Apostila Algoritmo para Programa C

O mais curto programa em C que é possível escrever é o seguinte:

main()   
}   
{

Todo o programa em C deverá conter uma e só uma função main().   
As chavetas { e } agrupam instruções, sendo equivalentes às palavras chave inicio e fim do Algoritmo.   
É possível colocar comentários em qualquer posição de um programa em C. Um comentário é qualquer texto delimitado pelos caracteres /\* e \*/. Por exemplo:

/\* O meu primeiro programa em C \*/   
main()   
{   
/\* Outro comentário \*/   
}

Os comentários não podem ser imbricados. Assim, o exemplo seguinte:

/\* O meu primeiro programa em C \*/   
main()   
{   
/\* Comentário /\* Mais um comentário \*/ \*/   
/\* Ilegal \*/   
}

… é ilegal.

Um programa mínimo mais útil

O seguinte programa produz uma saída, escrevendo algum texto no vídeo:

main()   
{   
printf("FATEC!\n");   
exit(0);   
}

printf e exit são funções que existem na biblioteca standard da linguagem C.

O pré-processador

Alguma informação mais básica agora.   
O pré-processador atua apenas ao nível do código fonte, modificando-o. Trabalha apenas com texto. Algumas das suas funções são:

1. remover os comentários de um programa;
2. interpretar directivas especiais a si dirigidas, que começam pelo carácter #.

Por exemplo:

1. #include - insere o conteúdo de um arquivo de texto no arquivo corrente. Esses arquivos são usualmente designados por cabeçalhos (*header files*) e têm a extensão .h:
2. #include <math.h> - Insere o conteúdo do arquivo math.h com a declaração das funções matemáticas da biblioteca standard.
3. #include <stdio.h> - Idem para as funções standard de entrada/saída.
4. #define - define um nome simbólico cujas ocorrências no arquivo serão substituídas por outro nome ou constante:
5. #define MAX\_ARRAY\_SIZE 100 - substitui todas as ocorrências de MAX\_ARRAY\_SIZE por 100.

A estrutura de um programa em C

A estrutura genérica de um programa em C é a que se apresenta a seguir, podendo alguns dos elementos não existir:

1. Comandos do pré-processador
2. Definições de tipos
3. Protótipos de funções - declaração dos tipos de retorno e dos tipos dos parâmetros das funções
4. Variáveis globais
5. Funções

Deverá existir sempre uma função main().

As funções têm a seguinte estrutura:

tipo nome\_da\_funcao(parâmetros)   
{   
variáveis locais

instruções em C   
}

Assim, o programa:

void main(void)   
{   
printf("Eu gosto do C\n");   
}

contém apenas uma função (a função main(), que é obrigatória), que não retorna nada (void) e que não tem parâmetros (outra vez void). Como instrução da função temos apenas a chamada a printf(), uma função da biblioteca standard que escreve no vídeo. Neste caso escreve uma cadeia de caracteres (*string*). A combinação \n no fim da string indica uma mudança de linha (o carácter *new line* ou *line feed*). Notar que no final de cada instrução existe sempre um terminador - ;

Se a chamada a printf() fosse: printf(".\n.1\n..2\n...3\n"); o que apareceria escrito no vídeo seria:

.   
.1   
..2   
...3

Variáveis

O C tem pré-definidos os seguintes tipos de dados simples:

|  |  |  |  |
| --- | --- | --- | --- |
| **Tipo de dados** | **Tamanho (bytes)** | **Limite inferior** | **Limite superior** |
| char | 1 | -128 | 127 |
| unsigned char | 1 | 0 | 255 |
| short int | 2 | -32768 | 32767 |
| unsigned short int | 2 | 0 | 65535 |
| int | 4 | -231 | +231 - 1 |
| long int | 4 | -231 | +231 - 1 |
| float | 4 | -3.2×10±38 | +3.2×10±38 |
| double | 8 | -1.7×10±308 | +1.7×10±308 |

Geralmente nos sistemas UNIX os tipos int e long int são equivalentes (inteiros de 32 bits). No entanto noutros sistemas é possível que o tipo int seja equivalente a um short int (inteiro de 16 bits). É necessário consultar a documentação do compilador para o sistema em questão.   
O C não tem um tipo booleano pré-definido, no entanto, poderá usar-se um char (ou melhor um unsigned char) ou um int para o efeito.   
O prefixo unsigned pode também ser usado com os tipos int e long int.

