

Configurando a IDE do Arduino para facilitar sua utilização e a visualização/depuração de Códigos

revisão-00, agosto de 2019

Elcids H. das Chagas

Obs.:

- ***este documento foi escrito considerando a versão 1.8.8 da IDE do Arduino.***

Sobre as funcionalidades da IDE do Arduino:

O objetivo aqui, é configurar a **IDE** do **Arduino**, de forma que ela fique mais amigável para visualização de código (programa ou "sketch" na **Linguagem C/C++**), e também facilite o processo de depuração (especificamente referindo-se às mensagens do **Processo de Compilação**).

Mas antes de tratar das configurações, vamos relembrar algumas das funcionalidades existentes na IDE, pois em alguns momentos certamente serão abordados pontos pouco conhecidos, e muitas vezes significativos para quem usa a IDE do Arduino.

Apesar de ser bem conhecida, nunca é demais lembrar das funcionalidades existentes na janela da IDE do Arduino, principalmente porque a todo instante há novos usuários descobrindo o "Mundo Arduino".

Não vamos falar sobre a instalação da IDE, uma vez que na Internet há uma infinidade de sites que orientam sobre isso, inclusive o próprio site oficial do Arduino (para o Windows, o link é este: www.arduino.cc/en/Guide/Windows). Há também a versão da IDE para ser usada "online", chamada de "**Arduino Web Editor**", a qual é relativamente amigável (e tem uma tendência para ser mais usada que a IDE instalada em computadores). Para saber mais sobre a versão Web, veja este link: https://create.arduino.cc/projecthub/Arduino_Genuino/getting-started-with-arduino-web-editor-on-various-platforms-4b3e4a.

Uma vez instalada a IDE, ela já inclui as placas "oficiais" da Plataforma e as respectivas bibliotecas (as "**Libraries**" ou simplesmente "**Libs**").

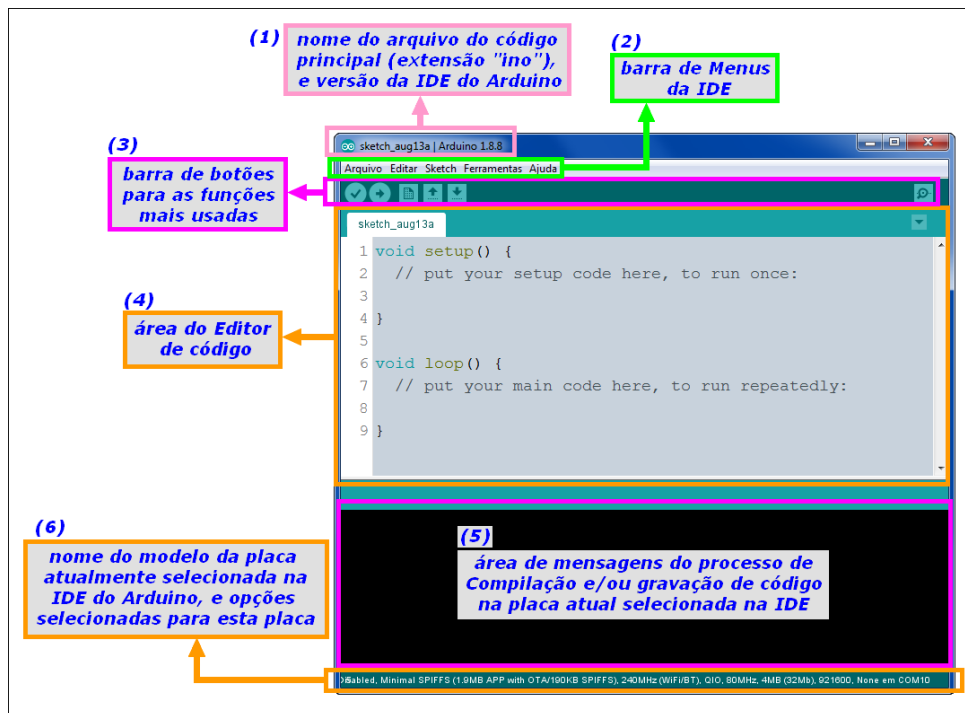
A sigla **IDE** significa "**Integrated Development Environment**", ou seja, Ambiente de Desenvolvimento Integrado. A palavra "Integrado" significa que é uma interface que **integra** diversas funcionalidades, onde as principais são:

- 1) Edição de código** (e recursos relacionados).
- 2) Processo de Compilação** para a **Linguagem C/C++** (além do **Compilador**, inclui o **Pré-Processador** e o **Linker** da Linguagem).
- 3) Processo de Gravação de código** nos Processadores das placas (no Arduino, este processo é também chamado de "upload" ou "carregar").
- 4) Assistência para atualização** da própria IDE e das Bibliotecas instaladas.

Os itens mais interessantes para se abordar são o **2)** e o **3)**, ou seja, sobre o **Processo de Compilação** (bastante importante) e sobre o **Processo de Gravação de código**. No entanto, isto exige um certo nível de detalhes e

um tratamento especial para cada processo, pois a quantidade de informações e conceitos é relativamente grande. Então isto não será abordado neste texto, mas posteriormente poderei escrever um artigo sobre esses processos, com características mais técnicas.

Após aberta, a aparência da IDE é mostrada na figura a seguir, onde também estão identificadas as principais funcionalidades da Interface:



Na figura anterior estão numerados de (1) a (6), itens da Interface que são descritos/elucidados a seguir:

- (1) **Nome do Arquivo do código principal e versão da IDE:** aqui é exibido o nome do arquivo principal que será compilado (com a extensão ".ino"). Também é exibido o número da versão da IDE.
- (2) **Barra de Menus da IDE:** aqui é possível acessar funcionalidades da IDE. Algumas das mais usadas são abrir arquivos e exemplos, ajustar as configurações da IDE, selecionar o modelo de Placa a ser usado e suas opções específicas, acessar a "ajuda", e gerar o **arquivo "HEXA"** para ser usado em simuladores ou em gravadores externos (no menu "**Sketch**" → "**Exportar binário compilado**").
- (3) **Barra de botões:** nesta versão da IDE, existem 6 botões, que permitem criar/abrir/salvar arquivos, compilar/gravar o código na placa atualmente em uso, e acessar o **Monitor Serial do Arduino**. Ver mais detalhes à frente, onde cada botão é descrito.

- (4) **Área do Editor de código:** aqui aparece o texto do código atual sendo trabalhado. É um Editor com funcionalidades semelhantes ao Notepad do Windows, ou seja, não há grandes recursos de edição. Mas é capaz de destacar palavras reservadas (veja a palavra "**void**" na figura, em cor diferente do restante do texto), e "palavras chaves" (estas são "marcadas" nas Bibliotecas para serem destacadas no Editor).

Este Editor também possui a funcionalidade de "*colapsar*" *funções* e *estruturas de código* (iniciadas pelo "{" e terminadas pelo "}"). Na área do Editor, é possível ter várias abas (semelhantes às aquelas usadas em navegadores de internet), onde em cada uma poderá ser editado um arquivo (se estiverem na mesma pasta do código principal, já serão abertos automaticamente pela IDE). O **código principal** a ser compilado, e cujo nome aparece no topo da janela da IDE, também aparecerá nesta área para ser editado).

Na figura, se pode notar que a cor de fundo no Editor, está na cor "cinza". Isto ocorre quando um **Editor "externo"** é utilizado. Nesta condição, não é possível editar nem salvar o código no Editor da IDE. Só é possível visualizar o código, e usar algumas funcionalidades do Editor (por exemplo "localizar" texto). Para mais detalhes sobre o uso de um Editor externo, veja mais à frente a seção "**Configurando as Preferências da IDE**".

