BRASILINO: BIBLIOTECA PARA ARDUINO QUE PERMITE O DESENVOLVIMENTO DE PROJETOS EM PORTUGUÊS DO BRASIL

Otacílio Saraiva Maia Neto - (4° Ano do Ensino Técnico Integrado)[1],

Thiago Augusto dos Santos Martins - (4° Ano do Ensino Técnico Integrado)[1],

Rômulo César Carvalho de Araújo - (Professor Tutor/Orientador)[1].

[1]INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DE PERNAMBUCO (IFPE) - *Campus* Recife.
Av. Prof. Luís Freire, 500 – Cidade Universitária.
CEP 50740-540–Recife – PE.

Resumo: Este artigo apresenta a biblioteca Brasilino, desenvolvida para a plataforma Arduino, permitindo a programação para este, utilizando funções traduzidas para o português do Brasil, além de uma série de novas funções implementadas que possibilitam o uso de sensores e atuadores voltados à robótica, internet das coisas (IoT) e automação. A finalidade da biblioteca é permitir a inserção de brasileiros que não possuem proficiência em língua inglesa, ou que preferem utilizar como língua base, o português do Brasil, em seus códigos, sendo uma ferramenta de uso didático e educacional.

Palavras Chaves: Arduino, Biblioteca, Brasilino, Educação, Português, Programação.

Abstract: This article presents the Brasilino library, developed for the Arduino platform, allowing the programming for this using functions translated into Portuguese of Brazil, as well as a number of new implemented functions that enable the use of sensors and actuators focused on robotics, internet of things(IoT) and automation. The purpose of the library is to allow the inclusion of Brazilians who do not have English language proficiency or who prefer to use as the base language, Portuguese of Brazil, in their code, and being a teaching and educational use tool.

Keywords: Arduino, Library, Brasilino, Education, Portuguese, Programing.

1 INTRODUÇÃO

As linguagens de programação inicialmente surgiram através dos cartões perfurados, inspirados na máquina de tear de Jacquard. Através dos padrões de furos feitos no cartão [HUSKEY, 1980]. Dessa forma era possível passar instruções ao computador. Com o passar do tempo, o método de transmitir estas instruções foi evoluindo de instruções em cartões perfurados para instruções escritas.

Surgiram assim as linguagens de baixo nível, utilizando palavras como MOV ou ADD, chamadas de mneumônicos. Porém, mesmo sendo funções escritas, ainda eram instruções complexas, devido aos seus parâmetros.Com a evolução da computação criou-se uma tendência de aproximar a linguagem de programação da linguagem humana, até que comandos como IF, WHILE e outros ainda mais abstraídos como "Arrays.sort()" foram implementados.

Contudo, quando falamos que a linguagem de máquina vem se aproximando da linguagem humana, estamos falando na verdade da língua inglesa. E de acordo com estudo realizado pela Education First em 2015 [First, 2015], que avaliou o nível de proficiência em inglês em 70 países incluindo o Brasil, concluiu-se que o Brasil apresenta um nível de proficiência baixo em língua inglesa, sendo o 41° na lista, ficando atrás de outros países da américa latina como Peru e Chile.

Devido a esse baixo nível de proficiência em língua inglesa apresentado no Brasil, muitos dos brasileiros acabam não conseguindo ingressar e se manter no estudo de programação, e por sua vez, nos ramos da robótica e automação. Visando quebrar o paradigma de que fazer robótica é difícil, e programar também, o Brasilino se propõe a romper, inicialmente, a barreira do inglês na programação, trazendo as instruções para o português do Brasil, além de trazer uma série de funções otimizadas e simplificadas. Com isso, os estudantes brasileiros podem ter um ponto de partida sem muitas dificuldades neste ramo da tecnologia.

2 O TRABALHO PROPOSTO

O objetivo desse projeto é desenvolver uma biblioteca para a plataforma Arduino, que possa integrar os brasileiros a esta plataforma, através da aproximação da linguagem de programação à lingua nativa do Brasil.

O projeto conta com uma extensa biblioteca de exemplos, que norteam as aplicações de sensores e atuadores, que antes necessitavam de diferentes bibliotecas, em uma única biblioteca. O Brasilino também se propõe a documentar toda a biblioteca e o seu desenvolvimento, sendo registrada com um projeto Open Souce na licença GNU GENERAL PUBLIC LICENSE Version 3 [License, 2007], permitindo assim que os usuários possam consultar as funções implementadas, bem como conhecer a fundo o funcionamento da biblioteca.

