Unidade II

A aplicação dos conceitos será feita utilizando a linguagem C++ padrão. Deve ficar claro que este não é um curso de linguagem C ou C++, portanto apenas os comandos e as técnicas que são aplicáveis à disciplina de *Linguagem e Técnicas de Programação* serão explicadas. Outras linguagens mais modernas, como Java, PHP ou mesmo o Interface Definition Language (IDL) do Common Object Request Broker Architecture (CORBA), usado em sistemas distribuídos, têm como base a linguagem C++. Portanto, esta será usada para mostrar como traduzir o pseudocódigo para uma linguagem de programação, e, na parte final da disciplina, as estruturas serão explicadas usando a própria linguagem C++.

2 INTRODUÇÃO À LINGUAGEM C

A linguagem C pertence a uma família de linguagens cujas características são: portabilidade, modularidade, compilação separada, recursos de baixo nível, geração de código eficiente, confiabilidade, regularidade, simplicidade e facilidade de uso.

Essa linguagem é adequada para a programação estruturada, tendo aplicação, principalmente, no desenvolvimento de sistemas operacionais, planilhas eletrônicas e gerenciadores de bancos de dados (HICKSON, 2005). Assim, o programa usado para processar o texto deste livro foi escrito em linguagem C.

A linguagem C é considerada de **médio nível**. Não é uma linguagem de máquina que só os sistemas operacionais reconhecem, as chamadas de **baixo nível**, como o Assembly, nem de **alto nível**, em que a própria linguagem fornece todos os recursos para o desenvolvimento do programa, como o Visual Basic. Pertence à classe dos programas compilados, que são escritos em texto e passam por traduções para adequar-se ao sistema operacional.

Segundo Cocian (2004), os pontos positivos que tornaram a linguagem C popular são:

- a portabilidade do compilador, pois existem compiladores para todos os sistemas operacionais;
- o conceito de bibliotecas padronizadas;
- a quantidade e a variedade de operadores poderosos;
- a sintaxe elegante;
- o fácil acesso ao *hardware*, quando necessário;
- a facilidade com que as aplicações podem ser otimizadas, tanto na codificação quanto na depuração, pelo uso de rotinas isoladas e encapsuladas.

2.1 Histórico

No final da década de 1960, uma linguagem chamada Basic Combined Programming Language (BCPL) – Linguagem de Programação Básica Combinada, desenvolvida por Martin Richards, da Universidade de Cambridge, em 1966 (POLLONI; PERES; FEDELI, 2009), serviu de base para a Linguagem B, desenvolvida por Ken Thompson para o computador PDP 7 e que funcionava no sistema operacional Unix. A linguagem B, em razão de problemas de velocidade e processamento, viria a ser corrigida na linguagem C (COCIAN, 2004), desenvolvida na Bell Laboratory por Dennis Ritchie, em 1972, e liberada para as universidades.

O fato de ser liberada para as universidades tornou essa linguagem popular, e havia diversos compiladores para os vários sistemas operacionais e computadores. Esse fato mostrou a necessidade de se padronizar a linguagem, o que aconteceu em 1983, quando a American National Standards Institute (ANSI) – correspondente à ABNT, no Brasil – estabeleceu esse padrão. Com o avanço do paradigma de programação orientada a objeto, foi desenvolvido o C++.

Observação

Como a linguagem C++ engloba totalmente a linguagem C e os recursos para esta disciplina são bem simples, a referência à linguagem C será válida para a linguagem C++.

Saiba mais

Caso haja a necessidade de consultar a normatização da linguagem C, basta consultar o ISO/IEC (2014). Apesar de extremamente técnico, pode ajudar na implantação de uma linguagem de programação em novos dispositivos físicos. Acesse:

ISO/IEC. JTC1/SC22/WG21: The C++ Standards Committee. *Open Standards*, 4 mar. 2014. Disponível em: <http://www.open-std.org/jtc1/sc22/wg21/>. Acesso em: 28 maio 2014.

2.2 Regras da linguagem

Como toda linguagem de programação, a linguagem C é muito rígida na sua sintaxe. Sintaxe são regras detalhadas para que um programa possa ser executado.

As regras estão relacionadas com os tópicos a seguir:

- tipos: definem as propriedades dos dados manipulados em um programa;

- declarações: definem o **identificador**, para alocar memória, definir conteúdo inicial e determinar funções;

- expressões: fórmulas que atribuem e alteram valores, bem como fazem chamada de funções de entrada e saída;

- funções: são os subprogramas em C, que, por sua vez, são chamados por outras funções executando tarefas específicas. Há funções básicas que estão definidas nas bibliotecas-padrão da linguagem, e outras que são desenvolvidas por terceiros, com rotinas mais específicas. As funções **printf()** e **scanf()**, por exemplo, que veremos ainda nesta unidade, as quais permitem, respectivamente, escrever na tela e ler os dados a partir do teclado, fazem parte da biblioteca básica. O programador também pode escrever as suas próprias funções, como veremos adiante.

