8 Funções

Vanessa Lindemann*

Além da função `main` e das diversas funções pré-definidas das bibliotecas padrões da linguagem, um programa em C pode conter funções definidas pelo próprio programador, que só serão executadas se forem, direta ou indiretamente, chamadas pela função principal.

As funções são criadas para organizar o código fonte do programa, resolvem problemas bem específicos e, se forem bem construídas, são facilmente reutilizadas. O programador pode, por exemplo, criar uma biblioteca (arquivo de cabeçalho) com funções para validar data, validar hora e desenhar bordas, que poderão ser utilizadas sempre que necessário, incorporando-as aos seus programas através da diretiva `#include`.

A sintaxe de uma função em linguagem C é apresentada a seguir.

```
tipo_de_retorno identificador(lista_de_parâmetros){
  // declaração de variáveis locais
  // corpo da função
  return(valor);
}
```

Tipo de retorno: a função pode retornar um valor para a função chamadora. O tipo de retorno indica qual o tipo do dado que será devolvido. Nos casos em que a função não retorna nenhum valor, funcionando como um procedimento, o tipo de retorno deve ser definido como `void`.

Identificador: indica o nome da função, através do qual o bloco de código correspondente à função será chamado.

Lista de parâmetros: os parâmetros de uma função são variáveis locais automaticamente inicializadas com os valores passados no momento da sua chamada. Cada um dos parâmetros deve ser declarado, indicando o seu tipo e nome. Se a função não receber nenhum valor por parâmetro, os parênteses ao lado do nome da função ficam vazios.

Declaração de variáveis locais: as variáveis de escopo local devem ser declaradas e só podem ter seus valores acessados dentro da função, enquanto esta estiver ativa.

Corpo da função: conjunto de instruções que compõem a função. É onde é resolvido o problema para o qual a função foi escrita.

* Doutora em Informática na Educação pela UFRGS (2008); professora dos cursos da Computação da ULBRA.

Return: o comando `return` finaliza a execução da função que está sendo executada. Se a função retornar algum valor este comando é obrigatório. Se a função for do tipo `void` (ou seja, não retorna nenhum valor), este comando não é necessário.

Existem duas formas de passagem de parâmetros às funções: passagem de parâmetros por valor (onde uma cópia do valor passado pela função chamadora é fornecido à função chamada) e passagem de parâmetros por referência (quando o que é passado da função chamadora à função chamada, é o endereço de memória das variáveis) – temas abordados nas próximas seções. Uma mesma função, se receber mais que um argumento, pode combinar passagem de parâmetro por valor e por referência.

8.1 Passagem de parâmetros por valor

Na passagem de parâmetros por valor o parâmetro passado para a função é uma cópia da variável (argumento), de forma que alterar a variável dentro da função que a recebeu não irá alterar a variável original, da qual ela foi copiada.

Um exemplo da passagem de parâmetros por valor é apresentado na Figura 8.1. As variáveis `a` e `b` são declaradas e inicializadas com zero na função principal (linha 11). Na linha 17, a instrução `incrementa(a,b);` faz a chamada à função, enviando as variáveis `a` e `b` como parâmetros. Neste momento, o programa desvia seu fluxo de execução para a linha 24, quando o conteúdo das variáveis `a` e `b` é copiado para as variáveis `x` e `y` (ou seja, `x` e `y` recebem zero). Dentro da função, as variáveis `x` e `y` são incrementadas (linhas 25 e 26) e têm seus valores exibidos na tela (linha 28). Ao encontrar o final da função, na linha 29, o fluxo de execução do programa retorna à função principal, onde irá executar a instrução seguinte à chamada da função – neste caso, linha 19. Na função principal, os valores das variáveis `a` e `b` são exibidos novamente na tela (linha 20), demonstrando que estes não foram alterados dentro da função `incrementa`.

Sempre que uma função é chamada antes do ponto onde ela foi criada, deve-se declarar o protótipo da função acima do início do programa. O protótipo corresponde à primeira linha da função (onde aparece o seu nome) finalizada por um ponto-e-vírgula – isto pode ser observado no exemplo da Figura 8.1. A função `incrementa` é chamada na linha 17, antes do ponto em que ela foi criada, que é entre as linhas 24 e 29, por isso, na linha 8, foi incluído o protótipo da função (o protótipo da função "avisa" ao compilador que a função chamada na linha 17 está descrita mais adiante, caso contrário, ele retornaria um erro de função não identificada).

