Excercise 2 (Threading 10p)

Assume that for the actions performed by the game (driving the AI, updating the game world, etc.) there is a common superclass `Game_Task` defined. The class defines a pure virtual function `perform()` that performs the action in question:

```
virtual void Game_Task::perform() = 0;
```

All actions are independent of each other (do not use the same data, etc.).

The game maintains a vector of actions. The actions are executed sequentially in a loop:

```
std::vector<Game_Task*> tasks;
...
int number_of_tasks = tasks.size();
for (int i = 0; i < number_of_tasks; i++)
{
    task[i]->perform();
}
// Continue only after all tasks are complete!
```

Outline how the loop above could be parallelized using

a) std::thread class (e.g. create two threads that pass through the operation vector in half)
b) asynchronous function calls (no need to limit the number of calls)

If/when you concretely code a solution, create a superclass Game_Task and a few subclasses for it, whose perform function consumes a suitable amount of processor time somewhere. Measure the time you get from the parallelization performance time benefit of parallelization.

Programming environment

In this course, we use Microsoft Visual Studio Community Edition 2022 for compiling the programs. It is widely used, and it has easy to use graphical user interface. It is very similar to Visual Studio Code, but it has the compiler included.

It is acceptable to use some other development environment if you are used to it (e.g., gcc and Linux).

Using Microsoft Visual Studio for Simple C++ Programs

A free version of Microsoft Visual C++ Studio is available in: `https://visualstudio.microsoft.com/downloads/` if you want to use the Visual Studio for your own computer. Select the 2022 Community Edition (be sure to check Desktop Development with C++).

To edit your C/C++ program:

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

In the New Project window:

1. Under Project types, select ConsoleApp, press Next.
2. Name your project and specify a location for your project directory.
3. Click 'Create'

   Once the project has been created, in the window on the left hand side you should see five folders:

   References
   External Dependencies
   Header Files
   Resource Files
   Source Files

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. If you want to run your code directly (without debugger), press CTRL-F5.

To add a new source file:

1. Right-click on Source Files and Select Add-> New Item
2. Select Code, and give the file a name. The default here will be a file with a *.cpp extension (for a C++ file).

This is an example of the C++ source file

```
/*
 * fun.cpp - funny C++ program skeleton
 *
 * Written by Jarkko Vuori/Metropolia
 */

#include <iostream>
using namespace std;
   .
   .
   .

int main() {
   .
   .
   .
   return 0;
}
```

All the input and output of our exercises is from the user keyboard and to the console screen. Therefore, you can use standard `cout` and `cin` objects 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.