Baseando-se no desenvolvimento colaborativo de códigos aberto, a Brasilino se propõe a permitir que seus usuários possam sugerir modificações através da plataforma de compartilhamento de códigos Github, bem como se torna escalonável para que bibliotecas com outros idiomas nativos possam ser implementadas, como a "Italino" por exemplo, em Italiano.

3 MATERIAIS E MÉTODOS

3.1 Brasilino.h

3.1.1 Introdução

Diferente de outras bibliotecas, grande parte da funcionalidade da Brasilino está implementada no arquivo de Header. Através de "*defines*" boa parte das instruções foram traduzidas. Logo nas primeiras linhas de código, como mostrado na figura 1 é possível ter uma boa noção do funcionamento da biblioteca.

```
#define definir define
#define usar define
```

Figura 1 – Início do Header.

Através da diretriva de compilação *define* é possível fazer com que, sempre que o compilador encontre a palavra que foi definida troque a mesma pela palavra escolhida. Permitindo que uma palavra como definir, que não é reconhecida pela linguagem Arduino, seja trocada por uma palavra que é reconhecida, como *define* por exemplo.

3.1.2 Multi traduções de instruções

Algumas das intruções ganharam mais de uma tradução, devido às suas multiplas funcionalidades, permitindo que o usuário se sinta mais livre para escolher qual a melhor palavra utilizar naquele contexto. Como mostrado na figura 2.

```
se(estadoBotao == PRESSIONADO){
    escreverDigital(led, ALTO); // Liga o led.
}
```

Figura 2 – Exemplo de multi traduções.

E como pode ser visto na figura 3, "PRESSIONADO" e "ALTO" são substituídos pelo compilador por "HIGH", porém quando estamos falando de um sensor de toque, por exemplo, faz mais sentido comparar com "PRESSIONADO" ou "LIBERADO", e quando estamos falando de saídas, como um diodo emissor de luz (LED), por exemplo, faz mais sentido escrever "ALTO" e "BAIXO" para representar os estados.

```
#define ALTO HIGH
#define BAIXO LOW
#define PRESSIONADO HIGH
#define LIBERADO LOW
```

Figura 3 – Implementação de multi traduções.

Partindo do princípio de criar novos significados que melhor se adaptem à compreensão inicial do usuário, a instrução "void" além de ter sido traduzida como "nulo" também foi traduzida como "funcao", permitindo para que quando o usuário quiser fazer menção a um tipo nulo utilize a instrução "nulo", porém quando for chamar as funções de setup e loop, que foram traduzidas como configurar e repetir, respectivamente, fique mais fácil a compreensão com "funcao configurar" e "função repetir" do que em "nulo configurar" e "nulo repetir", apesar das duas formas serem possíveis.

3.1.3 Funções simplificadas

Também foram implementadas funções simplificadas em relação as já existentes na linguagem Arduino, como a função "ligar" e "desligar", onde o parâmetro necessário é simplesmente o pino em que se quer ligar ou desligar como pode ser observado na figura 4.

```
#define ligar(pino) digitalWrite(pino, HIGH)
#define desligar(pino) digitalWrite(pino, LOW)
```

Figura 4 – Implementação de funções simplificadas.

3.1.4 Parâmetros simplificados

Algumas funções também tiveram seus parâmetros modificados, como por exemplo a função "esperar", que corresponde a função "delay" do Arduino, porém em vez de esperar um tempo em milissegundo aceita o tempo em segundo, permitindo também frações de segundo, como por exemplo 0.1 para representar um décimo de segundo, ou 100 milissegundos. E a função de espera com o parâmetro em milissegundos passou a se chamar "esperarMili", como podem ser observadas na figura 5.

```
#define esperar(tempo) delay(tempo*1000)
#define esperarMili(tempo) delay(tempo)
```

Figura 5 – Implementação de parâmetros simplificados.

3.1.5 Custo de memória

Um dos pontos positivos da biblioteca é que devido ao fato das funções da Brasilino terem sido implementadas através de diretivas de compilador, os programas em Brasilino possuem a mesma eficiência em gasto de memória que os programas em Arduino. Como pode ser visto na mensagem de compilação apresentada na figura 6 os exemplos "Blink" e seu equivalente em Brasilino "Piscar" apresentam ambos o tamanho de 1,030 bytes, o que representa aproximadamente 3% do espaço de memória de uma placa Arduino Uno.

```
O sketch usa 1.030 bytes (3%) de espaço
```

Figura 6 – Mensagem de compilação do "Piscar" e "Blink".