Para declarar variáveis em C, de um dado tipo, usa-se a seguinte regra:

tipo\_das\_variáveis lista\_de\_variáveis ;

Por exemplo:

int i, j, k;   
float x, y, z;   
char ch;

Definição de variáveis globais

As variáveis globais, visíveis em todas as funções de um programa, declaram-se fora e antes de todas as funções (só são visíveis a partir do local da daclaração). Por exemplo:

short number, sum;   
int bignumber, bigsum;   
char letter;

void main(void)   
{

}

É também possível inicializar as variáveis globais no momento da declaração. Usa-se para isso o operador de atribuição = . Por exemplo:

float sum = 0.0;   
int bigsum = 0;   
char ch = 'A';

void main(void)   
{

}

Poderíamos ter feito a mesma coisa de outra forma (mas mais ineficiente):

float sum;   
int bigsum;   
char ch;

void main(void)   
{   
sum = 0.0;   
bigsum = 0;   
ch = 'A';   
}

O C também permite múltiplas atribuições (colocar o mesmo valor em várias variáveis ao mesmo tempo):

a = b = c = d = 3;

Entrada e saída do valor de variáveis

As funções da biblioteca standard printf() e scanf() permitem escrever no vídeo e ler do teclado, respectivamente, o valor de variáveis. Estas funções têm como primeiro parâmetro uma *string* especificando o formato e a ordem das variáveis a escrever ou a ler.

Seguem-se como parâmetros as próprias variáveis pela ordem especificada. Na *string* de formatação indica-se o local e o tipo de um valor de variável através do carácter % seguido de uma letra indicadora do tipo. Alguns dos tipos suportados são:

1. %c - char
2. %d - int's
3. %f - float's

Um exemplo: printf("Os valores das três variáveis são: %c, %d, %f\n", ch, i, x);

A modificação de formatos, pode ocorrer para especificar largura e numero de casas decimais. Assim o modificador é colocado entre o sinal % e o código do formato. Se tivermos %10f informa que o campo terá 10 posições incluindo a parte inteira o ponto e a parte decimal. Se tivermos %12.3f informa que terá 12 posições no total com 3 casas decimais.

#include <stdio.h>   
void main(void)   
{   
double item;   
item = 10.12304;   
printf("%f\n", item);   
printf("%5.2f\n", item);   
}

Produz o resultado:

10.123040   
10.12

**Notas**: As *strings* em C definem-se entre aspas ". . .", os caracteres simples aparecem entre plicas '.'; o texto normal da *string* de formatação aparece tal e qual no vídeo, os valores das variáveis aparecem nos locais indicados pelo carácter %; seguem-se as próprias variáveis que deverão aparecer pela mesma ordem e com os tipos indicados na *string* de formatação.

Operações aritméticas

O C suporta as quatro operações aritméticas standard nas suas expressões que são representadas pelos símbolos habituais (+, -, \*, /). Além destas operações aritméticas são suportadas outras.

A atribuição é representada no C pelo operador =. (No C a atribuição é considerada uma operação aritmética). Exemplos: i = 4; ch = 'y';

Outros operadores aritméticos são os operadores de incremento e decremento, representados respectivamente por ++ e --. Em geral geram código mais eficiente do que a soma ou subtracção de 1 unidade. Assim a atribuição x = x + 1; pode ser substituída simplesmente por x++; Os operadores ++ e -- podem ser pósfixos (colocam-se após a variável a incrementar ou decrementar) ou podem ser préfixos (colocando-se antes da variável a incrementar ou decrementar. Quando o operador é pósfixo a expressão x++ tem como valor o conteúdo de x antes deste ser incrementado, enquanto que com o operador préfixo a expressão ++x representa já o valor de x depois de incrementado (em ambos os casos x é incrementado, o que muda é o valor que a combinação x++ ou ++x representa como conjunto). Vejamos o exemplo:

int x, y, w;

void main(void)   
{   
x = ((++z) - (w--)) % 100;   
}

Este programa seria equivalente a:

int x, y, w;

void main(void)   
{   
z = z + 1;   
x = (z - w) % 100;   
w = w - 1;   
}

Um outro operador aritmético do C é o operador módulo - %. Este operador é equivalente ao operador mod e tem como resultado o resto da divisão inteira. Só pode ser utilizado com valores inteiros. (Ver os exemplos anteriores).

O operador de divisão /, pode ser usado com inteiros e reais. No entando, se ambos os operandos forem inteiros o resultado é a divisão inteira. Assim a atribuição x = 3 / 2; coloca em x o valor 1, mesmo que x seja float ou double. Se quisermos colocar em x o resultado correcto da divisão real (se x for float ou double) deveremos escrever x = 3.0 / 2; ou x = 3 / 2.0; ou melhor ainda x = 3.0 / 2.0; (sem conversões de um inteiro para real).

