密码库 自己的

Your objective it to design the software to meet the specification -  
preferably writing C++ using classes available. The native Mbed OS  
classes can be found by going to:

你的目标是设计软件以满足规范要求

最好使用可用的类编写c++。本机Mbed操作系统

类可以通过以下方式找到:  
  
[https://os.mbed.com/docs/mbed-os/v6.16/apis/index.html](https://os.mbed.com/docs/mbed-os/v6.16/apis/index.html" \t "https://mail.google.com/mail/u/0/" \l "search/n.seed%40sheffield.ac.uk/_blank)  
  
Each class available is described here. To access any of these you just  
need **#include "mbed.h"** at the top of a cpp file. You are directed,  
particularly, to the **Input/Output Driver** classes and the **Time** classes.  
For example, to create a **digital output associated** with a port pin, you  
need to create an object of class **DigitalOut** e.g.  
  
DigitalOut mypin(<pinname>);  
  
Where valid pinnames can be found in  
[https://os.mbed.com/teams/Freescale/wiki/frdm-kl46z-pinnames](https://os.mbed.com/teams/Freescale/wiki/frdm-kl46z-pinnames" \t "https://mail.google.com/mail/u/0/" \l "search/n.seed%40sheffield.ac.uk/_blank)  
  
So to output to port A1 would be:  
  
DigitalOut mypin(PTA1);  
  
Once defined mypin can be used like a variable e.g.  
  
// set output to 0  
mypin = 0;  
  
//invert the output  
mypin = !mypin;  
  
etc.  
  
Another class that is useful is **Ticker** this allows you to call a  
function at regular intervals e.g.  
  
Ticker mytick;  
  
void MyFunc(void)  
{  
....  
}  
  
...  
  
void main(void)  
{  
  
     mytick.attach(MyFunc, 0.03);  
  
     ... other code here]  
  
}  
  
Once set up, MyFunc is called very 30ms (until detached). Note, MyFunc  
is a void/void function - you cannot pass parameters in/out - its called  
automatically. However, it can update data that is shared with main()  
for example. This is really useful for running code at regular intervals  
without affecting the program's ability to continue executing elsewhere.

一旦设置，MyFunc被称为每30毫秒(直到分离)。注意，MyFunc是一个void/void函数-你不能传入/传出参数-它会自动调用。但是，它可以更新与main()共享的数据。这对于定期运行代码而不影响程序在其他地方继续执行的能力非常有用。  
  
Additionally, if you click on Code at the top of:  
  
[https://os.mbed.com/docs/mbed-os/v6.16/apis/index.html](https://os.mbed.com/docs/mbed-os/v6.16/apis/index.html" \t "https://mail.google.com/mail/u/0/" \l "search/n.seed%40sheffield.ac.uk/_blank)  
  
  
copy the http link [https://os.mbed.com/users/Sissors/code/SLCD/](https://os.mbed.com/users/Sissors/code/SLCD/" \t "https://mail.google.com/mail/u/0/" \l "search/n.seed%40sheffield.ac.uk/_blank)  
  
 From within Mbed Studio, right-click on your program in the LH pane and  
select Add Library ...  
In the dialogue that opens up, paste the URL into Git or [os.mbed.com](http://os.mbed.com/" \t "https://mail.google.com/mail/u/0/" \l "search/n.seed%40sheffield.ac.uk/_blank) URL  
and click next on branch/tag select default and finish.  
在Mbed Studio中，右键单击你的程序，在LH窗格中选择Add Library…在打开的对话框中，将URL粘贴到Git或os.mbed.com URL中，然后单击下一步分支/标签选择默认并完成。  
Now you will see an SLCD folder in your program.  
  
To use it just #include "SLCD.h" in you program and declare  
  
SLCD disp;  
  
for example.  
  
