```
#include "usb.h"
#include "usb_host_hid.h"
#include "system_usb.h"
#include "app_host_hid_keyboard.h"

#include "text.h"

#include "init_global.h"
#include "main.h"

#include <pps.h>
#include <stdint.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#include <string.h>
#include <assert.h>


#define KB_SIZE     80


#define MAX         100
#define SHORT       10
#define VERYSHORT   3


/* Pointer characters and searches */



T2IP = 7;
OC4IP = 7;
OC3IP = 7;
          //void uitoa(WORD Value, BYTE* Buffer);

enum disk_command {clear, open_dir, open_file, write_file, scroll, dict, index, data, senses, extract,
print, parse_index1};
enum disk_goto {blank, read_dir, close_dir, read_file, close_file, create_file, read_file1};









char *response[] = {
```

```
    "How are you this beautiful day?",
    "Did you have a happy childhood?",
    "Did you hate your father?",
    "I'm not sure I understand.",
    "Tell me about your mother..",
    ""
};

char *trans[] = {
    "you","Let's not talk about me.",
    "think","Why do you think that?",
    "hate","So you hate something - tell me more.",
    "what","Why do you ask?",
    "want","Why do you want that?",
    "need","We all need many things - is this special?",
    "why","Remember, therapy is good for you",
    "know","How do you know that?",
    "bye","Your bill will be mailed to you.",
    "murder","I don't like killing.",
    "kill","It is wrong to kill.",
    "jerk","Don't ever call me a jerk",
    "can't","Don't be negative - be positive",
    "failure","Strive for success.",
    "never","Don't be negative - be positive",
    "unhappy","Why are you unhappy?",
    ""
};

char topics[MAX][40]; /*holds old topics*/

char token[80];
char *p_pos;
int res = 0; // index of response array
int head=0;
int tail=0;



//print flags
char init_res = 0;
char spare_res = 0;

void PIN_MANAGER_Initialize (void);
int getKeyCode (char *c);
int getLineCode (char *c);


int main(void)
{

PIN_MANAGER_Initialize ();
```

```
SYSTEM_In(SYSTEM_STATE_USB_HOST);
SYSTEM_In(SYSTEM_STATE_USB_HOST_HID_KEYBOARD);
//Initialize the stack
USBHostInit(0);
APP_HostHIDKeyboardInitialize();

USBHostTasks();
USBHostHIDTasks();
APP_HostHIDKeyboardTasks();

InitVideo();    // start the state machines
Clrscr();



 char KeyBuffer [ KB_SIZE]; // declar keyboard buffer


volatile int KBR, KBW; // head and tail or write and read pointers
KBR = 0;
KBW = 0;

unsigned char LineBuffer [ KB_SIZE]; // declar keyboard buffer
volatile int LR, LW; // head and tail or write and read pointers
LR = 0;
LW = 0;




memset(KeyBuffer, 0, sizeof KeyBuffer); // reset the array buffer at start up!! leave this here
 // for these, see PIC32, capturing inputs, keyboar

// Definitions
       int getKeyCode (char *c)
      {
         if ( KBR == KBW)
         return FALSE;// else buffer contains at least one key code
         *c = KeyBuffer[KBR++]; // extract the first key code
         KBR %= KB_SIZE; //wrap around the pionter
         return TRUE;
      } // end getKeyCode

       int getLineCode (char *c)
      {
         if ( LR == LW)
         return FALSE; // else buffer contains at least one key code
         *c = LineBuffer[LR++]; // extract the first key code
         LR %= KB_SIZE; //wrap around the pionter
         return TRUE;
```

```
    } // end getKeyCode


char s[80];

putsV(response[res++]);
cx=0;
cy++;




do
{
USBHostTasks();
USBHostHIDTasks();
APP_HostHIDKeyboardTasks();

if(init_res == 0)// how to wait here? use a flag
{
init_res = 1;
cx=0;
cy++;
putsV(": ");
}

p_pos = s;



    // reading keys
  if(global_var_1)// write to the buffer
  {
    KeyBuffer [ KBW] = global_var_1; // write in
    if(( KBW+1)%KB_SIZE !=KBR)       // check if buffer full
    KBW++; // else increment pointer
    KBW %= KB_SIZE; // wrap around
    global_var_1 = 0;
  }


     if(enter == 1) // ENTER IS PRESSED AFTER STRING INPUT
     {
        global_var_1 = 0;
        enter = 0;
        getKeyCode(KeyBuffer);
        strcpy(s,KeyBuffer);

             init_res =0;
             Clrscr();
```

```
            Home();

            respond(s);

            memset(KeyBuffer, 0, sizeof KeyBuffer); // reset the array buffer
            KBR = 0; // must reset these
            KBW = 0; // must reset these, or, buffer is spoiled

}

} while(strcmp(s,"bye"));

}//end main


/*create the doctor's responses*/

respond(char *s)
{
   char t[80];
   int loc;

   if(strlen(s)<VERYSHORT && strcmp(s,"bye"))
   {
      if(find_topic(t))
      {
         putsV("You just said: ");
         putsV(t);
         putsV(", ");
         putsV("tell me more.");
      }
      else
      {
         if (!*response[res]) res=0; // start over
         putsV(response[res++]);
      }
      return;
   }

   if(in_topics(s))
   {
      putsV("Stop repeating yourself!");
      cx=0;
      cy++;
      return;
   }

   if(strlen(s)>SHORT) assert_topic(s);

   do
   {
```

```
      get_token();
      loc=lookup(token);
      if(loc!=-1)
      {
         putsV(trans[loc+1]);
         return;
      }
   }while(*token);
   //comment of last resort



   putsV("Tell me more...");
   cx=0;
   cy++;

}



//lookup a keyword in trans table
lookup(char *token)
{
   int t;
   t=0;
   while(*trans[t])
   {
      if(!strcmp(trans[t],token)) return t;
      t++;
   }
   return -1;
}

//place a topic into the topics dbase
assert_topic(char *t)
{
   if(head==MAX)head =0; // wrap around
   strcpy(topics[head],t);
   head++;
}



/* retrieve a topic*/
find_topic(char *t)
{
   if(tail!=head)
   {
      strcpy(t,topics[tail]);
      tail++;
```

```
      /*wrap around if necessary*/
      if(tail==MAX) tail = 0;
      return 1;
   }
   return 0;
}


//see if in topics queue
in_topics(char *s)
{
   int t;
   for(t=0;t<MAX;t++)
      if(!strcmp(s,topics[t])) return 1;
   return 0;
}


//retrn a token from the input stream
get_token()
{
   char *p;

   p=token;
   //skip spaces
   while(*p_pos==' ')p_pos++;

   if(*p_pos=='\0')
   {
      *p++='\0';
      return; // end of input
   }
   if(is_in(*p_pos,".!?"))
   {
      *p=*p_pos;
      p++, p_pos++;
      *p='\0';
      return;
   }

// read word until
   while(*p_pos!=' ' && !is_in(*p_pos,".,;?!") && *p_pos)
   {
      *p=tolower(*p_pos++);
      p++;
   }
   *p='\0';
}

is_in(char *c, char *s)
{
```

```
    while(*s)
    {
        if(c==*s) return 1;
        s++;
    }
    return 0;
}
```