O C suporta também alguns modos expeditos de escrever atribuições. É comum termos de escrever nos programas expressões como i = i + 3; ou x = x \* (y + 2); Expressões deste tipo podem ser abreviadas para qualquer coisa como variável op= expressão; e que é equivalente a variável = variável op (expressão); Assim podemos reescrever as duas expressões anteriores como i += 3; e x \*= y + 2; Note-se que esta última expressão vale sempre x = x \* (y + 2); e nunca x = x \* y + 2;

|  |
| --- |
|  |

Operadores de comparação

O operador de teste de igualdade no C é ==. Note-se que é muito fácil trocar o teste de igualdade pelo operador de atribuição (só um =), o que conduz invariavelmente a erros difíceis de detectar. A seguinte instrução condicional está sintacticamente correcta: if (i = j) ... ,no entanto o que faz é copiar o valor de j para i e tem como resultado o valor de j, que é interpretado como TRUE se for diferente de 0 e FALSE no caso contrário. O que se pretendia era comparar i com j com resultado TRUE se fossem iguais...

O operador de teste de desigualdade é em C !=. Os outros quatro operadores de comparação são: <, <=, > e >=.

|  |
| --- |
|  |

Operadores lógicos

Os operadores lógicos servem para combinar resultados de comparações e são geralmente utilizados nas instruções condicionais. Os três operadores lógicos do C são:

1. not - !
2. and - &&
3. or - ||
4. Precedências
5. Todos os operadores do C têm um nível de precedência. Os operadores de nível mais elevado são avaliados antes dos operadores de nível mais baixo. Por exemplo, a expressão a + b \* c é avaliada como a + (b \* c) porque o operador \* tem maior precedência do que o operador +. Os operadores do mesmo nível de precedência são geralmente avaliados da esquerda para a direita (há excepções). Assim a - b - c é avaliado como (a - b) - c e não pela ordem inversa.
6. Na tabela seguinte podem ver-se todos os operadores do C pela ordem decrescente do nível de prioridade, e também a sua associatividade.

|  |  |
| --- | --- |
| **Operadores** | **Associatividade** |
| **() [] -> .** | esquerda para a direita |
| **! - ++ -- \* & (cast) sizeof** | direita para a esquerda |
| **\* / %** | esquerda para a direita |
| **+ -** | esquerda para a direita |
| **<< >>** | esquerda para a direita |
| **< <= > >=** | esquerda para a direita |
| **== !=** | esquerda para a direita |
| **&** | esquerda para a direita |
| **^** | esquerda para a direita |
| **|** | esquerda para a direita |
| **&&** | esquerda para a direita |
| **||** | esquerda para a direita |
| **?:** | direita para a esquerda |
| **= += -= \*= /= %= &= ^= |= <<= >>=** | direita para a esquerda |
| **,** | esquerda para a direita |

1. Assim a expressão a < 41 && 7 \* b < c é avaliada como (a < 41) && ((7 \* b) < c).   
   Da mesma forma a expressão a = b = fahr / celsius + limit é avaliada como a = (b = ((fahr / celsius) + limit)).

**Resumo Printf e Scanf (igual Escrever e Ler no Algoritmo)**

**printf()**

O comando printf() tem a função de mandar para a saída padrão uma string, por padrão a saída é o monitor, mas nada impede de mudar essa saída para um arquivo ou até mesmo para uma impressora.

A sua forma geral é: printf( <string> , [<variáveis>]);

Ex: printf("Hello World");

printf(" o número %d é %f ", x, y);

Enquanto que o comando scanf() tem a função de receber os dados(nomes,números,etc) que o usuário digita no teclado, armazenando em uma variável o conteúdo fornecido pelo mesmo.

A sua forma geral é: scanf( "tipo variável", &variável);

Ex: scanf("%f", &x);

Outro exemplo:

main()

{

int x;

printf(" Digite um valor! ");

scanf("%d", &x);

printf ("O valor digitado foi: %d", x);

}

Observações: Includes que talvez sejam necessários em seus programas:

#include <stdio.h>

#include <stdlib.h>

Execução Condicional

Veremos como tratar dos vários métodos de controlar o fluxo de execução de código num programa em C. Os operadores lógicos e de comparação são fundamentais para isso. Esses operadores são usados em conjunto com as instruções aqui tratadas.

