

KNr.

MNr.

Zuname, Vorname

Ges.)(80)

1.)(30)

2.)(20)

3.)(30)

Zusatzblätter:

Bitte verwenden Sie nur dokumentenechtes Schreibmaterial!

## 1 C und Makefile

### 1.1 C Fragen

Was ist der Unterschied zwischen Linken und Compilieren? Warum ist es sinnvoll diese Vorgänge zu trennen?

Wozu benötigt man Libraries?

Was ist der Unterschied zwischen Werte- und Variablenparametern?

Wie erfolgt die Umsetzung von Werte- und Variablenparametern in C? Geben Sie jeweils ein Beispiel an!

Erklären Sie mit einer Skizze welche Informationen am Stack bei Werte- und Variablenparametern abgelegt werden.

Wozu wird das Schlüsselwort `static` verwendet? Wann sollten Sie es nicht verwenden?



Welche Informationen werden in Headerdateien abgelegt und was gehört in die Implementierungsdateien?



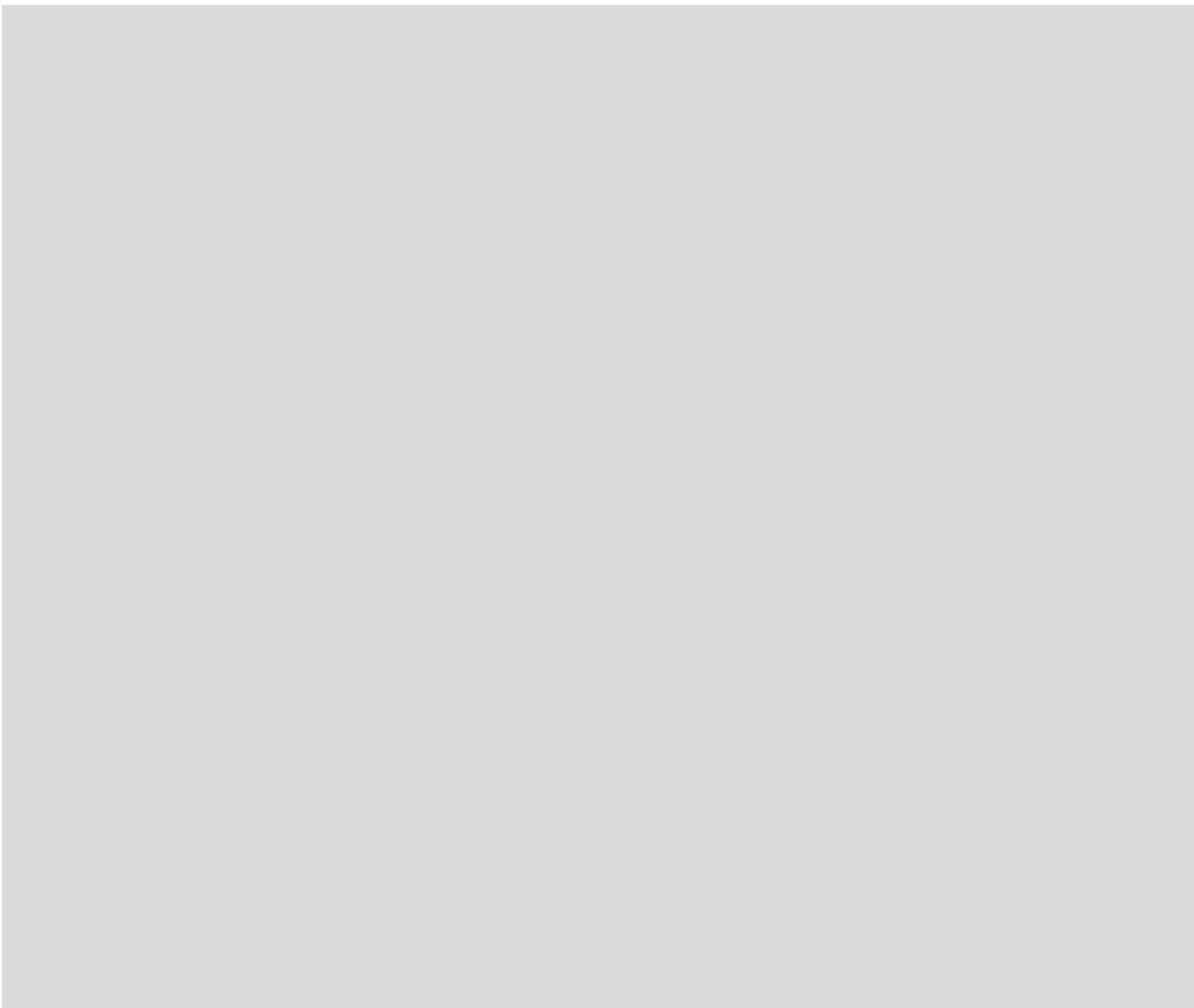
Wie können Sie mit dem Präprozessor das mehrfache Einbinden von Headerdateien verhindern? Erklären Sie dies anhand eines Beispiels!