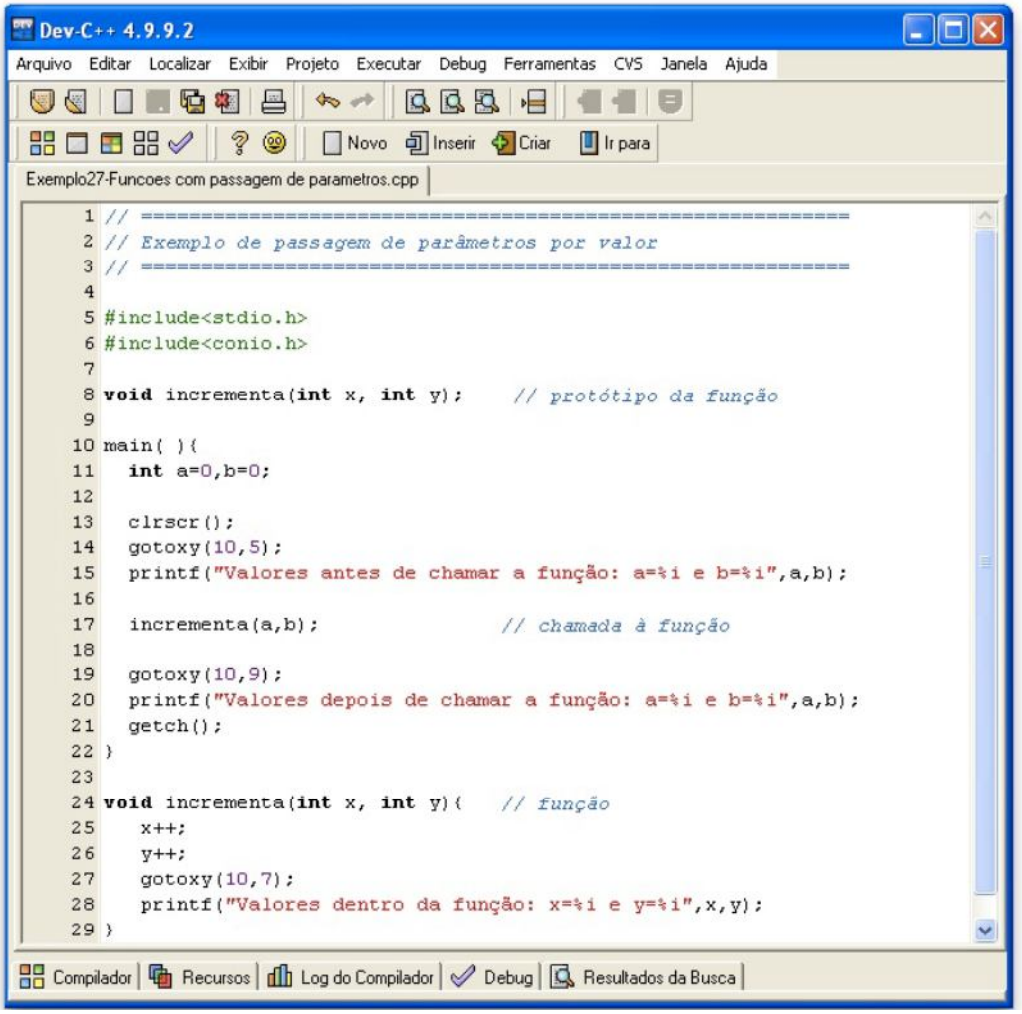

Figura 8.1 – Exemplo da passagem de parâmetros por valor

No exemplo da Figura 8.1, o tipo `void` antes do nome da função `incrementa` indica que esta não retorna nenhum valor à função chamadora, diferente do programa apresentado na Figura 8.2, que retorna um valor inteiro.

Este programa, lê dois valores na função principal (`valor1` e `valor2`) e faz a chamada à função `fsoma` (linha 20) enviando-os como parâmetros. As variáveis `v1` e `v2` recebem uma cópia do conteúdo de `valor1` e `valor2` (linha 27). Dentro da função `fsoma`, a variável `result` recebe o resultado da soma de `v1` e `v2` (linha 29). O conteúdo da variável `result` é enviado à função chamadora. Na linha 20, a variável `soma` recebe o retorno da função `fsoma`. O esquema da Figura 8.3 ilustra o vai e volta de valores e a função `main` e a `fsoma`.

