Universidade Federal Fluminense
Escola de Engenharia
Curso de Engenharia de Telecomunicações
Programa de Educação Tutorial

# Apostila para Introdução à Linguagem de Programação C

(Versão: A2016M03D16)

Autores: Thiago Chequer Coelho
Gustavo Araujo Machado
Rodrigo Duque Ramos Brasil

Tutor: Alexandre Santos de la Vega

Niterói-RJ
Março / 2016

# Sumário

# 1 Definição

C é uma linguagem de programação criada por Dennis Ritchie, em 1972 e que continua a ser muito utilizada até os dias atuais. Seu uso é bastante amplo, podendo ser utilizada para automatizar ferramentas e construção de softwares. Este tutorial demonstrará as funções e comandos de controles de fluxo básicos em C, apenas com o intuito de orientar um usuário a entender sobre a linguagem e ver como o código funciona.

# 2 Compiladores

Os compiladores são ferramentas utilizadas para traduzir uma determinada linguagem (código fonte) em um código objeto. Esse código objeto permite a um programa, através de uma linguagem binária, "conversar" com um sistema, por exemplo. Para C, é bastante utilizado o GCC (em Linux) e o Dev C++ (para Windows). No caso do GCC, temos um tutorial explicando a utilização dele no link:

*http://www.telecom.uff.br/pet/petws/downloads/dicas/DicasPetTeleGcc.pdf*

Com um compilador, fica mais fácil entender o funcionamento de um programa. No caso do Dev C++ (ou até o Code:Blocks), que é mais voltada para usuários do Windows, não disponibilizamos de uma apostila completa, no entanto, é uma opção para quem não tiver acesso ao ambiente unix.

# 3 Bibliotecas

Fazendo uma analogia, podemos comparar um programa com uma casa. Como sabemos, uma casa é dividida em vários cômodos com determinadas funções. Por exemplo, o quarto é onde podemos dormir (usar a cama) ou estudar (usar a mesa). Na sala, é um lugar de descanso, em que podemos assistir televisão (utilizar a TV), e na cozinha podemos utilizar o liquidificador e o fogão. Os cômodos seriam as bibliotecas e os objetos citados (como cama, mesa, etc) seriam funções.

Num programa, podemos escrever um código e colocá-lo como uma função, armazenando em uma biblioteca. Essa biblioteca pode armazenar inúmeras funções de variados tipos. Para deixar uma biblioteca organizada e não muito pesada, os programadores costumam dividir suas funções em "cômodos" diferentes (bibliotecas). Dessa forma, um usuário pode andar livremente em sua casa (programar), podendo incluir novos cômodos em sua casa à medida que ele precise. A biblioteca mais utilizada em C é a *stdio*, que contém as suas funções básicas (como escrever na tela, ler um valor, etc). A inclusão de bibliotecas ocorre, na grande maioria das vezes, em primeiro lugar, sendo feita da seguinte forma: `#include <stdio.h>`

.

# 4 Comentando o código

Dependendo do programa que estamos criando, podemos ter um código bem simples, ou até mesmo, muito complexo. Pensando à frente, é válido que o código seja bem organizado e de fácil compreensão. Por isso, é imprescindível uma estrutura que possibilite um entendimento do código. Uma boa maneira de organização é o uso de comentários. Em C podemos comentar de duas maneiras: Utilizando o comando "//", podemos escrever uma linha de comentário. Já com o uso de "/ * `seu texto aqui` * /", com o texto escrito entre os asteriscos, é possível escrever blocos inteiros de comentários. O código abaixo mostra as maneiras de comentar:

```
#include<stdio.h>

void main(){

//Uma forma de comentario.

printf("Voce lera apenas esta frase");

/* Segunda forma de comentario.
Tente executar e vera que nada alem do que esta
fora deste bloco de comentario aparecera*/

}
```

