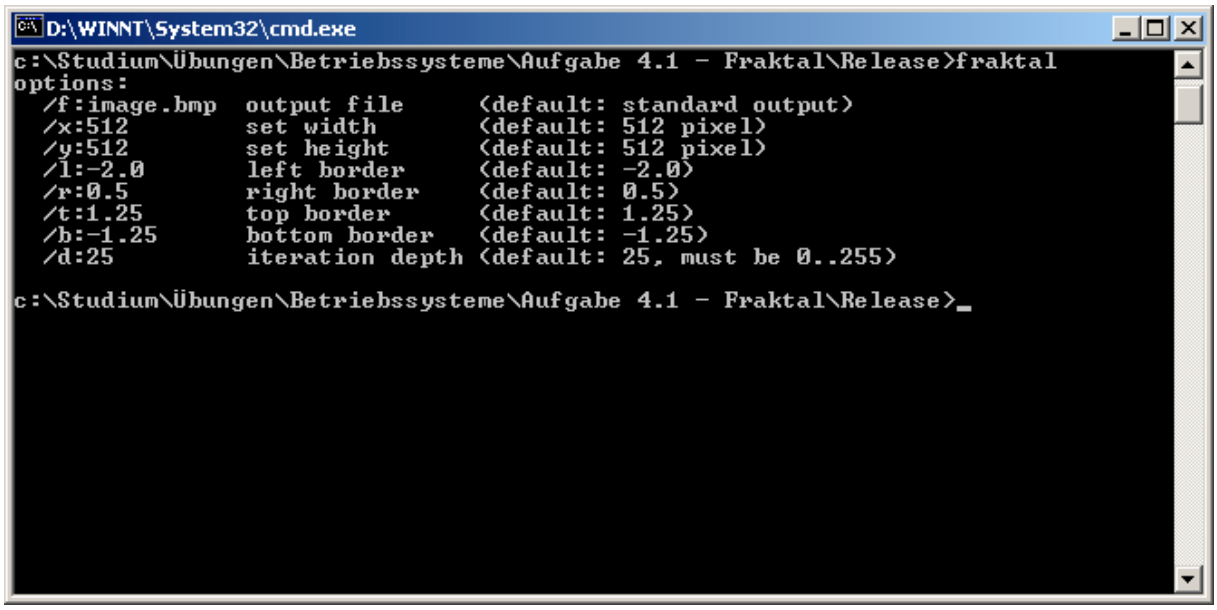
Aufgabe 4.1

Entwerfen, implementieren und führen Sie ein einfaches Programm vor, das fraktale Bilder berechnet (insbesondere die Mandelbrot-Menge). Das Programm soll eine Fläche mit vorgegebenen Ausmaßen füllen (z.B. 100x100 Pixel, die dem Bereich (-1,-i) bis (1, i) auf der komplexen Zahlenebene entsprechen), wobei die jeweilige Farbe von der Iterationstiefe an diesem Punkt abhängt. Wenn eine maximale Iterationstiefe überschritten wurde, z.B. 256, dann wird der entsprechende Punkt schwarz eingefärbt.
Das Programm soll die Daten im BMP-Format an die Standardausgabe schicken, wo sie mit einem normalen Bildbetrachter angeschaut werden können.

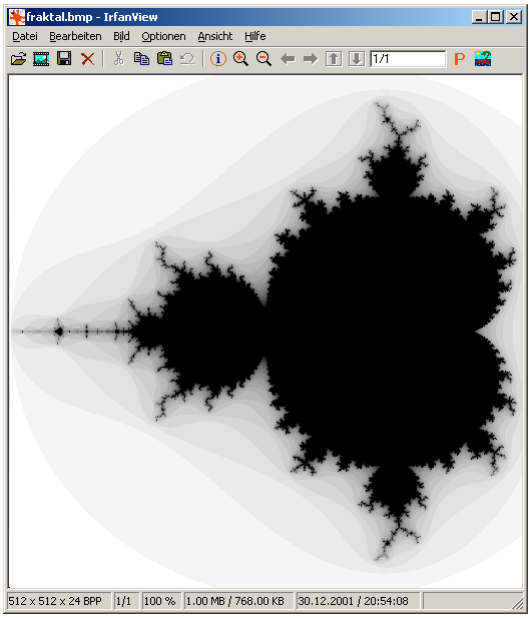
Das Programm zeigt, wenn es ohne Kommandozeilen-Parameter gestartet wird, Bedienungshinweise an:



Wird ein Parameter nicht angegeben, so kommt der Standardwert zum Zuge. Ein Aufruf über

```
fraktal /f:fraktal.bmp
```

generiert das folgende Bild:



Im Skript ist die Iterationsroutine zur Bestimmung eines einzelnen Punktes der Mandelbrot-Menge bereits enthalten und kann fast unverändert übernommen werden:

```
unsigned int nDepth = 0;
double wx = dX;
double wy = dY;

// calculate point's color
do
{
    const double old_wx = wx;
    nDepth++;

    wx = wx*wx - wy*wy + dX;
    wy = 2*old_wx * wy + dY;
} while (wx*wx + wy*wy <= 4 && nDepth <= nMaxDepth);
```

Die sich ergebende Iterationstiefe wird als Graustufenwert betrachtet, der in den Bereich von 0 bis 255 abgebildet wird. Letzteres ist notwendig, um unabhängig von der maximalen Iterationstiefe immer reines Weiß und reines Schwarz darstellen zu *können*, die Feinheit der Abstufungen variiert dann nur noch:

```
// convert iteration depth to grey and save it
nDepth = int((nMaxDepth-nDepth) * 255.0 / nMaxDepth);
*pImageCursor = RGB(nDepth, nDepth, nDepth);
```

Um alle Punkte durchlaufen zu können, müssen deren Koordinaten in Beziehung zu deren Pixelposition gebracht werden. Der entsprechende Code findet sich am Schleifenanfang und baut auf dem Dreisatz auf:

```
for (unsigned int nRow = 0; nRow < nHeight; nRow++)
{
    for (unsigned int nColumn = 0; nColumn < nWidth; nColumn++)
    {
        // get current coordinates
        const double dX = nColumn/(nWidth -1.0) * (dRight - dLeft  ) + dLeft;
        const double dY = nRow   /(nHeight-1.0) * (dTop   - dBottom) + dBottom;
        ...
    }
}
```

Alle bisher besprochenen Zeilen sind Bestandteil von `Calculate`. Interessant ist die Erzeugung einer BMp-Datei. Dabei hilft die Win32-API mit vordefinierten Strukturen weiter, in der MSDN finden sich auch (leider schwer verständliche) Beispiele im Bereich „GDI“. Im Endeffekt sind die Strukturen `BITMAPINFO` und `BITMAPFILEHEADER` mit den Bildparametern zu füllen, ich beschränke mich dabei auf unkomprimierte Truecolor-Dateien. Mit den vertrauten Win32-Funktionen `CreateFile` und `WriteFile` werden die Daten auf die Standardausgabe oder in eine Datei geschrieben.

```
// create new file, overwrite if neccessary
HANDLE hFile;
if (strFilename != NULL)
    hFile = CreateFile(strFilename, GENERIC_WRITE, 0, NULL,
                                    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
else
    hFile = GetStdHandle(STD_OUTPUT_HANDLE);

// failed to create file ?
if (hFile == INVALID_HANDLE_VALUE)
    return false;


DWORD dwBytesWritten;
// write file header
if (!WriteFile(hFile, &bmpHeader, sizeof(bmpHeader), &dwBytesWritten, NULL))
    return false;
// write info header
if (!WriteFile(hFile, &bmpInfo,   sizeof(bmpInfo),   &dwBytesWritten, NULL))
    return false;

// write image
if (!WriteFile(hFile, arImage, bmpInfo.bmiHeader.biSizeImage, &dwBytesWritten, NULL))
    return false;
```

In der `main`-Routine wird die Kommandozeile geparst, was zwar platzaufwändig, aber schnell zu programmieren ist. Anschließend wird ein Feld `arImage` angelegt, das lautet Einträge vom Typ `COLORREF` enthält. `COLORREF` ist die Win32-Repräsentation eines Truecolor-Farbwertes und besteht intern aus 32 Bit. In der Iterationsroutine wird das Makro `RGB` benutzt, welches aus den drei einzelnen Farbanteilen ein `COLORREF` generiert.