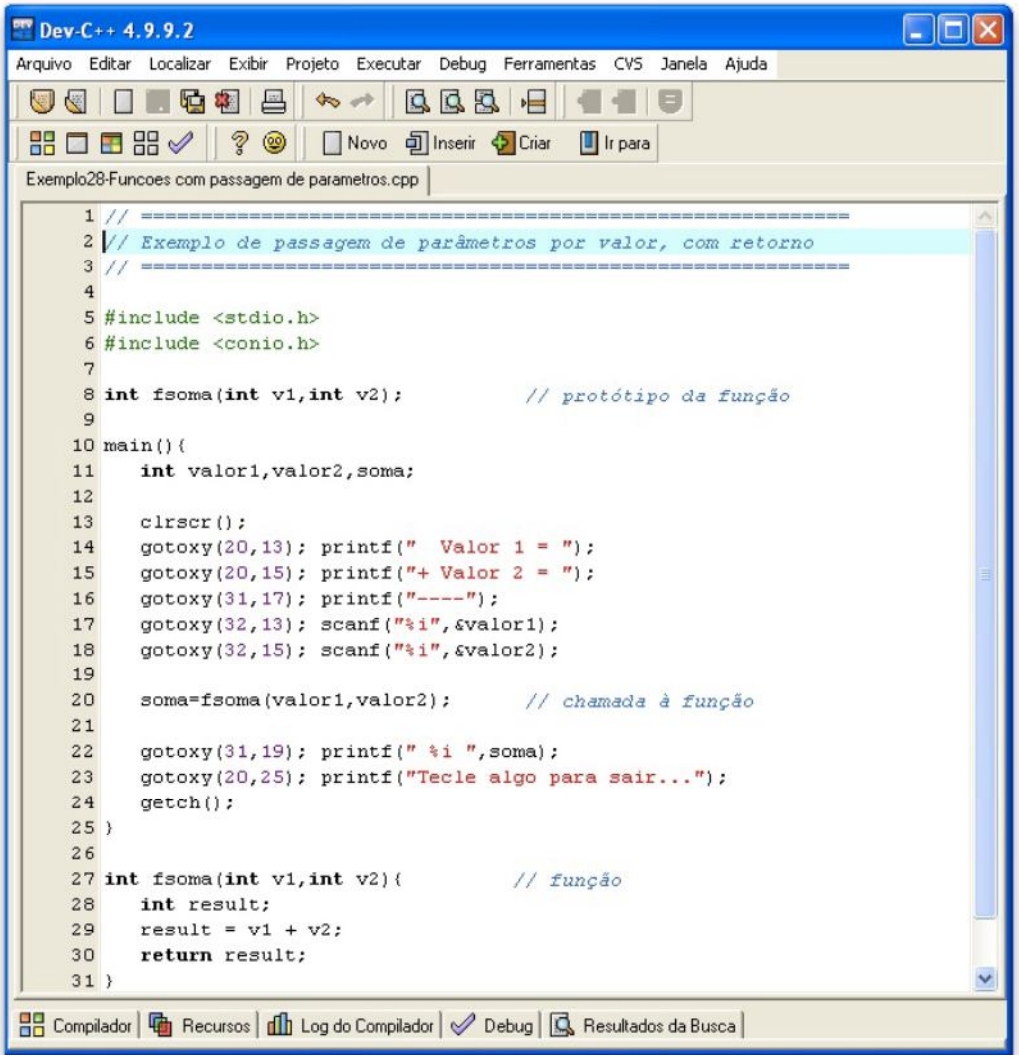

Figura 8.2 – Exemplo da passagem de parâmetros por valor, com retorno

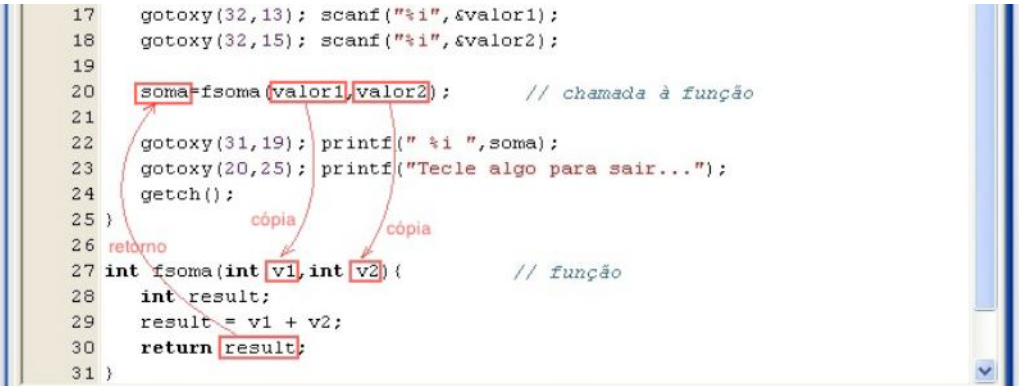

Figura 8.3 – Exemplo da passagem de parâmetros por valor, com retorno

8.2 Passagem de parâmetros por referência

Quando a função precisa alterar o valor da variável passada como argumento, é o endereço da variável que deve ser passado como parâmetro, o que caracteriza a passagem de parâmetros por referência. Como é o endereço de memória da variável que é passado como parâmetro, a variável que recebe o parâmetro, na função chamada, deve ser declarada como um ponteiro. Isto pode ser observado no programa da Figura 8.4. A diferença entre este programa e o anterior (apresentado na Figura 8.2) é que com a passagem de parâmetros por referência, o conteúdo das variáveis `a` e `b` é alterado de dentro da função `incrementa`.

