Aula 1: Introdução a Linguagem C

Profa.: Andréia Rodrigues Casare
E-mail: casareandreia@gmail.com
andreia.casare01@fatec.sp.gov.br

Criado: 02/2015
Atualizado: 02/2023

Histórico

- A Linguagem C foi criada em 1970 por Dennis Ritchie. É estreitamente associada ao sistema operacional UNIX, já que as versões atuais do próprio sistema foram desenvolvidas com a linguagem C.
- Devido ao crescente uso e interesse da comunidade de computação, em 1983, o American National Standards Institute (ANSI), estabeleceu um comitê para prover uma definição moderna e abrangente da linguagem C. O resultado foi uma padronização denominada ANSI-C em 1988. A maior contribuição deste trabalho foi a incorporação de uma biblioteca padrão, presentes em todas as variações/versões da linguagem, que fornece funções de acesso ao sistema operacional, entrada e saída formatada, alocação de memória, manipulação de strings (cadeias de caracteres), etc.

Conceitos básicos

- A filosofia básica da linguagem C é que os programadores devem estar cientes do que estão fazendo, ou seja, supõe-se que eles saibam o que estão mandando o computador fazer, e explicitem completamente as suas instruções. Assim, ela **alia a elegância e a flexibilidade das linguagens de alto nível** (ex: suporte ao conceito de tipo de dados) **com o poder das linguagens de baixo nível** (ex: manipulação de bits, bytes e endereços).
- O C é uma linguagem de programação de finalidade geral, utilizada no desenvolvimento de diversos tipos de aplicação, como processadores de texto, sistemas operacionais, sistemas de comunicação, programas para solução de problemas de engenharia, física, química e outras ciências, etc.

Conceitos básicos

- Após a implementação, **o programa-fonte** (um ou mais arquivos-fonte) é **submetido aos processos de compilação e linkedição** para gerar o programa executável (com extensão “exe”). Durante o processo de compilação, **cada arquivo-fonte é compilado separadamente**, produzindo um arquivo de código-objeto com a extensão “obj”. Estes arquivos-objeto contêm instruções em linguagem de máquina (códigos binários) entendidas somente pelos microprocessadores. Na linkedição, **todos os arquivos-objetos pertencentes ao projeto, bem como as bibliotecas declaradas nos códigos-fonte são processadas em conjunto**, visando a produção do arquivo executável correspondente.

Características gerais

- Gera **programas formados basicamente por funções**, o que **facilita a modularização e a passagem de parâmetros** entre os módulos;
- Inicia a execução a partir da **função main()**, necessária em todos os programas;
- Uso de **chaves** (**{ }**) para agrupar comandos pertencentes a uma estrutura lógica (ex: if-else, do-while, for, etc.) ou a uma função;
- Uso do **ponto e vírgula** (**;**) ao final de cada comando;
- É "**case sensitive**", ou seja, **o compilador difere maiúsculas de minúsculas**. Assim, se declarar uma variável de nome *idade*, esta será diferente de *Idade*, *IDADE*, etc. Além disso, **todos os comandos da linguagem devem ser escritos em minúsculo**.