# 5 Manipulando dados

## 5.1 Variáveis

As variáveis são como caixas onde você pode armazenar alguma informação. Computacionalmente, elas são posições na memória do computador capazes de armazenar um determinado valor atribuído pelo usuário ou por alguma etapa do programa. Para o computador, as variáveis são apenas conjuntos de *bits*, mas para facilitar o desempenho da execução do programa existem os tipos básicos de dados:

| **Tipo de dados** | **Descrição** |
|---|---|
| void | Declara explicitamente uma função que não retorna valor algum. |
| char | Armazena apenas uma letra. |
| int | Armazena apenas um número inteiro. |
| float | Armazena um número real com seis dígitos de precisão. |
| double | Armazena um número real com dez dígitos de precisão. |

Para utilizarmos uma variável é necessário declará-la. A declaração consiste do tipo e nome da variável. Veja abaixo um exemplo com declarações de algumas variáveis.

```
#include <stdio.h>

int main(){

char letra;
int num_inteiro;
float num_real;

//Seu codigo aqui

}
```

Variáveis também podem ter valores atribuídos ao serem declaradas. Veja o mesmo exemplo anterior, mas agora com as variáveis com valores atribuídos na declação.

```
#include <stdio.h>

int main(){

char letra = 'C';
int num_inteiro = 10;
float num_real = 12.5;

//Seu codigo aqui

}
```

## 5.2 Variáveis globais e locais

Uma variável global é declarada logo abaixo ao campo de inserção de bibliotecas. Desta forma, uma vez declarada uma variável global, todas as funções contidas no código reconhecerão esta variável.

Uma variável local é toda variável declarada dentro de uma função, e diferentemente das variáveis globais, esta será reconhecida apenas dentro do corpo desta função.

Abaixo, segue um exemplo de como declarar variáveis globais e locais.

```
#include <stdio.h>

int a = 10; //Esta e uma variavel global

int main(){

int b = 5; //Esta e uma variavel local
printf("Escrevendo a variavel global a = %d \n", a);
printf("Escrevendo a variavel local b = %d \n", b);

return 0;

}
```

Perceba que a variável global foi declarada antes da função main. Já a variável local foi declarada no interior da função. Neste exemplo simples, o uso de variáveis globais e locais não tem diferença considerável. Entretanto, quando trabalhamos com muitas funções em um código, utilizar as variáveis de forma estratégica pode favorecer o desempenho do programa e até mesmo evitar erros.

## 5.3 Modificadores

Anteriormente, vimos os tipos de dados que podemos manipular no programa. Em algumas ocasiões, os tipos básicos não nos atende da forma como esperamos, seja na quantidade de memória alocada, ou no valor máximo que pode-se armazenar. Por conta disso, os tipos básicos de variáveis podem ser modificadas. Essa propriedade se torna útil à medida que é preciso trabalhar com dados que necessitam de muitos bits de memória. A tabela abaixo mostra os modificadores aplicados a cada tipo.

| **Tipo** | **Bits** | **Descrição** |
|---|---|---|
| char | 8 | Define uma faixa de valores de -127 a 127 |
| signed char | 8 | Idem ao char |
| unsigned char | 8 | Define uma faixa de valores de 0 a 255 |
| int | 16 | Define uma faixa de valores de -32.767 a 32.767 |
| signed int | 16 | Idem ao int |
| unsigned int | 16 | Define uma faixa de valores de 0 a 65.535 |
| short int | 8 | Idem ao int |
| long int | 32 | Define uma faixa de valores de -2.147.483.647 a 2.147.483.647 |
| signed long int | 32 | Idem ao long int |
| unsigned long int | 32 | Define uma faixa de valores de 0 a 4.294.967.295 |
| float | 32 | Define faixa mínima de seis dígitos de precisão |
| double | 64 | Define faixa mínima de dez dígitos de precisão |
| long double | 80 | Define uma faixa mínima de dez dígitos de precisão |

Perceba que os modificadores *signed* não mudam os tipos básicos. Isso se dá pelo fato de que os tipos básicos já são, por padrão, modificados desta maneira.

