Introdução à linguagem C

Diego Raphael Amancio

Baseado no material do Prof. Thiago A. S. Pardo e do Prof. André Backes

Sintaxe e Comandos básicos de C

- A sintaxe são regras detalhadas para cada construção válida na linguagem C.

- **Tipos:** definem as propriedades dos dados manipulados em um programa.
- **Declarações:** expressam as partes do programa, podendo dar significado a um identificador, alocar memória, definir conteúdo inicial, definir funções.
- **Funções**: especificam as ações que um programa executa quando roda.

Funções e comentários

- As funções são as entidades operacionais básicas dos programas em C.
  - Exemplo: *printf*() e *scanf*()
- O programador também pode definir novas funções
- Os programas incluem também bibliotecas que permitem o uso de diversas funções que não estão definidas na linguagem
- Todo programa C inicia sua execução chamando a função main(), sendo obrigatória a sua declaração no programa principal.
- Comentários são colocados entre /**/ ou // (uma linha)

Exemplo de programa simples

```
#include <stdio.h>

//Bloco principal do programa
int main() {
   printf("Oi, Mundo.\n");
   return 0;
}
```

A linha include indica a biblioteca que o programa vai usar. Neste caso “stdio.h” é uma biblioteca que permite utilizar a função printf

Principais funções de I/O

- printf("expressão de controle", argumentos) ;
  - função de I/O, que permite escrever no dispositivo padrão (tela). A expressão de controle pode conter caracteres que serão exibidos e os códigos de formatação que indicam o formato em que os argumentos devem ser impressos. Cada argumento deve ser separado por vírgula.
- scanf("expressão de controle", argumentos);
  - é uma função de I/O que nos permite ler dados formatados da entrada padrão (teclado). A lista de argumentos deve consistir nos endereços das variáveis.

Printf- Comandos básicos

\n nova linha
\t tabulação
\" aspas
%c catacter simples
%d inteiro
%f ponto flutuante
%s cadeia de caracteres (string)
%u inteiro sem sinal

Exemplo:

```
printf( “Este é o numero dois: %d“ , 2 );

printf("%s está a %d milhões de milhas\ndo sol", "Vênus", 67);
```

Scanf-Comandos básicos

- Usa um operador para tipos básicos chamado operador de endereço e referenciado pelo símbolo "&“, que retorna o endereço do operando.

  - int x = 10; //Armazena o valor 10 em x
  - &x //retorna o endereço da memória onde x está

Exemplo-Scanf

```
#include<stdio.h>
int main() {
    int num;
    printf("Digite um número: ");
    scanf( "%d", &num);
    return 0;
}
```

Sintaxe e comandos básicos de C

- Cada instrução encerra com ; (ponto e vírgula)
- Letras minúsculas são diferentes de maiúsculas.
  - Palavra != palavra != PaLaVRA (case sensitive)
- As palavras reservadas da linguagem estão sempre em minúsculas.

Palavras reservadas (ANSI)

auto	break	case	char
const	continue	default	do
double	else	enum	extern
float	for	goto	if
int	long	register	return
short	signed	sizeof	static
struct	switch	typedef	union
unsigned	void	volatile	while

Sintaxe e comandos básicos de C

- As inclusões de bibliotecas devem estar acima de todas as outras declarações do programa.
- No início dos blocos declaramos todas as variáveis que vamos utilizar
- O comando de declaração de variáveis tem a sintaxe:

  Tipo nome1[=valor][,nome2[=valor]]...[,nomeN[=valor]

Exemplo

```
#include <stdio.h>

   void main () {
      int a = 10;
      printf (“Este é o valor de a : %d\n", a );
   }
```

Atribuição e Inicialização de variáveis

- Operador de atribuição (=)
  - int a; a = 10;
  - int a = 10; //inicializado na declaração
- Inicializar significa atribuir à mesma um valor inicial válido.
  - Ao se declarar a variável, a posição de memória da mesma contém um valor aleatório.

Onde declarar variáveis?

- Em três lugares, basicamente:
  - Dentro de funções: variáveis locais.
  - Na definição de parâmetros de funções: parâmetros formais.
  - Fora de todas as funções: variáveis globais.