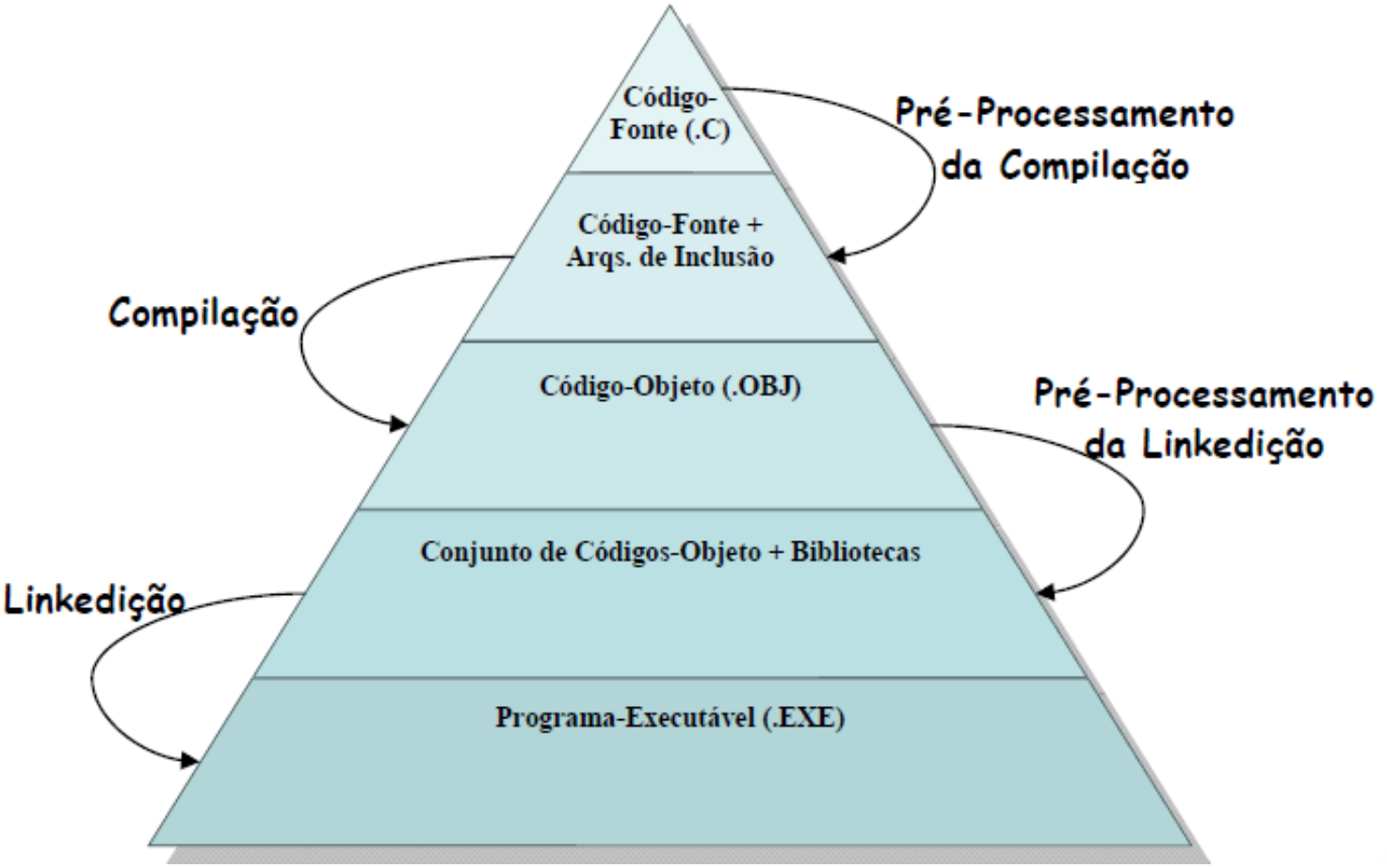
Estrutura básica de um programa C

```
Diretivas
Declarações de variáveis/constantes globais;
Prototipação de funções;
tipo_dado main(lista_parametros) /* Função Principal e Obrigatória */
{
        Declarações de variáveis/constantes locais;
        Comandos;
        Estruturas de controle (seleção e/ou repetição);
        Comentários;
        Chamada a outras funções;
}
tipo_dado func1(lista_parametros)
{
        Bloco de comandos;
}
```

Comentários no Programa

- Comentários são pedaços de código-fonte descartados do código pelo compilador. Eles não fazem nada. O objetivo deles é somente permitir que o programador insira notas ou descrições dentro do código-fonte.
- **// comentário de linha**
- **/* comentário de bloco */**

Etapas do processo de compilação/ linkedição



Protótipos de funções

- Protótipos permitem que o C **forneça uma verificação mais forte de tipos**, ou seja, através dos protótipos, o C pode encontrar e apresentar quaisquer conversões ilegais de tipo entre o argumento usado para chamar a função (argumento atual) e aquele utilizado na definição de seus parâmetros (argumento formal). Além disso, o uso de protótipo também permite **encontrar diferenças entre o número destes argumentos**.
- A forma geral da definição de protótipo de função:
  - ***tipo nm_função (tipo nm_arg1, tipo nm_arg2, ... , tipo nm_argN);***

Exemplo

```
# include <stdio.h>
unsigned int NUM;
int fat (unsigned int NRO); /* Protótipo da Função fat() */
void main (void)
{
        scanf("%u", &NUM);
        printf("\nO fatorial de %u é %d", NUM, fat(NUM));
}
Int fat(unsigned int NRO)
{
        if NRO < 2
                return (1);
        else
                return (NRO * fat(NRO-1));
}
```

