Praktikumstermin Nr. 11, INF
Dynamische Datenstrukturen - `make`, Game of Life

Die erste Aufgabe dieses Praktikums könnte auch auf Nicht-Windows-Rechnern durchgeführt werden, z.B. auf ihrem eigenen Linux oder MacOS Rechner. Dazu müssten Sie aber selbst wissen bzw. herausfinden, wie sie make auf ihrem Rechner installieren können (bzw. ob es schon installiert ist), wie auf ihrem Rechner die Befehle zum Starten des C++ Compilers und des Linkers von der Kommandozeile heißen etc. Im Zweifelsfalle (d.h. wenn dabei Probleme auftreten) müssen Sie die komplette erste Aufgabe auf einem Terminalrechner der FH durchführen. Ein erfolgreiches Absolvieren aller Aufgaben des Praktikums ist Pflicht, „Probleme mit dem eigenen Rechner" können wir (wie auch in den vorherigen Wochen des GIP Praktikums) nicht als Begründung für das Nicht-Absolvieren einer Aufgabe akzeptieren …

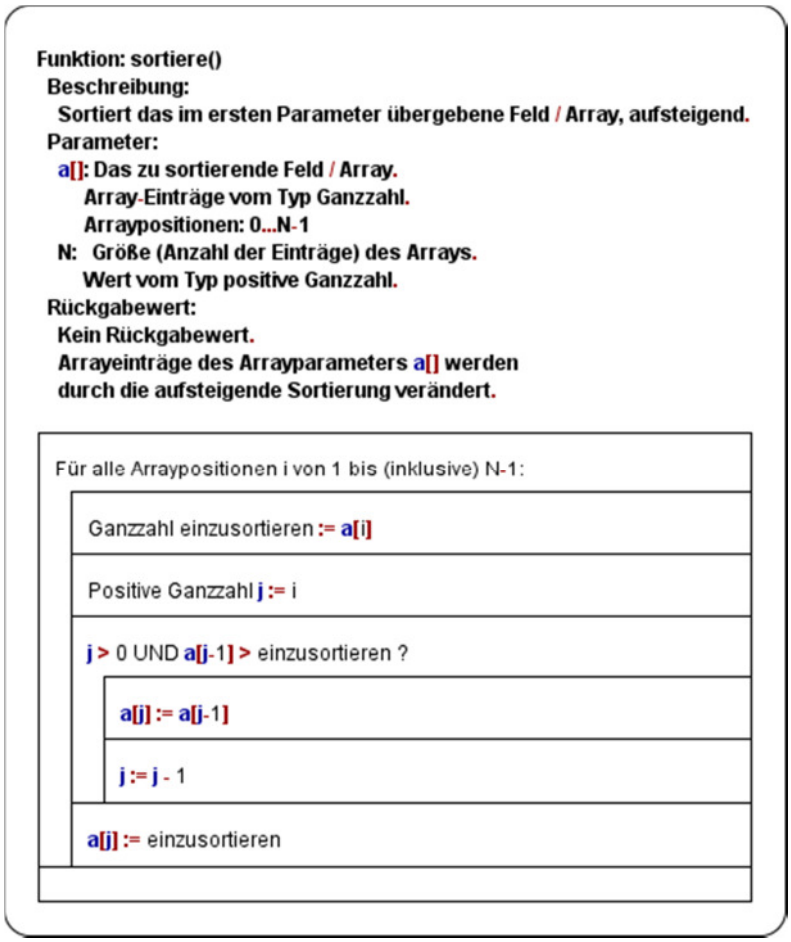
Aufgabe INF-11.01a: Sortierprogramm

Legen Sie in Visual Studio ein neues Projekt an (*Allgemein*, *Leeres Projekt*). Kopieren Sie das folgende Hauptprogramm in eine Datei `main.cpp`.

```
// Datei: main.cpp

#include <iostream>
using namespace std;

#include "sort.h"

int main()
{
	const int N = 10;
	int a[] = { 9, 3, 5, 2, 8, 6, 4, 3, 7, 8 };

	// Ausgabe des "Vorher" Zustands des Arrays...
	cout << "Vorher: ";
	for (int k = 0; k < N; k++)
		cout << a[k] << (k < N - 1 ? ", " : "\n");

	// Aufruf der sortiere() Funktion...
	sortiere(a, N);
```

```
	// Ausgabe des "Nachher" Zustands des Arrays...
	cout << "Nachher: ";
	for (int k = 0; k < N; k++)
		cout << a[k] << (k < N - 1 ? ", " : "\n");

	system("PAUSE");
	return 0;
}
```

Programmieren Sie in zwei Dateien `sort.cpp` und `sort.h` eine Funktion `sortiere()` gemäß folgendem Struktogramm:



Aufgabe INF-11.01b: make installieren & Makefile

Öffnen Sie das Verzeichnis mit den Dateien dieses Projekts im Windows Explorer: Klicken Sie dazu in Visual Studio mit der rechten Maustaste auf das gerade angelegte Projekt (im „Projektbaum" links) und wählen Sie

dann den relativ weit unten stehenden Menüeintrag `Ordner im Datei-Explorer öffnen`.

