ELEMENTOS BÁSICOS DA LINGUAGEM C: VARIÁVEIS, TIPOS, OPERADORES ARITMÉTICOS E ENTRADA/SAÍDA BÁSICA

NOTAS DE AULA 01

1. LINGUAGEM C

A origem da linguagem C é o resultado de um processo iniciado com uma antiga linguagem conhecida como BCPL. BCPL influenciou a criação de outra linguagem de nome B que, posteriormente, evoluiu para C. Ela foi Projetada em 1972, nos laboratórios da Bell/AT&T, por Brian Kernighan e Dennis Ritchie para um sistema operacional UNIX do computador PDP-11. O UNIX posteriormente saiu do laboratório para ser liberado para as universidades. Foi o suficiente para que o sucesso da linguagem atingisse proporções tais que, por volta de 1980, já existiam várias versões de compiladores C oferecidas por várias empresas, não sendo mais restritas apenas ao ambiente UNIX. Ainda hoje, a Linguagem C é amplamente utilizada em universidades e na construção de diversas aplicações computacionais. A simplicidade de sua implementação permitiu a extensão da linguagem e a criação de compiladores C para praticamente todas as plataformas de hardware e sistemas operacionais.

A linguagem C é uma linguagem altamente poderosa e provê todos os blocos de construção de programas fundamentais das linguagens de programação mais modernas: decisão (if-else); laços com testes de encerramento no início (while, for) ou no fim (do-while); e seleção de um dentre um conjunto de possíveis casos (switch). Talvez um dos maiores desafios dos programadores iniciantes na linguagem C seja o de aprender e utilizar ponteiros, corretamente. C oferece acesso a ponteiros e, com isso, a habilidade de **acessar e fazer aritmética com endereços de memória diretamente** (algumas linguagens modernas não permitem este tipo de habilidade, como JAVA).

Por outro lado, a linguagem C **não é uma linguagem rica na criação de tipos abstratos de dados** (como JAVA, Object Pascal, C++, etc.). Um tipo abstrato de dados constitui uma abstração de um objeto (ou coisa) que é mapeado do mundo real para o mundo computacional. Este mapeamento é materializado através de estruturas de representação de dados além de um conjunto de operações que é possível realizar com estes dados.

Ademais, C é uma linguagem apenas **compilada**. Isso significa que você pode traduzir seu programa para linguagem de máquina-alvo apenas uma vez e rodá-lo adequadamente quantas vezes desejar. O compilador C é capaz de traduzir o programa fonte inteiro, produzindo um outro programa equivalente, só que em linguagem diretamente executável pela máquina. A vantagem disso é que o compilador precisa traduzir uma instrução (ou um conjunto destas) apenas uma única vez, não importando quantas vezes ela será executada.

Entretanto, C não provê operadores para manipular diretamente objetos compostos, tais como cadeias de caracteres, nem facilidades de entrada e saída. Todos esses mecanismos devem ser fornecidos por funções explicitamente chamadas. Embora a falta de algumas dessas facilidades possa parecer uma deficiência grave (deve-se, por exemplo, chamar uma função para comparar

duas cadeias de caracteres), a manutenção da linguagem em termos modestos tem trazido benefícios reais. C é uma linguagem relativamente pequena e, no entanto, tornou-se altamente poderosa e eficiente.

A linguagem C é uma linguagem de propósito geral, sendo adequada à programação estruturada. Ela pode ser considerada uma linguagem de "nível-médio", porque, embora possua todas as características de estruturação de dados e programa em linguagens de alto nível, ela permite o acesso direto as estruturas de hardware de uma máquina, se isto for necessário. Ela é principalmente utilizada para escrever sistemas de software básico tais como o próprio sistema operacional, seus compiladores, analisadores léxicos, gerenciadores de bancos de dados, drivers de periféricos e dispositivos de estrada e saída, editores de texto, etc.

2. ELEMENTOS BÁSICOS DA LINGUAGEM C

A linguagem apresenta um conjunto de elementos básicos que compõem um programa escrito em C, tais como:

- Variáveis
- Tipos de dados primitivos e constantes numéricas, caracteres, textuais e valores lógicos.
- Comentários
- Expressões aritméticas, relacionais e lógicas
- Expressões (comandos) de atribuição de valor a variável

Nesta nota de aula apresentaremos alguns destes elementos.

2.1 Variáveis

Na matemática uma variável representa um elemento qualquer de um dado conjunto dentro de uma fórmula ou expressão matemática.

