Übung 4.1 – Taschenrechner polnische Notation:

a) Es soll das Programm des Taschenrechners mit polnischer Notation umgesetzt werden (gemäss Beispiel im Buch K&R S. 74 bis S.77 – siehe Anhang). Folgende zusätzliche Informationen hierzu:

- *Verwenden Sie das Template main.c aus dem Anhang als Vorlage*
- *"stdin" ist hier das File "inputfile1.txt" in dem Pfad von „main“ (File mit netbeans als Editor verändern – Win-Text-Editor macht kein EOF wie gewünscht)*
- *Sie können auch über die Konsole die Eingabe machen – EOF mittels ctrl+d*
- *Alle Funktionen sind zuerst in einem Source-File („main...“ File) zu halten*
- *Zeitbedarf 45': codieren, debugging und testen*

b) Teilen Sie das obige Taschenrechner Programm in Quelldatei, Headerfiles und main auf, so dass Sie eine Aufteilung gemäss Buch S.80 (siehe Anhang) erreichen. Speichern Sie Ihr Projekt unter einem neuen Namen und testen Sie Ihr Programm entsprechend.

c) Optional: Erweitern Sie Ihren Taschenrechner mit den Funktionen sin, exp, power

Lösung 4.1 a)

```
/**============================================================================================
Hochschule Luzern - Technik & Architektur                              Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                               http://hslu.ximit.ch
--------------------------------------------------------------------------------------------
Name        : Uebung_4-1.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Taschenrechner polnische Notation:
              o Verwenden Sie das Template main.c aus dem Anhang als Vorlage
              o "stdin" ist hier das File "inputfile1.txt" in dem Pfad von „main“ (File mit
                NetBeans als Editor verändern – Win-Text-Editor macht kein EOF wie gewünscht)
              o Sie können auch über die Konsole die Eingabe machen – EOF mittels ctrl+d
              o Alle Funktionen sind zuerst in einem Source-File („main...“ File) zu halten
              o Zeitbedarf 45': codieren, debugging und testen
============================================================================================*/

#include <stdio.h>
#include <stdlib.h>             /* fuer atof() */
#include <ctype.h>              /* fuer isdigit() */

/* Konstanten */
#define MAXOP 100               /* max Laenge von Operand und/oder Operator */
#define BUFSIZE 100             /* Buffer Size */
#define MAXVAL 100              /* Stack Size */
#define NUMBER '0'              /* Anzeige, Zahl */

/* Globale Variablen */
char buf[BUFSIZE];              /* Buffer */
int bufp = 0;                   /* Buffer-Pointer */
double val[MAXVAL];             /* Stack */
int sp = 0;                             /* Stack-Pointer */

/* Funktions-Deklaration */
void push(double f);    /* push: f auf den Stack bringen */
double pop(void);               /* pop: Wert von Stack holen und liefern */
int getop(char s[]);    /* getop: naechsten Operator und /oder numerischen Operanden holen */
int getch(void);        /* getch: Zeichen holen mit Buffer */
void ungetch(int c);    /* ungetch: Zeichen wieder zurueckstellen wenn zuviel geholt */
```

```c
/**
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv)
{
	/* Taschenrechner in umgekehrter polnischer Notation - Beispiel S. 74 K&R */

	int type;
	double op2;
	char s[MAXOP];

	//Oeffnet das File "inputfile.txt" in dem Pfad von main.c und nimmt das File als
	//Standard Eingabe anstelle Tastatur
	freopen("inputfile1.txt", "r", stdin);
	while ((type = getop(s)) != EOF) {
		switch (type) {
		case NUMBER:	// Zahl auf Stack pushen
			push(atof(s));
			break;
		case '+':		// + Plus
			push(pop() + pop());
			break;
		case '-':		// - Minus
			op2 = pop();
			push(pop() - op2);
			break;
		case '*':		// * Multiplikation
			push(pop() * pop());
			break;
		case '/':		// / Division
			op2 = pop();
			if (op2 != 0.0) {
				push(pop() / op2);
			} else {
				printf("Error: Division durch 0!\n");
			}
			break;
		case '\n':		// Resultat Ausgeben
			printf("Result:\t%.8g\n", pop());
			break;
		default:		// Unbekannter Befehl
				printf("Error: Unbekannter Befehle: %s\n", s);
				break;
		} // switch (type)
	} // while ( != EOF)

	fclose(stdin); // Schliesst die Standard-Eingabe stdin (das File von freopen)

	return(EXIT_SUCCESS);
}


/* push: f auf den Stack bringen */
void push(double f) {
	if (sp < MAXVAL) {
		val[sp++] = f;
	} else {
		printf("Error: Stack ist voll! %g kann nicht eingefuegt werden...\n", f);
	}
}

/*pop: Wert von Stack holen und liefern */
double pop(void) {
	if (sp > 0) {
		return val[--sp];
	} else {
		printf("Error: Stack ist leer!\n");
		return 0.0;
	}
}
```

```c
/*getop: naechsten Operator und /oder numerischen Operanden holen */
int getop(char s[]){
    int i, c;

    while ((s[0] = c = getch()) == ' ' || c == '\t')
                ; // whitespaces entfernen - tue nichts bei diese Zeichen

    s[1] = '\0'; // String terminieren
    if (!isdigit(c) && c != '.') {
                return c; /* keine Zahl --> Befehl */
        }
    i = 0;
    if (isdigit(c)) { /* ganzzahligen Teil sammeln */
        while (isdigit(s[++i] = c = getch())) // solange es eine Zahl ist
            ;
        }
    if (c == '.') { /* Dezimalstellen */
        while (isdigit(s[++i] = c = getch())) // solange es eine Zahl ist
            ;
        }
    s[i] = '\0'; // String terminieren
    if (c != EOF) { // solange nicht am Ende von der Zeile
        ungetch(c); // Zeichen zurück auf Stack
        }

    return NUMBER; // default return wert 0
}

/* getch: Zeichen holen mit Buffer */
int getch(void) {
        // return (bufp > 0) ? buf[--bufp] : getchar();
        if (bufp > 0) {
                return buf[--bufp];
        } else {
                return getchar();
        }
}

/* ungetch: Zeichen wieder zurueckstellen wenn zuviel geholt */
void ungetch(int c){
        if (bufp >= BUFSIZE) {
        printf("Error ungetch(): Zu viele Zeichen\n");
        } else {
        buf[bufp++] = c;
        }
}
```