Escopo das variáveis

- Escopo define onde e quando uma variável pode ser usada em um programa.
- **Variável global** - tem escopo em todo o programa:

```
#include <stdio.h>
int i = 0;                  /* variável global */
                            /* visível em todo arquivo */

void incr_i() { i++;}
...
void main() { incr_i(); printf("%d", i);}
```

Exercicio

```
#include <stdio.h>
int segundos = 34706;

void main ()
    {
       int minutos = segundos / 60;
       int horas = minutos / 60;
       printf ("34706 segundos correspondem a %d\n horas",
    horas );
       printf ( "e %d \n minutos", minutos);
    }
```

Identificadores

- São nomes usados para se fazer referência a variáveis, funções, rótulos e vários outros objetos definidos pelo usuário.
  - O primeiro caracter deve ser uma letra ou um sublinhado.
- Os 32 primeiros caracteres de um identificador são significativos.
- É case sensitive, ou seja, as letras maiúsculas diferem das minúsculas.

```
int x; /* é diferente de int X;*/
```

Variáveis e Constantes

- Constante: valor fixo que não pode ser modificado pelo programa. Exemplo:
  - Valores inteiros: 123, 1, 1000, -23
  - Strings: “abcd”, “isto é uma string!”, “Av. São Carlos, 2350”
  - Reais: 123.45F, 3.1415e-10F
- const é utilizada para determinar constantes:
  - const char LETRA_B =‘B’;
- Variável: Podem ser modificadas durante a execução do programa

Variáveis

- Em um programa C estão associadas a **posições de memória** que armazenam informações.
- Toda variável deve estar associada a um identificador.
- Palavras-chave de C não podem ser utilizadas como identificador (evita ambiguidade)
  - Ex.: int, for, while, etc...
- C é case-sensitive:
  - contador ≠ Contador ≠ CONTADOR ≠ cOntaDor

Tipos de dados básicos

- Define a **quantidade de memória** que deve ser **reservada para uma variável** e como os bits devem ser interpretados.
- O *tipo* de uma variável define os valores que ela pode assumir e as operações que podem ser realizadas com ela.
- Ex:
  - variáveis tipo *int* recebem apenas valores inteiros.
  - variáveis tipo *float* armazenam apenas valores reais.

Tipos de dados básicos

- Os tipos de dados básicos, em C, são 5:
  - Caracter: **char** (‘a’, ‘1’, ‘+’, ‘$’, ...)
  - Inteiro: **int** (-1, 1, 0, ...)
  - Real: **float** (25.9, -2.8, ...)
  - Real de precisão dupla: **double** (25.9, -2.8, ...)
  - Sem valor: **void**
- Todos os outros tipos são derivados desses tipos.

Void?

- Retorno void
  - Nao retorna nada
- Tipo void: tipo nao especificado
  - Exemplo:

```
int *meu_numero = (int *) malloc(sizeof(int));
```

Modificadores de Tipos

- Modificadores alteram algumas características dos tipos básicos para adequá-los a necessidades específicas.
- Modificadores:
  - **signed**: indica número com sinal (inteiros e caracteres).
  - **unsigned**: número apenas positivo (inteiros e caracteres).
  - **long**: aumenta abrangência (inteiros e reais).
  - **short**: reduz a abrangência (inteiros).

Tipos fundamentais agrupados por funcionalidade

- **Tipos inteiros**: char, signed char, unsigned char, short, int, long, unsigned short, unsigned long.

- **Tipos de ponto flutuante**: float, double, long double.

Arrays ou Vetores

- Um vetor é uma coleção de **variáveis do mesmo tipo** referenciadas por um nome comum.
- Uma determinada variável do vetor é chamada de **elemento** do vetor.
- Os elementos de um vetor podem ser acessados, individualmente, por meio de índices

Arrays ou Vetores em C

- Os elementos de um vetor ocupam **posições contíguas** na memória.
- Um vetor é considerado uma matriz - ou array - unidimensional.
- Em C, vetores (matrizes também) e ponteiros são assuntos relacionados.

Arrays ou Vetores

- Forma geral da declaração:
  - **tipo A[***expressão***]**
    - **tipo** é um tipo válido em C.
    - A é um identificador.
    - *expressão* é qualquer expressão válida em C que retorne um **valor inteiro positivo**.
  - Exemplos

    float salario[100];

    int numeros[15];

    double distancia[a+b]; //com a+b >= 0

Inicialização Vetores

- float F[5] = {0.0, 1.0, 2.0, 3.0, 4.0};
- int A[100] = {1};
  - Vetor A = ???

Inicialização Vetores

- float F[5] = {0.0, 1.0, 2.0, 3.0, 4.0};
- int A[100] = {1};
  - Vetor A = [1 0 0 0 ... 0].
- int A[100] = {1, 2};
  - A[0] recebe 1, A[1] recebe 2 e o restante recebe 0.
- int A[] = {2, 3, 4};
  - Equivale a: int A[3] = {2, 3, 4};
- **A primeira posição de um vetor é a posição 0**;

Exemplo

int c[10], i;

Primeira posição = ?