Na computação, entretanto, uma **variável** irá corresponder a uma **posição de memória** que poderá variar de conteúdo durante a execução do programa. Uma variável pode armazenar um **valor** nesta posição de memória. Este valor é o **conteúdo** da variável. Embora uma variável possa assumir diferentes valores, ela somente pode assumí-los um de cada vez, ou seja, ela só pode armazenar um valor por vez.

Toda variável deve ter um **identificador** ou **nome** que começa com uma letra (A,B,C,...Z ou a,b,c,...,z), podendo ser seguido de uma sequência de letras, dígitos ou caracter especial de sublinhado ('_'). Não são aceitos caracteres acentuados. Exemplos de identificadores: mediaNotas, Valor_maximo, TotalPorAluno, TAXA_DE_CAMBIO, entre outros. O número de caracteres permitido para identificadores depende da implementação de C utilizada, mas o padrão ANSI requer que compiladores C aceitem, pelo menos, identificadores contendo 31 caracteres. Normalmente, compiladores modernos dão liberdade para programadores sensatos escreverem identificadores do tamanho desejado.

Existem algumas palavras que já são utilizados pela própria linguagem C (por exemplo, while, for) ou por bibliotecas de rotinas (por exemplo, printf, abs, cos) e, portanto, não podem ser

utilizados como identificadores pelo programador. Estas palavras são conhecidas como **palavras reservadas** da linguagem.
Uma característica importante da linguagem C é que, diferentemente de algumas outras linguagens, ela faz distinção entre letras maiúsculas e minúsculas. Essa característica é chamada de *case-sensitive*. Isto significa, por exemplo, que duas variáveis com nomes: minhaVar e MinhaVar são diferentes.

2.2 Tipos de dados primitivos

Toda variável tem que ter um tipo de dado associado, que pode ser classificado genericamente como: inteiro, real, lógico, caracter e texto (*string*). Embora não seja obrigatório em todas as linguagens de programação, geralmente as variáveis têm que ser declaradas antes de ser usadas. A declaração serve para avisar ao computador que uma dada variável com um dado nome e tipo de dados deve ser criada. Na nossa disciplina estaremos assumindo que todas as variáveis tem que ser declaradas antes de ser usadas, ou seja, todas as variáveis devem ter um tipo associado no momento em que ela é criada.

As declarações de variáveis são expressões na forma:

<tipo-de-dados> <nome-da-variável> , ... , <nome-da-variável> ;

Onde **<tipo-de-dados>** e **<nome-da-variável>** são termos que podem ser substituídos por palavras que definem, respectivamente, o tipo de dados da variável e o nome da variável. Como pode ser visto na definição acima, também pode-se definir mais de uma variável por vez, bastando separar seus nomes por vírgulas. O caracter ponto-e-vírgula no fim da declaração é obrigatório.

Na linguagem C, o termo **<tipo-de-dados>** pode ser substituído por:

int: para indicar que será uma variável de tipo numérico inteiro;

float ou **double:** para indicar que será uma variável de tipo numérico real ou fracionário;

char: para indicar que será uma variável para **um** caracter;

As variáveis do tipo texto, comumente conhecidas pela expressão **string**, são definidas através do tipo caracter (char), informando-se qual o tamanho do *string* (ou cadeia) de caracteres. A definição é um pouco diferente da apresentada acima:

char <nome-da-varável> [<tamanho-do-string>] ;

onde os abre e fecha colchetes são obrigatórios e onde **<tamanho-do-string>** é uma constante numérica inteira.

Exemplos de declaração de variáveis:

int codigo, X1, matricula;

float a, b, nota;

char c1, c2;

char nome[50], cargo[50], descricao_do_produto[200];

Os tipos **float** e **double** também são conhecidos como tipos ponto flutuante. No caso do **double**, pode ser visto como um ponto flutuante com muito mais precisão. Existe também um tipo chamado **void**, que é o tipo vazio, ou um "tipo sem tipo". A aplicação deste "tipo" será visto posteriormente na disciplina.

Para cada um dos tipos de variáveis existem os modificadores de tipo. Os modificadores de tipo da linguagem C são quatro: **signed**, **unsigned**, **long** e **short**. Estes modificadores são palavras que alteram o tamanho do conjunto de valores que o tipo pode representar. Por exemplo, um modificador permite que possam ser armazenados números inteiros maiores ou pontos flutuantes com maior precisão, a exemplo do tipo **long double**. Outro modificador obriga que só números sem sinal possam ser armazenados pela variável. O resultado prático é que o conjunto praticamente dobra de tamanho. As tabelas abaixo mostra todos os tipos básicos definidos no padrão ANSI para linguagem C e também nestas tabelas está especificado o formato que deve ser utilizado para ler os tipos de dados com a função **scanf()**.

