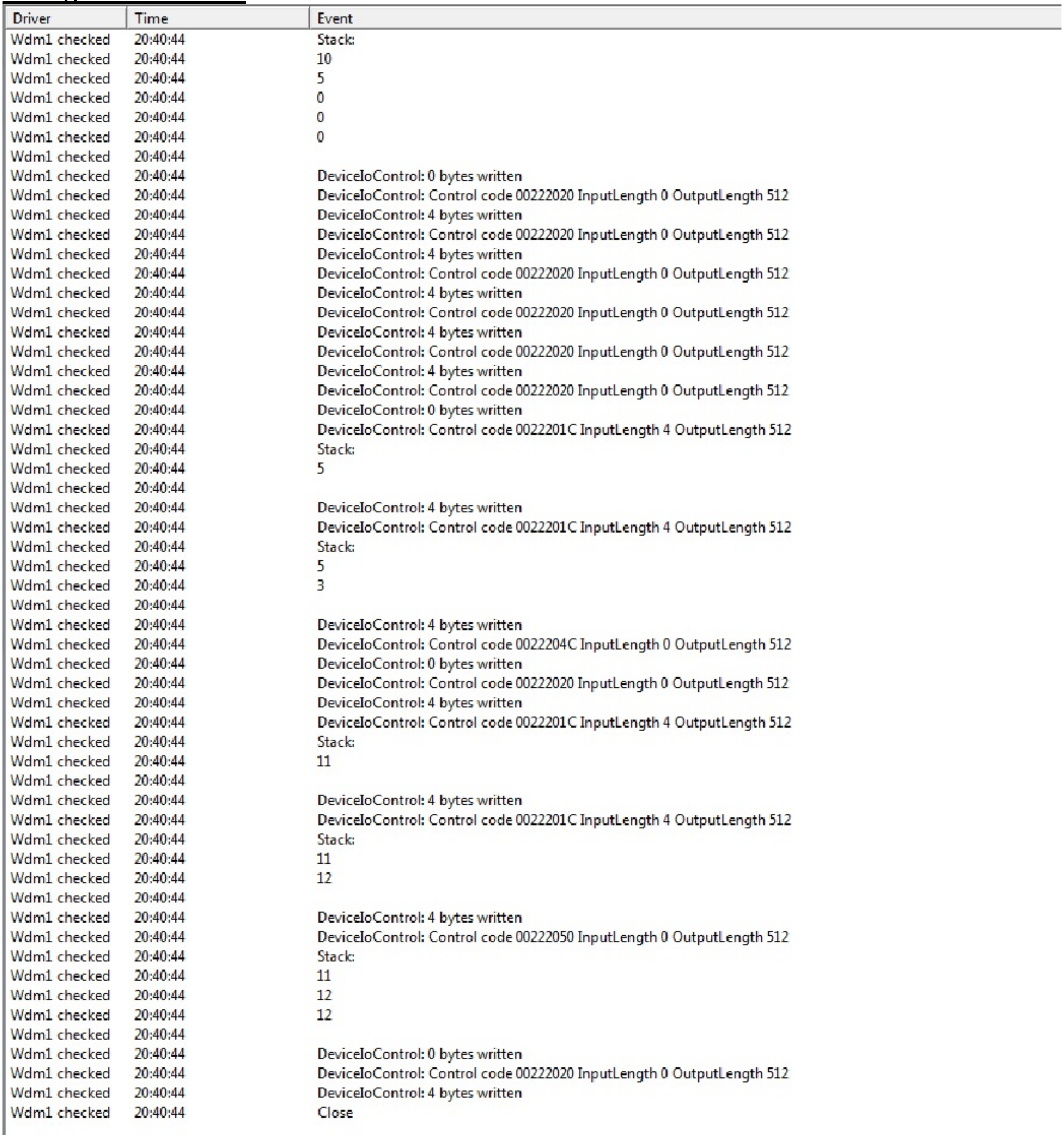
Aufgabe 1

Debug Print Monitor:



```
Driver          Time        Event
Wdm1 checked    20:40:44    Stack:
Wdm1 checked    20:40:44    10
Wdm1 checked    20:40:44    5
Wdm1 checked    20:40:44    0
Wdm1 checked    20:40:44    0
Wdm1 checked    20:40:44    0
Wdm1 checked    20:40:44
Wdm1 checked    20:40:44    DeviceIoControl: 0 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 0 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 0022201C InputLength 4 OutputLength 512
Wdm1 checked    20:40:44    Stack:
Wdm1 checked    20:40:44    5
Wdm1 checked    20:40:44
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 0022201C InputLength 4 OutputLength 512
Wdm1 checked    20:40:44    Stack:
Wdm1 checked    20:40:44    5
Wdm1 checked    20:40:44    3
Wdm1 checked    20:40:44
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 0022204C InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 0 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 0022201C InputLength 4 OutputLength 512
Wdm1 checked    20:40:44    Stack:
Wdm1 checked    20:40:44    11
Wdm1 checked    20:40:44
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 0022201C InputLength 4 OutputLength 512
Wdm1 checked    20:40:44    Stack:
Wdm1 checked    20:40:44    11
Wdm1 checked    20:40:44    12
Wdm1 checked    20:40:44
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222050 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    Stack:
Wdm1 checked    20:40:44    11
Wdm1 checked    20:40:44    12
Wdm1 checked    20:40:44    12
Wdm1 checked    20:40:44
Wdm1 checked    20:40:44    DeviceIoControl: 0 bytes written
Wdm1 checked    20:40:44    DeviceIoControl: Control code 00222020 InputLength 0 OutputLength 512
Wdm1 checked    20:40:44    DeviceIoControl: 4 bytes written