A primeira função que o programa executa é o main(), portanto é obrigatória a sua declaração no programa principal.

Outras regras importantíssimas:

- A linguagem é *case sensitive*; isso quer dizer que as letras maiúsculas são diferentes das letras minúsculas, na identificação de comandos, variáveis e funções.

Quadro 14 – A linguagem C é *case sensitive*

Abacaxi ≠ abacaxi

- Os comandos são separados por ponto e vírgula (";"), que deve ser usado com muito cuidado, principalmente, antes de blocos de comandos.

Observação

O compilador ignora as quebras de linha, portanto os comandos escritos em linhas diferentes, mas separados apenas por esse recurso, para o compilador, estão na mesma linha.

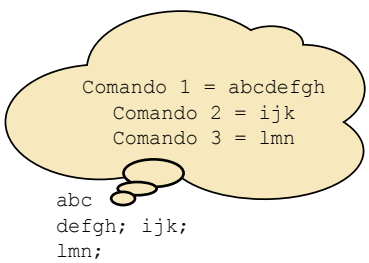


Figura 23 – Para o compilador, a separação dos comandos é feita pelo ';'

- A linguagem também permite o uso de comentários, que são colocados no texto entre "/*" e "*/", quando se quer escrever mais de uma linha. O uso do "//" também serve como comentário, e, neste caso, o compilador ignorará tudo o que estiver escrito a partir dele até o fim da linha.

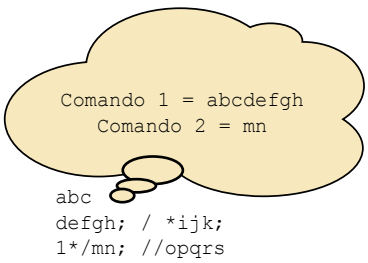


Figura 24 – O compilador ignora os comentários

- Pela definição padronizada pela ANSI, existem 32 palavras-chave (ou palavras reservadas) que formam a linguagem de programação C. Conforme o compilador, poderão existir mais palavras-chave, todas escritas em letras minúsculas. Nenhuma das palavras-chave poderá ser utilizada para dar nomes a variáveis e funções, pois isso gerará erro ao compilar.

Quadro 15 – Palavras reservadas pelo padrão ANSI

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*

Fonte: Hickson (2005, p. 28).

- Os identificadores são nomes usados para se fazer referência a variáveis, funções, rótulos e vários outros objetos definidos pelo programador. Como norma, conforme vimos no pseudocódigo, o primeiro caractere deve ser uma letra ou um sublinhado, e o restante podem ser números, letras ou sublinhado; apenas os 32 primeiros caracteres de um identificador são significativos. Como já foi dito, a linguagem C é *case sensitive*, ou seja, as letras maiúsculas diferem das minúsculas.

Lembrete

Sintaxe são regras detalhadas para que um programa possa ser executado.

2.2.1 Estrutura de um programa em C

Para programar em C partindo do zero, ou seja, de um texto totalmente vazio, é necessário apenas o bloco da função main(). Como há a necessidade de interações, é preciso colocar as bibliotecas no programa, e isso é feito em primeiro lugar.

Para termos uma ideia mais clara da estrutura, vamos elaborar o primeiro programa e compreender cada uma das linhas.

```
#include <stdio.h>
/*
Meu primeiro programa em C
*/
void main()
{
      printf("meu primeiro programa em C\n");
}
```

Ao executar o programa, teremos uma mensagem no console do DOS dizendo "Meu primeiro programa em C". Apesar de ser um programa muito simples, temos a estrutura mínima de um programa nessa linguagem. Vamos analisar cada parte dessa estrutura a seguir:

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

O comando *include* informa ao compilador que ele deve incluir o arquivo-cabeçalho *stdio.h*. Nesse arquivo-cabeçalho há uma série de funções de entrada e saída úteis. Sempre que quiser usar funções relativas a entrada e saída, verifique se o arquivo *stdio.h* as contém. A linguagem C fornece dezenas de arquivos-cabeçalhos com milhares de funções úteis. Podemos chamar os arquivos-cabeçalhos de bibliotecas.

