EMCU

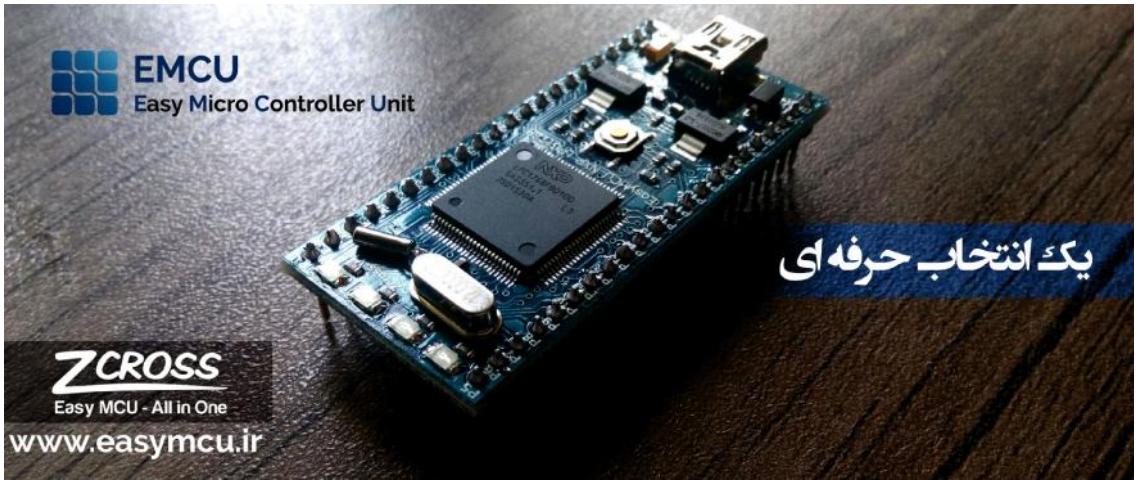
Easy Micro Controller Unit

با C++ بتازید...

مقدمه ای بر مفاهیم سناریو نویسی با C/C++

ویـــــــــژه ی میکروکنترلرها به زبان ساده

کاری از EasyMCU



https://telegram.me/EasyMCU

http://www.easymcu.ir

طعم خلاقیت را با ایزی. ام. سی. یو (EasyMCU) تجربه کنید

ارتباط با ما:

انجمن : http://forums.easymcu.ir

ایمیل : info@easymcu.ir

مقدمه مولف:

سلام عرض می‌کنم خدمت دوستان عزیزی که خلاقیت رو دوست دارند و پیوسته دنبال تغییر و چالش‌های جدید هستند، چرا که به واقع تغییر خود زندگی و زندگی خود تغییر هست. به شما دوست خوبم تبریک می‌گم که این کتاب رو دانلود کردی و داری می‌خونی و خواهان تغییر هستی و این ویژگی هست که شما رو از بقیه متمایز می‌کنه.

هدف از این نوشته و شروع پروژه‌ی EasyMCU این بوده که کانالی بسازیم و از بند محدودیت-های آموزشی رایج خلاص بشیم. بندی که همه مواردش به بَسکام و کُدویژن و برنامه نویسی طوطی‌وار و بدون فکر و بدون خلاقیت ختم می‌شه. هدف این بوده که خودمون رو بکشیم بالا و جهانی فکر کنیم و گام‌های بلندتری برداریم. مفاهیم رو بفهمیم و حفظ نکنیم، تامل کنیم، فکر کنیم، تحلیل کنیم و راه جدید بسازیم. سعی کنیم و شکست بخوریم و یاد بگیریم شکست خوردن اصلا بد نیست و تا شکستی نباشه، پیروزی هم نخواهد بود و در نهایت پیروز بشیم و به معنی واقعی و جان کلمه‌ی خلاقیت، خلاق باشیم، نه در حد اسم.

آموزش‌ها در این نوشته بدون فهرست هست و غیر رسمی و تا حدی ملموس عنوان شده که بچه-های سنین متوسطه هم بتوانند از مفاهیم ارائه شده در این نوشته برخوردار شوند.

هیچ موضوعی رو برای یادگیری به آینده موکول نکنید، برنامه نویسی رو تفریح بدونید و از سنین کم شروع کنید.

این نوشته جمع آوری جلسات آموزشی دوره EasyMCU از کانال تلگرامی EasyMCU هست و مربوط به ۵ جلسه اول می‌شود که کلا به مفاهیم برنامه نویسی و شیء گرایی با زبان بسیار ساده پرداخته شده و کاملا مقدمه و مجوز ورود به دنیای خلاقیت به واسطه ی EasyMCU هست. لازم به ذکر است پیام های انگیزشی ابتدای هر جلسه از کانال تلگرامی https://telegram.me/ENERGYPLUSSSSSSS می باشد.

EasyMCU حرکتی است که نحوه‌ی برنامه نویسی مغز مدارهای الکترونیکی رو راحت می‌کنه و به شما اجازه می‌ده انرژی و خلاقیتتون رو روی ابعاد عملیاتی پروژه متمرکز کنید و البته در کم‌ترین زمان ممکن با بیش‌ترین حد خلاقیت. برای اطلاعات بیش‌تر به وبسایت EasyMCU مراجعه کنید و مکرر وبسایت رو چک کنید، چرا که پیوسته آموزش‌ها و پروژه ها و ویدئوهای جدید به وبسایت اضافه می شه و می‌تونید بیش‌تر و بیش‌تر یاد بگیرید و بیش‌تر و بیش‌تر به بقیه یاد بدید و پروژه‌هاتون رو به اشتراک بذارید.

از طریق انجمن می‌تونید سوالاتتون رو بپرسید و مشکلاتتون رو حل کنید و پروژه‌هاتون رو با بقیه به اشتراک بذارید. در نهایت از طریق راه‌های ارتباطی عنوان شده می‌تونید نظرتون رو در مورد آموزش‌ها با ما در میان بگذارید، قطعا خوشحال و دلگرم می‌شویم و البته سطح و کیفیت آموزش‌ها به واسطه‌ی نظرات ارزشمند شما قطعا ارتقا پیدا خواهد کرد.

با آرزوی موفقیت برای همه‌ی عزیزان.

مرتضی زندی

جلسه اول:

با یاری حق جلسه‌ی اول از سری آموزش‌های EasyMCU رو شروع می‌کنیم. در این جلسه قدری با زبان C/C++ آشنا می‌شویم. اگر با زبان C آشنایی ندارید ممکنه از دور قدری سخت به نظر برسه، اما واقعیت اینه که وقتی یادش بگیرید، قطعا نمی‌تونید ازش دل بکنید!

