# Problem-Solving

## Define the Problem

**Input** – any available information (data) that is useful in solving the problem

**Output** – the end result

**Processing** – the actions needed to produce the **Output** from the **Input**

# Input Data comes from Input Devices

## For example:

Keyboard

Mouse

Touchpad/Touchscreen

Scanner

Microphone

VideoCam

# Output Data is sent to Output Devices

## For example:

Printer

Screen

Speaker

# Processing

## All computer programs can be written using these 3

## *control structures*:

**Sequence**

**Selection**

**Repetition**

(more on this to come…)

# Defining Inputs and Outputs

One of the first steps in solving a problem is to define all the input data and the outputs for the problem, along with the *basic* processing steps needed to solve the problem.

## Take Note:

Processing details are hidden at this level. You only describe processing at a HIGH level – that is, a **bird’s-eye view**.

**Example:**

A program is required to get two numbers from the user, add them together, and display the total.

## Input

* First number
* Second number

## Output

* Total

## Processing

1. Get first number
2. Get second number
3. Add numbers together
4. Displaytotal

**NOTE:The order is critical. For example, you can't add the numbers together until you *get* the first and second number.**

#### A

**The Structure Theorem**

Any computer program can be written by solely using these

3 control structures:

**Sequence**

♠

**Selection**

♠

**Repetition**

♠

# Sequence

## Processing one action after another

C

**Examples:**

Driving Directions

Take I5S to Capitol Hwy. Exit.

Turn right at intersection.

Turn right at next intersection (Capitol Hwy).

Drive up hill about ½ mile.

Look for Portland Community College sign.

Turn right into PCC entrance.

**Notice once again how critical the order is. You MUST follow the directions in the order they are written.**

Calculate Profit

Read Price

Read Cost

Profit = Price – Cost

**It's the same for a computer program. The computer always processes the instructions (each line of code) in the order in which they appear.**

# Selection

(aka If Statements)

## Choosing an action based on a condition

**Examples:**

**D**

**If it’s raining, put on a raincoat:**

If raining Then

Wear raincoat

EndIf

**If it’s sunny, go to the beach, otherwise see a movie:**

If sunny Then

Goto beach

Else

See movie

EndIf

**Count the total number of males and females in the room:**

If Gender isFemale Then

Add 1 toTotalFemales

Else

Add 1 toTotalMales

EndIf

## Take careful note of the indentation above.

## Notice that the *action statements* are indented one tab stop.

# Repetition

**(aka Loops)**

Performing a set of actions over and over

as long as the condition is true

**OR**

a specific number of times.

**Examples:**

#### **E**

###### EATING

While Still hungry

Get a spoonful of cereal

Place spoon in mouth

Chew food

Swallow food

End While

**OR**

Repeat 10 times

Get a spoonful of cereal

Place spoon in mouth

Chew food

Swallow food

EndRepeat

## Again, look carefully at the indentation in the examples above.

## Notice that the *action statements* are indented one tab stop.

**F, B**

# Program Data

## Variable

G

A **variable** is an area of storage in memory that holds a *single piece of data*. A variable always has a ***name*** (so you can refer to it) and a ***value*** (what the variable actually holds – the contents of the variable).

**Examples of variables**

**Name Value**

CustomerName John

Age 27

## Literal

Better known among programmers as “hard-coding”.

Actual numbers or letters are used.

For example:

Display “Hello”

(Note: Strings must be in quotes. Otherwise, the computer will think it’s a variable.)

# Types of Variables

## String

Alphanumeric plus special characters (“129 Morrison St.”). Always has quotes around it. Pretty much anything can be stored in a String variable. *But you can NOT do math on it.*

## Number

A whole number, commonly called an **Integer,** or a decimal number (0, 27, 3.25, -42) usually called a **Decimal**. No quotes. Only digits and a negative sign are allowed (and for decimal numbers, the decimal point).*But, you CAN do math on it.*

## Boolean

Can be equal to either **True** or **False**. Nothing else. Used often in the condition part of IF statements and loops.

# Working with Variables

## Creating a Variable

Before you can use a variable, it must be created in memory – **declared**. Suppose we wanted to declare a variable to hold someone's first name. Depending on the language, we might do something like this:

Dim FirstName As String

That statement declares the variable. Notice that we are giving the variable a *name*(FirstName) and a *type (*String). Variable names can have letters and numbers only. No special characters and no spaces.