Observação

O compilador, ao encontrar o comando #include, antes de iniciar o processo de compilação, adiciona a listagem da biblioteca à escrita pelo programador, formando uma só listagem, e só então o programa é compilado.

```
/*
Meu primeiro programa em C
*/
```

Lembrete

Conforme vimos anteriormente, o compilador ignora os comentários.

```
void main()
```

Define o nome da função *main*. Todo programa em C precisa ter essa função, pois esta é premissa de funcionamento e é a função que o programa executa ao ser inicializado. A palavra *void* indica que a função não retorna valor, conforme veremos mais adiante.

```
{
```

Indica o início do bloco do programa principal, o ponto em que o processamento começará.

```
    printf("meu primeiro programa em C\n");
```

Função que mostra na tela uma informação desejada. Está contida no arquivo-cabeçalho *stdio.h*, e é por isso que este foi incluso no programa. Corresponde ao **escreva** do pseudocódigo.

```
}
```

Fecha o bloco da função main(), encerrando o programa.

O programa corresponde ao pseudocódigo:

```
algoritmo "PrimeiroPrograma"
//
// Meu primeiro programa em C
// Seção de Declarações
inicio
    escreva("Meu primeiro programa em C")
fimalgoritmo
```

2.3 Tipos de informações, variáveis e constantes

Em pseudocódigo, vimos os tipos primitivos de dados: Inteiro, Real, Caractere e Lógico. Na linguagem C, os tipos primitivos sofreram modificações, visando aumentar a precisão ou adaptar a compatibilidade. A correspondência dos tipos está no Quadro 16.

Quadro 16 – Correspondência dos tipos primitivos de dados

Pseudocódigo	C
Inteiro	Int
Real	float ou Double
Caractere	Char
Lógico	bool (a partir do C++)

O nome da variável segue a nomenclatura das variáveis que vimos anteriormente nos pseudocódigos. O primeiro dígito deverá ser uma letra ou o caractere sublinhado, e o restante poderá ser letra, número ou o próprio sublinhado. O ANSI C não determina um limite no tamanho do nome, mas apenas os 32 primeiros dígitos é que são significativos.

A sintaxe da declaração de uma variável será:

```
<Tipo de dado>  <lista_de_nomes_de_variaveis> ;
```

Para declarar uma constante, acrescenta-se a palavra-chave *const* no início e atribui-se um valor a ela.

Sintaxe:

```
const <Tipo de dado>  <nome_da_variavel> = valor;
```

A tradução do pseudocódigo ficará assim:

```
algoritmo "declaracao"
const
   pi<-3.14159
var
   idade:inteiro
   saldo:real
   sexo:caracter
   matriculado:logico
inicio
// Seção de Comandos
fimalgoritmo
```



```
void main () {

      const pi = 3.14159;

      int idade;
      double saldo;
      char sexo;
      bool matriculado;

// Seção de Comandos
}
```

A linguagem permite também atribuir valor no momento da declaração da variável, assim como declarar muitas variáveis do mesmo tipo na mesma linha, ou realizar ambas as ações.

Exemplo:

```
int a;      /* declara uma variável do tipo int */
float a;    /* declara uma variável do tipo float */
```

```
int a = 5; /* cria e armazena o valor 5 em a */
int a,b,c; /* declara várias variáveis do tipo int */
int a,b=6,c;     /* declara várias variáveis do tipo int
                 atribuindo valor*/
```

Além dos tipos primitivos, temos modificações destes, e a faixa de números que são utilizáveis é determinada pelo número de bits de cada tipo modificado. Podemos ver essa precisão e a faixa numérica na Tabela 5.

Tabela 5 – Tipos primitivos com os modificadores

Tipo	Nº de bits	Intervalo	
		Valor inicial	Valor final
Char	8	-128	127
Unsigned char	8	0	255
Signed char	8	-128	127
Int	16	-32,768	32,767
Unsigned int	16	0	65,535
Signed int	16	-32,768	32,767
Short 1nt	16	-32,768	32,767
Unsigned short int	16	0	65,535
Signed short int	16	-32,768	32,767
Long int	32	-2.147.483.648	2.147.483.647
Signed long int	32	-2.147.483.648	2.147.483.647
Unsigned long int	32	0	4.294.967.295
Float	32	3,4E-38	3,4E+38
Double	64	1.7 E-308	1.7 E308
Long Double	80	3.4E-4932	3.4E+4932

Fonte: Hickson (2005, p. 30).

Conforme podemos ver, os tipos lógicos e as cadeias de caracteres não constam da tabela. Para esses tipos, a linguagem C tem um tratamento específico.

- Tipo lógico: a linguagem C trata como verdadeiro (*true*) o valor inteiro 1 e como falso (*false*) o valor inteiro 0; assim, a ausência do tipo lógico é compensada pelo correspondente numérico. Na versão C++, a linguagem passa a incorporar o tipo *bool*.