Wdm1 checked    20:40:44    Close
```

Testausgabe:

```
push and pop
5
push and add and pop
10
push and sub and pop
-3
push and mult and pop
5*3 = 15
push and invalid div and pop
5 / 3 = 0
push and invalid div 0 and pop
5 / 0 = 0
push to the limit
problem with driver or stack over/underflow
Unexpected error: (31, 'DeviceIoControl', 'A device attached to the system is not
functioning.')
pop to zero
problem with driver or stack over/underflow
Unexpected error: (31, 'DeviceIoControl', 'A device attached to the system is not
functioning.')
push and getdivrest and pop
5 modulo 3 = 0
duplicate
should be 12 = 12
```

Sourcecode:

wdm1-test.py:

```
# Test file for Wdm1

import win32file, win32api, sys
sys.path += ["DeviceDriverAccess/Release"]

from DeviceDriverAccess import GetDeviceViaInterface

from struct import *

# Constants for Wdm1
WDM1_GUID = pack("LHHBBBBBBBB", 0x1ef8a96b, 0x6c26, 0x42a4, 0xb9, 0x19, 0x82, 0x50,
0x93, 0x13, 0xbc, 0x5b)

FILE_DEVICE_UNKNOWN = 0x00000022
METHOD_BUFFERED = 0
METHOD_IN_DIRECT = 1
METHOD_OUT_DIRECT = 2
METHOD_NEITHER = 3
FILE_ANY_ACCESS = 0
```

```
ZERO_BUFFER = 0x801
REMOVE_BUFFER = 0x802
GET_BUFFER_SIZE = 0x803
GET_BUFFER = 0x804
UNRECOGNISED = 0x805
GET_BUILDTIME = 0x806
RPN_PUSH = 0x807
RPN_POP = 0x808
RPN_ADD = 0x809
RPN_SUB = 0x810
RPN_MULT = 0x811
RPN_DIV = 0x812
RPN_GETDIVREST = 0x813
RPN_DUPLI = 0x814

def CTL_CODE(DeviceType, Function, Method, Access):
    return (DeviceType << 16) | (Access << 14) | (Function << 2) | Method

class HWDevice:
    def __init__(self,guid):
        self.guid = guid
        self.drvHnd = None
        self.OpenDrv()

    def OpenDrv(self):
        """
        Open a handle to the device driver. If the driver is already open,
        close it first an reopen it.