A instrução if (igual comando “SE” em Algoritmo)

A instrução if tem a mesma função e estrutura de outras linguagens. Assume em C as duas formas básicas seguintes:

if (expressão)   
instrução;

ou

if (expressão)   
instrução\_1;   
else   
instrução\_2;

Pode ser facilmente imbricada para o teste de múltiplas condições, como se vê no exemplo seguinte:

if (expressão)   
instrução\_1;   
else if (expressão)   
instrução\_2;   
else   
instrução\_3;

Outro exemplo:

void main(void)   
{   
int x, y, z;

...   
if (x >= y) {   
z = x;   
...   
}   
else {   
z = y;   
...   
}   
...   
}

Notar a colocação das chaves alinhadas em cada bloco.

Outro exemplo:

#include <stdio.h>

int main ()

{

int num;

printf ("Digite um numero: ");

scanf ("%d",&num);

if (num>10)

   printf ("\n\nO numero e maior que 10");

if (num==10)

        {

        printf ("\n\nVoce acertou!\n");

        printf ("O numero e igual a 10.");

        }

if (num<10)

   printf ("\n\nO numero e menor que 10");

return(0);

}

**O else (igual “Senão” no Algoritmo)**

Podemos pensar no comando **else** como sendo um complemento do comando **[if](http://mtm.ufsc.br/~azeredo/cursoC/aulas/c410.html" \l "c410.html)**. O comando **[if](http://mtm.ufsc.br/~azeredo/cursoC/aulas/c410.html" \l "c410.html)** completo tem a seguinte forma geral:

*if (condição) declaração\_1;*  
*else declaração\_2;*

 A expressão da condição será avaliada. Se ela for diferente de zero a declaração 1 será executada. Se for zero a declaração 2 será executada. É importante nunca esquecer que, quando usamos a estrutura **if-else**, estamos garantindo que uma das duas declarações será executada. Nunca serão executadas as duas ou nenhuma delas. Abaixo está um exemplo do uso do **if-else** que deve funcionar como o programa da [seção anterior](http://mtm.ufsc.br/~azeredo/cursoC/aulas/c410.html#c410.html).

#include <stdio.h>

int main ()

{

int num;

printf ("Digite um numero: ");

scanf ("%d",&num);

if (num==10)

      {

      printf ("\n\nVoce acertou!\n");

      printf ("O numero e igual a 10.\n");

      }

else

      {

      printf ("\n\nVoce errou!\n");

      printf ("O numero e diferente de 10.\n");

      }

return(0);

}

**O if-else-if ( igual Se-Senão-Se no Algoritmo)**

A estrutura **if-else-if** é apenas uma extensão da estrutura **[if-else](http://mtm.ufsc.br/~azeredo/cursoC/aulas/c410.html" \l "c411.html)**. Sua forma geral pode ser escrita como sendo:

*if (condição\_1) declaração\_1;*  
*else if (condição\_2) declaração\_2;*  
*else if (condição\_3) declaração\_3;*  
*.*  
*.*  
*.*  
*else if (condição\_n) declaração\_n;*  
*else declaração\_default;*

 A estrutura acima funciona da seguinte maneira: o programa começa a testar as condições começando pela 1 e continua a testar até que ele ache uma expressão cujo resultado dê diferente de zero. Neste caso ele executa a declaração correspondente. Só uma declaração será executada, ou seja, só será executada a declaração equivalente à *primeira* condição que der diferente de zero. A última declaração (default) é a que será executada no caso de todas as condições darem zero e é opcional. Um exemplo da estrutura acima:

#include <stdio.h>

int main ()

{

int num;

printf ("Digite um numero: ");

scanf ("%d",&num);

if (num>10)

     printf ("\n\nO numero e maior que 10");

else if (num==10)

        {

        printf ("\n\nVoce acertou!\n");

        printf ("O numero e igual a 10.");

        }

     else if (num<10)

             printf ("\n\nO numero e menor que 10");

return(0);

}

A instrução switch (igual “Faça caso” no Algoritmo)

A instrução switch é semelhante à instrução case em outras linguagens, e tal como esta permite uma escolha múltipla de execução baseada no valor de uma expressão. A sua estrutura geral é a seguinte:

switch (expressão) {   
case item\_1:   
instrução\_1;   
break;   
case item\_2:   
instrução\_2;   
break;   
...   
case item\_n:   
instrução\_n;   
break;   
default:   
instrução;   
}

Os items que aparecem em seguida à palavra case são valores constantes (não podem ser expressões nem variáveis). Se a expressão inicial coincidir com algum desses items executa-se a instrução que vem logo a seguir. Caso contrário, e se a palavra default estiver presente (não é obrigatório), passa-se a essa instrução; se não estiver presente passa-se à instrução que se segue à instrução switch. A palavra break (é uma instrução do C) faz com que se passe imediatamente à instrução que se segue à presente (a instrução que virá a seguir a esta instrução switch no programa). Se não estiver presente a execução continua para a instrução do próximo case, o que não é usual em outras linguagens. Podemos considerar o seguinte exemplo:

switch (letter) {   
case 'A':   
case 'E':   
case 'I':   
case 'O':   
case 'U':   
nr\_de\_vogais++;   
break;   
case ' ':   
nr\_de\_espacos++;   
break;   
default:   
nr\_de\_consoantes++;   
}

Notar os 5 cinco primeiros casos (vogais), que conduzem à mesma instrução.