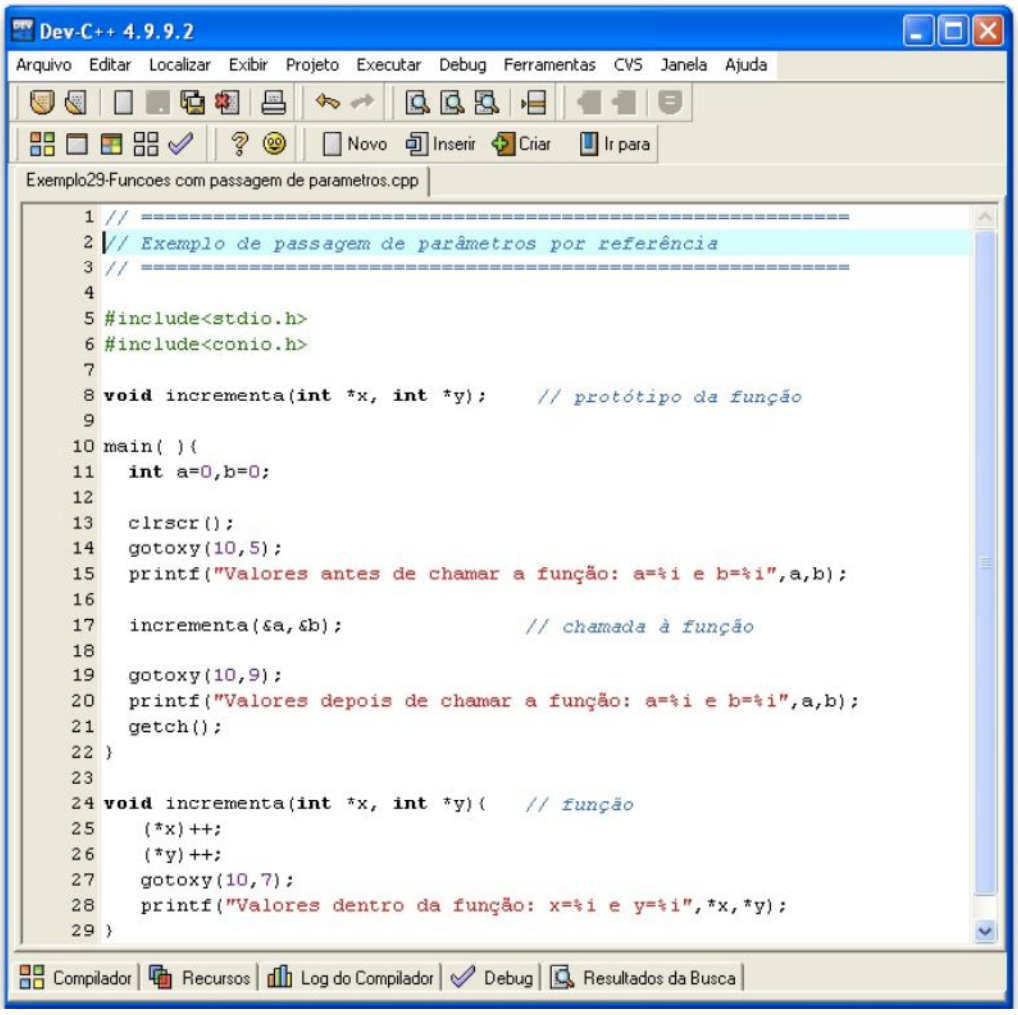

Figura 8.4 – Exemplo da passagem de parâmetros por referência

Como foi explicado anteriormente, o que caracteriza a passagem de parâmetros por referência é que, ao invés de fazer uma cópia do conteúdo das variáveis `a` e `b`, envia-se à função chamada, os endereços dessas variáveis. Para enviar o endereço de uma variável à função chamada, é

necessário incluir o `&` antes do seu nome, como pode ser observado na linha 17 da Figura 8.4. Consequentemente, na função chamada, os parâmetros devem ser declarados como ponteiros para receber os endereços enviados, como pode ser observado na linha 24 do programa da Figura 8.4. As instruções `(*x)++` e `(*y)++` (linhas 25 e 26) alteram o valor das variáveis originais `a` e `b`, o que pode ser confirmado pela saída do programa.

O programa da Figura 8.5 apresenta mais um exemplo da passagem de parâmetros por referência. Este programa lê dois caracteres na função `main` e faz a chamada à função `troca`, enviando os endereços de memória das variáveis lidas, para que estas tenham seus conteúdos trocados dentro da função `troca`. Quando o programa retorna o fluxo de execução para a função `main`, o conteúdo das variáveis é exibido na tela.