Lösung 4.1 b) & c)

main.c

```
/**=============================================================================================================
Hochschule Luzern - Technik & Architektur                                          Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                           http://hslu.ximit.ch
-------------------------------------------------------------------------------------------------------------
Name        : main.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Taschenrechner polnische Notation
=============================================================================================================*/

#include <stdio.h>
#include <stdlib.h>             /* fuer atof() */
#include <math.h>               /* fuer sin() und co */
#include "calc.h"               /* Eigene Methoden einbinden */

/* Konstanten */
#define MAXOP 100               /* max Laenge von Operand und/oder Operator */
#define PI 3.14159265   /* PI */

/**
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv)
{
        /* Taschenrechner in umgekehrter polnischer Notation - Beispiel S. 74 K&R */

        int type;
        double op2;
        char s[MAXOP];

        //Oeffnet das File "inputfile.txt" in dem Pfad von main.c und nimmt das File als
        //Standard Eingabe anstelle Tastatur
        freopen("inputfile1.txt", "r", stdin);
        while ((type = getop(s)) != EOF) {
                switch (type) {
                case NUMBER:    // Zahl auf Stack pushen
                        push(atof(s));
                        break;
                case '+':               // + Plus
                        push(pop() + pop());
                        break;
                case '-':               // - Minus
                        op2 = pop();
                        push(pop() - op2);
                        break;
                case '*':               // * Multiplikation
                        push(pop() * pop());
                        break;
                case '/':               // / Division
                        op2 = pop();
                        if (op2 != 0.0) {
                                push(pop() / op2);
                        } else {
                                printf("Error: Division durch 0!\n");
                        }
                        break;
                case FN_SIN:    // Sinus
                        push(sin(pop() * PI / 180.0));
                        break;
                case FN_COS:    // Cosinus
                        push(cos(pop() * PI / 180.0));
                        break;
                case FN_TAN:    // Tangens
                        push(tan(pop() * PI / 180.0));
                        break;
                case FN_EXP:    // Exp, e^x
                        push(exp(pop()));
                        break;
                case FN_POWER:  // pow, x^y
                        op2 = pop();
                        push(pow(pop(), op2));
                        break;
                case '\n':              // Resultat Ausgeben
                        printf("Result:\t%.8g\n", pop());
```

```
                        break;
                default:        // Unbekannter Befehl
                        printf("Error: Unbekannter Befehl: %s\n", s);
                        break;
                } // switch (type)
        } // while ( != EOF)

        fclose(stdin); // Schliesst die Standard-Eingabe stdin (das File von freopen)

        return(EXIT_SUCCESS);
}
```

