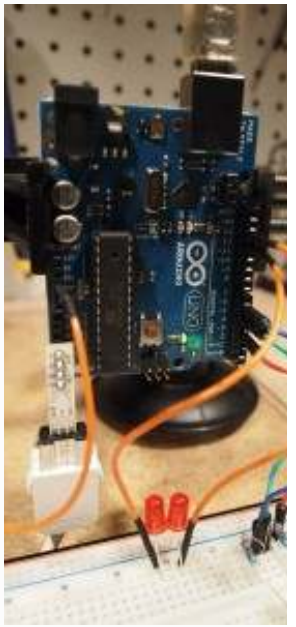
# A Library for Using the Wii Nunchuk In Arduino Sketches

POSTED ON 10 FEB 2012



There are several available libraries for communicating with the Wii nunchuk. I somehow found something lacking in all of them, so I wrotewiinunchuk.h for my projects. It is inspired by Tod E. Kurt. but my wiinunchuk.h boasts more powerful features and a more flexible API.

This library is free software and is licensed under the GPL v3.

**Update!** Due to relative popularity, I have created a Git repository for wiinunchuck.h. For the latest version, get it here.

The code is relatively well commented on its own. The header provides the functions

void nunchuk_setpowerpins()
void nunchuk_init()
int nunchuk_get_data()
void nunchuk_calibrate_joy()
inline unsigned int nunchuk_zbutton()
inline unsigned int nunchuk_cbutton()
inline int nunchuk_joy_x()
inline int nunchuk_cjoy_x()
inline int nunchuk_cjoy_y()
inline uint16_t nunchuk_accelx()
inline uint16_t nunchuk_accely()
inline uint16_t nunchuk_accelz()
inline int nunchuk_caccelx()
inline int nunchuk_caccely()
inline int nunchuk_caccelz()
inline int nunchuk_joyangle()
inline int nunchuk_rollangle()

inline int nunchuk_pitchangle()
void nunchuk_calibrate_accelxy()
void nunchuk_calibrate_accelz().

# Calibration

Although the header provides functions to pull raw data right from the nunchuck, in most typical applications it is useful to have the data from the nunchuck normalised so that certain reference points are defined to be zero. For example, the reading from the joystick potentiometer will range from about 24 to about 230 on both x and y axes. The normal reading when the joystick is centred is about 130 (it will be slightly different in for each axis). If you subtract the centre value from the instantaneous joystick readings, then it is zeroed at the centre and readings will be relative to the centre.

Reasonable default centre values are #defined near the top of the document:

```
#define DEFAULT_CENTRE_JOY_X 124
#define DEFAULT_CENTRE_JOY_Y 132
#define ACCEL_ZEROX 490
#define ACCEL_ZEROY 500
#define ACCEL_ZEROZ 525
```

These are loaded into program memory in `void nunchuk_init()`. If you need accuracy and plan to use the same nunchuk in all applications, it is worth your time to measure these for your nunchuk by spitting the rad nunchuk data to a serial terminal. For other cases, wiinunchuk.h provides functions

void nunchuk_calibrate_joy()
void nunchuk_calibrate_accelxy()
void nunchuk_calibrate_accelz().
The function `nunchuk_calibrate_joy()` will take the instantaneous reading of the joystick values and record them as the zero value. The end user should be warned not to manipulate the joystick during the moment it is called. Usually, this is at the beginning of the sketch before the action starts.

```
void nunchuk_calibrate_joy()
   {
   joy_zerox = joy_x;
   joy_zeroy = joy_y;
   }
```

`nunchuk_calibrate_accelxy()` and`nunchuk_calibrate_accelz()` will record zero values for the accelerometer axes. This has to be done in two steps since the force of gravity will always affect at least one of the accelerometer axes. If the nunchuk is held upright, the *xy* calibration will be clean since gravity is pulling towards negative *z*. Then, with the nunchuk on its side or face so that gravity affects only *x* or *y*, the *z* accelerometer can be read to take the zero value using `nunchuk_calibrate_accelz()`.

```
void nunchuk_calibrate_accelxy()
   {
   accel_zerox = nunchuk_accelx();
   accel_zeroy = nunchuk_accely();
   }

void nunchuk_calibrate_accelz()
   {
   accel_zeroz = nunchuk_accelz();
   }
```

