**Programming Logic – Beginning**

152-101

Designing Programs and Applications

<table>
<thead>
<tr>
<th>Notes</th>
<th>Activity</th>
</tr>
</thead>
</table>

**Quick Links & Text References**

- **Program Design** Pages 23 – 27
- **Algorithms** Pages 25 – 26
- **Levels of Design** Pages
- **IOPs** Pages
- **User Interface Design** Pages 18 – 20
  24 – 25
- **TOE charts** Pages
- **Flowcharts** Pages 26
  198, 201-202, 206
- **Pseudocode** Page 25 – 26
- **Testing Applications** Page 27

**Program Design**

- Program Design: Why Bother?
- Course requirement!
- Programming is part art, part engineering. Design is the engineering
  - Bridge builder example: don’t build bridge without design
  - Home builder example: don’t build a home without a blue print.
  - 40/20/40 Programming Rule:
    - 40% of your time should be spent planning/designing the program and application
    - 20% of your time should be spent coding
    - 40% of your time should be spent testing
      - This portion can vary greatly
    - If you are spending more time coding/testing – you are not spending enough time up front in the design phase!
- Step by step process.
  - Effort at each step simplifies the effort at the next, ultimately resulting in simpler programming effort
Notes

- Simplifies program development.
  - Large programs broken into smaller parts, each developed separately.
  - Obviously, small easier than large.
- More correct programs.
  - Testing of smaller sub-programs is easier and more complete.
- Improved maintenance of program.
  - Structured Program Design simplifies maintenance because smaller programs narrow search process for errors or where enhancements belong
- Maintenance also improved because design becomes part of program’s external documentation.
  - Good documentation helps maintainers find errors and add enhancements.
- Better productivity and efficiency.
  - The time spent designing and planning will reduce the time spent at the computer creating the solution.
  - This saves your company time as well.

Activities

**Algorithms**

- Computers are designed to follow instructions.
- The programs you will write are a set of instructions (using the C# programming language) that allows the computer to solve a problem or perform a task for the user.
- *Algorithm* – a set of well-defined steps for performing a task or solving a problem. Algorithms are written in English.
- Algorithms are converted into programming languages (Visual Basic, C#, Java, COBOL, RPG for example).
- The programming languages are converted into machine level language, which is used by the computer to perform the task or solve the specific problem.
Notes

### Levels of Design

- Levels of program design for object-oriented applications
  1. [Define what the application is to do](#)
  2. [Design User Interface (visual part the user sees)](#)
  3. [Create a TOE chart](#) to diagram the relationships between tasks, objects and events
  4. [Flowchart & pseudocode each event](#)
  5. Test flowchart and pseudocode

<table>
<thead>
<tr>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Explain Bowling Application</td>
</tr>
</tbody>
</table>

#### 1: Define what the application is to do:

- Develop IOP chart by identifying the purpose of the application, the input required, output needed and how to process the input into output.

**IOP**

- List the program’s inputs
  - Data that comes from a source outside the program such as user entry, a database/file.
  - What do you need from the user or the database?
- List the program’s outputs
  - End results of the program
  - Information displayed to the user or stored on the database/file.
  - What do you need to show the user or store in a database/file.
- List what processes are required to convert the inputs into the outputs
  - WHAT must be done, not HOW.
  - General steps to accomplish the program purpose
- Sample IOP Chart:
Notes

<table>
<thead>
<tr>
<th><strong>Inputs</strong></th>
<th><strong>Outputs</strong></th>
<th><strong>Processing</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>name</td>
<td>Get name</td>
</tr>
<tr>
<td>id</td>
<td>id</td>
<td>Get id</td>
</tr>
<tr>
<td>jobCode</td>
<td>withholdings</td>
<td>Get jobCode</td>
</tr>
<tr>
<td>salary</td>
<td>payment</td>
<td>Get salary</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Calculate withholdings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Calculate payment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display name</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display id</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display withholdings</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Display payment</td>
</tr>
</tbody>
</table>