3.2 Brasilino.cpp

No arquivo Brasilino.cpp, programado em linguagem C++ [Stroustrup, 2000], é onde estão implementadas as funções que não são nativas à linguagem Arduino, mas que são pertinentes para o desenvolvimento de projetos. Por exemplo funções para o uso de sensores específicos, como a função de obtenção da temperatura em Graus Celsius através de um Termistor de 10kΩ, que pode ser visto na figura 6.

```
dobro temperatura(inteiro valorAnalogico) {
 dobro Temp;
 Temp = log(10000.0*((1024.0/valorAnalogico-1)));
 Temp = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * Temp * Temp ))* Temp );
 Temp = Temp - 273.15;
 retorne Temp;
}
```

Figura 7 – Função de obtenção da temperatura através de um Termistor de 10kΩ implementada no arquivo Brasilino.cpp.

3.3 Tabela das instruções implementadas

Segue tabela com todas as instruções implementadas até a data 24/07/2016.

Tabela 1 – Instruções implementadas.

Instrução	Descrição	Exemplo
inteiro	Declara uma variável do tipo inteiro.	inteiro idade = 13;
decimal	Declara uma variável do tipo decimal;	decimal saldo = 3.45;
dobro	Declara uma variável do tipo dobro. Esta tem o dobro de bits de armazenamento quanto a variável decimal.	dobro lucro = 3.87909;
caractere	Declara uma variável do tipo caractere.	caractere entrada = 'c';
logico	Declara uma variável do tipo logico, podendo ser "verdadeiro" ou "falso".	logico dinheiro = verdadeiro;
constante	Declara uma variável constante.	constante tamanho = 4;
nulo	Declara uma variável do tipo nulo.	nulo sapatos;
verdadeiro	Parâmetro lógico utilizado para operações digitais.	se(a == verdadeiro){ //Ação caso "a" seja verdadeiro. }
falso	Parâmetro lógico utilizado para operações digitais.	se(a == falso){ //Ação caso "a" seja //falso. }
definir	Associa um valor constante a um nome.	#definir PINO 13
usar	Libera uma macro para uma nova função.	#usar BAUD
configurar	Função principal do programa, só será executada uma única vez.	funcao configurar(){ //Bloco de uma única //execução.}
Repetir	Função do programa que ficará repetindo enquanto o arduino estiver ligado.	funcao repetir(){ //Bloco que ficará //repetindo. Esta função //vem logo após a //função configurar }
definirPino	Define se o pino trata-se de uma entrada ou saída.	definirPino(PINO, TIPO);
Saida	Define o pino como uma saída.	saida(13); //Pino digital 13 //definido como saída.
Entrada	Define o pino como uma entrada.	entrada(12); //Pino digital 13 //definido como saída.
Se	Analisa se a condição dentro do parâmetro é verdadeira e executa uma ação.	se(a == b){ //Ação caso "a" seja //igual a "b" }
Senao	Executa uma ação se o parâmetro da condição "se" for falso.	se(a == b){ //Ação caso "a" seja //igual a "b" }senao{ //Ação caso "a" seja //diferente de "b" }

comparar....caso	Compara o parâmetro da função com os casos definidos. No exemplo “x” é comparado com os valores 1 e 2. É utilizado a instrução “sair” para que não se realize os próximos testes se algum já for o verdadeiro.	comparar(x){
		caso 1:
		//Ação caso “x” for //igual a 1
		sair;
		caso 2:
		//Ação caso “x” for //igual a 2
		sair;
		padrao:
		//Executa se não for //nenhum dos casos
		sair; }