## 5.4 Constantes

Muitas vezes, temos que repetir um mesmo valor muitas vezes em todo o código do programa. O problema surge, eventualmente, quando há a necessidade de mudar esses valores. Tecnicamente falando, dependendo do tamanho do programa, é inviável procurar linha por linha o valor a ser substituído. Uma boa forma de contornar esse problema é o uso de constantes. Em C é possível definir constantes. O termo utilizado na declaração é o `#define`. A declaração é feita logo abaixo da declaração das bibliotecas e não é utilizado o símbolo =, como no caso da atribuição de valores a variáveis. Veja o exemplo abaixo:

```
#include<stdio.h>

#define TESTE 65
#define TESTE2 'A'

int main(){

printf("%d", TESTE);
printf("%c", TESTE);
printf("%c", TESTE2);

return 0;

}
```

Criando as constantes TESTE e TESTE2 mostramos que elas não necessariamente precisam ser números. Compile o código acima e perceba que a variável TESTE, quando mostrada como tipo *char* (caractere), mesmo sendo um número, é mostrada como uma letra.

## 5.5 O uso de vetores e matrizes

Vetores são conjuntos de valores de um tipo de variável. Diferentemente de variáveis normais, que armazenam apenas um dado, um vetor pode armazenar vários dados de um mesmo tipo. A sintaxe para a declaração de um vetor é: `tipo nome_vetor[num_de_elementos]`. O exemplo abaixo demonstra o modo correto de declarar vetores:

```
int vetor_inteiro[10]; //Vetor de numeros inteiros
char vetor_letras[15]; //Vetor de letras (string)
float vetor_reais[5]; //Vetor de numeros reais
```

Matrizes, assim como vetores, armazenam conjuntos de informações. Entretanto, diferentemente do caso anterior, as matrizes são multidimensionais. A sintaxe para declaração de uma matriz é a seguinte: `tipo nome_vetor[num_de_linhas][num_de_colunas]`

## 5.6 Escrita e leitura em C

### 5.6.1 A função *printf*

A função *printf* é utilizada para escrever textos e dados na tela. Para utilizá-la, escrevemos o conteúdo que desejamos entre aspas da seguinte forma:

```
#include <stdio.h>

int main(){

printf("Hello World! \n");

return 0;

}
```

O programa apenas escreve **Hello World!** na tela. O termo `\n` é utilizado para pular linha. As chaves são para iniciar e encerrar a função *main*. O *int* antes da função *main* demostra que a função retornará um valor inteiro, justificando o *return 0* na penúltima linha.

### 5.6.2 A função *scanf*

A função *scanf* lê uma letra que o usuário digitará. É utilizada para armazenar um valor em uma dada variável.

Exemplo:

O programa apenas lerá o valor que o usuário digitar, armazenará o valor lido na variável "a" e escreverá o número na tela. `"%d"` indica que a variável a ser lida é do tipo inteiro e o `&a` está armazenando o valor lido em "a". O & indica que o valor deve ser armazenado no endereço de memória da variável a. Caso não tivéssemos o &, o valor não seria armazenado e o código apresentaria erros.

```
#include <stdio.h>

int main (){

int a;

printf ("Escreva um numero: ");
scanf ("%d", &a);
printf ("O numero digitado foi: %d.", a);

return 0;

}
```

### 5.6.3 Especificadores de formato

O termo `"%d"` é chamado especificador de formato. Os especificadores de formato fornecem informações sobre o tipo de parâmetro que serão impressos pela função *printf*. Abaixo, seguem alguns exemplos de especificadores de formatos utilizados em C.

| **Especificador** | **Descrição** |
|---|---|
| %c | Utilizado para caracteres |
| %s | Utilizado para strings |
| %d | Utilizado para números inteiros |
| %u | Utilizado para números inteiros decimais não sinalizados |
| %f | Utilizado para números reais de poucas casas decimais |
| %ld | Utilizado para números inteiros muito grandes (long int) |
| %i | Também utilizado para números inteiros muito grandes |
| %lx | Utilizado para números hexadecimais muito longos |
| %p | Utilizado para mostrar o endereço de memória de uma variável |
| %X | Mostra o valor armazenado em uma variável na base hexadecimal |
| %o | Mostra o valor armazenado em uma variável na base octal |
| %e | Mostra o valor armazenado em uma variável em base de dez |
| %% | Imprime o símbolo de porcentagem |
| %g | A função printf decide mostrar o valor em exponencial ou não |

