 1947
- Display year
What Happens?

Memory:

Declare Integer year
Set year = 1947
Display year

What Happens?

Memory:

Declare Integer year
Set year = 1947
Display year

Flowchart for Previous Slide

Another Example

Memory:

Declare Integer score
Set score = 98
Set score = 81

Another Example

Memory:

Declare Integer score
Set score = 98
Set score = 81

Another Example

Memory:

Declare Integer score
Set score = 98
Set score = 81

Another Example

Memory:

Declare Integer score
Set score = 98
Set score = 81
Another Example

Memory:

Declare Integer score
Set score = 98
Set score = 81

Data Conversion

- **Typecasting** – when one type of data is converted to another
- When the programming language converts the data implicitly, it is coerced
- When the programmer explicitly specifies how data will be converted, it is cast

Cast Example

Memory:

Declare Integer score
Set score = 98.6
Display score

Cast Example

Memory:

Declare Integer score
Set score = 98.6
Display score

Declare & Initialize

- Many programming languages allow you to declare and initialize at the same time
- The book also uses this notation

Declare Integer Age = 18

Assignment Example

- In the following example, the variable cash is set to the value 5.25
- The number 5.25 is written explicitly in the program and is called a literal

Set cash = 5.25
Display "I have ", cash, " dollars"
Assignment Example

- When the program gets to the display statement, it will use the current value of cash
- So, the variable can be used to output data

I have 5.25 dollars

Assignment Example 2

- Variables can be assigned more than once
- In this example, cash is set twice to different values

Set cash = 5.25
Display "I have ", cash, " dollars"
Set cash = 19.95
Display "Now I have ", cash, " dollars"

Assignment Example 2 Output

- The value of the variable will be printed using whatever value is currently has
- So, the program will print two different values

I have 5.25 dollars
Now I have 19.95 dollars

Variable Assignment & Calculations

- The value assigned to a variable doesn't have to be literal (string, integer, etc..)
- Often it is the result of a mathematical expression

Expressions

- Expressions are...
  - mathematical formulas
  - follows the format you know
- Operator Precedence
  - order which operators are computed
  - practically all languages have precedence levels
### Arithmetic Operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>Multiplication</td>
</tr>
<tr>
<td>/</td>
<td>Division</td>
</tr>
<tr>
<td>+</td>
<td>Addition</td>
</tr>
<tr>
<td>-</td>
<td>Subtraction (and unary minus)</td>
</tr>
<tr>
<td>mod</td>
<td>Modulus (remainder)</td>
</tr>
<tr>
<td>^</td>
<td>Exponent</td>
</tr>
</tbody>
</table>

### Precedence (from first to last)

<table>
<thead>
<tr>
<th>Operator</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Subexpression</td>
</tr>
<tr>
<td>^</td>
<td>Exponent</td>
</tr>
<tr>
<td>-</td>
<td>Unary minus</td>
</tr>
<tr>
<td>* / mod</td>
<td>Multiplication &amp; Division</td>
</tr>
<tr>
<td>+ -</td>
<td>Addition &amp; Subtraction</td>
</tr>
</tbody>
</table>

### Calculate Results

Set sum = \(11 \times 2 + 8 \times 4 - 7\)

Set sum = \(22 + (8 \times 4) - 7\)

Set sum = \(22 + 32 - 7\)

Set sum = \(54 - 7\)

Set sum = \(47\)

### Calculate 2 Results

Set result = \(7 + 3^2 \times (4 - 1)\)

Set result = \(7 + 3^2 \times 3\)

Set result = \(7 + 9 \times 3\)

Set result = \(7 + 27\)

Set result = \(34\)

### Calculate Results With Variables

Set \(x = 10\)
Set \(y = 2\)
Set value = \(x \times 3 + -y\)

Set value = \(x \times 3 + -2\)

Set value = \(30 + -2\)

Set value = \(28\)

### Named Constants

Chapter 2.5
**Named Constants**

- A **named constant** is a name that represents a value that **cannot** be changed
- Think of it like a variable that is "written in stone" – you can use it, but cannot change it