calc.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                       Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                        http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Name        : calc.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Taschenrechner polnische Notation
============================================================================================================*/
/* Konstanten */
#define NUMBER '0'              /* Anzeige, Zahl */
#define FN_SIN '-1'             /* Sinus */
#define FN_COS '-2'             /* Cosinus */
#define FN_TAN '-3'             /* Tangens */
#define FN_EXP '-4'             /* Exponent */
#define FN_POWER '-5'   /* Power */

/* Funktions-Deklaration */
void push(double f);    /* push: f auf den Stack bringen */
double pop(void);               /* pop: Wert von Stack holen und liefern */
int getop(char s[]);    /* getop: naechsten Operator und /oder numerischen Operanden holen */
int getch(void);        /* getch: Zeichen holen mit Buffer */
void ungetch(int c);    /* ungetch: Zeichen wieder zurueckstellen wenn zuviel geholt */
```

getch.ch

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                       Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                        http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Name        : getch.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Taschenrechner polnische Notation
              getch: Zeichen holen mit Buffer
              ungetch: Zeichen wieder zurueckstellen wenn zuviel geholt
============================================================================================================*/
#include <stdio.h>

/* Konstanten */
#define BUFSIZE 100             /* Buffer Size */

/* Globale Variablen */
char buf[BUFSIZE];              /* Buffer */
int bufp = 0;                   /* Buffer-Pointer */


/* getch: Zeichen holen mit Buffer */
int getch(void) {
        // return (bufp > 0) ? buf[--bufp] : getchar();
        if (bufp > 0) {
                return buf[--bufp];
        } else {
                return getchar();
        }
}

/* ungetch: Zeichen wieder zurueckstellen wenn zuviel geholt */
void ungetch(int c){
        if (bufp >= BUFSIZE) {
        printf("Error ungetch(): Zu viele Zeichen\n");
        } else {
        buf[bufp++] = c;
        }
}
```

getop.c

```
/**============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
--------------------------------------------------------------------------------------------
Name        : getop.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Taschenrechner polnische Notation
              naechsten Operator und /oder numerischen Operanden holen
============================================================================================*/

#include <stdio.h>
#include <stdlib.h>             /* fuer atof() */
#include <ctype.h>              /* fuer isdigit() */
#include <string.h>             /* fuer strcmp() */
#include "calc.h"               /* Eigene Methoden einbinden */

/*getop: naechsten Operator und /oder numerischen Operanden holen */
int getop(char s[]){
    int i, c, cOld;

    while ((s[0] = c = getch()) == ' ' || c == '\t')
                ; // whitespaces entfernen - tue nichts

    s[1] = '\0'; // String terminieren

    if (!isdigit(c) && c != '.') { // Befehl Sammeln
                cOld = c;
                i = 0;

                while (isalpha(s[++i] = c = getch())) { // solange es ein char ist
            ; // do nothing
                }
                s[i] = '\0'; // String terminieren
                if (!isalpha(c)) {
                        ungetch(c);
                }

                if (strcmp(s, "sin") == 0) {
                        return FN_SIN; // Sinus
                } else if (strcmp(s, "cos") == 0) {
                        return FN_COS; // Cosinus
                } else if (strcmp(s, "tan") == 0) {
                        return FN_TAN; // Tangens
                } else if (strcmp(s, "exp") == 0) {
                        return FN_EXP; // Exponent
                } else if (strcmp(s, "power") == 0) {
                        return FN_POWER; // Power
                } else {
                        return cOld; // keine Zahl --> Befehl
                }
        }
    i = 0;
    if (isdigit(c)) { /* ganzzahligen Teil sammeln */
        while (isdigit(s[++i] = c = getch())) // solange es eine Zahl ist
            ;
        }
    if (c == '.') { /* Dezimalstellen */
        while (isdigit(s[++i] = c = getch())) // solange es eine Zahl ist
            ;
        }
    s[i] = '\0'; // String terminieren
    if (c != EOF) { // solange nicht am Ende von der Zeile
        ungetch(c); // Zeichen zurück auf Stack
        }

    return NUMBER; // default return wert 0
}
```