# 6 Operadores

## 6.1 Operadores matemáticos

As operações matemáticas básicas na linguagem C são simples. Neste tutorial, serão demonstrados apenas os tipos mais básicos de operações:

| Operação | Exemplo |
|---|---|
| Soma | a = b + c; |
| Substração | a = b - c; |
| Multiplicação | a = b * c; |
| Divisão | a = b / c; |
| Potenciação | a = pow( b, c); |
| Raiz quadrada | a = sqrt(x); |
| Seno | a = sin(b); |
| Cosseno | a = cos(b); |
| Tangente | a = tan(b); |
| Arco seno | a = asin(b); |
| Arco cosseno | a = acos(b); |
| Arco tangente | a = atan(b); |

Tome cuidado com uma divisão, uma vez que pode resultar em um número real e você deve armazenar esse valor em uma variável do tipo float.

As funções trigonométricas operam ângulos em radianos. Tome cuidado para garantir que os ângulos não estejam em graus.

As operações de potenciação e raiz quadrada necessitam da biblioteca *math.h*.

## 6.2 Operadores relacionais e lógicos

Os operadores relacionais e lógicos são utilizados para comparar variáveis ou expressões. A resposta da comparação é um retorno verdadeiro(1) ou falso(0). Observe a tabela abaixo com os principais operadores utilizados em C.

| **Operador** | **Descrição** |
|---|---|
| < | Menor que |
| > | Maior que |
| == | Igual a |
| <= | Menor ou igual que |
| >= | Maior ou igual que |
| ! = | Diferente de |
| \|\| | Estrutura lógica OU |
| && | Estrutura lógica E |
| ! | Estrutura lógica NOT |

# 7 Estruturas condicionais

## 7.1 O teste condicional *if/else*

O comando *if* verifica uma condição. Caso a condição seja verdadeira, o programa realizará o que está na função (entre as chaves). Caso contrário, o programa ignorará o que está presente no *if* (entre as chaves).

Caso você queira realizar uma outra operação se a condição presente no *if* não for estabelecida, o else (caso contrário) cumprirá esse papel. Se a condição do *if* for satisfeita, o programa ignorará o que está presente no *else*.

```
#include <stdio.h>

void main (){

int valor;

printf ( Escreva um numero: ");
scanf ("%d", &valor);

if(valor > 5){
        printf("O valor e maior que 5. \n");
        }
else{
        printf("O valor nao e maior que 5. \n");
        }
}
```

O programa irá ler um número digitado pelo usuário e armazenará em uma variável o "valor". Feito isso, a função if verificará se o "valor" é maior que 5. Se for, aparecerá na tela:

**O valor é maior que 5**

Caso contrário, aparecerá:

**O valor não é maior que 5**

## 7.2 O teste condicional *switch*

A funcionalidade do teste condicional *switch* é bem parecido ao *if/else*. A diferença é que o *switch* permite diversas condições de comparações diretas como valor de dada variável. Observe abaixo o exemplo:

```
#include <stdio.h>

int num_inteiro;

int main(){

printf("Escreva um numero inteiro\n");
scanf("%d", &num_inteiro);

 switch (num_inteiro){
    case 1: printf("Voce escreveu o numero um"); break;
    case 2: printf("Voce escreveu o numero dois"); break;
    case 3: printf("Voce escreveu o numero tres"); break;
    default: printf("Voce escreveu um numero maior que tres"); break;
  }

return 0;

}
```

O código acima compara o valor lido na variável `num_inteiro` com os casos disponíveis. Se nenhum dos casos for satisfeito, a rotina indicada no `default`.

# 8 Estruturas de repetição

## 8.1 O laço *for*

O comando *for* é utilizado para determinar repetições presentes em seu intervalo. Enquanto a condição dada for verdadeira, a função se repetirá por várias vezes.

Exemplo:

```
#include <stdio.h>

void main(){

int i;

   for(i = 1; i <= 10; i++){
          printf("%d", i);
          }

}
```

Se você compilar esse programa, aparecerá na tela:

**12345678910**

Isso se deve porque na função, a variável i começa como 1. Terminando a parte do código entre chaves, a função irá verificar se a condição pedida (i <= 10) ainda é verdadeira. Caso seja, repetirá o código entre as chaves, mas, dessa vez, com um novo valor para i (i++), que estará somando um à variável i (i = i + 1), e , por isso, terá valor 2.

