The design for an algorithm, or a program, is often expressed in a general notation that is independent of the programming language that may eventually be used when the design is implemented.

Such a notation is called a *pseudocode*.

Many pseudocode notations have been devised. The one presented here is simple and small, and is adequate to express designs that use the language features covered in an introductory programming course.

In particular, we require the ability to express identifiers, assignment, arithmetic and logical operations, selection, iteration, lists (arrays), and functions.
Identifiers and Arithmetic Expressions

An identifier name will be expressed using any descriptive alphanumerics string the designer wishes to employ. The only restrictions are that the name must not be one of the pseudocode keywords defined later in this chapter and the name must contain only digits and letters.

Arithmetic operations will be expressed using the C++ notations:

- unary minus 
- addition 
- subtraction 
- multiplication 
- division 
- remainder

The usual algebraic rules will be used to determine validity and meaning of arithmetic expressions. If desired, grouping of terms will be indicated using parentheses only.
The assignment of a value to a variable will be indicated by a statement of the form:

```
<target variable name> <-- <expression>
```

Assignment indicates that the existing value, if any, of the target variable will be replaced by the value of the expression on the right-hand side of the assignment statement.
The words True and False will be used to indicate the corresponding logical values.

The following logical operations will be denoted as indicated:

- conjunction: and
- disjunction: or
- negation: not

Parentheses may be used to group logical expressions.
In any algorithm it is vital that the designer specify the flow of execution; that is, the order in which the instructions are to be carried out.

We consider three essential modes of execution:

- sequential flow
- selection
- iteration

The flow control mechanisms presented in the following slides are generic. They have analogs in C++, but the syntax used here is not that of the C++ language.
In sequential flow, a sequence of instructions are carried out in the order they are listed. No instruction is skipped over and no instruction is done more than once, unless it is listed more than one time.

There is no general notation for sequential flow. Instead, sequential flow is assumed unless some other flow control mechanisms are present.

For example:

```cpp
NumEmployees <-- 0
Read an id number into:  IDNumber
Add 1 to NumEmployees
```
The simplest selection situation is to determine whether or not to carry out a particular action. If a certain condition is in effect, the action will be taken. Otherwise, the action will not be taken.

This is called the “simple if” — we will denote it as follows:

\[
\text{If} \quad < \text{some condition} > \quad \text{Then} \\
\quad < \text{some action(s)} > \\
\quad \text{Endif}
\]

where <some condition> stands for an expression that is either true or false, a Boolean expression and <some action(s)> stands for a sequence of zero or more instructions that form the body of the if statement (the if clause).
A slightly more complex selection scenario arises when choosing between two alternatives. If a certain condition is in effect, then one action will be taken. Otherwise, the other action will be taken.

This is called the “if - else” — we will denote it as follows:

\[
\text{If} \quad \text{< some condition >} \quad \text{Then} \\
\quad \text{< some action(s) >} \\
\text{Else} \\
\quad \text{< other action(s) >} \\
\text{Endif}
\]
Forms of iteration are distinguished according to how the decision to stop repeating the action is made.

In many situations, an action will be repeatedly carried out as long as some condition holds (and will be stopped as soon as that condition no longer holds). This is called an “event-controlled” iteration.

In other situations, an action will be carried out a specific number of times. This is called a “count-controlled” iteration.
The most basic event-controlled iteration mechanism is the “while loop”, denoted as follows:

```
While < some condition > Do
    < some action(s) >
Endwhile
```

If the condition is false, the action(s) inside the while loop are never carried out. Otherwise, they are carried out until the condition becomes false. Thus, the loop is “waiting” for an event to occur, namely for the given condition to become false.
The basic count-controlled iteration mechanism is the “for loop”, denoted by:

\[
\text{For } \text{counter} = \text{start} \text{ to } \text{counter} = \text{stop} \text{ Do} \\
< \text{some action(s)} > \\
\text{Endfor}
\]

Here, the value of counter is automatically increased by 1 at the end of each pass through the loop. The number of iterations is determined entirely by the initial value given to counter and the value of stop.
To denote an ordered list of data items we borrow the C++ notation for an array:

myList[]

denotes a list. To denote a particular item in the list, place a constant or identifier within the brackets:

myList[4]

denotes the item at position 4. Positions are numbered starting at 0.
A function body is represented using the notations already defined. The beginning and end of the function body are denoted as shown below. The parameter list, if any, is indicated by means of a C++ style header comment preceding the function body:

```cpp
///////////////////////////////////////////////////////////////////////
// FedTax is used to calculate the proper income tax for an employee.
// Parameter:
//   GrossPay  employee’s gross pay amount
// Return value:
//   FIT      income tax withheld for employee
Begin FedTax:

   I. If GrossPay is in high bracket Then
       FIT <-- HiTaxRate * GrossPay
   Else If GrossPay is in middle bracket Then
       FIT <-- MidTaxRate * GrossPay
   Else
       FIT <-- LoTaxRate * GrossPay
   Endif

   II. Return FIT

End FedTax.
```
The pseudocode notations defined should be sufficient for most needs. However, these are suggestions, not a straightjacket. If you need additional notations, invent them. Just be sure to pick reasonable notations, explain them, and then use them consistently in your design documents.

In many cases it is preferable to express a condition or action with an English phrase. Note the if-statement conditions on the previous slide.

The following appendix shows a design outline for the payroll program that may be used for the first programming assignment in your class. It illustrates many of the ideas presented here.

Other design outlines will be posted on the course website.