stack.c

```
/**=============================================================================================
Hochschule Luzern - Technik & Architektur                                   Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                    http://hslu.ximit.ch
-----------------------------------------------------------------------------------------------
Name        : stack.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Taschenrechner polnische Notation
              Implementiert den Stack
=============================================================================================*/

#include <stdio.h>
#include "calc.h"               /* Eigene Methoden einbinden */

/* Konstanten */
#define MAXVAL 100              /* Stack Size */

/* Globale Variablen */
double val[MAXVAL];             /* Stack */
int sp = 0;                             /* Stack-Pointer */

/* push: f auf den Stack bringen */
void push(double f) {
        if (sp < MAXVAL) {
                val[sp++] = f;
        } else {
                printf("Error: Stack ist voll! %g kann nicht eingefuegt werden...\n", f);
        }
}

/*pop: Wert von Stack holen und liefern */
double pop(void) {
        if (sp > 0) {
                return val[--sp];
        } else {
                printf("Error: Stack ist leer!\n");
                return 0.0;
        }
}
```

Übung 4.2 – Makros

Schreiben Sie Makros:

- *MIN(x, y) liefert den kleineren der beiden Werte*
- *MAX(x, y) liefert den grösseren der beiden Werte*
- *SWAP(t,x,y) vertauscht die zwei Argumente x und y vom Typ t.*

Testen Sie Ihr Programm mit verschiedenen Testfällen.

```
/**============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
--------------------------------------------------------------------------------------------
Name        : Uebung_4-2.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Schreiben Sie Makros
              o MIN(x, y) liefert den kleineren der beiden Werte
              o MAX(x, y) liefert den grösseren der beiden Werte
              o SWAP(t,x,y) vertauscht die zwei Argumente x und y vom Typ t.
============================================================================================*/

#include <stdio.h>
#include <stdlib.h>

#define MIN(x, y)               ((x) < (y) ? (x) : (y))
#define MAX(x, y)               ((x) > (y) ? (x) : (y))
#define SWAP(t, x, y)   { t tmp; tmp = x; x = y; y = tmp; }

/**
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv)
{
        int x, y;
        char* a, *b;

        x = 5;
        y = 10;
        a = "Hallo";
        b = "Du";

        printf("Value x: %i\n", x);
        printf("Value y: %i\n", y);
        printf("> Min: %i\n", MIN(x,y));
        printf("> Max: %i\n", MAX(x,y));
        printf("> Swap...\n");
        SWAP(int, x, y);
        printf("  > Value x: %i\n", x);
        printf("  > Value y: %i\n", y);

        printf("----------------------------------------\n");
        printf("String a: %s\n", a);
        printf("String b: %s\n", b);
        printf("> Min: %s\n", MIN(a,b));
        printf("> Max: %s\n", MAX(a,b));
        printf("> Swap...\n");
        SWAP(char*, a, b);
        printf("  > Value a: %s\n", a);
        printf("  > Value b: %s\n", b);



        fflush(stdin); //flushes the input stream and gets rid of '\n'
        printf("\n\nPress [Enter] to exit...");
        getchar();
        return(EXIT_SUCCESS);
}
```

