Um programa é uma sequência de código organizada de tal forma que permita resolver um determinado problema. Um programa pode ser desenvolvido em módulos distintos e/ou em subprogramas. Dessa forma terá que existir um critério ou formato de escrita de programas bem definido que indique ao compilador, entre todo o conjunto de código escrito pelo programador, qual a instrução ou local onde irá começar a execução do mesmo.

A título de exemplo, um programa em PASCAL começa a execução na primeira instrução existente no único bloco BEGIN...END que não esteja associado quer a uma função quer a um procedimento. É absolutamente necessário que esse bloco BEGIN...END sejá único e se encontre depois de todas as funções e procedimentos.

No caso de C, existe uma função em que são colocadas todas as instruções que queremos que sejam executadas. Essa função chama-se main(), e todo o código a executar é colocado entre {}. Ao conjunto de código existente entre chaves chama-se Bloco.

Vamos então escrever o nosso primeiro programa em C.

Main(){

}

Nota: o nº de cada uma das linhas serve para facilitar a localização de código. Assim, deve-se apenas escrever o código respectivo.

Em seguida, podemos criar o arquivo executável correspondente e mandá-lo executar

Como pode se observar, este programa faz uma das coisas que mais gosto na vida, e que é – não fazer nada.

Observemos agora com maior atenção cada uma das linhas .

A primeira linha é composta pela palavra main, que é o local onde começam todos os programas em C.

Para indicar que se trata de uma função, a palavra main é seguida com parênteses\_main()\_, pois em C qualquer função tem que ser seguida por parênteses.

Os parênteses sem mais nada após o nome da função indicam que a função não recebe qualquer informação do mundo exterior.

É necessário não esquecer que C é Case Sensitive, isto é, faz diferenciação entre maiúsculas e minúsculas, não sendo portanto a mesma coisa escrever main(), Main(), MAIN() ou mAIN(). Todas as instruções de C são escritas com letra minúscula, e só se deve utilizar letras maiúsculas quando desejamos utilizar variáveis, mensagens ou funções escritas por nós.

Capítulo UM

O código executado pela função main() é escrito entre as chaves {}. No caso do nosso, programa, como não escrevemos qualquer instrução o programa simplesmente começa e acaba logo em seguida. E assim escrevemos o primeiro programa em C.

Nota: Caso seu compilador tenha apresentado um WARNING com uma mensagem semelhante a “Function should return a value” ou “main: no return value” é porque, em princípio, está-se usando um compilador de C++. Pode-se ignorar o Warning sem qualquer problema. Mas se o leitor quiser que desapareça coloque, antes da função main, a palavra void(que significa “nada”).

Nesse caso, o seu programa deverá ser escrito da seguinte forma:

void main(){

}

mais tarde você saberá o porquê.

É habitual que o primeiro programa escrito em qualquer linguagem seja para fazer a apresentação da linguagem ao mundo, Vamos então escrever o mais conhecido dos programas de C.

#include <stdio.h>

main(){

printf(“Hello Word”);

}

Depois de executado, esse programa proporciona a seguinte saída:

Hello World

Esse programa é um tudo semelhante ao anterior , com a exceção da existência de uma linha de código entre as chaves.

A linha 4: é a responsável pela apresentação da mensagem que queremos imprimir.

A mensagem que queremos imprimir é Hello World.

Sempre que queremos tratar conjuntos de caracteres temos que colocá-los entre aspas, para que sejam considerados como um todo “Hello World”.

Sendo C uma linguagem com muito poucas palavras reservadas, não é de surpreender que C não possua mecanismos de Entrada e Saída incorporados. Em vez disso, ela recorre à sua potente biblioteca de funções para fornecer esse tipo de serviço.

Uma das funções que permite a escrita na tela é a função **printf = print + f**ormatado. Tratando-se de uma função, tem que ser necessariamente escrita com parênteses – **printf()**.Dentro dos parênteses é feita a comunicação com a função. Nesse caso passamos a string (conjunto de caracteres) que queremos que seja escrita – **printf(“Hello World”)**;

Em C, cada instrução deve ser terminada com um ponto-e-vírgula(;), obtendo-se assim a linha nº4 do programa **printf(“Hello World”);**

Notar que o caractere aspas “ é um único caractere e não pode ser substituído pelo caractere aspas simples ‘ou por duas aspas simples’’

Como foi referido anteriormente, C não possui mecanismo de Entrada e Saída. Para resolver esse problema remos que recorrer a um conjunto de funções que existem em **Bibliotecas de Funções.** Isso implica que temos que adicionar à linguagem um conjunto de outras funcionalidades que, por defeito, ela não nos proporciona.

