Introduction to Programming
Visual Basic .NET

CS A109

Introduction and Brief History of Programming

• Hardware
  – Physical components that make up a computer
• Computer program or software
  – A self-contained set of instructions used to operate a computer to produce a specific result
Knowing About: Computer Hardware

• Computer hardware components
  – Memory unit
    • Stores information in a logically consistent format
      – Each memory location has an address and data that can be
        stored there, imagine a long line of mailboxes starting at
        address 0 and going up to addresses in the billions
    • Two types of memory: RAM and ROM
      – Random Access Memory, Read Only Memory (misnamed)
  – Control unit
    • Directs and monitors the overall operation of the
      computer

Knowing About: Computer Hardware (Continued)

– Arithmetic and logic unit (ALU)
  • Performs all the arithmetic and logic functions
    – E.g. knows how to add, subtract, manipulate data
– Input/output (I/O) unit
  • Provides access to and from the computer
  • E.g. network, keyboard, video
– Secondary storage
  • Provides a permanent storage area for programs and
    data
Evolution of hardware
- 1950s: all hardware units were built using relays and vacuum tubes
- 1960s: introduction of transistors
- mid-1960s: introduction of integrated circuits (ICs)
- Present computers: use of microprocessors

Bit: smallest and most basic data item in a computer; represents a 0 or a 1

Byte: a grouping of eight bits
- E.g., 00010001
- What does this represent?

Word: a grouping of one or more bytes
First-Generation and Second-Generation (Low-Level) Languages

• Low-level languages
  – First-generation and second-generation languages
  – Machine-dependent languages
  – The underlying representation the machine actually understands

• First-generation languages
  – Also referred to as machine languages
  – Consist of a sequence of instructions represented as binary numbers
  – E.g.: Code to ADD might be 1001. To add 1+0 and then 1+1 our program might look like this:
    • 1001 0001 0000
    • 1001 0001 0001

First-Generation and Second-Generation (Low-Level) Languages (Continued)

• Second-generation languages
  – Also referred to as assembly languages
  – Abbreviated words are used to indicate operations
  – Allow the use of decimal numbers and labels to indicate the location of the data

• Assemblers
  – Programs that translate assembly language programs into machine language programs
  – Our add program now looks like:

    • ADD 1.0 1001
    • ADD 1,1 0001
    1001 0000
    1001 0001
Third-Generation and Fourth-Generation (High-Level) Languages

- High-level languages
  - Third-generation and fourth-generation languages
  - Programs can be translated to run on a variety of computer types
- Third-generation languages
  - Procedure-oriented languages
  - Object-oriented languages
- Our Add program might now look like:

```
sum = value1 + value2
```

```
1001
0001
0000
1001
0001
0001
```

The Evolution of Programming Languages (Continued)
• Procedure-oriented languages
  – Programmers concentrate on the procedures used in the program
  – Procedure: a logically consistent set of instructions which is used to produce one specific result

• Object-oriented languages
  – Items are represented using self-contained objects
  – Often used for graphical windows environments, ability to re-use code efficiently

Third-Generation and Fourth-Generation (High-Level) Languages (Continued)

• Graphical user interface (GUI)
  – Provides a graphical way for the user to interact with the program
  – Uses events

• Event
  – A specific procedure that is connected to an object

• Visual languages
  – Permit the programmer to manipulate graphical objects directly, with the language providing the necessary code
  – Permit users to access and format information without the need for writing any procedural code
The Visual Basic .NET Platform

- Visual Basic .NET is in a sense one step removed from a typical high-level language
- VB.NET runs using a “Virtual Machine” or “Common Language Runtime”
  - The physical computer simulates a virtual computer that runs your program
- What is .NET?
  - Microsoft’s vision of the future of applications in the Internet age
    - Increased robustness over classic Windows apps
    - New programming platform
    - Built for the web
  - .NET is a platform that runs on the operating system

.NET

- .NET is actually a program that sits on top on the Operating System (currently all the Windows 9x, NT, ME & XP; subset exists for FreeBSD, Linux, MacOS)
- Provides language interoperability across platforms
- Strong emphasis on Web connectivity, using XML web services to connect and share data between smart client devices, servers, and developers/users
- Platform/language independent
.NET Framework

C#  VB.NET  C++.NET  Other

Common Language Specification

Framework Class Library

ASP.NET
- Web Services
- Web Forms
- ASP.NET Application Services

Windows Forms
- Controls
- Drawing
- Windows Application Services

ADO.NET
- XML
- Threading
- IO

Network
- Security
- Diagnostics
- Etc.

Common Language Runtime

Memory Management
- Common Type System
- Lifecycle Monitoring

Operating System

Visual Studio .NET

.NET: Language-Independent, Mostly Platform Specific

Person.vb  (Visual Basic)

Person

Address

Company

Address.cs  (C#)

Company.cbl  (Cobol)

CLR

Deploy

Windows

CLR

Others?

CLR
What Makes a Quality Program?

