In this course we use Microsoft Visual Studio 2012 for compiling the programs. It is widely used (in real companies), and it has fairly easy to use graphical user interface.

Using Microsoft Visual Studio for Simple C Programs

A free version of Microsoft Visual C++ Express 2012 is available at: `http://msdn2.microsoft.com/en-us/express/aa700735.aspx` if you want to use the Visual Studio for your own computer.

Metropolia has Microsoft Visual Studio installed to the main server system which is operated using VMware View Client. To start Visual Studio:

1. Click the WMware icon on the desktop
2. then press Connect (to the connection server desktop.metropolia.fi)
3. finally select the Visual desktop (from the list)
4. Now you have connection to the virtual Windows environment running on the main server. Then click the 'Windows Logo' –button on the bottom left part of the screen
5. Select All Programs -> Programming -> Microsoft Visual Studio 2012. Now the Visual Studio should be appeared on your screen.
6. Visual Studio asks “Choose your default environment setting”, select 'Visual C++ Development Settings'

To edit your C program:

From the main menu select File -> New -> Project

In the New Project window:

1. Under Project types, select Win32 - Win32 Console Application
2. Name your project, and specify a location for your project directory. Note that the default project location is on the virtual machine whose filesystem is destroyed when you log off from the system. Therefore change the project location using 'Browse' button to your own network filesystem (e.g. drive letter z:)
3. Click 'OK', then 'next'
   In the Application Wizard:
4. Select Console application
5. Select Empty project
6. Deselect Precompiled header
   Once the project has been created, in the window on the left hand side you should see three folders:
   - Header Files
   - Resource Files
   - Source Files
7. Right-click on Source Files and Select Add-> New Item
8. Select Code, and give the file a name. The default here will be a file with a *.cpp extension (for a C++ file), so remember to include in your filename the filetype .c (for a C file).

To compile and run:

1. Press the green play button. By default, you will be running in debug mode and it will run your code and bring up the command window.
2. To prevent the command window from closing as soon as the program finishes execution, start the program from the menu Debug-Start-without-Debugging (or pressing Ctrl-F5).

   The other way to prevent window to close is to add the following line to the end of your main function:

   ```
   fflush(stdin); getchar();
   ```

   The first library function call will clear the keyboard buffer and the second library function waits for any input key, and will therefore keep your console window open until a key is pressed.

This is an example of the C source file

```
/*
 * fun.c – funny C program skeleton
 *
 * Written by Jarkko Vuori/Metropolia
 */

#include <stdio.h> // standard functions declared here
   .
   .
   .

int main(void) {
   .
   .
   .
   fflush(stdin); getchar();
}
```

All the input and output of our exercises is from the user keyboard and to the console screen. So you can use standard `scanf()` and `printf()` library functions for the input/output operations.

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);
}
```

Hint: Visual Studio reformats the selected source code by ctrl-k ctrl-f command. This can be useful when importing code from different sources.

Excercise 1. (Data abstraction and procedure abstraction)

The time of the day is expressed using hours and minutes (from the midnight, in 24h format). Write a program, that first reads the starting time (hours and minutes) of a certain process and then the ending time of the same process. When starting time and ending times are entered, the program calculates the duration of the process. The duration is also expressed in hours and minutes. Finally the program displays the duration of the process.

You have to use abstraction (procedure abstraction as well as data abstraction) in the implementation. Firstly this means that you have to divide the program into pieces using functions according the description above. Clearly the program consists of the following subtasks (functions):

1) read time (format hh mm)
2) check whether of the given time is earlier (extra feature, not mandatory)
3) calculate the time difference
4) display the time.

You have to use parameters to pass information to and from the functions. Secondly it means that the times are regarded as high level concepts where internal details of times do not "disturb" high level thinking.

Use meaningful names for the datatypes and functions (e.g. Ttime, read_time, print_time, time_diff) because we are using them later on (it is easier to memorize descriptive names than bizarre ones).

Remark. No special error handling is needed.

Hint1. The easiest way to calculate time difference is as follows: Convert first both times into minutes from the midnight. Then subtract the smaller minute value from the larger and finally convert time difference in minutes into hours and minutes (in backconversion you need the mathematical remainder operation which is marked in C-language by the character %).

Hint2. Visual Studio may be angry with the scanf() function (because scanf-function is prone to the buffer overflow security error). In order to bypass compiler's error messages put the statement `#define _CRT_SECURE_NO_WARNINGS` before the first `#include` line in your program.

Extra additional feature (optional, 0p)

To make time more generic and better reusable in possible other projects write an additional function compare_times that uses the same concept than standard C string comparison function compare in the sense that the function returns -1, if time1 is earlier than time2. The function returns 0 if times are equal and it return 1 if time1 is later than time2.

The prototype of comparison function is then like

```
int compare_times(Ttime time1, Ttime time2);
```

This function can be tested with a simple fraction of code like

```
if (compare_times(t1, t2) == -1)
    printf("\nTime 1 is earlier");
 else if (compare_times(t1, t2) == 0)
    printf("\nTimes are equal");
 else if (compare_times(t1, t2) == 1)
      printf("\nTime 1 is later");
```

and entering different times for testing.