Para ter acesso a esse conjunto de funções teremos que incluir a sua definição no nosso programa.

A linha **#include<stdio.h>**  não é C, mas sim uma diretiva que indica o compilador (mais propriamente ao pré-processador) que deverá adicionar ao processo de compilação um arquivo existente em alguma parte no disco do seu computador, chamado **stdio.h** de forma que o compilador tenha acesso a um conjunto de informações sobre funções que virá a utilizar.

Esses arquivos têm sempre a extensão .h, pois não têm código, mas apenas os cabeçalhos(headers) das funções que representam. São por isso habitualmente designados por header files.

Desse modo, a linha **#include<stdio.h>**  significa “adiciona o arquivo **stdio.h** ao meu programa”. Exatamente nessa posição. Como não é uma instrução de C, não é seguida de;

O arquivo stdio.h permite o acesso a todas as funções de entrada e saída normais; **stdio** quer dizer **standard input/output**

include <stdio.h> Inclui também as funções de Entrada e Saída

main() O Programa começa aqui

{ Início do bloco de instruções

printf(“Hello World”); Escrever a string “Hello World” usando a função printf

} Fim do bloco de instruções (e fim de programa)

o mesmo programa poderia ser escrito através de um outro conjunto de instruções equivalentes:

#include <stdio.h>

main(){

printf(“Hello”);

printf(“ ”);

printf(“World”);

}

O resultado seria exatamente o mesmo. Aquilo que fizemos foi invocar a função printf várias, vom partes da string que originalmente queríamos escrever.

O compilador de C é particularmente liberal no que diz respeito á forma como escrevemos o código. Este pode ser escrito como casa programador muito bem entender. O programa anterior poderia ser escrito da seguinte forma:

#include <stdio.h>

main ( ) {

printf(

“Hello”

);

printf( “ “ )

; printf(“ world ” );}

sendo o resultado o mesmo, pois o que interessa não é a forma, mas o conteúdo. Espaços em branco são ignorados pelo compilador .

O programa original apresenta um pequeno problema. Depois de escrita a mensagem Hello World, o cursor fica colocado imediatamente após a palavra World e não na linha seguinte, como sera normal. A razão para que tal se verifique é que ninguém mandou o programa mudar de linha após escrever a mensagem.

Como é que C resolve esse problema?

Normalmente as linguagens de programação apresentam funções ou instruções distintas para a escrita na tela seguida ou não de uma nova linha. Por exemplo, PASCAL apresenta as instruções WRITE e WRITELN.

No caso de C a filosofia é outra . A função printf escreve todos os caracteres que lhe sejam enviados entre aspas. O problema que se depara a seguir é: como representar a mudança de linha ?

Tradicionalmente, a mudança de linha é denominada New Line, e em C é representada pelo símbolo \n.

dessa forma, deveremos alterar o nosso programa para

#include <stdio.h>

main(){

printf(“Hello World\n”);

}

depois de executado , esse novo programa proporciona a saída esperada, deixando o cursor na linha seguinte à string que escreveu na tela. Notar que nos interessava que a mudança de linha fosse realizada apenas depois de escrever a string Hello World. Para tal, o caractere indicador de mudança de linha deve ser escrito após Hello World, mas é um caráctere como outro qualquer, logo também faz parte da string.

O caractere especial New Line é representado por dois caracteres \n. Ora, n New Line é um caractere como outro qualquer, e por isso pode ser escrito tantas vezes quantas necessárias.

A saída gerada por printf(“Hello\n\nwor\nld\n”);

seria:

Hello

wor

ld

escreva um programa em C que tenha a seguinte saída :

C

is the greatest

Language

A solução é simples:

#include <stdio.h>

main(){

printf(“C\n”);

printf(“is the greatest\n”);

printf(“Language\n”);

}

ou usando um único printf()

#include<stdio.h>

main (){

printf(“C\n is the greatest \n Language\n”);

}

como se pode verificar, não é necessário deixar qualquer espaço em branco antes ou depois do símbolo\n.