زبان C بسیار استاندارد هست و خیلی راحت می‌شه باهاش ارتباط برقرار کرد. پس اگر با زبان C آشنایی ندارید، نگران نباشید و از تغییر نترسید، در این دوره‌ی آموزشی یاد می‌گیرید سکوی پرتاب اختصاصی خودتون را بسازید و در پایان دوره قادر خواهید بود موشک‌های خلاقیت خودتون رو از روی این سکوهای پرتاب شلیک کنید!

☑ مختصر صحبت انگیزشی با دوستانی که تازه قصد ورود به وادی برنامه نویسی رو دارند:

کسی که به معنای واقعی کلمه موفق می‌شه خودش رو به وادی برنامه نویسی برسونه، یه سری پرده‌ها از جلو چشمش کنار می‌ره، زندگیه اون فرد قبل از ورود به وادی برنامه نویسی هرگز قابل قیاس با بعد از ورودش نیست! تا دیروز یک ماشین که می‌دید از کنارش رد می‌شد، نهایت درگیر ظاهرش می‌شد و اما امروز ذهنش بسیار تحلیل‌گر شده و موقع دیدن ماشین، به ظرافت‌های ساخت نهانش هم توجه ویژه می‌کنه، موقع پیچ زدن دیفرانسیل و ... اش رو تجسم می‌کنه و بررسی می‌کنه چطوری داره کار می‌کنه ...! هرچیزی رو سعی می‌کنه تحلیل کنه ...!

بله، دلیلش اینه که طعم خالق بودن و خلق کردن رو چشیده! شاید براتون عجیب باشه، ولی سناریوی برنامه نویسی شباهت بسیار زیادی به سناریوی زندگی داره ... !

شئ‌ها متولد می‌شن، هر کدوم عملکردها و ویژگی‌های خاصی رو دارن، زندگی می‌کنن، نقششون رو ایفا می‌کنن و می‌میرند و از صحنه‌ی زندگی محو می‌شن و ... !

واو ...! حس خلق کردن، فوق العاده نیست!

وقتی قراره یه سناریوی زندگی برای یک پردازنده بنویسیم، باید از منطق ۰ و ۱ هم دید خوبی داشته باشیم تا بتونیم بهترین سناریوی ممکن رو براش بنویسیم.

برنامه نویسی رو هرگز به عنوان درس قبول نکنید! از زمان تفریحتون براش وقت بذارید!

شما می‌تونید پردازنده رو سرباز یا فرمانده بدونید، یا مغز یا هر چیز دیگه‌ای ... ، می‌تونید باهاش

رابطه‌ی پدر فرزندی هم برقرار کنید! دقت کنید که این فرزند بسیار منطقی هست! پس اگر عملکرد مناسب نداشت اشکال از پدرش هست یعنی شما! پس اگر دنبال مقصر هستید، مقصر هم مشخص شد!

برای اینکه سناریوی تربیت فرزند رو خوب بتونیم بنویسیم به نام خدا شروع می‌کنیم ...

شما که یک نویسنده هستید باید سناریو رو با قواعد معینی بنویسید. تنه‌ی اصلی برنامه رو عملکردهای یک انسان در نظر بگیرید. اسم ایشون main هست، کارایی رو که باید در زندگیش انجام بده زیر خودش می‌نویسیم.

```
int main()
{


}
```

به این ترتیب کل نقشی رو که این انسان باید بازی کنه بین { } می‌نویسیم.

حالا قراره چه نقشی بازی کنه؟ خوب این بستگی به ما داره که سناریوی چه نقشی رو بهش پیشنهاد بدیم!

برای اینکه یکم هیجانی بشه باید سناریو رو اکشن کنیم!

پس نیازه یه سری ابزار به آقای main اضافه کنیم که بتونه مثلا تبدیل به کاماندو بشه و ماموریتش رو انجام بده.

چه ابزراهایی؟ مثلا یک کلت کمری + چاقو + مسلسل + نارنجک + لباس فرم.

ببینیم این ابزارها چطوری اضافه می‌شن ...

```
#include  ”kamando.h”
#include  ”kolt.h”
#include  ”chaghoo.h”
#include  ”mosalsal.h”
#include  ”narenjak.h”
#include  ”lebaseForm.h”
int main()
{
```

}

و اما سناریوی آقای کاماندو رو درون تنه ی اصلی برنامه می‌نویسیم. آقای علی یک کاماندو هست که جلوش رو نگاه می‌کنه و با کُلتِ کمری یک تیر شلیک می‌کنه و دیگه کاری نمی‌کنه. برنامه‌اش به شکل زیر می‌شه:

```
#include ”kamando.h”
#include ”kolt.h”
#include ”chaghoo.h”
#include ”mosalsal.h”
#include ”narenjak.h”
#include ”lebaseForm.h”

int main()
{
        Kamando ali;
        Kolt kolt;
        ali.look(front);
        kolt.shoot();
```

```
    while(1);
}
```

در خط اول اصطلاحا علی یک شئ از کلاس کاماندو تعریف می‌شه و خط بعد یک شئ کلت از کلاس کلت. پس تا اینجا علی را کاماندو خلق کردیم و یک کلت هم خلق کردیم.

خط بعد متود look اجرا می‌شه با ورودی front به معنی جلو، یعنی علی جلو را نگاه می‌کند.

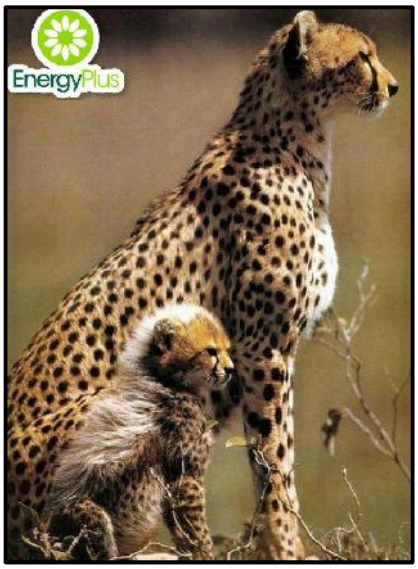
خط بعد متود shoot اجرا می‌شه که باعث می‌شه کلت یک گلوله شلیک کنه.