Kopieren Sie die Dateien `make.exe`, `libiconv2.dll` und `libintl3.dll` aus Ilias in dieses (im Datei-Explorer angezeigte) Verzeichnis.
Achtung: Ilias benennt aus Sicherheitsgründen die drei hochgeladenen Dateien des `make` Programms um. Sie müssen die Dateien nach dem Herunterladen wieder umbenennen in ihre ursprünglichen Namen:

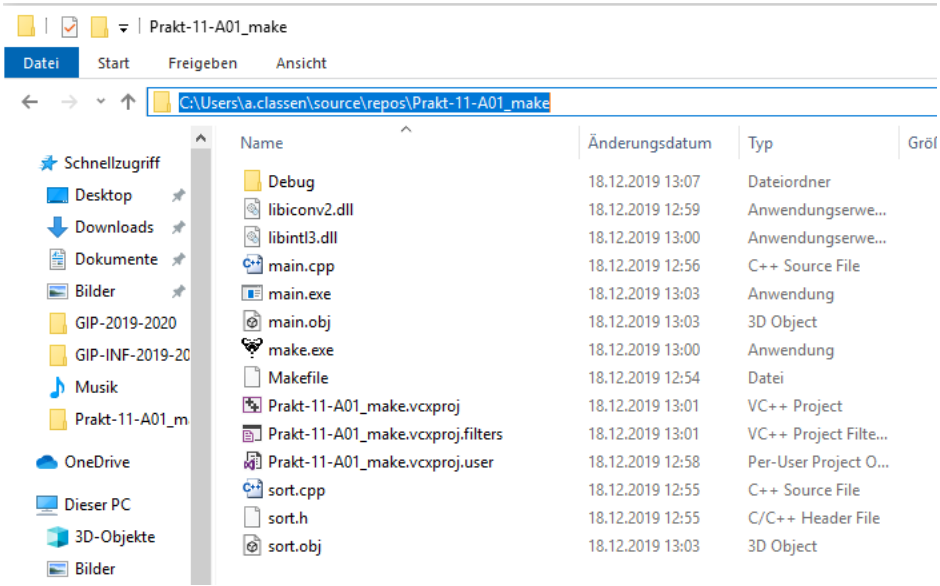
```
makeexe.sec       => make.exe
libiconv2dll.sec  => libiconv2.dll
libintl3dll.sec   => libintl3.dll
```

Diese Umbenennung können Sie nach dem Herunterladen z.B. im gerade geöffneten Windows Datei-Explorer vornehmen.

Öffnen Sie die `Eingabeaufforderung`: Geben Sie dazu im Windows-Menü `Start` (ganz unten links auf dem Bildschirm) die Zeichenfolge `cmd` ein und klicken Sie auf den oben im Menü erscheinenden Eintrag zur `Eingabeaufforderung`.
Geben Sie in diese Eingabeaufforderung folgendes Kommando ein:

```
call "C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise\VC\Auxiliary\Build\vcvarsamd64_x86.bat"
```

Wechseln Sie in der Eingabeaufforderung in das Verzeichnis, in dem sich die Dateien ihres Projekts befinden. Das ist der Pfad, den Sie im eben geöffneten Windows Datei-Explorer oben eingeblendet sehen, wenn Sie in den weißen Bereich rechts neben dem Verzeichnisnamen klicken:



Der genaue Pad wird bei Ihnen anders lauten!

Der Verzeichniswechsel in der `Eingabeaufforderung` geschieht dann mittels des Befehls

```
cd /d "C:\Users\a.classen\source\repos\GIP-2019-2020\05_HalloWelt"
```

(… noch einmal: der Pad wird bei Ihnen anders lauten! Copy-Paste des obigen Befehls wird nicht funktionieren!)