**Rules for IOPs:**

- Use the table format above
  - Use our Excel template to simplify
- Inputs and Outputs are nouns, simply a list of the program’s inputs and NET outputs.
- Process column contains processes required to convert inputs into outputs. Each step begins with a verb (process)

- Steps to complete IOP:
  - Start with the easy stuff. Inputs, then outputs.
    - These are usually clearly specified in the program requirements.
    - If not, get clarification, don’t assume.
  - Fill in the processes for converting inputs to outputs.
    - Get all inputs
    - Calculate appropriate outputs
      - Most outputs need to calculated
      - Some outputs are simply *echoed*. No processing required
    - Display all outputs.

**Individual exercise:**

Create an IOP for a program to generate a pay sheet. User enters the employee’s name, total hours worked, hourly pay, OT start, OT rate. The program displays the employee’s name, regular hours worked, OT hours worked, regular pay, OT pay and Gross pay.

OT = Overtime
OT Start=Hours after which overtime starts (usually 40)
OT Rate=multiplier for overtime pay (usually 1.5, i.e. time and a half)
**Notes**

<table>
<thead>
<tr>
<th>Activity</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create screen design for Meters to Yards</td>
<td>Create screen design for Years to Days</td>
</tr>
<tr>
<td>Create Bowling screen design</td>
<td>Individual exercise:</td>
</tr>
<tr>
<td>Design the screen layout for Paysheet.</td>
<td></td>
</tr>
</tbody>
</table>

**Design User Interface (Level 2)**

- Plan the design of the user interface – the windows or web pages the user will interact with to complete the tasks.
- Design the user interface so that it is simple, self-explanatory and without distracting features.
- Some basic user interface guidelines are:
  - **Color**
    - Use dark text on light background
    - Avoid bright colors
  - **Text**
    - Use default fonts
    - Use standard sizes
  - **Flow of controls**
    - left to right (buttons on right) **OR**
    - top to bottom (buttons on bottom)
  - **Define logical tab order**
  - **Assign tool tips**
  - **Provide keyboard access**
    - Use keyboard access keys
    - Assign a default button
    - Assign a cancel button
  - **Group common controls**
  - **Set the form location**
  - **Sentence caps (labels)/Book Title caps (buttons)**
  - **Provide a Splash/Welcome Screen**
Common Windows Controls
- Visual Studio provides a wide variety of Windows controls for gathering input, displaying information, selecting values, showing graphics, using the application and more. Below is a list of common controls.
  - Form
  - Text Box
  - Check Box
  - Combo Box
  - Group Box
  - Date Pickers
  - Button
  - Label
  - Radio Button
  - List Box
  - Picture Box

<table>
<thead>
<tr>
<th>Notes</th>
<th>Activity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run demo of various controls</td>
<td>Create TOE chart for Meters to Yards</td>
</tr>
<tr>
<td></td>
<td>Create TOE chart for Years to Days</td>
</tr>
<tr>
<td></td>
<td>Create TOE chart for Bowling</td>
</tr>
<tr>
<td></td>
<td>Create an TOE chart for Pay Sheet</td>
</tr>
</tbody>
</table>