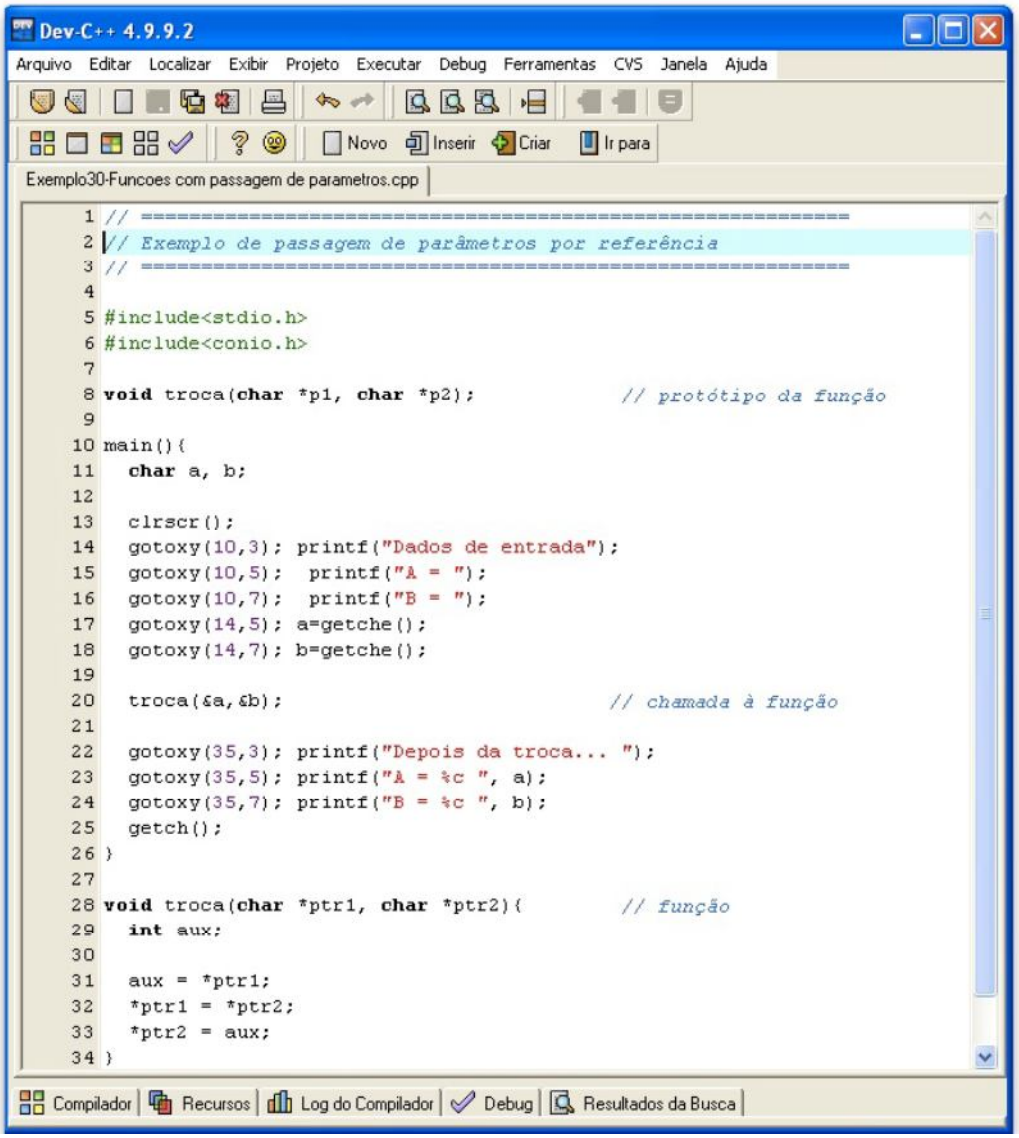

Figura 8.5 – Exemplo da passagem de parâmetros por referência

Os endereços são passados como parâmetros por duas razões: (a) pela eficiência – é mais rápido passar o endereço de uma estrutura, por exemplo, do que a estrutura em si; (b) pela possibilidade de manipular fisicamente uma variável externa a uma função – normalmente, o resultado de uma chamada de função é expresso pelo valor de retorno da função, entretanto, se os endereços das variáveis são passados como parâmetros, a função pode afetar diretamente quantas variáveis forem necessárias na função chamadora.

8.3 Vetores como argumento de funções

Sempre que um vetor for passado como parâmetro, obrigatoriamente, ele será passado por referência. Ao fazer a chamada a uma função com um vetor como argumento, o que é enviado à função chamada é um ponteiro que armazena o endereço do primeiro elemento do vetor. O programa da Figura 8.6 exemplifica como passar um vetor como parâmetro para uma função.

