Módulo 2 - Programando em C

Expressões e Identificadores em C

Este assunto de C é a base para qualquer tipo de código. Durante o processo de programação, você trabalhará com tipos diferentes de dados, variáveis, funções, e precisará informar ao compilador o que fazer naquela parte específica do código.

Como primeiro assunto das expressões, vamos entender as variáveis em C:

- Variáveis = guardar dados

Tipos de dados

As variáveis precisam de tamanhos diferentes para comportarem tamanhos diferentes. Existem 5 tipos de dados:

- `char`: caracter;
- `int`: número inteiro, 2 ou 4 bytes;
- `float`: número real com seis dígitos de precisão;
- `double`: número real com dez dígitos de precisão;
- `void`: vazio, significa que naquele ponto você espera não encontrar nada. Não entra no assunto de varáveis, mas é importante na construção de funções.

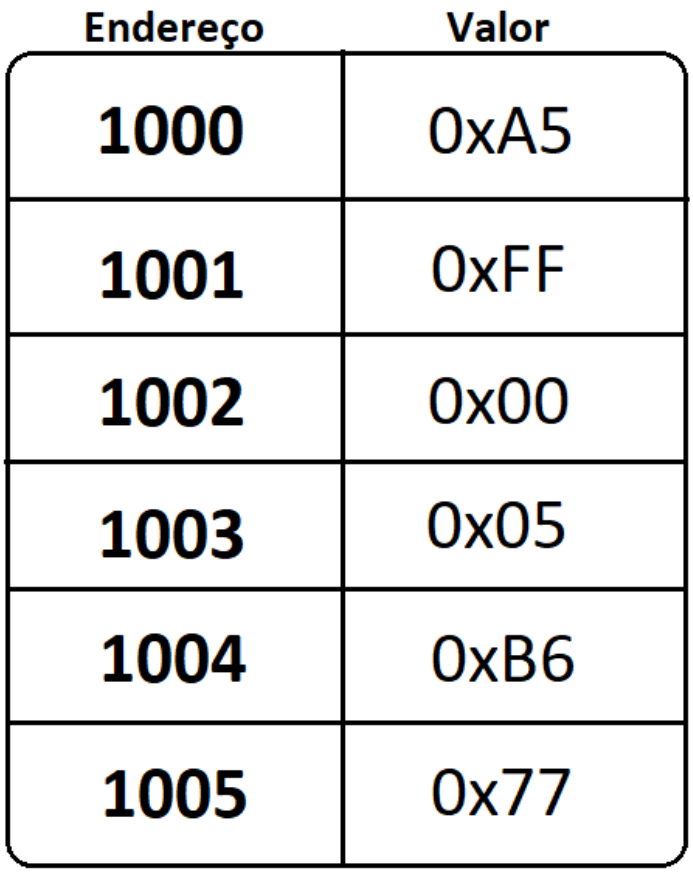
E 4 tipos de modificadores:

- `signed` → variável com sinal
- `unsigned` → variável sem sinal
- `long` → dobra o tamanho da variável
- `short` → diminui o tamanho da variável

Tipo	Bits	Inicio	Fim
char	8	-127	127

Tipo	Bits	Inicio	Fim
unsigned char	8	0	255
signed char	8	-127	127
int	16	-32767	32767
unsigned int	16	0	65535
signed int	16	-32767	32767
short int	16	-32767	32767
unsigned short int	16	0	65535
signed short int	16	-32767	32767
long int	32	-2147483647	2147483647
signed long int	32	-2147483647	2147483647
unsigned long int	32	0	4294967295
float	32	Seis díg e prec	
double	64	dez díg de prec	
long double	80	dez díg de prec	

Todo o computador possui uma memória, A memória se comporta como um armário cheio de espaços vazios onde é possível armazenar coisas. Nestes espaços, o programador tem a opção de colocar “etiquetas” determinando o que será armazenado lá, tornando este espaço reservado. Esta etiqueta, quando falamos em memória, faz o papel das **variáveis**. Ou seja, variáveis são instruções no código que dizem ao processador que você quer reservar um espaço na memória para armazenar algum valor. Cada uma dessas variáveis terão um endereço de memória, que represetam o valor que você salva nesta variável:



Veja alguns exemplos:

```
char letra; // Guarda um espaço para armazenar caracteres
unsigned int numero1, numero2; // Guarda dois espaços diferentes
                                                                //
float preco_total;  // Armazena valores com casas decimais, por
```

