# Storing Data in Files

# Why Save Data?

Almost every program needs to save data.

- Spreadsheets save numbers and formulae.
- Word processors store text.
- Web browsers save bookmarks.
- Games save progress and scores.

Unfortunately, when a program stops, any data in memory (RAM) is lost.

# Where to Save Data?

To save data, programs store information in some kind of file.  Eventually, the program will use that file to retrieve the (hopefully) useful data at a later time.

There are a number of file types that can be used to save data, but the most common is a simple text file.

# Text Files

A text file contains... text.  There are no special codes or formatting.  Thus it is typically used to store simple data made up of text and numbers.

A text file would not be used for complicated data such as images, audio, or video.

# File “Handles”

When we create a text file for personal use, we try to identify it with a meaningful name (e.g., addresses) so it can be easily identified for future use.

A program needs to create a “handle” to use as a reference to a particular filename. This allows the program to consistently keep track of the file while the program is running.

# Opening a File

In Turing, the file handles are integers. First we need to create an integer variable.

```
var fileNum : int
```

Next we open the file by associating the file handle with the name of our text file.

```
open : fileNum, "someFile.txt", put
```

# Opening a File for Output

In Turing, the file handles are integers. First we need to create an integer variable.

```
var fileNum : int
open : fileNum, "someFile.txt", put
```

Notice that we had to specify the operation that was going to be used on the file. In this case, we are opening the file for output.

This means only the put command can currently be used with this file.

# Output Data to a File

```
var fileNum : int
open : fileNum, "someFile.txt", put

put : fileNum, "some data for a file"

close : fileNum
```

The syntax of the put command is slightly different than what we have used before.

When finished with a file, it is important to close the file.

# Input Data from a File

To retrieve data from a file, the process is very similar. Now the file must be opened for input.

```
var fileNum : int
var data : string
open : fileNum, "someFile.txt", get
```

In order to get some data, it must be stored in a variable. When reading from text files, it is safest to use strings, although other data types are fine (provided you are very sure of the file).

# Input Data from a File

```
var fileNum : int
var data : string
open : fileNum, "someFile.txt", get

get : fileNum, data
put data  % output data to screen

close : fileNum
```

Again, the syntax is slightly different than a typical get statement.

# Input Data from a File

```
var fileNum : int
var data : string
open : fileNum, "someFile.txt", get

get : fileNum, data : *
```

Retrieving data from a file has the same limitations as getting input from the user. If there are spaces that you want to include in the string, you must put the wildcard (*) at the end of the get statement.