        """
        self.CloseDrv()
        try:
            name = GetDeviceViaInterface(self.guid)
        except:
            raise IOError (1, "Wdm1 Device not found")

        desiredAccess = win32file.GENERIC_READ | win32file.GENERIC_WRITE
        self.drvHnd = win32file.CreateFile(name,
                                           desiredAccess,
                                           win32file.FILE_SHARE_WRITE,
                                           None,
                                           win32file.OPEN_EXISTING,
                                           0,
                                           0)

    def CloseDrv(self):
        """
        Close the handle to device driver
        """
        if self.drvHnd is not None:
            win32file.CloseHandle(self.drvHnd)
            self.drvHnd = None

    def Write(self, string):
        win32file.WriteFile(self.drvHnd, string, None)

    def Read(self, numofbytes=1):
        hr, result = win32file.ReadFile(self.drvHnd, numofbytes, None)
        return result

    def SetFilePointer(self, distance):
        win32file.SetFilePointer(self.drvHnd, distance, win32file.FILE_BEGIN)
    def DeviceIoControl(self, function, input):

        IOCTL_USB_GET_DEVICE_DESCRIPTOR = CTL_CODE(FILE_DEVICE_UNKNOWN, function, 
METHOD_BUFFERED, FILE_ANY_ACCESS)

        try:
            result = win32file.DeviceIoControl(self.drvHnd, 
IOCTL_USB_GET_DEVICE_DESCRIPTOR, input, 512)
        except win32file.error, e:
            print "problem with driver or stack over/underflow"
            print "Unexpected error:", e
            result = 0

        return result


d = HWDevice(WDM1_GUID)

print "Clear buffer ..."
d.DeviceIoControl(REMOVE_BUFFER,"")

bufferLength = d.DeviceIoControl(GET_BUFFER_SIZE,"")
result, = unpack('i', bufferLength)
print "Buffer length should be zero. Buffer Length = %d" % result

print "Write buffer ('Hello World Buffer! :D') ..."
d.Write("Hello World Buffer! :D")

bufferLength = d.DeviceIoControl(GET_BUFFER_SIZE,"")
result, = unpack('i', bufferLength)
print "Buffer length after write = %d" % result

print "Read 5 bytes from buffer ..."
result = d.Read(5)
print "Read bytes = %s" % result

print "Move FilePointer 5 bytes back ..."
d.SetFilePointer(5)

print "Read 50 bytes from buffer ..."
result = d.Read(50)
print "Read bytes = %s" % result

print "Clear buffer ..."
d.DeviceIoControl(REMOVE_BUFFER,"")

bufferLength = d.DeviceIoControl(GET_BUFFER_SIZE,"")
result, = unpack('i', bufferLength)
print "Buffer length should be zero. Buffer Length = %d" % result

dateTime = d.DeviceIoControl(GET_BUILDTIME,"")
print dateTime

print "push and pop"
d.DeviceIoControl(RPN_PUSH, pack("I", 5));
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print result
print "push and add and pop"
d.DeviceIoControl(RPN_PUSH,  pack("I", 5));
d.DeviceIoControl(RPN_PUSH, pack("I", 5));
d.DeviceIoControl(RPN_ADD, "");
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print result

print "push and sub and pop"
d.DeviceIoControl(RPN_PUSH,  pack("I", 5));
d.DeviceIoControl(RPN_PUSH, pack("I", 2));
d.DeviceIoControl(RPN_SUB, "");
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print result

print "push and mult and pop"
d.DeviceIoControl(RPN_PUSH,  pack("I", 10));
d.DeviceIoControl(RPN_PUSH,  pack("I", 5));
d.DeviceIoControl(RPN_PUSH,  pack("I", 3));
d.DeviceIoControl(RPN_MULT, "");
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print "5*3 = %d" % result

print "push and invalid div and pop"
d.DeviceIoControl(RPN_PUSH,  pack("I", 5));
d.DeviceIoControl(RPN_PUSH,  pack("I", 3));
d.DeviceIoControl(RPN_DIV, "");
