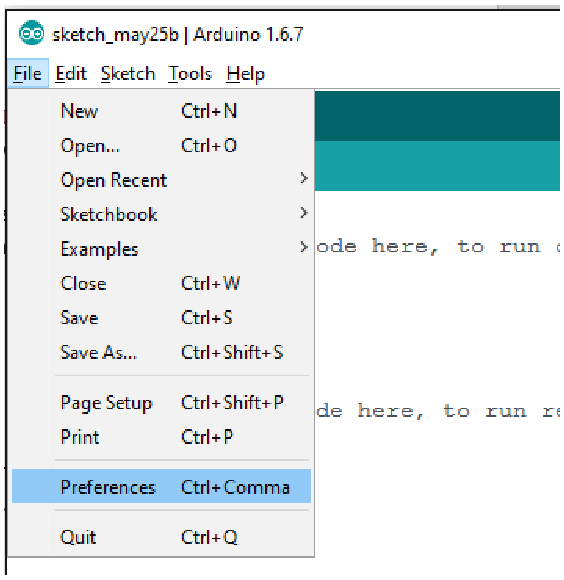
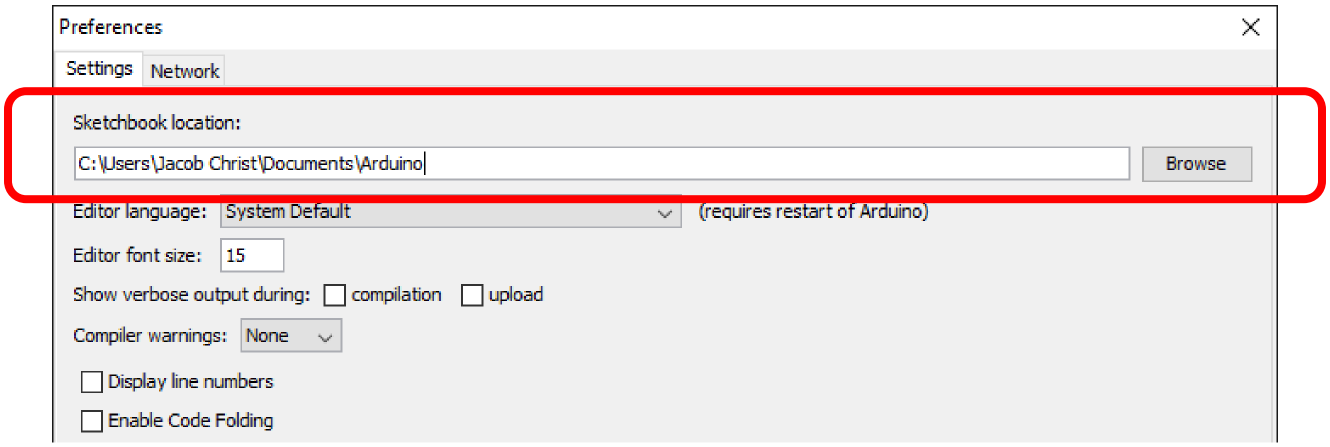
Lab 8 - Contributed Libraries and Custom Libraries

A custom or contributed library is a bit of code that can be reused over and over in many different projects and is not necessarily stored specifically with the project. Libraries are stored in a folder called "libraries" within the sketch folder. You can find the location of the sketch folder by opening the Preferences dialog box found in the File menu of Arduino IDE shown here:



```
ode here, to run
de here, to run r
```

Most libraries have been created for the same reason. To make the code reusable from project to project and to hide complex code so you don't have to look at it every time you edit your main code.



Sketchbook location: C:\Users\Jacob Christ\Documents\Arduino

Editor language: System Default (requires restart of Arduino)

Editor font size: 15

Show verbose output during: ☐ compilation ☐ upload

Compiler warnings: None

☐ Display line numbers

☐ Enable Code Folding

A simple library will consist of three files:

kewords.txt, library.h and library.cpp, where "library" is the name of the library that we are creating.

Part 1: Make new Library

In the sketchbook folder, if it doesn't already exist create a folder called libraries.

