a).

```
CreateDir.bash
  0 # !/ bin / bash
  1 # cli - args . bash
  2
  3 $(mkdir $1)
```

```
Linok-2 :: Desktop/Embedded Des Enabling Robotics/lab4 » ./CreateDir.bash aaaaaaaaaaaaaaaaaaaaa
Linok-2 :: Desktop/Embedded Des Enabling Robotics/lab4 » ls
CreateDir.bash          Makefile              ShowDate.bash          ZedBoard.h              aaaaaaaaaaaaaaaaaaaaa zedMain.o
Lab-4_Assignment.pdf    Pre-Lab4.odt          ZedBoard.cpp           ZedBoard.o              zedMain.cpp           zedboard
```

b).

```
ShowDate.bash
  1 # !/ bin / bash
  0
  1 echo "The current date is $(date)"
~
```

```
Linok-2 :: Desktop/Embedded Des Enabling Robotics/lab4 » ./ShowDate.bash
The current date is Sat Feb 20 18:42:51 EST 2016
```

c).

```
1 Makefile  1 zedMain.cpp | 1 ZedBoard.cpp | 1 ZedBoard.h
  0 GCC = g++
  1 CFLAGS = -g -Wall
  2 OBJS = zedMain.o ZedBoard.o
  3 EXE = zedboard
  4
  5 $(EXE): $(OBJS)
  6     $(GCC) $(OBJS) -o $(EXE)
  7
  8 zedMain.o: zedMain.cpp ZedBoard.h
  9     $(GCC) $(CFLAGS) -c zedMain.cpp
 10
 11 ZedBoard.o: ZedBoard.cpp ZedBoard.h
 12     $(GCC) $(CFLAGS) -c ZedBoard.cpp
 13
 14 clean:
 15     rm $(OBJS) $(EXE)
~
```

```
#ifndef ZEDBOARD_H
#define ZEDBOARD_H

// Class Definition
class ZedBoard {
    private:
        char *pBase;    // virtual address where I/O was mapped
        int fd;         // file descriptor for dev memory
        int dummyValue; // for testing without a Zedboard
    public:
        ZedBoard();     // Default Constructor
        ~ZedBoard();    // Destructor
        void RegisterWrite(int offset, int value);
        int RegisterRead(int offset);
        void Set1Led(int ledNum, int state);
        void SetLedNumber(int value);
};

#endif
```

1 Makefile 1 zedMain.cpp 1 ZedBoard.cpp | 1 ZedBoard.h

```
21 #include <iostream>
20 #include "ZedBoard.h"
19
18 /**
17  * Operates the Zedboard LEDs and switches
16  */
15 int main()
14 {
13     // Initialize
12     ZedBoard *zed = new ZedBoard();
11
10     int value = 0;
 9     std::cout << "Enter a value less than 256: ";
 8     std::cin >> value;
 7     std::cout << "value entered = " << value << std::endl;
 6
 5     // Show the value on the Zedboard LEDs
 4     zed->SetLedNumber(value);
 3     delete zed;
 2     // Done
 1     return 0;
 0 }
```