Outro exemplo:

#include <stdio.h>

#include <conio.h>

int main (void )

{

int valor;

printf ("Digite um valor de 1 a 7: ");

scanf("%d", &valor);

switch ( valor )

{

case 1 :

printf ("Domingo\n");

break;

case 2 :

printf ("Segunda\n");

break;

case 3 :

printf ("Terça\n");

break;

case 4 :

printf ("Quarta\n");

break;

case 5 :

printf ("Quinta\n");

break;

case 6 :

printf ("Sexta\n");

break;

case 7 :

printf ("Sabado\n");

break;

default :

printf ("Valor invalido!\n");

}

getch();

return 0;

}

Agora como ficaria o mesmo programa usando comandos if else aninhados (encadeados).

#include <stdio.h>

#include <conio.h>

int main (void )

{

int valor;

printf ("Digite um valor de 1 a 7: ");

scanf("%d", &valor);

if (valor == 1)

printf ("Domingo\n");

else

if (valor == 2)

printf ("Segunda\n");

else

if (valor == 3)

printf ("Terça\n");

else

if (valor == 4)

printf ("Quarta\n");

else

if (valor == 5)

printf ("Quinta\n");

else

if (valor == 6)

printf ("Sexta\n");

else

if (valor == 7)

printf ("Sabado\n");

else

printf ("Valor invalido!\n");

getch();

return 0;

}

Obs: getch() espera que o usuário digite uma tecla e retorna este caractere.

O retorno zero indica ao SO que o programa foi bem sucedido. Cada outro número retornado indica o código de uma condição de erro.

O mesmo programa poderia ser executado sem os comando getch() e return().

Ciclos e Iteração

Veremos como tratar das várias instruções do C utilizadas na construção de ciclos ou execução repetida (e controlada) de outras instruções.

A instrução while (igual o “Enquanto” no Algoritmo)

A instrução while é semelhante à que outras linguagens possuem. No entanto a utilização de uma expressão genérica como controle pode abrir novas possibilidades. A sintaxe geral da instrução while é:

while (expressão)   
instrução;

Um exemplo simples:

int x=3;

void main(void)   
{   
while (x > 0) {   
printf("x = %d\n", x);   
x--;   
}   
}

escreve no vídeo ...

x = 3   
x = 2   
x = 1

Uma vez que a instrução while pode receber uma expressão geral em vez de apenas uma condição (o valor da expressão é interpretada como TRUE se for diferente de 0, e como FALSE se for igual a 0), as instruções seguintes são todas legais:

while (x--);   
while (x = x + 1);   
while (x += 5);

|  |
| --- |
|  |

A instrução do-while (igual o “Repita – até que” no Algoritmo)

A instrução do-while testa a expressão que dela faz parte depois de executar as instruções do ciclo e continua enquanto se mantiver em TRUE. O ciclo é executado pelo menos 1 vez. A sua sintaxe é:

do   
instrução;   
while (expressão);

O seguinte código

int x = 3;

void main(void)   
{   
do   
printf("x = %d\n", x--);   
while (x > 0);   
}

escreve ...

x = 3   
x = 2   
x = 1

|  |
| --- |
|  |

A instrução for (igual a instrução “Faça “ no Algoritmo)

A sintaxe da instrução for é diferente do habitual para outras linguagens. Uma instrução for é então definida como:

for (expressão1; expressão2; expressão3)   
instrução;

A expressão1 é o inicializador, a expressão2 constitui o teste de terminação, e a expressão3 é o modificador (é executada em cada ciclo e pode fazer mais do que um simples incremento ou decremento). A instrução for é equivalente a um ciclo while com a seguinte construção:

expressão1;   
while (expressão2) {   
instrução;   
expressão3;   
}

Por exemplo, o seguinte código

int x;

void main(void)   
{   
for (x=3; x>0; x--)   
printf("x = %d\n", x);   
}

escreve no vídeo ...