Lassen Sie sich mittels des Befehls `dir` den Inhalt des Verzeichnisses ausgeben und prüfen Sie, dass die drei heruntergeladenen `make` Dateien auch wirklich in dem Verzeichnis existieren und auch richtig von Ihnen umbenannt wurden.
Prüfen Sie auch, dass sich ihre `sort.h` , `sort.cpp` und `main.cpp` Dateien ebenfalls in diesem Verzeichnis befinden (falls nicht, haben Sie in einem der vorigen Schritte einen Fehler gemacht).

Legen Sie nun eine leere Datei `Makefile` in diesem Verzeichnis an. Da der mit Windows mitgelieferte Standard-Editor `notepad` an jede neue Datei immer die Endung `.txt` anhängt, was wir bei einem `Makefile` auf jeden Fall verhindern müssen, müssen wir eine Datei mit Dateiname ohne Endung `.txt` erzwingen. Geben Sie dazu in der `cmd.exe` ein:

```
type nul > Makefile
```

Dies legt eine leere Datei an mit dem Namen `Makefile`. Wenn Sie sich jetzt mit `dir` den Verzeichnisinhalt anzeigen lassen, müssten Sie dort dann auch die Datei `Makefile` sehen …

Öffnen Sie nun die Datei `Makefile` im `notepad` Editor: Geben Sie dazu in `cmd.exe` den folgenden Befehl ein:

```
notepad Makefile
```

„Programmieren" Sie die Datei `Makefile` gemäß folgenden Anforderungen (siehe auch Vorlesungsunterlagen):

- Beachten Sie bei allen Regeln des Makefiles, dass die Kommandos der Regeln mittels des `TAB` (Tabulator) Zeichens eingerückt sein müssen. Einrückungen über Leerzeichen werden als Fehler behandelt und führen zur „`missing separator`" Fehlermeldung von `make`.

- Definieren Sie eine Regel für ein abstraktes Ziel `clean`. Dieses Ziel soll zur Löschung aller Objektcode-Dateien `*.obj` führen. Benutzen Sie zum Dateilöschen den Befehl `del`

- Das ausführbare Programm Ihres „Software-Projekts" in diesem Praktikumsversuch soll später `main.exe` heißen.
  Verwenden Sie in den Regeln des Makefiles an den Stellen, wo eigentlich `main.exe` stehen würde, die Variable `MAIN_EXECUTABLE`.

- Definieren Sie eine Regel für ein weiteres abstraktes Ziel `distclean`. Dieses Ziel soll vom abstrakten Ziel `clean` abhängen und zusätzlich zur Löschung des ausführbaren Programms `$MAIN_EXECUTABLE` führen.

- Definieren Sie eine Regel für ein weiteres abstraktes Ziel `all`.
  Dieses Ziel soll vom ausführbaren Programm `$MAIN_EXECUTABLE` abhängen. Es sind in dieser Regel dann keine Kommandos erforderlich.

- Definieren Sie in den ersten Zeilen des Makefiles folgende Variablen:
  ```
  CXX = cl
  CXXFLAGS = /EHsc /nologo
  MAIN_EXECUTABLE = main.exe
  ```

- Definieren Sie eine Regel für das ausführbare Programm `main.exe`. Dieses soll von den Objektcode-Dateien `main.obj` sowie `sort.obj` abhängen.
  Als Kommando soll der Linker `link` aufgerufen werden, unter Benutzung der Optionen `/nologo` und `/OUT:$@` sowie der automatischen Variablen `$^`.

- Definieren Sie eine Musterregel für `%.obj` Dateien. Diese sollen von den entsprechenden `%.cpp` und `%.h` Dateien abhängen.
  Der Compileraufruf soll erfolgen wie in der vorigen Regel. Allerdings muss dem Compiler bei dieser Regel zusätzlich die Option `/c` mitgegeben werden, damit er Objektcode-Dateien erzeugt (d.h. nur compiliert, aber den Linker nicht startet) und nicht versucht, eine ausführbare Datei zu erzeugen (d.h. Compiler *und Linker* ausführt).
  Verwenden Sie die automatische Variable `$<` (`$@` braucht nicht verwendet zu werden).

Aufgabe INF-11.01c: Compilieren und Ausführen

Compilieren Sie Ihr Programm durch Eingabe von `make all` oder `make`.

Korrigieren Sie eventuelle Programmierfehler bzw. Fehler im `Makefile`.