To write to the SLCD you can just use:  
  
SLCD.printf("abcd");  
  
(remember there are **only 4 digits**)  
  
SLCD inherits from the standard stream library and so functions like  
printf() now appear to be members of SLCD and can be accessed via  
SLCD.print())

1. w.r.t the **FPU error**.浮点运算  
     
   There is a file under  
   mbed-os/cmiss5/cmsis/TARGET\_CORTEX\_M/include/core\_cm0plus.h (and core\_cm0.h)  
     
   If you look at core\_cm0plus.h then it defines \_\_FPU\_USED 0 - which  
   indicates that the Cortext CM0 and CM0+ do not have a HW floating point  
   processor (FPU) in them. It then goes on to check defines for various  
   compilers e.g. \_ARM\_CC\_VERSION=6010050, \_GNUC\_ and generates the error  
   in cases where it thinks the compiler will be generating FPU  
   instructions to do calculations on floats (rather than implementing them  
   directly using ARM code i.e. assuming that there is no FPU.  
     
   Now given that this is an ARM compiler and ARM knows that the CM0+  
   (which is what is inside the processor) does not support an FPU it is  
   reasonable to assume that it will actually generate the code for FPU  
   instructions in software. Check this is you like:  
     
   void main() {  
     
   float a, b;  
     
     a = 3.7;  
     
     b = 4.3;  
     
     printf("%f\n", a \*b);  
     
     while (true) ();  
     
   }  
     
   Use View-Serial Monitor in MBed Studio to open a serial monitor (which  
   is where the printf output will appear) and see what happens.

3

Hopefully you should have identified that:

**Erik / SLCD and Kenji Arai / Keypad**

**https://os.mbed.com/search/repository?q=keypad**

are classes that it would make sense to use in your software.

I would also hope that you've done some division of labour and that each person in the group will have been planning out what their part of the code will do and how they might expect other people to access and use their code **(interfaces are key**, here).

You should try to ensure that each part of the code is written in a separate cpp file with an associated h file that will allow other people to access the code simply. I don't necessarily expect you to write your code as a class (it would be helpful of course) but the code you write should be functionally cohesive - if you're writing code to handle passwords then do just that - don't conflate this with other parts of the overall code - clean interfaces between code work best. So, for example if you were writing code to handle password(s) then you might, within your code:

define one or more arrays to hold current passwords:

have a function to allow an array holding a password entered to be submitted for comparison against the passwords held with a return value identifying if it found a match.

have a function to allow an array holding a password to be submitted and used to replace one of the passwords held.

One thing I should say is that you should always use parameters to pass values into functions - do not use globally shared variables - C supports parameter passing for really good reasons.

Obviously, if you need to store variables that are persistent (i.e they outlive a function call) then these do need to be declared outside of functions but they should be used carefully. **So, looking at our password example**. In the corresponding cpp file we might have:

char arrays holding passwords;

functions that allow users to access (read/update) passwords - these functions perform checks to ensure that users only access the passwords in controlled ways;

In the associated h file, only the functions users can actually call have external declarations. The names of data variables (e.g. passwords) are not included and cannot, therefore, be accessed by code elsewhere. This limits the access to the underlying data only to the functions used to access the data.

希望你已经发现了:

Erik / SLCD和Kenji Arai /键盘

是在您的软件中使用有意义的类。

我还希望您已经进行了一些分工，小组中的每个人都已经计划好了他们的代码部分将做什么，以及他们希望其他人如何访问和使用他们的代码(在这里，**接口是关键**)。

您应该尝试确保代码的每个部分都写在单独的cpp文件中，并带有一个相关联的h文件，以便其他人可以简单地访问该代码。我并不一定希望你把你的代码写成一个类(这当然是有帮助的)，但你写的代码应该在功能上是内聚的——如果你写的代码是处理密码的，那么就这样做——不要把它与整个代码的其他部分混为一谈——代码之间干净的接口工作得最好。例如，如果你正在**编写处理密码**的代码，那么你可能会在你的代码中:

定义一个或多个数组来保存当前密码:

拥有一个函数，允许将持有输入密码的数组提交，以便与持有的密码进行比较，如果找到匹配，则返回值标识。有一个函数允许一个持有密码的数组被提交，并被用来替换持有的密码之一。