Tipos básicos de dados

Tipos	Descrição	Tamanho Aprox.	Faixa Mínima
char	Caractere	8 bits = 1 byte	-127 a 127
int	Inteiro	16 bits = 2 bytes	-32.767 a 32.767
float	Ponto flutuante	32 bits = 4 bytes	7 dígitos de precisão
double	Ponto flutuante de precisão dupla	64 bits = 8 bytes	10 dígitos de precisão
void	Sem valor		

Tipos de dados modificados

Tipo	Num de bits	Formato para leitura com scanf	Intervalo	
			Inicio	Fim
Char	8	%c	-128	127
unsigned char	8	%c	0	255
Signed char	8	%c	-128	127
Int	16	%i ou %d	-32.768	32.767
unsigned int	16	%u	0	65.535
signed int	16	%i ou %d	-32.768	32.767
Short int	16	%hi ou %hd	-32.768	32.767
unsigned short int	16	%hu	0	65.535
signed short int	16	%hi ou %hd	-32.768	32.767
Long int	32	%li ou %ld	-2.147.483.648	2.147.483.647
signed long int	32	%li ou %ld	-2.147.483.648	2.147.483.647
unsigned long int	32	%lu	0	4.294.967.295
Float*	32	%f, %e ou %g	3,4E-38	3.4E+38
Double*	64	%lf, %le ou %lg	1,7E-308	1,7E+308
long double*	80	%Lf	3,4E-4932	3,4E+4932

Constantes

- **Variáveis do tipo *const* não podem ser modificadas pelo programa**, podendo, entretanto, receber um valor inicial.
  - ***const int pi = 3.14;***

Declaração de variáveis

- Todas as variáveis do C devem ser declaradas antes de serem usadas. A forma geral de uma declaração é:
  - ***tipo lista_de_variáveis;***

- Sendo:
  - ***Tipo***: um tipo de dado válido em C mais quaisquer modificadores; e
  - ***Lista_de_variáveis***: um ou mais nomes de identificadores separados por vírgula.

Inicialização de variáveis

- ***int NRO = 1000;***
- ***char LETRA = 'a', NOME[80] = "José";***
- A variável LETRA receberá o caracter 'a' e o vetor NOME receberá a string "José". Cada letra da string será associada a uma posição do vetor NOME, como ilustrado:
  - ***NOME[0] = 'J', NOME[1] = 'o', NOME[2] = 's', NOME[3] = 'é', NOME[4] = '\0'***
- Observe que, apesar da string de inicialização possuir apenas 4 caracteres, é inserido um caracter especial '**\0**' (***terminador nulo***) na quinta posição do vetor (*NOME[4]*). Este caracter deve ser utilizado nos testes condicionais das estruturas de controle (seleção ou repetição) para identificar o final da string, como no exemplo:

Inicialização de variáveis

```
void main(void)
{
        char LETRA, FRASE[100];
        int I = 0, CONT = 0;
        printf("Digite uma frase:");
        scanf("%s\n\n",&FRASE);
        printf("Digite a letra que será contada:");
        scanf("%c\n\n",&LETRA);
        while ((FRASE[I] != '\0') && (I <= 100))
        {
                If (FRASE[I] = LETRA)
                {
                        CONT++; /* Equivalente ao comando CONT = CONT+1 */
                }
                ++I;
        }
        Printf("A letra %c apareceu %d vezes dentro da frase.\n", LETRA, CONT);
}
```

Operadores

Operador	*Finalidade*
+	Soma (inteira e ponto flutuante)
-	Subtração ou Troca de Sinal (inteira e ponto flutuante)
*	Multiplicação (inteira e ponto flutuante)
/	Divisão (inteira e ponto flutuante)
%	Resto da Divisão (somente de inteiros)

Operadores de incremento e decremento

- A linguagem C permite a utilização dos operadores unários “**++**” e “**--**“ para incrementar (somar 1) ou decrementar (subtrair 1) o conteúdo de uma variável, respectivamente. Exemplos de sua utilização:
- *++X; /* Equivalente à expressão X = X + 1 */*
- *X++; /* Equivalente à expressão X = X + 1 */*
- *--X; /* Equivalente à expressão X = X - 1 */*
- *X--; /* Equivalente à expressão X = X - 1 */*

