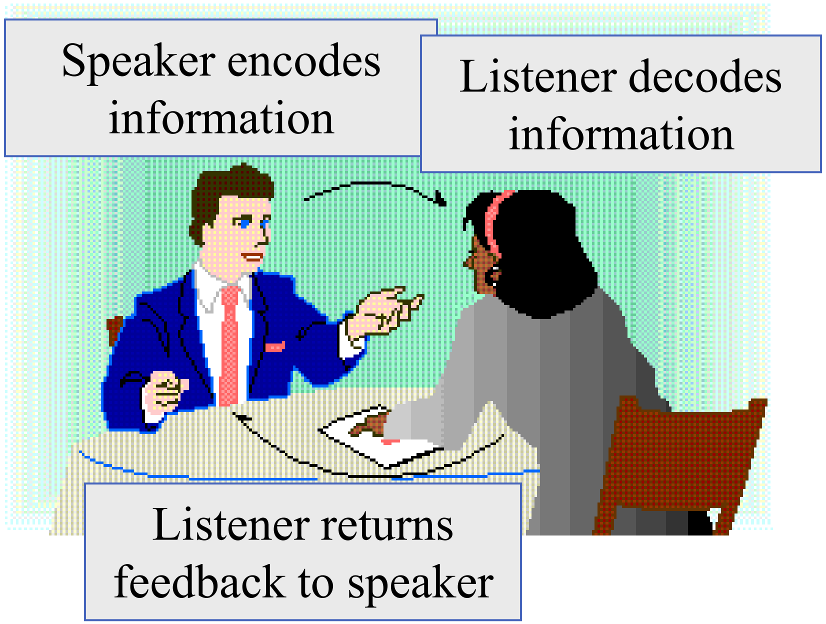
Computer Languages, Algorithms and Program Development

Computer Languages, Algorithms and Program Development

- In this chapter:
  - What makes up a language and how do we use language to communicate with each other and with computers?
  - How did computer programming languages evolve?
  - How do computers understand what we are telling them to do?
  - What are the steps involved in building a program?
  - How can we create something that would be visible on the WWW?

Communicating with a Computer

- Communication cycle
  - One complete unit of communication.
    - An idea to be sent.
    - An encoder.
    - A sender.
    - A medium.
    - A receiver.
    - A decoder.
    - A response.



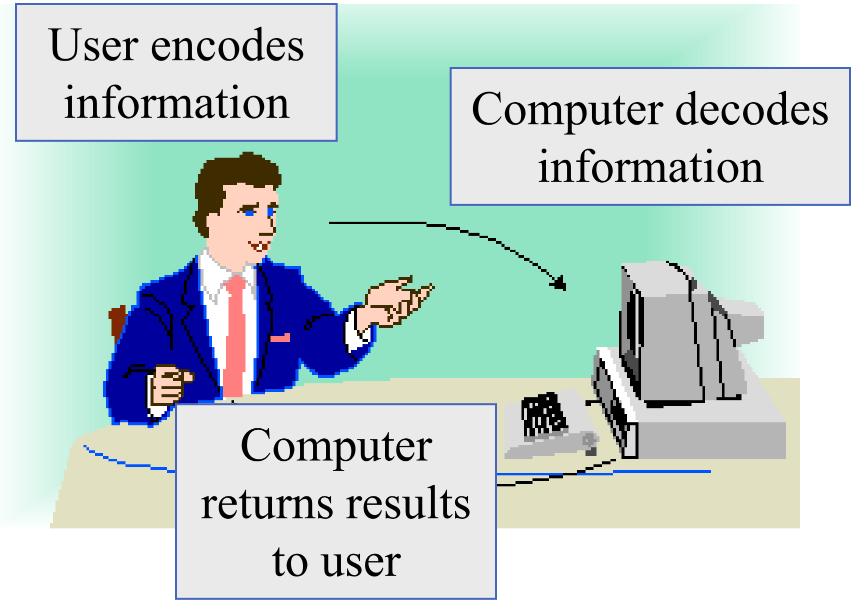
Speaker encodes information

Listener decodes information

Listener returns feedback to speaker

Communicating with a Computer

- Substituting a computer for one of the people in the communication process.
  - Process is basically the same.
    - Response may be symbols on the monitor.