Enquanto	Esta função executa continuamente enquanto o teste do parâmetro for verdadeiro.	enquanto(1 == 2){
		//Ações a serem //executadas se o //parâmetro for //verdadeiro
		}

para	Executa um bloco de instruções enquanto uma condição for satisfeita. É utilizado um contador para incrementar, ou decrementar, e terminar o *loop*.	Para(x = 0; x < 2; x = x+1){
		//Executa este bloco //enquanto “x” for menor //que 2
		}

contarAte	Aplicação da função “para” onde se escolhe o número de interações.	contarAte(5){
		//Executa este bloco de //instruções 5 vezes.
		}

escreverSerial	Mostra no Monitor Serial o valor colocado como parâmetro.	escreverSerial(“BR”);
escreverSerialn	Mostra no Monitor Serial o valor colocado como parâmetro e depois pula uma linha.	escreverSerialn(“BR”);
escrever	Escreve o valor colocado como parâmetro, no Monitor Serial.	Serial.escrever(10);

lerSerial	Ler o valor recebido pelo Monitor Serial.	lerSerial();
esperar	Espera durante o tempo determinado, no seu parâmetro, para executar a próxima instrução.	esperar(1); //O parâmetro deve ser //escrito na unidade de //segundos, podendo //ser um valor decimal.
escreverDigital	Função que escreve o valor “ALTO” ou “BAIXO” nos pinos digitais.	escreverDigital(PINO, VALOR);
ALTO	Parâmetro que indica o estado alto no pino.	escreverDigital(10, ALTO);
BAIXO	Parâmetro que indica estado baixo no pino.	escreverDigital(10, BAIXO);
ligar	Função que escreve o valor ALTO no pino.	Ligar(PINO);
Desligar	Função que escreve o valor BAIXO no pino.	Desligar(PINO);
definirPino	Função que define o valor do pino como “ENTRADA” ou “SAIDA”.	definirPino(PINO, TIPO);
ENTRADA	Parâmetro que indica tratar-se de uma entrada.	definirPino(12, ENTRADA);
SAIDA	Parâmetro que indica tratar-se de uma saída.	definirPino(11, SAIDA);
PRESSIONADO	Parâmetro para botão pressionado.	se(estadoBotao == PRESSIONADO)
LIBERADO	Parâmetro para botão pressionado.	se(estadoBotao == LIBERADO)

3.4 Reconhecimento de palavras chave

Utilizando a IDE da Arduino é possível perceber que quando palavras chave, como a palavra "void" por exemplo, são inseridas, as mesmas ganham uma cor de destaque, permitindo assim que o programador identifque que acertou na sintaxe da instrução. Isto é feito através do sistema de *highlights* da IDE do arduino, que utiliza de um arquivo chamado "keywords.txt" para realizar o reconhecimento das palavras chave, caso a biblioteca contenha um arquivo "keywords.txt", as palavras chave inseridas dentro deste arquivo também serão reconhecidas pela IDE.

O sistema utiliza de 4 tipos de reconhecimento de palavras, acendendo com 4 cores diferentes:

- KEYWORD1, acende em laranja escuro e foi utilizado para as funções especiais, como por exemplo as funções comunicação Serial;
- KEYWORD2, acende em laranja claro e foi utilizado nos métodos e funções, como por exemplo a função ligar;
- KEYWORD3, acende em verde claro e foi utilizado em estruturas específicas, como laços de repetição por exemplo;
- LITERAL1, acende em azul e foi utilizado em constantes e tipos de variáveis.

As palavras chave da Brasilino foram inseridas no arquivo "keywords.txt" e permitem os mesmos *highlights* que as suas funções originais.

3.5 Exemplos disponíveis

Atualmente a biblioteca conta com 7 exemplos que podem ser acessados através da IDE da Arduino [Arduino, 2016] pelas abas Arquivo->Exemplos->Brasilino, como pode ser visto na figura 8.

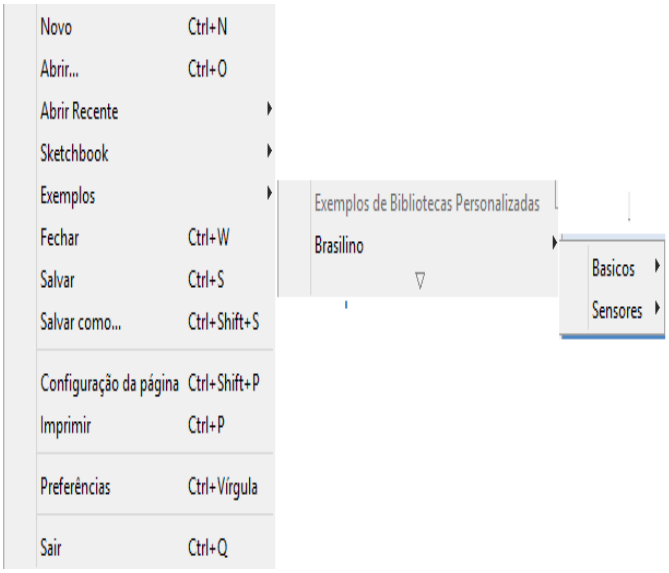

Figura 8 – Aba de exemplos da Brasilino.