- (5) **Área de mensagens do processo de Compilação e/ou gravação de código na placa atual selecionada na IDE:** essa área deveria ser uma das mais acessadas na IDE, mas infelizmente isso não costuma ocorrer, provavelmente porque a maioria dos usuários do Arduino desconhece sua importância. Aqui são exibidas mensagens resultantes do **Processo de Compilação**. Porém quando se instala a IDE, quase nenhuma mensagem é exibida, e para ver as mensagens é preciso configurar sua exibição, o que será mostrado mais adiante (na seção "**Configurando as Preferências da IDE**"). O Processo de Compilação, consiste de diversas etapas: execução do **Pré-Processador**, execução do **Compilador**, execução do **Linker**, e execução do **Gravador de Código**. Normalmente, o Pré-Processador da Linguagem não exibe mensagens (embora possa exibir, caso encontre violações das regras do Pré-Processamento). Mas as demais operações do Processo de Compilação, exibem diversas mensagens. Estas podem ser classificadas basicamente em 4 tipos:

5.1) Mensagens de Warning: são mensagens importantes. Mas a maioria dos usuários costuma ignorar os **Warnings**, porque muitas vezes não sabem o que eles significam (e infelizmente também não pesquisam sobre eles no vasto mundo da Internet). Um *Warning* não vai impedir seu código de ser compilado, mas você deve entender que o **Compilador** está sendo legal e te mandando o seguinte recado: "*querido, eu entendo que você está querendo fazer*

isso, mas você está usando um método inadequado, e eu vou quebrar o seu galho e deixar passar, mas saiba que você pode ter problemas mais à frente se não mudar isso no seu código". Acho que não é preciso dizer mais nada, não é? Uma coisa importante: alguns *warnings* aparecem apenas na primeira vez que o código é compilado, e "desaparecem" a partir da segunda compilação, então é preciso ficar atento às *mensagens de warnings* especialmente na primeira compilação.

Sempre é exibido o número da linha do código no Editor, à qual o *Warning* se refere, e muitas vezes o Compilador exhibe também sugestões de como eliminar aquele *Warning*.

5.2) Mensagens de Erro: essa todos já viram, e quando um erro ocorre, ele *impede* que o **Processo de Compilação** chegue ao fim com sucesso. Se existem diversos erros, será exibida uma mensagem para cada erro, e em cada mensagem é possível ver o número da linha à qual o erro se refere (o número da coluna também, mas não é muito útil).

É comum que um erro do tipo "bobo" (por exemplo: esqueceu-se de colocar um ";" no final da linha de código), provoque diversos outros erros serem apontados no código, quando na verdade não existem estes outros erros. Neste caso, basta se corrigir o erro "bobo" e os erros decorrentes disso irão desaparecer.

Também é comum que o Compilador exhiba sugestões de como eliminar o erro. Então é preciso entender bem o que ele está querendo nos dizer.

Em 99,99% dos casos, as mensagens de erro são geradas pelo Compilador. Mas em alguns casos, será o **Linker** que apontará erros. Estes erros apontados pelo Linker, são geralmente bem mais graves, e muitas vezes estão relacionados ao uso da Memória disponível no Processador da Placa. Mas sempre há alternativas para resolver o problema. O importante é entender a mensagem, para que se possa saber o melhor caminho a seguir para solucionar o impasse.

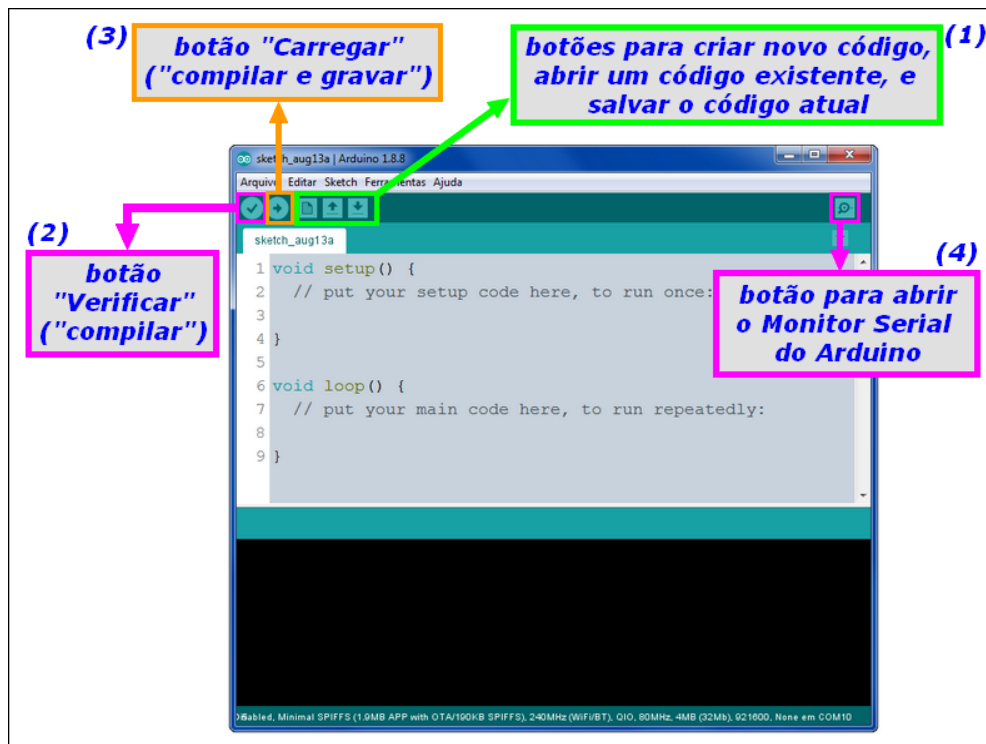
5.3) Mensagens do resultado do Processo de Compilação: estas mensagens indicam quanto de memória seu código está usando, e como este uso está distribuído (**Memória de Programa**, **Memória RAM**, área que "sobrou" para o **"Heap"** e **"Stack"**). E claro, diz também se o Processo de Compilação terminou em sucesso ou não.

5.4) Mensagens do Processo de Gravação do Código no Processador da Placa: além de exibir detalhes do Processo de Gravação (por exemplo quanto já foi gravado, em %), são exibidas mensagens de identificação da Placa (em algumas placas, somente nesta etapa é que são identificadas certas

características, como a capacidade real da Memória de Programa da placa), e eventuais solicitações para se apertar algum botão na placa (por exemplo para que a placa inicie o Processo de Gravação). E claro, também é informado se o Processo de Gravação terminou ou não com sucesso.

- (6) Nome do modelo da placa e opções selecionadas para esta placa:** na base da janela (ou "rodapé") é exibida qual placa atualmente está selecionada para ser usada. Se esta placa também possuir opções de configuração, um resumo destas configurações é exibido nesta base, além da porta "COM" correspondente à placa.

Sobre os botões existentes na IDE, vale a pena descrever rapidamente alguns deles, por motivos de clarificação (pois as descrições na própria IDE podem ser vistas como confusas e até mesmo como equivocadas). Na figura a seguir, são identificados estes botões:



Conforme a numeração de **(1)** a **(4)** na figura anterior, segue a descrição e clarificação das funções associadas aos botões:

- (1)** Botões para criar um novo código, abrir código existente, e salvar o código atual: estes três botões tem as funções mais evidentes, e praticamente dispensam esclarecimento. O único ponto que vale a pena ressaltar, é que quando se cria um novo código, uma janela adicional da IDE é aberta. Nesta nova janela pode-se selecionar outro modelo de placa completamente diferente da selecionada na

janela anterior, e caso seja o mesmo modelo de placa então os settings para esta placa podem ser totalmente diferentes daquela na outra janela.

- (2) Botão “Verificar”: este é sem dúvida o nome que foi da pior “escolha” para descrever a função do botão. É possível entender o motivo pelo qual os criadores do Arduino fizeram isso, mas não vejo isso como algo educativo. Nomes são nomes, mas se você puder escolher um que desperte alguma curiosidade no usuário e o leve a pesquisar, seria certamente melhor. O nome mais adequado para este botão, seria certamente “*compilar*” (ou “*compilar o código*”), porque quando você clica nele, é iniciado o **Processo de Compilação**. E como foi dito antes, este Processo tem várias etapas que são executadas automaticamente e em sequência: executa o **Pré-Processador**, executa o **Compilador**, e finalmente executa o **Linker**. Como resultado, serão exibidas na área de mensagens do Processo de Compilação, todo o andamento do processo, conforme já descrito anteriormente.
- (3) Botão “Carregar”: o nome deste botão não está tão “ruim” como no caso do botão “Verificar”, mas certamente poderia melhorar. Em inglês, o nome é “Upload”, e quer dizer que após um código ter sido compilado, este código será gravado na *Memória de Programa do Processador* da placa atual, a fim de que se possa ver seu funcionamento “real”. Ocorre que quando se clica nesse botão, a IDE do Arduino, executa novamente todo o **Processo de Compilação** (mesmo que ele já tenha sido feito através do botão “Verificar”), e em seguida executa o **Processo de Gravação** do código na placa. Em suma: a função do botão “Carregar” não é apenas gravar na placa, já que sempre é feito o Processo de Compilação e em seguida o de Gravação. Logo, serão exibidas as mensagens do Processo de Compilação, além das mensagens do Processo de Gravação na placa. Então, qual nome seria adequado para este botão? Talvez “*Compilar e Gravar*”.
- (4) Botão para abrir o **Monitor Serial**: a princípio parece não haver muito o que falar sobre o Monitor Serial, que é também chamado de “**Terminal do Arduino**”. Mas existe sim muito o que dizer sobre ele. Infelizmente isso não será feito aqui, pois estenderia muito o texto. Mas é preciso saber algumas coisas sobre o Monitor Serial.