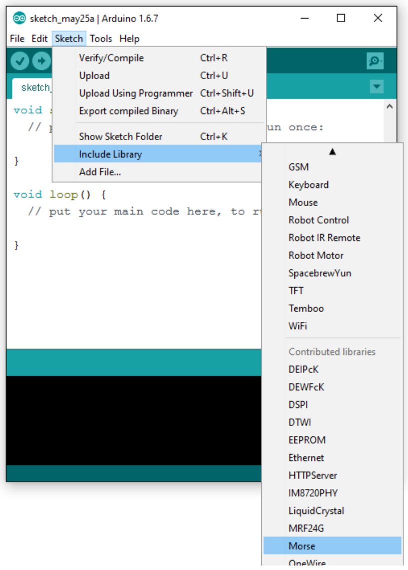
In the libraries folder create a new folder called Morse.
In that folder make three empty files **keywords.txt, Morse.h** and **Morse.cpp**. Make sure folder options for "Hide knows file types" is turned off.

If the Arduino IDE is currently running, restart the Arduino IDE and verify the Library shows up in the:

Sketch->Include Library->Contributed Libraries section.

In a new project include this library by selecting it from the menu. This should add the following line to start of your program:

#include <Mores.h>



Check off

Call the instructor to verify detection of Mores library in your program.

Part 2: Make new Library

Unfortounatly the Arduino IDE doesn't let you edit .h, *.cpp or *.txt files. For this reason, you will need to use an external editor such as notepad in Windows. Edit the Morse.h file and add this code:

```
//statements that begin with # are directives that tell the compiler what to do
#ifndef Morse_h // if Morse_h is defined skip to endif
#define Morse_h // define it now so that if this header file is included again it will
                //not try to redefine things (causes errors)

#include "WProgram.h" // this contains the pin abstraction functions

class Morse // a class is like a variable that contains other variables and functions
            // except when they are in a class they are called attributes and methods
{
  public: //public means that it can be called outside of the class
    Morse(int pin, int timeunit); //same name as the class no return type it is called
                                  //when you make an instance of the class
    void dot(); //this is a function prototype they omit the actual function and
                //end with ; they say I want this function and I'll tell you about it
                //later
    void dash();
  private: //private means that only methods within this class can use these
    int _pin;
    int _timeunit;
};

#endif
```

Edit the Morse.cpp file and add this code:

```
#include "WProgram.h" // this contains the pin abstraction functions
#include "Morse.h" // include the header file
Morse::Morse(int pin, int timeunit) //Morse:: means that this function belongs to the
                                    //Morse class
{
  pinMode(pin, OUTPUT); //set the pin as an output
  _pin = pin; //store the pin and timeunit in the appropriate attribute
  _timeunit=timeunit;
}

void Morse::dot() // this is the later where the methods are defined
{
  digitalWrite(_pin, HIGH);
  delay(_timeunit);
  digitalWrite(_pin, LOW);
  delay(_timeunit);
}

void Morse::dash()
{
  digitalWrite(_pin, HIGH);
  delay(_timeunit*3);
  digitalWrite(_pin, LOW);
  delay(_timeunit);
}
```

Create a **global** instance the Morse class near the top of your program below the #include statement that includes the library in your sketch. That way you can call the functions/methods of that instance.

```
Morse morse(PIN_LED1,150); //the pin you want the output on and the speed it outputs at
```

Test the morse class by sending out three dots then three dashes then three final dots (SOS) in your loop. Be sure to add a delay of about 1 second after sending out an SOS.

```
morse.dot(); //how to call the method
morse.dash(); //how to call the method
```

Run the code and see if it works.

Check off

Call the instructor over see your program in action.

Part 3: Add to Library

Now that library is not very useful because you have to remember the code for each letter, lets add to the library so that it can figure out the code from a letters you enter.
Add the following code to Morse.cpp just after #include"Morse.h"

```
struct morse_struct{ //defines a type that contains two chars
  char plcs; //number of dots/dashes in letter
  char code; //the code in binary 1 for dash 0 for dot
};

morse_struct lettermap[36]= { //create an array to code for the letters
// {number of dots/dashes in letter, the code in binary 1 for dash 0 for dot}
  {5,0b11111}, //0 -----
  {5,0b01111}, //1 .----
  {5,0b00111}, //2 ..---
  {5,0b00011}, //3
  {5,0b00001}, //4
  {5,0b00000}, //5
  {5,0b10000}, //6
  {5,0b11000}, //7
  {5,0b11100}, //8
  {5,0b11110}, //9
  {2,0b01}, //A
  {4,0b1000}, //B
  {4,0b1010}, //C
  {3,0b100}, //D
  {1,0b0}, //E
  {4,0b0010}, //F
  {3,0b110}, //G
  {4,0b0000}, //H
  {2,0b00}, //I
  {4,0b0111}, //J
  {3,0b101}, //K
  {4,0b0100}, //L
  {2,0b11}, //M
  {2,0b10}, //N
  {3,0b111}, //O ---
  {4,0b0110}, //P
  {4,0b1101}, //Q
  {3,0b010}, //R
  {3,0b000}, //S ...
  {1,0b1}, //T
  {3,0b001}, //U
  {4,0b0001}, //V
  {3,0b011}, //W
  {4,0b1001}, //X
  {4,0b1011}, //Y
  {4,0b1100}}; //Z
```