Suponhamos agora que queríamos escrever um programa que apresentasse a seguinte saída:

**Hoje está um “LINDO” dia !!!**

A nossa primeira tentativa seria

#include <stdio.h>

main (){

printf(“Hoje está um “LINDO” dia !!!\n”);

}

1. Ora, acontece que esse programa vai originar um erro de compilação, uma vez que tudo aquilo que queremos escrever deverá estar entre aspas. Nesse caso, a string a ser escrita acaba junto á primeira seta e não junto á segunda seta, como deveria.

O compilador detecta que após a string a ser escrita (“Hoje está um”) existe algo mais, além do fecho da string, e emite um erro de sintaxe.

O problema que se apresenta é: se as aspas servem para delimitar uma string, como poderemos escrever uma frase (string) em que as aspas sejam um caractere não um delimitador?

A resolução desse problema é obtida através da colocação de uma \ antes das aspas que se pretende escrever, que devem ser tratadas como caracteres normais e não como delimitadores de strings:

#include<stdio.h>

main(){

printf(“Hoje está um \”LINDO\” dia !!!\n”);

}

dessa forma, obtemos a saída desejada.

**Caractere especial \**

O símbolo. \ é utilizado para retirar o significado especial que um caractere apresenta. No caso do caractere aspas (“), retira-lhe o significado de delimitador, passando a ser considerado simplesmente como o caractere aspas.

No caso do \n (e outros), serve para representar um caractere que, de outro modo, seria difícil ou q uase impossível de representar.

A lista completa dos caracteres que podem ter que ser representados, precedidos do caractere especial \, é:

\7 bell(sinal sonoro do computador)

\a bell(sinal sonoro do computador)

\b Backspace

\n New Line (mudança de linha)

\r Carriage return

\t Tabulação Horizontal

\v Tabulação Vertical

\\ Caractere \ (forma de representar o próprio caractere \)

\’ Caractere ‘ (aspas simples)

\” Caractere “ (aspas)

\​? Caracter (Ponto de interrogação)

%% Caracter %

Comentários \*/

No dia a dia, é habitual escrevemos notas sobre algo que não queremos. Quantos de nós já não foram para casa ou para o trabalho com um guardanapo de papel de alguma lanchonete ou restaurante escrito com algumas notas sobre o que fazer naquele dia ou a quem telefonar na parte da tarde?

Por vezes, quando se estuda por um livro em que algo não é apresentado de forma clara o melhor é usar um lápis e tomar algumas notas, para que da próxima vez não tenhamos tanto trabalho para perceber aquilo que lá estava apresentado.

São essas notas que nós tomamos no dia-a-dia que também nos podem ser particularmente úteis quando estamos escrevendo um programa. A essas notas damos habitualmente o nome de **Comentários.**

Os comentários não se destinam a ser interpretados pelo compilador, e o programa executável não terá qualquer sinal deles.

Um comentário em C é qualquer conjunto de caracteres compreendido entre os sinais de / *\* e \*/ .*

*U*ma vez que os comentários não têm qualquer interferência num programa, servem apenas para documentação de código. O seu objetivo é facilitar a vida do programador que tem que olhar para um determinado projeto em C, evitando que este tenha que perceber todo o código para saber o que determinado conjunto de instruções faz.

Exemplo de cabeçalho de um programa:

*/\*Comentários e Companhia \**/

*/\* AUTOR : Isaac Gonçalves \**/

*/\* DATA : 08*/09/2022 \*/

Um comentário pode estender-se por mais de uma linha, e até é habitual usá-lo dessa forma:

/\* Comentários

\* AUTOR : Isaac Gonçalves

\*DATA : 08/09/2022

\*/

Os asteriscos horizontais ou verticais não têm qualquer significado especial. Servem unicamoente para forçar o sentido estético de apresentação do comentário. O que realmente interessa é que o compilador ignora o que existe entre os símbolos /*\* e \**/.