Note: Globals commented out because they are currently unused.

```
 0 #include <sys/mman.h>
 1 #include <iostream>
 2 #include "ZedBoard.h"
 3
 4 // Physical base address of GPIO
 5 // const unsigned gpio_address = 0x400d0000;
 6
 7 // Length of memory-mapped IO window
 8 const unsigned gpio_size = 0xff;
 9
10 const int gpio_led1_offset = 0x12C;  // Offset for LED1
11 // const int gpio_led2_offset = 0x130;  // Offset for LED2
12 // const int gpio_led3_offset = 0x134;  // Offset for LED3
13 // const int gpio_led4_offset = 0x138;  // Offset for LED4
14 // const int gpio_led5_offset = 0x13C;  // Offset for LED5
15 // const int gpio_led6_offset = 0x140;  // Offset for LED6
16 // const int gpio_led7_offset = 0x144;  // Offset for LED7
17 // const int gpio_led8_offset = 0x148;  // Offset for LED8
18
19 // const int gpio_sw1_offset = 0x14C;  // Offset for Switch 1
20 // const int gpio_sw2_offset = 0x150;  // Offset for Switch 2
21 // const int gpio_sw3_offset = 0x154;  // Offset for Switch 3
22 // const int gpio_sw4_offset = 0x158;  // Offset for Switch 4
23 // const int gpio_sw5_offset = 0x15C;  // Offset for Switch 5
24 // const int gpio_sw6_offset = 0x160;  // Offset for Switch 6
25 // const int gpio_sw7_offset = 0x164;  // Offset for Switch 7
26 // const int gpio_sw8_offset = 0x168;  // Offset for Switch 8
27
28 // const int gpio_pbtnl_offset = 0x16C;  // Offset for left push button
29 // const int gpio_pbtnr_offset = 0x170;  // Offset for right push button
30 // const int gpio_pbtnu_offset = 0x174;  // Offset for up push button
31 // const int gpio_pbtnd_offset = 0x178;  // Offset for down push button
32 // const int gpio_pbtnc_offset = 0x17C;  // Offset for center push button
33
```

```
/**
 * Constructor Initialize general-purpose I/O
 *  - Opens access to physical memory /dev/mem
 *  - Maps memory at offset 'gpio_address' into virtual address space
 *
 * @param  None Default constructor does not need arguments.
 * @return  None Default constructor does not return anything.
 */
ZedBoard::ZedBoard(){
    std::cout << "\nStarting...." << std::endl;
    dummyValue = 99;
    char dummyChar;
    pBase = &dummyChar;
    //fd = open( "/dev/mem", O_RDWR);
    //pBase = (char *) mmap(NULL,gpio_size,PROT_READ | PROT_WRITE,
    //        MAP_SHARED,fd,gpio_address);
    /* Check error */
    if (pBase == MAP_FAILED)
    {
        std::cerr << "Mapping I/O memory failed - Did you run with 'sudo'?\n";
        exit(1); // Returns 1 to the operating system;
    }
}

/**
 * Destructor to close general-purpose I/O.
 * - Uses virtual address where I/O was mapped.
 * - Uses file descriptor previously returned by 'open'.
 *
 * @param  None Destructor does not need arguments.
 * @return  None Destructor does not return anything.
 */
ZedBoard::~ZedBoard(){
    munmap(pBase, gpio_size);
    close(fd);
    std::cout << "\nTerminating...." << std::endl;
}
```

```
/**
 * Write a 4-byte value at the specified general-purpose I/O location.
 *
 * - Uses base address returned by 'mmap'.
 * @parem offset    Offset where device is mapped.
 * @param value     Value to be written.
 */
void ZedBoard::RegisterWrite(int offset, int value)
{
    //* (int *) (pBase + offset) = value;
    dummyValue = value;
}

/**
 * Read a 4-byte value from the specified general-purpose I/O location.
 *
 * - Uses base address returned by 'mmap'.
 * @param offset    Offset where device is mapped.
 * @return      Value read.
 */
int ZedBoard::RegisterRead(int offset)
{
    //return * (int *) (pBase + offset);
    return dummyValue;
}

/**
 * Changes the state of an LED (ON or OFF)
 *
 * - Uses base address of I/O
 * @param ledNum    LED number (0 to 7)
 * @param state State to change to (ON or OFF)
 */
void ZedBoard::Set1Led(int ledNum, int state)
{
    std::cout << "\nWriting to LED " << ledNum << std::endl;
    //RegisterWrite(gpio_led1_offset + (ledNum * 4), state);
    std::cout << "LED offset: " << std::hex << gpio_led1_offset+(ledNum*4) << "    ";
    std::cout << "LED state: " << state << std::endl;
}
```

```
/**
 * Show lower 8 bits of integer value on LEDs
 *
 * - Calls Set1Led() to set all LEDs
 * @param value Value to show on LEDs
 */
void ZedBoard::SetLedNumber(int value)
{
    std::cout << "\nWriting to LEDs...." << std::endl;
    for(int i = 0; i < 8; i++) {// write to all LEDs
        Set1Led(i, (value / (1<<i)) % 2);
    }
}
```

d).

Makefile

```
GCC = g++
CFLAGS = -g -Wall
OBJS = WiimoteBtns.o main.o
EXE = WiimoteBtns

$(EXE): $(OBJS)
    $(GCC) $(OBJS) -o $(EXE)

WiimoteBtns.o: WiimoteBtns.cpp WiimoteBtns.h
    $(GCC) $(CFLAGS) -c WiimoteBtns.cpp

main.o: main.cpp WiimoteBtns.h
    $(GCC) $(CFLAGS) -c main.cpp

clean:
    rm $(OBJS)
```

```
1 Makefile | 1 WiimoteBtns.cpp | 1 main.cpp  1 WiimoteBtns.h
13 #ifndef ZEDBOARD_H
12 #define ZEDBOARD_H
11
10 // Class Definition
 9 class WiimoteBtns {
 8     private:
 7         int fd; // File descriptor
 6     public:
 5         WiimoteBtns();  // Default constructor
 4         ~WiimoteBtns(); // Destructor
 3         void Listen();
 2         void ButtonEvent(int code, int value);
 1 };
 0
 1 #endif
```

```
1 Makefile | 1 WiimoteBtns.cpp  1 main.cpp  1 WiimoteBtns.h
 4 #include "WiimoteBtns.h"
 3
 2 int main()
 1 {
 0     WiimoteBtns wiimote;
 1     wiimote.Listen();
 2
 3     // Close Wiimote event file
 4     return 0;
 5 }
```