Ich habe die Absicht, den Wanderpokal für die kleinste EXE-Datei bei mir zu behalten und aus diesem Grunde wieder an den Compiler- und Linker-Einstellungen solange herumgespielt, bis das Resultat niedliche 4096 Bytes waren. An diesem Wert mussten selbst EXE-Kompressoren aufgeben, ich vermute, dass ein kleineres Programm bei gleicher Funktionalität nur mit Assembler zu erreichen ist. Nachdem ich bei der letzten Hausaufgabe Matt Pietriks LIBCTINY benutzte, wollte ich mir diesmal eine Alternative suchen, die auf externen Quellcode verzichtet. Fündig wurde ich bei der `pragma`-Direktive. Sie erlaubt mir, sowohl das Segment-Alignment zu verkleinern, als auch die Standard-Bibliothek *nicht* statisch zu linken, sondern die auf Windows-2000-Systemen vorhandene MSVCRT.DLL zu verwenden:

```
// no 4k alignment
#pragma comment(linker, "/OPT:NOWIN98")
// use system‘s dll instead of statically linking da code
#pragma comment(linker, "/NODEFAULTLIB:libc.lib")
#pragma comment(linker, "/NODEFAULTLIB:libcd.lib")
#pragma comment(lib, "msvcrt.lib")
#pragma comment(lib, "gdi32.lib")
// watermark
#pragma comment(exestr, "(C)2001-2002 by Stephan Brumme")
```

Quelltext

Der gesamte Code steckt in fraktal.cpp.

fraktal.cpp:

```
//////////////////////////////////////////////////////////
// Betriebssysteme I - Windows 2000
// Assignment 4.1: A simple fractal generator
//
// author:          Stephan Brumme
// last changes:    December 30, 2001


// include Win32-API
#include <windows.h>
// C library
#include <stdio.h>


// let‘s cut down the exe‘s size !!!
// no 4k alignment
#pragma comment(linker, "/OPT:NOWIN98")
// use system‘s dll instead of statically linking da code
#pragma comment(linker, "/NODEFAULTLIB:libc.lib")
#pragma comment(linker, "/NODEFAULTLIB:libcd.lib")
#pragma comment(lib, "msvcrt.lib")
#pragma comment(lib, "gdi32.lib")
// watermark
#pragma comment(exestr, "(C)2001-2002 by Stephan Brumme")



// save image as 24-bit-BMP
bool WriteBMP(const char* strFilename, unsigned int nWidth, unsigned int nHeight, const
COLORREF arImage[])
{
    // in the beginning there was nothing - except a bitmap info header !

    // code taken from MSDN (see GDI: "Storing an Image"), simplified
    BITMAPINFO bmpInfo;
    bmpInfo.bmiHeader.biSize          = sizeof(BITMAPINFOHEADER);
    bmpInfo.bmiHeader.biWidth         = nWidth;
    bmpInfo.bmiHeader.biHeight        = nHeight; // bottom-up DIB !
    bmpInfo.bmiHeader.biPlanes        = 1;
```

```
    bmpInfo.bmiHeader.biBitCount      = 32; // bit depth: RGB+reserved
    bmpInfo.bmiHeader.biClrUsed       = 0; // true color !
    bmpInfo.bmiHeader.biClrImportant  = 0;
    bmpInfo.bmiHeader.biCompression   = BI_RGB;
    bmpInfo.bmiHeader.biSizeImage     = ((nWidth*32+31) & ~31) / 8 * nHeight;
    bmpInfo.bmiHeader.biXPelsPerMeter =
    bmpInfo.bmiHeader.biYPelsPerMeter = 3780; // 96 dpi

    // create header out of the info structure
    BITMAPFILEHEADER bmpHeader;
    // magic bytes "BM"
    bmpHeader.bfType = 0x4d42;
    // offset to the raw image data
    bmpHeader.bfOffBits   = sizeof(BITMAPFILEHEADER) + bmpInfo.bmiHeader.biSize; +
    // size of the entire file
    bmpHeader.bfSize = bmpHeader.bfOffBits + bmpInfo.bmiHeader.biSizeImage;
    // reserved fields
    bmpHeader.bfReserved1 = 0;
    bmpHeader.bfReserved2 = 0;


    // create new file, overwrite if neccessary
    HANDLE hFile;
    if (strFilename != NULL)
        hFile = CreateFile(strFilename, GENERIC_WRITE, 0, NULL,
                                        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    else
        hFile = GetStdHandle(STD_OUTPUT_HANDLE);

    // failed to create file ?
    if (hFile == INVALID_HANDLE_VALUE)
        return false;


    DWORD dwBytesWritten;
    // write file header
    if (!WriteFile(hFile, &bmpHeader, sizeof(bmpHeader), &dwBytesWritten, NULL))
        return false;
    // write info header
    if (!WriteFile(hFile, &bmpInfo,   sizeof(bmpInfo),   &dwBytesWritten, NULL))
        return false;

    // write image
    if (!WriteFile(hFile, arImage, bmpInfo.bmiHeader.biSizeImage, &dwBytesWritten, NULL))
        return false;

    // close file and: DONE !
    CloseHandle(hFile);
    return true;
}


// generate mandelbrot image
void Calculate(COLORREF arImage[], unsigned int nWidth, unsigned int nHeight,
               double dLeft, double dRight, double dTop, double dBottom, unsigned int
nMaxDepth)
{
    // walk through the image
    COLORREF* pImageCursor = arImage;

    // process all points
    // remind that BMPs are built bottom-up !
    for (unsigned int nRow = 0; nRow < nHeight; nRow++)
    {
        for (unsigned int nColumn = 0; nColumn < nWidth; nColumn++)
        {
            // get cuurent coordinates
            const double dX = nColumn/(nWidth -1.0) * (dRight - dLeft  ) + dLeft;
            const double dY = nRow   /(nHeight-1.0) * (dTop   - dBottom) + dBottom;

            // initialize
            unsigned int nDepth = 0;
            double wx = dX;
            double wy = dY;

            // calculate point's color
```