User encodes information

Computer decodes information

Computer returns results to user

Communicating with a Computer

A breakdown can occur any place along the cycle...

- Between two people:
  - The person can’t hear you.
  - The phone connection is broken in mid-call.
  - One person speaks only French, while the other only Japanese.
- Between a person and a computer:
  - The power was suddenly interrupted.
  - An internal wire became disconnected.
  - A keyboard malfunctioned.

When communicating instructions to a computer, areas of difficulty are often part of the encoding and decoding process.

Communicating with a Computer

- Programming languages bridge the gap between human thought processes and computer binary circuitry.
  - **Programming language**: A series of specifically defined commands designed by human programmers to give directions to digital computers.
    - Commands are written as sets of instructions, called **programs**.
    - All programming language instructions must be expressed in binary code before the computer can perform them.

The Role of Languages in Communication

- Three fundamental elements of language that contribute to the success or failure of the communication cycle:
    - Semantics
    - Syntax
    - Participants

The Role of Languages in Communication

- **Semantics**: Refers to meaning.

- Human language:
  - Refers to the meaning of what is being said.
  - Words often pick up multiple meanings.
  - Phrases sometimes have idiomatic meanings:
    - let sleeping dogs lie

      (don’t aggravate the situation by “putting in your two cents”)

- Computer language:
  - Refers to the specific command you wish the computer to perform.
    - Input, Output, Print
    - Each command has a very specific meaning.
    - Computers associate one meaning with one computer command.

The Role of Languages in Communication

- **Syntax**: Refers to form, or structure.

- Human language:
  - Refers to rules governing grammatical structure.
    - Pluralization, tense, agreement of subject and verb, pronunciation, and gender.
  - Humans tolerate the use of language.
    - How many ways can you say no? Do they have the same meaning?

- Computer language:
  - Refers to rules governing exact spelling and punctuation, plus:
    - Formatting, repetition, subdivision of tasks, identification of variables, definition of memory spaces.
  - Computers do not tolerate syntax errors.

The Role of Languages in Communication

- **Participants**:
  - Human languages are used by people to communicate with each other.
  - Programming languages are used by people to communicate with machines.

- Human language:
  - In the communication cycle, humans can respond in more than one way.
    - Body language
    - Facial expressions
    - Laughter
    - human speech

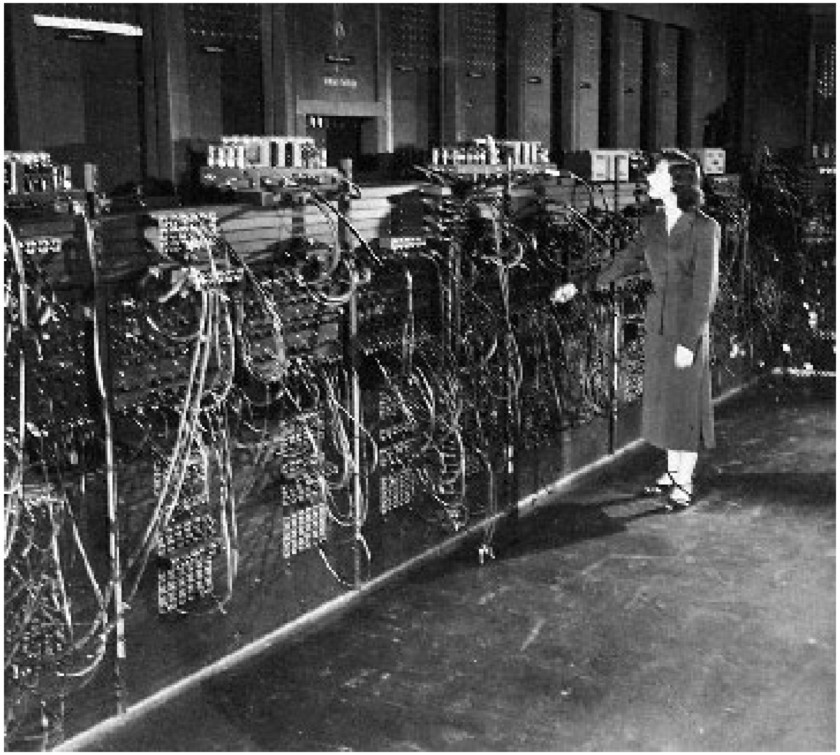
- Computer language:
  - People use programming languages.
  - Programs must be **translated** into binary code.
  - Computers respond by performing the task or not!