# Initialisation

`nunchuk_setpowerpins()` sets up the pins.

```
void nunchuk_setpowerpins()
   {
   #define pwrpin PORTC3
   #define gndpin PORTC2
   DDRC |= _BV(pwrpin) | _BV(gndpin);
   PORTC &= ~_BV(gndpin);
   PORTC |=  _BV(pwrpin);
   delay(100); // wait for things to stabilize
   }
```

The function assumes that the nuncuk is connected using a WiiChuk, or at least that you are using the same pin configuration. That is to say A2, A3, A4 and A5 are power (-), power (+), data and clock, respectively. Even though these are analog pins, this function treats A2 and A3 as digital pins and sets them high and ground to power the nunchuk with 5V. As I write that, I recall that the nunchuk spec calls for less than 5V, but WiiChuk users don't have much choice. If you cut the connector on your nunchuck and are plugging in directly, I would try using the Arduino's 3.3V power instead.

The `nunchuk_init()` function sets up communication with the nunchuk by establishing the i2c bus and handshaking. The way this is done should work for OEM Nintendo nunchuks and for the el-cheapo knock-offs.

```
void nunchuk_init()
   {
   Wire.begin();
   delay(1);
   Wire.beginTransmission(0x52);  // device address
   #if (ARDUINO >= 100)
      Wire.write((uint8_t)0xF0);  // 1st
initialisation register
      Wire.write((uint8_t)0x55);  // 1st
initialisation value
      Wire.endTransmission();
      delay(1);
      Wire.beginTransmission(0x52);
      Wire.write((uint8_t)0xFB);  // 2nd
initialisation register
      Wire.write((uint8_t)0x00);  // 2nd
initialisation value
   #else
      Wire.send((uint8_t)0xF0);   // 1st
initialisation register
      Wire.send((uint8_t)0x55);   // 1st
initialisation value
      Wire.endTransmission();
      delay(1);
      Wire.beginTransmission(0x52);
      Wire.send((uint8_t)0xFB);   // 2nd
initialisation register
      Wire.send((uint8_t)0x00);   // 2nd
initialisation value
   #endif
   Wire.endTransmission();
   delay(1);
   //
   // Set default calibration centres:
   //
   joy_zerox = DEFAULT_CENTRE_JOY_X;
   joy_zeroy = DEFAULT_CENTRE_JOY_Y;
   accel_zerox = ACCEL_ZEROX;
   accel_zeroy = ACCEL_ZEROY;
   accel_zeroz = ACCEL_ZEROZ;
   }
```

Some other libraries do a more canonical sort of initialization that causes the data to be encrypted and is not compatible with 3rd party nunchucks. The function finishes off by loading the default zero values for the sensors into memory. They can hereon be changed directly or using the calibration functions discussed earlier.

# Communication

We expect to get back six bytes of information:

| Byte | Contains |
|---|---|
| 1 | Joystick x |
| 2 | Joystick y |
| 3 | First 8 of 10-bit accelerometer *x* value |
| 4 | First 8 of 10-bit accelerometer *y* value |
| 5 | First 8 of 10-bit accelerometer *z* value |
| 6 | • Last two bits from acc *z*<br>• last two bits from acc *y*<br>• last two bits from acc *x*<br>• 1-bit for Cbutton state<br>• 1-bit for Z-button state |

The function `nunchuk_get_data()` requests and receives the six bytes data, reads it into a six element array `nunchuk_buf`.

```
int nunchuk_get_data()
   {
   int cnt=0;
     // Request six bytes from the chuck.
   Wire.requestFrom (0x52, 6);
   while (Wire.available ())
      {
     // receive byte as an integer
      #if (ARDUINO >= 100)
         nunchuk_buf[cnt] = Wire.read();
      #else
         nunchuk_buf[cnt] = Wire.receive();
      #endif
      cnt++;
      }
   //
```

```
    //Send read address, zero-byte.
    #if (ARDUINO >= 100)
       Wire.write((uint8_t)0x00);
    #else
       Wire.send((uint8_t)0x00);
    #endif

    if (cnt >= 5)
       {
       return 1;   // success
       }
    return 0; // failure
    }
```

# Processing and output

The remaining functions deal with unpacking our six-byte data package, `nunchuk_buf[]` and presenting it in various useful forms. Since these functions are small and likely to be in a program loop, I cast them all as static inline functions to avoid emission of object code and have the code injected directly into the points of use in the Arduino sketches.