Tabela: Tipos primitivos para valores inteiros.

Tipo	Tamanho em Memória (bytes)	Intervalo de Valores	Código de Formatação
char	1	-128 a 127	%hhd (como número) ou %c (como caractere)
unsigned char	1	0 a 255	%hhd (como número)
short	2	-32768 a 32767	%hd
unsigned short	2	0 a 65535	%hu
int	4	-2147483648 a 2147483647	**%d**
unsigned int	4	0 a 4294967296	%ud
long	8	-9223372036854775807 a 9223372036854775808	%ld
unsigned long	8	0 a $1{,}844674407370955 \times 10^{19}$	%lu
long long	8	-9223372036854775807 a 9223372036854775808	%lld
unsigned long long	8	0 a $1{,}844674407370955 \times 10^{19}$	%llu

Tabela: Tipos primitivos para valores reais.

Tipo	Tamanho em Memória (bytes)	Intervalo de Valores	Código de Formatação
float	4	1.175494×10^{-38} a 3.402823×10^{38}	**%f** (notação normal) ou **%e** (not. científica)
double	8	$2.225074 \times 10^{-308}$ a 1.797693×10^{308}	%lf (notação normal) ou %le (not. científica)
long double	16	$3.362103 \times 10^{-4932}$ a $1.189731 \times 10^{4932}$	%Lf (notação normal) ou %Le (not. científica)

2.3 Comentários

Um comentário em um algoritmo ou programa é um trecho de texto que serve apenas para aclarar para um leitor humano, detalhes do que está sendo feito. Podem ser incluídos em qualquer parte do programa. Os comentários poderão ser colocados nos programas em C de duas formas:

/* entre barra-asterisco e asterisco-barra */

ou

// após duas barras e até o fim da linha.

no caso de comentários entre /* e */ eles podem se estender por várias linhas:

/*

Este e' um comentario

que se estende por

cinco linhas distintas

***/**

2.4 Operadores e Expressões Aritméticas

Denomina-se expressão aritmética aquela cujos operadores são aritméticos e cujos operandos são constantes e/ou variáveis numéricas. O conjunto básico de operadores é o tradicionalmente empregado na matemática, apenas que adaptado aos teclados e caracteres disponíveis nos computadores:

OPERADOR	OPERAÇÃO
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão inteira

Todos os operados acima são operadores binários, ou seja, necessitam de dois operandos. Os operadores + e – também podem ser utilizados como operadores unários. Explicaremos isso quando necessário.

A **precedência** é a usual da matemática: primeiro são feitas as operações multiplicativas: *, / e %, depois são feitas as aditivas: +. -. Operadores com mesma precedência são avaliados da esquerda para a direita. Assim, na expressão:

a + b * c /d

Executa-se primeiro a multiplicação, seguida da divisão, seguida da soma. Podemos utilizar parênteses para alterar a ordem de avaliação de uma expressão. Assim, se quisermos avaliar a soma primeiro, podemos escrever:

(a + b) * c /d

O uso de parênteses é livre, devendo ser utilizados quando se quer forçar uma precedência ou associatividade não usual ou quando se quer deixar a expressão mais clara e legível. Podem existir expressões entre parênteses aninhadas dentro de outras expressões entre parênteses. A única restrição é que os parêntese estejam “pareados”, ou seja, que nunca falte parênteses a esquerda ou a direita. Na prática o uso de expressões com parênteses é desejável e recomendável.

Exemplos:

Y = (2+3) * X

T = Y/(2 * (Z+44 % C))

valor = ((indice+1)/(total*2)) % valor_maximo

Vale lembrar que o operador de divisão / pode realizar uma divisão inteira ou de ponto flutuante. Mas o resultado vai depender dos operandos: se ambos são inteiros, então o resultado da divisão é inteira. Se apenas um dos operandos for do tipo ponto flutuante, C irá transformar todo o resultado para ponto flutuante. Veja exemplos:

```
int x = 5/2;          /* resulta em x = 2 */
double x = 5.0/2;     /* resulta em x = 2.5 */
```

O operador resto da divisão (%) devolve resto da divisão inteira entre os dois operandos. Exemplo:

```
x = 19 % 4;           /* resulta em x = 3 */
```