The Programming Language Continuum

- In the Beginning...Early computers consisted of special-purpose computing hardware.
  - Each computer was designed to perform a particular arithmetic task or set of tasks.
  - Skilled engineers had to manipulate parts of the computer's hardware directly.
    - Some computers required "fat-fingering".
      - **Fat-fingering**: Engineer needed to position electrical relay switches manually.
    - Others required programs to be hardwired.
      - **Hardwiring**: Using solder to create circuit boards with connections needed to perform a specific task.

The Programming Language Continuum

- ENIAC
  - Used programs to complete a number of different mathematical tasks.
    - Programs were entered by plugging connector cables directly into sockets on a plug-in board.
      - Set-up could take hours.
      - A program would generally be used for weeks at a time.



The Programming Language Continuum

- In the beginning… To use a computer, you needed to know how to program it.
- Today… People no longer need to know how to program in order to use the computer.
- To see how this was accomplished, lets investigate how programming languages evolved.
  - First Generation - Machine Language (code)
  - Second Generation - Assembly Language
  - Third Generation - People-Oriented Programming Languages
  - Fourth Generation - Non-Procedural Languages
  - Fifth Generation - Natural Languages

The Programming Language Continuum

- First Generation - Machine Language (code)
  - **Machine language** programs were made up of instructions written in binary code.
    - This is the “native” language of the computer.
    - Each instruction had two parts: Operation code, Operand
      - **Operation code** (**Opcode**): The command part of a computer instruction.
      - **Operand**: The address of a specific location in the computer’s memory.
    - **Hardware dependent**: Could be performed by only one type of computer with a particular CPU.

The Programming Language Continuum

- Second Generation - Assembly Language
  - **Assembly language** programs are made up of instructions written in mnemonics.

```
READ     num1
READ     num2
LOAD     num1
ADD      num2
STORE    sum
PRINT    sum
STOP
```

  - **Mnemonics**: Uses convenient alphabetic abbreviations to represent operation codes, and abstract symbols to represent operands.
  - Each instruction had two parts: Operation code, Operand
  - Hardware dependent.
  - Because programs are not written in 1s and 0s, the computer must first translate the program before it can be executed.

The Programming Language Continuum

- Third Generation - People-Oriented Programs
  - Instructions in these languages are called statements.
    - **High-level languages**: Use statements that resemble English phrases combined with mathematical terms needed to express the problem or task being programmed.
    - Transportable: NOT-Hardware dependent.
    - Because programs are not written in 1s and 0s, the computer must first translate the program before it can be executed.

The Programming Language Continuum

- Pascal Example: Read in two numbers, add them, and print them out.

```
Program sum2(input,output);
var
  num1,num2,sum : integer;

begin
  read(num1,num2);
  sum:=num1+num2;
  writeln(sum)
end.
```

The Programming Language Continuum

- Fourth Generation - Non-Procedural Languages
  - Programming-like systems aimed at simplifying the programmers task of imparting instructions to a computer.
  - Many are associated with specific application packages.
    - Query Languages:
    - Report Writers:
    - Application Generators:

The Programming Language Continuum

- **Query Languages:**
  - Enables a person to specify exactly what information they require from the database.
  - Usually embedded within database management programs.
- **Report Writers:**
  - Takes information retrieved from databases and formats into attractive, usable output.
- **Application Generators:**
  - A person can specify a problem, and describe the desired results.
  - Included with many micro-computer programs (macros).

The Programming Language Continuum

- Fourth Generation - Non-Procedural Languages (cont.)
  - **Object-Oriented Languages**: A language that expresses a computer problem as a series of objects a system contains, the behaviors of those objects, and how the objects interact with each other.
    - **Object**: Any entity contained within a system.
      - Examples:
        - » A window on your screen.
        - » A list of names you wish to organize.
        - » An entity that is made up of individual parts.
    - Some popular examples: C++, Java, Smalltalk, Eiffel.