To start, we capture the C and Z buttons with `nunchuk_cbutton()`and `nunchuk_zbutton()`. The one-bit on/off values are stored in the last two bits of the sixth byte. We just bit shift an appropriate value and check the state of the last bit of that byte. The checking of the byte value is done using C's inline ternary conditional operator`(condition)? true : false;`.

```
 static inline unsigned int nunchuk_zbutton()
    {
    return ((nunchuk_buf[5] >> 0) & 1) ? 0 : 1;
    }

 static inline unsigned int nunchuk_cbutton()
    {
    return ((nunchuk_buf[5] >> 1) & 1) ? 0 : 1;
    }
```

There isn't much to be done about the raw *x* and *y* values, so we just return them with `nunchuk_joy_x()` and `nunchuk_joy_x()`.

```
 static inline int nunchuk_joy_x()
    {
    return (int) nunchuk_buf[0];
```

```
    }

 static inline int nunchuk_joy_y()
    {
    return (int) nunchuk_buf[1];
    }
```

We also want the option of having the calibrated (zeroed and centre) joystick values which are returned with functions `nunchuk_cjoy_x()`and `nunchuk_cjoy_x()`

```
 static inline int nunchuk_cjoy_x()
    {
    return (int)nunchuk_buf[0] - joy_zerox;
    }

 static inline int nunchuk_cjoy_y()
    {
    return (int)nunchuk_buf[1] - joy_zeroy;
    }
```

The accelerometer readings are a more difficult matter because of the 10-bit values, the least significant two bits are stored in byte six, which is a mish-mash of data. So, take for example the function`nunchuk_accelx()`.

```
 static inline uint16_t nunchuk_accelx()
    {
    return (  0x0000 | ( nunchuk_buf[2] << 2 ) +
       ( ( nunchuk_buf[5] & B00001100 ) >> 2 )  );
    }
```

First, `0x0000 | ( nunchuk_buf[2] << 2 )` takes a blank 16-bits (0x0000) and does a bitwise OR (|) with the 8-bit nunchuk_buf[2] which has been bit-shifted by 2 (<<2). The bitwise OR simply takes nunchuk_buff[2] and imprints it onto 0x0000 creating a 16-bit version of nunchuk_buf[2]. The bit-shift ensures that the last two bits are blank so we can add in the last two bits which are stored in nunchuk_buf[5]. So, to this we add `( nunchuk_buf[5] & B00001100 ) >> 2 )` which takes the pair of bits starting second from the left of nunchuk_buf[5] and zeros everything else. Then those two bits are shifted to the end to be added to the 16-bit copy of nunchuk_buf[2] which has the right-most two bits empty and waiting.

Rinse and repeat for the *y* and *z* accelerometer values.

```
 static inline uint16_t nunchuk_accely()
    {
```

```
    return (  0x0000 ^ ( nunchuk_buf[3] << 2 ) +
       ( ( nunchuk_buf[5] & B00110000 ) >> 4 )  );
    }

 static inline uint16_t nunchuk_accelz()
    {
    return (  0x0000 ^ ( nunchuk_buf[4] << 2 ) +
       ( ( nunchuk_buf[5] & B11000000 ) >> 6 )  );
    }
```

The sketch may optionally want the calibrated accelerometer values which are offered by the following three functions.

```
 static inline int nunchuk_caccelx()
    {
       return (int)(nunchuk_accelx() - accel_zerox);
    }

 static inline int nunchuk_caccely()
    {
       return (int)(nunchuk_accely() - accel_zeroy);
    }

 static inline int nunchuk_caccelz()
    {
       return (int)(nunchuk_accelz() - accel_zeroz);
    }
```

The final three functions are concerned with angle. I include them because some people have a rather serious trigonometry allergy. The vector magnitudes which compliment these angles, I'll leave to the Pythagorean wizardry of whom ever may need them.

You'll note the use of the `atan2()` function. This is exactly like vanilla`atan()`, except that it keeps track of which quadrant we are in by the signs of the opposite and adjacent lengths, and also handles vectors one the axes gracefully.

```
 static inline int nunchuk_joyangle()
    {
    double theta;
    theta = atan2( nunchuk_cjoy_y(), nunchuk_cjoy_x()
 );
    while (theta < 0) theta += 2*M_PI;
    return (int)(theta * 180/M_PI);
    }
```

The little while-loop in here makes sure that we get a positive angle by rotating angles into their positive equivalent.