Nennen Sie 3 Einsatzzwecke von Funktionspointern?

Erstellen Sie Beispielcode für Funktionspointer. Definieren Sie zuerst einen Funktionspointer names `pFunc` auf eine Funktion, welche einen Rückgabewert vom Typ `int` zurückliefert und einen `char*` als Argument erwartet. Lassen Sie `pFunc` auf die Funktion `strlen` aus der Standardlibrary zeigen, und rufen Sie Funktion anschließend über `pFunc` auf um die Länge des Strings `teststring` zu ermitteln.

## 1.2 Makefile

Erstellen Sie das Makefile für ein Programm `mytool` welches aus 2 Modulen besteht mit Sourcedateien `module1.c` und `module2.c` sowie zugehörigen Headerdateien `module1.h` und `module2.h`. Gehen Sie davon aus, dass Funktionalität von Modul 1 in Modul 2 verwendet wird und umgekehrt! Vergessen Sie nicht auf das Target `clean`!



## 2 Argumentbehandlung (20)

Implementieren Sie die Argumentbehandlung für das Programm *mirror*, welches eine Webseite in das aktuelle Verzeichnis spiegelt. Das Programm *mirror* besitzt die folgende Aufrufsyntax:

*mirror* [-l] [-t *timeout*] [-m *maximum*] [-f *filetype*] *url*

- l Diese Option bewirkt, dass Links, welche sich nicht auf dem Zielsystem befinden, nicht ausgeführt werden.
- t *timeout* Spezifiziert ein ganzzahliges Timeout in Sekunden (im Dezimalsystem).
- m *maximum* Spezifiziert ein maximales ganzzahliges Downloadvolumen in Megabyte (im Dezimalsystem).
- f *filetype* Spezifiziert, dass nur Files eines bestimmten Filetypes gespeichert werden sollen (z.B. jpg, pdf, doc). Es handelt sich dabei um eine beliebige Zeichenkette.
- url* Die Angabe der Zieladresse erfolgt im Argument *url*. Da das Programm *mirror* nur HTTP unterstützt, müssen Sie überprüfen, ob die angegebene Adresse mit der Zeichenkette `http://` beginnt. Verwenden Sie dazu die Funktion `strncmp()` mit entsprechender Fehlerbehandlung.

Beispiele für gültige Aufrufe:

```
./mirror http://www.mywebpage.at
./mirror -t 5 http://www.mywebpage.at
./mirror -t 2 -m 50 http://www.mywebpage.at
```

Beispiele für ungültige Aufrufe:

```
./mirror http://www.mywebpage.at
./mirror -l -t 5 -m 20
./mirror -t 2 -m 50 ftp://www.mywebpage.at
```

Implementieren Sie die Argumentbehandlung für die oben erklärten Optionen. Bei der Implementierung sind folgende Punkte zu beachten:

- Die Anzahl und das Format der Argumente sind zu überprüfen. Verwenden Sie dazu die Funktion `int getopt(int argc, char *const argv[], const char *optstr)`.
- Bei ungültigen Optionen bzw. ungültigen Kombinationen der Optionen ist die Funktion `void usage(void)` zu verwenden.
- Bei mehrmaliger Verwendung von Argumenten ist die Funktion `void usage(void)` aufzurufen.
- Verwenden Sie stets den Pointer `optarg`, um auf die Optionsargumente zuzugreifen.
- Vergessen Sie nicht die Fehlerbehandlung bei Funktionsaufrufen (z.B. bei der Konvertierung von Argumenten mit `strtol()`)!

```

#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>
#include <assert.h>
#include <errno.h>

#define EXIT_FAILURE -1
#define PROTOCOL      "http://"

void usage(void) {
    (void) fprintf(stderr, "Usage: mirror [-l] [-t timeout] [-m maximum]
[-f filetype] url \n");
    exit(EXIT_FAILURE);
}

int main(int argc, char **argv) {

/* Declaration and initialization of variables */