Exemplo de aplicação

Para contornar a ausência do tipo lógico na linguagem C, são utilizados artifícios como o do código a seguir.

```
enum boolean {
```

```
    true = 1, false = 0
};
// Permitindo a sua declaração como um tipo qualquer:
typedef  enum boolean  bool;
void main () {
    bool b = true;
    if (b) {
        b = false;
    }
}
```

- Tipos caractere e cadeia: a linguagem C não trata os caracteres como tipos literais, mas como números. Assim, o tipo *char*, como podemos ver na Tabela 5, usa 8 bits ou um byte, armazenando, portanto, um número cujo valor máximo é 255. Cada caractere tem um correspondente numérico, obedecendo a uma tabela. Normalmente, obedece à chamada Tabela ASCII (American Standard Code for Information Interchange – Código-Padrão Americano para o Intercâmbio de Informação). Podemos ver o padrão na Tabela 6.

Tabela 6 – Tabela ASCII

	0	1	2	3	4	5	6	7	8	9
30			branco	!	"	#	$	%	&	!
40	(	)	*	+	,	-	.	/	0	1
50	2	3	4	5	6	7	8	9		;
60	<	=	>	?	@	A	B	C	D	E
70	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y
90	Z	[	\	]	A	_	`	a	b	c
100	d	E	F	G	h	i	j	k	l	m
110	n	O	P	Q	r	S	t	u	v	w
120	x	Y	Z	{	|	}	~			

Fonte: UFPA (2013).

Conforme essa tabela, ao armazenarmos a letra 'A' em uma variável do tipo *char*, o conteúdo será o número 65.

O armazenamento de cadeias de caracteres é feito em um conjunto de variáveis homogêneas (vetores) do tipo *char*. A principal característica é que existe um dígito que indica o final da cadeia, utilizando, obrigatoriamente, o caractere nulo ('\0'). Assim, caso saibamos quantos dígitos uma variável do tipo cadeia irá armazenar, é necessário declarar mais um dígito, para o controle.

Exemplo de aplicação

Para armazenarmos quatro caracteres, devemos alocar cinco espaços.

```
char escola[5];
escola[0] ='U';
escola[1] ='n';
escola[2] ='i';
escola[3] ='p';
escola[4] ='\0'
```

Quando o tamanho for indefinido, será necessário colocar a indicação de fim de caractere.

```
char escola[] = {'U', 'n', 'i', 'p', '\0'};
```

- Constantes de barra invertida: o C possui algumas constantes para facilitar a exibição de resultados na tela, a manipulação de cadeias ou a impressão. São caracteres que podem ser usados como quaisquer outros. A lista completa dos códigos de barra invertida é dada no Quadro 17.

Quadro 17 – Códigos de barra invertida

Código	Significado
\b	Retrocesso (*back*)
\f	Alimentação de formulário (*form feed*)
\n	Nova linha (*new line*)
\r	Retorno de carro (*carriage return*)
\t	Tabulação horizontal (*tab*)
\"	Aspas
\'	Apóstrofo
\0	Nulo (0 em decimal)
\\	Barra invertida
\v	Tabulação vertical
\a	Sinal sonoro (*beep*)
\N	Constante octal (N é o valor da constante)
\xN	Constante hexadecimal (N é o valor da constante)

Fonte: Hickson (2005, p. 19).

2.4 Operadores

A correspondência entre o pseudocódigo e a linguagem C, dos operadores matemáticos, é mostrada na Tabela 7.

Tabela 7 – Correspondência entre operadores matemáticos

Pseudocódigo	C	Exemplo em pseudocódigo	Exemplo em C
<-	=	A<-7	A=7;
+	+	A<-3+5	A=3+5;
-	-	A<-3-5	A=3-5;
*	*	A<-3*5	A=3*5;
/	/	A<-3/5	A=3.0/5;
DIV	/ entre operandos inteiros	A<-3 div 5	A=3/5;
MOD	%	A<-7 mod 5	A=7 % 5;
**	pow() da biblioteca math.h	A<-4 ** 2	A=pow(4,2);

A linguagem C tem, ainda, algumas sintaxes específicas não presentes nos pseudocódigos.

- Atribuição múltipla: nesse caso, o número cinco é atribuído, inicialmente, à variável x, e, em seguida, o conteúdo da variável x (número cinco) é atribuído à variável y.

```
y = x = 5;
```

- Operadores de incremento e decremento: autoincremento ou decremento são operadores não convencionais que atuam sobre a própria variável, aumentando ou diminuindo uma unidade. A sintaxe é:

```
<variável>++
ou
++<variável>
```

Exemplo de aplicação

```
int a=5;
a++;
```

Corresponde a:

```
int a=5;
a=a+1;;
```

A variável a, ao executar a operação a++, incrementa em uma unidade o valor guardado. As operações a++ ou ++a têm o mesmo resultado, porém o mecanismo do processamento trabalha de forma diferente.