Os comentários podem também ser colocados dentro de expressões ou de instruções, o que pode aumentar a complexidade de leitura do programa.

printf(“Hello World” /\* falta o \n \*/) *\* Notar o ponto e vírg. \**/;

A única situação em que os caracteres /\* \*/ não são ignorados pelo compilador acontece quando se encontram dentro de uma string.

printf(“Um comentário em C começa com /\* e termina \*/\n”);

Nesse caso os caracteres /\* \*/ perdem o seu significado especial e passam a ser tratados como caracteres normais dentro da string. A saída seria

Um comentário em C começa com /*\* e termina com \**/

mostrando que os símbolos /\* \*/ não são ignorados pelo compilador quando se encontrar dentro de uma string.

Os compiladores não permitem, em geral, a existência de comentários dentro de comentários:

/\* início do comentário

*/\* printf(“Ola\n”);Comentário interno \**/

\*/

porque seguindo a regra dos comentários em C, o comentário inicial terminava quando encontrasse o símbolo ***, que encerra o comentário. A extensão do comentário variaria então apenas entre a primeira ocorrência de \* e a prime****ira ocorrência de \*/, detectando então um final de comentário sem o correspondente início.*

*Alguns compiladores permitem, no entanto, que se faça um compilação usando a opção de “Nested comments”, verificando os comentários que estão dentro uns dos outros. Essa opção não é reconhecida pelo ANSI.*

***RESUMO:***

*Em C, um programa começa com a função* ***main().***

*O código a executar é colocado entre {}.*

*Um Bloco é formado por qualquer conjunto de instruções entre {}.*

*Cada instrução deve ser seguida de ponto-e-vírgula(;).*

*A disposição do código é arbitrária e depende das preferências de cada programador.*

*C faz distinção entre minúsculas e maiúsculas – diz se que é* ***Case sensitive.***

*As strings em C são delimitadas pelo caractere aspas “.*

*Para escrever uma mensagem na tela usa-se a função printf().*

*A função printf não faz parte da linguagem C. Pertence á sua biblioteca de funções.*

*Para termos acesso a esta e a outras funções de entrada/saída devemos incluir nos nossos programas o arquivo stdio.h através da diretiva ao pré-processador #include <stdio.h>.*

*As linhas começadas por # (#include, #define, etc.) não são C, mas sim diretivas ap pré-processador. Por isso não devem ser seguidas de ponto-e-vírgula.*

*A representação de caracteres especiais ou de caracteres que, de outra forma, seriam difíceis de representar faz-se através de um conjunto de dois ou mais caracteres, sendo em geral o primeiro caractere a barra invertida (\-backslash).*

*Os* ***comentários*** *são escritos em /\* \*/ e são simplesmente ignorados pelo compilador (a menos que se encontrem dentro de uma string).*

*Não podem existir comentários dentro de comentários.*

***EXERCÍCIOS***

*#include <stdio.h>*

*main(){*

*printf(“1 -\tClientes\n); para colocar uma tabulação*

*printf(“2 - \tFornecedores\n”);*

*printf(“3 - \tFaturas\n\n”); mais uma linha em branco*

*printf(“0 - \t Sair\n”);*

*Escreva um programa em C que apresente duas linhas com a string “Aqui vai um Apito”, ouvindo-se ao final de cada string um sinal sonoro.*

*#include <stdio.h>*

*main(){*

*printf(“Aqui vai um apito\a\n”); \a siinal sonoro*

*printf(“Aqui vai um apito \7\n”) \7 sinal sonoro*

*}*

*Escrevam um programa em C que indique qual o significado dos seguintes caracteres especiais: \n, \\, \t, %%.*

*#include <stdio.h>*

*main ()*

*printf(“Programa que representa os Caracteres Especiais\n\n”);*

*printf(“*[*\\n\t-\t*](../../../../../../t-/t)*<ENTER>\n”);*

*printf(\\\\\\\\t-\t\\\n”);*

*printf(*[*\\t\t-\t*](../../../../../../t-/t)*<TAB>\n);*

*printf(“%%%%\t-\t%%\n”);*

***Tipos de Dados Básicos***

***Objetivos***

*Tipos de dados em C char, int, float, e double*

*Variáveis declaração, regras para o nome de variáveis*

*Atribuição simples e encadeada*

*Inteiros, reais e caracteres características próprias e gerais*

*Operadores numéricos +, -, \*, /, e %*

*Leitura e escrita de variáveis e expressções funções printf, scanf, e getchar*

*Formatos de leitura e escrita %c, %d, etc*

*Caracteres versus inteiros*

*Casting promoção de expressões / variáveis a tipos diferentes*

*Alguns erros comuns*

***Introdução***

*Sempre que abrimos a nossa geladeira nos deparamos com um enorme variedade de recipientes para todo tipo de produtos: sólidos, liquidos, regulares, irregulares etc.*