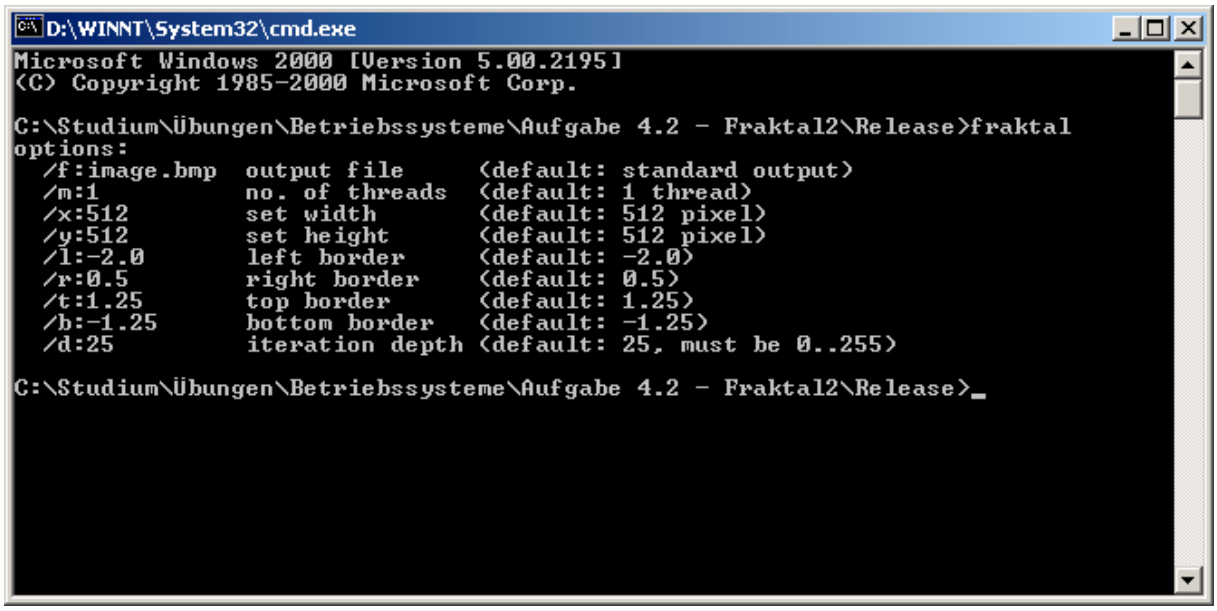
```
            do
            {
                const double old_wx = wx;
                nDepth++;

                wx = wx*wx - wy*wy + dX;
                wy = 2*old_wx * wy + dY;
            } while (wx*wx + wy*wy <= 4 && nDepth <= nMaxDepth);

            // we counted once too much
            nDepth--;

            // convert iteration depth to grey and save it
            nDepth = int((nMaxDepth-nDepth) * 255.0 / nMaxDepth);
            *pImageCursor = RGB(nDepth, nDepth, nDepth);
            pImageCursor++;
        }
    }
}


// here we go !
int main(int argc, char* argv[])
{
    // default fractal parameters
    unsigned int nWidth  = 512;
    unsigned int nHeight = 512;
    double dLeft   =  -2.0;
    double dRight  =   0.5;
    double dTop    =   1.25;
    double dBottom =  -1.25;
    unsigned int nMaxDepth = 25;
    const char* strFilename = NULL;

    // no command line parameters or help requested ?
    if (argc == 1)
    {
        printf("options:\n"
               "  /f:image.bmp  output file     (default: standard output)\n"
               "  /x:512        set width       (default: 512 pixel)\n"
               "  /y:512        set height      (default: 512 pixel)\n"
               "  /l:-2.0       left border     (default: -2.0)\n"
               "  /r:0.5        right border    (default: 0.5)\n"
               "  /t:1.25       top border      (default: 1.25)\n"
               "  /b:-1.25      bottom border   (default: -1.25)\n"
               "  /d:25         iteration depth (default: 25, must be 0..255)\n"
               );

        return 1;
    }

    // parse command line
    int nParameter = 1;
    while (nParameter < argc)
    {
        // parse one parameter
        const char* strOption = argv[nParameter];

        // simple verification
        if (strOption == NULL      ||
            strlen(strOption) < 4  ||
            strOption[0] != '/'    ||
            strOption[2] != ':')
        {
            printf("unknown option '%s'\n", strOption);
            return 1;
        }

        // value the parameter should contain
        const char* strValue = &(strOption[3]);

        // set parameters
        switch(strOption[1])
        {
        // filename
```

```
        case 'f':
        case 'F': strFilename = strValue; break;

        // image size
        case 'x':
        case 'X': nWidth  = atoi(strValue); break;
        case 'y':
        case 'Y': nHeight = atoi(strValue); break;

        // borders
        case 'l':
        case 'L': dLeft   = atof(strValue); break;
        case 'r':
        case 'R': dRight  = atof(strValue); break;
        case 't':
        case 'T': dTop    = atof(strValue); break;
        case 'b':
        case 'B': dBottom = atof(strValue); break;

        // iteration depth
        case 'd':
        case 'D': nMaxDepth = atoi(strValue); break;

        // error
        default:  printf("unknown option '%s'\n", strOption);
                  return 1;
        }

        // next one
        nParameter++;
    }


    // allocate memory
    COLORREF* arImage = new COLORREF[nWidth*nHeight];
    // create fractal
    Calculate(arImage, nWidth, nHeight, dLeft, dRight, dTop, dBottom, nMaxDepth);
    // write to file
    WriteBMP(strFilename, nWidth, nHeight, arImage);
    // free memory
    delete arImage;

    // done
    return 0;
}
```