**Why use them?**
- Makes programs more self-explanatory
- If a change to the value occurs, it only has to be modified in one place

**Named Constant Example**

- The following declares a constant **TAX_RATE**
- Many languages like to type constants in all caps – the book uses this approach

```plaintext
Constant Real TAX_RATE = 0.07
```

**Hand Tracing a Program**

- **Hand tracing** is a simple debugging process for locating hard to find errors in a program
- Involves creating a chart with a column for each variable, and a row for each line of code
- You then pretend to be the computer (how fun!)
- Go line by line and update each variable as the program does
- This can help find bugs and help you understand the code
### Hand Trace Example: Cube

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>Side</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare Integer Side</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Declare Integer Volume</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Set Side = 4</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>Set Volume = Side ^ 3</td>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>Display Volume</td>
<td>4</td>
<td>64</td>
</tr>
</tbody>
</table>

### Hand Trace Example: Tax

<table>
<thead>
<tr>
<th>Pseudocode</th>
<th>Price</th>
<th>Tax</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare Real Price</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Declare Real Tax</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Declare Real Total</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Set Price = 32.50</td>
<td>32.50</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>Set Tax = Price * 0.08</td>
<td>32.50</td>
<td>2.60</td>
<td>?</td>
</tr>
<tr>
<td>Set Total = Price + Tax</td>
<td>32.50</td>
<td>2.60</td>
<td>35.10</td>
</tr>
<tr>
<td>Display Total</td>
<td>32.50</td>
<td>2.60</td>
<td>35.10</td>
</tr>
</tbody>
</table>

### Documenting a Program

- **External documentation** describes information that was written to benefit the user of the application.
- It is designed to be understood by your typical user – ideally being easy to understand by a novice.
- So, this type of documentation is sometimes written by a technical writer.

- **Internal documentation** is information left by the programmer for other programmers to read.
- It is included with the program itself – the end user will never read it or know it ever existed.

### Chapter 2.7

- Real-World programs can become large and complex.
- As a result, efforts are needed to keep them organized.
- And, to make it easier for them to be understood and edited by others.
In computer science, we refer to this documentation as comments. The comments depend on what the programmer is trying to tell the reader. Vital to maintaining any complex program.

Example Types of Comments
- Explanation of what a segment of code does
- Notes on future improvements
- Log changes made by a programmer – what, when and why
- Bugs and other problems

Book Pseudocode: Comments
- The book denotes a comment by using two normal slashes
- Everything after the slashes is a comment and does not change the meaning of your program

// This is a comment

In fact, comments never change your program – they are passive. But, programmers often "disable" code by "commenting it out"

// Display "This is disabled"
Display "This is not"

Commented Cube Example

Declare real side
Declare real volume

//Read the side of a cube
Input side

//Width * length * depth
//All three are equal on a cube
Set volume = side ^ 3
How to Format Your Code

- Computer programs can become complex
- As a result, it is a good idea to follow a set of rules to make sure they are readable and easy to maintain

### Style Guide

- Use descriptive names
- They should be long enough to describe the contents of the variable
- ... but keep them as short
- Using the correct naming convention & stick to it

### Variable Names

- The following variable names are far too long
- They are descriptive, yes, but take waaaaaay too much space

```plaintext
Set taxRateInSacramentoCounty = 0.08
Set taxOnMyBill = totalBillAmount * taxRateInSacramentoCounty
```

### Bad Variable Names

- Keep them short enough to not waste space
- But, still keep them descriptive

```plaintext
Set taxRate = 0.08
Set tax = bill * taxRate
```

### Better Variable Names

- Add comments to tell yourself (or a reader) what the program does
- It is also good to write down who edited the file and when
- Flowgorithm is pretty easy to understand, but many languages are not
It is considered good style to put all your variable declarations at the beginning of the program.

Why?
- keeps them all in one place
- insures you declared the variable before you use it

The following mixes declares and assignment statements
- It is hard to read – and easy to forget what you declared!

Always Declare Variables First

Mixed-up Declarations

Nice and Organized

Notice that the code is easier to read
- As an added benefit: it is easy to we can check if the data types are consistent (e.g. all reals)