Text:

1. Algorithms

An algorithm is a general solution of a problem which can be written as a verbal description of a precise, logical sequence of actions. Some examples of algorithms are

- ✓ Cooking recipes
- ✓ Assembly instructions for appliances and toys
- ✓ Precise directions to reach a friend's house

A computer program is an algorithm expressed in a specific programming language. An algorithm is the key to developing a successful program.

Informally, an ***algorithm*** is any well-defined computational procedure that takes some value, or set of values, as ***input*** and produces some value, or set of values, as ***output***. An algorithm is thus a sequence of computational steps that transform the input into the output.

We can also view an algorithm as a tool for solving a well-specified ***computational problem***. The statement of the problem specifies in general terms the desired input/output relationship. The algorithm describes a specific computational procedure for achieving that input/output relationship.

Suppose a business office requires a program for computing its payroll. There are several people employed. They work regular hours, and sometimes overtime. The task is to compute pay for each person as well as compute the total pay disbursed. Given the problem, we may wish to express our recipe or algorithm for solving the payroll problem in terms of repeated computations of total pay for several people. The logical modules involved are easy to see.

```
Algorithm: PAYROLL

    Repeat the following while there is more data:

        get data for an individual,

        calculate the pay for the individual from the current data,

            and, update the cumulative pay disbursed so far,

        print the pay for the individual.

    After the data is exhausted, print the total pay disbursed.
```

The word algorithm comes from the name of a Persian author, Abu Ja'far Mohammed ibn Musa al Khowarizmi (c. 825 A.D.) who wrote a textbook on mathematics. An

examination of the latest edition of Webster's dictionary defines its meaning as "any special method of solving a certain kind of problem." But this word has taken on a special significance in computer science, where algorithm has come to refer to *a precise method useable by a computer for the solution of a problem*. This is what makes the notion of an algorithm different from words such as process, technique or method.

1.1. Properties of an Algorithm

1.1.1. **Finiteness**: An algorithm is composed of a finite set of steps (i.e. they terminate after a finite number of operations), each of which may require one or more operations.

1.1.2. **Definiteness**: The possibility of a computer carrying out these operations necessitates that certain constraints be placed on the type of operations an algorithm can include. For example, each operation must be definite, meaning that it must be perfectly clear what should be done. Directions such as "compute 5/0" or "add 6 or 7 to x" are not permitted because it is not clear what the result is or which of the two possibilities should be done.

1.1.3. **Effectiveness**: Another important property each operation should have is that it be definite; each step must be such that it can, at least in principle, be done by a person using pencil and paper in a finite amount of time. Performing arithmetic on integers is an example of an effective operation, but arithmetic with real numbers is not, since some values may be expressible only by an infinitely long decimal expansion. Adding two such numbers would violate the effectiveness property.

1.1.4. **Input & Output**: An algorithm produces one or more outputs and may have zero or more inputs which are externally supplied.

An algorithm is a well-ordered collection of unambiguous, effectively computable instructions that produce a result and halt in a finite amount of time.

Computational Procedure

There is another word for an algorithm which obeys all of the above properties except termination, and that is computational procedure. One important example of a computational procedure is the operating system of a digital computer. This procedure is designed to control the execution of jobs, such that when no jobs are available, it does not terminate, but continues in a waiting state until a new job is entered. Though computational procedures include important examples such as this one, we will restrict our study to those computational procedures which always terminate.

A related consideration is that the time for termination should be reasonably short. For example, an algorithm could be devised which, for any given position in the game of chess, decides if that is a winning position. The algorithm works by examining all possible moves and countermoves that could be made from the starting position. The

difficulty with this algorithm is that even using the most modern computers it may take **billions of years** to make the decision. Therefore, we will be very concerned with **analyzing** the efficiency of each of our algorithms.

In order to help us achieve the criterion of definiteness, algorithms will be written in a programming language. Such languages are designed so that each legitimate sentence has a unique meaning. A program is the expression of an algorithm in a programming language. Sometimes words such as procedure or subroutine are used synonymously for program.

2. Pseudo-code Conventions

We have seen that an algorithm is a well-ordered collection of unambiguous, effectively computable instructions that produce a result and halt in a finite amount of time.