x = 3   
x = 2   
x = 1

As próximas instruções for são todas legais:

for (x = 0; x <= 9 && x != 3; x++);   
for (x = 0, y = 4; x <= 3 && y < 9; x++, y += 2);   
for (x = 0, y = 4, z = 1000; z; z /= 10);

Reparar no uso do operador , para executar múltiplas acções. No terceiro ciclo, a execução continua até que z se torne 0 (FALSE).

break e continue

As instruções break e continue permitem-nos controlar melhor a forma de execução dos ciclos:

1. break - termina imediatamente a execução de um ciclo ou da instrução switch
2. continue - salta imediatamente para a avaliação da expressão de controle do ciclo

Arrays

Os arrays em C são semelhantes aos que são definidos e usados em outras linguagens. No entanto há algumas diferenças na sintaxe de declaração e utilização e alguma interligação com os apontadores, como veremos mais à frente.

**Arrays unidimensionais e multidimensionais**

Os arrays declaram-se no C como se declaram as outras variáveis, acrescentando apenas um valor para a sua dimensão. Assim começa-se por indicar o tipo de valores que vão estar contidos em cada posição do array, seguindo-se o nome do próprio, e terminando com a indicação da dimensão (nº de elementos) entre parêntesis retos. Por exemplo, para definir um array de 50 inteiros poderia usar-se a seguinte declaração:

int numbers[50];

Nota: Em C os arrays começam sempre no índice 0 e vão até ao valor da dimensão menos 1. Na definição de cima os índices válidos do array numbers vão desde 0 até 49.

O acesso aos elementos individuais do array faz-se utilizando também parêntesis retos. Exemplos:

terceiro\_valor = numbers[2];

numbers[5] = 100;

Os arrays multidimensionais declaram-se, indicando as várias dimensões umas a seguir às outras, da mesma forma que a primeira dimensão. Exemplo, para duas dimensões:

float matriz[50][50];

Para maiores dimensões basta acrescentar à declaração mais elementos [...]:

double big\_array[10][10][22]...[8];

Os acessos aos elementos individuais faz-se da mesma forma:

valor = matriz[10][4];

matriz[0][1] = 100;

ex:

/\* notas \*/

main()

{

float nota[5];

float soma;

int i;

for (i=0;i<5;i++)

{

printf("Digite a nota do aluno %d: ",i);

scanf("%f",&nota[i]);

}

soma=0;

for (i=0;i<5;i++)

{

soma=soma+nota[i];

}

printf ("media das notas:%f\n ", soma/5);

printf ("media das notas:%5.2f\n ", soma/5);

}

**Exemplo Bidimensional**

#include<stdio.h>

#include<conio.h>

int main(void)

{

int i, j, matriz[2][2] = {10,20,30,40};

/\* Imprimindo os atribuídos para a matriz \*/

for (i = 0; i <= 1; i++)

{

for(j = 0; j <=1; j++)

{

printf ("matriz[%d][%d] = %d\n", i,j, matriz[i][j]);

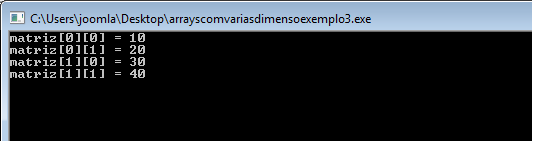
}

}

getch();

return 0;

}

[](http://linguagemc.com.br/wp-content/uploads/2012/11/arrayscomvariasdimensoexemplo3.png)

Tela de execução Valores atribuídos para a matriz

Outro exemplo

#include<stdio.h>

#include<conio.h>

int main(void)

{

int matriz[2][2] = {{11,22},{32,44}};

int i,j;

/\* Imprimindo os atribuídos para a matriz \*/

for (i = 0; i <= 1; i++)

{

for(j = 0; j <=1; j++)

{

printf ("matriz[%d][%d] = %d\n", i,j, matriz[i][j]);

}

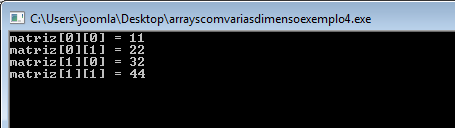
}

getch();

return 0;

}

**Tela de execução**

[](http://linguagemc.com.br/wp-content/uploads/2012/11/arrayscomvariasdimensoexemplo4.png)

Tela de execução outra forma de atribuição

**Declarando e inicializando de vetores**  
**Exemplo 1:**

Podemos declarar e inicializar um vetor com um tamanho constante, como abaixo:

int numeros[5] = {10, 20, 30, 40, 50};  
**Exemplo 2:**

Iniciando apenas alguns elementos do vetor:

int valores[5] = {2,4,6};

será equivalente a

int valores[5] = {2,4,6,0,0};

Isto ocorre porque apenas alguns itens do vetor foram inicializados.

