Universidade Federal de Santa Catarina
Centro Tecnológico
Departamento de Automação e Sistemas
Engenharia de Controle e Automação
Introdução à Informática para Controle e Automação - DAS 5334

Aula 12 — Função `main()`[1]

1 Linguagem C

A linguagem Arduino que foi usada até a aula anterior foi criada baseada na Linguagem C e em alguns elementos da linguagem C++. Assim, não será muito difícil passar a fazer programas em C ao invés de Arduino. Todas as estruturas de controle de fluxo, como laços de repetição e condicionais, declaração e inicialização de variáveis e constantes, criação e chamada de funções, uso de bibliotecas criação de *arrays*, seguem a mesma sintaxe que foi vista com Arduino.

As principais diferenças estarão nas funções que são oferecidas nas biblitecas de cada uma das linguagens. Por exemplo, em Arduino os resultados eram tipicamente mostrados em um LCD e as entradas eram obtidas por meio de botões, enquanto em C, tipicamente, a saída será o monitor e as entradas o teclado e o mouse. Evidentemente, as funções para uma caso e outro são completamente diferentes.

Como ocorre com a linguagem Arduino, um programa em C precisa ser compilado e *linkado*. Porém, ao invés do arquivo gerado ser carregado no microcontrolador, ele é executado em um sistema operacional. Em Windows é gerado um arquivo com extensão `exe` e em Linux um arquivo com extensão `out`. Basta executar o arquivo para rodar o programa.

Mas a primeira grande diferença que deve ser assimilada para o uso da linguagem C é a estrutura do programa.

2 Estrutura de um programa em C

Todo programa em Arduino tem que ter pelo menos duas funções, `setup()` e `loop()`. Em C, todo programa tem que ter uma função chamada `main()`, como na listagem a seguir.

```c
#include <stdio.h>

int main() {
    printf("primeiro programa\n");
    return 0;
}
```

A função `main` define o funcionamento de um programa em C. É como se, quando um programa em C é executado, o sistema operacional chamasse a função `main`, isto é, quando a função `main` encerra, o programa encerra.

Veja que função main na listagem acima está definida exatamente como seria definida uma função em Arduino. Há tipo de dado a ser devolvido (`int`, o nome da função (`main`) e parênteses (`()`) que poderiam conter uma série de parâmetros indentificados pelo seu tipo e nome. Nesse caso a função main não tem parâmetros. O corpo da função é delimitado por chaves (`{}`) e, no exemplo, possui apenas um comando (uma função) encerrado com ponto e vírgula (`;`) que mostra na tela o texto `primeiro programa`.

Ainda que a função `main` possa não devolver valor algum, isto ser definida com `void`, a maioria dos compiladores exige que seja devolvido um valor do tipo `int` por seguirem o padrão ISO. Neste curso o valor devolvido não deve ser fator de preocupação, basta saber que o convencional é devolver o valor zero quando o programa executou corretamente.

[1] Baseado em conteúdo do livro Think Python 2nd Edition by Allen B. Downey e conteúdo da página oficial do Arduino.

3 Saída de dados

A saída de dados na tela pelo programa anterior foi feito com a função `printf()`. Essa função permite a saída formatada de dados no dispostivo padrão (tipicamente o monitor). O uso dessa função requer a biblioteca `stdio.h`, disponível por padrão em todas as instalações de compiladores de linguagem C, importada por meio da diretiva `#include`.

A forma geral da função `printf` é a seguinte:

```
prinf(controle, arg1, arg2, ...);
```

Note que apenas o primeiro argumento é obrigatório e todos os demais são opcionais.

O `controle` é o texto que se deseja exibir, com a possibilidade de inclusão de caracteres especiais, chamados especificações de conversão, que incluem no texto os valores dos argumentos, na mesma ordem em que aparecem na chamada da função. Para cada argumento após `controle` deve haver uma especificação de conversão correspondente.

Uma especificação de conversão é iniciada por um caractere `%` e encerrada por um caractere de conversão. Os principais caracteres de conversão são:

- `d` que espera (ou converte para) um argumento do tipo `int`
- `f` que espera (ou converte para) um argumento do tipo `float` ou do tipo `double`
- `c` que espera (ou converte para) um argumento do tipo `char`
- `s` que espera uma cadeia de caracteres (caracteres entre aspas (`""`)).
- `l` que espera (ou converte para) um argumento do tipo `long int`

Por exemplo, o programa a seguir mostra na tela os valores da variável global e da variável local.

```c
#include <stdio.h>

float num1 = 2.5;

int main() {
    int num2 = 3.
    printf("O valor de num2 é %d e o valor de num1 é %f", num2, num1);
}
```

Ou seja, o programa mostra na tela o texto `O valor de num2 é 3 e o valor de num1 é 2.5", num2, num1`.