Quando há uma placa conectada, ao se clicar no botão, abre-se uma nova janela que é chamada de Monitor Serial (se não existir uma placa conectada, será exibida uma mensagem de erro na área de mensagens da IDE). Essencialmente, esta janela permite que se veja os caracteres enviados pelo código do Arduino executando na placa, e que também se envie à placa os caracteres digitados pelo usuário. Esses caracteres devem ser do **padrão ASCII** (a famosa “**Tabela ASCII**”). O padrão de caracteres ASCII,

comporta originalmente 128 caracteres (variantes podem ter 256 caracteres, mas isto não é padronizado e deve ser evitado). Esses caracteres são numerados sequencialmente, e por isso é comum chamá-los de "**Códigos ASCII**". A numeração inicia em **0** (zero) e vai até **127** (ou seja, contando com o "**0**", são 128 códigos diferentes).

No entanto, os 32 primeiros códigos (ou números) são chamados de caracteres "**não-imprimíveis**", o que significa que estes caracteres não são impressos em uma tela. Mas então o que estes 32 caracteres são? Eles são "**caracteres de controle**", e recebem este nome porque permitem que se controle alguns comportamentos de quem os recebe. Através de um exemplo é mais fácil se entender isso. Por exemplo o Código ASCII de número "**10**", indica para quem receber este caractere, avançar uma linha na tela de exibição, ou em outras palavras, que terminou uma linha de caracteres, e portanto qualquer outro caractere recebido após esse "**10**", fará parte da próxima linha de caracteres na tela.

Não vale a pena aqui neste momento ficar falando sobre todos os 32 códigos de controle (alguns poderão não ter efeito, dependendo de quem os recebe). Mas é importante que se saiba que são os 32 primeiros caracteres ASCII. Em **numeração Hexadecimal** na **Linguagem C/C++**, os 32 primeiros códigos, vão do **0x00** (o "zero") até o **0x1F** (o "31"), e portanto esta é a faixa dos códigos de controle ASCII (os famosos caracteres "não-imprimíveis"). A partir do próximo código, que é o "32", todos os caracteres são "imprimíveis", o que significa que se forem enviados para um receptor que trabalhe com o padrão ASCII, estes caracteres poderão ser exibidos em uma tela (ou mesmo impressos em papel, se o receptor for uma impressora).

Logo, existem 96 caracteres "imprimíveis" (resultado de 128 códigos menos os 32 "não-imprimíveis"). No grupo "imprimível", estão os caracteres que todo mundo conhece, como o alfabeto ocidental (letras minúsculas e maiúsculas), os números de **0** a **9**, e diversos símbolos (por exemplo o "+", "#", "?", "*", etc). Aqui é importante lembrar que o primeiro caractere imprimível, o "32", é justamente o caractere de "**espaço**", ou seja, aquele correspondente à **barra de espaço** do **teclado** de um **Computador** (ou de **Smartphones** e **Tablets**). Em Hexadecimal na Linguagem C/C++, o "32" é o **0x20**, e por isso é comum se ver em programas do Arduino, alguma comparação com este valor (seja o 32 ou o 0x20) para se testar se o caractere recebido é imprimível ou não (ou seja, se a comparação indicar que o código é **menor que 0x20**, então este caractere é "**não imprimível**"). Assim o código do programa consegue determinar o que irá fazer com um **caractere ASCII** que ele tenha recebido.

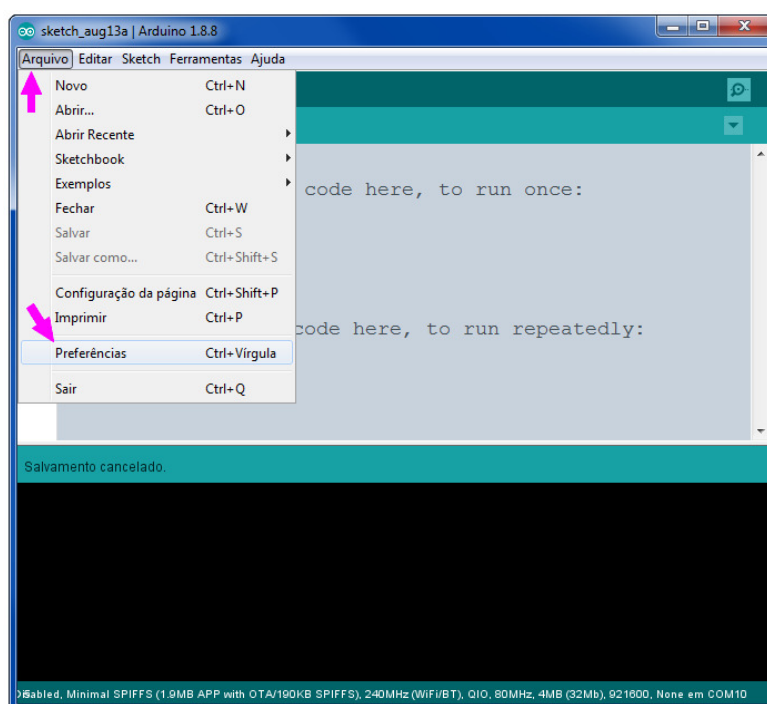
Uma curiosidade: o último caractere do padrão ASCII, o "**127**", é considerado um caractere imprimível, mas algumas aplicações consideram esse caractere como sendo o equivalente à **tecla "delete"** dos teclados de Computador, e assim quando estas

aplicações recebem o "127" (em Hexadecimal é o **0x7F**) elas aplicam no texto o mesmo efeito que a tecla delete teria. Já **Displays LCD**, ao receberem o código ASCII "127", costumam imprimir na tela um caractere igual ao caractere "espaço".

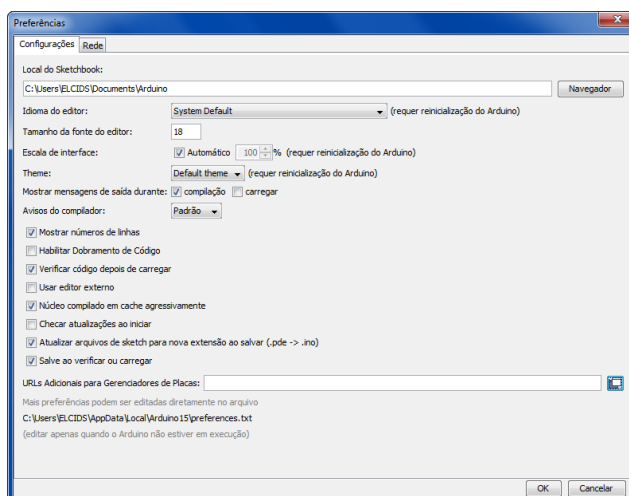
Para usar o **Monitor Serial**, é preciso que sua velocidade de comunicação (a **taxa de bits**) seja a mesma que a placa conectada está usando. Além disso o próprio "formato" de bits deve ser igual ou compatível com o formato de bits da placa. Também, é preciso configurar no Terminal o que ele deve fazer ao receber alguns caracteres ASCII não-imprimíveis (especificamente o "CR" e o "LF"), e se ele deve transmitir algum caractere não imprimível quando se digita no teclado a tecla "Enter". Por motivos de não se estender mais neste texto, essas configurações do Monitor Serial não serão abordadas aqui (e também porque são configurações um tanto intuitivas, o que permite ao usuário descobrir o efeito de cada *setting* da janela do Monitor Serial).

