



Programming in 'C'

File Handling with C

Topics

- File Types (text / binary)
- Writing files
- Reading files
- Displaying the contents of a file.

Slide 2 (of 18)

File Types

- Files are used to as a permanent record of data
- There are two basic formats of how the data is held :
 - Text
 - Binary.

Slide 3 (of 18)

Text Files

- If the information is held in a text file, the data is stored as ASCII characters, which could be numbers.
- The data is structured into lines and thus cannot be regarded as simply being a sequence of values.
- Each line is terminated by a carriage return (“\r”) and line feed (“\n”) characters whilst the end-of-file marker is <CTRL><Z>.

Slide 4 (of 18)

Binary Files

- For information held in binary format, each data item takes up a pre-determined amount of space as follows:
 - Integer - 2 bytes
 - Real - 4 bytes
 - Character - 1 byte.
- NOTE: Other data items could be data structures such as
 - arrays, records, etc.

Slide 5 (of 18)

File Handling Modes In C

- In C, file handling can be done by either :
 - Standard (stream) I/O
 - System (low-level) I/O.

Slide 6 (of 18)

Different Ways Of Reading & Writing Data

- The different ways of reading and writing in the two modes are summarised in the following table :

Data Type	System I/O	Standard I/O
Character		fgetc(), fputc()
String		fgets(), fputs()
Formatted		fscanf(), fprintf()
Record (Block)	read() or write()	fread(), fwrite()

- NOTE :** read() and write() are not defined in the ANSI C standard; specific to Turbo Borland C/C++

Slide 7 (of 18)

Opening Mode Parameter

Mode String	Meaning
"r"	Open for reading. The file must already exist.
"w"	Open for writing. If the file exist its contents will be written over. If it does not exist it will be created
"a"	Open for append. Material will be added to the end of an existing file, or a new file will be created.
"r+"	Open for both reading and writing. The file must already exist.
"w+"	Open for both writing and reading. If the file exists its contents will be written over. If the file does not exist it will be created.
"a+"	Open for both reading and appending. If the file does not exist it will be created

Steps In File Handling

- Create the file descriptor (logical file).
- Open the physical file and link to the logical file.
- Use the file to read/write data.
- Close the physical file.

Slide 9 (of 18)

Writing to a File

```
/*Global Variables*/
FILE *fptr /* FILE is a data type,
           *fptr is a pointer to filename */

/* ===== Function ===== */
void SaveData(void){

/*Open file for Writing to */
  fptr = fopen("C:\\MYDATA\\TESTDAT.TXT" , "w");

/*Send data to file */
  fprintf (fptr, "%d," , Value1);

/*Close file and place end of file marker */
  fclose(fptr);
}
```

Slide 10 (of 18)

Writing to a File

```
/*Global Variables */
FILE *fptr /* FILE is a data type,
           *fptr is a pointer to filename */

/* ===== Function =====
DOS file name
Indicates WRITE mode
Data separator
void SaveData(void){

/*Open file for Writing to */
  fptr = fopen("C:\\MYDATA\\TESTDAT.TXT" , "w");

/*Send data to file */
  fprintf (fptr, "%d," , Value1);

/*Close file and place end of file marker */
  fclose(fptr);
}
```

Reading from a File

```
/*Global Variables */
FILE *fptr /* FILE is a data type,
           *fptr is a pointer to filename */

/* ===== Function ===== */
void ReadData(void){

/*Open file for Reading */
  fptr = fopen("C:\\MYDATA\\TESTDAT.TXT" , "r");

/*Gets data from file */
  fscanf (fptr, "%d," , &Value1);

/*Close file and place end of file marker */
  fclose(fptr);
}
```

Slide 12 (of 18)

Reading from a File

```
/*Global Variables */
FILE *fptr /* FILE is a data type,
            *fptr is a pointer to filename */

/* ===== Function =====
void ReadData(void){
/*Open file for Reading */
    fptr = fopen("C:\\MYDATA\\TESTDAT.TXT", "r");
/*Gets data from file */
    fscanf (fptr, "%d", &value1);
/*Close file and place end of file marker */
    fclose(fp);
}
```

DOS file name
Indicates READ mode
Data separator
Address Operator

Writing to files

- Two basic ways of writing to a file:
 - 'on the fly' - in real time as the data is created:
 - Slow as file needs to be continually used throughout program run, but holds large amounts of data
 - from an array stored in memory:
 - fast but amount of data limited to size of array

Slide 14 (of 18)

Writing Data - Example

```
/*declare file pointer */
FILE *WindSpeed;

/*opens file for writing data to */
WindSpeed = fopen("demodat.txt", "w");

/*loop to save each wind speed reading */
for(x=0; x<80; x++){
/*send data to file */
    fprintf(WindSpeed, "%d", Wind[x]);
}

/*close file */
fclose(WindSpeed);
```

Slide 15 (of 18)

Reading Data - Example

```
/*declare file pointer */  
FILE *WindSpeed;  
/*opens file for reading data from */  
WindSpeed = fopen("demodat.txt","r");  
/*loop to read each wind speed reading */  
do{  
/*gets data to file */  
fscanf(WindSpeed, "%d",&Wind[x]);  
x++; /*increase counter */  
} while(!feof(WindSpeed))  
/*close file */  
fclose(WindSpeed);
```

Displaying Data Files

- Text editor, e.g. Turbo C
- Wordprocessor, Word
- DOS command 'type *filename.ext*'

Slide 17 (of 18)

Summary

- Why use files - Backing store v Main Memory
- File types - text / binary
- Writing and Reading from files in 'C'

Slide 18 (of 18)