*Cada um dos recipientes foi desenhado e moldado de forma a guardar um tipo de bem ou um produto bem definido.*

*Temos, assim, copos e garrafas para armazenar os líquidos, prateleiras com buracos de diâmetro certo para a colocação de ovos e ainda um enorme conjunto de recipientes em plástico para guardar o mais variado leque de produtos, sejam quadrados ou redondos, altos ou baixos, regulares ou não.*

*Se pensarmos um pouco, veremos que os recipientes referidos anteriormente não foram produzidos para guardar um determinado ou bem, mas antes uma gama de produtos cujas características e/ou formas são semelhantes. Por exemplo, um jarro, quando é produzido, não tem por objetivo armazenar água ou vinho, mas sim armazenar líquidos em geral.*

*Do mesmo modo, um recipiente quadrado com 20 cm pode armazenar fatias de presuntos (quadradas), fatias de mortadela(redondas) ou último pedaço do bolo de aniversário da Mimi, o qual, devido á tenra idade da aniversáriante, apresenta uma forma muito pouco definida, não tanto pelo formato ou consistência, mas antes pelos maus-tratos que sofreu por parte dos convidados, que o foram cortando e retalhando sem grande sentido estético ou geométrico.*

*Facilmente se verifica que armazenar ovos não é semelhante a armazenar água ou outro líquido qualquer. Da mesma forma, a maneira como se medem os ovos (1, 2,3,12,..) não é semelhante à forma como se medem os liquidos (0, 2; 1,5;…).Não faz qualquer sentido falar em 0.32 ovos ou em duas dúzias de água.*

*Ora , ninguém tem a necessidade de fazer um curso ou de ler livro para saber qual o tipo de recipiente que melhor se adapta a cada produto que se quer guardar na geladeira.*

*Na programação em C é necessário, no entanto, primeiro conhecer aquilo que queremos guardar e só então selecionar os melhores recipientes para a tarefa.*

*Os diversos formatos de recipientes para armazenar produtos na nossa geladeira corresponde em C aos* ***Tipos de dados básicos.***

*Estes são apenas quatro (char, int, float e double), serão apresentados detalhadamente em seguida.*

***Variáveis***

*Sempre que desejamos guardar um valor que, por qualquer razão, não seja fixo, devemos fazê-lo utilizando variáveis.*

*Nota:Uma variável é nada mais que um nome que damos a uma determinada posição de memória para conter um valor de um determinado tipo.*

*Como o seu próprio nome indica, o valor contido em variável pode variar ao longo da execução de um programa.*

*Uma variável deve ser sempre definida antes de ser usada. A definição de uma variável indica ao compilador qual o tipo de dado que fica atribuído ao nome que indicarmos para essa variável.*

*A definição de variáveis é feita utilizando a seguinte sintaxe:*

*tipo var 1[var2, ……,var5];*

*Existe ainda um outro tipo , o tipo* ***ponteiro*** *que pode ser considerado um tipo básico.*

***Exemplos***

*int i; i é uma variável do tipo inteiro*

*char ch1, novo\_char; ch1 e novo\_char são vars do tipo char*

*float pi, raio, perimetro;*

*double total, k123;*

*Nota: A declaração de variáveis tem que ser sempre realizada antes de sua utilização e antes de qualquer instrução.*

*main(){*

*Declaração de variáveis*

*Instrução1;*

*Instrução2;*

*}*

*As variáveis são sempre armazenadas em memória, e são uma forma simples de referenciar posições de memória. O tipo lhes está associado indica o número de bytes que serão utilizados para guardar um valor nessa variável.*

***Nomes de Variáveis***

*o nome que se vai atribuir a variáveis em C implica observar um número reduzido de regras:*

*O nome de uma variável pode ser constituído por letras do alfabeto(minúsculas ou maiúsculas), dígitos(0….9) e ainda pelo caractere undercore(\_).*