Aufgabe 4.2

Verändern Sie Ihr Programm aus Aufgabe 4.1 derart, dass mehrere Threads disjunkte Bereiche (Linien) des Bildes berechnen. Die Anzahl der zu verwendenen Threads muss als Kommandozeilen-Parameter einstellbar sein. Ihr Programm sollte einen speziellen Master Thread besitzen, der die Arbeit verteilt und auf die Fertigstellung des Fraktals durch die Worker Threads wartet, bevor das Programm beendet wird.

Die Bedienung ist nahezu unverändert geblieben, es ist lediglich die Möglichkeit hinzugekommen, die Anzahl der zu benutzenden Threads mit dem Parameter `/m` festzulegen:



Ein Großteil des bereits in Aufgabe 4.1 entwickelten Codes konnte, leicht modifiziert, wiederverwendet werden. Dazu zählt u.a. die Berechnung der Mandelbrotmenge in `Calculate` und das Schreiben einer BMP-Datei mittels `WriteBMP`.

Eine erste Schwierigkeit stellt die Übergabe der Parameter an die einzelnen Threads dar. Ich entschied mich dazu, eine Struktur `TFractalParameter` einzuführen, die diese Aufgabe übernimmt und der Argumentliste von `Calculate` aus Aufgabe 4.1 entspricht:

```
// parameter that are passed to each thread
struct TFractalParameters
{
    COLORREF* arImage;
    unsigned int nWidth;
    unsigned int nHeight;
    double dLeft;
    double dRight;
    double dTop;
    double dBottom;
    unsigned int nMaxDepth;
};
```

Gemäß Konvention muss sich der Ausführungspfad eines Threads in einer Funktion der Form

```
DWORD WINAPI ThreadProc(LPVOID lpParameter);
```

befinden. `Calculate` wird dementsprechend umformuliert zu

```
DWORD WINAPI Calculate(void *pParameter);
```

`pParameter` wird dabei als Zeiger auf `TFractalParameter` benutzt.