我要说的一件事是，你应该**总是使用参数来将值传递到函数中**——**不要使用全局共享的变量**——C支持参数传递是有充分理由的。

显然，如果你需要存储持久的变量(即它们比函数调用更持久)，那么它们确实需要在函数外部声明，但应该谨慎使用。看一下密码的例子。在相应的cpp文件中，我们可能有:

保存密码的Char数组;

允许用户访问(读取/更新)密码的函数-这些函数执行检查，以确保用户仅以受控的方式访问密码;

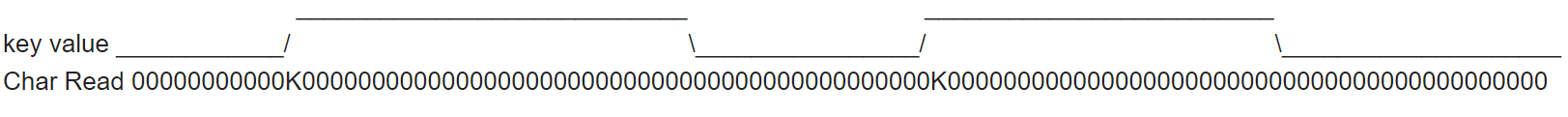
在相关的h文件中，只有用户可以实际调用的函数具有外部声明。数据变量的名称(例如密码)不包括在内，因此不能被其他地方的代码访问。这将对底层数据的访问限制为仅用于访问数据的函数。

I think I lead you astray with the Keypad I pointed to - it seems to be for a different processor. Instead, try:

[Keypad - Class library for a polling-based 4x4 or 3x4 keyp… | Mbed](https://os.mbed.com/users/codzy/code/Keypad/" \t "https://mail.google.com/mail/u/0/" \l "inbox/_blank)

This is inherently simpler but can be adapted for our use.

The class only has one function read() but this merely returns the instantaneous state of the key. The problem is that as long as you hold your finger on a key it returns a value whereas you want it to just return a value when you  press the key initially.



We want to only record a non-zero value, K, around the onset of a button push.

To do this I've edited the Keypad files and attach them here.

So, we can rename ReadKey() function to scan() and edit it so that we call it repeatedlyThis is done by creating a thread and assigning the scan() function to thread and sleeping it for 10ms after each scan - the function is now an infinite loop

we generate KeyValue but also record last\_key (the previous key value) and when (KeyValue != last\_key) then we must be at button press or depress.

At this point we can set CharValue to the current value of KeyValue ... job done.

Now we need to write a new ReadKey() function that checks CharValue, sets it back to 0 once we've read it and returns its value.

I've added some code so it won't update CharValue until the previous key has been read (this means that if you press another key whilst yo are waiting to read the key from your main program then it will miss it - a small price).

I've also added a function KeyReady() that checks if a key stroke is waiting without reading it.

The original class supported a 4x4 key pad but if you look at the differences between the attached files and the original ones you will see that I've removed one of the columns and adjusted values/indices so that it only checks 3 columns - not 4. See if you can follow what I've done. Make sure you've got the keypad connected to the ribbon cable correctly or it won't work as you expect.

In the main() function (also attached), I create the keypad and an LCD display and when I read a key I write it to the display endlessly. When the cursor is at the end of the 4 digit LCD it just continues from the start.

**The program I sent** yesterday should allow you to see how input and output from a program might work. If you look ar the bottom of main() you will find some commented-out text. These are scribblings to exemplify how you might partition code. You are looking for clean interfaces and functional elements that encapsulate well defined activity. Functions might be grouped together around data they control - you're half way to a class then.

In this example, the are two functions: GetPassword; CmpPassword. I'm positing that when you call GetPassword, it may return a password into the array (a pointer to which gets passed into the function). The typed return is non-zero (true) if there is a password returned. For CmpPassword you pass in two passwords and it returns true if they are the same. Then there are two code snippets showing how the functions might be used. Note that both functions are used in both snippets confirming,  in part, that the functional organisation is sensible.

Now you could put such functions in a class with password data hidden away in the class. So a user might see a NormalEntry() function that employs the first snippet (using the hidden passwords) to determine when the door should open. An UpdatePassword() function might use the second snippet to control the update on one hidden password.