Programming Fundamentals

♦ Computers are really very dumb machines -- they only do what they are told to do.

♦ Most computers perform their operations on a very primitive level.

♦ The basic operations of a computer system is called the computer's instruction set.

♦ In order to solve a problem using a computer, we must express the solution to that problem in a language that the computer can understand - through the instruction set.

♦ A computer program is just a collection of the instructions necessary to solve a specific problem.

♦ The approach or method that we use to solve the problem is called an algorithm.

♦ To develop a program to solve a particular problem, we first express the solution to the problem in terms of an algorithm.

♦ With the algorithm in hand, we can then write the instructions necessary to implement the algorithm on a particular computer system.

Example - Algorithms

Problem: Develop a program that tests if a number is even or odd.

Algorithm #1: Divide the number by two.
If the remainder is zero, then the number is even.
Otherwise (the remainder is one), the number is odd.

Algorithm #2: Test the least significant bit of the number.
If the bit is one, then the number is odd.
Otherwise (the bit is zero), the number is even.
Higher-Level Languages

- When computers were first developed, the only way they could be programmed was with binary numbers that corresponded directly to the machine instructions and locations in the computer's memory. (Machine language).

- **Assembly language** enabled the programmer to work with the machine on a slightly higher level.

- A special program called an **assembler** translates the assembly language programs from its symbolic format into machine language.

- Because a one-to-one correspondence exists between assembly language instructions and machine language instructions, assembly language is called a **low-level language**.

- The programmer must still learn the instruction set of the particular computer system in order to write a program.

- Assembly language programs are not **portable** -- they will not run on a different type of computer without being rewritten (machine-dependent).

- Operations of a higher-level language are much more sophisticated -- one statement would result in many different machine instructions being executed.

- Standardization of the syntax of a higher-level language mean that a program could be written to be machine independent -- a program could run on any machine that supported the language with few or no changes.

- An **interpreter** is a special program that translates the statements of a program developed in a higher-level language into machine language.

- This operation is done on-the-fly – as the interpreter reads each high-level language statement, it executes one or more machine-language instructions.

- An **compiler** is a special program that translates the statements of a program developed in a higher-level language into machine language – by producing a separate machine-language program called an **executable**.
Operating Systems

- An operating system is a (machine-language) program that controls the entire operation of a computer system.
- All input/output (I/O) operations are channeled through the operating system.
- The operating system must also manage the computer's resources and must handle the execution of machine-language (i.e., executable) programs.

Writing and Running Interpreted Programs

- An interpreter is a program that analyzes a program developed in a particular computer language and then executes it on your particular computer system.
- The program that is to be interpreted is first typed into a file on the computer.
- A text editor must usually be used to enter a program into a file.
- Once the source program has been entered into a file, you can then run it through the interpreter for that particular language.
- The interpretation process is initiated by typing a special command. When this command is entered, the name of the file that contains the program must also be specified.
- Sometimes you only have to give the name of the program, and the operating system will automatically run the interpreter for it.
- **Step 1:** the interpreter examines each statement in the program and checks for syntax and semantic errors.
- **Step 2:** the interpreter translates each statement to a "lower" form -- usually directly to machine language.
- When the program is executed, each of the statements of the program is **sequentially** executed.
If the program requests any data from the user, called **input**, the program will temporarily suspend its execution and wait for the user to enter the data.

Results that are displayed by the program are called **output**. (Normally displayed on the screen or terminal.)

If all goes well (*and it probably won't the first time the program is executed*), the program will do what it is supposed to do.

If things go wrong, it will be necessary to go back and re-analyze the program's logic. This is called **debugging** -- you try to remove all the bugs (features?) from the program.

In order to debug a program, the source code usually has to be changed. Then the entire process of interpreting and executing the program is repeated...

**The Programmer's Drinking Song**

"99 bugs in the code,
99 little bugs.
Fix one bug, run it again, ...
100 bugs in the code."
Structured Programming and Control Structures

♦ In this course, we teach *structured programming*, a disciplined approach to programming that results in programs that are easy to read and understand and are less likely to cause errors.

♦ We follow accepted program style guidelines (such as using meaningful names for identifiers) to write code that is adequately documented with comments and is clean and readable.

♦ Obscure tricks and programming shortcuts are strongly discouraged.

♦ Program maintenance involves modifying a program to remove previously-undetected bugs and to keep it up-to-date.

♦ It is not uncommon to maintain a program for five years or more, often after the programmers who originally coded it have left the company or have moved onto other positions.

Control Structures

♦ Structured programming uses control structures to control the flow of statement execution in a program.

♦ The control structures of a programming language let us combine individual statements into a single program entity with one entry point and one exit point.

♦ We can then write a program as a sequence of control structures rather than a sequence of individual statements.

♦ There are three categories of control structures:

1. **Sequence**
2. **Selection**
3. **Repetition** (iteration)
Sequence

In batch files, sequence is illustrated by statements (commands). Control flows from statement 1 to statement 2 and so on – from the top of the batch file until the bottom.

The command interpreter never goes backwards or skips over commands – unless you use selection and repetition structures in your programs.