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print "5 / 3 = %d" % result

print "push and invalid div 0 and pop"
d.DeviceIoControl(RPN_PUSH,  pack("I", 5));
d.DeviceIoControl(RPN_PUSH,  pack("I", 0));
d.DeviceIoControl(RPN_DIV, "");
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print "5 / 0 = %d" % result


print "push to the limit"
d.DeviceIoControl(RPN_PUSH,  pack("I", 5));
d.DeviceIoControl(RPN_PUSH,  pack("I", 0));
d.DeviceIoControl(RPN_PUSH,  pack("I", 0));
d.DeviceIoControl(RPN_PUSH,  pack("I", 0));
d.DeviceIoControl(RPN_PUSH,  pack("I", 0));


print "pop to zero"
d.DeviceIoControl(RPN_POP, "");
d.DeviceIoControl(RPN_POP, "");
d.DeviceIoControl(RPN_POP, "");
d.DeviceIoControl(RPN_POP, "");
d.DeviceIoControl(RPN_POP, "");
d.DeviceIoControl(RPN_POP, "");
```

```
print "push and getdivrest and pop"
d.DeviceIoControl(RPN_PUSH,  pack("I", 5));
d.DeviceIoControl(RPN_PUSH,  pack("I", 3));
d.DeviceIoControl(RPN_GETDIVREST, "");
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print "5 modulo 3 = %d" % result

print "duplicate"
d.DeviceIoControl(RPN_PUSH,  pack("I", 11));
d.DeviceIoControl(RPN_PUSH,  pack("I", 12));
d.DeviceIoControl(RPN_DUPLI, "");
value = d.DeviceIoControl(RPN_POP, "");
result, = unpack('i', value)
print "should be 12 = %d" % result

d.CloseDrv()
```

Stack.h:

```
#ifndef STACK_H
#define STACK_H

size_t const STACK_MAX = 5;

struct Stack {
       int     data[STACK_MAX];
       int     size;
};
typedef struct Stack Stack;

void Stack_Init(Stack *S);
bool Stack_Push(Stack *S, int d);
bool Stack_Pop(Stack *S, int &d);
bool Stack_Dup(Stack *S);
bool Stack_IsEmpty(Stack *S);
bool Stack_IsFull(Stack *S);

#endif
```

Stack.cpp:

```
#include "Stack.h"

void Stack_Init(Stack *S)
{
       S->size = 0;
}

bool Stack_Push(Stack *S, int d)
{
       if (S->size < STACK_MAX)
       {
              S->data[S->size] = d;
              S->size++;
              return true;
       }
       else
       {
              return false;
       }
}
```

```
bool Stack_Pop(Stack *S, int &d)
{
	if (S->size == 0)
	{
		return false;
	}
	else
	{
		S->size--;
		d = S->data[S->size];
		return true;
	}
}

bool Stack_Dup(Stack *S)
{
	if (Stack_IsEmpty(S) || Stack_IsFull(S))
	{
		return false;
	}
	else
	{
		int val = S->data[(S->size)-1];
		S->data[S->size] = val;
		S->size++;
		return true;
	}
}

bool Stack_IsEmpty(Stack *S)
{
	return S->size == 0;
}

bool Stack_IsFull(Stack *S)
{
	return S->size == STACK_MAX;
}
```

RpnCalculator.h:

```
#ifndef RPN_CALC_H
#define RPN_CALC_H

#include "Stack.h"

bool RpnCalculator_Add(Stack *s);
bool RpnCalculator_Substract(Stack *s);
bool RpnCalculator_Multiply(Stack *s);
bool RpnCalculator_Divide(Stack *s);
bool RpnCalculator_Modulo(Stack *s);

#endif
```

RpnCalculator.cpp:

```
#include "RpnCalculator.h"

bool RpnCalculator_Calc(Stack *s, char operation)
{
    int a = 0;
    int b = 0;
    int result = 0;

    if (!Stack_Pop(s, a))
    {
        return false;
    }
    if (!Stack_Pop(s, b))
    {
        return false;
    }

    switch (operation)
    {
        case '+':
        {
            result = a + b;
            break;
        }
        case '-':
        {
            result = a - b;
            break;
        }
        case '*':
        {
            result = a * b;
            break;
        }
        case '/':
        {
            if (b == 0)
            {
                return false;
            }
            result = a / b;
            break;
        }
        case '%':
        {
            if (b == 0)
            {
                return false;
            }
            result = a / b;
            break;
        }
    }