4 Função `main` com parâmetros

Em `C` é possível passar argumentos para a função `main` no momento da chamada do programa. A passagem de argumentos por linha de comando é muito comum no sistema operacional linux. Por exemplo, o comando `ls` quando exectuado em um terminal linux lista os arquivos do diretório corrente. O comando `ls` é na verdade um programa em C. Esse comando pode ser chamado com uma série de argumentos que indicam, por exemplo, que diretório se quer listar (ao invés do diretório corrente) ou a forma como se deseja que a lista seja exibida. Por exemplo, a chamada a `ls -l` mostra a lista de arquivos com um arquivo por linha. Notar que `-l` é um argumento que é passado para a função `main` do programa `ls`.[2]

A listagem a seguir exemplifica como a função `main` deve ser definida para que argumentos possam ser passados na chamada de um programa e como os parâmetros devem ser usados.

```c
#include <stdio.h>

void main (int argc, char *argv[]) {
    int i;
    for (i = 0; i < argc; i++) {
        printf("%d: %s\n", i, argv[i]);
    }
}
```

[2]Em ambientes de desenvolvimento que permitem executar o programa por teclas de atalho ou por botões do tipo `run` normalmente oferecem um local onde é possível inserir os argumentos da chamada do programa.

O parâmetro `argc`, do tipo `int`, recebe como valor o número de argumentos com que o programa foi chamado, incluindo o nome do próprio programa. Assim, na chamada `ls` o valor de `argc` seria um e na chamada de `ls -l` o valor de `argc` seria dois.

O parâmetro `argv` é um apontador para um *array* de cadeias de caracteres que contêm os argumentos que foram passados na chamada do programa, ou seja, um *array* de `argc` posições. Assim, na chamada `ls`, `argv` seria um *array* de apenas uma posição com um ponteiro apontando para a cadeia de caracteres `"ls"` e na chamada de `ls -l`, `argv` seria um *array* de duas posições, cada uma com um ponteiro apontando, respectivamente para as cadeias de caracteres `"ls"` e `"-l"`.

Como este curso não abordou o uso de ponteiros, não é necessário preocupar-se com a notação de ponteiros, pois é possível, nesse caso, fazer o uso apenas por meio de índices, como em *arrays*.

O corpo da função `main` do programa acima possui a declaração de uma variável do tipo inteiro que será usado pelo laço `for`. Veja que a estrutura do laço `for` é identica à da linguagem Arduino. O laço `for` será percorrido tantas vezes quanto for o valor de `argc`. Dentro do laço `for` são mostrados na tela o valor da variável `i` e o valor na posição correspondente à `i` do *array*. Veja que o uso de `argv[i]` com a especificação de conversão `%s` mostra exatamente a cadeia de caracteres apontada pelo ponteiro naquela posição, não sendo necessário preocupar-se com a notação de ponteiros.

Como exemplo, se o nome do programa acima, fosse `teste` e a chamada fosse realizada da como `teste arg1 arg2 joazinho`, o valor de `argc` seria quatro e o programa mostraria o seguinte na tela:

```
0: teste
1: arg1
2: arg2
3: joazinho
```

5 Problemas

Problema 1

Escreva um programa em linguagem C que reproduza o Problema 2 da Aula 1 e que funcione como um Arduino, isto é, que contenha as funções `setup()` e `loop()`. Como o terminal corre indefinidamente, não será necessário usar uma função como a `clear()`. Para fazer o programa esperar um pouco antes de cada atualização da tela, use a função `sleep()` por meio da biblioteca `unistd.h` no Linux ou a função `Sleep()` por meio da biblioteca `Windows.h` no Windows, ao invés de `delay()`.

Problema 2

Escreva um programa em linguagem C que faz a conversão de valores de temperatura de graus Celsius para graus Fahrenheit e de graus Fahrenheit para graus Celsius, dado que T_C é a temperatura em graus Celsius, T_F é a temperatura em graus Fahrenheit e:

$$T_\mathrm{F} = \frac{9}{5} T_\mathrm{C} + 32$$

$$T_\mathrm{C} = \frac{5}{9} (T_\mathrm{F} - 32)$$

A temperatura a ser convertida e a escala para a qual deve ocorrer a conversão devem ser passados como argumento na chamada do programa, ou seja, via função `main`. Por exemplo, para converter a temperatura de 25,0 °C para graus Fahrenheit a chamada do programa `converte` deve ser da seguinte maneira: `converte F 25.0`.
Para converter a cadeia de caracteres de `argv` para um número real use a função `atof` que recebe uma cadeia de caracteres e a transforma no número real correspondente, por meio da biblioteca `stdlib.h`. Para comparar strings, pesquise a função `strcmp`.