Add this method to the Morse.cpp file **and add a prototype in the public section of the Morse.h file**.

```
void Morse::pinChr(char in)
{
  char plcs;
  char code;
  char i;
  if(in>0x60 && in<0x7B) //if lowercase letter
  {
    in = in-0x20; //make it upper so this method is not case sensitive
  }
  if(in>0x40 && in<0x5B) //if letter
  {
    plcs = lettermap[in-'A'+ 10].plcs; //store the value from the array
    code = lettermap[in-'A'+ 10].code; //the "-'A'+ 10" gets the offset
  }
  else if(in>=0x30 && in<=0x39) //if number
  {
    plcs = lettermap[in-'0'].plcs; //store the value from the array
    code = lettermap[in-'0'].code; // the"-'0'" gets the correct offset
  }
  else //not a letter or number use space
  {
    delay(_timeunit*7); //separation between words in morse is seven units
    return; //exit function
  }
  i=plcs;
  while(i>0)
  {
    if((code&(1<<(i-1))) > 0) //test place in code for 1 or 0
      dash();
    else
      dot();
    i--; //subtract 1 from i
  }
  delay(_timeunit*3); // separation between letters is 3 units
}
```

Add a prototype in the **public** section of the Morse.h file.

```
void pinChr(char in);
```

Test it out by changing the loop() to look like this:

```
morse.pinChr('S'); //how to call the method
morse.pinChr('O');
morse.pinChr('S');
delay(1000);
```

Check off

Call the instructor over see your program in action.

Part 4: Add more to Library

Now the library is better but it does not handle strings we can add those easily.
Add this method to Morse.cpp.

```
void Morse::pinStr(char* s)
{
  char i=0;
  while(s[i]!=0)
  {
    pinChr((char)s[i++]);
  }
}
```

<u>Don't forget to add a prototype to Morce.h in the public section of the class</u>.

Now change your loop to use the pinStr command:

```
morse.pinStr("SOS"); //how to call the method
delay(1000);
```

Check off

Call the instructor over see your program in action.

Part 5: Add Syntax Highlighting to Morse

Syntax highlighting causes keywords in the Arduino IDE to change color when they are identified to help the user of the library know that they have choosen a keyword related to the library. Follow the template and make syntax highlighting for the Morse library. Remember, add this to keywords.txt. NOTE: If you copy and paste this template you will need to change the space after the keyword to a TAB character. When this document is converted to a PDF the TAB gets changed to a space.

```
#######################################
# Syntax Coloring Map For Morse
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################

#######################################
# Methods and Functions (KEYWORD2)
#######################################
dash   KEYWORD2
#######################################
# Instances (KEYWORD2)
#######################################
Morse KEYWORD1

#######################################
# Constants (LITERAL1)
#######################################
```

You need to restart the IDE for the keywords to take effect.

Check off

Call the instructor over see your syntax highlighting working.

Part 6:

Use this newly tested library in your parser project. Open an old project that has a parser and a command called "morse s" command where s is a string to be interpreted by the Morse class. It would work similar to the flip s command from a previous lab except that it will blink the led in morse code for the string you enter. **Hint, add this to your parser:**

```
 else if(strncmp(command,"morse", 5) == 0) {
   morse.pinStr(&command[6]);
 }
```

Check off

Call the instructor over to demonstrate the morse command in your parser.

Part 7: Output Morse to Serial

Make two new functions for the Morse library "write" that outputs the dots and dashes (.-) or a space if not a letter or number for one character, and "print" that outputs a string as Morse code out the Serial port.

hints:
Use the pinChr as a guide for the write() and replace the dash() and dot() functions with Serial.print()
Use the pinStr methods as a guide for print
Don’t forget your prototypes in the .h file

```
void Morse::write(char in){
  // start with the pinChr()
}

void Morse::print(char* s){
  // start with pinStr
}
```

Have the "morse s" command also print the string to the Serial Monitor.

Check off

Call the instructor over see your program in action.