Exemplo

int c[10], i;

Primeira posição = 0 → c[0]

Ultima posição = ?

Exemplo

int c[10], i;

Primeira posição = 0 → c[0]

Ultima posição = 9 → c[9]

Exemplo

- Implemente em C um programa que leia 100 números reais e imprima o desvio padrão.

Exercício

- Implemente em C um programa que leia 100 números reais e imprima o desvio padrão.

- Leitura da i-ésima posição do vetor

  scanf(“%f”, &v[i]);

  scanf(“%f”, v + i);

Strings

- Não existe um tipo String em C.
- Strings em C são uma array do tipo char que termina com ‘\0’.
- Para literais string, o próprio compilador coloca ‘\0’.

Exemplo de String

```
#include <stdio.h>

void main() {
    char re[4] = “aim";
    //char re[4] = {‘a’,‘i’,‘m’,‘\0’}
    printf ( "%s", re );
}
```

Declaração: duas formas

```c
#include <stdio.h>

int main(int, char **)
{
    char ola[] = "ola";

    printf(ola);

    return 0;
}
```

Em forma de array

```c
#include <stdio.h>

int main(int, char **)
{
    const char *ola = "ola";

    printf(ola);

    return 0;
}
```

Em forma de ponteiro

Leitura de uma String

- scanf: não lê espaços em branco
  - scanf(“%s”, stringName): não é necessário o operador &
- gets: lê espaços em branco

```
#include <stdio.h>
void main(){
    char re[80];
    printf ("Digite o seu nome: ");
    gets(re);
    printf ("Oi %s\n", re);
}
```

Como contar caracteres?

Contando caracteres

```c
#include <stdio.h>

int contaChar(const char *str)
{
    int i = 0;

    for(;str[i] != 0; ++i);

    return i;
}

int main(int, char **)
{
    char ola[] = "ola";

    printf("A string %s possui %d caracteres\n", ola, contaChar(ola));

    return 0;
}
```

Biblioteca <string.h>

- Biblioteca que contém as funções para mexer com strings
  - Ex: strlen → a função retorna um valor inteiro com o número de caracteres da String

Referência: www.cplusplus.com/reference/cstring

Exemplo

```
#include <stdio.h>
#include <string.h>
main() {
    char re[80];
    printf ("Digite a palavra: ");
    scanf ("%s", re);
    int size = strlen(re);
    printf ( "Esta palavra tem %d caracteres.\n", size );
}
```

Comparando strings (igualdade)

```c
#include <stdio.h>

int main(int, char **)
{
    char ola[] = "ola";
    char ola2[] = "ola";

    if(ola == ola2)
        printf("Iguais");
    else
       printf("Nao sao iguais");

    return 0;
}
```

Saída ?

Comparando strings (igualdade)

```c
#include <stdio.h>

int main(int, char **)
{
    char ola[] = "ola";
    char ola2[] = "ola";

    if(ola == ola2)
        printf("Iguais");
    else
       printf("Nao sao iguais");

    return 0;
}
```

Saída ?

Não são iguais

Por quê?

Comparando strings (igualdade)

```
int saoIguais(const char *s1, const char *s2)
{
    int i;
    for ( i = 0; s1[i] == s2[i]; ++i )
    {
        if( s1[i] == '\0' )
            return 1;
    }
    return 0;
}
```

Comparando strings (igualdade)

int strcmp (const char * str1, const char * str2);

- Compara as strings str1 e str2
- Retorna
  - zero, se são iguais
  - <0, se str1 < str2
  - >0, se str2 > str1

Comparando duas strings

strcmp é equivalente a código abaixo?

```
char *a, *b;
a = "abacate";
b = "uva";
if (a < b)
   printf( "%s vem antes de %s no dicionário", a, b);
else
   printf( "%s vem depois de %s no dicionário", a, b);
```

Comparando duas strings

strcmp é equivalente a código abaixo?

```
char *a, *b;
a = "abacate";
b = "uva";
if (a < b)
   printf( "%s vem antes de %s no dicionário", a, b);
else
   printf( "%s vem depois de %s no dicionário", a, b);
```

Comparação de ponteiros

Copiando strings

```
1  int main(int, char **)
2  {
3      char str1[] = "abc";
4      char str2[10];
5
6      str2 = str1;
7
8      return 0;
9  }
```