**Develop the TOE Chart (Level 3)**

- Develop a TOE (Task – Object – Event) chart to list the application tasks, objects and events.
  - Tasks:
    - Separate/Group the tasks into parts: Input, Output and Process tasks.
    - Information or commands the user will enter (based on input processing in IOP)
    - Results/output the application will display or store (based on output processing in IOP)
    - Processing needed to take the input and develop it into results/output (based on processing logic in IOP)
  - Objects:
    - Assign each task an object in the interface (text boxes, labels, buttons, etc.) and give the object a unique name.
    - Use 3-letter prefixes to designate the type of object (see examples below)
  - Event:
    - Events are assigned to the objects which will perform a task to process the information.
    - TextBoxes and Labels generally do not have events associated with them.
    - Buttons and menus/toolbar buttons perform tasks when they are clicked.
<table>
<thead>
<tr>
<th>Tasks</th>
<th>Objects</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Get name</td>
<td>txtName</td>
<td></td>
</tr>
<tr>
<td>Get city</td>
<td>txtCity</td>
<td></td>
</tr>
<tr>
<td>Get state</td>
<td>txtState</td>
<td></td>
</tr>
<tr>
<td>Get zip</td>
<td>txtZip</td>
<td></td>
</tr>
<tr>
<td>Get phone</td>
<td>txtPhone</td>
<td></td>
</tr>
<tr>
<td>Get credits</td>
<td>txtCredits</td>
<td></td>
</tr>
<tr>
<td><strong>Processing:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Calculate tuition</td>
<td>btnCalculate</td>
<td>btnCalculate Click</td>
</tr>
<tr>
<td>Clear screen</td>
<td>btnClear, list all input &amp; output objects</td>
<td>btnClear Click</td>
</tr>
<tr>
<td>End application</td>
<td>btnExit</td>
<td>btnExit Click</td>
</tr>
<tr>
<td><strong>Outputs:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Display tuitionCost</td>
<td>lblTuitionCost, btnCalculate</td>
<td>btnCalculate Click</td>
</tr>
</tbody>
</table>

**Flowcharts & Pseudocode** (Level 4)

- This step fills in the last details of the program. This step shows how the actual tasks are accomplished – the algorithm to solve the task. This step consists of multiple diagrams, one for each part of the program.
- Any program can be written using a combination of three processing control structures: sequential processing, decision processing, and repetitive processing.
- To begin building your program design abilities, we’ll start by learning how to design and code sequential processing programs. After you’ve mastered/experienced this, we’ll move on to the other two (later in the course)
- This step of design can be done using a multitude of tools/techniques; you’ll learn about two of them: flowcharts and pseudocode.
Flowcharts

- Flowcharts are pictorial representations of the steps each part of your program will have to go through to accomplish its tasks. Five symbols are used in flowcharting to represent sequential processing steps. We’ll spend more time on the symbols for decisions and repetition later.

**TERMINAL**

Terminals are used to designate the beginning and end of a flowchart (one event module). At the beginning of the flowchart, the terminal contains the event module name and the type of the event. At the end of a flowchart the terminal contains the word End.

**PROCESS**

Process boxes hold mathematical equations or assignments. Another way to look at it—process boxes change the contents of a memory location (variable). In programming, equations are written result first, then the formula (A=B+C).

**INPUT/OUTPUT**

In object-oriented programs, input is typically done using form objects and therefore don’t appear in flowcharts. I/O boxes are used to designate operations that cause the screen (form display) to change. One I/O box is generally used to represent one output command.

Process vs. IO

- Beginners often have trouble determining whether to use a Process box or an Input/Output box.

*If the appearance of the screen changes, use an Input/Output box. Otherwise, use a process box.*
Connectors are used to connect flowchart parts that are large distances apart on the same page (e.g. bottom to top of next column). **Off-page connectors** can be used to connect one page of a flowchart to another. Generally, flowcharts that don’t fit on a page represent modules that are too complex. Put a unique label in the connector and its connecting **twin** so the connector can be easily distinguished from other connectors.

**Decision or Repetition**

This symbol is used when making decisions (Did the person work more than 40 hours?) and also for repetition (Are there more employees to process?). There is one entry and one exit from this symbol. Used for IF, Do…While, Repeat… Until, FOR. This topic will be discussed later in the course.

**Flowchart Rules**

- One output box (I/O) per output command.
- Lines connecting symbols should have arrows pointing to the following symbol
- If abbreviated variable names are used, I recommend a **legend** or glossary box near the bottom of the flowchart.