Übung 4.3 – Ampel

Schreiben Sie eine kleine Ampel-Simulation, welche der Reihe nach die Zustände RED – GREEN – GREEN_BLINKING – YELLOW – RED – etc. annimmt. Teilen Sie das Programm in drei Files main.c, ampel.c und ampel.h auf und halten Sie den Zustand als statische, lokale Variable im File ampel.c fest. Das File ampel.c soll die drei Funktionen getState(), nextState() und printState() zur Verfügung stellen, welche Sie im main.c testen.

main.c

```
/**=========================================================================================================
Hochschule Luzern - Technik & Architektur                                   Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                    http://hslu.ximit.ch
---------------------------------------------------------------------------------------------------------
Name        : main.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Ampel
              Schreiben Sie eine kleine Ampel-Simulation, welche der Reihe nach die Zustände
              RED – GREEN – GREEN_BLINKING – YELLOW – RED – etc. annimmt.
=========================================================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <windows.h> /* Sleep() */
#include "ampel.h"

/**
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv)
{
        int i;

        for(i=0; i<3; i++) // dreimal wiederholen
        {
                // Rot
                printState();
                printf("  State: %i\n", getState());
                Sleep(5000); // 6sec delay
                nextState();

                // Gruen
                printState();
                printf("  State: %i\n", getState());
                Sleep(3000); // 4sec delay
                nextState();

                // Gruen-Blinkend
                printState();
                printf("  State: %i\n", getState());
                Sleep(1000); // 1sec delay
                nextState();

                // Gelb
                printState();
                printf("  State: %i\n", getState());
                Sleep(1000); // 1sec delay
                nextState();

                //printf("\n----------------------\n\n");
        } // dreimal wiederholen


        fflush(stdin); //flushes the input stream and gets rid of '\n'
        printf("\n\nPress [Enter] to exit...");
        getchar();
        return(EXIT_SUCCESS);
}
```

ampel.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                         http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Name        : ampel.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Ampel
============================================================================================================*/

/**
 * Gibt den aktuellen Zustand zurück
 * @return Int-Wert für den aktuellen Zustand
 */
int getState(void);

/**
 * Setz die Ampel auf den nächsten Status, Zustand
 */
void nextState(void);

/**
 * Gibt den aktuellen Zustand auf stdout aus
 */
void printState(void);
```

ampel.c

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                         http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Name        : ampel.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Ampel
============================================================================================================*/

#include <stdio.h>
#include "ampel.h"

typedef enum State_ {
        RED, // 0
        GREEN, // 1
        GREEN_BLINKING, // 2
        YELLOW, // 3
        STATE_COUNT
} State;

static int currState = 0;


int getState(void)
{
        return currState;
}

void nextState(void)
{
        if (currState < (STATE_COUNT - 1)) {
                currState++;
        } else {
                currState = 0;
        }
}


void printState(void)
{
        switch (currState) {
        case RED:
                printf("Rot\n");
                break;
        case GREEN:
                printf("Gruen\n");
                break;
        case GREEN_BLINKING:
                printf("Gruen blinkend\n");
                break;
        case YELLOW:
                printf("Gelb\n");
                break;
        } // switch
}
```

Übung 4.4 – Message Queue

Machen Sie eine einfache „Message Queue“, welche mind. die drei Funktionen initialize, enqueue and dequeue enthält. Die „Messages“ sollen aus einem Integerwert bestehen. enqueue schreibt die neue „Message“ an das Ende der „Queue“, dequeue liest das nächste Element aus der „Queue“. Falls keine „Messages“ vorhanden sind soll die Funktion dequeue blockieren. (Tipp: siehe verkettete Liste) Achten Sie auf ein sauberes „Allozieren und Freigeben“ der Speicherplätze und testen Sie Ihr Programm entsprechend!