خط بعد اصطلاحا یک حلقه‌ی بی نهایت است که خط برنامه تا ابد درش گرفتار می‌شه و کاری انجام نمی-ده.

تا اینجا هدف این بوده که یه دید کلی نسبت به چهارچوب کد نویسی C/C++ و قدری شئ‌گرایی پیدا کنید. فعلا جزئیات مطرح نیست. در جلسات بعد به مرور وارد جزئیات می‌شیم.

زبان C++ زائیده‌ی زبان C هست که در واقع شئ‌گرایی + C می‌شه C++، برای اینکه مفهوم شئ-گرایی جا بیفته نیاز به گذر زمان هست و باید اول مفهوم و مزایای اون رو متوجه شد بعد واردش شد و اینطور خواهید فهمید زندگی چقدر آسون‌تر و دلچسب‌تر می‌شه. روش ما ورود از کل به جزء هست، چون یک سری بحث انتزاعی فکر نمی‌کنم برای دوستان جذابیتی داشته باشه و مفهوم حرف اول رو می‌زنه ... .

جلسه دوم:



پیام انگیزشی:

"در دنیا فقط یک نفر وجود دارد که باید از او بهتر باشید و آن کسی نیست جز گذشته خودتان!!

تغییر فقط نیازِ زندگی نیست، خودِ زندگی ست ..."

مروری غنی شده بر جلسه قبل:

- یادگرفتیم تنه اصلی برنامه در تابع main قرار می گیره. زندگی به ازاء هر آکولاد باز { شروع می شه و این زندگی با آکولاد بسته } به مرگ منتهی می شه. هر { و } ای که ساخته بشه اصطلاحا یه Scope جدید ایجاد می شه. برای مثال شیئ Ali از کلاس کاماندو در scope تابع main تعریف شده، پس فقط در همین اسکوپ می شناسنش و بیرون از تابع main شناخته شده نیست. اصطلاحا شیء Ali محلی (local) تعریف شده، به تعبیر دیگری فقط بچه های محل می شناسنش، بچه هایی که در محله ی main نیستن نمی شناسنش.
- برای اینکه پردازنده بعد از قدری فعالیت به مرگ نرسه، قبل از { یک حلقه ی while(1) گذاشتیم که خط برنامه مداوم در حلقه گیر کنه.
- یادگرفتیم امکانات و تجهیزات رو بسته به سناریوی مد نظر انتخاب کنیم و با #include بین ۲ تا دابل کوتیشن " " به سناریو اضافه کنیم.
- در زبان C هر کدام از موارد include شده یک کتابخانه خوانده می شود و شامل تعدادی { متغییر و تابع } می باشد.

❖ در زبان C++ هر کدام از موارد include شده یک کتابخانه است که کلاس هم خوانده می شود و شامل تعدادی { فیلد و متود } می باشد.

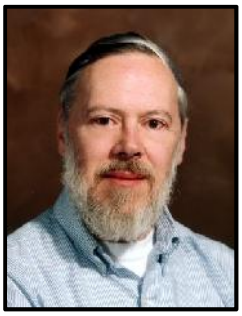
همین طور دیدیم که از هر کلاس می توانیم تعدادی شئ تعریف و در واقع در سناریو خلق کنیم. این اشیاء تمام ویژگی ها و رفتارهای کلاسی را که ازش تعریف شدند دارا می باشند.

فیلد: ویژگی های کلاس و متود: روش و عملکردهای یک کلاس هستند. برای مثال در کلاس kolt {ظرفیت گلوله در خشاب، رنگ اسلحه، قطر کالیبر اسلحه و ... ها} در قالب فیلد تعیین می شوند.

در کلاس kolt متود shoot و در کلاس camando متود look یک رفتار و عملکرد بودند.

فیلدها مثل متغییرها و متودها مثل تابع ها در C هستند، اما نباید به جای همدیگر اشتباهی استفاده بشوند. وقتی از فیلد و متود صحبت می شه تجهیزات و قابلیت ها را در قالب کلاس ها داریم تصور و دسته بندی می کنیم. اما متغییر و تابع هیچ یک از مفاهیم دسته بندی رو شامل نمی شه و یکی از ضعف های بزرگ C همین بود و منجر به بوجود آمدن C++ شد.

خیلی راحت حرف از C و C++ می زنیم نامردیه که از بزرگ مردهای سازندشون یاد نکنیم و یا اصلا نشناسیمشون!



Dennis Ritchie سازنده ی زبان C هست (در اوایل دهه ۷۰ میلادی) که متاسفانه با استیو جابز در یک ماه فوت کردند.

Bjarne Stroustrup توسعه دهنده ی C++ هست که اساسا " C + کلاس ها " نامیده شده و برای غنی کردن زبان C بوده (در آزمایشگاه Bell و در سال ۱۹۸۳)

خب بحث شئ گرایی تا همین جا کافیه!

با بحث هایی در مورد ظرف ها بر مبنای بیت (bit) و بایت (Byte) جلسه رو ادامه می دیم ...

ممکنه این قسمت رو دوستان بدونن، اگر می دونید می تونید این قسمت رو نخونده رها کنید ...

سوال: چرا نیاز به ظرف ها داریم؟

پاسخ: برای نگه داشتن اطلاعات!

شما برای آب خوردن نیاز به لیوان دارید، برای درست کردن شربت به پارچ و ... . موقه نوشتن سناریو نیاز به پردازش اطلاعات داریم، پس باید اطلاعات رو در ظرف های مناسب قرار بدیم و ذخیره کنیم و در زمان نیاز اطلاعات ذخیره شده رو برداریم و پردازش کنیم. گاهی اطلاعات را خودمان تولید می کنیم و گاهی اطلاعات رو از بیرون می گیریم.

بعضی وقت ها اطلاعات به اندازه ی گالن ۲۰ لیتری هستند، پس باید تو ظرفی از نوع گالن ذخیره بشن، تو مخزن هم می شه ریختش، اما اگر تو پارچ بریزید فقط یک پارچ رو تونستید ذخیره کنید، باقیش می ریزه رو زمین و دیگه قابل جمع آوری نیست و از دست می ره ...

کوچکترین واحد سازنده ی حافظه بیت هست، بیت رو یک لیوان در نظر بگیرید (ظرفیت گنجایش) و هر ۸ بیت، یک بایت رو تشکیل می ده، پس بایت می شه ۸ تا لیوان یه چیزی تو مایه های یک پارچ آب. اطلاعات بزرگ تر رو می تونیم در یک گالن ۲۰ لیتری جا بدیم، معادلش ۲ بایت درکنار هم هستند که تشکیل یک ظرف ۱۶ بیتی یا ۲ بایتی می دن.