O operador módulo, %, não se aplica a valores reais (seus operandos devem ser do tipo inteiro). Veja um outro exemplo muito comum do uso deste operador: podemos citar o caso em que desejamos saber se o valor armazenado numa determinada variável inteira x é par ou ímpar. Para tanto, basta analisar o resultado da aplicação do operador %, aplicado à variável x e ao valor 2.

```
x % 2          /* se resultado for zero então número é par */
x % 2          /* se resultado for um então número é ímpar */
```

Outros operadores muito utilizados na linguagem C são os operadores **++** e **--** . Eles são operadores unários que devolvem, respectivamente, o incremento e o decremento de uma variável numérica. Podem ser usados de duas formas:

1) Antes da variável (pré-incremento ou pré-decremento)

Incrementa (ou decrementa) a variável antes de realizar a avaliação da expressão onde a mesma está localizada.

```
x = 10;
y = 20;
x = ++y + 2;
```

Resulta em x = 23 e y = 21, e seria equivalente a:

```
x = 10;
y = 20;
y = y + 1;
x = y + 2;
```

2) Após a variável (pós-incremento ou pós-decremento)

Incrementa (ou decrementa) a variável após realizar a avaliação da expressão onde a mesma está localizada.

```
x = 10;
y = 20;
x = y-- + 2;
```

Resulta em x = 22 e y = 19, e seria equivalente a:

```
x = 10;
y = 20;
x = y + 2;
y = y - 1;
```

Caso os operandos para ++ ou -- sejam únicos na expressão, é irrelevante se estes são utilizados antes ou após a variável. Dessa forma

```
++x;
```

é equivalente a

```
x++;
```

Exemplo:

```
int x = 10;
x++; /* podia ser ++x; */
printf("%d", x);
```

Veja, abaixo, os seguintes comandos equivalentes:

```
int a;
a = a + 1;
a += 1;
a++;
++a;
```

2.5 Operador de Atribuição

O uso deste operador é bem simples. Você deve ter percebido seu uso em códigos mostrados nas aulas anteriores. Este operador coloca na área de memória da variável localizada à esquerda do operador o resultado da expressão localizado à direita do operador.

Sintaxe: <**var**> = <**expressao**>

Este operador simplesmente copia o valor calculado da <**expressão**> para dentro da variável <**var**>. Por exemplo, supondo que x seja uma variável inteira, então a expressão:

x = 5

armazena um novo valor (5) dentro da variável x.

Exemplos:

```
int x = 10, y;
y = 10*(20 - x);
```

Importante: deve se ler a expressão:

x = 5

não como "x igual a 5" mas como "a variável x recebe 5". É muito importante recordar isto: o operador = não é de igualdade, mas de atribuição, qualquer valor antigo que estivesse armazenado em x seria irremediavelmente perdido e x passaria a armazenar o novo valor que é 5. O significado é completamente diferente no caso da expressão x==5 que testa se x é igual a 5 retornando verdadeiro se isto for verdade e falso caso contrário, mas sem alterar de nenhuma forma o valor armazenado em x. Veremos nas próximas aulas o uso deste operador (==).

O operador de atribuição simples é associativo pela direita e pode ser usado várias vezes numa mesma expressão. Sendo assim a expressão:

x = y = 10;

z = w = y = 10*(20 - x);

No primeiro caso, a ordem de avaliação é da direita para a esquerda. Assim, o computador avalia y = 10, armazenando 10 em y e, em seguida, armazena em x o valor produzido por y = 10, que é 10. Portanto, ambos, x e y, recebem o valor 10.

Além deste operador simples de atribuição a linguagem C oferece uma série de operadores “extras” de atribuição que servem para abreviar expressões de atribuição utilizadas com muita frequência:

Operador	Sentença abreviada	Sentença não-abreviada equivalente
+=	`m += n`	`m = m + n`
-=	`m -= n`	`m = m - n`
*=	`m *= n`	`m = m * n`
/=	`m /= n`	`m = m / n`
%=	`m %= n`	`m = m % n`

Assim, ao invés de escrever:

numero_de_dias = numero_de_dias + (i*2);

basta escrever:

numero_de_dias += i*2;

3. SAÍDA DE DADOS FORMATADA COM PRINTF

A função printf é, talvez, a função mais usada da biblioteca padrão de C por programadores. Isso porque ela é responsável por tratar a saída de valores (variáveis, constantes, resultado de expressões, etc.). A função printf é uma função nativa da biblioteca padrão da linguagem C, portanto, sempre que o programador precisar utilizar esta função em seu programa, ele deverá referenciar a seguinte biblioteca no inicio do programa:

#include <stdio.h>

Essa função tem por finalidade imprimir dados na tela. Isto é feito através da sintaxe geral:

printf(“expressão de controle”, lista de argumentos);

O primeiro parâmetro é geralmente uma cadeia de caracteres (em geral, delimitada com aspas) que especifica o formato de saída dos argumentos (constantes, variáveis e expressões) listados em sequência. Em outras palavras, na “expressão de controle” são inseridos todos os caracteres a serem exibidos na tela e/ou códigos de formatação, responsáveis por indicar o formato em que os argumentos devem ser impressos, como visto em sala de aula. Esses argumentos (constantes, variáveis e expressões) devem estar incluídos no espaço “lista de argumento” e caso contenha mais de um devem ser separados por vírgula em sequência. Para cada argumento que se queira imprimir, deve existir um especificador de formato correspondente nesta cadeia de caracteres inicial. Veja o exemplo a seguir:

printf(“O Valor de a = %d e b = %d”, a, b); // assumindo a e b inteiros

Note que os especificadores de formato variam com o tipo do valor e a precisão na qual queremos que os argumentos sejam impressos. Estes especificadores são precedidos pelo caractere % e podem ser, entre outros:

%c - caracter

%d - inteiro

%f - ponto flutuante (double ou float)

%s - string (cadeia de caracteres)

Exemplos:

```
char palavra[20];
printf(“Digite uma palavra:”);
scanf(“%s”, palavra);
printf(“A palavra digitada foi: %s”, palavra);
```

```
--- outro exemplo ---
char sexo = ‘M’;
float peso = 85.3;
int idade = 18;
printf(“Idade= %d Peso= %f Sexo= %c”, idade, peso, sexo);
```

Além dos especificadores de formato, podemos incluir textos no formato (estes serão mapeados diretamente para a saída). Assim, a saída é formada pela cadeia de caracteres do formato onde os especificadores são substituídos pelos valores correspondentes.

\n - nova linha

\t - tabulação (tab)

\" - aspas

\\ - barra

%% - caractere %

Podemos indicar, também no formato, a largura mínima do campo, o preenchimento de casas decimais, alinhamento à esquerda bem como o tipo de preenchimento de campos. Estas informações devem vir entre o sinal de % e o especificador de formato propriamente dito.

Exemplos:

printf("%.2f \n", 3456.78); // 3456.78

printf("%10.3f \n", 3456.78); // 3456.780

printf("%010.3f \n", 3456.78); // 003456.780

printf("%3.1f \n", 3456.78); // 3456.8

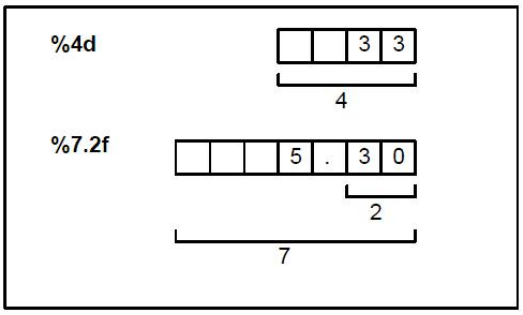
--- outro exemplo ---

printf("%d \n", 33); // 33

printf("%04d \n", 33); // 0033

printf("%4d \n", 33); // 33

Graficamente:



4. ENTRADA DE DADOS COM SCANF

A função scanf é utilizada para realizar entrada de dados de uso geral. Esta função é bastante parecida como printf (porém, no sentido inverso; ela lê números, caracteres, strings, etc.) Assim,

como o printf, é necessário incluir a biblioteca <stdio.h> no código fonte. A entrada de dados pode seguir os seguintes formatos (ver mais detalhes na seção 2.2):

%c - especifica um caracter

%d - especifica um número inteiro

%f - especifica um número ponto flutuante

%s - especifica uma string (cadeia de caracteres)

Exemplos:

```
float peso;
int idade;
char palavra[20];
printf(“Digite seu peso:”);
scanf(“%f”, &peso);
printf(“Digite sua idade:”);
scanf(“%d”, &idade);
printf(“Digite uma palavra:”);
scanf(“%s”, palavra);
```

Note que as variáveis numéricas, como peso e idade, necessitam ser precedidas com o **&**, entederemos mais a frente na disciplina o porquê dessa inclusão. È um erro comum esquecer o **&**, o que não gera um erro de compilação, mas gera um erro em tempo de execução.