The Programming Language Continuum

- Fifth Generation - Natural Languages
  - **Natural-Language**: Languages that use ordinary conversation in one’s own language.
    - Research and experimentation toward this goal is being done.
      - Intelligent compilers are now being developed to translate natural language (spoken) programs into structured machine-coded instructions that can be executed by computers.
      - Effortless, error-free natural language programs are still some distance into the future.

Assembled, Compiled, or Interpreted Languages

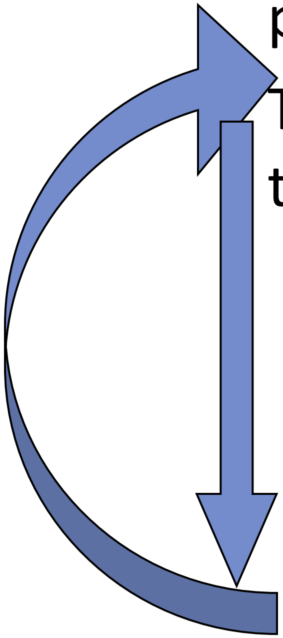
- All programs must be translated before their instructions can be executed.
- Computer languages can be grouped according to which translation process is used to convert the instructions into binary code:
  - Assemblers
  - Interpreters
  - Compilers

Assembled, Compiled, or Interpreted Languages

- **Assembled languages**:
  - **Assembler:** a program used to translate Assembly language programs.
  - Produces one line of binary code per original program statement.
    - The entire program is assembled before the program is sent to the computer for execution.

Assembled, Compiled, or Interpreted Languages

- **Interpreted Languages:**
  - **Interpreter:** A program used to translate high-level programs.



  Translates one line of the program into binary code at a time:

  - An instruction is **fetched** from the original source code.
  - The Interpreter checks the single instruction for errors. (If an error is found, translation and execution ceases. Otherwise…)
  - The instruction is translated into binary code.
  - The binary coded instruction is **executed**.
  - The fetch and execute process repeats for the entire program.

Assembled, Compiled, or Interpreted Languages

- **Compiled languages**:
  - **Compiler:** a program used to translate high-level programs.
  - Translates the entire program into binary code before anything is sent to the CPU for execution.
    - The translation process for a compiled program:
      - First, the Compiler checks the entire program for syntax errors in the original **source code**.
      - Next, it translates all of the instructions into binary code.
        - » Two versions of the same program exist: the original **source code** version, and the binary code version (**object code**).
      - Last, the CPU attempts execution only after the programmer requests that the program be executed.

Programming for Everyone

- Several ways to control what your computer does or the way it accomplishes a particular task:
  - Using Macros
  - Using HTML to create Web Pages
  - Scripting
- Each allows customization of current applications.

Programming for Everyone

- **Using Macros**
  - **Macro**: Set of operations within the computer application that have been recorded for later execution.
    - Once recorded, the macro can be used repeatedly on any document within that application.
    - In word processors, macros are commonly used to speed up repetitive tasks.
      - Example: SIG can be stored as a macro that includes a signature message at the end of a document.

James R. Emmelsohn
Director of Public Relations,
Martin Electronics, Detroit Division

Programming for Everyone

- Using HTML to create Web Pages
  - HTML (HyperText Markup Language): A computer language consisting of special codes intended to design the layout (or markup) of a Web page.
    - Web browsers interpret the HTML code and display the resulting Web pages.
    - Web browser: A program that displays information from the WWW.
    - Each line of HTML is called a tag (formatting instruction).

Programming for Everyone

<HTML>	• Designates an HTML document
<HEAD>	• Beginning of Header section
<TITLE> Title of Web Page </TITLE>	• Contents of Title bar
</HEAD>	• End of Header section
<BODY bgcolor=#ffffff text=#000000 >	• Background=white, text=black
<BODY>	• Top of the body of the document
<H1>	• H1=largest text size, H6 is smallest
<CENTER> Sample Web Page	• CENTER turns on centering
</CENTER> </H1>	• Turns off centering and large text
<HR>	• Displays a horizontal rule: thin line
<A HREF=“http://www.dogpile.com”> dogpile search engine </A>	• Links to the dogpile search engine
</BODY>	• </BODY> and </HTML>designate the bottom of the document
</HTML>	

Programming for Everyone

- **Scripting**
  - **Scripting**: A series of commands, written to accomplish some task.
    - Very similar to the concept of a program.
    - Extends the capabilities of the application where it is being used.
    - Examples of scripting languages:
      - Perl, C++, VBScript, JavaScript
      - **JavaScript**: A scripting language that allows the Web page designer to add functional features to a formatted web page created in HTML.