Vamos supor que a variável a armazene o valor 5.

```
x = a++;
```

Esse código atribui 5 a x, trabalhando da seguinte forma: o conteúdo de a é atribuído a x e depois é feito o incremento. Contudo, veja outro caso:

```
x = ++a;
```

Nesse exemplo, atribuímos 6 a x, pois é feito o incremento na variável a, e depois o resultado é atribuído a x.

No final, ambos passam a valer 6, pois seu valor foi incrementado em uma unidade, mas o valor final de x é diferente em cada um dos casos.

Para entender melhor, veja a comparação a seguir.

Tabela 8 – Exemplo de comparação

A expressão	Equivale a
b = a++	b = a a = a + 1
b = ++a	a = a + 1 b = a

Os operadores de incremento e decremento podem ser aplicados somente a variáveis; uma expressão do tipo x = (i + 1)++ é ilegal.

- Operadores de atribuição: os operadores de atribuição (Tabela 9) resultam na substituição do conteúdo do termo à esquerda da expressão. Com exceção da igualdade, todos os operadores resultam em formas similares de execução.

Tabela 9 – Operadores unários

Operador	Exemplo	Pseudocódigo	Ação
+=	a+=b	a<-a+b	a recebe o resultado da soma de a com b
-=	a-=b	a<-a-b	a recebe o resultado da subtração de a com b
*=	a*=b	a<-a*b	a recebe o resultado da multiplicação de a com b
/=	a/=b	a<-a/b	a recebe o resultado da divisão de a com b
%=	a%=b	a<-a%b	a recebe o resto da divisão de a com b

Operador relacional: todos os operadores de comparação (Tabela 10) dos pseudocódigos têm o seu equivalente na linguagem C.

Tabela 10 – Operadores relacionais em C

Pseudocódigo	C	Exemplo	Exemplo em C
=	==	A=B	A==B
>	>	A>B	A>B
<	<	A<B	A<B
>=	>=	A>=B	A>=B
<=	<=	A<=B	A<=B
<>	!=	A<>B	A!=B

Conforme a tabela anterior, a atenção deve ser dada aos operadores de igualdade e diferença que podem levar ao engano no processo de transcrição.

- Operador lógico: a transcrição dos operadores lógicos requer um cuidado especial, pois a sintaxe é significativamente diferente (Tabela 11).

Tabela 11 – Operadores lógicos em C

Pseudocódigo	C	Exemplo	Exemplo em C
E	&&	A **e** B	A && B
OU	||	A **ou** B	A||B
Não	!	**Não** A	!A

- Conversão de tipo (*cast*): o que acontece quando ocorre a divisão entre duas variáveis numéricas de tipos diferentes? Na linguagem C, ocorre uma avaliação automática da expressão, ajustando, antes, o resultado para o tipo de maior precisão. Assim, se dividirmos uma variável inteira com o número 5 por uma variável *double* com o valor 2,5, o programa converterá a primeira variável para *double* e realizará o cálculo, devolvendo um valor *double*.

  Alguns compiladores não permitem a atribuição de tipos de dados diferentes. Para evitar essas incompatibilidades, a linguagem tem um modelador de tipo. Esse modelador se chama *cast*. O uso do *cast* consiste em colocar o novo tipo entre parênteses.

Exemplo de aplicação

Vejamos as seguintes situações:

```
int a;
a = 3.5;
```

Alguns compiladores irão gerar um aviso de incompatibilidade de tipo. Para evitar esse aviso, deve-se usar o *cast*.

```
int a;
a = (int) 3.5;
```

A situação seguinte resulta em erro.

```
int a;
a = 3.5 % 2;
```

Nesse caso, a operação mod (%) requer dois operandos inteiros, e, no caso, 3,5 não é inteiro. Nessa situação, o programa não será compilado. Para corrigir, será necessário utilizar o *cast*.

```
int a;
a = (int) 3.5 % 2;
```

- Precedência na ordem de avaliação dos operadores: a precedência na ordem das operações obedece à mesma sequência vista no pseudocódigo, exceto pelos operadores característicos da linguagem C a seguir, que precedem a sequência já aprendida:

```
!  ++ -- - (cast)
```

Lembrete

Na linguagem C, a operação de atribuição é =, e a de relacionamento é ==. Assim como em algoritmo, os operadores são diferenciados: ← para atribuição e = para relacionamentos.