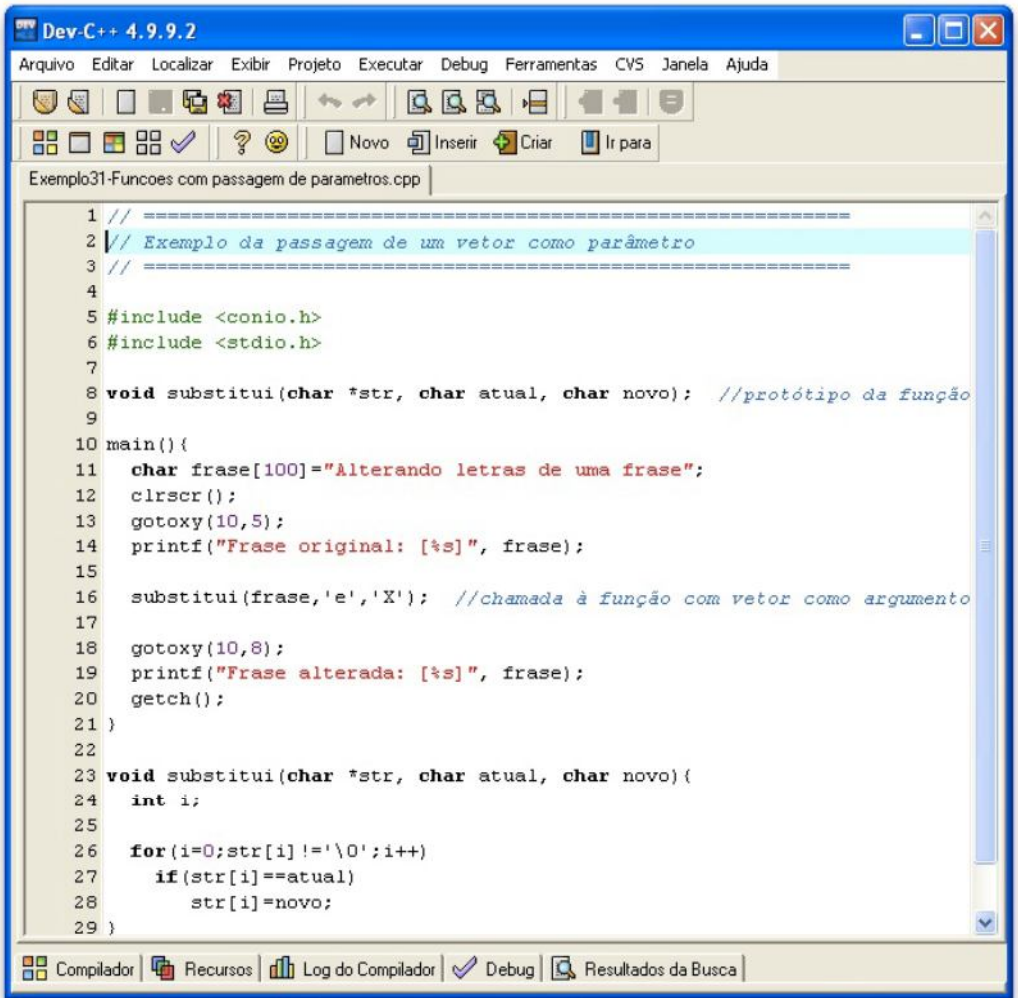

Figura 8.6 – Exemplo de vetor como argumento da função

Na linha 11, a *string* `frase` é inicializada com o literal *Alterando letras de uma frase* – este é exibido na tela através da instrução da linha 14. A chamada à função `substitui` aparece na linha 16, onde a *string* `frase` é enviada à função chamada, junto com os caracteres *e* e *X* (como foi explicado no capítulo anterior, em linguagem C, sempre que o nome de um vetor aparece sem o índice, este corresponde ao endereço de memória do seu primeiro elemento).

Na função `substitui`, consequentemente, foram declarados três parâmetros para receber os valores enviados: `char *str`, `char atual`, `char novo` – um ponteiro para receber o endereço do primeiro elemento do vetor e duas variáveis do tipo `char` para receber as letras *e* e *X*. A estrutura `for` (linha 26) é usada para percorrer todas as posições da variável `frase` e substituir as letras *e* por *X* (linhas 27 e 28). Ao retornar o fluxo de execução do programa para a função `main`, a variável `frase` é exibida já com as alterações realizadas dentro da função.

Apesar de vetores serem enviados como parâmetros sempre por referência, como foi explicado no início desta seção, o primeiro parâmetro formal da função `substitui`, declarado como `char *str`, também poderia ser declarado como `char str[100]` ou `char str[]`, como ilustram os exemplos do Quadro 8.1 – nos dois exemplos a variável `v1` terá seus valores incrementados em um dentro da função.