The instructions that are unambiguous and effectively computable depend on the *computing agent* executing the algorithm. Establishing a well-ordered collection of those instructions depends on the language used to describe the algorithm. While we can write algorithms for ourselves without deep reflection on this definition (since we understand what is effectively computable and unambiguous for ourselves), if we want to communicate algorithmic solutions to others, we must establish a reasonable common computing agent. Below we describe a set of instructions that will define a language and effectively computable instructions for a *pseudo-machine* that will serve as our target computing agent.

2.1. Computational State

Our computing agent must be able to keep track of a variety of values needed in executing the algorithm. The values involved in the execution of an algorithm make up the *state* of the computation. The state is dynamic over the execution time of the algorithm, as values associated with particular attributes of the algorithm may change as the algorithm executes.

2.2. Values or Variables

As algorithm writers, we need a way to describe the various values in the state of the computation. Each value will be described with a name, written in lowercase letters, and called a *variable*. For example, we may want to write an algorithm that outputs the integers between one and ten. In order to know which value to output next, we could establish a variable count that will represent the count to output next.

2.3. Lists of values and indices (Arrays)

Sometimes we need to keep track of lists of associated values, for example a list of ten names. Rather than think up ten different variable names, we can use list notation and refer to *name*[1]*, name*[2]*, ..., name*[10] to refer to the ten names. We say that name is the *list* of values and that the integer in [ ] is the *index* indicating which value in the list we are referring to.

When using a *list* of values, we may use a variable to keep track of which item in the list we are interested in. For example, we may have a variable *i* that keeps track of which name we want to look at. *name*[i] would then refer to the value in *list* name found at the index indicated by the value of *i*.

2.4. Interacting with the User

After an algorithm designer writes an algorithm, a user decides to run the algorithm on a particular computing agent. The computing agent will need to interact with the user at least once, when outputting the result to the user. The agent may also need to receive information from the user.

2.4.1. Output from computing agent to user

Any of the following may be used to indicate the agent is outputting information to the user:

Output
Print
Display

The command can be followed by text in quotes, which is output verbatim, and variables, whose value is output. Some examples:

Output "Please enter an integer"
Output "The current counter value is " count

2.4.2. Input from user to agent

Input from the user is indicated by Input followed by a list of variables that will store the values input from the user.

For example, “Input n” inputs a single value, storing it into the variable named n.

2.5. Changing the value of a variable

Algorithms need to be able to change the value of variables. To set the value of a variable to a new value we will use the following command:

Set *variable* To *value*

Where *variable* is the name of the variable whose value the algorithm is changing and value is an expression whose value should be stored in the variable.

2.6. Expressions.

Many times, the expressions set into the variable are mathematical expressions. The usual operators +, -, *, / are available. We also have two other operators, div and mod.

The value “x div y” is the integer portion of x / y.

The value “x mod y” is the integer remainder of x / y.

For example,

7 div 2 is 3 and 7 mod 2 is 1.

Expressions can also be *boolean*. Boolean expressions have only two possible values, true or false. The operators used in boolean expressions include the usual comparison operators: <, ≤, >, ≥, = and logical operators, AND, OR, NOT.

2.7. Conditional statements

To describe a point in the algorithm in which steps are executed only if some condition is true, we use the if statement.

```
if (condition) then
BEGIN
statements...
END
```

"statements... " indicates any statements may be placed between the BEGIN/END. The statements are executed only if the *condition* expression evaluates to true.

One may also want to indicate an alternative set of statements to be executed if the condition evaluates to false. In this case the statement is written as:

```
If (condition) Then
BEGIN
        statements...
END
Else
BEGIN
        statements...
END
```

2.8. Iteration

To indicate repetition of statements in an unambiguous way, the algorithm will use one of the following statements:

```
Repeat
        statements...
Until (condition)
```

executes the statements inside the Repeat/Until block, then tests the condition. If the condition is false, the statements are executed again. Note that the statements are always executed at least once.

```
RepeatWhile (condition)
BEGIN
        statements...
END
```

tests the condition. If it is true, the statements inside the BEGIN/END block are executed, and then the condition is tested again. When the condition is false, control falls out of the loop.

Example

The following algorithm inputs an integer and outputs the values between 1 and that value.

```
Output "Please enter a positive integer value"
```

```
Input n
Set count To 1
RepeatWhile (count < n+1)
BEGIN
   Output count
   Set count To (count + 1)
END
```

Note: With careful use of indentation, the BEGIN/END pair may be omitted since they are understood.