Configurando as Preferências da IDE:

Praticamente todas as configurações, são feitas no diálogo (ou janela) de **Preferências da IDE do Arduino**. Esse diálogo é aberto através do **Menu "Arquivo"**, opção "**Preferências**" (pode-se fazer isso também através das teclas de atalho "**Ctrl + vírgula**"). A figura a seguir mostra o acesso através do *Menu*:



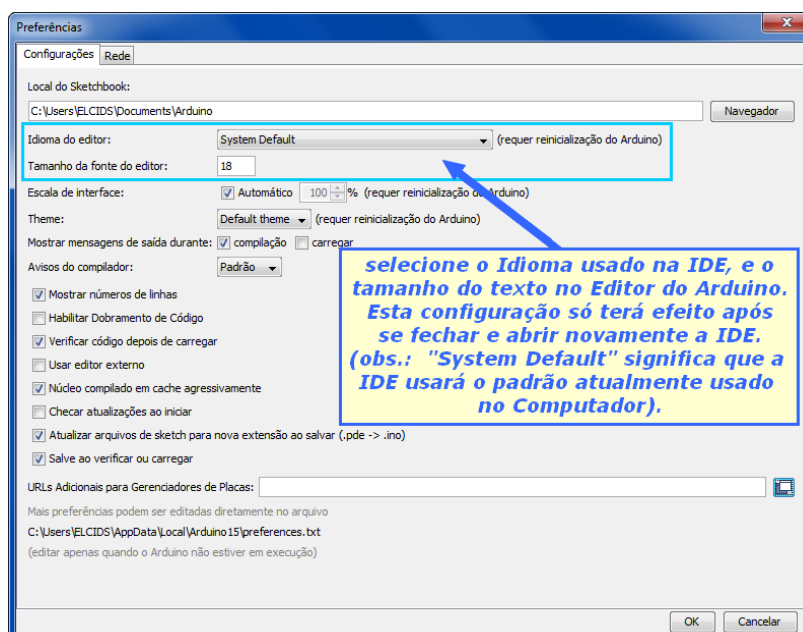
Ao ser aberta, a janela de Preferências é visualizada como mostrado na figura a seguir (esta aparência é da **IDE 1.8.8**):



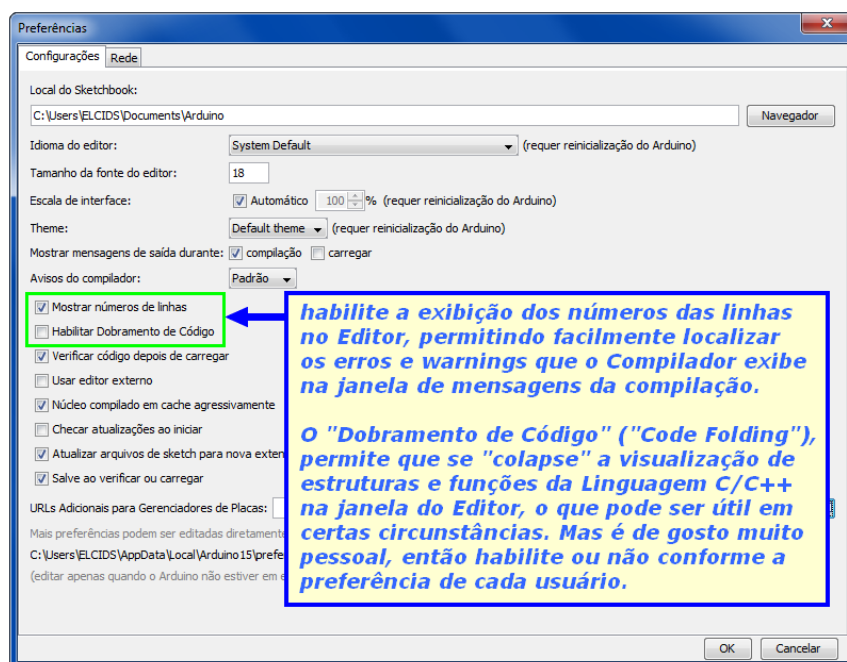
Após uma configuração ter sido feita, é preciso clicar no **botão OK** para confirmar as alterações. No entanto alguns "settings" (ajustes), só terão efeito quando se fecha e se abre novamente a IDE. Quando se quer descartar as alterações no diálogo de Preferências, basta clicar no botão "Cancelar".

Configurações Gerais e do Editor de Texto:

A figura a seguir mostra os settings para o idioma utilizado na IDE, e o texto (a "fonte") usado no **Editor de Texto**. Configure conforme achar mais conveniente. Observe que estes settings terão efeito somente após se fechar e abrir novamente a IDE.



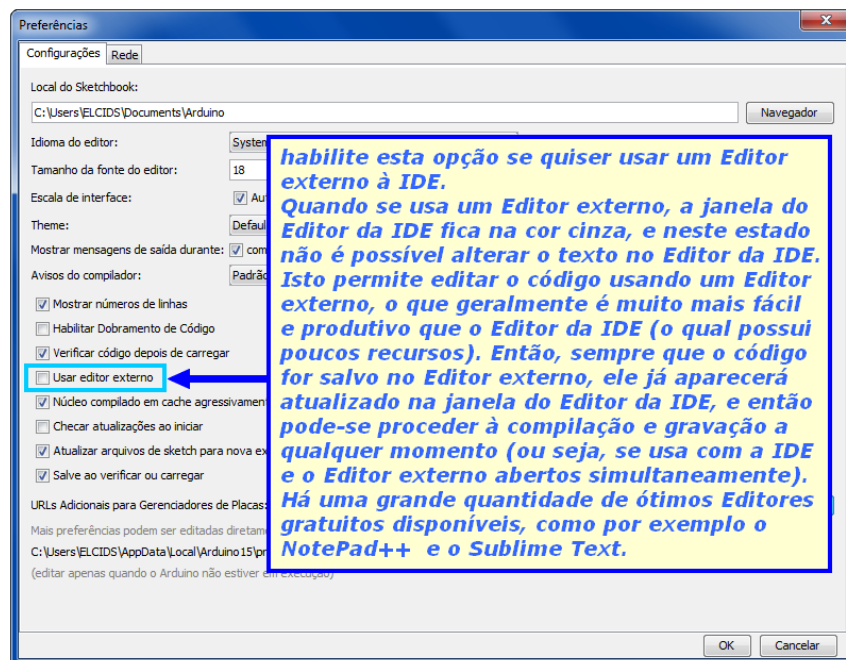
Na próxima figura, podem ser vistas *configurações adicionais* para o **Editor de Texto**. A figura é auto-explicativa:



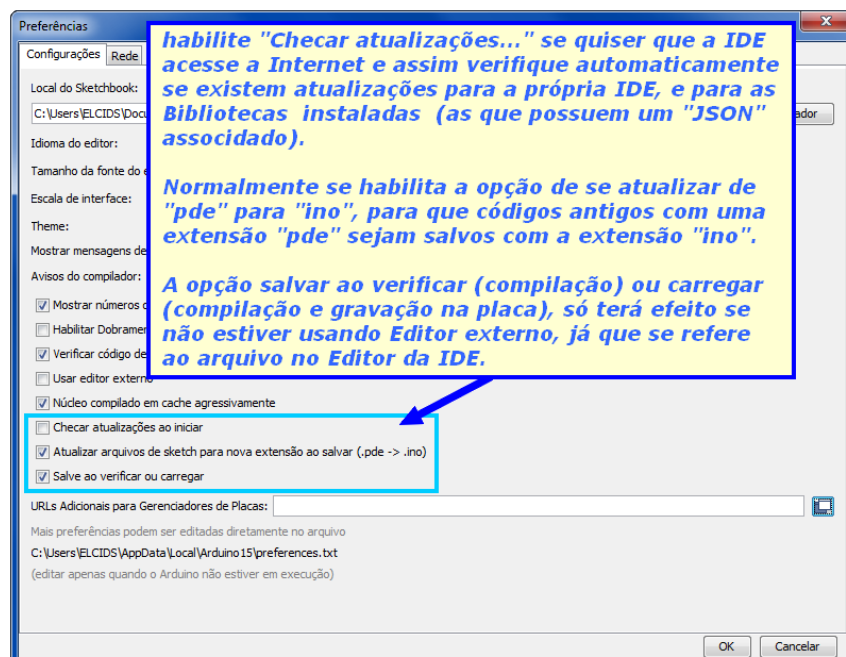
A exibição dos **números de linhas** no Editor, ajuda muito, pois torna fácil a localização dos *warnings* e *erros* apontados no Processo de Compilação. Mesmo que se esteja usando um *Editor "externo"* (ver essa configuração mais adiante), ainda é conveniente mostrar os números de linhas no Editor da IDE.

Sobre o "**Dobramento de Código**" (o "*Code Folding*"), a melhor forma de se entender o que ocorre, é abrir um código (por exemplo o famigerado "*Blink*"), e então ligue e desligue a opção na janela de Preferências, observando em seguida o que ocorre no Editor de Texto. Essencialmente, isso habilita ou desabilita o que chamamos de opção de "*colapsar*" as estruturas no código, que nada mais é do que "*esconder*" temporariamente trechos do código no Editor, clicando-se nos sinais "-" e "+" que aparecem no início de cada estrutura de código (uma estrutura geralmente começa com um "{" e termina com um "}"). É uma configuração muito pessoal (algumas pessoas preferem ligar a opção, outras preferem desligar).