Quadro 8.1 – Exemplos da declaração de vetor como parâmetro formal

Exemplos de vetor como argumento da função	

`main( ){`
`  int i, v1[5]={0,0,0,0,0};`
 	
`  incrementa(v1);`
 	
`  for(i=0;i<5;i++)`
`  printf("%i ",v1[i]);`
`}`
 	
`void incrementa(int v2[5]){`
`  int i;`
`  for(i=0;i<5;i++)`
`  v2[i]++;`
`}`
	`main( ){`
	`  int i,v1[5]={0,0,0,0,0};`
	
	`  incrementa(v1);`
	
	`  for(i=0;i<5;i++)`
	`  printf("%i ",v1[i]);`
	`}`
	
	`void incrementa(int *v2){`
	`  int i;`
	`  for(i=0;i<5;i++){`
	`  *v2=*v2+1;`
	`  v2++;`
	`  }`
	`}`

8.4 Matrizes como argumento de funções

Os programas apresentados nas Figuras 8.7 e 8.8 exemplificam como usar matrizes como argumentos de funções. No primeiro exemplo, o programa lê uma matriz 3x4 na função `main` e um valor que será pesquisado nela. Na linha 23 ocorre a chamada à função `consulta`, que têm como argumentos a matriz e o valor lido. Na função chamada, a matriz é percorrida e seus elementos são comparados com o valor a ser pesquisado nela (linha 34), sendo que cada vez que o valor é encontrado, a variável `cont` é incrementada em um (linha 35). Neste exemplo, a função `consulta` não retorna nenhum valor à função `main`, por isso esta foi declarada com o tipo `void`.

Exemplo32-Funções com passagem de parametros.cpp

```
1  // ============================================================
2  // Exemplo da passagem de uma matriz como parâmetro
3  // ============================================================
4
5  #include <conio.h>
6  #include <stdio.h>
7
8  void consulta(int m[3][4],int v);       // protótipo da função
9
10 main(){
11   int mat[3][4],valor,l,c;
12   clrscr();
13   gotoxy(10,5);   printf("Matriz 3x4 ");
14   for(l=0;l<3;l++){
15     for(c=0;c<4;c++){
16        gotoxy(c*4+10,l*2+7);
17        scanf("%i",&mat[l][c]);
18     }
19   }
20   gotoxy(40,7);  printf("Valor a consultar na matriz: ");
21   scanf("%i",&valor);
22
23   consulta(mat,valor);    //chamada à função com matriz como argumento
24
25   gotoxy(10,20);  printf("Pressione qualquer tecla para sair...");
26   getch();
27 }
28
29 void consulta(int m[3][4],int v){       // função
30   int l,c,cont=0;
31
32   for(l=0;l<3;l++)
33     for(c=0;c<4;c++)
34        if(m[l][c]==v)
35           cont++;
36   gotoxy(40,9);
37   printf("%i aparece %i vezes na matriz.",v,cont);
38 }
```

Figura 8.7 – Exemplo de matriz como argumento da função

No exemplo da Figura 8.8, o programa lê uma matriz 4x4 na função `main` e a envia como parâmetro à função `maior`, que irá percorrer a matriz

e identificar qual o maior elemento. O maior elemento, por sua vez, é enviado via retorno à função `main`.

Exemplo33-Funções com passagem de parametros.cpp

```
1  // ============================================================
2  // Exemplo da passagem de uma matriz como parâmetro
3  // ============================================================
4
5  #include <conio.h>
6  #include <stdio.h>
7
8  int maior(int m[4][4]);       // protótipo da função
9
10 main(){
11   int mat[4][4],retorno,l,c;
12   clrscr();
13   gotoxy(10,5);   printf("Matriz 4x4 ");
14   for(l=0;l<4;l++){
15     for(c=0;c<4;c++){
16        gotoxy(c*4+10,l*2+7);
17        scanf("%i",&mat[l][c]);
18     }
19   }
20
21   retorno=maior(mat);     //chamada à função com matriz como argumento
22
23   gotoxy(10,17);  printf("Maior elemento da matriz = %i",retorno);
24   gotoxy(10,20);  printf("Pressione qualquer tecla para sair...");
25   getch();
26 }
27
28 int maior(int m[4][4]){       // função
29   int l,c,ma=0;
30
31   for(l=0;l<4;l++)
32     for(c=0;c<4;c++)
33        if(m[l][c]>ma)
34           ma=m[l][c];
35   return ma;
36 }
```

Figura 8.8 – Exemplo de matriz como argumento da função

8.5 Estruturas como argumento de funções

Os exemplos do Quadro 8.2 demonstram como fazer a passagem de parâmetros por valor e por referência quando o argumento da função é uma estrutura. No primeiro caso, a estrutura que está sendo passada à função vai ser copiada, membro a membro, para o parâmetro formal da função. No segundo exemplo, na passagem de parâmetros por referência, usa-se um ponteiro para a estrutura.