main.c

```
/**=============================================================================================
Hochschule Luzern - Technik & Architektur                                  Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                   http://hslu.ximit.ch
-----------------------------------------------------------------------------------------------
Name        : main.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Message Queue
              Machen Sie eine einfache „Message Queue“, welche mind. die drei Funktionen
              initialize, enqueue and dequeue enthält. Die „Messages“ sollen aus einem Integerwert
              bestehen. enqueue schreibt die neue „Message“ an das Ende der „Queue“, dequeue liest
              das nächste Element aus der „Queue“. Falls keine „Messages“ vorhanden sind soll die
              Funktion dequeue blockieren.
==============================================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include "queue.h"


/**
 * The main procedure to run the application.
 * @param argc Number of command line arguments
 * @param argv The forwarded command line arguments
 * @return Application return (error) code
 */
int main(int argc, char** argv)
{

        // init MessageQueue
        MessageQueue* aQueue = initialize();
        int i;

        // Write Queue
        for(i=0; i<10; i++) {
                printf("in:  %d\n",enqueue(aQueue,i));
        }
        printf("-----------------------------------------------------\n");

        // Read Queue
        for(i=0; i<5; i++) {
                printf("out: %d\n",dequeue(aQueue));
        }
        printf("-----------------------------------------------------\n");

        // Write Queue
        for(i=500; i>490; i--) {
                printf("in:  %d\n",enqueue(aQueue,i));
        }
        printf("-----------------------------------------------------\n");

        // Read Queue
        for(i=0; i<15; i++) {
                printf("out: %d\n",dequeue(aQueue));
        }
        printf("-----------------------------------------------------\n");


        // DEAD LOCK
        // Read Queue --> Queue is empty!
        printf("out: %d\n",dequeue(aQueue));

        return(EXIT_SUCCESS);
}
```

queue.h

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                        Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                         http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Name        : queue.h
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Message Queue
============================================================================================================*/


typedef struct Node {
  struct Node* next;            // next node pointer
  int data;                             // value
} Node;

typedef struct MessageQueue {
  Node* head;                           // first node pointer
  Node* rear;                           // rear node pointer
  int size;                                     // queue size
} MessageQueue;


/**
 * Allocate Memory for the Queue
 * @return Pointer for the new Queue
 */
MessageQueue* initialize();

/**
 * Add a new Value to the Queue
 * @param aQueue Pointer to the Queue
 * @param aValue Value to add
 * @return Value that has been added
 */
int enqueue(MessageQueue *aQueue, int aValue);

/**
 * Get a Value from the Queue
 * @param aQueue Pointer to the Queue
 * @return Value from the Queue
 */
int dequeue(MessageQueue *aQueue);
```

queue.c

```
/**============================================================================================================
Hochschule Luzern - Technik & Architektur                                   Mikrocontroller - HS2012
Copyright 2012 Felix Rohrer                                                    http://hslu.ximit.ch
------------------------------------------------------------------------------------------------------------
Name        : queue.c
Author      : Felix Rohrer <felix.rohrer@stud.hslu.ch>
Version     : 2012-10-07v1.0
Description : Message Queue
============================================================================================================*/

#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include "queue.h"


MessageQueue* initialize() {
        //create a new Queue
        MessageQueue* aQueue = (MessageQueue*) malloc(sizeof(MessageQueue));     /* allocate Memory */
        aQueue->size=0;                                                          /* init Size 0 */
        return aQueue;
}

int enqueue(MessageQueue *aQueue, int aValue) {
        //create a new Node
        Node* newNode = (Node*) malloc(sizeof(Node));    /* allocate Memory */
        newNode->data = aValue;                          /* set the data of the Node */
        newNode->next = NULL;                            /* newNode is the last Element (no next) */

        if (aQueue->size == 0) {                         /* check if the Queue is empty */
                aQueue->head=newNode;                    /* if the Queue is empty set the HEAD */
        } else {
                aQueue->rear->next = newNode;            /* if the Queue is not empty set the next node */
        }

        aQueue->rear=newNode;                            /* set the new Node as the new REAR */
        aQueue->size++;                                  /* increase the size of the Queue */
        return aValue;
}

int dequeue(MessageQueue *aQueue) {
        int tmpData;
        Node* removeNode;

        while (aQueue->size == 0) {             /* wait if the queue is empty --> DEAD LOCK !!! */
                printf("Queue is empty...\n");
                Sleep(2000);
        }

        tmpData = aQueue->head->data;           /* get Data */
        removeNode = aQueue->head;              /* temp var. used for free() memory */
        aQueue->head = aQueue->head->next;      /* remove Node from Queue */

        free(removeNode);                       /* release memory */
        aQueue->size--;                         /* decrease Queue size */
        return tmpData;
}
```