Copiando strings

```
1  int main(int, char **)
2  {
3      char str1[] = "abc";
4      char str2[10];
5
6      str2 = str1;
7
8      return 0;
9  }
```

Erro de compilação

Exercício

- Implementação da função de cópia

Copiando strings

```
void copia(char destino[], char origem[])
{
    int i;
    int size = strlen(origem);

    for ( i=0 ; i < size; i++)
        destino[i] = origem[i];

    destino[i] = '\0';
}
```

Para copiar o conteúdo de uma string em outra

- Usa se a função: “para” é a string onde vai se copiar a informação nova e “de” é a string antiga

  strcpy(para, de);

```
#include <stdio.h>
#include <string.h>

main() {
    char str[80];
    strcpy (str, "Alo");
    printf ("%s", str);
}
```

Funções de conversão

- De string para double
  - Exemplo: “51.2341” para 51.2341
- strtod
  - double strtod(const char *toConvert, char **endPtr)
    - toConvert: string a ser convertida
    - endPtr: ponteiro para a string restante após a conversão

strtod

Saída: 51.2
% are omitted

```c
int main()
{

    //String a ser convertida
    const char *string = "51.2% are omitted";

    double d;
    char *stringPtr;

    d = strtod ( string, &stringPtr);

    printf( "%f\n%s\n", d, stringPtr);

}
```

Outras funções de conversão

- strtoi(const char *str, char **endPtr, int base)
  - Converte str para um long int
  - str: string a ser convertida
  - endPtr: string restante não convertida
  - base: base da conversão

Outras funções de conversão

- strtoul(const char *str, char **endPtr, int base)
  - Converte str para um unsigned long int
  - str: string a ser convertida
  - endPtr: string restante não convertida
  - base: base da conversão

sprintf

- Funciona de forma análoga ao printf
  - Escreve em string e não na tela

- Exemplo:

```
//Cria a string str como “o número é 10”
int num = 10
sprintf ( str, “o numéro é%d”, num );
```

sscanf

- Análoga a função scanf
- Exemplo

```
char s[] = “312 3.14159”;
int x; double y;

sscanf( s, “%d%f”, &x, &y);
```

Expressões

- Em C, expressões são compostas por:
  - Operadores: +, -, %, ...
  - Constantes e variáveis.
- Precedência: ( )
- Exemplos

```
x;
14;
x + y;
(x + y)*z + w – v;
```

Expressões

- Expressões podem aparecer em diversos pontos de um programa:
  - comandos /* x = y; */
  - parâmetros de funções /* sqrt (x + y); */
  - condições de teste /* if (x == y) */

Expressões

- Expressões retornam um valor:

```
x = 5 + 4 /* retorna 9 */
```

  - esta expressão retorna 9 como resultado da expressão e atribui 9 a x

```
((x = 5 + 4) == 9) /* retorna true */
```

  - na expressão acima, além de atribuir 9 a *x*, o valor retornado é utilizado em uma comparação

- Expressões em C seguem, geralmente, as regras da álgebra.

Operadores Aritméticos

- Operadores unários
- Operadores binários

Operadores Unários

+ : mais unário (positivo)	/* + x; */
- : menos unário (negativo)	/* - x; */
! : NOT ou negação lógica	/* ! x; */
&: endereço	/* &x; */
*: conteúdo (ponteiros)	/* (*x); */
++: pré ou pós incremento	/* ++x ou x++ */
-- : pré ou pós decremento	/* -- x ou x -- */

Operador ++

- Incremento em variáveis
  - ++ pode ser usado de modo pré ou pós-fixado
  - Exemplo:

```
int x =1, y =2;
x++; /* equivale a x = x + 1*/
++y; /* equivale a y = y + 1*/
```

- Não pode ser aplicado a constantes nem a expressões.

Operador ++

- A instrução ++x
  1. Executa o incremento
  2. Depois retorna x
- A instrução x++
  1. Usa o valor de x
  2. Depois incrementa.