That creates the variable. But the variable is initially empty.  *It has no value.*

## Giving a Variable a Value

Here's an example of how we give, or **assign** a value to a variable:

FirstName = "Jenny"

That does NOT say that FirstName is equal to "Jenny". The equal sign, in this case, is actually an **assignment operator**. It tells the computer to take whatever is on the *right side* of the equal sign, and stick it into the variable that's on the *left side* of the equal sign. After that line of code finishes running, the variable FirstName will no longer be empty. It will contain the value "Jenny".

Note that Jenny is in double-quotes. All strings must be in double-quotes. When we assign to *number* variables however, we do NOT use quotes:

Age = 32

The variable, Age, now contains the value 32.

If, right after that line of code, we did this:

Age = 102

whatever was in **Age** before will get *wiped out* and replaced by the value of 102. That's how variables work.

**Another example**

Let's declare 3 variables:

Dim Total As Decimal

Dim Subtotal As Decimal

Dim Tax As Decimal

Let's assign a value to **Subtotal**:

Subtotal = 20.00

Let's assign a value to **Tax**:

Tax = 1.00

Now let's add **Subtotal** to **Tax** and stick it (assign it) into the variable **Total**:

Total = Subtotal + Tax

Once again, the computer first looks on the *right side* of the equal sign. If there's any math to be done, it does it. In this case, it adds up the values of **Subtotal** and **Tax**.Then, it takes the result and assigns it to whatever variable is on the *left side* of the equal sign – in this case, the variable **Total**.**Total** is now equal to 21.00.

## Adding 1 to a Variable

It's often useful to add 1 to a variable. Here's how you do it. Let's suppose we have a variable called **Counter**, and that Counter is equal to zero. Then we do this:

Counter = Counter + 1

**Counter** is now equal to 1. Why? Remember that the computer will look on the right side of the equal sign and do whatever math is indicated. In this case, we're telling it to add 1 to **Counter**. **Counter** starts off being equal to zero. So, 0 + 1 is 1. It then takes that result and sticks it into whatever variable is on the left side of the equal sign – **Counter**, wiping out whatever was previously in **Counter**. Now **Counter** is equal to 1. If we run the line of code a second time:

Counter = Counter + 1

Counter is now equal to 2.

## Subtracting 1 from a Variable

If you know how to add 1 to a variable, subtracting 1 is easy.Again, suppose we have a variable called **Counter**, but this time Counter is equal to 10. Then we do this:

Counter = Counter – 1

Now, Counter is equal to 9.

If we do it again:

Counter = Counter – 1

Counter will be equal to 8.

## Accumulating to a Variable (Keeping a Running Total)

This is really similar to adding 1 to a variable. The difference is that instead of adding 1, you're adding some other value. Suppose we have declared two variables:

Dim Price As Decimal

Dim Total As Decimal

Now let's set **Total** equal to zero:

Total = 0

Now suppose we get a price from the user:

Get Price

And we want to add it to **Total**. But we don't want to wipe out what's already in **Total**. Instead, we want to add to the current value of **Total**. Here's how we do it:

Total = Total + Price

If **Total** is initially equal to 0, and suppose that we get a price of 1.99, then Total will be equal to 1.99. Then we get a second price:

Get Price

Suppose **Price** is equal to 2.99. We add it to **Total** (which is currently equal to 1.99):

Total = Total + Price

1.99 plus 2.99 is equal to 4.98, which then gets assigned back into **Total**, replacing whatever was there before.

# Pseudocode

## Words for Basic Computer Operations

Pseudocode (meaning "false code") is a shorthand that programmers use in their first attempts to come up with a program that will solve the problem. We do this so that we don't get bogged down in the details of the language that we are working in.

There are no hard and fast rules for pseudocode.It's usually for your eyes only. At some point, you will translate your pseudocode into *actual* code. Here are some *general* standards that are often followed:

# Input Examples

* **Get** (from the keyboard, mouse, etc.)

Example: Get FirstName

* **Read** (from a file)

Example: Read StudentScore

# Output Examples

* **Display**(to the screen)

Example: Display FirstName

* **Write** (to a file)

Example: Write FirstName

* **Print** (to a printer)

Example: Print FirstName

# Assignment (assigning a value to a piece of data)

FirstName = “Sarah”

NbrOfStudents = 0

# Math

Total = Subtotal + Tax

GrandTotal = Total + Shipping

### Some rules to follow:

* **Every statement appears on one or more lines *by itself***

**DON'T!**

Get FirstNumber. Get SecondNumber.