1 Makefile 1 WiimoteBtns.cpp 1 main.cpp | 1 WiimoteBtns.h

```
 0 #include <stdlib.h>
 1 #include <fcntl.h>
 2 #include <unistd.h>
 3 #include <iostream>
 4 #include "WiimoteBtns.h"
 5
 6 /**
 7  * Opens /dev/input/event2 and checks for errors.
 8  */
 9 WiimoteBtns::WiimoteBtns() {
10
11     // Open Wiimote event file
12     int fd;
13     fd = open("/dev/input/event2", O_RDONLY);
14
15     if (fd == -1) // Error openining the file descriptor
16     {
17         std::cerr << "Error: Could not open event file - forgot sudo?\n";
18         exit(1);
19     }
20 }
21
22 /**
23  * Closes the file descriptor
24  */
25 WiimoteBtns::~WiimoteBtns() {
26     close(fd);
27 }
```

```
32
31 /**
30  * Enters an infinite loop where a new event is read from the virtual file.
29  * When an event is ready, its associated code and value fields are passed
28  * in an invocation to ButtonEvent().
27  */
26 void WiimoteBtns::Listen() {
25     while (true)
24     {
23         // Read a packet of 32 bytes from Wiimote
22         char buffer[32];
21         read(fd, buffer, 32);
20
19         // Extract code (byte 10) and value (byte 12) from packet
18         int code = buffer[10];
17         int value = buffer[12];
16
15         this->ButtonEvent(code, value);
14     }
13 }
12
11 /**
10  * Displays a message on the screen reporting two values.
 9  *
 8  * @param code, a number showing the code
 7  * @param value, a number showing the value
 6  */
 5 void WiimoteBtns::ButtonEvent(int code, int value) {
 4
 3     // Print them
 2     std::cout << "Code = " << code << ", value = " << value << '\n';
 1
 0 }
```

e).

```
1 Makefile  1 WiimoteAccel.h | 1 WiimoteAccel.cpp | 1 main.cpp
15 GCC = g++
14 CFLAGS = -g -Wall
13 OBJS = WiimoteAccel.o main.o
12 EXE = WiimoteAccel
11
10 $(EXE): $(OBJS)
 9     $(GCC) $(OBJS) -o $(EXE)
 8
 7 WiimoteBtns.o: WiimoteAccel.cpp WiimoteAccel.h
 6     $(GCC) $(CFLAGS) -c WiimoteBtns.cpp
 5
 4 main.o: main.cpp WiimoteAccel.h
 3     $(GCC) $(CFLAGS) -c main.cpp
 2
 1 clean:
 0     rm $(OBJS) $(EXE)
```

```
1 Makefile  1 WiimoteAccel.h  1 WiimoteAccel.cpp | 1 main.cpp
 3 #ifndef ZEDBOARD_H
 2 #define ZEDBOARD_H
 1
 0 // Class Definition
 1 class WiimoteAccel {
 2     private:
 3         int fd; // File descriptor
 4     public:
 5         WiimoteAccel(); // Default constructor
 6         ~WiimoteAccel(); // Destructor
 7         void Listen();
 8         virtual void AccelerationEvent(int code, int acceleration);
 9 };
10
11 #endif
~
```

1 Makefile | 1 WiimoteAccel.h 1 WiimoteAccel.cpp 1 main.cpp

```
 0 #include "WiimoteAccel.h"
 1 #include <iostream>
 2 #include <fcntl.h>
 3
 4 /**
 5  * A class constructor opens /dev/input/event0 (instead of /dev/input/event2 as
 6  * before) and checks for errors.
 7  */
 8 WiimoteAccel::WiimoteAccel()
 9 {
10     fd = open("/dev/input/event0", O_RDONLY);
11     if (fd == -1) // Error opening the file descriptor
12     {
13         std::cerr << "Error: Could not open event file - forgot sudo?\n";
14         exit(1);
15     }
16 }
17
18 /**
19  * A class destructor closes the file.
20  */
21 WiimoteAccel::~WiimoteAccel()
22 {
23     // Close Wiimote event file
24     close(fd);
25 }
26
```

```
31
30 /**
29  * A public function called Listen() enters an infinite loop where a new
28  * acceleration event is read from the virtual file. When found, the associated
27  * code and acceleration values are passed to a public function called
26  * AccelerationEvent().
25  */
24 void WiimoteAccel::Listen() {
23     while (true)
22     {
21         // Read a packet of 16 bytes from Wiimote
20         char buffer[16];
19         read(fd, buffer, 16);
18
17         // Extract code (byte 10) and value (byte 12) from packet
16         int code = buffer[10];
15         short acceleration = * (short *) (buffer + 12);
14
13     }
12 }
11
10 /**
 9  * A public function called AccelerationEvent() takes a code and acceleration
 8  * as integer arguments, and displays a message on the screen reporting these
 7  * two values.
 6  */
 5 void WiimoteAccel::AccelerationEvent(int code, int acceleration)
 4 {
 3     // Print them
 2     std::cout << "Code = " << code <<
 1         ", acceleration = " << acceleration << '\n';
 0 }
```

```
#include <stdlib.h>
#include <unistd.h>
#include "WiimoteAccel.h"

int main()
{

    WiimoteAccel wiimoteAccel;
    wiimoteAccel.Listen();

    return 0;
}
```