CP2 Exercise 7 **04/12/2006**

1. Re-examine the calendar programs from the previous exercises and extract the most used functions from them into a separate source file with a separate header, containing the necessary type definitions and prototypes (*suggested filenames: calendar.c & calendar.h*). Then create a statically linked library libcal.a. Try building the calendar exercises, using the newly created library.

Notes:
a) secure `calendar.h` *using include guards to prevent multiple inclusion of the file*
b) to link an executable with a library in the same directory, the current directory must be added to the library path:
`gcc -ansi prog.c -L. -lcal -o program`
to add the current directory to the include path, the -I. option is necessary.

2. Similarly to the above exercise extract the functions from last week's tree exercises (*insert, depth, printInOrder, printPreOrder, printPostOrder*) into a separate source file with a separate header, containing the necessary type definitions and prototypes. Then create a statically linked library libtree.a. Then build the tree exercises, using the newly created library.

Under unix-like systems (*e.g. Linux*) library files are usually archives containing one or more C object files. These libraries are usually called *lib**X**.a*, where ***X*** is a string (*the name of the library*). The filename extension ***.a*** shows that it is an archive file. To enable programs to access functions from objects contained within a library, header files containing the prototypes of these functions must be provided.

Archives are created using the ***ar*** command:

```
syntax:   ar [option] archive_filename [object_file(s)]
examples: ar -r libX.a file.o
          ar -r libX.a file1.o file2.o
```

(useful) ***ar*** options:
-r add or replace object files in the archive
-d delete specified object files from the archive
-x extract copies of specified object files from archive