Testlauf *(Benutzereingaben sind zur Verdeutlichung unterstrichen)*:

```
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make> make distclean
del *.obj
del main.exe
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make> make distclean
del *.obj
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make\*.obj konnte
nicht gefunden werden
del main.exe
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make\main.exe konnte
nicht gefunden werden
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make> make all
cl /EHsc /c main.cpp
main.cpp
cl /EHsc /c sort.cpp
sort.cpp
link /nologo  main.obj sort.obj /OUT:main.exe
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make> make all
make: Nothing to be done for 'all'.
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make> main.exe
Vorher: 9, 3, 5, 2, 8, 6, 4, 3, 7, 8
Nachher: 2, 3, 3, 4, 5, 6, 7, 8, 8, 9
Drücken Sie eine beliebige Taste . . .
C:\Users\a.classen\Documents\Visual Studio\Projects\GIP\Praktikum-make> make distclean
del *.obj
del main.exe
```

Aufgabe INF-11.02: Conway's *Game of Life*

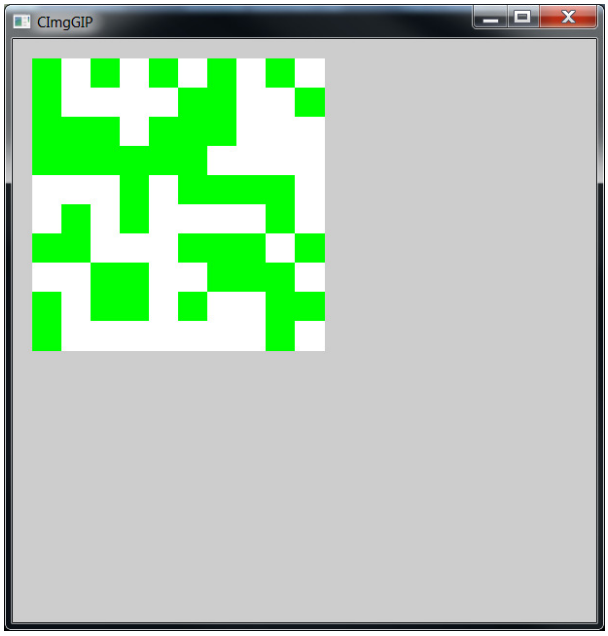
Diese Aufgabe können Sie wieder wie üblich komplett in Visual Studio durchführen, ohne die Benutzung von `make` …

Schreiben Sie unter Benutzung der *`CImg`* Library (mittels der Headerdatei `CImgGIP05.h`, die Sie schon aus einem vorherigen Praktikumsversuch kennen) ein C++ Programm, welches das Spiel *Game of Life* realisiert. Dieses stellt die zeitliche Entwicklung einer „Zellkolonie" dar.

https://de.wikipedia.org/wiki/Conways_Spiel_des_Lebens

Die Größe des Feldes sei als Konstante `grid_size` vorgegeben. Benutzen Sie ein zweidimensionales statisches Array zur Realisierung des Spielfeldes.

Das Spiel startet je nach Benutzereingabe entweder mit einer zufälligen Belegung des Feldes oder mit einer vorgegebenen Belegung. Benutzen Sie für die zufällige Belegung den Funktionsaufruf `gip_random(0,1);` um jeweils eine Zufallszahl „entweder 0 oder 1" zu erzielen.