```

```

while ((c = getopt(argc, argv, "l:t:m:f:")) != -1) {

    switch (c) {

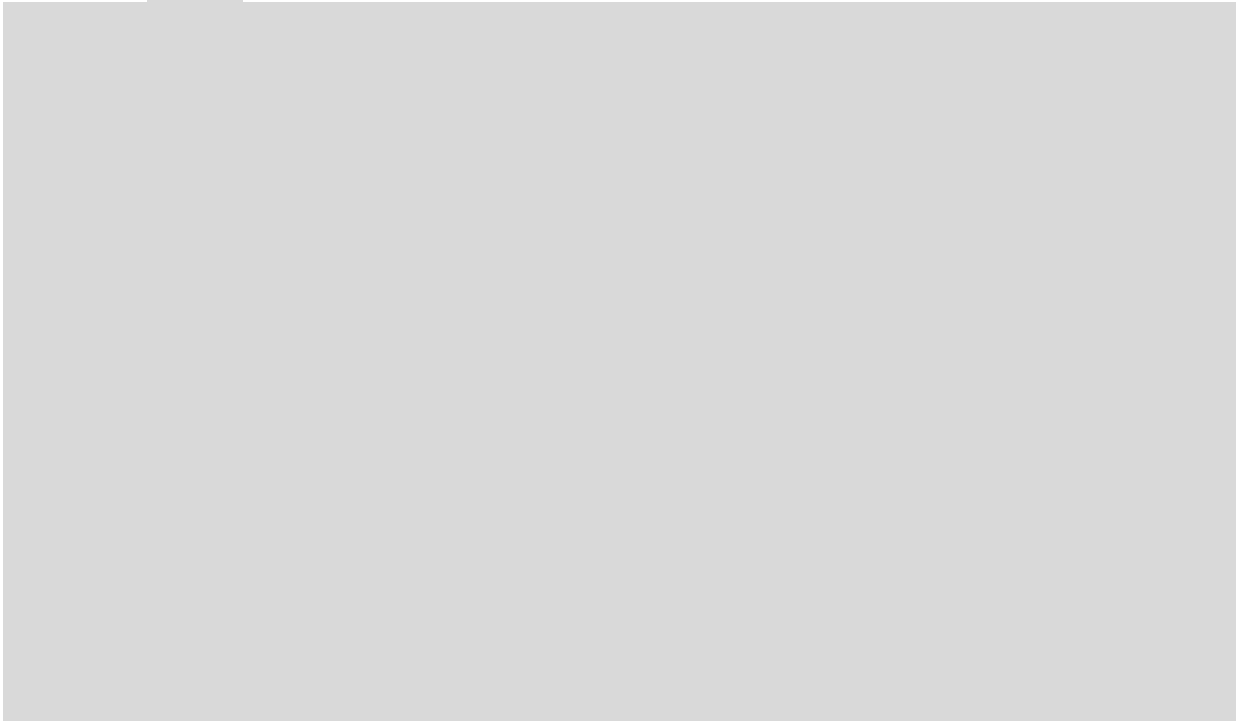
        case 'l':

```

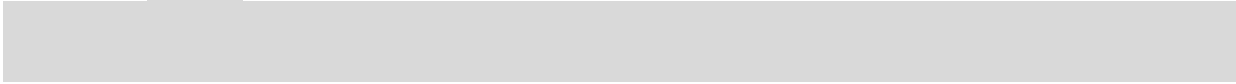
case :



