Introduction to Programming Design for Controllers

An Online Continuing Education Course for Engineers

Course Number: IC-1004

Credit: 1 Hour / 1 PDH / 1 CPD

Introduction to Programming

The modern concept of the computer originated in the mid 1940's. In 1945, John Von Neumann proposed a computer architecture that included storing instructions and data in the computer. This is the computer architecture of most computers today, and is referred to as the "Von Neumann" computer architecture. The programming of early computers involved preparing a sequence of machine coded instructions. In 1954, John Backus invented the first popular high level programming language, Fortran (Formula Translator). Fortran is oriented to scientific programming and allowed one to write programs that could run on different computers. Fortran facilitated writing programs such as Finite Element Analysis programs that aided the engineering analysis of structures and mechanical components.

The number of computing devices has grown from several research computers to over a billion computers today. In addition to the computer and programmable logic controllers (PLC), embedded computers are very popular, especially in hand-held devices. The demand for programming has been driven by the need to program the many computing devices currently available. An international standard, IEC 61131-3 was published in 1993 for standardization of programmable controller programming. The purpose of the standard was to make programming more consistent among PLC systems.

Program Development

A **program** is defined as a set of coded instructions for insertion into a machine, which then performs the desired sequence of operations. Creating a program can be divided into seven steps:

1. Define the problem. Specify inputs, outputs and the processing task.
2. Analyze the requirements. Divide the processing task into subtasks. Create a hierarchical chart
3. Logic design of the program. Prepare flowcharts, pseudocode, and state diagrams as required.
4. Code the program. Convert the design into a formal programming language. Write source code.
5. Compile and run. Convert the source code to a machine executable code. Run the program.
6. Test the program. Run test scenarios to verify outputs are correct. Correct programming errors.
7. Document and maintain. Prepare user manuals and maintain the program.

Defining the Inputs and Outputs

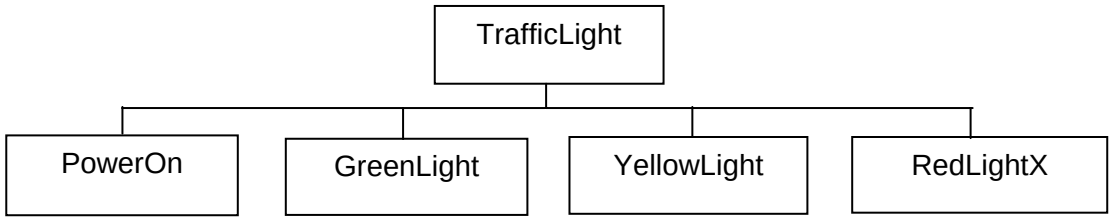
Specifying the program's inputs and outputs is an essential part of defining the programming problem. All inputs have to be converted into a data type before processing. The basic unit of data is the **bit**, a binary digit. The bit has a value of 0 or 1. This can represent the state of a switch, relay, or sensor. A bit can also represent an output control signal that can turn on lights, motors, valves and pumps. It is often referred to as a **Boolean (BOOL)** data type. This is a common data type for inputs and

outputs of programmable controllers. It is also common to group 16 bits in a word. A bit input can be identified as I:3/0 or PowerOn. Examples of outputs are O:4/2 and RedLight. Using locations to reference data is referred to as **addressing**. A symbolic address is a way to reference input and output data using descriptive words. For example, "PowerOn" is a symbolic address for a Power On switch input. Giving input and output addresses descriptive names makes the program more understandable.

There are a number of other useful data types. A sixteen bit signed number is often referred to as an **integer (INT)**. An integer can represent positive and negative numbers of up to 32,767. This data type is useful for counting. Decimal representation of numbers is useful to represent the output of sensors such as weight, pressure and temperature. Decimal numbers require 32 bits and are referred to as a **float or REAL** data type. Eight bits is called a byte and is sufficient to code a character. The ASCII code refers to a binary coding scheme for coding characters. Textual data is usually longer than one character and requires more than one byte. The string of characters is a **text or STRING** data type. This data type is useful for messages. There are also DATE and TIME data types. Strong data typing can prevent mismatching data which can have unpredictable results. Data typing is common to most high level programming languages.