Building a Program

- Whatever type of problem needs to be solved, a careful thought out plan of attack, called an algorithm, is needed before a computer solution can be determined.

1) Developing the algorithm.
2) Writing the program.
3) Documenting the program.
4) Testing and debugging the program.

Building a Program

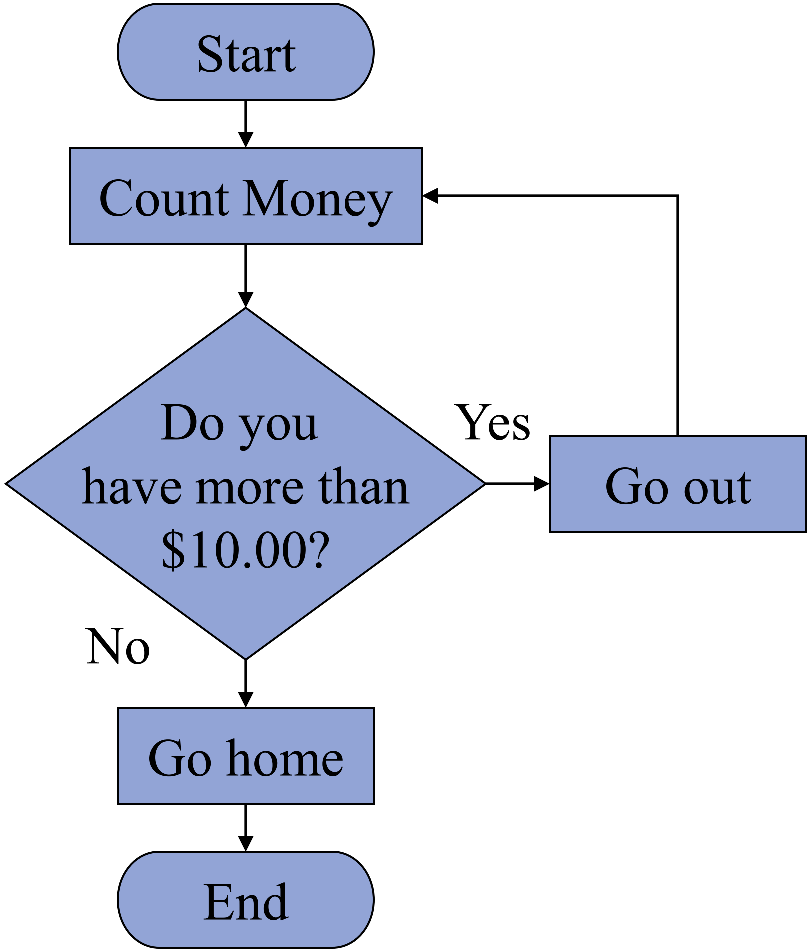
- 1) Developing the algorithm.
  - **Algorithm**: A detailed description of the exact methods used for solving a particular problem.
  - To develop the algorithm, the programmer needs to ask:
    - What data has to be fed into the computer?
    - What information do I want to get out of the computer?
    - **Logic**: Planning the processing of the program. It contains the instructions that cause the input data to be turned into the desired output data.