Os exemplos estão divididos em duas abas, Básicos e Sensores, onde contém os exemplos gerais de uso da biblioteca, sendo Exemplos da aba Básicos (Figura 9) e Exemplos da aba Sensores (Figura 10).

3.5.1 Exemplos da aba Básicos

- analogicoSerial: Exemplo que realiza uma leitura analógica e escreve o seu valor no Serial Monitor do Arduino. Introduzindo as funções iniciarSerial, EscreverSerial e lerAnalogico;
- BaseMinimo: Exemplo com a base mínima necessária para iniciar o código em Brasilino;
- controleGradual: Exemplo que realiza o controle gradual da luninosidade de um led, variando entre 0 e 255. Introduzindo as funções escreverAnalogico, contarDe, contarAte;
- Piscar: Exemplo que faz um led piscar, introduzindo as funções saida, ligar, desligar e esperar.

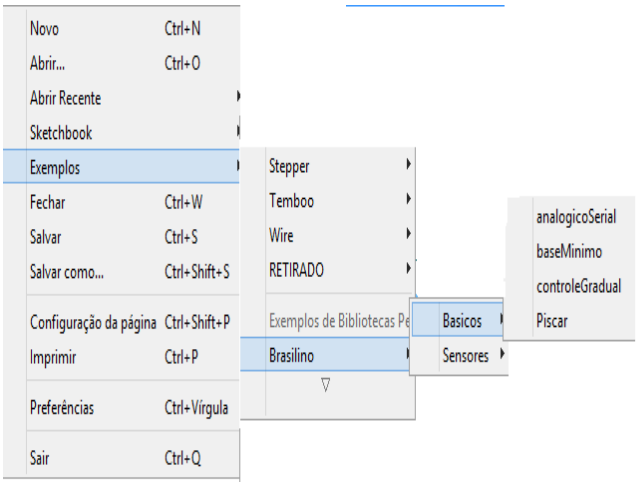

Figura 9 – Aba de exemplo da aba Básicos.

3.5.2 Exemplos da aba Sensores

- Botão: Exemplo que mostra como realizar o controle de um diodo emissor de luz em função de um botão;
- Luminosidade: Exemplo que mostra como realizar a leitura de luminosidade utilizando um Resistor Dependente de Luz (LDR) de 10kΩ e imprimir sua leitura no Monitor Serial da Arduino.
- Temperatura: Exemplo que mostra como realizar uma leitura de temperatura utilizando um Termistor de 10kΩ e imprimir sua leitura em graus Celsius no Monitor Serial da Arduino;

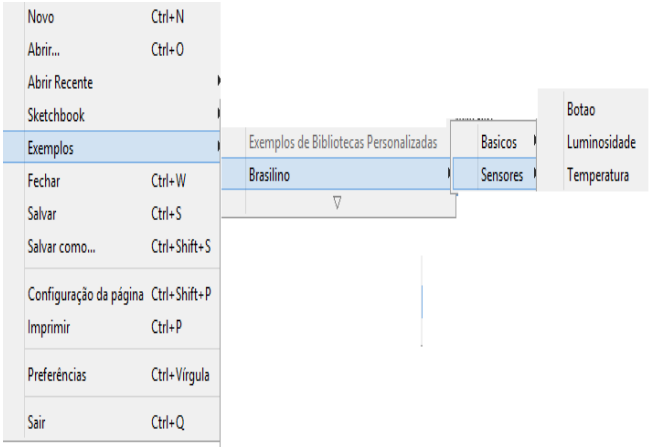

Figura 10 – Aba de exemplos Sensores.

3.6 Github

A biblioteca completa, incluindo os códigos fontes e toda a documentação se encontra atualmente disponível no Github através do link: https://github.com/OtacilioN/Brasilino.

Todo o desenvolvimento da Brasilino foi feito através da plataforma Gihub, onde é possível acompanhar o passo a passo do desenvolvimento da biblioteca através do histórico de *Commits*, bem como o histórico de versões lançadas.