البته هنوز با حجم اطلاعات بیشتری سرو کار داریم و می تونیم در مخزن آب خانه اون ها رو

منتقل کنیم، معادلش ٤ بایت در کنار هم هستند که تشکیل یک ظرف ۳۲ بیتی یا ٤ بایتی می دن!

واو ...! خدای من هنوز نیاز به حجم بیشتری از اطلاعات هست؟ پس مخزن آب شهر تقدیم به شما! معادلش ۸ بایت در کنار هم هستند که تشکیل یک ظرف ٦٤ بیتی یا ۸ بایتی می دن!

بله اینجوریاست! بذل و بخشش البته حدی داره و اگر آدم مُصرفی باشید به مشکلات جدی می خورید! نمونه ای از اصراف اینه که هر پارچ آب رو در گالن ۲۰ لیتری ذخیره کنید یا بدتر در مخزن آب خانه!

به یاد داشته باشید که همیشه منابع ما محدود هستند مخصوصا در زمینه ی Embbeded systems، پس در مصرف منابع صرفه جویی کنید!

همون طور که خدمتتون عرض کردم سناریو برنامه نویسی شباهت زیادی با سناریوی زندگی داره ... !

قدری جزئیات در مورد متغییرها:

یک بیت گنجایش یک بیت را دارد، یعنی یا ۰ هست یا ۱، یا true هست یا false، یا HIGH است یا LOW ، یا ۰ ولت است یا ۳,۳ ولت، یا ۰ ولت است یا ۵ ولت، یا ...

پیوسته بین ۲ حالت (لیوان یا پر است یا خالی ← این تفکیک پذیریه ماست)

دقت کنید که تفکیک پذیریه پردازنده یک پارچ است، یعنی یک بایت. پس چه یک بیت تعریف کنید و چه یک بایت، حجم اشغال شده توسط پردازنده یک بایت است، یعنی یک پارچ. (در حالت پیش فرض)

```
2 ^ 0 = 1
bool var1 = true;
boolean var2;
```

۱بایت (یک پارچ آب) گنجایش ۸ بیت دارد یعنی مقدار ۰ تا ۲۵۵

```
2 ^ 8 = 256
unsigned char var1;
unsigned byte var2;
```

۲ بایت (یک گالن ۲۰ لیتری) گنجایش ۱۶ بیت دارد یعنی ۰ تا ۶۵۵۳۵

```
2 ^ 16 = 65536
unsigned short var;
```

٤ بایت (یک مخزن آب خانه) گنجایش ۳۲ بیت دارد یعنی ۰ تا **۴۲۹۴۹۶۷۲۹۵**

```
2 ^ 32 = 4294967296
unsigned int var1;
unsigned long var2;
```

مخزن آب شهر چون هدیه هست، بررسیش با خودتون ...

متغییر ها می تونن دارای علامت یا بدون علامت باشن، متغییرهای علامت دار اعداد منفی رو هم می تونن ذخیره کنند. مثل این می مونه که وسط پارچ، وسط گالن ۲۰ لیتری، وسط مخزن آب رو به عنوان مقدار صفر در نظر بگیرید. ظرفیت ها در حالت با علامت و بدون علامت یکسان است، اما در رنج مقادیر متفاوت.

تمام متغییرهایی که بالا تعریف شدند بدون علامت " unsigned " هستند. یعنی فقط رنج ۰ تا مقدار اعلام شده را شامل می شن.

اگر " unsigned " را از پشت تعاریف برداریم متغییرها علامت دار می شوند.

رنج ها به ترتیب:

۱بایتی ← ۱۲۸- تا ۱۲۷

۲بایتی ← ۳۲۷٦۸- تا ۳۲۷٦۷

٤بایتی ← ۲۱٤۷٤۸۳٦٤۸- تا ۲۱٤۷٤۸۳٦٤۷

دقت کنید پردازنده Zcross از نوع ARM و ۳۲ بیتی هست، پس متغییر int هم ۳۲ بیتی است، در پردازنده های ۸ بیتی ۱٦ بیتی است! لذا ممکنه جداولی ببینید که int را ۱٦ بیتی در نظر گرفته اند، اما در حقیقت سایز پایه بستگی به پردازنده دارد!

مقدار دقیق برای پردازنده های مختلف رو می شه با دستور sizeof() چک کرد.

و اما اعداد اعشاری رو چطور می تونیم ذخیره کنیم؟

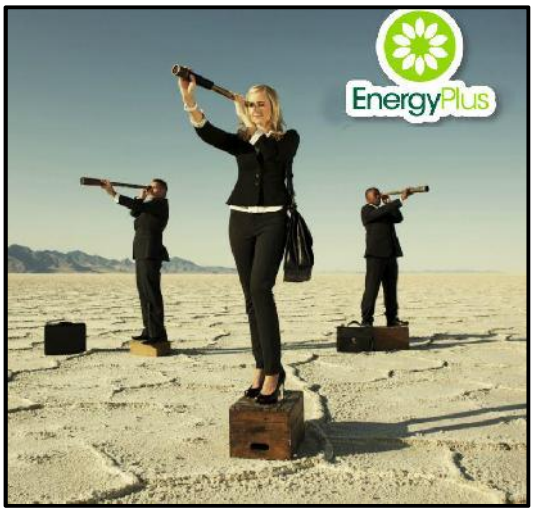
```
float var1;
double var2;
```

این متغییرها اعداد اعشاری رو با الگوریتم خاصی ذخیره می کنن و رنج مقادیری که شامل می شوند به صورت زیر است ...

float	4 byte	1.2E-38 to 3.4E+38
double	8 byte	2.3E-308 to 1.7E+308

در زبان C++ متغییرها رو هر جایی از برنامه می تونید تعریف کنید، با این وجود توصیه می کنیم برای نظم بیشتر متغییرهای اساسی در کنار هم و قبل از اجرای متودها تعریف شوند.

جلسه سوم:



پیام انگیزشی:

انسان هم میتواند دایره باشد و هم خط راست انتخاب

با خودتان هست،

تا ابد دور خودتان بچرخید یا تا بی نهایت ادامه بدهید.

تا اینجای کار یک دید کلی نسبت به بحث شئ گرایی پیدا کردیم. اگر جلسات قبل رو مطالعه نکردید، در انجام این کار لحظه ای درنگ نکنید! چون جلسات سریالی هستند ...