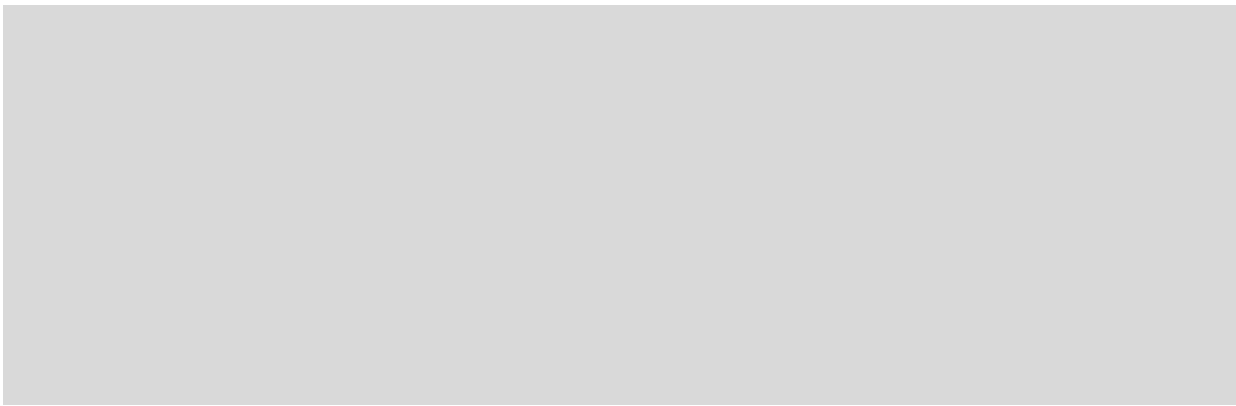
case :



case :

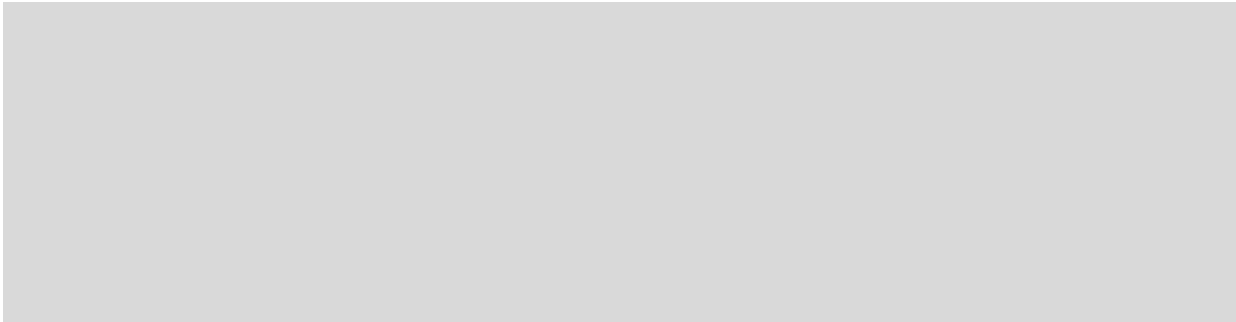




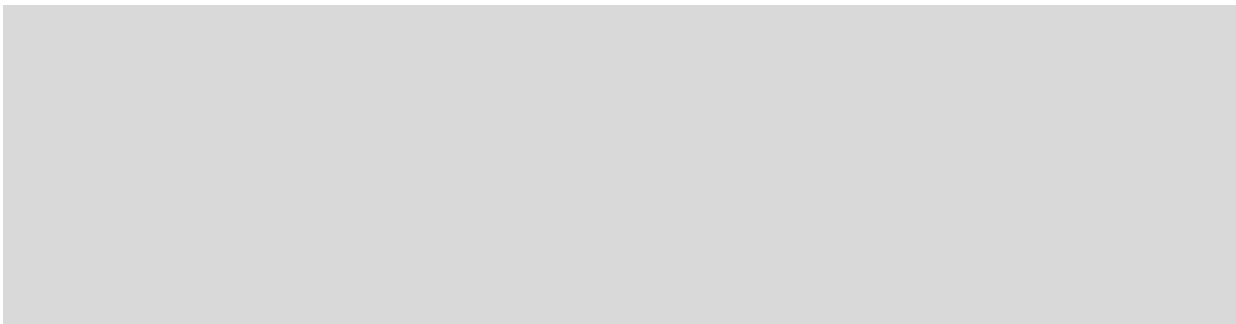


```
    case '?':
        usage();
        break;
    default:
        assert(0);
    }
}

/* Check for correct number and multiple use of arguments */
```



```
/* Check for correct protocol type as specified in PROTOCOL */
```



```
/* ..... */

return 0;
}
```

### 3 Message Queues und Named Pipes (30)

Die Tanzschule “Swing Swing” beauftragt Sie, ein simples Verwaltungssystem für eine Tanzpartnerbörse zu erstellen. Dabei sollen Frauen und Männer die Möglichkeit haben, ihr Interesse an einem Tanzpartner bei einem zentralen Server anzumelden. Die Anmeldungen sollen in einer Warteliste eingetragen, und in der Reihenfolge ihres Eingangs abgearbeitet werden. Das bedeutet, dass jeweils die erste Dame in der Warteliste mit dem ersten Herrn in der Warteliste ein Paar bilden soll. Ist ein Paar gebildet, so sollen der entsprechende Herr und die entsprechende Dame aus der Warteliste wieder entfernt werden.