Neste caso, quando o número de itens inicializados é menor que o número total de itens do vetor, os itens não inicializados são automaticamente zerados.

**Exemplo Programa 1**

#include<stdio.h>

#include<conio.h>

int main(void)

{

float notas[5] = {7, 8, 9.5, 9.9, 5.2};

// declarando e inicializando o vetor notas

printf("Exibindo os Valores do Vetor \n\n");

printf("notas[0] = %.1f\n", notas[0]);

printf("notas[1] = %.1f\n", notas[1]);

printf("notas[2] = %.1f\n", notas[2]);

printf("notas[3] = %.1f\n", notas[3]);

printf("notas[4] = %.1f\n", notas[4]);

getch();

return 0;

}

**Exemplo Programa 2**

#include<stdio.h>

#include<conio.h>

int main(void)

{

int i;

float notas[5] = {7, 8, 9.5, 9.9, 5.2};

// declarando e inicializando o vetor notas

printf("Exibindo os Valores do Vetor \n\n");

for( i = 0 ; i <= 4; i++)

{

printf("notas[%d] = %.1f\n",i, notas[i]);

}

getch();

return 0;

}

**Strings**

Na linguagem C os strings são simplesmente arrays de caracteres. Apenas na biblioteca standard se encontra uma vasta gama de funções de manuseamento de strings.

Uma declaração de uma string capaz de conter 50 caracteres:

char nome[51];

Como o C não manuseia strings directamente todas as seguintes instruções são ilegais:

char nome[50], apelido[50], nome\_completo[100];

nome = "Arnold"; /\* Ilegal \*/

apelido = "Schwarznegger"; /\* Ilegal \*/

nome\_completo = "Mr. " + nome + ' ' + apelido; /\* Ilegal \*/

No entanto a seguinte declaração com inicialização é válida:

char nome[50] = "Dave";

String é uma matriz do tipo char terminada pelo caracter null (‘\0’).

Para imprimir um string no vídeo pode usar-se a função printf(), com um especificador, no 1º parâmetro igual a %s, no local da variável do tipo string (char []):

printf("%s", nome);

EX1:

/\* programa nome \*/

char nome [10];

main()

{

printf (" digite o nome : ", nome);

scanf ("%s",nome);

printf ("\n o nome digitado e :%s",nome);

}

EX2:

/\* programa nome \*/

char nome [20];

main()

{

printf (" digite o nome : ", nome);

gets (nome);

printf ("\n o nome digitado e :%s",nome);

printf("\n");

puts(nome);

printf("\n");

puts(&nome[4]);

}

FUNÇÃO strlen()

Aceita um endereço de string como argumento e retorna o tamanho da string armazenada a partir deste endereço até um caracter antes do ‘\0’ .

EX:

/\* ex strlen \*/

char nome [30];

int num;

main()

{

printf("Digite o nome: ");

gets(nome);

printf("\n");

num=strlen(nome);

printf("%d",num);

}

FUNÇÃO strcat

Concatena duas strings, isto é, junta uma string ao final da outra. Ela toma dois endereços de strings como argumento e copia string no final da primeira e esta combinação gera uma nova primeira string. A segunda string não é alterada.

EX:

/\* ex strcat \*/

char nome1 [30], nome2 [30];

int num;

main()

{

printf("Digite o nome: ");

gets(nome1);

printf("\n");

printf("Digite o 2 nome: ");

gets(nome2);

printf("\n");

strcat(nome1,nome2);

printf("%s\n",nome1);

printf("%s\n",nome2);

}

FUNÇÃO strcmp()

Compara duas strings.

EX:

/\* ex strcmp \*/

char nome1 [30], nome2 [30];

int num;

main()

{

printf("Digite o nome: ");

gets(nome1);

printf("\n");

printf("Digite o 2 nome: ");

gets(nome2);

printf("\n");

if (strcmp(nome1,nome2) == 0 )

{

printf("Os nomes sao iguais: %s E %s",nome1,nome2);

}

else

{

printf("Os nomes sao diferentes: %s E %s",nome1,nome2);

}

}

Funções

Um programa em linguagem C é, como já se disse, essencialmente uma colecção de funções. Essas funções são muito semelhantes às que são possíveis de definir noutras linguagens. O próprio programa principal em C é uma função - a função main().