*O primeiro caractere não pode ser um dígito. Terá que ser uma letra ou caractere underscore. No entanto, é desaconselhável a utilização deste último como primeira letra identificadora de uma variável.*

*Maiúsculas e minúsculas representam caracteres, diferentes, logo variáveis distintas.*

*Uma variável não pode ser por nome uma palavra reservada da própria linguagem C. Assim, não podemos ter uma variável denominada float,if, ou for, uma vez que essas palavras são instruções ou tipos da própria linguagem.*

*Não é aconsellhável a utilização de caracteres acentuados (ã, õ á, é,) no nome das variáveis distintas.*

*O caractere underscore(\_) é habitualmente utilizado para fazer separação entre palavras que representam uma única variável. Ex Num\_Cliente, Id\_Fatura, Vou\_Continuar etc.*

*O número de caracteres que o nome de uma variável pode ter depende do copilador, mas é normal que sejam permitidos nomes de variáveis com até 32 caracteres (ou mais)*

***Nomes das Variáveis (cuidados a seguir)***

*o nome de uma variável deve ser descritivo daquilo que ela armazena.*

*Ex: xxx, yyy vs Id\_Ciente, num\_fatura*

*O nome de uma variável não deve ser todo escrito em maiúsculas, pois identificadores totalmente escritos em maiúsculas são tradicionalmente utilizados pelos programadores em C para referenciar constantes.*

*Caso o nome de uma variável use mais do que uma palavra, utilize o caractere underscore ou a diferença entre minúsculas e maiúsculas para as separar, facilitando assim aleitura.*

***Exemplos:***

*nomeprincipalcliente*

*nome\_principal­\_cliente*

*NomePrincipalCliente*

*Nome\_Principal\_Cliente*

*Não utilize o caractere underscore(\_) para iniciar o nome de uma variável*

***Atribuição***

*sempre que uma variável é declarada, estamos solicitando ao compilador para reservar espaço em memória para armazená-la. Esse espaço passará a ser referenciado através do nome da variável.*

*No caso do inteiro, o espaço em bytes que lhe é reservado varia com as arquiteturas em que é utilizado. Em microcomputadores o seu valor é normalmente 2 bytes, enquanto em máquinas maiores é habitualmente de 4 bytes.*

*Independente do número de bytes que ocupe, o nome da variável referenciada a totalidade do espaço ocupado pela variável.*

*Sempre que uma variável é definida, um conjunto de bytes fica associado a ela. Ora, esses bytes têm bits com o valor 1 e outros com valor 0 , constituindo um número qualquer. Dessa forma, quando uma variável é criada fica automaticamente com um valor que não é 0 nem 1, nem qualquer valor predefinido, mas sim um valor qualquer aleatório que resulta da disposição dos bits que se encontra nos bytes reservados para a representação dessa variável.*

*Nota: Quando uma variável é declarada fica sempre com um valor, o qual resulta do estado aleatório dos bits que a constituem.*

*Desse modo, uma variável poderá ser iniciada com um valor através de um operação de atribuição.*

*A atribuição de um valor só pode ser realizada para variáveis. Ao realizar uma atribuição o valor anterior presente na variável é eliminado, ficando nela o novo valor que lhe foi atribuído.*

*É essa capacidade que certos objetos têm de possuir diferentes valores que lhes confere o nome de* ***variáveis,*** *isto é, o seu conteúdo pode variar ao longo da execução de um programa.*

*Uma atribuição é realizada obedecendo à seguinte sintaxe:*

***variável = expressão;***

*A atribuição de valores em C é realizada através do sinal de = , sendo a variável a alterar* ***SEMPRE*** *colocada no lado esquerdo da atribuição, e o valor a atribuir no lado direito.*

***Exemplo*** *para colocar o valor 17 na variável num faz se:*

*int num; Declaração da Variável num*

*num = 17 num passa a ter o valor 17*

*Nota: uma variável pode ser automaticamente iniciada quando se faz a sua declaração.*

*As duas linhas anteriormente escritas poderiam ser agrupadas numa única linha.*

*int num = 17 num é declarada do tipo int e automaticamente iniciada com o valor 17*