Através do Github outros usuários podem acessar os códigos fonte, implementar modificações e sugerí-las através do "pull request" [Dabbish, 2012], estas sugestões serão analisadas pelos administradores do repositório e caso sejam pertinentes serão aceitas, sendo assim oficialmente mantidas dentro da biblioteca. Desta forma a biblioteca irá funcionar através do desenvolvimento colaborativo, permitindo que novas funções e funcionalidades sejam agregadas à ela, bem como permitindo que outros usuários possam se inspirar e criar suas próprias bibliotecas para outros idiomas.

4 RESULTADOS E DISCUSSÃO

4.1 O teste

A biblioteca brasilino foi apresentada à 15 alunos do ensino técnico, do IFPE *Campus* Recife, com idades entre 15 e 17 anos, que não haviam tido contato com linguagens de programação anteriormente. Primeiramente, foi mostrado o exemplo "Piscar" da Brasilino, em seguida foi mostrado o exemplo correspondente "Blink" da Arduino, então foi pedido para que os estudantes dissertassem acerca dos dois programas apresentados, através de uma análise qualitativa dos dados a biblioteca Brasilino apresentou os resultados apresentados a seguir:

Todos os estudantes conseguiram compreender o exemplo "Piscar" e seriam capazes de realizar modificações no código, bem como todos os estudantes julgaram o exemplo "Piscar" de mais fácil entendimento que o exemplo "Blink".

4.2 Pesquisa em tempo real

Atualmente existe uma pesquisa online realizada através do Google Forms, sobre a aceitação da biblioteca Brasilino, onde após responder a pesquisa é possível ter acesso aos dados obtidos sem divulgar os nomes dos voluntários. A pesquisa se encontra disponível através do link: **http://tinyurl.com/pesquisaBrasilino**.

5 CONCLUSÕES

Concluímos através das pesquisas realizadas, que a biblioteca Brasilino obteve êxito em seu propósito educacional, sendo ferramenta importante no desenvolvimento da internet das coisas (IoT) e da robótica educacional no Brasil.

Através da biblioteca é permitida a inserção de novos nichos estudantis, antes segregados pela dificuldade com a língua inglesa. Uma vez que a biblioteca Brasilino mantém a mesma sintaxe que a "Arduino Programing Language" [Arduino, 2016] e as linguagens C e C++, ao se aprender a programar utilizando com a biblioteca Brasilino o usuário pode facilmente migrar para as linguagens em sua forma nativa sem a utilização da biblioteca.

Por tratar-se de uma biblioteca o Brasilino não exclui a possibilidade de que o usuário possa digitar instruções nativas da linguagem Arduino, em inglês, mesclando-os com instruções da biblioteca, em português do Brasil, permitindo assim que a migração possa ser feita de forma gradual, diminuindo o impacto pela migração.

Uma vez que a biblioteca é direcionada ao público iniciante na linguagem de programação, a mesma pode não proporcionar uma experiência tão interessante ao público com experiência na área, pois tal público já se encontra habituados a utilizar as instruções escritas na língua inglesa.

A Brasilino pode ser utilizada como base no desenvolvimento de bibliotecas para tradução e simplificação de comandos, permitindo que outras bibliotecas, utilizando como referência outros idiomas base possam ser desenvolvidas, permitindo assim que estudantes de países com dificuldades em língua inglesa possam também ter um ponto de partida.

Vale frisar que a biblioteca continua em desenvolvimento, e que se encontra totalmente aberta à colaboração da comunidade, permitindo assim a expansão das instruções implementadas bem como de suas funcionalidades.

REFERÊNCIAS BIBLIOGRÁFICAS

Arduino (2016). Site oficial: www.arduino.cc, acessado em 24/07/2016.

Dabbish, Laura et al. Social coding in GitHub: transparency and collaboration in an open software repository. In: **Proceedings of the ACM 2012 conference on Computer Supported Cooperative Work**. ACM, 2012. p. 1277-1286.

First, Education. EF English proficiency index. 2015. Disponível em: http://media.ef.com http://media.ef.com/__/~/media/centralefcom/epi/downloads/full-reports/v5/ef-epi-2015-portuguese.pdf, acessado em 24/07/2016.

Huskey, Velma R.; Huskey, Harry D.. (october 1980). "Lady Lovelace and Charles Babbage" *Annals of The History of Computing* **2** (4): 384. Arlington, VA: American Federation of Information Processing Societies.

License, GNU General Public. version 3. **Free Software Foundation. http://www. gnu. org/copyleft/gpl. html. Last checked May**, v. 13, p. 2010, 2007.

Stroustrup, B. (2000). A Linguagem de Programação C++. Ed.Addison-Wesley.