A little trick to get the orientation of the nunchuk from the accelerometers is to sniff out the gravity vector. If the accelerometer is properly zeroed, and all other external forces are negligible (i.e., gravity is the only significant fore on the numchuk), then the ratios of the accelerometer readings will give the roll and pitch of the nunchuk.

Roll (`nunchuk_rollangle()`) is the angle created by tipping an upright nunchuk to the left or right as viewed from behind. More precisely, with the z-axis pointing up from the joystick, the y-axis point out of the c-button and the x-axis pointing from the right side, roll is the angle created between the z-axis and the nunchuk when it is rotated about the y-axis.

```
static inline int nunchuk_rollangle()
   {
   return (int) (  atan2( (double) nunchuk_caccelx(),
      (double) nunchuk_caccelz() ) * 180 / M_PI  );
   }
```

Then by the same definitions, pitch is the angle created between the nunchuk and the z-axis when the nunchuk is rotated about the x-axis and 'tipped' forward or backward.

```
static inline int nunchuk_pitchangle()
   {
   return (int) (  atan2( (double) nunchuk_caccely(),
      (double)nunchuk_caccelz() ) * 180 / M_PI  );
   }
```

If you haven't already, **click here to Download wiinunchuk.h.**

I hope you find this useful. Please feel free to post links to your projects or ask questions. Enjoy!



This entry was written by Tim Teatro, posted on February 10, 2012 at 01:34, filed under Electronics, Software & FOSS and tagged Arduino, C,Nintendo Wii, Software, Wii Nunchuk. Bookmark the permalink. Follow any comments here with the RSS feed for this post. Post a comment or leave a trackback: Trackback URL.

# 34 Comments »

MARCH 5, 2012 AT 17:55

Paul *says:*

I have this error. can you help.

In function ‘void nunchuk_init()’:
error: ‘Wire’ was not declared in this scope In function ‘void nunchuk_send_request()’:
In function ‘int nunchuk_get_data()’:

Reply

MARCH 5, 2012 AT 18:33

Tim Teatro *says:*

Are you using the latest Arduino IDE? It looks like Wire.h isn’t in your compiler scope. Wire is the library used for I2C communication. It should be included with Arduino.h or WProgram.h.

Let us know.

Reply

DECEMBER 2, 2012 AT 19:07

MK *says:*

I have the latest Arduino 1.0.2. I get the same error. What can I do to fix this?

Reply

DECEMBER 2, 2012 AT 19:39

Tim Teatro *says:*

Make sure your environment is set up and that your libraries are in place. I can’t be more specific than that at the moment. Try compiling a minimum working example where you are using Wire.h. If you can do that, then the problem is with my code. If not, then it is in your environment. Either way, you will have more information that will help diagnose the problem.

Reply

MARCH 30, 2012 AT 16:48

Derek Phillips *says:*

Thanks Tim, I will have fun with this. Numbers are looking good so far.

Reply

APRIL 1, 2012 AT 19:16

Tim Teatro *says:*

You're very welcome! Feel free to post a link to or some pictures of what you are working on.

Reply

APRIL 30, 2012 AT 03:03

Dain *says:*

We also want the option of having the calibrated (zeroed and centre) joystick values which are returned with functions nunchuk_cjoy_x() and nunchuk_cjoy_x()

——

Shouldn't that be

nunchuk_cjoy_x() and nunchuk_cjoy_y()

or am I being dense?

Reply

JULY 18, 2012 AT 18:23

Laurent *says:*

It's work fine: thanks.
I plan to do an earthquake sensor with the following Jonathan Thomson's web journal idea and your .h. Nevertheless, I think that it's alredy exist.

Thanks again.

Reply

AUGUST 25, 2012 AT 14:45

john *says:*
could you please direct me to some code using the wiichuck to control 2 servos in a pan/tilt config? Everything I find online seems to be outdated, and yours is the only library I can get to verify on my UNO correctly.

I am still a noob! Although I can get a wired parallax thumbstick to work just fine, I want to go wireless, and trying to write the code myself is frustrating.

