HANDLING FILE INPUT / OUTPUT OPERATIONS IN C

The C/C++ programming language does not have any built-in instructions to perform the Input / Output operations. They are provided by library functions.

Standard „formated" input/output file library < stdio.h >

I / O operations performed by **streams**

Streams are represented by specific "file variables" of the type **FILE**. Such a variable/structure is created automatically when you open the stream (it contains information about the file name, opening mode, etc.). All further operations on the stream, require a pointer to this structure.

Standard streams for input and output (opened automatically)

- **stdin** – standard input stream (console - keyboard)
- **stdout** – standard output stream (console - monitor)
- **stderr** – standard error messages stream (console)
- **stdprn** – standard printer stream

User defined streams:

```
FILE  *file,  *result_data_file ;          // definition of „file variables"
```

1. Accessing functions: **fopen** (returns a pointer to FILE) and **fclose**

FILE * **fopen**(char *__file_name__, char *__open_mode__)

The type of open_mode is defined by characters:

- **r** – open for reading (read only)
- **w** – open for writing (create new file)
- **a** – open for appending at the end
- **+** – with the possibility to update (overwrite)
- **b** – open as **binary** stream
- **t** – open as **text** stream

int **fclose**(FILE *stream) // function that closes the pointed file stream

```
FILE  *file;                  // Example creation of a new binary file, with the ability to update
file = fopen( "a:\\results.dat", "w+b" );
if( file == NULL )                                          // controling the I/O errors
   {
      printf( "Error while opening output file" );
      return -1;
   }
. . .
fclose( file );                // closing (releasing) the FILE, before exiting the program
```

2. List of functions for WRITING the data to file stream

Writing data to a TEXT file (data encoded in ASCII codes, spaces, NL, …)

```
int  fputc  ( int character, FILE *stream )          // writing a single character
int  fprintf ( FILE *stream, char *format, . . . )   // writing words / numbers
                          // high level formated output, with the same arguments as printf( )
int  fputs  ( char *text, FILE *stream )             // writing text lines (C-strings)
```

Writing data to a BINARY file (data encoded in rough bytes)

```
int  fwrite ( void*    memory_buffer,
              size_t   block_size,   size_t  block_count,
              FILE *   stream)
    // function that copies (block_count*block_count) bytes
    // from the specified memory area (buffer) to the stream (file)
```

Example

```
#include <stdio.h>
struct  TStudent
   {
      char  surname [31];
      char  name[16];
      int  age;
   };

int main( void )  {
   FILE *file;
   TStudent   student_array[10];
   if ( (file = fopen( "test.bin" ,  "wb" ) ) != NULL )
      {  // Record the entire content of the student database to a binary file
         fwrite( student_array, sizeof(TStudent), 10 , file );
         fclose( file );
      }

   if ( (file = fopen( "test.txt" ,  "wt" ) ) != NULL )
      {  // Record the entire content of the student database to a text file
         for( int i = 0;  i < 10;  i++ )
            fprintf(  file,  "%s  %s  %d \n", student_array[ i ].surname,
                      student_array[ i ].name, student_array[ i ].age );
         fclose( file );
      }
   return 0;
}
```

If we use predefined **stdout** (standard output stream) as a stream identifier, then the output will be made on the console / screen monitor.

eg. `fprintf( stdout, "format" , ..... )  ≡  printf( "format" , ..... )`

3. List of functions for READING the data from a file stream

Reading the data from a **TEXT** file (data encoded in ASCII codes, spaces, NL, …)

```
int  fgetc ( FILE *stream )                          // reading the single character

int  fscanf ( FILE *stream, char *format, . . . )    // reading words/numbers
                                                     // high level formated input, with the same arguments as scanf( )

char* fgets ( char *text, int max_length, FILE *stream )   // reading "lines"
                                                     // Loading a C-string (char array) consisting of at most (max_length-1) characters
```

Reading the data from a **BINARY** file (data encoded in rough bytes)

```
int  fread ( void*    memory_buffer,
             size_t   block_size,   size_t  block_count,
             FILE *   stream)
             // function that copies (block_count*block_count) bytes
             // from the file stream to the specified memory area (buffer)
```

Example

```
#include <stdio.h>
struct  TStudent
   {
      char  surname[31];
      char  name[16];
      int  age;
   };

void main( void )  {
   FILE *file;
   TStudent   student_array [10];
   int count;
   if ( (file = fopen( "test.bin" ,  "rb" ) ) != NULL )
      {  // Read the content of the student database from a binary file
         count = 0;
         while( fread( &student_array[count],sizeof(TStudent),1,file)==1)
            count++;
         fclose( file );
      }

   if ( (file = fopen( "test.txt" ,  "rt" ) ) != NULL )
      {  // Import the content of the student database from a text file
         for( int i = 0;  ( ! feof(file) )  && (i < 10);  i++ )
            fscanf( file,  "%s %s %d" , student_array[ i ].surname,
                    student_array[ i ].name,  &(student_array[ i ].age) );
         fclose( file );
      }
   return 0;
}
```

4. Auxiliary file-positioning functions:

int **feof** (FILE *__stream__) // *tests reaching the end of the given file*

int **fseek** (FILE *__stream__, long __offset__, int __origin__)

// *Move the file indicator position*
// *relatively to specified origin:*
SEEK_SET - the beginning of the file
SEEK_CUR - the current position
SEEK_END - the end of the file

long **ftell** (FILE *__stream__) // *returns current position of file indicator*

int **fflush** (FILE *__stream__) // *„ sweeps" the buffer of a given stream*

Example task on final test:

```
// Function that determines the position of the maximum value in a binary file
#include <stdio.h>
long Maximum( char *file_name )
{
   FILE *data_file;
   long position=0, max_pos = –1;
   double buffer, maximum;
   if ( (data_file = fopen( file_name , "rb" ) ) != NULL )
      {
         while( fread( & buffer, sizeof(double), 1, data_file) == 1)
            {
               if(position == 0 )
                  {
                     maximum = buffer;
                     max_pos = 0;
                  }
               else
                  if( l buffer > maximum )
                     {
                        maximum = buffer;
                        max_pos = position;
                     }
               position ++;
            }
         fclose( data_file );
      }
   return max_pos ;
}
```