Orientieren Sie sich an folgendem Programmgerüst (welches auch als Datei in Ilias vorliegt) für Ihre Lösung:

```
#include <iostream>
#define CIMGGIP_MAIN
#include "CImgGIP05.h"
using namespace std;
using namespace cimg_library;

const int grid_size = 10; // Anzahl an Kaestchen in x- und y-Richtung
const int box_size = 30;  // size der einzelnen Kaestchen (in Pixel)
const int border = 20;   // Rand links und oben bis zu den ersten Kaestchen (in
Pixel)

// Prototyp der Funktionen zum Vorbelegen des Grids ...
void grid_init(bool grid[][grid_size]);

int main()
{
    bool grid[grid_size][grid_size] = { 0 };
    bool next_grid[grid_size][grid_size] = { 0 };

    // Erstes Grid vorbelegen ...
```

```
    grid_init(grid);

    while (gip_window_not_closed())
    {
        // Spielfeld anzeigen ...
        // gip_stop_updates(); // ... schaltet das Neuzeichnen nach
                               //     jeder Bildschirmänderung aus

        // TO DO

        // gip_start_updates(); // ... alle Bildschirmänderungen (auch die nach
                                //     dem gip_stop_updates() ) wieder anzeigen

        gip_sleep(3);

        // Berechne das naechste Spielfeld ...
        // Achtung; Für die Zelle (x,y) darf die Position (x,y) selbst *nicht*
        // mit in die Betrachtungen einbezogen werden.
        // Ausserdem darf bei zellen am rand nicht über den Rand hinausgegriffen
        // werden (diese Zellen haben entsprechend weniger Nachbarn) ...

        // TO DO


        // Kopiere das naechste Spielfeld in das aktuelle Spielfeld ...

        // TO DO

    }
    return 0;
}

void grid_init(bool grid[][grid_size])
{
    int eingabe = -1;
    do {
        cout << "Bitte waehlen Sie die Vorbelegung des Grids aus:" << endl
            << "0 - Zufall" << endl
            << "1 - Statisch" << endl
            << "2 - Blinker" << endl
            << "3 - Oktagon" << endl
            << "4 - Gleiter" << endl
            << "5 - Segler 1 (Light-Weight Spaceship)" << endl
            << "6 - Segler 2 (Middle-Weight Spaceship)" << endl
            << "? ";
        cin >> eingabe;
        cin.clear();
        cin.ignore(1000, '\n');
    } while (eingabe < 0 || eingabe > 6);

    if (eingabe == 0)
    {
        // Erstes Grid vorbelegen (per Zufallszahlen) ...

        // TO DO

    }
```

```
    else if (eingabe == 1)
    {
        const int pattern_size = 3;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '.' },
            { '*', '.', '*' },
            { '.', '*', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
    else if (eingabe == 2)
    {
        const int pattern_size = 3;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '.' },
            { '.', '*', '.' },
            { '.', '*', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
    else if (eingabe == 3)
    {
        const int pattern_size = 8;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '.', '.', '*', '*', '.', '.', '.' },
            { '.', '.', '*', '.', '.', '*', '.', '.' },
            { '.', '*', '.', '.', '.', '.', '*', '.' },
            { '*', '.', '.', '.', '.', '.', '.', '*' },
            { '*', '.', '.', '.', '.', '.', '.', '*' },
            { '.', '*', '.', '.', '.', '.', '*', '.' },
            { '.', '.', '*', '.', '.', '*', '.', '.' },
            { '.', '.', '.', '*', '*', '.', '.', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+1] = true;
    }
    else if (eingabe == 4)
    {
        const int pattern_size = 3;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '.' },
            { '.', '.', '*' },
            { '*', '*', '*' },
        };
        for (int y = 0; y < pattern_size; y++)
```

```
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
    else if (eingabe == 5)
    {
        const int pattern_size = 5;
        char pattern[pattern_size][pattern_size] =
        {
            { '*', '.', '.', '*', '.' },
            { '.', '.', '.', '.', '*' },
            { '*', '.', '.', '.', '*' },
            { '.', '*', '*', '*', '*' },
            { '.', '.', '.', '.', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
    else if (eingabe == 6)
    {
        const int pattern_size = 6;
        char pattern[pattern_size][pattern_size] =
        {
            { '.', '*', '*', '*', '*', '*' },
            { '*', '.', '.', '.', '.', '*' },
            { '.', '.', '.', '.', '.', '*' },
            { '*', '.', '.', '.', '*', '.' },
            { '.', '.', '*', '.', '.', '.' },
            { '.', '.', '.', '.', '.', '.' },
        };
        for (int y = 0; y < pattern_size; y++)
            for (int x = 0; x < pattern_size; x++)
                if (pattern[y][x] == '*')
                    grid[x][y+3] = true;
    }
}
```

Eine Zelle wird in einem leeren Feld neu geboren, wenn im vorigen Grid drei der umgebenden Nachbarzellen belebt waren.

Nachbarzellen: Eine Zelle, die nicht am Rand des Spielfelds liegt, hat 8 Nachbarn: oben, unten, links, rechts und 4x diagonal. Die Zelle selbst zählt nicht zu ihren eigenen Nachbarn. Zellen am Rand des Spielfelds haben weniger Nachbarn.

Eine Zelle bleibt in einem Feld am Leben, wenn im vorigen Grid zwei oder drei der umgebenden Nachbarzellen belebt waren.

In allen anderen Fällen wird keine Zelle geboren bzw. eine dort lebende Zelle stirbt wegen zu vieler oder zu weniger Nachbarn, d.h. das Feld wird im nächsten Spielfeld unbewohnt sein.

Stellen Sie das Spielfeld über die Zellgenerationen hinweg graphisch dar.

Testläufe: *(Animation der vorgegebenen Muster siehe Wikipedia Seite)*

```
Bitte waehlen Sie die Vorbelegung des Grids aus:
0 - Zufall
1 - Statisch
2 - Blinker
3 - Oktagon
4 - Gleiter
5 - Segler 1 (Light-Weight Spaceship)
6 - Segler 2 (Middle-Weight Spaceship)
?
```