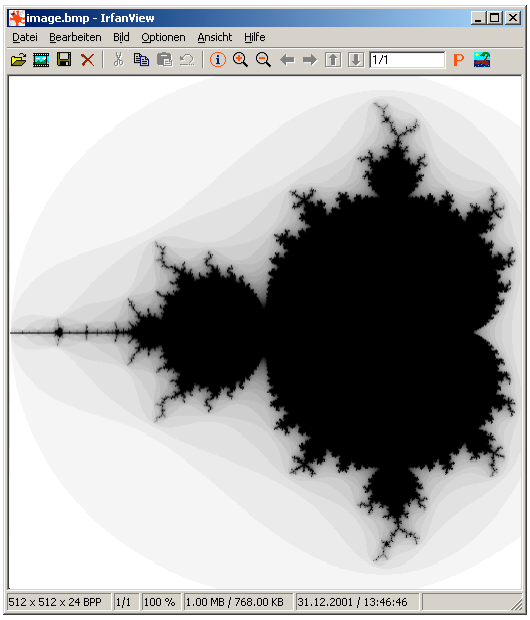
Die Erzeugung eines Threads übernimmt `CreateThread`. Zuvor werden der zu berechnende Bildausschnitt bestimmt:

```
    // store thread handles
    HANDLE* arhThread = new HANDLE[nThreads];
    DWORD   dwThreadID;

    // thread parameters
    TFractalParameters* param = new TFractalParameters[nThreads];
    const double dHeightDelta = (dTop - dBottom)/nThreads;

    // launch threads
    for (unsigned int i=0; i<nThreads; i++)
    {
        // pixel area
        param[i].nWidth    = nWidth;
        param[i].nHeight   = nHeight*(i+1)/nThreads - nHeight*i/nThreads;
        // image storage, take offset into account
        param[i].arImage   = arImage;
        param[i].arImage  += (nHeight-nHeight*i/nThreads-param[i].nHeight)*nWidth;
        // mandelbrot area
        param[i].dLeft     = dLeft;
        param[i].dRight    = dRight;
        param[i].dTop      = dTop-i*dHeightDelta;
        param[i].dBottom   = param[i].dTop-
dHeightDelta*param[i].nHeight/(nHeight/(double)nThreads);
        // iteration depth
        param[i].nMaxDepth = nMaxDepth;

        // start thread
        arhThread[i] = CreateThread(NULL, 0, Calculate, &(param[i]), 0, &dwThreadID);
        if (strFilename != NULL)
            printf("starting thread %d (ID: %d)\n", i+1, dwThreadID);
    }
```

Nachdem alle Threads gestartet wurden, muss der Hauptthread warten, bis alle ihre Arbeit erledigt haben, erst danach kann das Ergebnis auf den Datenträger geschrieben werden:

```
    // wait for threads to finish
    WaitForMultipleObjects(nThreads, arhThread, true, INFINITE);
```

Das resultierende Bild gleicht dem aus Aufgabe 4.1:



Quelltext

Erneut findet sich der gesamte Code in einer einzigen Datei:

fraktal.cpp

```
//////////////////////////////////////////////////////
// Betriebssysteme I - Windows 2000
// Assignment 4.2: A multi-threaded fractal generator
//
// author:          Stephan Brumme
// last changes:    December 31, 2001


// include Win32-API
#include <windows.h>
// C library
#include <stdio.h>


// let's cut down the exe's size !!!
// no 4k alignment
#pragma comment(linker, "/OPT:NOWIN98")
// use system's dll instead of statically linking da code
#pragma comment(linker, "/NODEFAULTLIB:libc.lib")
#pragma comment(linker, "/NODEFAULTLIB:libcd.lib")
#pragma comment(lib, "msvcrt.lib")
#pragma comment(lib, "gdi32.lib")
// watermark
#pragma comment(exestr, "(C)2001-2002 by Stephan Brumme")


// parameter that are passed to each thread
struct TFractalParameters
{
    COLORREF* arImage;
    unsigned int nWidth;
    unsigned int nHeight;
    double dLeft;
    double dRight;
    double dTop;
    double dBottom;
    unsigned int nMaxDepth;
};



// save image as 24-bit-BMP
bool WriteBMP(const char* strFilename, unsigned int nWidth, unsigned int nHeight, const
COLORREF arImage[])
{
    // in the beginning there was nothing - except a bitmap info header !

    // code taken from MSDN (see GDI: "Storing an Image"), simplified
    BITMAPINFO bmpInfo;
    bmpInfo.bmiHeader.biSize          = sizeof(BITMAPINFOHEADER);
    bmpInfo.bmiHeader.biWidth         = nWidth;
    bmpInfo.bmiHeader.biHeight        = nHeight; // bottom-up DIB !
    bmpInfo.bmiHeader.biPlanes        = 1;
    bmpInfo.bmiHeader.biBitCount      = 32; // bit depth: RGB+reserved
    bmpInfo.bmiHeader.biClrUsed       = 0; // true color !
    bmpInfo.bmiHeader.biClrImportant  = 0;
    bmpInfo.bmiHeader.biCompression   = BI_RGB;
    bmpInfo.bmiHeader.biSizeImage     = ((nWidth*32+31) & ~31) / 8 * nHeight;
    bmpInfo.bmiHeader.biXPelsPerMeter =
    bmpInfo.bmiHeader.biYPelsPerMeter = 3780; // 96 dpi

    // create header out of the info structure
    BITMAPFILEHEADER bmpHeader;
    // magic bytes "BM"
    bmpHeader.bfType = 0x4d42;
    // offset to the raw image data
    bmpHeader.bfOffBits   = sizeof(BITMAPFILEHEADER) + bmpInfo.bmiHeader.biSize; +
```

```
    // size of the entire file
    bmpHeader.bfSize = bmpHeader.bfOffBits + bmpInfo.bmiHeader.biSizeImage;
    // reserved fields
    bmpHeader.bfReserved1 = 0;
    bmpHeader.bfReserved2 = 0;


    // create new file, overwrite if neccessary
    HANDLE hFile;
    if (strFilename != NULL)
        hFile = CreateFile(strFilename, GENERIC_WRITE, 0, NULL,
                                        CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
    else
        hFile = GetStdHandle(STD_OUTPUT_HANDLE);

    // failed to create file ?
    if (hFile == INVALID_HANDLE_VALUE)
        return false;


    DWORD dwBytesWritten;
    // write file header
    if (!WriteFile(hFile, &bmpHeader, sizeof(bmpHeader), &dwBytesWritten, NULL))
        return false;
    // write info header
    if (!WriteFile(hFile, &bmpInfo,   sizeof(bmpInfo),   &dwBytesWritten, NULL))
        return false;

    // write image
    if (!WriteFile(hFile, arImage, bmpInfo.bmiHeader.biSizeImage, &dwBytesWritten, NULL))
        return false;

    // close file and: DONE !
    CloseHandle(hFile);
    return true;
}


// generate mandelbrot image
DWORD WINAPI Calculate(void *pParameter)
{
    // cast
    TFractalParameters *param = (TFractalParameters*) pParameter;

    // walk through the image
    COLORREF* pImageCursor = param->arImage;

    // process all points
    // remind that BMPs are built bottom-up !
    for (unsigned int nRow = 0; nRow < param->nHeight; nRow++)
    {
        for (unsigned int nColumn = 0; nColumn < param->nWidth; nColumn++)
        {
            // get current coordinates
            const double dX = nColumn/(param->nWidth -1.0) * (param->dRight - param->dLeft  )
+ param->dLeft;
            const double dY = nRow   /(param->nHeight-1.0) * (param->dTop   - param->dBottom)
+ param->dBottom;

            // initialize
            unsigned int nDepth = 0;
            double wx = dX;
            double wy = dY;

            // calculate point's color
            do
            {
                const double old_wx = wx;
                nDepth++;

                wx = wx*wx - wy*wy + dX;
                wy = 2*old_wx * wy + dY;
            } while (wx*wx + wy*wy <= 4 && nDepth <= param->nMaxDepth);

            // we counted once too much
            nDepth--;
```

```
            // convert iteration depth to grey and save it
            nDepth = int((param->nMaxDepth-nDepth) * 255.0 / param->nMaxDepth);
            *pImageCursor = RGB(nDepth, nDepth, nDepth);
            pImageCursor++;
        }
    }

    return 0;
}



// here we go !
int main(int argc, char* argv[])
{
    // default fractal parameters
    unsigned int nWidth  = 512;
    unsigned int nHeight = 512;
    double dLeft   =  -2.0;
    double dRight  =   0.5;
    double dTop    =   1.25;
    double dBottom =  -1.25;
    unsigned int nMaxDepth = 25;
    const char* strFilename = NULL;
    unsigned int nThreads = 1;