Building a Program

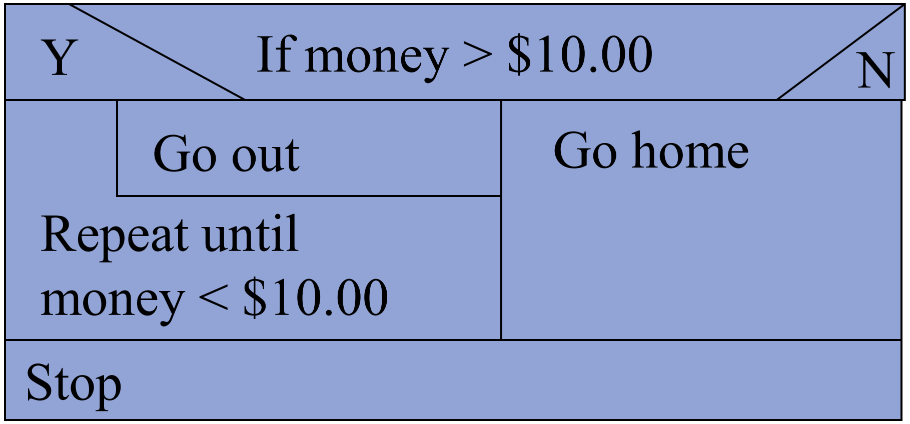
- A step-by-step program plan is created during the planning stage.
- The three major notations for planning detailed algorithms:
  - **Flowchart**: Series of visual symbols representing the logical flow of a program.
  - **Nassi-Schneidermann charts**: Uses specific shapes and symbols to represent different types of program statements.
  - **Pseudocode**: A verbal shorthand method that closely resembles a programming language, but does not have to follow a rigid syntax structure.

Building a Program

Flow chart:



Nassi-Schneidermann chart:



Pseudocode:

```
1.  If money < $10.00 then go home
          Else Go out
2.  Count money
3.  Go to number 1
```

Building a Program

- 2) Writing the Program
  - If analysis and planning have been thoroughly done, translating the plan into a programming language should be a quick and easy task.

- 3) Documenting the Program
  - During both the algorithm development and program writing stages, explanations called documentation are added to the code.
    - Helps users as well as programmers understand the exact processes to be performed.

Building a Program

- 4) Testing and Debugging the Program.
  - The program must be free of **syntax errors**.
  - The program must be free of **logic errors**.
  - The program must be **reliable**. (produces correct results)
  - The program must be **robust**. (able to detect execution errors)

  - **Alpha testing**: Testing within the company.
  - **Beta testing**: Testing under a wider set of conditions using “sophisticated” users from outside the company.

Software Development: A Broader View

Measures of effort spent on real-life programs:
Comparing programs by size:

Type of program	Number of Lines
The compiler for a language with a limited instruction set.	Tens of thousands of lines
A full-featured word processor.	Hundreds of thousands of lines
A microcomputer operating system.	Approximately 2,000,000 lines
A military weapon management program. (controlling missiles, for example)	Several million lines

Software Development: A Broader View

- Measures of effort spent on real-life programs: Comparing programs by time:
  - Commercial software is seldom written by individuals.
    - **Person-months** - equivalent to one person working forty hours a week for four weeks.
    - **Person-years** - equivalent to one person working for twelve months.
    - Team of 5 working 40 hours for 8 weeks = ten person-months.

Web Page Design Software: Dreamweaver

- What is Web page design software?
  - The programs that help create pages and their associated HTML.
  - Dreamweaver: A visual Web page editor primarily for use by Web design professionals.

- Why is it needed?
  - Allows creation of Web pages without knowledge of HTML .

Web Page Design Software: Dreamweaver

- What minimal functions must it have?
  - WYSIWIG: “What you see is what you get.”
    - Web page designers see exactly what it will look like.
  - Allows selection of color scheme. (Background and text)
  - Allows text manipulation. (Typing text where you want it, changing the size, color or style)
  - Allows importation and layout of images.

Web Page Design Software: Dreamweaver

- What types of support are available to enhance its use?
  - Applets extend the capabilities of HTML.
    - **Applet**: A short application program, usually written in Java, which adds enhancement and/or functionality to a Web page.
- Is special support hardware available?
  - Creating audio/visual materials for the WWW:
    - Photo digitizers or scanners, video digitizer, and audio digitizer.
    - Once these are in a standard digital format, they can be imported to Web development programs.

Web Page Design Software: Dreamweaver

- One final note:
  - Dreamweaver and other Web page design software create Web pages. You still need a place to keep your Web page.
    - **ISP (Internet Service Provider):** A company or organization that is used as an access point to the WWW.
      - The ISP will put your Web page on its server.
      - You will be given an address where you or others can access your Web page.