Die Aufrufsyntax des Anmeldeprogramms soll wie folgt aussehen:

```
anmeldung {-f | -m } Vorname Nachname
```

`anmeldung` muss also entweder mit dem Parameter `-f` (für weiblich) oder mit dem Parameter `-m` (für männlich) aufgerufen werden. Vor- und Nachname können jeweils maximal 40 Zeichen betragen. Da das Programm `anmeldung` von einem Web-Interface automatisiert aufgerufen wird, können Sie davon ausgehen, dass die übergebenen Argumente korrekt sind und eine Überprüfung der Argumente daher nicht notwendig ist.

Der Aufruf `anmeldung` bewirkt, dass die angegebene Person in die Warteliste eingetragen wird.

Das zentrale Server-Programm `server` wird ohne Argumente gestartet. Die Warteliste für Damen und Herren soll mit einer *einzigsten* Message Queue realisiert werden, die exklusiv vom Server angelegt wird.

Der Server soll die Queue abarbeiten, und wann immer ein Paar gebildet werden kann (also wann immer mindestens eine Dame und ein Herr in der Queue registriert sind), dieses bilden und es in folgender Form auf `stdout` ausgeben:

```
Neues Paar: Vorname-Herr Nachname-Herr UND Vorname-Dame Nachname-Dame
```

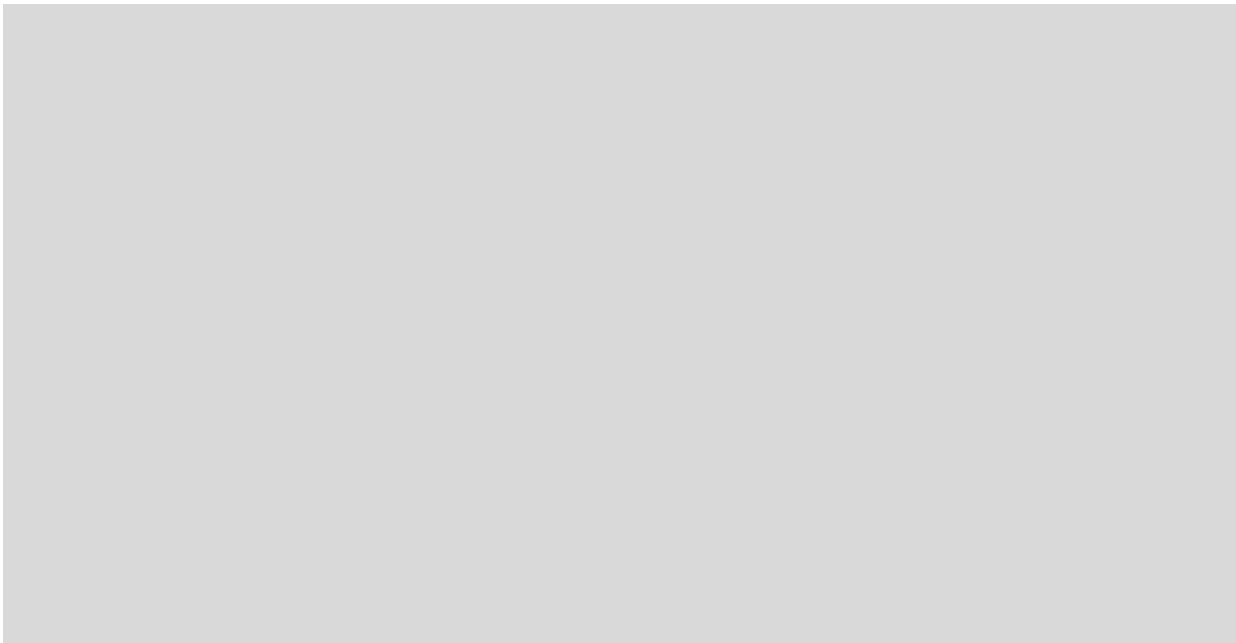
Lösen Sie das Beispiel entsprechend den Richtlinien der LU! Implementieren Sie dazu die im Text beschriebene Funktionalität in den Files `header.h`, `anmeldung.c` und `server.c`.