***exemplo :*** *colocar na variável val o mesmo valor da variável num*

*val = num; val recebe o valor que está em num*

*Nota: em C é possível atribuir o mesmo valor a várias variáveis.*

***Exemplo:*** *colocar o valor 5 nas variáveis a, b, c, e d previamente declaradas:*

*a = 5;*

*b = 5;*

*c = 5;*

*d = 5;*

*ou, então, pode-se-ia fazer*

*a = b = c = d = 5;*

*Isso só é possível porque em C, sempre que se faz uma atribuição, o valor atribuído é devolvido (como se tratasse de uma função), podendo ser utilizado por outras expressões ou variáveis.*

*Vamos , então, ver como funciona o exemplo anterior em detalhe.*

*Suponhamos que as quatro variáveis eram iniciadas com valores distintos:*

*a = 1;*

*b = 2;*

*c = 3;*

*d = 4;*

*Qual é o valor das variáveis a, b, c e d se, em seguida, fosse executada a seguinte instrução:*

*a = b = c= d = 5;*

*Duas respostas são normalmente apresentadas por que olha pra essa instrução pela primeira vez:*

*1 todas as variáveis ficam o valor 5.*

*2 A variável a fica com o valor de b , b fica com o de c, c fica com o de d e apenas o d fica com o valor 5.*

*aparentemente , a resposta 2 parece mais consistente com aquilo que sabemos de programação, pois as atribuições seriam realizadas da esquerda para a direita, seguindo o sentido normal de execução das instruções.*

*No entanto é a resposta 1 que está correta. Por quê?*

*A razão é muito simples, e tem muito a ver com as características da linguagem C. Quando são escritas várias atribuições consecutivas, estas são realizadas não da esquerda para a direita , mas sim da direita para a esquerda.*

*Vamos então verificar como é realizada essa instrução.*

*1 Como se trata de atribuições, estas são realizadas da direita para a esquerda.*

*2 A primeira a ser realizada é d = 5.*

*3 Como foi referido anteriormente, o valor atribuído a d é devolvido como resultado da atribuição:*

*4 O valor devolvido (5) é então atribuido a c .*

*5 como é uma atribuição, devolve o valor atribuído (5) que é colocado em b e em a pelo mesmo processo.*

*6 A atribuição do valor 5 à variável a devolve também um valor, mas que não é aproveitado.*

***Inteiros int***

*As variáveis declaradas do tipo inteiro são utilizadas para armazenar valores que pertencem ao conjunto dos números naturais(sem parte fracionária) positivos e negativos. Ex: 2, -345, +115, 0.*

*como já foi dito antes, a definição de uma variável num do tipo inteiro é realizada da seguinte instrução:*

*int num; qual é o valor com que fica ??*

**Operações sobre inteiros**

Uma vez que estamos falando de números inteiro, é possível realizar um conjunto de operações sobre eles, cujos resultado é sempre um valor inteiro.

+ soma

- subtração

\* Multiplicação

/ divisão inteira

% resto da divisão (módulo)

Se em relaçãoà soma, à multiplicação não haverá muito a dizer, o mesmo já não se aplica aos operadores / e %.

Nota: qualquer operação entre inteiros retorna um inteiro.

Assim, da divisão entre 21 e 4 não irá resultar 5,25, como se poderia, pensar, uma vez que o resultado uma operação entre os dois inteiros (21 e 4) tem sempre como resultado um inteiro.

O quociente da divisão é obtido pelo operador divisão /, e o resto da divisão é devolvido pelo operador Módulo %.

Vamos agora aprender como se pode escrever inteiros na tela.

Observe com atenção o seguinte programa:

#include<stdio.h>

main(){

int num = 123;

printf(“O valor de num = %d e o valor seguinte = %d\n”, num, num + 1);

}

No programa é feita a declaração da variável num tipo int, que é automaticamente iniciada com valor 123. Como foi referido antes, as variáveis têm, obrigatoriamente, que ser declarada antes de serem utilizadas e antes de qualquer instrução (como o printf).

Em seguida é invocada a função printf com um conjunto algo estranho de parâmetros. Vamos então compreender aquilo que é passado á função printf.

Queremos escrever na tela a seguinte string:

**O valor de num = 123 e o valor seguinte = 124**

está é a string que deveria ser passada ao printf:

“O valor de num = 123 e o valor seguinte = 124\n”

Acontece, no entanto, que o valor de num está guardado numa variável, e não podemos colocar a variável num dentre da string printf, uma vez que o printf iria escrever a string “num” em vez do valor que estaria guardado na variável.