Defining the Processing Tasks

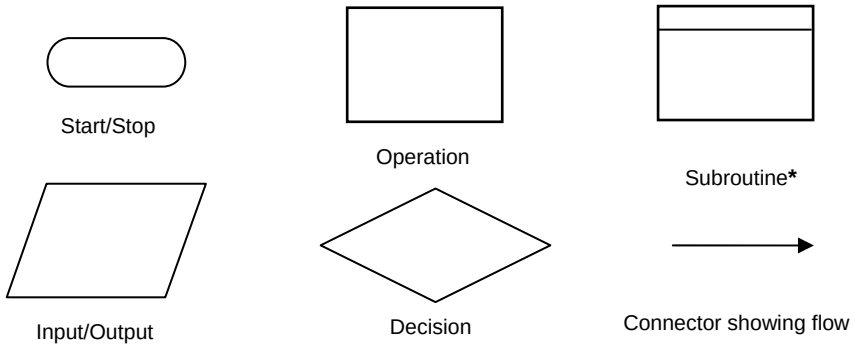
IEC 61131 International Standard provides methodologies for organizing programmable controller programs. Programmable controllers are used to automate a variety of systems: HVAC, alarm, traffic control, process control, water distribution, assembly lines, and robots. The programming components are organized in a hierarchical manner. Programs are built from **function blocks**. Function blocks are sequences of code or graphical components to accomplish a specific task. Most programming languages allow a program to be divided into functional parts; other names for these are subroutine, function, procedure and method. The following hierarchical chart shows the organization of a traffic light program showing the individual function blocks. PowerOn controls startup, and the others control the lights.



Logic Design

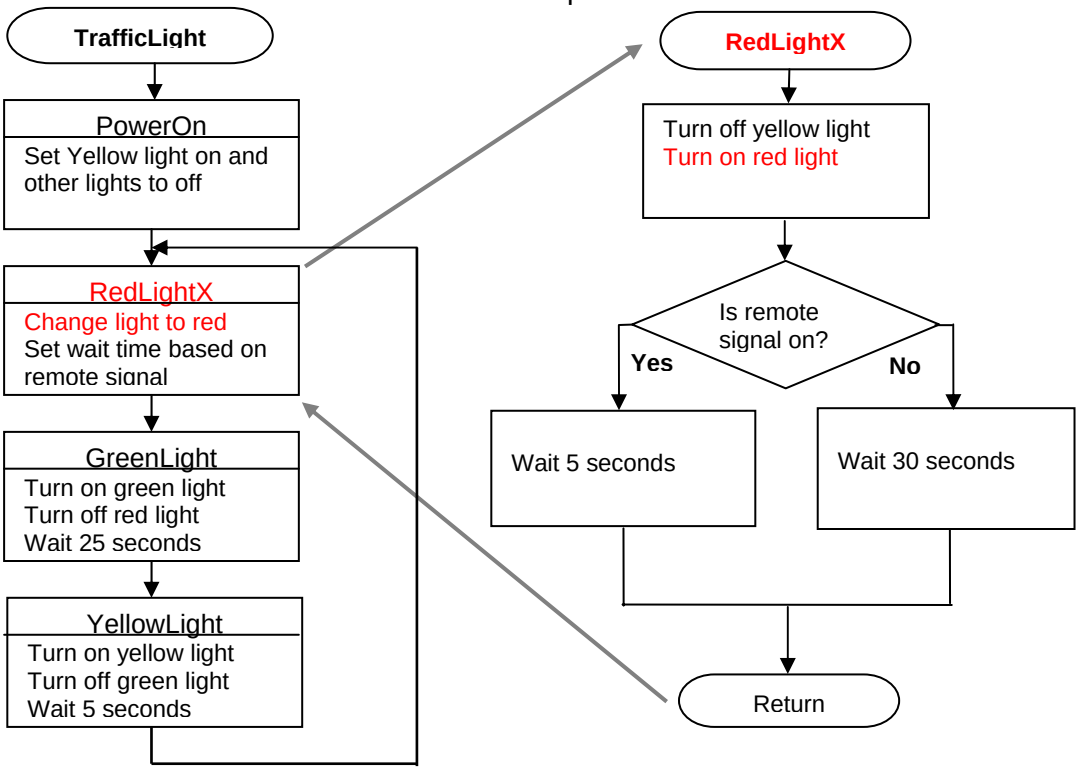
A flow chart is a common design tool to graphically organize the sequence of instructions. Flow charts have been used for designing programs since the development of the modern computer. The following shows the common flowchart symbols. Start/stop symbols indicate the beginning and ending of an instruction sequence. Most programs require inputs and outputs. These are shown using a parallelogram. An operation symbol commonly indicates actions, value assignments, and calculations.