Uma iteração do tipo **i++** é conhecida como incrementação, e decrementação para o caso de **i−−**.

Isso se repete até i valer 10, que finalizará a condição pedida. Caso escrevêssemos "for(i = 1; i <= 10; i=i+2)", i aumentaria de 2 em 2 a cada for, resultando em:

**13579**

O laço *for* também é bastante útil quando utilizado em conjunto com vetores.

```
#include <stdio.h>

void main(){

int i;
int cont[10] ;

  for(i = 0; i < 10; i++){
        scanf("%d", &cont[i]);
        }
  for(i = 0; i < 10; i++){
        printf("%d", cont[i]);
        }
}
```

Se você compilar esse programa, primeiro você irá digitar 10 números inteiros, e depois, os números serão mostrados na tela. O índice i irá percorrer por todas as casas o vetor cont. Assim, cada valor inserido pelo usuário será armazenado em um espaço de memória do vetor.

## 8.2 O laço *while*

O comando *while* é semelhante ao *for*, podendo realizar as mesmas tarefas. No entanto, o *while* é mais utilizado para realizar funções infinitas. O teste que determina a condição de continuidade do laço é feito no início. No caso da condição ser falsa, a rotina contida no laço não é executada. Além disso, o *while* sustenta uma condição que pode depender de outros fatores que não sejam incrementos de variáveis.

```
#include <stdio.h>

void main (){

int valor = 1;

  while(valor <= 10){
        printf("%d", valor);
        valor = valor + 1;

        }

}
```

Observe que caso não houvesse a linha "*valor = valor + 1;*" , o *loop* (repetição) seria infinito. Diferentemente do *for*, o *while* necessita da atenção o programador quanto ao fim do laço. Caso não seja apresentado método de fuga da repetição, o laço será infinito, podendo causar problemas na execução do código.

## 8.3 O laço *do while*

O laço *do while* tem, por essência, a mesma função do laço *while*. Entretanto, diferentemente da função *while* a função *do while* executa ao menos uma vez a rotina contida dentro do laço e compara a condição ao final. Se a condição for verdadeira, a repetição é sustentada, caso contrário, o código executa o próximo passo do código.

```
#include <stdio.h>

void main (){

int valor = 1;

  do{
        printf("%d", valor);
        valor = valor + 1;
        while(valor<=10);


   }
}
```

# 9 Comando *break*

O comando *break* é comumente usado nas estruturas de repetição, como: *while*, *for*, *do-while* e em estruturas condicionais, como: *switch*. Basicamente, ele é utilizado para sair de forma abrupta de uma estrutura. Se no código utilizado uma instrução de *break* ocorrer, ela sairá da iteração em questão.

```
#include <stdio.h>

void main (){

int a, b;

printf("Digite um valor para a:\n");
scanf("%d", &a);
printf("Digite um valor para b:\n");
scanf("%d", &a);

 while(a<=b){
        a = a+1;
        if(a == 5){
                break;
        }

    printf("Novo valor de a:");
    printf("%d\n, a);

  }

return 0;

}
```

# 10 O uso de Funções

Uma função é um bloco de código externo ao corpo principal do programa, o qual pode ser solicitado quantas vezes necessário. O modo de escrever uma função é a seguinte:

```
tipo_de_retorno nome_da_funcao(lista de parametros){

// codigo da funcao

}
```

Para definir os parâmetros de uma função o programador deve explicitá-los como se estive declarando uma variável, entre os parênteses do cabeçalho da função. Caso precise declarar mais de um parâmetro, basta separá-los por vírgulas. A forma de chamar a função é a seguinte:

```
#include <stdio.h>

void SOMA(float a, int b){ // basta separar os parametros por virgulas

float result; // a declaracao de variaveis e igual ao que
// se faz na funcao main

result = a+b;
printf("A soma de %d com %6.3f = %6.3f\n, a,b,Result);

}

void main(){

int a;
float b;

a = 10;
b = 12.3;

SOMA(b,a); // Chamada da funcao SOMA(12.3,10)

}
```

Funções são importantes para melhor organização do código. Isso evita o acúmulo de linhas de código no corpo principal e possibilita uma melhor interpretação do código pelo programador.