• A program that functions properly does not necessarily mean the grade will be an “A”
• A quality program should have the following characteristics:
  – Meet the specifications dictated to the programmer
  – Readability
  – Modularity
  – Efficiency
  – Robustness
  – Usability

Readability

• Code that is readable and understandable makes it much easier to maintain in the future
  – Yourself, other programmers
• Typical method: Add comments to the program
  – Explain program purpose and code that was attempted
• External documentation
  – Project specifications
  – Software requirements
  – Software design documents
Modularity

- Program written in an orderly fashion
  - Problems divided into smaller sub-problems
  - Each sub-problem solved in isolation and assembled together, capable of standing on its own
  - Testing performed on individual modules and modules as a whole
- Allows incremental testing and development
  - Avoid the “Big Bang” approach

Efficiency

- Should the goal of the programmer be to write compact, super-efficient code or to write clear, readable code that may run a little more slowly and take up more room?
  - It depends on the situation, but generally the readable/slow/larger code for VB.NET programs
  - Other languages used when high performance is required
- Lots of typed code does not necessarily mean a larger compiled program
- Overall it is better to strive for efficient code that is readable and understandable to make maintenance easier
Robustness

- If a program is written for a task, how does it handle:
  - Larger sized input?
  - Unexpected input?
  - Error condition?
- Ideally the program should never crash, but exit gracefully or design the GUI to not allow invalid cases.
  - Example: Enter your birthday (MM/DD/YYYY):

Usability

- A program must be correct to be useful, but it is possible to have a correct but unusable program
- If the user can’t easily use the program, the program will probably not be used
  - E.g. London Ambulance System
Algorithms

• Before jumping into programming, practice by understanding the concept of an algorithm
• Algorithm
  – A step-by-step sequence of instructions that must terminate
  – Example: Steps in a recipe to make a cheesecake.
    • Describe an algorithm to determine who has the highest grade on an exam
    • Describe an algorithm to find your optimal date
• Describing algorithms
  – Pseudocode
    • Use of English-like phrases to describe an algorithm
  – Formula
    • Use of mathematical equations to describe an algorithm
  – Flowchart
    • Use of diagrams that employ symbols to describe an algorithm

Algorithms (Continued)

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Name</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>Terminal</td>
<td>Indicates the beginning or end of a program</td>
</tr>
<tr>
<td>☐</td>
<td>Input/Output</td>
<td>Indicates an input or output operation</td>
</tr>
<tr>
<td>☐</td>
<td>Process</td>
<td>Indicates computation or data manipulation</td>
</tr>
<tr>
<td>▷◁</td>
<td>Flow lines</td>
<td>Used to connect the flowchart symbols and indicate the logic flow</td>
</tr>
<tr>
<td>☐</td>
<td>Decision</td>
<td>Indicates a program branch point</td>
</tr>
<tr>
<td>☐</td>
<td>Loop</td>
<td>Indicates the initial, limit, and increment values of a loop</td>
</tr>
<tr>
<td>☐</td>
<td>Predefined</td>
<td>Indicates a predefined process, as in calling a program unit</td>
</tr>
<tr>
<td>○</td>
<td>Connector</td>
<td>Indicates an entry to, or exit from, another part of the flowchart</td>
</tr>
</tbody>
</table>
Class Average Algorithm

- **Problem:** Calculate and report the grade-point average for a class
- **Discussion:** The average grade equals the sum of all grades divided by the number of students

**Output:** Average grade

**Input:** Student grades

**Processing:** Find the sum of the grades; count the number of students; calculate average

---

![Flowchart](image-url)
Pseudocode

Program:

Determine the average grade of a class
Initialize Counter and Sum to 0
While there is more data
    Get the next Grade
    Add the Grade to the Sum
    Increment the Counter
Loop
Computer Average = Sum/Counter
Display Average

Hierarchy Chart
Algorithm Selection

- Our choice of algorithm can have a large impact on the performance of the program
- Consider the problem of searching for a name in a phone book
- The phone book is sorted by name

Algorithm 1 – Linear Search

- Search through the list of names for our target starting at the beginning and go through them one at a time until we reach the end

Apple, Bob
Atto, Tom
Attrick, Jerry
DeBanque, Robin
Fresco, Al
Guini, Lynn
Oki, Kerry
Wright, Eaton

Must check up to 8 names
What if millions of names?
Algorithm 2 – Binary Search

- Takes advantage of the fact that the list of names is sorted
- Start at the name in the middle of the list and compare to the target
  - If equal, there is a match!
  - If the target is alphabetically less, repeat the process on the first half of the list
  - If the target is alphabetically greater, repeat the process on the second half of the list
  - Stop and no match if the list is ever empty

Searching for Guini, Lynn

Apple, Bob
Atto, Tom
Attrick, Jerry
DeBanque, Robin
Fresco, Al
Guini, Lynn
Oki, Kerry
Wright, Eaton

Only checked 3 names!
What if millions of names?
Algorithm Analysis

• Algorithm 1 runs in time linear to the number of names;
  – 1 million names requires searching on average 500,000 names, all 1 million in the worst case
• Algorithm 2 runs in time $\log_2 n$ to the number of names
  – This is because we cut the size in half each time
  – For 1 million names, $\log_2 1000000$ is about 20.
  – Much better performance than algorithm 1!
  
  – This is a case where you would want to consider the efficiency if used with a large number of names. If the program were to always run with a small number of names, algorithm 1 is simpler and will run fast enough on a typical computer