Definição

A sintaxe geral para a definição de uma função é a seguinte:

tipo\_de\_retorno nome\_da\_função (def\_parâmetro, def\_parâmetro, ...)   
{   
variáveis\_locais

instruções   
}

Por exemplo, uma função para calcular o valor médio de dois números, poderia ser definida da seguinte forma:

float average(float a, float b)   
{   
float ave;

ave = (a + b) / 2;   
return ave;   
}

Esta função poderia depois ser chamada da função main() como se mostra no exemplo seguinte:

void main(void)   
{   
float a=5, b=15, result;

result = average(a, b);   
printf("Média = %f\n", (double)result);   
}

Repare-se na instrução de return na função, que além de a terminar é também a responsável pela definição do valor de retorno da mesma.

Funções void

As funções void são funções que não retornam qualquer. Estas funções são definidas tendo como tipo de retorno a palavra void. A terminação pode fazer-se simplesmente pelo esgotar das instruções contidas na função (diz-se que ocorre uma situação de *fall-through* nessas circunstâncias) ou através de uma instrução de return sem a especificação de qualquer valor adicional. A palavra void também deverá ser usada no lugar dos parâmetros se a função não tiver nenhum. Um exemplo:

void squares(void)   
{   
int k;

for (k=1; k<=10; k++)   
printf("%d\n", k\*k);   
}

void main(void)   
{   
squares();   
}

Na chamada de funções sem argumentos é sempre obrigatório utilizar parêntesis, sem nada lá dentro, como se vê acima.

**Outro Exemplo de Função**

#include<stdio.h>

#include<conio.h>

/\*

Nome da função: multiplica

recebe como parâmetos dois valores inteiros (N1,N2)

objetivo: multiplicar os valores recebidos nos parâmetros.

retorno: um parâmetro inteiro (res) contendo o resultado

\*/

int multiplica(int N1, int N2) //multiplica recebe N1,N2 e retorna um int

{

int resultado;

resultado = N1 \* N2;

return(resultado); //retornando o valor para main

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* função principal (main) \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int main(void)

{

int V1, V2, resultado;

printf("Digite o primeiro valor:");

scanf("%d", &V1);

printf("Digite o segundo valor:");

scanf("%d", &V2);

//chama a função e recebe o retorno

resultado = multiplica(V1,V2);

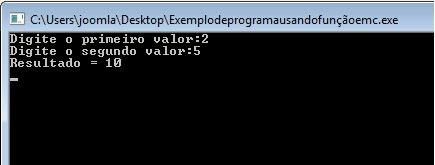
printf("Resultado = %d\n", resultado);

getch();

return 0;

}

**TELA DE EXECUÇÃO**

[](http://linguagemc.com.br/wp-content/uploads/2012/05/Exemplodeprogramausandofun%C3%A7%C3%A3oemc.png)

Tela de execução do programa usando função em C

# **Recursividade em C**

Chamamos de **recursividade ou recursão** quando uma função chama a si mesma.

Em uma função recursiva, a cada chamada é criada na memória uma nova ocorrência da função com comandos e variáveis “isolados” das ocorrências anteriores.

A função é executada até que todas as ocorrências tenham sido resolvidas.

Porém um problema que surge ao usar a recursividade é como fazê-la parar. Caso o programador não tenha cuidado é fácil cair num loop infinito recursivo o qual pode inclusive esgotar a memória…

Toda recursividade é composta por um **caso base** e pelas **chamadas recursivas**.

**Caso base:** é o caso mais simples. É usada uma condição em que se resolve o problema com facilidade.

**Chamadas Recursivas:** procuram simplificar o problema de tal forma que convergem para o caso base.

**Exemplo clássico de recursividade: fatorial**

//Cálculo de fatorial com função recursiva

#include <stdio.h>

#include <conio.h>

//protótipo da função fatorial

double fatorial(int n);

int main(void)

{

int numero;

double f;

printf("Digite o numero que deseja calcular o fatorial: ");

scanf("%d",&numero);

//chamada da função fatorial

f = fatorial(numero);

printf("Fatorial de %d = %.0lf",numero,f);

getch();

return 0;

}

//Função recursiva que calcula o fatorial

//de um numero inteiro n

double fatorial(int n)

{

double vfat;

if ( n <= 1 )

//Caso base: fatorial de n <= 1 retorna 1

return (1);

else

{

//Chamada recursiva

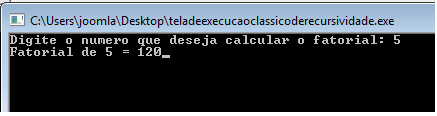
vfat = n \* fatorial(n - 1);

return (vfat);

}

}

**Tela de Execução**

[](http://linguagemc.com.br/wp-content/uploads/2012/06/teladeexecucaoclassicoderecursividade.png)

Tela de execução clássico de recursividade fatorial