A configuração para o uso de um **Editor "externo"**, pode ser vista na próxima figura, que também é "auto-explicativa":



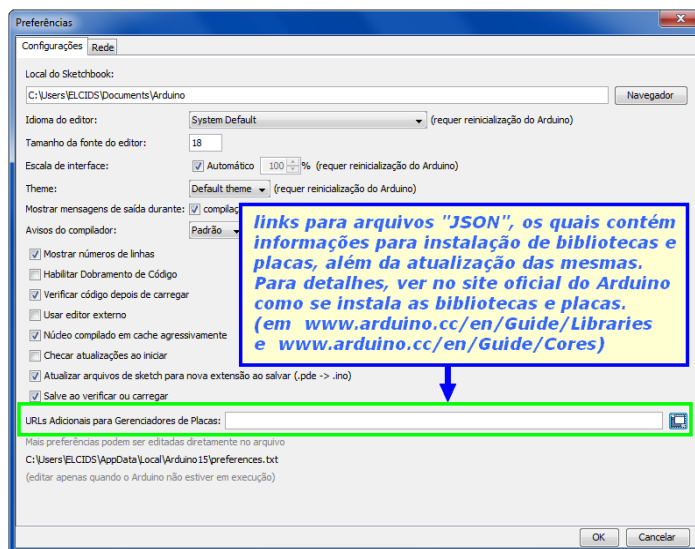
A próxima configuração "geral", está relacionada às atualizações da própria IDE e Bibliotecas instaladas. Observe que a opção não faz a atualização automaticamente, mas sim verifica automaticamente se existem atualizações, e quando houver será perguntado ao usuário se ele quer fazer as atualizações. Há também duas configurações relacionadas a salvar os arquivos. A figura a seguir mostra estas configurações, e também é "auto-explicativa".



Uma configuração permite que se inclua links para arquivos "**JSON**" (*Java Script Object Notation*). Estes arquivos podem até estar no seu próprio computador, mas geralmente eles estão hospedados em algum site na Internet. Então é praticamente certo que os links serão para sites na Internet. E o que fazem estes arquivos "**JSON**"? **No Arduino**, estes arquivos contêm informações relacionadas a processos de instalação de Placas. Exemplo: para instalar as diversas placas do **ESP32**, inclui-se na lista de links, o link para o arquivo "**JSON**" que foi criado pela *Espressif* (fabricante original do ESP32) e hospedado em um site da própria *Espressif*. Neste caso o link a ser incluído na lista de links, é esse: https://dl.espressif.com/dl/package_esp32_index.json

Note que não tem muito sentido ir ao site e abrir o arquivo "**JSON**" por conta própria, a menos que se queira fazer o processo "no braço", mas isto daria um certo trabalho. Então o que fazemos é simplesmente incluir o link na lista, e a própria IDE do Arduino se encarregará de fazer todo o trabalho automaticamente. Mas para isto ela precisa de um "empurrãozinho", que é ir no *Menu "Ferramentas"*, na opção da placa atualmente selecionada, e clicar na opção "**Gerenciador de Placas**", e então a IDE irá verificar as atualizações para todas as placas instaladas e também para as placas descritas nos arquivos "**JSON**". Depois disso, basta olhar a lista de placas que a IDE preparar e clicar em "*Instalar*" ou "*atualizar*", conforme a opção de cada placa (claro, conforme se deseja também).

A figura a seguir mostra onde se inclui o link para um arquivo "**JSON**".



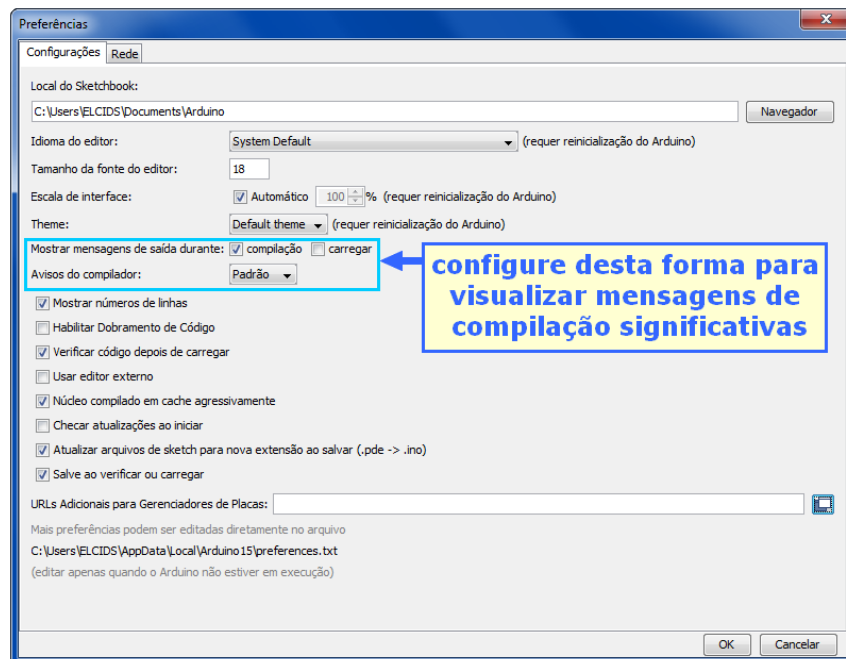
Observe que na janela só aparece uma linha para inserir o link. Porém ao final da linha há um botão (ou ícone), que ao ser clicado, abre uma janela onde se pode ver todos os links já inseridos, e a partir dessa janela pode-se também inserir ou deletar links.

Costuma-se dizer que esta configuração também se aplica às atualizações das bibliotecas das placas. Isto porque normalmente os arquivos "**JSON**" além de possuírem "instruções" para instalar placas, também podem ter instruções para atualizar as bibliotecas para estas placas.

Configurações para o Processo de Compilação:

São poucas as configurações para o **Processo de Compilação**.

A primeira destas configurações, se refere à forma de exibição das mensagens do Processo (ver itens **5.1** a **5.4** na descrição da **área de mensagens na IDE**, no início deste documento). É aconselhável deixar a configuração como mostrado na figura a seguir:

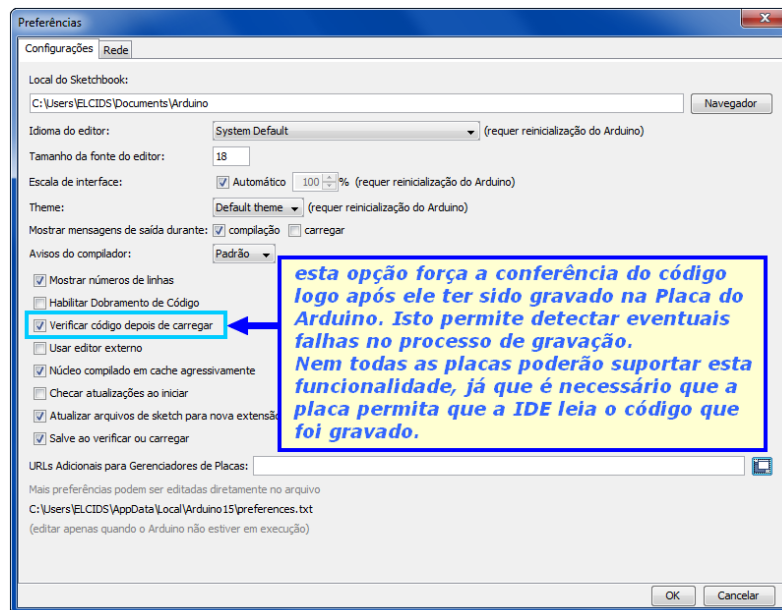


Há várias possibilidades de configuração para exibição das mensagens, mas a forma que está mostrada na figura anterior, é a mais adequada para 99,99% dos casos.