فهمیدیم شئ های متفاوتی با نام های مختلف می توانیم از یک کلاس خلق کنیم و بسازیم، که این شیئ ها ویژگی های کلاس شان را دارا هستند.

همینطور فهمیدیم کلاس ها شامل (فیلدها و متودها) هستند و کاربردی مشابه { متغییر و تابع } دارند، اما از نظر مفهومی تفاوت دارند. اما واقعا این کلاس ها چی هستند؟ و چه شکلی هستند؟

```
Class kolt
{
        public:
                Kolt();
                void reload();
                void shoot();
                .
                .
                .
        private:
```

```
    uchar khanNum;
    uint16_t color;
    uint weight;
    .
    .
    .
};
```

برای نمونه کلاس kolt رو در مثال بالا مشاهده کردید، که برخی متودها و فیلدهاش رو با هم دیدیم. برای استفاده از این متودها و فیلدها در ابتدا باید یک شئ خلق کنیم.

```
Kolt kolt;
```

در این لحظه constructor کلاس کارش رو انجام می ده و شیئ رو خلق می کنه، در کلاس بالا Kolt() در واقع constructor (سازنده) کلاس kolt هست. با خلق هر شئ اولین محل گذر "سازنده" ی هر کلاس می باشد و به صورت خودکار فراخوانی می شود و تنظیمات اولیه را برای اشیاء انجام می دهد.

برای دسترسی به متود و فیلد از {dot} بعد از اسم شئ استفاده می کنیم. به این ترتیب طبق دسته بندی کلاس ها وارد قفسه ی کلاس Kolt می شویم و متود مورد نظر را فراخوانی می کنیم.

```
Kolt.shoot();
```

در اینجا kolt یک شئ از کلاس Kolt می باشد و shoot متود کلاس Kolt است که طبیعتا شیئ ساخته شده از این کلاس هم شامل این متود می باشد. پس به صورت بالا برای هر شئ می تونیم دسترسی به متودها و فیلدها داشته باشیم.

این کلاس ها که در قالب کتابخانه به سناریو include شدند، فایل هایی با پسوند ".h" هستند که شامل ساختار و تعاریف می شوند، فایل دیگری هم با پسوند ".cpp" در کنار ".h" ها قرار می گیرد که در واقع implementation (پیاده سازی) فایل ".h" می باشد. ".cpp" در EasyMCU به صورت خودکار به سناریو پیوست سناریو می شود و این مطلب صرفا جهت تکمیل و داشتن دید بهتر عنوان شد.

بیشتر از این وارد جزئیات شئ گرایی نمی شیم و در همین حد اطلاعات از شئ گرایی نیاز ما را در استفاده از EasyMCU برطرف می کند.

خب حالا یک سناریو رو با هم ببینیم که درش شرط باشه.

یک بچه به نام علی خلق می کنیم که وقتی گرسنه بشه یک عدد سیب می خوره و در غیر این صورت می گه خوابمه!

```
#include “child.h”
#include “human.h”
#include “fruit.h”

int main()

{

	Child ali;
	Fruit apple;

	bool hungry = false;

	hungry = ali.isHungry();

	if(hungry == true)
	{
		ali.eat(apple);
	}
	else
	{
		ali.say(“I wanna sleep!”);
	}

	while(1);

}
```

- در این سناریو ابتدا علی و سیب رو خلق کردیم.

چون می خواستیم شرط چک کنیم، یک متغییر تعریف کردیم که از نوع bool باشه.

- دستور شرطی رو با if به معنی "اگر" شروع می کنیم و شرط مد نظر رو در پرانتز روبروش می نویسیم.

در شرطی که در سناریوی بالا گذاشتیم مقدار پرانتز اگر true به معنی "صحیح" شد یعنی شرط برقرار شده و دستورات زیر if اجرا می شوند، اگر مقدار پرانتز false به معنی "غلط" شد دستورات زیر if اجرا نمی شوند، بلکه دستورات زیر else اجرا می شوند.

اگر علی گرسنه باشه با متود ()eat و با ورودی apple یک سیب می خوره!

در غیر این صورت با متود ()say اطلاع رسانی می کنه که خوابش می یاد!

- برای اینکه تابع main به مرگ نرسه قبل از "}" پایان حلقه ی تهی بینهایت ;(1)while رو نوشتیم که خط برنامه در حلقه گیر کند و به مرگ نرسد.

حتما دقت کردید که سناریو فقط یک بار در طول زندگی شرط گرسنه بودن علی رو چک می کنه و بعد کار خاصی انجام نمی ده، یه جورایی به کما می ره!

همه می دونیم زمین گرد هست و تمام مدارات و کره ها گرد هستند و نقطه ای به نام شروع و پایان وجود ندارد، فقط حرکت هست که به زندگی معنی می ده ... ، تاریخ هایی که بر کرات گذشته و مجدد هم تکرار و تکرار و تکرار می شوند (جنگ و کشتار و فساد و ...)

در ابتدای خلقت خدا آسمان ها و زمین و کرات و ... رو خلق کرد و بعد یک چرخه بر رفت و اومد شب و روز و گردش فصول و ... ها قرار داد و ... .

تا اینجا کاری که ما کردیم بخش خلقت و تنظیمات اولیه بوده، سناریوهایی که نوشتیم هیچ حرکت و چرخشی نداشتند، یجورایی خلقتی که در سناریوهامون داشتیم بیهوده بوده ... !

پس بیاید یه سناریویی بنویسیم که چرخش داشته باشه و عالم پویا ای داشته باشه که یک سری موارد

مثل گذر شب و روز درش لحاظ شده باشه و علی هر وقت گرسنه اش شد بتونه سیب بخوره و در غیر این صورت بگه خوابم می یاد.