Thanks, John

Reply

OCTOBER 1, 2012 AT 20:59
Tim Teatro *says:*
Not sure where there is any. I haven't done it myself, but it shouldn't be too hard. Take small steps. Get the servos working by turning a potentiometer. Then try mapping the output from the y-axis thumb stick to the servo (instead of the potentiometer). If you don't want to do this manually, I think the Arduino API has a map() function which will take the numerical output from the thumb stick and map it to the integer range that the servo accepts.

Let me know how it goes!

Reply

NOVEMBER 10, 2012 AT 23:43
Eduardo *says:*
When i press the z button it indicates that both of z and c are pressed, when i press both, it shows that only z is pressed. Anyone know how to fix it ?

Reply

DECEMBER 2, 2012 AT 19:36
Tim Teatro *says:*
I'd look at your wiring. Let us know.

Reply

DECEMBER 2, 2012 AT 20:03
Eduardo *says:*
i have used one of these (http://todbot.com/blog/wp-content/uploads/2008/01/wiichuck_adapter1.jpg) and

wired the d on the pwm 20 and the c on the pwm 21. Im using the arduino mega.

Reply

DECEMBER 2, 2012 AT 22:46

Chris Milne *says:*

Great Library!

Just a note to anybody trying to use the Nyko Kama Wired Nunchuck from Adafruit. We found that it is treated as a wireless nunchuck when using this code, so DON'T comment out this line if you're using the Nyko Kama Wired Nunchuck. It'll work just fine then. Otherwise you end up getting maxed out values.

Reply

DECEMBER 8, 2012 AT 13:41

Michał Margula *says:*

Great job! But to be honest – I keep in hand nunchuck for the first time and it is really hard to tell which axis are X, Y and Z. You explained in your post but I have no idea what means "keep upright". Should joystick be perpendicular to the ground or parallel?

I think best would be a picture of nunchuck and axis drawn on it :).

Once again thanks for great job!

Reply

Pingback: *Arduino Nunchuck project | Alex Stevenson*

Pingback: *Arcerino | Gruppo Linux Como*

APRIL 24, 2013 AT 23:28

Unkwrr *says:*

Heey..

I have the sameproblem asmentioned in this thread here:

http://arduino.cc/forum/index.php?PHPSESSID=ae3812c9eee19f13c5cb8ea777347174&topic=21723.0

with my Arduino MEGA.

nunchuk_get_data() always returns 0 so i think there must be the error.

Reply

MAY 10, 2013 AT 18:36

Tim Teatro *says:*

I think you need to look more closely at which pins the MEGA wants to use, and which pins are used by my library. You may need to modify my library to comply with the pin configuration on the MEGA.

If you manage this, please tell us about it.

Reply

OCTOBER 17, 2013 AT 15:52

Kevin Watknis *says:*

I appreciate what your doing but I am getting errors like this when I use this library.
In file included from WiichuckDemo.ino:1:
/Users/resista/Documents/Arduino/libraries/wiinunchuckmaster/wiinunchuck.h: In function ‘void nunchuk_init()’:
/Users/resista/Documents/Arduino/libraries/wiinunchuckmaster/wiinunchuck.h:113: error: ‘Wire’ was not declared in this scope
/Users/resista/Documents/Arduino/libraries/wiinunchuckmaster/wiinunchuck.h: In function ‘void nunchuk_send_request()’:
/Users/resista/Documents/Arduino/libraries/wiinunchuckmaster/wiinunchuck.h:162: error: ‘Wire’ was not declared in this scope
/Users/resista/Documents/Arduino/libraries/wiinunchuckmaster/wiinunchuck.h: In function ‘int nunchuk_get_data()’:
/Users/resista/Documents/Arduino/libraries/wiinunchuckmaster/wiinunchuck.h:181: error: ‘Wire’ was not declared in this scope

Reply

NOVEMBER 20, 2013 AT 18:31

Tim Teatro *says:*

It looks like you have a problem with the Wire library. I’ve used this very recently, with nothing but the arduino package that comes with Ubuntu (Mint, actually). I’m using a nunchuck to control a robot in my lab.

Have you solved this problem, or would you like a hand to debug?

Reply

JANUARY 1, 2014 AT 18:45

Matthias Ohrnberger *says:*

@ Kevin

hopefully this reply is much too late as you may have already solved your problems with the Wire library, but I assume it is just a missing include of Wire.h before the wiinunchuck.h include? You need to:

#include “Wire.h”
#include “wiinunchuck.h”

in your main sketch to make things work – if this is what you did while receiving the compile errors above then it is likely that – as pointed out by Tim – you may have an installation problem with libraries (which looks very unusual to me).