Exercício 4

Diga quais são os valores das variáveis y e x em cada momento de execução do seguinte programa

```
int main (void)  {
   int x = 10;  int y = 0;
   y = ++x;
   printf (“%d %d”, x, y);
   y = 0; x = 10;
   y = x++;
   printf (“%d %d”, x, y);
   return(0);
}
```

Exercício 4

Diga quais são os valores das variáveis y e x em cada momento de execução do seguinte programa

```
int main (void)  {
   int x = 10;  int y = 0;
   y = ++x;
   printf (“%d %d”, x, y);
   y = 0; x = 10;
   y = x++;
   printf (“%d %d”, x, y);
   return(0);
}
```

X = 10
Y = 0

Exercício 4

Diga quais são os valores das variáveis y e x em cada momento de execução do seguinte programa

```
int main (void)  {
   int x = 10;  int y = 0;
   y = ++x;
   printf (“%d %d”, x, y);
   y = 0; x = 10;
   y = x++;
   printf (“%d %d”, x, y);
   return(0);
}
```

X = 11
Y = 0

Exercício 4

Diga quais são os valores das variáveis y e x em cada momento de execução do seguinte programa

```
int main (void)  {
   int x = 10;  int y = 0;
   y = ++x;
   printf (“%d %d”, x, y);
   y = 0; x = 10;
   y = x++;
   printf (“%d %d”, x, y);
   return(0);
}
```

X = 11
Y = 11

Exercício 4

Diga quais são os valores das variáveis y e x em cada momento de execução do seguinte programa

```
int main (void)  {
   int x = 10;  int y = 0;
   y = ++x;
   printf (“%d %d”, x, y);
   y = 0; x = 10;
   y = x++;
   printf (“%d %d”, x, y);
   return(0);
}
```

X = 10
Y = 0

Exercício 4

Diga quais são os valores das variáveis y e x em cada momento de execução do seguinte programa

```
int main (void)  {
   int x = 10;  int y = 0;
   y = ++x;
   printf (“%d %d”, x, y);
   y = 0; x = 10;
   y = x++;
   printf (“%d %d”, x, y);
   return(0);
}
```

X = 10
Y = 10

Exercício 4

Diga quais são os valores das variáveis y e x em cada momento de execução do seguinte programa

```
int main (void)  {
   int x = 10;  int y = 0;
   y = ++x;
   printf (“%d %d”, x, y);
   y = 0; x = 10;
   y = x++;
   printf (“%d %d”, x, y);
   return(0);
}
```

X = 11
Y = 10

Operador --

- Decremento
  - O operador -- decrementa seu operando de uma unidade.
  - Funciona de modo similar ao operador ++.

Operadores binários

- São eles:
  - +: adição de dois números /* x + y */
  - - : subtração de dois números /* x - y */
  - * : multiplicação de dois números /* x * y */
  - / : quociente de dois números /* x / y */
  - %: resto da divisão inteira /* x % y */
    - Só aplicável a operandos inteiros.

Operadores de Atribuição

= : atribui	x = y;
+= : soma e atribui	x += y; <=> x = x + y;
-= : subtrai e atribui	x -= y; <=> x = x - y;
*= : multiplica e atribui	x *= y; <=> x = x * y;
/= : divide e atribui	x /= y; <=> x = x / y;
%= : divide e atribui resto	x %= y; <=> x =x%y;

Exercício

1- Diga o resultado das variáveis x, y e z depois da seguinte seqüência de operações:

```
int x, y, z;
x=y=10;
z=++x;
x=-x;
y++;
x=x+y-(z--);
```

a) x = 11, y = 11, z = 11
b) x = -11, y = 11, z = 10
c) x = -10, y = 11, z = 10
d) x = -10, y = 10, z = 10
e) Nenhuma das opções anteriores

Operadores Relacionais

- Aplicados a variáveis que obedeçam a uma relação de ordem, retornam 1 (true) ou 0 (false)

Operador	Relação
>	Maior do que
>=	Maior ou igual a
<	Menor do que
<=	Menor ou igual a
==	Igual a
!=	Diferente de

Operadores Lógicos

- Operam com valores lógicos e retornam um valor lógico verdadeiro (1) ou falso (0)

Operador	Função	Exemplo
&&	AND (E)	(c >='0' && c<='9')
||	OR (OU)	(a=='F' || b!=32)
!	NOT (NÃO)	(!var)

Exercício

Considerando as variáveis fornecidas, calcule o resultado das expressões.

int a = 5, b=4; float f = 2.0; char c =‘A’;

a) a++ + c * b

b) (((3*2.0)-b*10) && a) || (f / a) >= 3

c) c || 0 && 3+2 >= 2*3-1 && f != b || 3 > a

Exercícios (entrega – parte 1)

1. Escreva uma função que transforme uma string (que contenha apenas dígitos) em um número inteiro, para uma dada base.
   - int strtoi(char *string, int base)

2. Escreva uma função que receba duas strings, que as junte numa string só e imprima o tamanho e o conteúdo da string final.