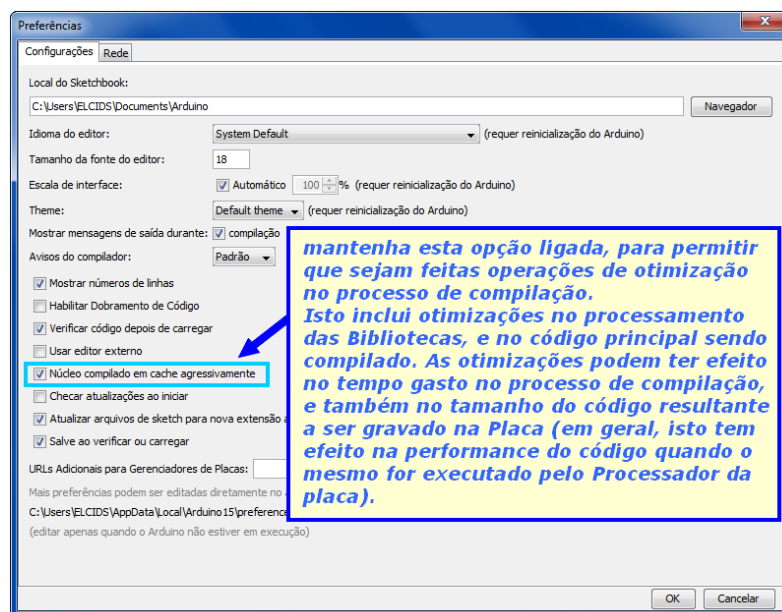
Não vamos falar aqui sobre as diversas possibilidades, pois no contexto deste documento elas acrescentam muito pouco.

Configurando como mostrado na figura anterior, serão exibidas as mensagens que são realmente significativas para se resolver problemas do **Processo de Compilação** (ver itens **5.1** a **5.4** na descrição da **área de mensagens na IDE**, no início deste documento).

A próxima configuração se refere à ligar ou desligar a opção de conferir se o código foi gravado corretamente na Placa atualmente conectada. Fazer esta conferência, pode não ser possível em todas as placas. No Arduino **Leonardo** por exemplo, a conferência é possível mas é comum que falhe, devido à forma característica que o Arduino Leonardo se comporta. A configuração é mostrada na figura a seguir, que também é "auto-explicativa":



Nas configurações, há uma opção para se habilitar um Processo de Compilação “em cache mais agressivo”, conforme mostrado na figura a seguir:



Esta opção está relacionada à forma como o Compilador tenta otimizar a compilação evitando compilar Bibliotecas que já tenham sido compiladas no mesmo processo. Por exemplo, vamos supor que a Biblioteca “WiFi” esteja incluída no código. Então quando o Compilador encontrar o “include” para esta biblioteca, ele irá compilar a mesma. Porém é provável que em outras bibliotecas sendo usadas para acessar a rede WiFi, também existam “includes” para a mesma biblioteca.

Se o Compilador for esperto o suficiente, ele só irá compilar uma única vez. Para que isto seja possível, é necessário que a primeira vez que a biblioteca WiFi é compilada, o resultado seja “guardado” em algum lugar, permitindo que este resultado seja acessado sempre que o Compilador encontrar um novo “include” para a biblioteca. Esse “lugar” é o “cache” de compilação. Este “cache” pode ter dentro dele as diversas bibliotecas já compiladas. Então isto faz com que o Compilador não perca tempo recompilando uma biblioteca, mesmo que existam várias referências à ela (os “includes”).

Mas porque existe esta opção? Ela não deveria estar sempre ligada?

Esta realmente não é uma pergunta muito fácil de se responder. A opção de se poder ligar ou desligar esta configuração, está mais relacionada com o processo de desenvolvimento da própria IDE do Arduino. Os profissionais que trabalham nesse desenvolvimento, podem optar por caminhos que nem sempre são óbvios nem facilmente entendíveis por outras pessoas que não estão trabalhando naquilo. Então isto tem muito mais a ver com a evolução da confiabilidade do Processo de Compilação.

Em geral, deixa-se a opção ligada (como mostrado na figura anterior). Mas caso se desconfie que alguma coisa muito estranha ocorre no Processo de Compilação (por exemplo, se ele costuma dar resultados diferentes, ou se hora ele funciona outra hora não funciona), pode-se tentar remediar isso desligando-se a opção. Mas claro isto é uma *situação extremamente incomum*, e muito, muito dificilmente, será necessário tentar algo assim.

Configuração para o “TAB” no Editor da IDE:

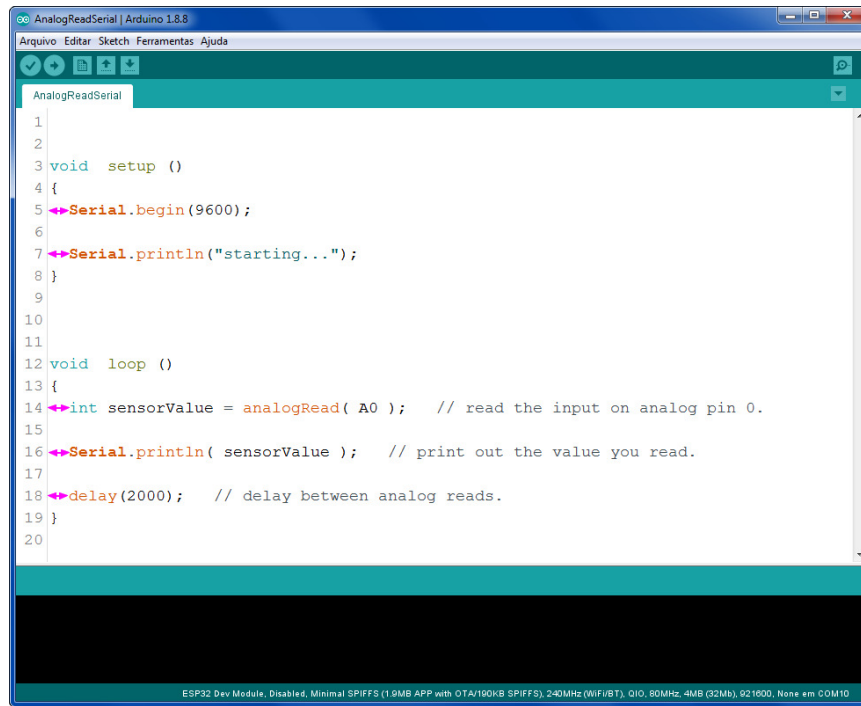
Apesar desta configuração estar totalmente relacionada ao **Editor de Texto** da **IDE** do **Arduino**, deixei para tratar da mesma de forma separada. Isto porque para fazer esta configuração, é necessário acessar o “**arquivo de preferências**” da **IDE**. O ideal seria se a configuração estivesse disponível na janela de Preferências, pois é um *setting* extremamente simples. Mas infelizmente não está. Ocorre que o “**arquivo de preferências**” da IDE, possui uma infinidade de opções de configurações de todo tipo, e como é um arquivo de texto simples (daqueles que se edita com o Notepad do Windows), é preciso ter cuidado para não se alterar por acidente algo importante na IDE.

Mas como este *setting* apesar de muito simples, pode também ser muito significativo, vale a pena acessar o “arquivo de preferências” para fazer a configuração mais adequada.

Estamos falando do “**tamanho do TAB**” no texto do Editor da IDE. E o que é este “**TAB**”? É o **caracter ASCII “TAB”** (código ASCII de número **9**). Mas na verdade a coisa toda é muito mais que isso.

Para entender, é melhor ver qual a diferença quando o tamanho do "TAB" está configurado logo após a instalação da IDE (neste caso o tamanho é de **2 caracteres**), e como fica após alterarmos o tamanho para o padrão mais utilizado (que é o tamanho de **8 caracteres**).

Na figura a seguir, pode-se ver um código que é praticamente uma cópia do exemplo "**AnalogReadSerial**" do Arduino:



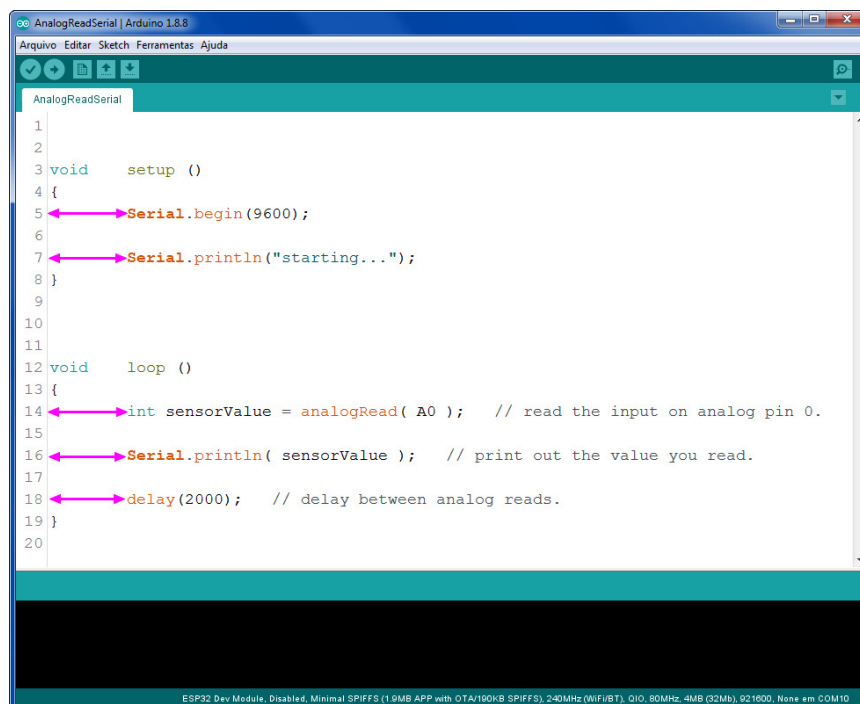
```
1
2
3 void setup ()
4 {
5   Serial.begin(9600);
6
7   Serial.println('starting...');
8 }
9
10
11
12 void loop ()
13 {
14   int sensorValue = analogRead( A0 ); // read the input on analog pin 0.
15
16   Serial.println( sensorValue ); // print out the value you read.
17
18   delay(2000); // delay between analog reads.
19 }
20
```