2.5 Entrada e saída de informações

A linguagem C não possui comandos de entrada e saída. Esses comandos são feitos por meio de funções. Para utilizar tais funções, existe uma biblioteca-padrão, o stdio.h, cujo nome é um mnemônico de Standard Input Output. A inclusão da biblioteca é feita no início do programa, por meio da instrução:

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

- Saída: a função que executa a visualização de informações pela tela é o printf, segundo um determinado formato. A sintaxe da função é:

```
printf("  formato  ",lista de variáveis/expressões)
```

A função é formada por duas partes. A primeira parte determina o formato da saída. Para cada valor contido em uma variável deve existir um especificador do formato de saída. As variáveis ficam listadas na segunda parte dos parâmetros da função. Os especificadores de formato determinam o

tipo e a precisão dos valores que queremos mostrar na tela. Esses especificadores são precedidos do caractere % (Quadro 18).

Quadro 18 – Especificadores de formato

Especificador	Característica
%c	Especifica um *char*
%d	Especifica um *int*
%u	Especifica um *unsigned int*
%f	Especifica um *double* (ou *float*)
%e	Especifica um *double* (ou *float*) no formato científico
%g	Especifica um *double* (ou *float*) no formato mais apropriado (%f ou %e)
%s	Especifica uma cadeia de caracteres

Fonte: Hickson (2005, p. 19).

Exemplo de aplicação

Considere este código:

```
#include <stdio.h>
void main(){
      printf ("%d %f\n", 34, 5.6);
}
```

Ao ser executado, o programa resultará na seguinte saída:

Figura 25 – Saída formatada

Além dos especificadores, podemos incluir textos no formato, que são apresentados na saída junto com os dados. Essa saída é formada pela cadeia de caracteres do formato, em que os especificadores são substituídos pelos valores contidos nas variáveis correspondentes.

Exemplo de aplicação

Ao executar:

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

```
      printf ("Inteiro = %d Real = %g\n", 34, 5.6);
}
```

Resultará em:

Figura 26 – Saída formatada

Também é possível colocar os constantes de barra invertida na definição do formato.

No formato é possível, ainda, determinar a precisão e o tamanho dos campos em que os valores são mostrados.

Exemplo de aplicação

Considere o programa a seguir:

```
#include <stdio.h>
void main(){
      printf ("|%4d|   |%7.2f|\n", 34, 5.6);
}
```

Esse programa apresenta o seguinte resultado:

Figura 27 –Saída formatada

Os caracteres "|" mostram o limite dos campos.

Alterando o programa:

```
#include <stdio.h>
void main(){
      printf ("|%10d|   |%10.3f|\n", 34, 5.6);
}
```

Teremos a seguinte saída:

Figura 28 – Saída formatada

Notamos que o espaço ocupado pelo 34 ficou mais largo, e o ocupado pelo 5.6, além de estar mais largo, tem um zero a mais.

Para entender:

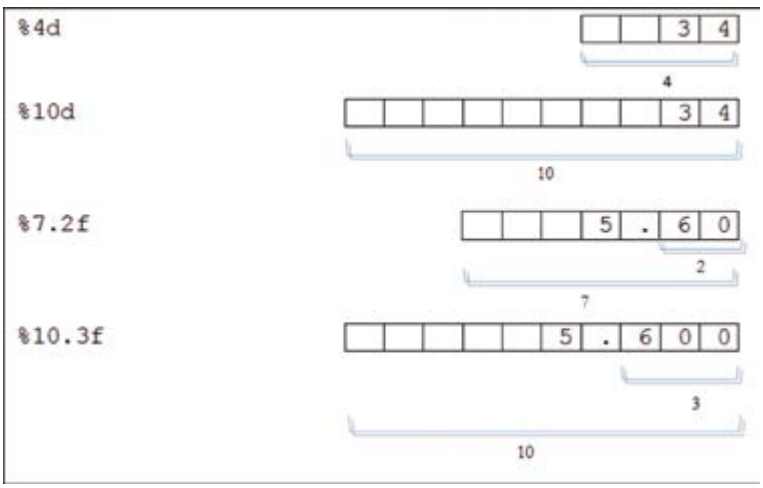


Figura 29 – Dimensionamento dos especificadores de formato

Nos inteiros, o número que fica entre o sinal de % e o "d" determina a quantidade de dígitos que o valor ocupará. Na formatação de ponto flutuante, o número antes do ponto determina o total de dígitos, e o número depois da vírgula, a quantidade de dígitos após a vírgula. Note que a vírgula conta um dígito.

Lembrete

O nome da biblioteca-padrão de entrada e saída é stdio.h (**st**an**d**art **i**nput/**o**utput). Uma falha muito comum é usar a grafia incorreta st**u**dio.h, causando erro de compilação.