The decision symbol indicates a test or condition that will result in two possible paths: one for when the condition is true, "yes"; and one for when the condition is false, "no".



Basic Flowchart Symbols

Using flow charts, programs can be divided into subroutines. Subroutines break a large program into smaller sections. In the following Traffic Light program, it is desired to sequence through red, green, and yellow lights. In this program, each subroutine turns on a different color light. The flowchart of the following Traffic Light program shows the fundamental flow control structures. All computer programs can be created using three fundamental control structures: sequence, selection, and iteration. In this program, the sequence of subroutines turns on each light. A consecutive series of steps is a **sequence control structure**. The connecting line returns to the start of the sequence to show repetition. A repetition of steps is a loop or **iteration control structure**. In the RedLightX subroutine, if the remote signal is on (the condition),the red light is on 5 seconds; otherwise, the red light is on 30 seconds. This is a selection or decision control structure. A **selection control structure** bases the next step on a condition that is true or false.



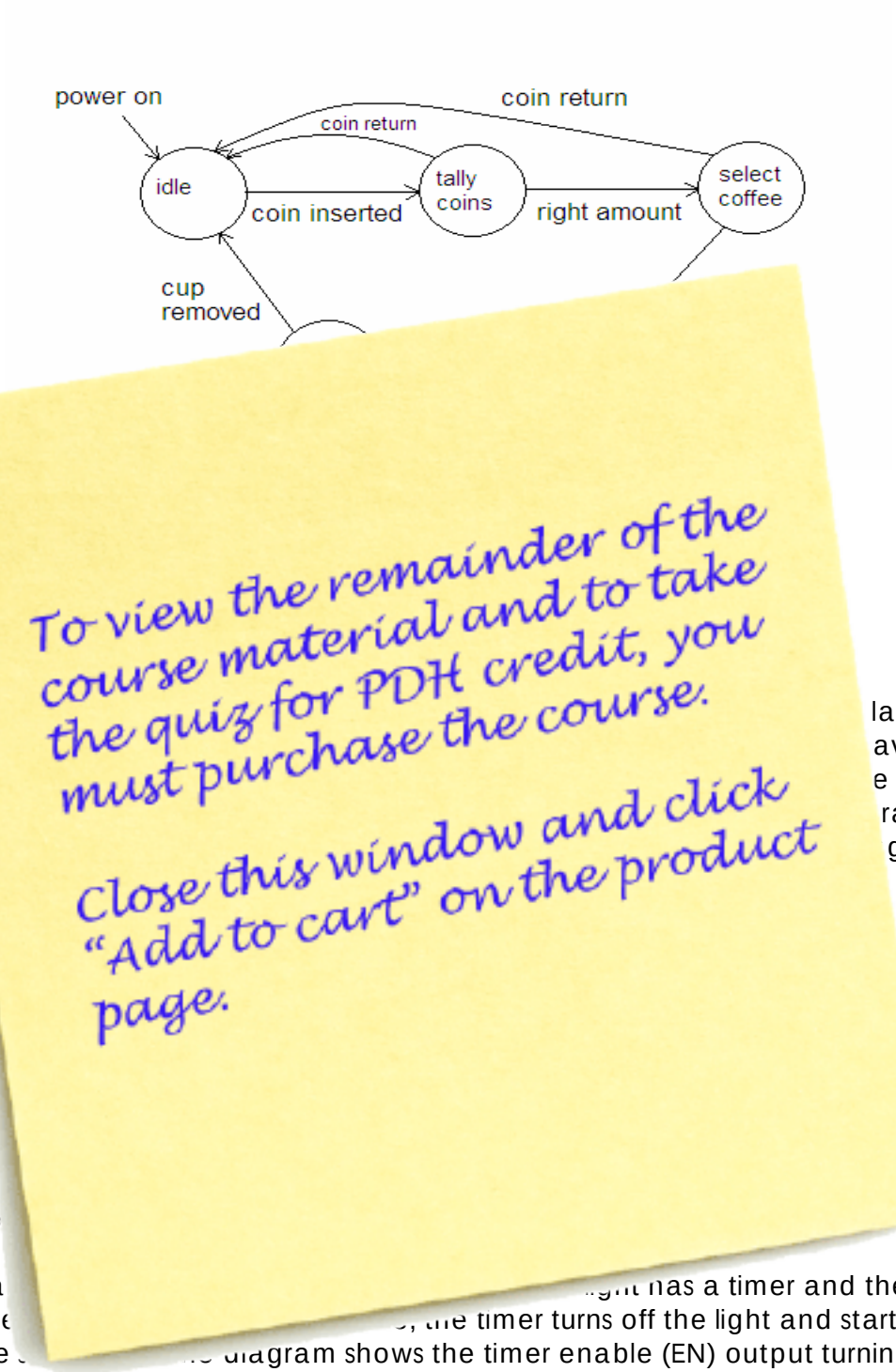
Pseudocode is a design tool that describes the instructions in an outline fashion. Pseudocode is similar to programming languages, so coding into a computer language can be easier. The following pseudocode is an example of a simple vending machine program design that dispenses coffee or coke and calculates the change. In the beginning of a pseudocode program, one defines the variables used in the program.

```
VARIABLES
price $0.50, $1.00
change $0.00, $0.25, $0.50
quarter = $0.25
selection = “coffee”, “coke”
dollarReader = True, False
```