Observe que há algumas setas duplas na cor rosa ↔ no início de algumas linhas. Cada uma dessas setas representa o espaçamento ocupado por 2 caracteres, pois este é o "tamanho do TAB" no Editor desta IDE. Mas o que é o "tamanho do TAB" ? O "TAB" é um caractere ASCII "**não-imprimível**" (ver no início deste documento quando é descrito sobre o **Monitor Serial do Arduino**), cujo código ASCII é "**9**" (ou **0x09** em Hexadecimal). Embora o "TAB" seja "não-imprimível" (como são todos os caracteres que tem código ASCII menor que 32), ele afeta como é apresentado o texto imediatamente após o mesmo. Neste caso, os Editores de Texto entendem que quando se encontra um caractere "TAB", eles devem avançar até "**N**" **espaços em branco** antes de exibir o próximo caractere. Como nesta IDE o "tamanho do TAB" está configurado para 2 caracteres (ou seja N=2), em todo lugar que há um caractere "TAB" o Editor irá exibir até 2 espaços em branco, e depois disso exibe os demais caracteres que seguem o "TAB".


Logo, o TAB é usado justamente para fazer uma "tabulação" no texto, fazendo-o ficar mais "alinhado", como se estivesse em uma tabela com colunas de largura pré-definida (em programação, isto é tecnicamente chamado de "**denteamento**", em alusão à palavra "**dente**"). E como se usa o "TAB"?

Simples: primeiro é preciso que o Editor esteja configurado para inserir o caractere "TAB" toda vez que se digitar a tecla "TAB" no teclado (aquela tecla que fica logo acima da tecla "Caps Lock"). Do contrário, ele irá inserir até "N" caracteres de espaço, como se tivesse sido digitado a barra de espaço até "N" vezes. Neste ponto alguém poderia perguntar: mas não é a mesma coisa? Não, não é, pois o código ASCII "TAB" ocupa apenas um caractere no texto, e não "N" caracteres. E isso não é uma questão apenas de se economizar espaço no arquivo texto. Quando se usa o "TAB", as posições de tabulação são fixas na linha, ou seja, cada "TAB" encontrado, o texto avança para a próxima posição de tabulação, fazendo com que a exibição do texto fique mais homogênea e organizada, e muito mais fácil de ler (por isso o avanço é de até "N" caracteres, e poderá ser menos, caso sejam necessários menos caracteres para se chegar à próxima posição de tabulação). No entanto, o uso do "TAB" é ainda mais significativo quando se trabalha com uma **Linguagem de Programação Estruturada**, que é o caso do **C/C++** no Arduino. Para facilitar o reconhecimento das estruturas no código, procura-se colocar estas estruturas com espaçamentos que as torne mais facilmente identificáveis, e o "TAB" é uma mão na roda para isso.

Vejamos o que ocorre quando o tamanho do "TAB" é alterado para 8 caracteres, no mesmo código anterior:



```
1
2
3 void setup ()
4 {
5     Serial.begin(9600);
6
7     Serial.println("starting...");
8 }
9
10
11
12 void loop ()
13 {
14     int sensorValue = analogRead( A0 ); // read the input on analog pin 0.
15
16     Serial.println( sensorValue ); // print out the value you read.
17
18     delay(2000); // delay between analog reads.
19 }
20
```

Observar que as setas  agora estão indicando que o espaçamento mudou para 8 caracteres. Neste código muito simples, isso nem faz diferença. Mas imagine um código com diversas estruturas, funções, vários controles "if", "while", "for", etc, sendo que alguns controles podem ainda estar "dentro" de outros (tecnicamente isso é chamado de "aninhamento"). Numa situação como esta, o "TAB" faz uma enorme diferença, pois facilita tremendamente para o programador organizar seu código, em tabulações pré-definidas.

Mas aqui há dois problemas relacionados ao "TAB" na IDE do Arduino.

O primeiro problema: não há como configurar a IDE com a opção de inserir o caractere ASCII "TAB", ou seja, quando se digita a tecla "TAB", o Editor sempre irá inserir até "N" caracteres de espaço. Pode ser que isto mude em alguma revisão da IDE, mas atualmente é assim.

O segundo problema, é que normalmente o "tamanho do TAB" na IDE é configurado para apenas **2 caracteres**, o que é normalmente muito ruim para a identificação das estruturas da **Linguagem C/C++** no Editor de texto. O tamanho mais universalmente utilizado (e portanto praticamente padrão), é usar um "TAB" de **8 caracteres**, que torna fácil trabalhar com o código e também facilmente identificáveis as estruturas da Linguagem C/C++. Mas felizmente este problema é facilmente resolvido, pois é possível alterar o "tamanho do TAB" na IDE do Arduino.

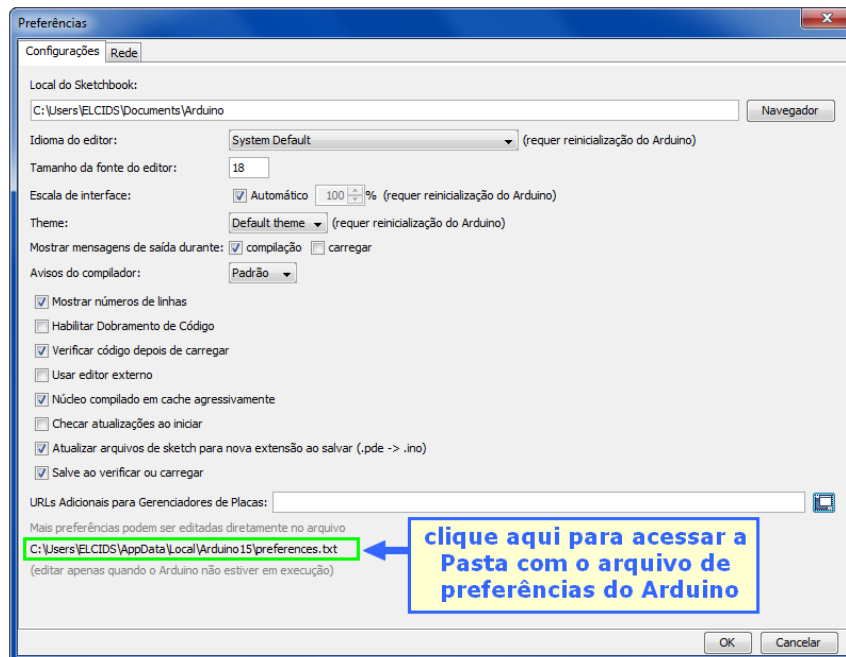
De qualquer forma, alguém poderia perguntar: mas para que alterar o "tamanho do TAB", se não há como inserir o caractere ASCII "TAB" através do Editor da IDE ??? De fato, se mudarmos o "tamanho do TAB" para 8, ao digitarmos a tecla "TAB", o Editor irá inserir até 8 caracteres de espaço. Mas quando se está usando um código editado em um outro Editor, então faz toda diferença configurar o "tamanho do TAB" para o padrão mais utilizado (no caso 8 caracteres). Isto porque embora o Editor da IDE não insira caracteres ASCII "TAB", ele é capaz de interpretá-los se eles já estiverem no arquivo do código, e assim ele avança para a próxima posição de tabulação sempre que há um caractere ASCII "TAB" no texto.

Isto facilita muito a visualização das estruturas do C/C++ no Editor, e permite que se veja estas estruturas exatamente como estava vendo a pessoa que criou o código em outro Editor. Então isso ajuda muito, quando se obtém códigos de exemplos e referências em sites diversos na Internet. Mas é claro, isso se o sujeito que trabalhou no código utilizou um Editor capaz de inserir os códigos ASCII "TAB".