    if (!Stack_Push(s, result))
    {
        return false;
    }
    return true;
}
```

```
bool RpnCalculator_Add(Stack *s)
{
	return RpnCalculator_Calc(s, '+');
}

bool RpnCalculator_Substract(Stack *s)
{
	return RpnCalculator_Calc(s, '-');
}

bool RpnCalculator_Multiply(Stack *s)
{
	return RpnCalculator_Calc(s, '*');
}

bool RpnCalculator_Divide(Stack *s)
{
	return RpnCalculator_Calc(s, '/');
}

bool RpnCalculator_Modulo(Stack *s)
{
	return RpnCalculator_Calc(s, '%');
}
```

<u>Ioctl.h:</u>

```
//	DeviceIoControl IOCTL codes supported by Wdm1

#define IOCTL_WDM1_ZERO_BUFFER CTL_CODE( \
                    FILE_DEVICE_UNKNOWN,              \
                    0x801,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_REMOVE_BUFFER CTL_CODE(      \
                    FILE_DEVICE_UNKNOWN,              \
                    0x802,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_GET_BUFFER_SIZE CTL_CODE(    \
                    FILE_DEVICE_UNKNOWN,              \
                    0x803,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_GET_BUFFER CTL_CODE(  \
                    FILE_DEVICE_UNKNOWN,              \
                    0x804,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_UNRECOGNISED CTL_CODE(       \
                    FILE_DEVICE_UNKNOWN,              \
                    0x805,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_GET_BUILDTIME CTL_CODE(      \
                    FILE_DEVICE_UNKNOWN,              \
                    0x806,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_PUSH CTL_CODE(           \
                    FILE_DEVICE_UNKNOWN,              \
                    0x807,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_POP CTL_CODE(     \
                    FILE_DEVICE_UNKNOWN,              \
                    0x808,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_ADD CTL_CODE(     \
                    FILE_DEVICE_UNKNOWN,              \
                    0x809,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_SUB CTL_CODE(     \
                    FILE_DEVICE_UNKNOWN,              \
                    0x810,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_MULT CTL_CODE(    \
                    FILE_DEVICE_UNKNOWN,              \
                    0x811,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_DIV CTL_CODE(     \
                    FILE_DEVICE_UNKNOWN,              \
                    0x812,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_GETDIVREST CTL_CODE(     \
                    FILE_DEVICE_UNKNOWN,              \
                    0x813,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)

#define IOCTL_WDM1_RPN_DUPLI CTL_CODE(   \
                    FILE_DEVICE_UNKNOWN,              \
                    0x814,                                          \
                    METHOD_BUFFERED,                         \
                    FILE_ANY_ACCESS)
```

Dispatch.cpp:

```
#include "wdm1.h"
#include "Ioctl.h"
#include "RpnCalculator.h"

KSPIN_LOCK BufferLock;
PUCHAR Buffer = NULL;
ULONG  BufferSize = 0;

int const dateTimeSize = 21;
char dateTimeBuffer[dateTimeSize];

// RPN Stack
Stack s;
```

```
// Print Stack
void DebugPrintStack(){
	int i = 0;
	DebugPrint("Stack: ");
	for (i = 0; i < s.size; i++){
		DebugPrint("%d", (int)s.data[i]);
	}
	DebugPrint("\n");
}


NTSTATUS Wdm1Create(IN PDEVICE_OBJECT fdo,
	IN PIRP Irp)
{
	PIO_STACK_LOCATION IrpStack = IoGetCurrentIrpStackLocation(Irp);
	DebugPrint("Create File is %T", &(IrpStack->FileObject->FileName));

	Stack_Init(&s);

	// Complete successfully
	return CompleteIrp(Irp, STATUS_SUCCESS, 0);
}

...
...
...