**Introduction to Visio.**

Create flowcharts for Meters to Yards and Years to Days

Create flowchart for Bowling

Individual exercise:

Flowchart Pay Sheet program
Pseudocode

- Pseudocode is structured, English-like statements that represent the sequence of steps a program follows to complete its task. Students tend to like pseudocode because it is much easier to generate than the pictorial design techniques; pseudocode can be quickly generated using any word processor. However, as programmers gain experience, their pseudocode begins to look more and more like actual code, making it less useful as documentation for non-programmers.

- Supervisors, therefore, prefer the pictorial design techniques. New automated tools have made generating and changing flowcharts easier.

- Where pseudocode is used, it is often strictly controlled, almost making it a language of its own.

- Pseudocode usually starts out with the event module’s name and ends with the keyword End (or Return). Between these lines are English-like descriptions of the steps the program must accomplish. *Keywords* are used for certain types of processing.
### Keywords for Sequential Pseudocode Processing
- **Outputting Information:**
  - `DISPLAY` (output to screen)
  - `PRINT` (output to printer)
- **Arithmetic:**
  - Mathematical formulas may be preceded by the keyword `CALCULATE`, but this is not required. Formulas are usually self-evident without the keyword.
  - Simply list the formula (result = equation) in your pseudocode.
  - Initializing or assigning a value may be preceded by the keyword `INITIALIZE`, but this is not required.
- **Receiving Information:**
  - `READ` (input from a file)
- **Decisions**
  - `IF, THEN, ELSE, ELSEIF, ENDIF, and CASE` (To be discussed fully later)
- **Repetition**
  - `REPEAT…UNTIL, DO…WHILE, FOR` (To be discussed fully later)

### Rules for Pseudocode
- As mentioned above, most organizations have strict rules for pseudocode. We only have a couple:
  - You may only use the keywords you have been taught.
  - Keywords must standout
    - Use `UPPERCASE`
    - or use `bold`
    - or both

### Activity
- Create pseudocode for Meters to Yards and Years to Days
- Create pseudocode for bowling
- Individual exercise:
- Complete Checkpoint pg 27
Testing Algorithms (Step 5)

- Use either your IOP or TOE as a guide
- For each input, select an appropriate value
- Determine the expected value of all program outputs
- Use a calculator where necessary
  - If necessary, determine value of working variables as well
  - Create multiple sets of test data to thoroughly test the application.
- Create at least one set of test data that has invalid inputs and designate what results you expect (might be "Error").

<table>
<thead>
<tr>
<th>Test Plan for: program name</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong></td>
</tr>
<tr>
<td>Test 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Anticipated Equations</strong></td>
</tr>
<tr>
<td>Processing</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td><strong>Anticipated Results</strong></td>
</tr>
<tr>
<td>Outputs</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>
Test Plan for: *Sales Amount*

<table>
<thead>
<tr>
<th></th>
<th>Assigned or Expected Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inputs:</strong></td>
<td>Test 1</td>
</tr>
<tr>
<td>salesTotal</td>
<td>25.00</td>
</tr>
<tr>
<td>taxRate</td>
<td>5.5%</td>
</tr>
<tr>
<td><strong>Processing</strong></td>
<td></td>
</tr>
<tr>
<td>tax = salesTotal * taxRate</td>
<td>tax = 25.00 *.055</td>
</tr>
<tr>
<td>invoiceTotal = salesTotal + tax</td>
<td>invoiceTotal = 25 + 1.38</td>
</tr>
<tr>
<td><strong>Outputs</strong></td>
<td></td>
</tr>
<tr>
<td>invoiceTotal</td>
<td>26.38</td>
</tr>
</tbody>
</table>

There may seem to be a lot of redundant information here, but when you combine this Test Plan tool with a debugger, they blend together very well. While using a debugger, you can touch variable names (like salesTotal and tax in the example above) in a program that is running to see what values they contain. You can compare these values with the values in your test plan to help uncover errors.