Existem algumas regras sobre como as variáveis devem ser declaradas, algumas são obrigatórias e outras apenas convenções:

- Sempre começar com letras, de preferência letra minúscula;
- Case sensitive (letras maiúsculas e minúsculas tem diferença)
- Não conter espaços em branco;
- Não conter acentos, ç, entre outros elementos que não estão presentes na língua inglesa

- Não utilizar caracteres especiais ou símbolos, exceto _ (underline)
- Tornar o nome da variável o mais legível possível

```
// Certo:
char count, test23, high_balance, fazAlgo
// Errado:
char 1count, test!, high…balance, faz Algo
```

As variáveis não podem ser declaradas com muitas abreviações, porém eles também não podem conter muitos caracteres, pois o compilador pode desconsiderar os ultimos caracteres. Além disso, existem palavras reservadas na linguagem C que já possuem funções específicas, por isso, não devem ser utilizadas como nome de variáveis. São elas:

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

Argumentos printf e PlaceHolders

Para imprimir o valor de qualquer variável no console, ela precisa antes ser transformada em um conjunto de caracteres. Isso é feito utilizando placeholders, eles devem ser escolhidos de acordo com o tipo da váriável declarada e passados dentro da função printf. Abaixo a tabela de placeholders e um exemplo de sua utilização.

%c	Caracter
%s	String de caracteres
%u	unsigned int
%d	signed int

%i	signed int
%o	octal
%x	Hexadecimal
%X	Semelhante ao anterior
%f	Float
%e	Float com notação científica (e minúsculo)
%E	Float com notação científica (E maiúsculo)
%g	Usa %e ou %f, o que for mais curto
%G	Usa %E ou %F, o que for mais curto
%n	Carrega o tamanho da string em uma var

Alguns outros comandos de string especiais do printf

\a	Toca um alerta sonoro padrão do sistema
\b	Backspace
\n	quebra de linha
\t	Tabulação horizontal (TAB)
\r	Retorna ao início da linha
\0	Caracter nulo
\v	Tabulação vertical

```
char caracter = 'a';
char saudacao[] = "Bom dia";
int total = 50;
float preco = 9.99;

printf("Caracter: %c - String: %s - Inteiro: %d - Decimal: %.2f"
printf("Inteiro: %d - Decimal: %.2f", total, preco);
```

```
printf("Oi, tudo bem? Tenho x anos e programo.\n");
printf("Valor inteiro: %d.\n", 10);
printf("Valor real: %f.\n", 3.141592);
```

```
printf("Valor real com apenas duas casas: %.2f.\n", 3.141592);
printf("Dado de caracter: %c\n", 'a');
printf("Dado de texto %s\n", "testanto");
```

Mostrando o endereço

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

int main(){
    int variavel = 10;

    printf("valor: %d\n", variavel);
    printf("endereco: %p", &variavel);
}
```

Utilizar o %g faz com que o compilador decida se é melhor utilizar %f ou %e, sendo decidido aquele com saída mais curta.

```
#include <stdio.h>

int main(void) {
    double f;
    f = 100;
    printf("%g\n", f); // 100
    f = 10000;
    printf("%g\n", f); // 10000
    f = 1000000;
    printf("%g\n", f); // 1e+06
    f = 10000000;
    printf("%g\n", f); // 1e+07
}
```

%n

```
#include <stdio.h>
void main(void){
        int count;

        printf("isso%n é um teste\n", &count); // isso é um test
        printf("%d", count);                   // 4
}
```

Scanf

Utilizando a função scanf do C é possível ler valores pelo console e guardá-los em variáveis. Para isso, a função recebe o endereço da variável junto com o placeholder adequado. A forma de passar o endereço da variável é utilizando o operador & junto do nome da variável. Veja o exemplo:

```
int preco;

printf("Digite o preco do produto: ");
scanf("%d", &preco);
```

```
#include <stdio.h>

int main(void){
    int int1;
    int int2;
    char caracter;
    char string[64];

    // O scanf permite a formatação do formato de entrada
    // Se você digitar dois inteiros separados por vírgula
    // o console irá entender a separação
    printf("Digite dois inteiros separados por ,: ");
    scanf("%d,%d", &int1, &int2);
    printf("Os inteiros foram: %d", int1, int2);
```

```
    // Digite uma string
    printf("\n\nDigite uma string: ");
    scanf("%s", string);
    printf("A string digitada foi: %s", string);

    // Isso não suporta uma string, somente um caracter
    printf("\n\nDigite um caracter: ");
    // Deve-se deixar um espaço antes do %c pois ainda existe ur
    scanf(" %c", &caracter);
    printf("Os caracter foi: %c", caracter);

    return 0;
}
```

Caso do buffer no scanf explicado com getchar(). O espaço gera um caracter \n que fica em buffer e é lido pelo proximo scanf. Por isso, se dá um espaço para este caracter ser ignorado.