    // no command line parameters or help requested ?
    if (argc == 1)
    {
        printf("options:\n"
               "  /f:image.bmp  output file     (default: standard output)\n"
               "  /m:1          no. of threads  (default: 1 thread)\n"
               "  /x:512        set width       (default: 512 pixel)\n"
               "  /y:512        set height      (default: 512 pixel)\n"
               "  /l:-2.0       left border     (default: -2.0)\n"
               "  /r:0.5        right border    (default: 0.5)\n"
               "  /t:1.25       top border      (default: 1.25)\n"
               "  /b:-1.25      bottom border   (default: -1.25)\n"
               "  /d:25         iteration depth (default: 25, must be 0..255)\n"
               );

        return 1;
    }

    // parse command line
    int nParameter = 1;
    while (nParameter < argc)
    {
        // parse one parameter
        const char* strOption = argv[nParameter];

        // simple verification
        if (strOption == NULL      ||
            strlen(strOption) < 4  ||
            strOption[0] != '/'    ||
            strOption[2] != ':')
        {
            printf("unknown option '%s'\n", strOption);
            return 1;
        }

        // value the parameter should contain
        const char* strValue = &(strOption[3]);

        // set parameters
        switch(strOption[1])
        {
        // filename
        case 'f':
        case 'F': strFilename = strValue; break;

        // image size
        case 'x':
        case 'X': nWidth  = atoi(strValue); break;
        case 'y':
```

```
        case 'Y': nHeight = atoi(strValue); break;

        // borders
        case 'l':
        case 'L': dLeft   = atof(strValue); break;
        case 'r':
        case 'R': dRight  = atof(strValue); break;
        case 't':
        case 'T': dTop    = atof(strValue); break;
        case 'b':
        case 'B': dBottom = atof(strValue); break;

        // iteration depth
        case 'd':
        case 'D': nMaxDepth = atoi(strValue); break;

        // threads
        case 'm':
        case 'M': nThreads = atoi(strValue); break;

        // error
        default:  printf("unknown option '%s'\n", strOption);
                  return 1;
        }

        // next one
        nParameter++;
    }


    // allocate memory
    COLORREF* arImage = new COLORREF[nWidth*nHeight];

    // store thread handles
    HANDLE* arhThread = new HANDLE[nThreads];
    DWORD   dwThreadID;

    // thread parameters
    TFractalParameters* param = new TFractalParameters[nThreads];
    const double dHeightDelta = (dTop - dBottom)/nThreads;

    // launch threads
    for (unsigned int i=0; i<nThreads; i++)
    {
        // pixel area
        param[i].nWidth    = nWidth;
        param[i].nHeight   = nHeight*(i+1)/nThreads - nHeight*i/nThreads;
        // image storage, take offset into account
        param[i].arImage   = arImage;
        param[i].arImage  += (nHeight-nHeight*i/nThreads-param[i].nHeight)*nWidth;
        // mandelbrot area
        param[i].dLeft     = dLeft;
        param[i].dRight    = dRight;
        param[i].dTop      = dTop-i*dHeightDelta;
        param[i].dBottom   = param[i].dTop-
dHeightDelta*param[i].nHeight/(nHeight/(double)nThreads);
        // iteration depth
        param[i].nMaxDepth = nMaxDepth;

        // start thread
        arhThread[i] = CreateThread(NULL, 0, Calculate, &(param[i]), 0, &dwThreadID);
        if (strFilename != NULL)
            printf("starting thread %d (ID: %d)\n", i+1, dwThreadID);
    }

    // wait for threads to finish
    WaitForMultipleObjects(nThreads, arhThread, true, INFINITE);

    // write to file
    if (strFilename != NULL)
        printf("writing to %s ... ", strFilename);
    if (!WriteBMP(strFilename, nWidth, nHeight, arImage))
        printf("FAILURE !!!\n");

    // free memory
    delete arImage;
```

```
    for (i=0; i<nThreads; i++)
        CloseHandle(arhThread[i]);
    delete arhThread;

    // done
    return 0;
}
```

Aufgabe 4.3

Vergleichen Sie Konzepte und Eigenschaften von named pipes *unter Win32 und UNIX Sys V (siehe* `mkfifo()`*).*

Die *named pipes* dienen zur Interprozeß-Kommunikation. Sie stellen eine Verbindung zwischen Prozessen her, deren Speicherbereiche eigentlich durch das jeweilige Multitasking-Betriebssystem voreinander abgeschirmt sind. Unter Unix ist die Bezeichnung FIFO (first in, first out) geläufiger.

Windows 2000 erlaubt die Versendung sowohl auf Byte-orientierte Art und Weise als auch in Nachrichtenform. Der Unterschied liegt darin, dass Nachrichten stets eine Einheit bilden und vom Sender und vom Empfänger jeweils einen Funktionsaufruf verlangen. Bei Byte-basierter Versendung ist diese enge Kopplung nicht gegeben und daher im Betriebssystem einfacher zu implementieren. Unix Sys V beschränkt sich deshalb darauf.

Unter Windows 2000 können Pipes uni- und bidirektional funktionieren, Sys V erlaubt nur ersteres. Es ist auch möglich, *named pipes* mit der Win32-API netzwerkwert zu nutzen, wozu Sys V nicht in der Lage ist.