Operadores de incremento e decremento

- Estes operadores podem ser pré-fixados ou pós-fixados. A diferença é que quando são pré-fixados, eles incrementam a variável antes de usar seu valor. Quando são pós-fixados, eles utilizam o valor da variável antes de incrementá-la.

```
int X = 10, Y;
Y = X++;
/* Neste caso Y = 10 e X = 11 */
```

```
int X = 10, Y;
Y = ++X;
/* Neste caso Y e X = 11 */
```

Operadores relacionais

Operador	*Finalidade*
==	Igual
>	Maior
<	Menor
>=	Maior e igual
<=	Menor e igual
!=	Diferente

Operadores lógicos

Operador	*Finalidade*
&&	AND (E)
||	OR (OU)
!	NOT (NÃO)

Comandos de entrada e saída – printf()

- A função ***printf()*** possibilita a saída de valores (constantes, variáveis ou resultados de expressões). Esta função converte e imprime seus argumentos na saída padrão seguindo o formato especificado na ***string de controle***. Sua sintaxe geral é:
  - ***printf("string de controle", lista de argumentos);***

- A ***string de controle***, delimitada por aspas, descreve tudo que a função irá colocar na tela. Esta string não mostra apenas os caracteres que devem serão apresentados, mas também os respectivos formatos e posições das constantes, variáveis e expressões listadas na ***lista de argumentos***.

Alguns especificadores de formato da função printf()

Especificador	Tipo do Argumento	Descrição
%d	int	Valor inteiro decimal
%i	int	Valor inteiro decimal
%o	int	Valor inteiro octal
%x	int	Valor inteiro hexadecimal
%X	int	Valor inteiro hexadecimal
%u	int	Valor inteiro decimal sem sinal (*unsigned int*)
%c	int	Um caracter em formato ASCII (código binário correspondente)
%s	char	Uma cadeia de caracteres (*string*) terminada em "\0"
%f	float	Valor em ponto flutuante no formato [-]m.dddddd, onde o Nº de d's é dado pela precisão (padrão é 6)
%e	float ou double	Valor em ponto flutuante em notação exponencial no formato [-]m.ddddd e±xx, onde o Nº de d's é dado pela precisão
%E	float ou double	Valor em ponto flutuante em notação exponencial no formato [-]m.ddddd E±xx, onde o Nº de d's é dado pela precisão
%g	float ou double	Valor em ponto flutuante no formato %e (quando o expoente for menor que -4 ou igual a precisão) ou %f (nos demais casos). Zeros adicionais e um ponto decimal final não são impressos
%G	float ou double	Valor em ponto flutuante no formato %E (quando o expoente for menor que -4 ou igual a precisão) ou %f (nos demais casos). Zeros adicionais e um ponto decimal final não são impressos
%%	-	Exibe o caracter '%'

Exemplo formato printf()

```
#include <stdio.h>
void main(void) {
    short int CH = 47;
    printf("%04d \n", 21);
    printf("%6d \n", 21);
    printf("%10.3f \n", 3456.78);
    printf("%3.1f \n", 3456.78);
    printf("%hd %c \n", CH);
}
```

```
0021
    21
 3456.780
3456.8
47 /
```

Comandos de entrada e saída – scanf()

- A função ***scanf()*** lê caracteres da entrada padrão (teclado), interpreta-os segundo a especificação de formato e armazena os resultados em variáveis declaradas no programa. Sintaxe:
- ***scanf ("string de controle", lista de endereços de memória das variáveis);***
- **A representação do endereço de memória das variáveis** que receberão os dados de entrada é fornecido **pelo operador "&", seguido pelo nome da variável**. Além disso, assim como na função ***printf()***, esta função também utiliza os caracteres especiais de controle via códigos de barra invertida.

Alguns especificadores de formato da função scanf()

Especificador	Descrição
%d	Indica um valor inteiro decimal
%i	Indica um valor inteiro (podendo estar na base decimal, octal com inicial 0 (zero) ou hexadecimal com inicial **0X**
%u	Indica um valor inteiro decimal sem sinal
%c	Indica um caracter (*char*)
%s	Indica uma *string*
%f, %e, %g	Indica um valor em ponto flutuante de precisão simples (*float*)
%lf, %le, %lg	Indica um valor em ponto flutuante de precisão dupla (*double*)
%o	Indica um valor inteiro octal (com ou sem zero inicial)
%x ou %X	Indica um valor inteiro hexadecimal (com ou sem "0x" inicial)