```
#include <stdio.h>

int main(void){
    char * p;
    char str[256];

    p = str;

    // O ENTER do teclado cria um caracter \n
    while((*p++ = getchar()) != '\n');
    *p = '\0';
    printf("%s", str);
    return 0;
}
```

Ponteiros

Em linguagem C, um ponteiro é uma variável que armazena o endereço de memória de outra variável. Em outras palavras, um ponteiro aponta para a localização na memória onde um valor específico está armazenado. O uso de ponteiros é uma característica poderosa da linguagem C, mas também pode ser uma fonte de erros difíceis de depurar. Aqui estão os conceitos-chave relacionados aos ponteiros em C:

Os operadores são o & e o *. Eles antecedem alguma variável, podemos ler o operador & como “endereço de” e o * como “no endereço de”

Se este endereço contém o valor de uma outra variável, então o ponteiro aponta para aquela variável. Os operadores especiais para ponteiros são: * e &

```
// Variável count inicializada com o endereço 100
int count = 100;

// Variável m recebe o valor do endereço de count
int *m = &count;

// Variável q recebe o valor m, m é o endereço de count, então q
int q = *m;
printf("%d\n", m); // Coloca esse como ponteiro também
printf("%d", q);   // Colocar o operador & neste
```

```
// Era uma vez em algum trecho de código...

int x = 4; // Declaro uma variável de 16 bits com valor 4
                     // Inteiro chamado x tem como valor 4

int *pX = &x; // Declaro um pointer com valor o endereço de x
                            // Pointer inteiro chamado pX tem co
```

Os ponteiros tem semelhança com os operadores de multiplicação e de AND bit a bit, mas não possuem nenhuma relação com eles.

Um ponteiro só pode passar por operações de adição ou subtração. Cada vez que um ponteiro é incrementado, ele aponta para o próximo endereço de memória.

O ponteiro sobre incremento de acordo com seu tipo de dado. Se ele for declarado como um ponteiro de int, ele incrementará de 2 em 2 casas decimais.

Você também pode adicionar valores inteiros à ponteiros, e não apenas incrementá-los.

```
#include <stdio.h>

int main(void){
    int *p, i[10];
    p = i;         // p recebe o endereço da matriz
    p[5] = 100;    // Posso atribuir um valor utilizando índice
    *(p+5) = 100;  // Posso atrubuir um valor utilizando aritmét
}
```

```
char *s = "HI!"; // Endereço do primeiro caracter
printf("%d\n", *s);
printf("%d\n", *(s+1));
printf("%d\n", *(s+2));
printf("%d\n", *(s+3));
```

Os ponteiros em C são poderosos, mas com grande poder vem grande responsabilidade. Eles podem levar a erros de acesso indevido à memória (como violações de segmentação) se não forem usados com cuidado. Portanto, é importante garantir que você saiba o que está fazendo ao trabalhar com ponteiros e que evite acessar áreas de memória não alocadas ou liberadas.

Arrays

Em linguagem C, um array é uma estrutura de dados que permite armazenar um conjunto de elementos do mesmo tipo em uma única variável. Os arrays são uma parte fundamental da linguagem C e são amplamente usados para armazenar e manipular coleções de dados. Aqui estão os principais conceitos relacionados aos arrays em C:

```
// Sem inicializar, deve-se passar o tamanho dela para o prograr
char texto[8];
```

```
// Se ela for inicializada, o tamanho dela automaticamente é o t
char texto[] = "BOM DIA";

// Você pode inicializar um array quando o declara,
// fornecendo os valores iniciais entre chaves {}. Por exemplo:
int numeros[5] = {1, 2, 3, 4, 5};
```

Arrays se organizam sequencialmente na memória, cada elemento ocupa o espaço equivalente ao seu tipo declarado (char, int, float). Eles estão muito correlacionados a ponteiro, já que não existe um tipo de dado chamado array, eles se comportam como endereços dentro de seu código.

Os elementos de um array são acessados usando índices, que são números inteiros que indicam a posição do elemento no array. O primeiro elemento tem índice 0. Por exemplo:

```
int numeros[5] = {1, 2, 3, 4, 5};
int terceiro_elemento = numeros[2]; // Acessa o terceiro element


// Os ponteiros também recebem arrays.
char *s = "HI!";        // Endereço do primeiro caracter
printf("%d\n", &s[0]);
printf("%d\n", &s[1]);
printf("%d\n", &s[2]);
printf("%d\n", &s[3]);
```

Strings

O uso mais comum para arrays são as strings. Strings são um array de caracteres terminados por um caractere nulo (0 ou ‘\0’)

```
int main()
{
    // Quando não sei exatamente quantos caracteres terão de ent
    char name[20];
```

```
    printf("Enter your name: ");
    scanf("%s", name); // Sem o & mesmo
    printf("Name is %s: ", name);

    // Apenas 1 caractere de entrada
    char letter;
    printf("Enter a letter: ");
    scanf("%c", &letter); // Passa o endereço de letter
    printf("Letter is %c ", letter);

    // Caracteres com espaços
    char name[20];
   printf("Enter your name: ");
   fgets(name, 20, stdin); // Sem o & mesmo
   printf("Name is %s ", name);
}
```

```
#include <stdio.h>

int main () {
   char str1[20], str2[30];

   printf("Enter name: ");
   scanf("%19s", str1);

   printf("Enter your website name: ");
   scanf("%29s", str2);

   printf("Entered Name: %s\n", str1);
   printf("Entered Website:%s", str2);

   return(0);
}
```

String vs Caractere

Em linguagem C, uma "string" e um "caractere" são conceitos relacionados, mas eles têm diferenças significativas:

1. **Caractere (char)**:
   - Um "caractere" em C é uma unidade básica de texto que representa um único caractere alfanumérico, símbolo ou controle (como letras, números, pontuação, etc.).
   - Em C, o tipo de dado para representar um único caractere é `char`. Um caractere é armazenado em um único byte de memória.
   - Um caractere é definido entre aspas simples, como `'A'` ou `'1'`.

   Exemplo:

```
char letra = 'A';
```

2. **String**:

- Uma "string" é uma sequência de caracteres que representa texto. Ela é basicamente um array de caracteres (`char`) que pode conter zero ou mais caracteres.
- Em C, as strings são representadas como arrays de caracteres terminados por um caractere nulo `'\0'`. O caractere nulo indica o final da string e é usado para que as funções de manipulação de strings saibam onde a string termina.
- A biblioteca padrão C fornece várias funções para manipular strings, como `strlen`, `strcpy`, `strcat`, etc.

Exemplo:

```
char nome[] = "João"; // Isso é uma string
```

Diferenças importantes entre caracteres e strings em C:

- Um caractere é uma única unidade de texto, enquanto uma string é uma sequência de caracteres.

- Um caractere é armazenado em uma única posição de memória (um byte), enquanto uma string é armazenada como um array de caracteres em posições contíguas de memória.
- As strings são terminadas por um caractere nulo `'\0'`, que não faz parte do conteúdo real da string, mas indica o final da string.
- Caracteres são usados para representar um único caractere, enquanto strings são usadas para representar texto mais longo.

É importante notar que, em C, o tratamento de strings é um pouco diferente de linguagens de programação que têm tipos de dados de string integrados. Você geralmente precisa usar funções específicas da biblioteca C para trabalhar com strings, e a manipulação de strings requer cuidado para evitar erros de acesso indevido à memória.

Arrays de strings

O programa abaixo recebe várias strings e depois as retorna no console linha por linha.

```
#include <stdio.h>

#define MAX 16
#define LEN 16

int main(void){
    char text[MAX][LEN];
    char t=0, i=0, j=0;

    printf("Digite uma linha vazia para sair.\n\n");

    for(t; t<MAX; t++){
        printf("%d: ", t);
        gets(text[t]);
        if(!*text[t]) break;
    }

    for(i; i < t; i++){
```

```
        for(j=0; text[i][j]; j++) putchar(text[i][j]);
        putchar('\n');
    }
}
```

Operadores de atribuição e aritméticos

Os operadores são opções de cálculo que C oferece, veja quais são eles:

```
int a = 6;
int b = 5;
int x;

// + Adição
x = a + b;
// x = 11

// - Subtração
x = a - b;
// x = 1

// * Multiplicação
x = a * b;
// x = 30

// / Divisão
x = a / b;
// x = 0

// Caso a divisão não seja exata, o programa arredonda automatic
// Para ser possível trabalhar com números exatos, deve-se decla
// como float
float res;
res = 6 / 5
```