NTSTATUS Wdm1DeviceControl(IN PDEVICE_OBJECT fdo,
	IN PIRP Irp)
{
	PIO_STACK_LOCATION IrpStack = IoGetCurrentIrpStackLocation(Irp);
	NTSTATUS status = STATUS_SUCCESS;
	ULONG BytesTxd = 0;

	ULONG ControlCode = IrpStack->Parameters.DeviceIoControl.IoControlCode;
	ULONG InputLength = IrpStack->Parameters.DeviceIoControl.InputBufferLength;
	ULONG OutputLength = IrpStack->Parameters.DeviceIoControl.OutputBufferLength;

	DebugPrint("DeviceIoControl: Control code %x InputLength %d OutputLength %d",
		ControlCode, InputLength, OutputLength);

	// Get access to the shared buffer
	KIRQL irql;
	KeAcquireSpinLock(&BufferLock, &irql);
	switch (ControlCode)
	{
		///////	Zero Buffer
	case IOCTL_WDM1_ZERO_BUFFER:
		// Zero the buffer
		if (Buffer != NULL && BufferSize > 0)
			RtlZeroMemory(Buffer, BufferSize);
		break;

		///////	Remove Buffer
	case IOCTL_WDM1_REMOVE_BUFFER:
		if (Buffer != NULL)
		{
			ExFreePool(Buffer);
			Buffer = NULL;
			BufferSize = 0;
		}
		break;
```

```
        ///////     Get Buffer Size as ULONG
    case IOCTL_WDM1_GET_BUFFER_SIZE:
        if (OutputLength < sizeof(ULONG))
            status = STATUS_INVALID_PARAMETER;
        else
        {
            BytesTxd = sizeof(ULONG);
            RtlCopyMemory(Irp->AssociatedIrp.SystemBuffer, &BufferSize, 
sizeof(ULONG));
        }
        break;

        ///////     Get Buffer
    case IOCTL_WDM1_GET_BUFFER:
        if (OutputLength > BufferSize)
            status = STATUS_INVALID_PARAMETER;
        else
        {
            BytesTxd = OutputLength;
            RtlCopyMemory(Irp->AssociatedIrp.SystemBuffer, Buffer, BytesTxd);
        }
        break;

        ///////     Get DateTime
    case IOCTL_WDM1_GET_BUILDTIME:
    {
        if (OutputLength < dateTimeSize){
            status = STATUS_INVALID_PARAMETER;
        }
        else {
            memset(dateTimeBuffer, 0, dateTimeSize);
            strcpy(dateTimeBuffer, __DATE__);
            strcat(dateTimeBuffer, " ");
            strcat(dateTimeBuffer, __TIME__);
            DebugPrint("DateTime: %s", dateTimeBuffer);
            BytesTxd = dateTimeSize;
            RtlCopyMemory(Irp->AssociatedIrp.SystemBuffer, dateTimeBuffer, 
dateTimeSize);
        }
    }
        break;

        /////// ------------- RPN STACK --------------------
    case IOCTL_WDM1_RPN_PUSH:
    {
        int value = 0;
        RtlCopyMemory(&value, Irp->AssociatedIrp.SystemBuffer, 4); // 4 byte = 32bit
        BytesTxd = 4;
        if (!Stack_Push(&s, value)){
            status = STATUS_UNSUCCESSFUL;
            BytesTxd = 0;
        }
        DebugPrintStack();
    }
        break;

    case IOCTL_WDM1_RPN_POP:
    {
        if (OutputLength < 4) {
            status = STATUS_INVALID_PARAMETER;
        }
        else {
            int value = 0;
            if (Stack_Pop(&s, value)){
                BytesTxd = 4;
                RtlCopyMemory(Irp->AssociatedIrp.SystemBuffer, &value, BytesTxd);
            }
            else {
                BytesTxd = 0;
                status = STATUS_UNSUCCESSFUL;
            }
        }
    }
        break;

    case IOCTL_WDM1_RPN_ADD:
        if (!RpnCalculator_Add(&s)){
            BytesTxd = 0;
            status = STATUS_UNSUCCESSFUL;
        }
        break;

    case IOCTL_WDM1_RPN_SUB:
        if (!RpnCalculator_Substract(&s)){
            BytesTxd = 0;
            status = STATUS_UNSUCCESSFUL;
        }
        break;

    case IOCTL_WDM1_RPN_MULT:
        if (!RpnCalculator_Multiply(&s)){
            BytesTxd = 0;
            status = STATUS_UNSUCCESSFUL;
        }
        break;

    case IOCTL_WDM1_RPN_DIV:
        if (!RpnCalculator_Divide(&s)){
            BytesTxd = 0;
            status = STATUS_UNSUCCESSFUL;
        }
        break;

    case IOCTL_WDM1_RPN_GETDIVREST:
        if (!RpnCalculator_Modulo(&s)){
            BytesTxd = 0;
            status = STATUS_UNSUCCESSFUL;
        }
        break;

    case IOCTL_WDM1_RPN_DUPLI:
        if (!Stack_Dup(&s)){
            BytesTxd = 0;
            status = STATUS_UNSUCCESSFUL;
        }
        DebugPrintStack();
        break;

        ///////     Invalid request
    default:
        status = STATUS_INVALID_DEVICE_REQUEST;
    }
    // Release shared buffer
    KeReleaseSpinLock(&BufferLock, irql);

    DebugPrint("DeviceIoControl: %d bytes written", (int)BytesTxd);

    // Complete IRP
    return CompleteIrp(Irp, status, BytesTxd);
}
```