@ Tim – thanks so much for this nice and clean library – although I’m pretty new to arduino stuff, it made my nunchuck work immediately after attempting to apply older code for some hours without any success.
I was already disassembling the whole device to make sure that it receives appropriate power and to check all wiring. Seems that my (cheap) buy was definitely not compatible with the older codes – Interestingly I can’t see too much difference except your nunchuk_init() routine that sends initialization bytes twice … is this the only trick to deal with these cheap, non-original nunchuck devices? The routines for computing joystick-, roll- and pitch- angles are also highly appreciated – thanks a lot for making it public. Great stuff

Reply

JANUARY 2, 2014 AT 16:06

Tim Teatro *says:*

You’re welcome!

Reply

DECEMBER 15, 2013 AT 01:15

Nathan Sandland *says:*

Calibration doesn’t work because joy_x and joy_y variables never get set (but are read by calibration)

Reply

JANUARY 31, 2014 AT 15:57

Neil Hendrick *says:*

Would love to see an example sketch that outputs Wiichuck to Serial.

Reply

APRIL 5, 2014 AT 16:52

Tim Teatro *says:*

This fella has written a simple program that just poles some info from the nunchuk and pushes it over serial.

https://github.com/capncanuck/wii-serial

Reply

MAY 8, 2014 AT 20:32

Miss Frankonia *says:*

I'm using your library and everything works fine. Great work, thanks!
I'm unsing it to interact with processing software so far.
That way I also may visualize the acceleration data.

But I'm wondering what is the maximum sampling rate. So far I measure something like 60Hz. Is there any way to increase that? I estimate the key to increase is hidden in the wire.h lib. Can you give me any hint where to do any changes?

Miss Frankonia

Reply

JUNE 2, 2014 AT 03:28

Tim Teatro *says:*

I'm not sure. It may be limited by the Nunchuk hardware.

What are you doing that requires more than 60 Hz? You've really got me curious!

Reply

JUNE 2, 2014 AT 20:46

Miss Frankonia *says:*

Max. sampling rate

You are wondering for what more than 60 Hz sampling rate are good for. That's a fair comment – because you don't need it for a human machine interface as a joystick or the like.
I was wondering if I can abuse the nun-chuck internal sensor for something else (e.g. measuring the accelleration of a water rocket, a model race care or the like, or use it as a synchronizing circuit for a picture of vision circuit, measuring the movement of a pendulum …).
The nun-chuck is extremely cheap (cheaper than an ADXL345 pcb) and thanks to websites like yours it is very well documented and many libraries are available.

JUNE 1, 2014 AT 23:56

S Scott *says:*

This looks like a very clean and comprehensive code, thank you! I've been struggling with setting up this wiichuck all day (using various codes), and every sketch refuses to recognize the wire.h function. I re-installed Arduino 1.0.5 with no improvement. I can't find any resources that address wire.h not being recognized. Do you have any ideas??

Reply

JUNE 2, 2014 AT 03:26

Tim Teatro *says:*

No clue. Are you using Linux, Mac or Windows?

Reply

JUNE 2, 2014 AT 04:08

S Scott *says:*

Thanks for your fast reply!

Windows Vista (which itself is a clean install after many Win8 conflicts). Do you know if it's possible to locate the Arduino libraries separately, and then manually load them? Or what and where I can look to confirm that the Wire library is actually in my files? (although there's no reason it shouldn't be as I haven't made any changes to the Arduino install).

One odd thing, when I installed Arduino 1.0.5, there were many files I had to 'skip' or ignore – mostly java related, it seems, but also some .dlls I didn't identify. I don't remember all these orphan files when I first installed Arduino (prior to the Win8 debacle).

Reply

JUNE 2, 2014 AT 18:41

S Scott *says:*

Ok, I seemed to have addressed the wire library problem by adding a call-out at the start of the code similar to what Matthias suggested above; but now I'm getting these errors which I don't even understand:

core.a(main.cpp.o): In function `main':
C:\Program Files (x86)\Arduino\hardware\arduino\cores\arduino/main.cpp:11: undefined reference to `setup'
C:\Program Files (x86)\Arduino\hardware\arduino\cores\arduino/main.cpp:14: undefined reference to `loop'

Reply