In this exercise we familiarize ourselves to the GNU/Linux & gcc environment.

Coding Practices

In order to have our programs more readable we use the following coding rules:

1. Intendation. We use here the K&R intendation style (so-called because it was used in Kernighan and Ritchie's book *The C Programming Language*) because it is commonly used in C. It keeps the first opening brace on the same line as the control statement, indents the statements within the braces (by 4 spaces), and puts the closing brace on the same indentation level as the control statement (on a line of its own). An example:
```
int main(int argc, char *argv[]) {
    ...
    while (x == y) {
        something();
        somethingelse();
        if (some_error)
            do_correct();
        else
            continue_as_usual();
    }
    finalthing();
    ...
}
```
2. Variable names. Use descriptive names for the variable. Your own defined types must start with a capital T, e.g.
```
typedef struct {
    int re;
    int im;
} Tcomplex;
```
3. Use empty lines to separate program groups (to separate functions from other functions, variable declarations from the code, etc.), e.g.
```
/*
 * Calculate CCITT (HDLC, X25) CRC
 */
unsigned crc(unsigned char *blk, unsigned len) {
    const unsigned poly = 0x8408;   // x^16+x^12+x^5+1

    register unsigned       result;
    register unsigned char  ch;
    int                     i;
    unsigned char          *p;

    result = 0xffff;
    for (p = blk; p < blk+len; p++) {
        ch = *p;
        for (i = 0; i < 8; i++) {
            if ((result^ch) & 0x001) {
                result >>= 1;
                result ^= poly;
            } else
                result >>= 1;
            ch >>= 1;
        }
    }
    return (result);
}


/*
 * Add extension to the filename
 */
char *AddExtension(char *FileName, char *Extension) {
    static char  Name[_MAX_PATH];
    char        *s1;

    /* copy basename */
    s1 = Name;
    while(*FileName && *FileName != '.')
        *s1++ = *FileName++;

    /* copy extension (if there are already no extension) */
    strcpy(s1, !*FileName ? Extension : FileName);

    return(Name);
}
```

Excercise 1a (Familiarization with Unix/Linux environments available in the lab as well familiarization with operating system services and development environment, 0,5p)

The main goal of the first exercise is to familiarize with the linux environment available in the lab. This is the Edunix machine that can be used via terminal connection. Edunix computer is running RedHat Enterprise Linux.

The purpose of the first exercise is that the given program is put to run in Edunix.

Remark. Instead of Edunix, you can also use your own portable computer, if it is running posix-compliant operating system (for example MacOsX).

Put the given program to work in practice (the program is as an appendix to this description). The program displays the contents of the working directory on the screen.

The program uses operating system services (system calls). These system calls are opendir, readdir and closedir.

You can choose yourself the editor you use to edit the program. If you have used to use command line editors like Emacs, Pico, Vi, etc, you can use them. If you feel uncomfortable with these text based editor, you can use NotePad++ in Windows and save the file in the Z: drive. Then after saving the file you can compile it in the PuTTY-Window.

Excercise 1b (Concurrent programs, system calls, 0,5p)

Give you a demonstration how processes are running concurrently and how they can access the same device (in this case this device is display so that you really can see it). This demonstration can be done by writing two simple programs. The first program displays all the time (in the infinite loop) character 1 and the second program displays in similar way continuously character 2. You can use one program with different command line parameters, like '1' and '2'.

When the programs are written, run them concurrently so that they use the same terminal. This can be done in the following way. Start the first program on the background using the command `./progr1.exe &`. After that start the second program in normal way (you can enter the command even you cannot see it, because program 1 does output all the time).

Now you should see that there is output from both programs on the display.

Remark 1. Use system call write() to write character to the display with the file descriptor STDOUT_FILENO. Information about the system calls can be obtained with the `man` command line command, e.g. `man 2 write`.

Remark 2. Use ctrl-C to stop the program. To stop the background process, you need to bring it first to the foreground using command `fg`.

Appendix 1 (Test program)

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

int main(int argc, char *argv[]) {
    DIR           *dp;
    struct dirent *dirp;

    printf("Directory content:\n");
    if ((dp = opendir(".")) != NULL) {
        while ((dirp = readdir(dp)) != NULL) {
        if (strcmp(dirp->d_name, ".") == 0 ||
            strcmp(dirp->d_name, "..") == 0)
          continue;

            printf("%s\n", dirp->d_name);
        }

        closedir(dp);
    } else
        fprintf(stderr, "Can't open current directory\n");
}
```