CSC180: Lecture 29

Wael Aboulsaadat

wael@cs.toronto.edu

http://portal.utoronto.ca/

Acknowledgement: These slides are partially based on the slides supplied with Prof. Savitch book: Problem Solving with C

Enumeration

- Enumeration is a user-defined data type. It is defined using the keyword enum and the syntax is:

  enum tag_name {name_0, …, name_n} ;

- The tag_name is not used directly. The names in the braces are symbolic constants that take on integer values from zero through n. As an example, the statement:

  enum colors { red, yellow, green } ;

- creates three constants. red is assigned the value 0, yellow is assigned 1 and green is assigned 2.

Enumeration Example

```
/* This program uses enumerated data types to  access the elements of
    an array */
#include <stdio.h>
int main( ) {
   int March[5][7]={{0,0,1,2,3,4,5},{6,7,8,9,10,11,12},
                    {13,14,15,16,17,18,19},{20,21,22,23,24,25,26},
                    {27,28,29,30,31,0,0}};
    enum days {Sunday, Monday, Tuesday,  Wednesday, Thursday,
    Friday, Saturday};
    enum week {week_one, week_two, week_three, week_four,
    week_five};


    printf ("Monday the third week  of March is: March
        %d\n", March [week_three] [Monday] );
    return 0;
}
```

Enumeration

- Values can be set explicitly with **=**

- Example:

```
enum Months { JAN =1, FEB, MAR, APR, MAY , JUN,
              JUL, AUG, SEP, OCT, NOV, DEC};
```

  - Creates a new type enum Months in which the identifiers are set to the integers **`1 to 12`**

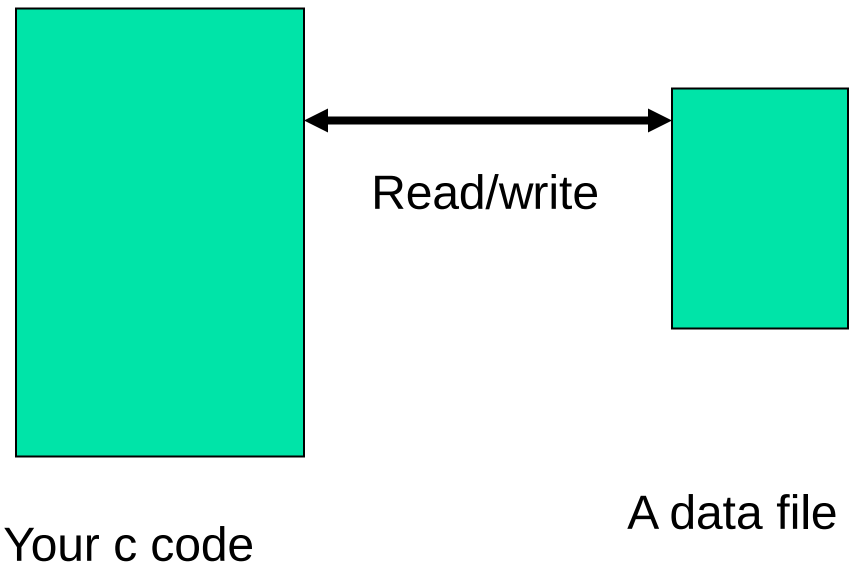
File I/O

File

- A stream of bytes
  - Text file: user readable
  - Binary file: machine readable

Access Files

- What operations we can do with a file?
  - Open
  - Read / Write
  - Close



Access File by File Pointer

- File pointer: declare for each file used
  - Declared as

```
FILE *filepointername;
```

Example:

```
        FILE *infile, *outfile;
```

What’s in a FILE struct?

- Name
  `C:\myinput.txt`
- Read/Write
- Type (binary or ASCII text)
- Access (security; single/multiple user)
- Current Reading/Writing Position in the file
- …

File i/o function calls

```
fopen(filename, file_access)
```

- `filename` *is the location + name of the file to open*
  - *A CString   “C:\\myfile.txt”*

File i/o function calls

`fopen(`*`filename, file_access`*`)`

- `File_access`
    - t: text
    - b:binary
    - r: read (for input)
    - w: write (for output)
        - If file not exist, create it
        - If file exists, erase file content (writes over it)
    - a: append to end of file, for updating
        - If file not exist, create it
    - r+: read and write to a file; do not overwrite the old file
    - w+: read and write destroy and create a new file
    - a+: read and append and create a new file

File i/o function calls

`fopen(`*`filename, file_access`*`)`

- returns:

  file_handle, that is the address of `FILE` (a `FILE *`) on success

  or

  `NULL` (zero) on failure

File i/o function calls

```
fclose(file_handle)
```

- Closes a file
- This is recommended for input files (to free up system resources)
- This is required for output files (as often times the O/S does not write the last bit of a file out to the disk until the file is closed).

File i/o function calls

`fprintf(`*`file_handle, format_specifier, 0 or more variables`*`)`

- *`file_handle`*: is address returned by `fopen()`
- *format_specifier:* same as for `printf()`
- *0 or more variables:* same as `printf()`

File i/o function calls

```
fscanf(file_handle,format_specifier,
1 or more variable address )
```

- *`file_handle`:* is address returned by `fopen()`
- Read like scanf does, just from a file

- Returns number of arguments read and assigned or EOF if end of file is reached before anything is assigned

Sample program

- Read three integer values from the file `myinput.txt`
- Determine sum and average
- Write the original three values as well as the sum and average to the file `myoutput.txt`

The program (part 1)

```
#include <stdio.h>
#include <stdlib.h>

void main()
{
   FILE *infile;
   FILE *outfile;
   int x,y,z,sum;
   float avg;

   // Open input file, exit if error
   infile=fopen("myinput.txt","r+t");
   if (infile==NULL)
   {
         printf("Error opening myinput.txt\n");
         exit(0);
   }

   // Generally file opens are done as below
   if ((outfile=fopen("myoutput.txt","w+t"))==NULL)
   {
         printf("Error opening myoutput.txt\n");
         exit(0);
   }
```

The program (part 2)

```
   // read the three values
   // its a good idea to  account for \n's in the file
   fscanf(infile,"%d\n",&x);
   fscanf(infile,"%d\n",&y);
   fscanf(infile,"%d\n",&z);

   // sum and avg
   sum = x+y+z;
   avg = (float)sum/3.0;

   // print out values
   fprintf(outfile,"Values: %d, %d, %d\n",x,y,z);
   fprintf(outfile,"Sum: %d\n",sum);
   fprintf(outfile,"Avg: %7.2f\n",avg);

   // close the files
   fclose(infile);
   fclose(outfile);
}
```

File i/o function calls

```
fgets(buffer, n, file_handle)
```

- Reading lines (CStrings)
- buffer is where the line is stored
- n is the max number of characters to be stored in buffer
- *file_handle:* is address returned by `openf()`
- Reads characters from file and stores them in buffer
- Stops when ‘\n’ is reached or when n-1 characters have been read
- Returns NULL on failure and buffer on success

File i/o function calls

```
fputs(buffer, file_handle)
```

- Writing CStrings to file
- Writes the contents of buffer to file_handle
- *file_handle:* is address returned by `openf()`
- Writes each character until the ‘\0’ is reached
  - Does not write ‘\0’ to the file