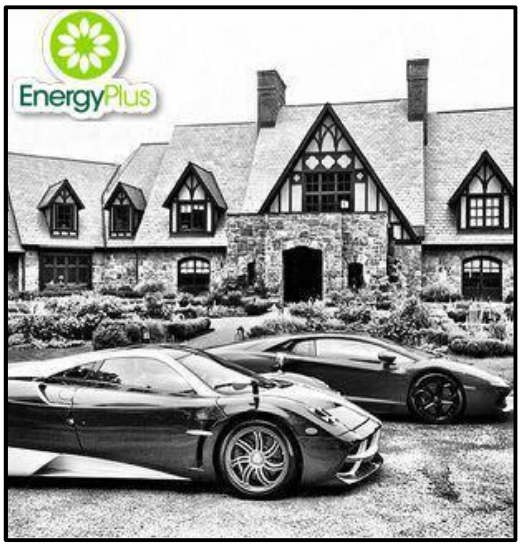
```
#include “child.h”
#include “human.h”
#include “fruit.h”
#inlude “universe.h”
int main()
{
	Univers earth;
	Child ali;
	Fruit apple;
	bool hungry = false;

	while(1)
	{
		if(earth.state())
		{
			earth.dayLight();
		}
		else
		{
			earth.night();
		}
		hungry = ali.isHungry();
		if(hungry == true)
		{
			ali.eat(apple);
		}
		else
```

```
            {
                ali.say("I wanna sleep!");
            }
        }
}
```

- همینطور که دیدید ابتدا زمین و علی و سیب رو خلق کردیم.
- بعد یک حلقه تا ابد خلق کردیم و به زندگی چرخش دادیم. شرط هایی رو به صورت متناوب چک می کنیم و مطابق با نتیجه شون اگر شرط برقرار باشه یا نه عمل می کنیم.

حالا سناریوی ما پویا شده ، هم روز رو داریم و هم شب رو ، همینطور علی هر وقت گرسنه شد می تونه سیب بخوره!

جلسه چهارم:



پیام انگیزشی:

هیچ وقت فراموش نکن که چرا شروع کردی!

مروری بر جلسه قبل:

در جلسه قبل دیدیم که در ابتدای برنامه اجزا رو خلق می کنیم و بعد از اون تنظیمات اولیه یا به عبارت دیگه کارهایی که فقط یکبار در زمان زندگی باید رخ دهند را قرار می دهیم.

از آنجا که کل زندگی در رفت و آمد مکرر و چرخش معنی پیدا می کنه یاد گرفتیم کارهایی که مداوم باید انجام شوند را باید در حلقه ی بینهایت زندگی یا به عبارت دیگه چرخ روزگار بذاریم!

دیدیم که به عنوان شرط دستور if هر چیزی که نتیجه اش درست یا غلط باشد می توانیم قرار دهیم. حتی می تونیم تابع فراخوانی کنیم و مقدار برگشتی و خروجی تابع یا متود به عنوان نتیجه شرط می تواند مورد استفاده قرار گیرد.

می توانیم else را استفاده نکنیم.

اگر بخواهیم چند شرط به موازات هم چک شود می توانیم چند if پشت سر هم بنویسیم:

```
if(condition1)
{
}
if(condition2)
```

```
{
}
.
.
.
if(condition N)
{
}
```

همینطور اگر بخواهیم در بین چند شرط تنها یکی از آنها که اولویت بالاتر دارد اجرا شود از else if استفاده می کنیم.

```
if(condition1)
{
}
else If(condition2)
{
}
.
.
.
else If(condition N)
{
}
else
{
}
```

دقت کنید که مفهوم "{" و "}" همه جا یکسان است و درون این scope هر چیزی می توانید بنویسید، متغییر تعریف کنید، تابع فراخوانی کنید و یا حتی شرط بگذارید و مجدد از if استفاده کنید و به

هر تعداد بخواهید می توانید if تو در تو داشته باشید ... .

همینطور دیدید که "==" به مفهوم شرط برابری است ، و نباید با "=" اشتباه شود که مفهوم انتساب را دارد. در شرط ها به صورت عمومی از " = " استفاده نمی کنیم، در غیر این صورت جواب شرط همیشه "صحیح" هست!

- برای نابرابری از " != " استفاده می شود ، مفهومی عکس " == " دارد.
- همینطور در شرط ها می توانید از علامت های ">" "=>" "<" "=<" هم استفاده کنید.

و البته در یک پرانتز بجای یک شرط ، می توانید چند شرط مختلف داشته باشید.

اگر از "&" به معنی " و " بین شرط ها استفاده کنید، به این مفهوم است که تنها در صورت صحیح بودن همه ی شرط ها ، شرط if صحیح می شود.

```
if(con1 & con2 & con3 & … & conN )
```

اگر از " | " به معنی " یا " بین شرط ها استفاده کنید، به این مفهوم است که تنها در صورت درست بودن حداقل یک شرط از بین همه ی شرط ها ، شرط if صحیح می شود.

```
if(con1 | con2 | con3 | … | conN )
```

برای تعیین حدود می توانید از پرانتز استفاده کنید، و می توانید به صورت ترکیبی از شرط ها استفاده کنید.

```
if((con1 > 50) & (con2 != 4) | ((con3 & 0x02) == 0x02 ))
```

پس تا اینجای کار باید یادگرفته باشید در حد قابل قبولی از if() استفاده کنید.

قبل تر در مورد حلقه ی while صحبت کرده بودیم که با گذاشتن شرط ۱ برای while() یک حلقه ی بینهایت درست می کردیم. چرا که ۱ به عنوان شرط به مفهوم همیشه صحیح است.

در واقع پرانتز جلوی while هم یک عبارت شرطی مثل if() قبول می کند و تا زمانی که شرط صحیح باشد حلقه پیوسته دستورات را پشت سر هم و حلقه وار اجرا می کند، به عبارت دیگه تا زمانی که مثلا قیامت نشده زمین به دور خودش می چرخه ... .

اگر شرط حلقه نقض شود، حلقه متوقف می شود و خط برنامه از حلقه در می آید و فرامین بعد از scope حلقه را اجرا می کند.

```
#include “headers.h”
int main()
{
	Universe universe;
	.
	.
	.
	Planet  earth;
	// The routin of finite world
	while(universe.getState() != universe.ghiyamat )
	{
		universe.dayAfterDay();
	}
	// ghiyamat start
	earth.end();
	moon.end();
	.
	.
	.
	universe.makeRealWorldVisible();
	while(1)
	{
		universe.stayInfinite();
	}
}
```

در مثال فوق کتابخانه ها رو در قالب یک header کلی include کردیم که صفحه زیاد شلوغ نشه، در واقع به مواردی که include می کنیم، علاوه بر کتابخانه ها و یا کلاس ها به فرم کلی می تونیم header هم بگیم.

وقتی از دوتا " / " به نام اِسلَش پشت سر هم استفاده می کنیم ، می تونیم جملات راهنمایی رو برای خودمان بنویسیم. به این کار اصلاحا comment (کامنت) گذاری می گن و این خط ها جزء برنامه نوشته شده نیستند و صرفا جهت راهنما برای تفهیم کد برای خودمان نوشته می شوند. اصطلاحا کامپایل نمی شن یا به عبارت دیگه به کد ماشین تبدیل نمی شوند.