**INSTEAD:**

Get FirstNumber

Get SecondNumber

* **Each statement usually begins with a verb (action word)**

**Note:**

This rule does NOT apply to IF statements or loops.

BUT, it DOES apply to each statement *within* them.

* **Following the verb is one or more nouns, which often represent pieces of data**

The noun should be *specific* and *unambiguous*. For example:

**Get DepartmentNumber** is better than **Get Number**

**Write CustomerName** is better than **Write Name**

* **Use mathematical formulas when appropriate:**

Price = Quantity \* Cost

Profit = Price- Cost

* **Keep it simple.**
* **BE CONSISTENT**

**Usually, only you use your pseudocode.**

**But occasionally, it may be used to describe your solution to others.**

# Algorithms

## The "recipe" for solving the problem, i.e. *the solution*

# An algorithm must:

* be clear, precise and unambiguous
* give the correct output **in all cases**
* be written in pseudocode
* declare all necessary variables

# Develop the problem definition into an algorithm using pseudocode:

###### Let's suppose that the purpose of our program is to:

###### Get two numbers from the user

###### Add the two numbers together

###### Display the total

**First, let's try to figure out what variables the program needs, and declare those variables. We'll need a variable to hold the first number, a variable to hold the second number, and a variable to hold the total:**

Dim FirstNumber As Decimal

Dim SecondNumber As Decimal

Dim Total As Decimal

**Notice that I'm also specifying the variables' *type,* e.g. Decimal**

**Now let's write the pseudocode to solve the problem:**

Get FirstNumber

Get SecondNumber

Total = FirstNumber + SecondNumber

Display Total

**What we have above is the algorithm (written in pseudocode) that solves the problem.**

**Now let's make it a little more challenging. Let's display a message to the user if the first number is greater than the second number. We need an IF statement for that. Here's what it might look like:**

If FirstNumber > SecondNumber

Display "The first number is greater than the second number!"

End If

**Notice that the action statement (any statement *inside* the IF statement) is indented one tab stop.**

**Ok, here's the entire algorithm, written (mostly) in pseudocode:**

Dim FirstNumber As Decimal

Dim SecondNumber As Decimal

Dim Total As Decimal

Get FirstNumber

Get SecondNumber

Total = FirstNumber + SecondNumber

Display Total

If FirstNumber > SecondNumber

Display "The first number is greater than the second number!"

End If

# Desk-Checking Algorithms

## What?

To desk-check an algorithm, we use pencil and paper (believe it or not!) to walk thru each line of code, one by one, as if *we are the computer.* We keep a list of all the variables and what they contain.

## Why?

After an algorithm is written, it must be tested for correctness, because most errors occur during the development of the algorithm. If not detected, these errors are passed on to the program where they are much more difficult to detect.

**The sooner you look for bugs, the easier they are to find and fix.**

## How?

* **Choose simple input test cases**

2 or 3 valid cases

1 or 2 invalid cases

* **Establish what the results should be**
* **Make a list of relevant variable names**
* **Walk each test case through the algorithm – on paper**
* **Keep a step-by-step record of the contents of each variable**
* **Make sure algorithm reaches the end**
* **Compare the expected results with the actual results**

# The Rules of Indentation

1. Everything **inside** of an IF statement gets indented one tabstop:

If WorkerIsHourly Then

**Get HoursWorked**

**Get HourlyRate**

**GrossPay = HoursWorked \* HourlyRate**

Else

***'salaried employee***

**Get AnnualSalary**

**GrossPay = AnnualSalary / 52**

EndIf

Display GrossPay

**NOTE: The IF statement itself does NOT get any extra indentation. However, all statements INSIDE of an IF statement get indented.**

1. Everything **inside** of a LOOP gets indented one tabstop:

WhileMoreWorkers

**Get data for next worker**

**If worker is hourly Then**

Get HoursWorked

Get HourlyRate

GrossPay = HoursWorked \* HourlyRate

**Else**

*'salaried employee*

Get AnnualSalary

GrossPay = AnnualSalary / 52

**EndIf**

**Display GrossPay**

EndWhile

**NOTE: The Loop statement itself doesNOT get any extra indentation. However, all statements INSIDE of a Loop get indented.**

J