Pseudocode programs include words and symbols that describe computer operations such as input, output, AND, OR, add (+), subtract (-), multiply (*) and divide (/). The “=” can mean a comparison of two values, or an assignment of a value to a variable. Pseudocode can implement the fundamental control structures: series of instructions (sequence);“if...then...else” statements (selection); “do while” statements for repeating instructions based on a condition being true(iteration). These are shown below.

```
START:
output "Make A Selection"
input selection
if selection = “coffee” then
        price = $0.50
else
        price = $1.00
endif
if dollarReader = True
    change = $1.00 – price
    dowhile  change is greater than or equal to $0.25
        dispense a quarter
        change = change - $0.25
        endwhile
        dispense selection
        output "Thank You"
endif
END
```

State diagrams define a system using states and transitions between states. States are written in the circles and directional lines show the transitions to the next state. Only one state is active at one time. A state diagram for a simple coffee machine is given below. Here we can see that when powered up, the machine will start in an idle state. The transitions are based on the inputs of sensors in the coffee machine. The states can indicate the operations that need to be further defined in detail.



Codin

Coding language. The pro available for the e personnel availab rams using the IEC g of the program

Ladder D r logic. This is the ased on relay and rical technicia compone input compone uts are relay soler anced functions s gram is shown bel

The diagra ght has a timer and the timer turns on the , the timer turns off the light and starts the next timer in the diagram shows the timer enable (EN) output turning on the light by closing normally open (EN) contacts. After the preset wait time, the timer done (DN) output turns off the light by opening normally closed (DN) contacts and starts the next timer by closing the normally open RedLight.DN contacts.