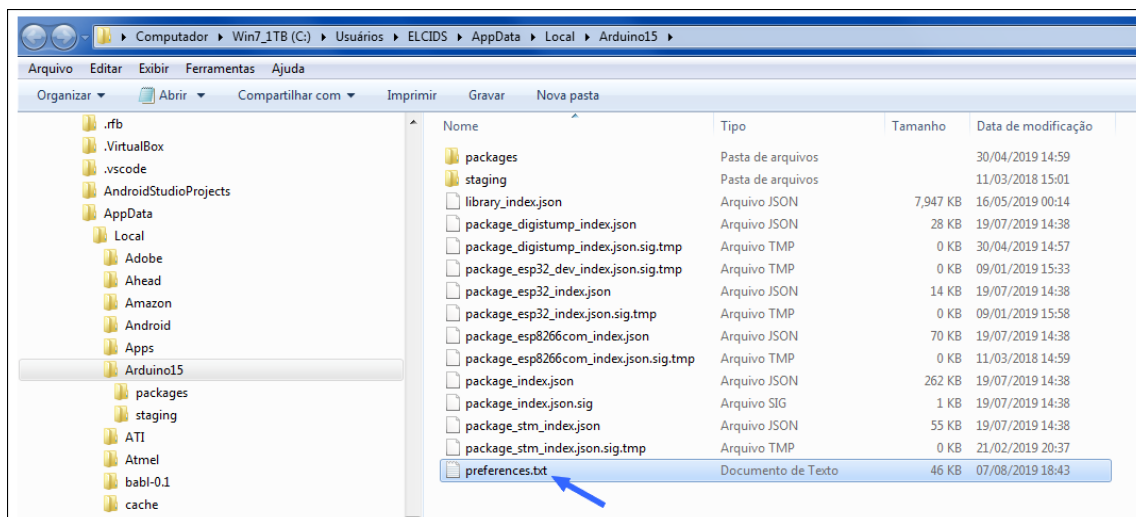
Praticamente todos os ótimos Editores gratuitos existentes (e são muitos), permitem que se configure o "TAB" da forma que se desejar. Mas nem sempre a configuração default quando se instala um destes Editores, é a que insere os caracteres "TAB", e às vezes o tamanho do TAB também não está configurado para 8 caracteres (neste caso, isto afeta a visualização apenas no próprio Editor). Então às vezes é preciso também abrir as configurações no Editor e ajustá-las da melhor forma possível.

Então, vejamos como alterar o "tamanho do TAB" para a IDE do Arduino. Como dito anteriormente, isto *não é feito* na janela de Preferências, mas em um **arquivo "txt"** que fica em uma pasta de trabalho do Arduino.

Felizmente, para encontrar este arquivo “txt”, pode-se usar um atalho que está na janela de **Preferências**, conforme mostrado na figura a seguir:

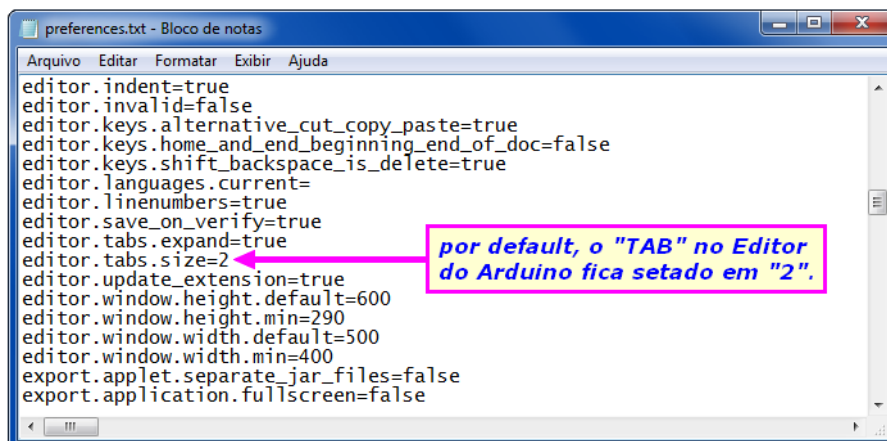


Clicando-se no atalho mostrado na figura anterior, irá abrir-se o **Explorer** do **Windows**, mostrando a pasta onde fica o **arquivo “txt” das Preferências do Arduino**. Isto é mostrado na figura a seguir (se no Windows não estiver configurado para exibir a *extensão dos arquivos*, então o nome do arquivo aparecerá apenas como “preferences”, ao invés de “preferences.txt”):

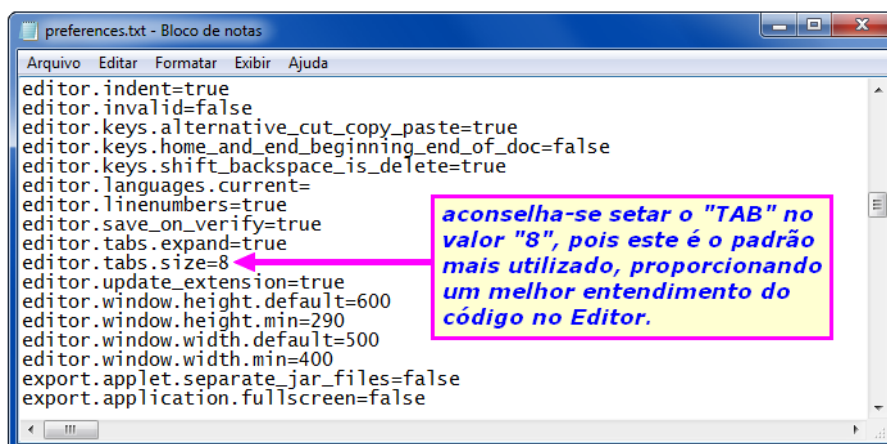


Atenção: esta é uma daquelas “pastas ocultas” no Windows. Então não faça nenhuma outra alteração ali, a não ser a que será descrita aqui. O mesmo vale para qualquer outra pasta/arquivo nas imediações (do contrário poderão ocorrer grandes problemas no Arduino e no Windows).

Para fazer a alteração, é preciso antes fechar a IDE do Arduino (do contrário a alteração não terá efeito). Então após a IDE ter sido fechada, abra o arquivo "**preferences.txt**" usando qualquer Editor de texto que desejar. Na figura a seguir ele foi aberto usando o "Notepad" do Windows (que é inclusive o padrão para abrir arquivos ".txt"):



Vá até a linha mostrada na figura anterior. Esta linha possui o texto "**editor.tabs.size=2**", o que indica um "**tamanho do TAB**" de **2 caracteres**. Altere o valor de **2** para **8**, para que fique configurado com o tamanho mais universalmente usado para o "TAB". Isto pode ser visto na figura a seguir:



Após feita a alteração e conferida, salve o arquivo e feche o Editor usado (no caso da figura, o Notepad do Windows). Feche também a janela do *Explorer do Windows*, para não correr o risco de alterar algo acidentalmente.

Atenção: não altere nenhum outro setting no arquivo de Preferências da IDE, se não conhecer o que significa, ou se não souber como deve ser alterado (do contrário poderá ter comportamentos anormais na IDE do Arduino). Se tiver algum receio, antes de qualquer alteração, faça uma cópia do arquivo original (a cópia pode ser deixada na mesma pasta do arquivo original).

Após esta configuração no arquivo de Preferências, o "tamanho do TAB" já aparecerá com 8 caracteres no Editor da IDE do Arduino. Mas claro, para ver o efeito disso é preciso abrir algum código que tenha sido escrito em um outro Editor externo à IDE, e que seja capaz de inserir os caracteres ASCII "TAB". De forma surpreendente, até o próprio *Notepad* do *Windows* insere os caracteres ASCII "TAB" quando se digita a tecla "TAB" no teclado, então pode-se usá-lo para editar um código qualquer e verificar o efeito do "TAB" quando o código for aberto na IDE do Arduino.

Obviamente que ao se digitar a tecla "TAB", agora a tabulação no Editor da IDE avançará até 8 caracteres na tela. Porém como o Editor da IDE do Arduino não insere o caractere "TAB", isto será equivalente a se digitar até 8 vezes a tecla de espaço (mesmo assim a visualização das estruturas da Linguagem C/C++ terá melhorado muito).

Quanto mais estruturas da **Linguagem C/C++** existirem no código, e se existirem também estruturas "*aninhadas*" (quando uma estrutura está "dentro" de outra), mais interessante será a visualização com a configuração feita para o "TAB".

AFA

(Arduino for all)