Quadro 8.2 – Exemplos de estrutura como argumento da função

Estrutura como argumento da função	
Passagem de parâmetros por valor	**Passagem de parâmetros por referência**

`struct notas {`
`  float med;`
`  char result;`
`};`
 	
`void modifica(struct notas caluno);`
 	
`main( ){`
`  struct notas aluno;`
 	
`  aluno.med=8;`
`  aluno.result='A';`
`  printf("Dados iniciais:");`
`  printf("%.1f ",aluno.med);`
`  printf("%c",aluno.result);`
 	
`  modifica(aluno);`
 	
`  printf("Dados depois da função:”);`
`  printf("%.1f ",aluno.med);`
`  printf("%c",aluno.result);`
`}`
 	
`void modifica(struct notas caluno){`
`  caluno.med=3;`
`  caluno.result='R';`
`  printf("Cópia do aluno);`
`  printf("%.1f ",caluno.med);`
`  printf("%c",caluno.result);`
`}`
	`struct notas {`
	`  float med;`
	`  char result;`
	`};`
	
	`void modifica(struct notas *ptaluno);`
	
	`main( ){`
	`  struct notas aluno;`
	
	`  aluno.med=8;`
	`  aluno.result='A';`
	`  printf("Dados iniciais:);`
	`  printf("%.1f ",aluno.med);`
	`  printf("%c",aluno.result);`
	
	`  modifica(&aluno);`
	
	`  printf("Dados depois da função");`
	`  printf("%.1f ",aluno.med);`
	`  printf("%c",aluno.result);`
	`}`
	
	`void modifica(struct notas *ptaluno){`
	`  ptaluno->med=3;`
	`  ptaluno->result='R';`
	`  printf("Aluno);`
	`  printf("%.1f ",ptaluno->med);`
	`  printf("%c",ptaluno->result);`
	`}`

Ponteiros que apontam para estruturas podem ser declarados e inicializados da mesma forma que ponteiros que apontam para variáveis simples. A única particularidade deste tipo de ponteiro é que, para acessar indiretamente o valor armazenado no membro de uma estrutura através de um ponteiro, usa-se o operador -> (seta) e não o * (asterisco) utilizado nos demais tipos de ponteiros. O uso desse operador pode ser observado na função `modifica` do exemplo de passagem de parâmetros por referência do Quadro 8.2

8.6 Estruturas como retorno de funções

O programa da Figura 8.7, que verifica se uma data é válida, exemplifica o uso de uma estrutura como argumento e retorno da função.

Exemplo34-Funções com passagem de parametros.cpp

```
1  // ============================================================
2  // Exemplo de estrutura como argumento e no retorno da função
3  // ============================================================
4
5  #include<conio.h>
6  #include<stdio.h>
7
8  struct tipodata {
9     int dia, mes, ano, valida;
10 };
11
12 tipodata verificadata(tipodata vdata);
13
14 main(){
15   tipodata data;
16
17   printf("Dia = ");  scanf("%i",&data.dia);
18   printf("Mês = ");  scanf("%i",&data.mes);
19   printf("Ano = ");  scanf("%i",&data.ano);
20
21   data=verificadata(data);
22
23   if(data.valida==1)
24     printf("%02i/%02i/%i = data válida!",data.dia,data.mes,data.ano);
25   else
26     printf("%02i/%02i/%i = data inválida",data.dia,data.mes,data.ano);
27   getch();
28 }
29
30 tipodata verificadata(tipodata vdata){
31
32   if(vdata.ano>=1900 && vdata.ano<9999)
33     // ========== Testa os meses com 30 dias ==========
34     if(vdata.dia>=1 && vdata.dia<=30 &&
35       (vdata.mes==4 || vdata.mes==6 || vdata.mes==9 || vdata.mes==11))
36         vdata.valida=1;
37     else
38       // ========= Testa os meses com 31 dias ==========
39       if(vdata.dia>=1 && vdata.dia<=31 &&
40         (vdata.mes==1 || vdata.mes==3 || vdata.mes==5 ||
41          vdata.mes==7 || vdata.mes==9 || vdata.mes==11))
42            vdata.valida=1;
43       else
44         // ======= Testa o mês de fevereiro ==========
45         if(vdata.dia>=1 && vdata.dia<=28 && vdata.mes==2)
46             vdata.valida=1;
47         else
48           // ====== Testa o mês de fevereiro, ano bissexto ========
49           if(vdata.dia>=1 && vdata.dia<=29 && vdata.mes==2 &&
50              (vdata.ano%4==0 && vdata.ano%100!=0)||vdata.ano%400==0)
51                vdata.valida=1;
52           else
53                vdata.valida=0;
54
55    return (vdata);
56 }
```

Figura 8.7 – Exemplo de estrutura como argumento e retorno da função