Exemplo

```c
# include <stdio.h>
int main(void)
{
    float SOMA, NRO;
    int CONT;
    CONT = SOMA = 0;
    printf ("\n Entre com os números a serem somados ou digite CTRL-Z para terminar \n");
    while (scanf("%f",&NRO) == 1)
    {
        CONT++;
        if (CONT == 10)
        {
            printf("\t Subtotal: %.2f \n", SOMA += NRO);
            CONT = 0;
        }
        else{
            SOMA += NRO;
        }
        printf ("\t Total: %.2f \n", SOMA);
    }
}
```

Declarações de variáveis – exemplos

- int a,b, c;
- float n1, n2;
- double x, y, media, soma;

#include <iostream>

- Comandos que começam com um sinal de numeração (#) são diretivas do préprocessador.
- Elas não são linhas de código executáveis, mas indicações para o compilador.
- Nesse caso, o comando #include <iostream> diz ao pré-processador do compilador para incluir o arquivo de cabeçalho padrão iostream. Esse arquivo específico inclui as declarações da biblioteca básica de entrada-saída do C++.

Impressão de códigos especiais

Código	Ação
\n	Leva o cursor para a próxima linha
\t	Executa uma tabulação
\b	Executa um retrocesso
\f	Leva o cursor para a próxima página
\a	Emite um sinal sonoro (*beep*)
\"	Exibe o caractere "
\\	Exibe o caractere \
%%	Exibe o caractere %

Exemplo

```c
# include <stdio.h>
# include <locale.h>

int main(void)
{
    setlocale(LC_ALL, "Portuguese");
    printf("%s tem %d","José",67);
    printf("\n olá mundo !!");
    printf("\a");
}
```

Exemplos definindo a quantidade casas decimais

```
#include <stdio.h>

int main()
{
 printf("Default: %f \n",3.1415169265);
 printf("Uma casa: %.1f \n",3.1415169265);
 printf("Duas casas: %.2f \n",3.1415169265);
 printf("Três casas: %.3f \n",3.1415169265);
 printf("Notação Cientifica: %e \n",3.1415169265);
}
```

A função main (principal)

- Todo programa em C/C++ existe uma função denominada *main*.
- A função main é o ponto de entrada do programa.
- Estrutura padrão

```
int main()
{
    printf("olá mundo");
}
```

Acentuação

- Para conseguir acentuar no C, deve ser usado a biblioteca locale.h com a função setlocale.
- Exemplo:

```
#include <stdio.h>
#include <locale.h>
int main(){
    setlocale(LC_ALL, "Portuguese");
    printf("olá mundo !!");
    printf("n ç é â");
    return 0;
}
```

Exemplo 1

```c
# include <stdio.h>
# include <locale.h>

int main(void)
{
    setlocale(LC_ALL, "Portuguese");
    float n1,n2,media;
    printf("\nDigite a primeira nota: ");
    scanf("%f",&n1);
    printf("\nDigite a segunda nota: ");
    scanf("%f",&n2);
    media = (n1+n2)/2;
    printf("Média: %.2f",media);
}
```

Exemplo 2

```
# include <stdio.h>
# include <locale.h>
# include <cstdlib>


int main(void)
{
    setlocale(LC_ALL, "Portuguese");
    float num1, num2, media;
    printf("Número 1: ");
    scanf("%f",&num1);
    printf("Número 2: ");
    scanf("%f",&num2);
    media = (3 * num1 + 5 * num2) / 8;
    printf("media %.2f:",media);
    system("pause");

}
```

Exemplo 3

```c
# include <stdio.h>
# include <locale.h>
# include <cstdlib>


int main(void)
{
    setlocale(LC_ALL, "Portuguese");
    float sal, novosal;
    printf("\nDigite o salário do funcionário :");
    scanf("%f",&sal);
    novosal = sal + sal * 25/100;
    printf("\nNovo salário = %.2f",novosal);
    system("pause");

}
```

Exercício

1. Crie um programa que receba dois números inteiros. Calcule:
   - Acrescente mais 10 ao primeiro número;
   - Acrescente mais 5 ao segundo número.
   - Calcule o produto desses dois números e exiba na tela o resultado.

Exercício

- Crie um programa que leia um valor de hora (24 horas) e informe quantos minutos se passaram desde o início do dia.