```
// Mesmo que c seja float, a divisão vem em int
int a = 10;
int b = 4;
float c;
c = a / b;
// c = 2

// Se na divisão uma var for float, aí o resultado vem como floa
int a = 10;
float b = 4.0f;
float c;
c = a / b
// c = 2.5

// Resto da divisão
// caso a divisão não resulte em um número inteiro, este operado
// essa parte que passou do número exato
res = 9 % 2
// res = 1, então 9 não é par

res = 8 % 2;
// res = 0, então 8 é par
```

Casting

Casting em linguagem C refere-se à conversão explícita de um valor de um tipo de dado para outro tipo de dado. Isso é feito usando operadores de casting para alterar o tipo de dado de uma expressão. O casting é necessário quando você deseja forçar uma conversão de tipo que o compilador não faria automaticamente, ou quando você deseja evitar avisos do compilador sobre perda de dados. Os operadores de casting em C são os seguintes:

```
// Declaro uma variável e já inicializo ela
unsigned int inteiro = 10000;

// Vou colocar este inteiro dentro deste char
```

```
unsigned char resultado;

// Com este método, apenas a parte baixa do inteiro vai para res
resultado = (unsigned char) inteiro;

// inteiro >> 0011 1110 1000
// resultado >> 1110 1000
```

Operadores de atribuição

```
char n;
n = 3
n = n + 1;
n = n - 2;
n = n / 2;
n += 1;
n -= 1;
// Incremento
n++;
n--;
```

Algumas bibliotecas apresentam operadores uteis, como a **<math.h>**. Na biblioteca **<math.h>** estão contidas diversas funções matemáticas básicas, com ela podemos trabalhar com funções trigonométricas, funções para cálculo de raiz quadrada, valor absoluto, entre outras. Esta listagem apresenta as seguintes funções conforme definidas pelo padrão C99. A inclusão das funções da biblioteca matemática são feitas com a opção de compilação **-lm**. Vale lembrar que todas as funções dessa biblioteca retornam valores do tipo double. Veremos, a seguir, todas elas.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>//necessária para usar as funções matemáticas
int main (void)
{
  double x = 9.75;
```

```
double arredonda_pbaixo = 0.0;
double arredonda_pcima = 0.0;
double raiz_quadrada = 0.0;
double potencia = 0;

double seno = 0;
double cosseno = 0;
double tangente = 0;

double logaritmo_natural = 0;
double logaritmo_xbase10 = 0;

printf("\n********* Utilizando a biblioteca math.h **********

printf("\nFuncoes de arredondamento\n\n");
printf("Valor original de x = %f\n",x);

arredonda_pbaixo = floor(x);
printf("Valor aproximado para baixo %f \n", arredonda_pbaixo

arredonda_pcima = ceil(x);
printf("Valor aproximado para cima %f \n", arredonda_pcima);

printf("\n-------------------------------------------------

printf("\nFuncoes de raiz e potenciacao \n\n");
printf("Valor original de x = %lf\n",x);
raiz_quadrada = sqrt(x);
printf("Valor da raiz quadrada %f \n",raiz_quadrada);

x = ceil(x); //arredondando o x para cima, x passa a valer 10

potencia = pow(x,2); //elevando o valor de x ao quadrado
printf("Valor de %.2lf ao quadrado %.2f \n",x,potencia);
```

```
  printf("\n-------------------------------------------------------

  printf("\nFuncoes trigonometricas\n\n");

  x = 0; //atribuindo zero em x para fazer os cálculos trigonome

  seno = sin(x);
  printf("Valor de seno de %.2f = %.2f \n",x,seno);

  cosseno = cos(x);
  printf("Valor de cosseno de %.2f = %.2f \n",x,cosseno);

  tangente = tan(x);
  printf("Valor de tangente de %.2f = %.2f \n\n",x,tangente);

  printf("\n-------------------------------------------------------

  printf("\nFuncoes logaritmicas\n\n");

  x = 2.718282;
  logaritmo_natural = log(x);
  printf("Logaritmo natural de x %.2f = %.2f \n",x,logaritmo_nat

  x = 10;
  logaritmo_xbase10 = log10(x);
  printf("Logaritmo de x na base 10 %.2f = %.2f \n",x,logaritmo_
  printf("\n-------------------------------------------------------

  getch();
  return(0);
}
```