Assim , aquilo que nós queremos escrever é:

O valor de num = <inteiro> e o valor seguinte = <inteiro>

em que <inteiro> representa o valor inteiro que está guardado numa variável, constante ou é o resultado de qualquer expressão.

Ora, sempre que quisermos escrever um inteiro dentro de um printf devemos substituir o valor desse inteiro por um formato de escrita lembrar que printf quer dizer print + format) que, naquele preciso local, representará o inteiro a ser escrito.

Nota: O formato de escrita de um inteiro na função printf é %d.

Vamos então colocar o símbolo %d no local onde queremos escrever os inteiros:

O valor de num = %d e o valor seguinte = %d\n

Falta apenas indicar ao printf quais os valores que terá que colocar nos locais assinalados por %d.

printf(“ O valor de num = %d e o valor seguinte = %d\n”, num, num + 1);

Para tal, finalizamos a string que queremos escrever e colocamos por ordem as variáveis ou valores que irão ser substituídos em cada %d, separados por vírgula.

Nesse caso, o primeiro %d será substituído pelo resultado da soma de num + 1.

Obtém-se, assim, a saída esperada:

O valor de num = 123 e o valor seguinte = 124

Da mesma forma que existe a função printf para a escrita de valores, existe a função correspondente para a leitura de valores – a função scanf.

#include<stdio.h>

main(){

int num;

printf(“Introduza um Nº : ”);

scanf(“%d”, &num);

printf(“O número introduzido foi %d\n”, num);

}

A função scanf(leitura formatada) funciona de forma semelhante á função printf. Uma vez que ela foi implementada para a leitura de valores, a string inicial deve conter apenas o formato das variáveis que queremos ler.

Depois de especificados os formatos de leitura na string, devem ser colocados todas as variáveis correspondente pela ordem em que ocorrem os formatos, precedidas de um &.

Nota: para ler qualquer variável do tipo int, char, float ou double utilizando a função scanf é preciso preceder cada variável de um &. caso não se faça isso, a execução do programa poderá ter resultados inesperados.

No caso do programa anterior, queremos ler um valor para uma variável. Para tal usamos a função:

scanf()

O primeiro parâmetro dessa função é uma string com os formatos de leitura. Como só queremos ler uma variável, haverá apenas um formato de leitura. Sendo a variável que queremos ler do tipo inteiro, o formato de leitura será %d.

scanf(“%d”)

Em seguida, temos que indicar qual variável que irá receber o valor inteiro a ser lido. Essa variável, como é do tipo inteiro, tem que levar um & antes do seu nome.

Scanf(“%d”, &num)

e assim se obtém a linha 8 , que permite realizar a leitura de um inteiro e armazená-lo numa variável.

O inteiro, depois de lido, é guardado na variável num e, em seguida, o seu valor é escrito na tela através da função printf.

Nota: a string enviada para a função scanf não deve conter outros caracteres que não sejam os caracteres indicadores de formatos. Um erro comum é terminar a string com \n, o que está completamente errado e faz com que a função printf não termine a leitura dos valores assim que estes são introduzidos.

O que faz o seguinte programa?

#include <stdio.h>

main(){

int n1, n2;

printf(“Introduza dois números: “);

scanf(“%d%d”, &n1,&n2);

printf(“O resultado de %d + %d = %d\n”, n1, n2, n1 + n2);

}

Nesse caso são declaradas duas varáveis n1 e n2 do tipo int.

são pedidos dois valores;

São lidos dois inteiros e colocados em n1 e n2

**inteiros e variações**

como foi mencionado anteriormente, o tamanho em bytes de um inteiro varia de arquitetura para arquitetura , sendo os valores mais habituai de 2 ou 4 bytes.

É importante saner qual a dimensão de um inteiro quando se desenvolv um aplicação, caso contári corre-se o risco de tentar armazenar um valor numa variável inteira com um n de bytes insuficiente. Para saber qual a dimensão de um inteiro (ou de qualquer tipo de variável), o C disponibiliza um operador denominado sizeof, cuja sintaxe é semelhantes á utilizada para invocar uma funça