#### header.h

**Notiz von @sueszli: "Message Queue" und "Named Pipes" sind nicht mehr Teil der LVA (Stand 2023)**



## anmeldung.c

```
#include everything  
#include "header.h"  
int main (int argc, char **argv)  
{  

```



```
}
```

## **server.c**

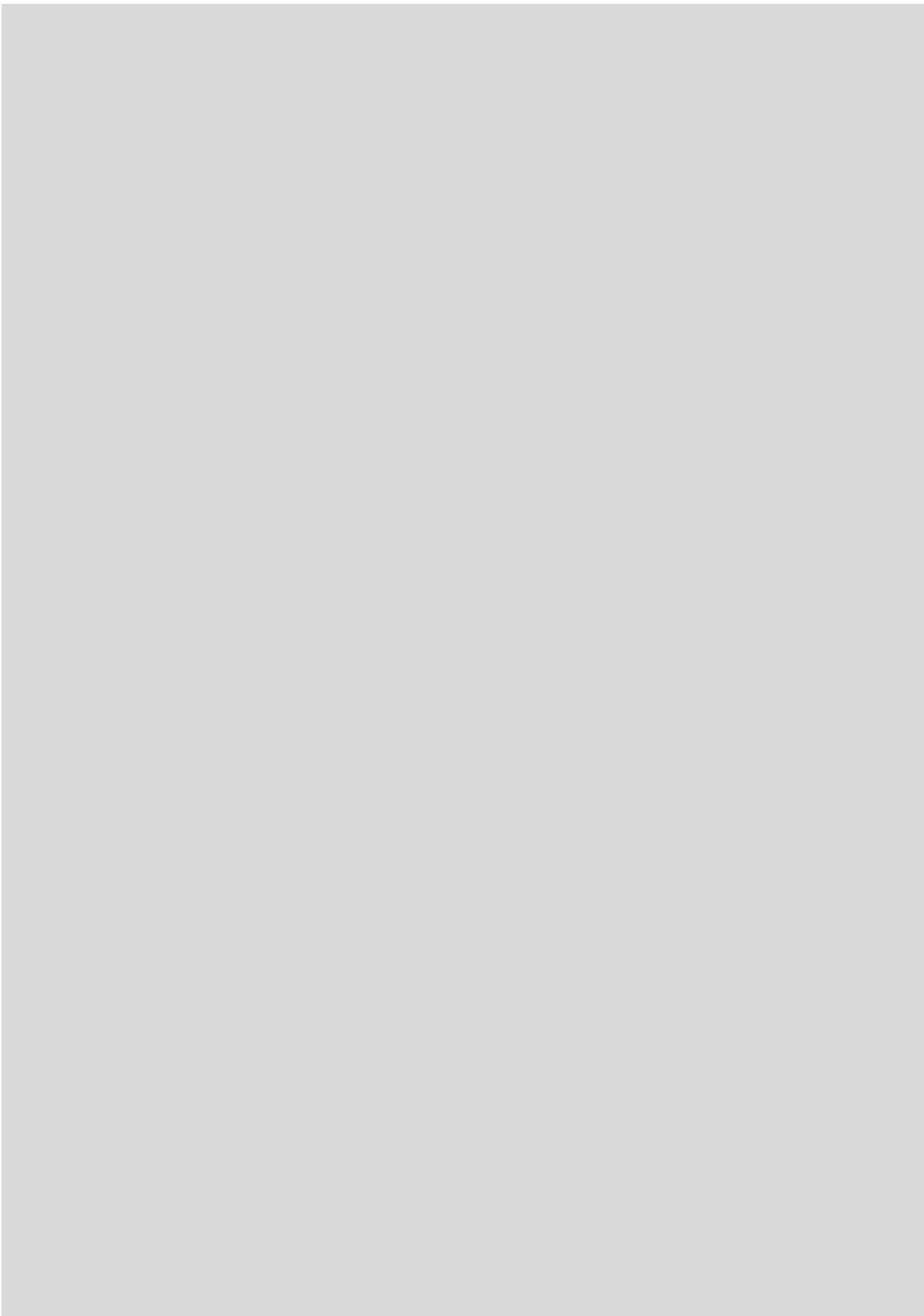
```
#include everything
```

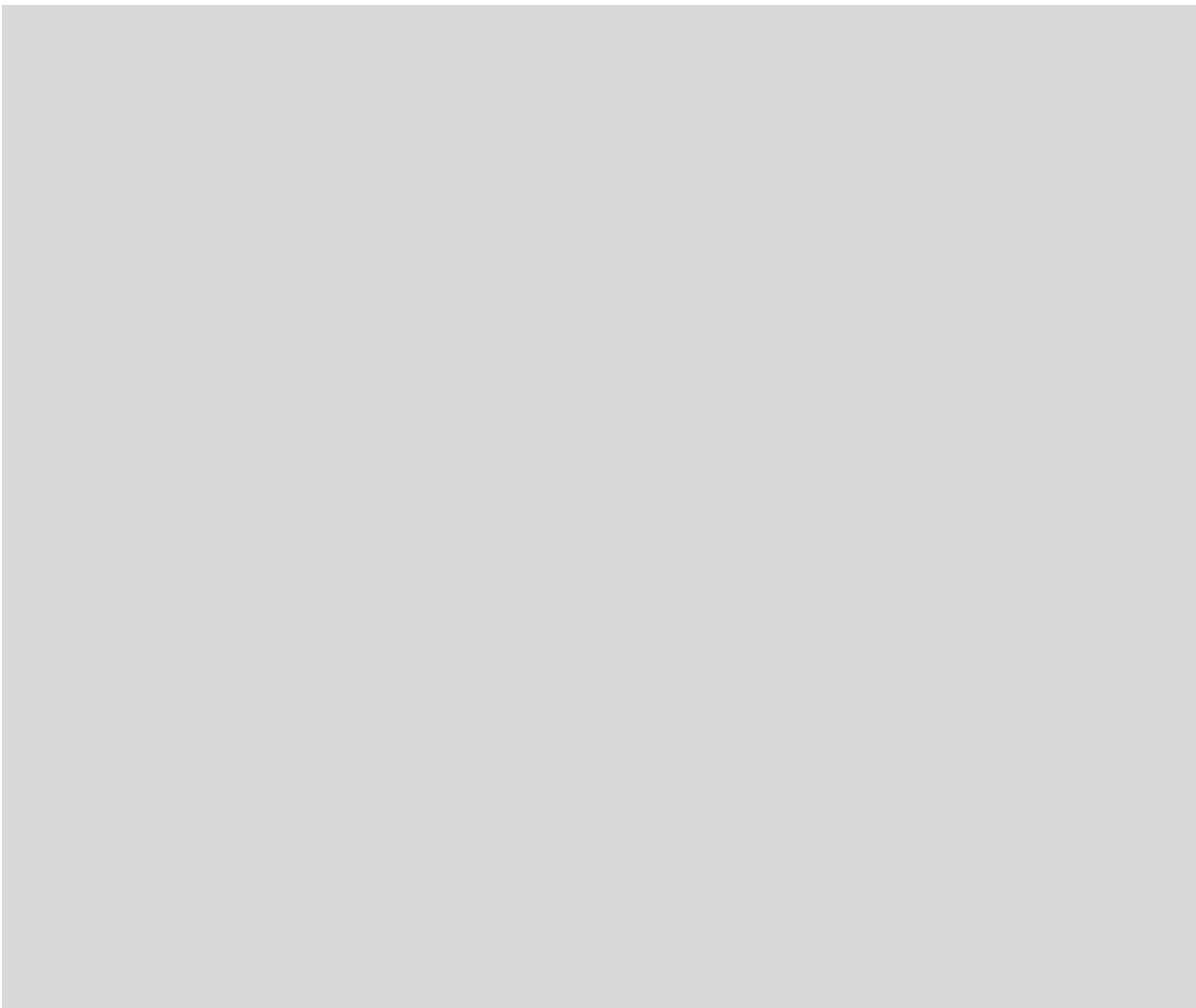
```
#include "header.h"
```

```
int main (int argc, char **argv)
```

```
{
```







}