Exemplo de aplicação

Traduzir o seguinte pseudocódigo:

```
algoritmo "media"
var
   nota1,nota2,media:real
inicio
      nota1<-8
      nota2<-10
      media<-(nota1+nota2)/2
      escreva("A media entre ",nota1," e ",nota2, " é ", media)
fimalgoritmo
```

Resolução:

```
#include <stdio.h>
void main(){
      double nota1,nota2,media;
      nota1=8;
      nota2=10;
      media=(nota1+nota2)/2;
      printf("A media entre %4.1f e %4.1f é %4.1f\n",nota1,nota2,media);
}
```

Resultando em:

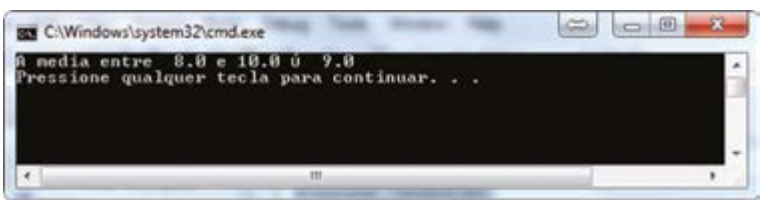


Figura 30 – Saída com os especificadores de formatação

- Entrada: os valores digitados no teclado são capturados pela função scanf, que também pertence à biblioteca stdio.h. Assim como o printf tem duas partes, uma que determina o formato de leitura e a segunda variável, que irá receber o valor digitado. A sintaxe da função é:

```
scanf("  formato  ",&variável)
```

Os especificadores de tipos do formato são similares aos utilizados na função printf. A função scanf utiliza especificadores diferentes para o tipo *float* e o tipo *double* (Quadro 19).

Quadro 19 – Relação dos especificadores de formato na entrada

Especificador	Característica
%c	Especifica um *char*
%d	Especifica um *int*
%u	Especifica um *unsigned int*
%f,%e,%g	Especifica um *float*
%lf, %le, %lg	Especifica um *double*
%s	Especifica uma cadeia de caracteres

Fonte: Hickson (2005, p. 14).

A diferença é que o formato deve ser seguido por uma lista de endereços de variáveis (na função printf, passamos os valores de constantes, variáveis e expressões). No tópico sobre ponteiros, esse assunto será tratado em detalhes. De maneira mais ampla, para o scanf ler um valor e atribuí-lo a uma variável, é necessário passar o endereço da variável que receberá o valor digitado. O operador & retorna o endereço de uma variável. Assim, para ler um inteiro, devemos ter:

```
int i;
scanf("%d",&i);
```

O formato também pode obrigar a digitar entrada dentro de um dado padrão. Por exemplo, para obrigar a entrada de dois números inteiros separados por dois-pontos, a sintaxe é:

```
scanf("%d:%d", &hora, &min);
```

Um espaço em branco dentro do formato faz que sejam ignorados eventuais brancos da entrada.

Exemplo de aplicação

Faça a tradução para a linguagem C e teste com as notas 4.0 e 7.0:

```
algoritmo "media"
var
   nota1,nota2,media:real
inicio
      escreva("entre com a nota1 e nota2:")
      leia(nota1,nota2)
      media<-(nota1+nota2)/2
      escreva("A media entre ",nota1," e ",nota2, " é ", media)
fimalgoritmo
```

Tradução:

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

```
void main(){
      double nota1,nota2,media;
      printf("Entre com nota1 nota2: ");
      scanf("%lf %lf",&nota1,&nota2);
      media=(nota1+nota2)/2;
      printf("A media entre %4.1f e %4.1f = %4.1f\n",nota1,nota2,media);
{
```

Executando:

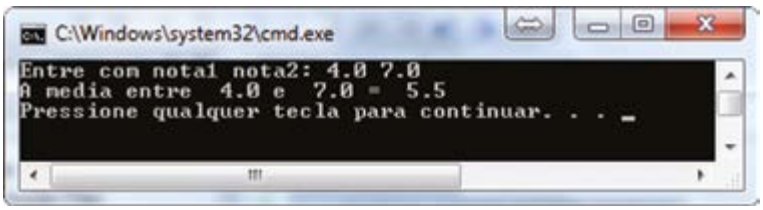


Figura 31 – Saída do programa com leitura em uma linha

Saiba mais

Para os usuários do sistema operacional Linux, a linguagem C++ pode ser encontrada no *site* dos desenvolvedores, o GCC:

<http://gcc.gnu.org/>.

Resumo

A Unidade II foi uma introdução à linguagem C, que será utilizada para executar os nossos programas pensados em Portugol. Vimos como o C se encaixa nas diversas linguagens de programação e um pequeno histórico.

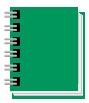
Como toda linguagem de programação, a linguagem C é muito rígida quanto à sua sintaxe. Aprendemos as regras que definem os tipos de dados, os nomes de variáveis, as expressões, as palavras reservadas que não devem ser utilizadas como nomes de variáveis, a separação de comandos e principalmente o fato de ser *case sensitive*, ou diferenciar a letra maiúscula da minúscula.