در ابتدا جهان را خلق کردیم و کرات را، برای اینکه به وجود دنیای واقعی برسیم (بریم سراغ اصل کار)، قبل از آن باید شرایطی محیا شود، اما دقیقا معلوم نیست کی محیا شود، به عبارت دیگه معلوم نیست این کارها را چند بار باید انجام دهیم.

بنابراین می بینید که از حلقه ی while مشروط استفاده کردیم (مثلا مشروط به وقوع قیامت). زمانی که قیامت بشه این شرط نقض می شه و خط برنامه از dayAfterDay دنیای فانی در می یاد و از بعد از scope حلقه while مشروط پی گیری می شه.

به این ترتیب اجزاء فعلی به پایان می رسند و تابع صور اصلی جهان فراخوانی می شه و تا ابد در حلقه while(1) خواهد ماند.

یک نمونه حلقه ی دیگر می توانیم داشته باشیم که تعداد دفعات اجرای آن مشخص باشد.

حلقه ی for()

```
for(i=0;i<10;i++)
{
}
```

به این ترتیب یک کانتر به نام i ، یک شرط مثل if() ، در مثال فوق (i<10) و یک گام برای حلقه، در مثال فوق (i++ یا i+=1) تعیین می کنیم، هر بار که scope حلقه ی for شروع شود و به مرگ برسد، i یک واحد افزایش پیدا می کند و این تا زمانی اتفاق می افتد که شرط برقرار باشد، یعنی در مثال فوق تا زمانی اتفاق می افتد که i کمتر از 10 باشد. اگر i برابر 10 شد، چون 10<10 نیست، شرط نقض می شود و حلقه ی for کاملا می میرد و خطوط برنامه از خط های بعدی ادامه پیدا می کند.

به عبارت دیگه با ۳ مقدار شروع، شرط و گام جاودانگی حلقه ی for را تعیین می کنید که چند تا جان داشته باشه!

هر بار که scope اش به " } " یعنی مرگ می رسه یکی از جان هاش رو از دست می ده. حلقه ی for و scope اش تا جایی ادامه می ده که کل جان هاش رو از دست بده و اون وقت از صحنه ی روزگار حذف می شه.

در سناریوی پایین می نویسیم که علی هر وقت گرسنه شد 5 عدد سیب بخوره.

```
#include “headers.h”
int main()
{
	Child ali;
	Fruit apple;
	while(1)
	{
		if(ali.isHungry())
		{
			for(uchar h=0; h<5; h++)
			{
				ali.eat(apple);
			}
		}
	}
}
```

در مثال فوق دقت کنید که یک متغییر جدید به نام h در حلقه تعریف کردیم!

بله چون نمی خواستیم برای این کار کوچک فضای الکی از RAM اشغال کنیم، با شروع حلقه، h متولد می شه و با مرگ کامل حلقه فضای اشغال شده از RAM هم آزاد می شه!

در واقع متغییر h در scope حلقه تعریف شده و محلی هست، یعنی فقط بچه های محله ی حلقه ی for می شناسنش.

دقت کنید بجای اینکه از حلقه ی for استفاده کنیم می تونستیم 5 بار بنویسیم ...

```
ali.eat(apple);
ali.eat(apple);
ali.eat(apple);
ali.eat(apple);
ali.eat(apple);
```

اما این کار اصولی نیست، چون باعث می شه دستور خوردن ۵ برابر فضای اضافه از Flash را اشغال کند، کنترل و خواناییش هم کم می شه.

جلسه پنجم:

پیام انگیزشی:

روزهای خوب در انتهای جاده منتظر شما هستند لطفآ از حرکت باز نایستید.

تا کنون اصول C/C++، شئ گرایی، تنه اصلی برنامه، هدرها، کلاس ها، متغییرها و حلقه ها آشنا شدیم، این جلسه از بخش C/C++ 10% را به محوریت چگونگی تعریف توابع به پایان می بریم ...!

اصلا توابع چه هستند و به درد چه کاری می خورند؟

وقتی یه پروسه در سناریو داریم که در جاهای مختلف سناریو مکرر استفاده می شود، به جایی که تمام کدهای پروسه را بارها در جاهای مختلف سناریو بنویسیم، می تونید یکبار در یک جای مشخص بنویسیم و براش یک اسم با مسما بذاریم.

مثلا سناریو ای داریم می نویسیم که علی یک بچه شیطون هست که چندین بار در روز دنبال گربه های محله اشون می گرده، اون ها رو پیدا می کنه و می گیره و در ماشین لباسشویی می اندازتشون ...! این کار رو صبح ، ظهر و همینطور عصر انجام می ده!

خوب بجایی که این ها رو مداوم تکراری در متن سناریو بنویسیم، می تونیم یک پروسه به نام شیطنت بسازیم و یک بار پروسه ی شیطنت رو با جزئیات در یکجا بنویسیم. از این به بعد هرجا علی به سرش می زد که شیطنت کنه، سناریو رو ارجاع می دیم به پروسه ی شیطنت.

مزیتش اینه که به جای اینکه سه بار پروسه شیطنت رو از اول بنویسیم، فقط یکبار می نویسیم و بارهای بعدی به اون ارجاع می دیم. طول سناریوی نوشته شده روی برگه هم کمتر می شه، به عبارت دیگه حافظه ی flash میکرو کمتر اشغال می شه! خوانایی سناریو و دسته بندیش هم راحت تر می شه!

```
#include “headers.h”

void sheytanat(void)
{
	uchar num;
	while(num == 0)
	{
		num = Ali.search(cat, street);
		if(num > 0)
		{
			num = ali.capture(cat);
		}
	}
	ali.throw(cat, washingMachine);
	washingMachine.start;()
	delay(5); // 5 minutes delay
	washingMachine.stop;()
	ali.release(cat);
}
int main()
{
```

```
        Boy  ali;
        Animal  cat;
        day  day;
        location street;
        Device  washingMachine;
        while(1)
        {
                if(day.isMorning())
                {
                        ali.getUp();
                        sheytanat();
                }
                else if(day.isNoon())
                {
                        ali.eat(food);
                        sheytanat();
                }
                else if(day.isEvening())
                {
                        sheytanat();
                }
        }
}
```

فرم تعریف تابع و استفاده از آن به صورت مثال بالا هست.