Selection

Selection structures allow the computer to make decisions in your programs.

In all of our previous algorithms, we execute each algorithm step exactly once in order.

♦ Often, we are faced with situations which require two or more alternative courses of action, based on the input data.

♦ For example, a program to predict your health based on your resting heart rate.

♦ In batch files, we use if statements with logical (Boolean) expressions to form selection structures.

♦ Boolean expressions evaluate to either true or false.

♦ There are eight (8) forms of if statements. In each case, the if statement will cause the computer to execute a specified command if the Boolean condition evaluates to true. If the condition is false, then the command is ignored.

♦ In most cases, the command will be a goto.
if "string1" == "string2" command

if not "string1" == "string2" command

string1 and string2 can be a combination of literal characters (i.e., letters, numbers, and symbols that you can type on the keyboard) and variable names.

If string1 is identical to string2 (the same count of the same characters in the same order) then the condition is true and the command is executed or not.

The quotes around each string are used as placeholders, just in case one of the strings is empty (i.e., contains no characters.)

if errorlevel n command

if not errorlevel n command

n is the error level number from 0 to 255. Zero is used to indicate that no error has occurred (the previous command has completed successfully.) A non-zero value means something went wrong with the previous command.

if exist file command

if not exist file command

if exist directory\nul command

if not exist directory\nul command

Here, both file and directory can be a simple filename, a combination of directory names and filenames, separated by backslashes, with optional drive letter and colon. Variables can also be used.
**Important:**
In the case of checking if a directory exists, the \nul must be added to the name of the directory in question. Batch files can only check for the existence of files; nul is the name of a special (imaginary) file that happens to exist in each directory.

**Repetition**

- The third element of structured programming is repetition, which causes the computer to execute a sequence of operations several times.

- The only way to do repetition in batch files is with the goto command and a matching label that appears before the goto. Such a structure is called a loop.

```plaintext
|---:labelname
  |
  |
  | loop
  |
  +--- goto

The colon character (:) identifies a label, which the command interpreter remembers.

- Besides looping, goto is also used for two other special cases: branching and subroutine calls.

- Goto is related to selection statements in two ways:
  1. The usual command for an if statement is a goto, to branch to a label when the condition is true. (This is called a conditional branch; a goto by itself is called an unconditional branch.)
  2. An if statement is required to exit from a loop structure; otherwise, the batch file will contain an infinite (endless) loop.
Examples

1. *Sequence*

```Shell
echo Hello
echo Goodbye
```

First Hello is output; then Goodbye.

2a. *Selection - simple if command*

```Shell
if "%1" == "/?" goto help
```

Checks the first command-line parameter (%1) for a /? switch requesting help.

If the condition is true (i.e., the left side is identical to the right side), the goto command is executed and branches to a label called help. If the first command-line parameter is not provided (i.e., empty) or anything other than /?, the goto is skipped.

In both cases, the batch file resumes sequential processing: either after the :help label (the true case) or after the if statement (the false case).

2b. *Selection using an if-else structure*

```Shell
xcopy c:\ w:\ /e /i /d /c /r /y
if errorlevel 0 goto copyok
echo Error backing up files from drive C to drive W.
goto end

:copyok
...
```

There is no else keyword in batch files, instead an if statement, two labels and two goto statements are used.

In the above example, the xcopy command is used to backup all files and directories on drive C that have changed since the last backup. The files are copied to drive W (presumably a network drive on a server or other PC.)
If the `xcopy` command succeeds, then it will set the `errorlevel` to zero. Otherwise, it will set `errorlevel` to a non-zero value (typically, \textit{but not necessarily}, to 1). Since we cannot guarantee what the `errorlevel` will be set to when an error occurs, it is safer to check for `errorlevel` zero.

If no errors occurred, the condition is true and the `goto copyok` command will skip over the `echo` and `goto end` statements, and sequence will resume after the `copyok` label.

If an error occurred, the `goto copyok` command will be skipped and sequence will continue with the `echo` and `goto end` commands. (The end label is not shown, but it is somewhere well after the `copyok` label.)

3a. \textit{Repetition - endless loop}

```
:loop
echo Batch files suck!
goto loop
```

This will repeat the `echo` statement over and over and over again, until:

1) the user presses either a \texttt{Ctrl-C} or \texttt{Ctrl-Break} key combination and opts to terminate the batch file;

2) the user closes the Command Prompt window;

3) the user ends the VDM process using the Task Manager;

4) the user logs off or shuts down Windows;

5) Windows crashes (e.g., Blue Screen of Death);

6) the power goes out and/or the UPS battery drains;

7) the PC suffers a hardware failure;

8) any or all of the above events in combination.
3b. *Repetition - conditional loop*

```
:getfile
shift
if "%1" == "" goto end
copy %1 %todir%
goto getfile
:end
```

In this case (from Lab #2), the batch file reads command-line parameters, representing files to be copied. The `shift`, `if`, and `copy` statements are repeated until the `if` condition is true, that is, there are no more files left to copy (`%1` will be an empty string.)

The `goto end` statement causes the loop to end.

The `end` label is the last line in the batch file, and causes the batch file to end, as well.