Vistas as regras, passamos a abordar o funcionamento do programa, a estrutura que comportará a programação e a biblioteca básica de entrada e saída. Vimos também os tipos principais, como são declarados e a tradução do pseudocódigo para a linguagem de programação.

Como a linguagem C precisa de uma biblioteca para facilitar a entrada e a saída, e como trata os caracteres de modo dúbio, por exemplo, número e letra, vimos os comandos '\', que permitem a formatação do valor, tanto na saída (printf) quanto na entrada (scanf).

Além das operações aritméticas tradicionais de soma, subtração, multiplicação, divisão e resto (+,-, *, /, mod), vimos os operadores unários, como os de incremento, decremento, acúmulo de soma e multiplicação, e redução de subtração e divisão (++, - -. += -=,/=.*=), bem como os operadores lógicos e, ou, não (&& , ||, ~).

Vimos também como podemos transformar um tipo em outro, o importante comando do *cast*.



Exercícios

Questão 1. As questões abaixo são referentes às aplicações das linguagens de programação C e C++. Podemos considerar que a linguagem C é adequada para a programação estruturada; é usada principalmente para o desenvolvimento de sistemas operacionais, planilhas eletrônicas e gerenciadores de bancos de dados (HICKSON, 2005). É uma linguagem considerada de "médio nível". As chamadas de "baixo nível" (que só os sistemas operacionais reconhecem, como o *Assembler*) não são linguagens de máquina e nem são de "alto nível" (linguagens que fornecem seus próprios recursos de desenvolvimento de programa). Com base nessas informações, escolha a alternativa com as afirmações consideradas corretas:

A) Alguns dos pontos positivos da linguagem C são: a escalabilidade do compilador, as bibliotecas padronizadas, a grande variedade de operadores, a facilidade na sintaxe, o acesso ao *hardware* e a otimização de aplicações.

B) A linguagem C não é rígida na sua sintaxe. A sintaxe são regras detalhadas para que um programa possa ser executado. Essas regras são elencadas como: tipos, declarações, expressões e funções.

C) A linguagem C não permite atribuir valores no momento da declaração da variável, tais como declarar várias variáveis do mesmo tipo na mesma linha.

D) Na linguagem C, o incremento ou decremento automático são operadores considerados convencionais que atuam sobre a própria variável, aumentando ou diminuindo uma unidade.

E) A linguagem C não possui comandos de entrada e saída; estes comandos são feitos através de funções. A utilização dessas funções está contida na biblioteca-padrão, o stdio.h.

Resposta correta: alternativa E.

Análise das alternativas

A) Alternativa incorreta.

Justificativa: os principais pontos positivos da linguagem C são a portabilidade do compilador, as bibliotecas padronizadas, a grande variedade de operadores, a facilidade na sintaxe, o acesso ao *hardware* e a otimização de aplicações.

B) Alternativa incorreta.

Justificativa: quanto à rigidez da linguagem C, ela é considerada bastante rígida em sua sintaxe. Sintaxe são regras detalhadas para que um programa possa ser executado. Essas regras são elencadas como: tipos, declarações, expressões e funções.

C) Alternativa incorreta.

Justificativa: a linguagem C permite atribuir valor no momento da declaração de suas variáveis, tais como declarar várias variáveis do mesmo tipo na mesma linha.

D) Alternativa incorreta.

Justificativa: na linguagem C, o incremento ou decremento automáticos são operadores considerados não convencionais que atuam sobre a própria variável, aumentando ou diminuindo uma unidade.

E) Alternativa correta.

Justificativa: a linguagem C não possui comandos de entrada e saída; estes comandos são feitos através de funções, da biblioteca padrão, o stdio.h.

Questão 2. Antes de montar um algoritmo precisamos ter uma noção do que iremos fazer, ou pelo menos uma estrutura mental de como realizar a tarefa, ou o problema proposto. O objetivo principal da lógica de programação é demonstrar técnicas para resolução de problemas, e consequentemente, automatização de tarefas. Analise as afirmativas abaixo:

I – Na linguagem computacional temos três tipos de algoritmos mais utilizados: descrição narrativa, fluxograma e pseudocódigo ou portugol.

II – A descrição narrativa consiste em entender o problema proposto e escrever sua solução através da linguagem específica, ou seja, a língua portuguesa.

III – No fluxograma, os passos para a resolução de um problema são feitos através de um símbolo padrão.

IV – O foco do pseudocódigo é a lógica e a rigidez sintática.

É correto apenas o que se afirma em:

A) I.

B) IV.

C) I e III.

D) II e III.

E) II e IV.

Resolução desta questão na plataforma.