این تابع بدون ورودی و خروجی بود، به عبارتی دیگر، ورودی و خروجی از نوع void بود.

توابع می توانند دارای یک یا چند ورودی باشند اما خروجی نداشته باشند.

```
void func(uchar input1, int input2, float input3, …)
```

همینطور می توانند علاوه بر ورودی ، خروجی هم داشته باشند.

```
int output func(uchar input1, int input2, float input3, …)
{
        int result;
        .
        .
        .
        return result;
}
```

طبیعیه که هر تابع نهایتا می تواند یک خروجی داشته باشد. علاوه بر موارد فوق، تابع می تواند بدون ورودی باشد اما خروجی داشته باشد.

```
float output function(void)
{
        float result;
        .
        .
        .
        return result;
}
```

ورودی برای زمانی هست که تابع قراره پردازشی روی ورودی انجام دهد. خروجی برای زمانی است که تابع از انجام پروسه اش مقداری را به عنوان خروجی به برنامه برگشت دهد، مثلا نتیجه ی یک محاسبه.

با این توضیحات یه سری قضایای دیگه بایستی براتون روشن شده باشه، قضایایی که در طول این جلسات به چشمتون خورده.

جلسه رو با کد اسکی ASCII و آرایه و استرینگ (String) ادامه می دیم ...

هر کاراکتری که روی نمایشگر یا LCD چاپ می شه، یک بایت فضا اشغال می کند و یک کد معرف دارد، جدول کدهای اسکی را می توانید گوگل کنید و با جزئیات همه کدها رو بررسی کنید. ASCII Table

مثلا کد اسکی دکمه ی Enter روی کیبرد برابر عدد " ۱۳ " هست یا کد اسکی عدد یک برابر "٤٩" هست. کد اسکی حرف c برابر عدد "۹۹" و حرف C برابر عدد "٦٧" هست.

پس کاراکترها اینجوری شناسایی میشن!

و اما آرایه: چند تا ظرف از یک جنس را که به هم متصل کنیم یک آرایه داریم.

```
int var1[9];
char var2[16];
```

آرایه ها را از هر نوع ظرفی می شه تعریف کرد، همینطور تعداد ظرف های به هم چشبیده را می شه در براکت [ ] تعیین کرد. روش خواندن آرایه به این صورته: مثلا آرایه ی var1 از نوع int هست و ۹ خانه دارد.

دقت کنید در زمان استفاده از var1 در براکت روبروش می تونیم اعداد بین ۰ تا ۸ قرار بدیم. مجاز نیستیم از ۹ استفاده کنیم، دلیل این هست که آرایه ها از ۰ شروع می شوند و اگر تعداد بشماریم از ۰ تا ۸ جمعا ۹ خانه داریم. پس مفهوم تعریف و استفاده رو دقت کنید.

استرینگ (String):

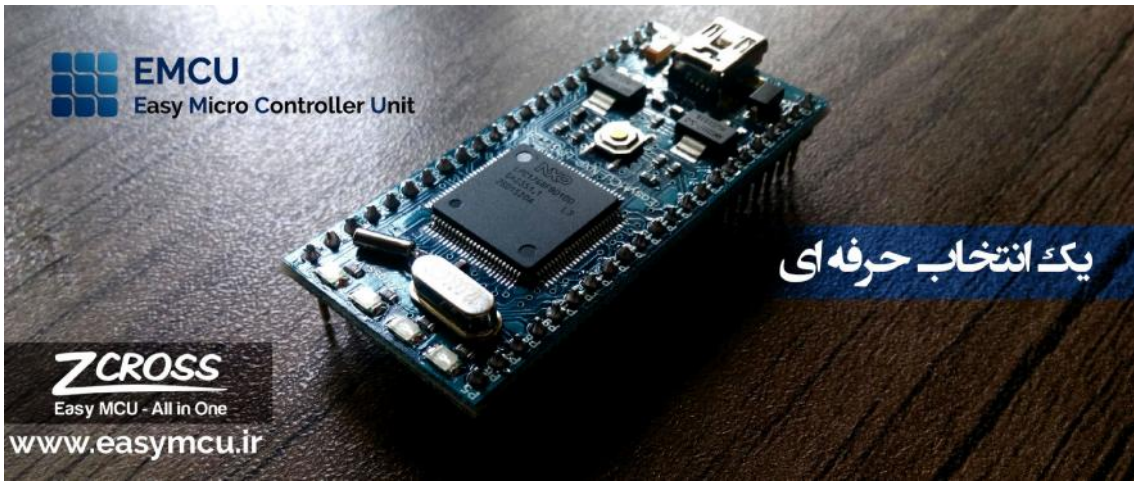
استرینگ در واقع یک کلاس هست که برای کار با کاراکترها و نوشته ها ازش استفاده می کنیم و متود های بدرد بخوری مثل پیدا کردن، حذف کردن و ... کاراکتر یا نوشته ی خاصی رو به ما می ده.

هر نوشته ای (مثلا یک جمله) در واقع در یک آرایه از نوع char ذخیره شده است و با مقدار NULL (‘\0’) پایان می یابد. علامت NULL خودش یک بایت فضا اشغال می کند و به ما کمک می کند که پایان جمله را تشخیص دهیم و جمله ی استاندارد باید با NULL پایان یابد.

این موارد هم گوشه ی ذهنتون داشته باشید، به موقعش بدردتون می خوره ...

به شما دوستان عزیزی که برای پیشرفت خودتون این ۵ جلسه از آموزش C/C++ رو با ما بودید تبریک می گم، قطعا شما جزء معدود افرادی هستید که فکرای بزرگی در سر دارید، قصد تغییر دارید، همچنان با ما باشید و به راهتون ادامه بدید، آخر جاده اتفاق های خوبی در انتظارتون هست ...

اگر همراه های خوبی بوده باشید قطعا متوجه شدید که در طول این ۵ جلسه درکتون از برنامه نویسی یه سر و گردن بالاتر از قبل رفته و البته تمام این ها مقدمه و بیس کار بود. توصیه می کنم جلسه های بعدی که به مراتب جالبتر خواهد بود رو از دست ندید ...!



https://telegram.me/EasyMCU

http://www.easymcu.ir

طعم خلاقیت را با ایزی. ام. سی. یو (EasyMCU) تجربه کنید

ارتباط با ما:

انجمن : http://forums.easymcu.ir

ایمیل : info@easymcu.ir