

ESTRUTURA DE DADOS

FUNÇÕES

Olá!

Ao final desta aula, você será capaz de:

1. Compreender o uso de funções definidas pelo programador;
2. Compreender a diferença entre parâmetros passados por valor e parâmetros passados por referência;
3. Implementar funções com e sem retorno, com e sem passagem de parâmetros;
4. Compreender escopo de variáveis (global e local);
5. Implementar funções tendo vetores como parâmetros;
6. Construir sua biblioteca de funções.

1 Libere sua imaginação. Construa suas funções.

Todo programa codificado em C++ é formado por uma ou mais funções, mas a única obrigatória é a função *main*.

Na disciplina de Algoritmos, construímos programas com uma só função e você deve se lembrar do tamanho deles no início. Apesar de pequenos, mesmo assim, era difícil depurar os erros.

Embora tenhamos continuado com uma única função, nossos programas foram aumentando à medida que incorporamos novas instruções e identificar e corrigir erros para que fosse possível a compilação, tomou-se uma tarefa muito trabalhosa. Nesta aula, iremos entrar em contato com o conceito de **modularização** e como consequência, aprenderemos a construir funções.



2 Processando Informações

2.1 Modularização

O grau de complexidade que alcança alguns programas nos leva a dividi-lo em módulos para que possamos melhorar o entendimento do programa, depurar os erros com mais facilidade, simplificar a manutenção porque podemos trocar somente um determinado módulo, reutilizar o código no mesmo programa ou em outro programa, etc. Esses módulos são "pequenos programas" com tarefas bem definidas. São conhecidos em algumas linguagens por procedimentos e, na linguagem C++, por funções. Chamamos essa técnica que decompõe um programa em partes menores de modularização.

2.2 Função

Por que criar funções se a linguagem C++ apresenta um conjunto razoável de funções pré-definidas?

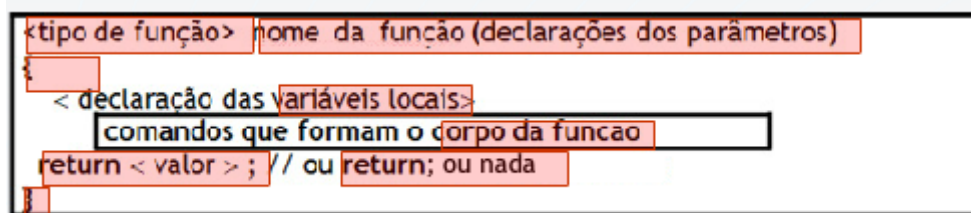
Porque, por maior que seja esse conjunto, nunca atenderá, totalmente, às necessidades de todos os programas.

Conceito de Funções

Uma função é um conjunto de comandos limitado por um par de chaves e precedido por um cabeçalho.

Na construção de uma função, todas as estruturas que você estudou na disciplina de Algoritmos poderão ser usadas no corpo da função, mas uma função é independente da outra e, por essa razão, jamais poderá ser criada dentro de outra função.

Definição da função:



Tipo de função: Tipo de retorno da função.

Nome da função (declaração dos parâmetros): Tipo e nome de cada um dos parâmetros se houver. Havendo mais do que um, separe-os por vírgulas. Mesmo que tenham o mesmo tipo, o tipo não poderá ser omitido.

Variáveis locais: Declarações de variáveis que serão utilizadas dentro da função (tipo e nome).

Corpo da função: Conjunto de comandos.

Return (valor): return< valor >; / / Este comando só existe se a função retornar um valor para a função chamadora.

Return (ou nada): return; ou nada// Para a função do tipo void.

Entendemos que a criação de uma função é resultado de uma necessidade que você constatou e da ausência de uma função pré-definida.

Alguns exemplos de funções que você poderá criar:

- Função que calcule a média aritmética;
- Função que reajusta salário;
- Função que exibe um menu;
- Função que gera um outro vetor;
- Função que ordena um vetor;
- Função que remove elementos de uma fila.

O Protótipo de uma Função

O protótipo de uma função contém o tipo de retorno da função, o nome da função, seguido de um par de parênteses que podem ou não ter parâmetros. É finalizado, obrigatoriamente, com o ;(ponto-e-vírgula).

Em outras palavras, o protótipo de uma função é o cabeçalho da função com ;(ponto-e-vírgula) ao final.

<tipo de função> nome_da_função (declarações dos parâmetros);

Localização das Funções

As funções poderão ser colocadas antes, ou depois, da *main*.

Antes: Se colocadas antes da *main*, elas serão "conhecidas" pelo compilador antes de serem chamadas, evitando maiores problemas, mas, à medida que o número de funções cresce, diminui a legibilidade do programa.

Depois: Se colocadas depois da *main*, aumentamos a legibilidade, mas criamos um problema: função é chamada antes da identificação dela pelo compilador. Esse contratempo foi resolvido de uma forma muito simples: os protótipos das funções serão localizados antes da *main* e as definições, depois da *main*. Veja um exemplo na seguinte figura.

```
tipo função1(...);  
tipo função2(...);  
tipo função3(...);  
int main()  
{  
    ...  
}  
tipo função1(...)  
{  
    ...  
}  
tipo função2(...)  
{  
    ...  
}  
tipo função3(...)  
{  
    ...  
}
```

Atenção

1 - Não confunda. Definição é o cabeçalho e o corpo enquanto que protótipo é o cabeçalho com ponto e vírgula ao final, podendo até se apresentar de forma simplificada como veremos mais adiante.

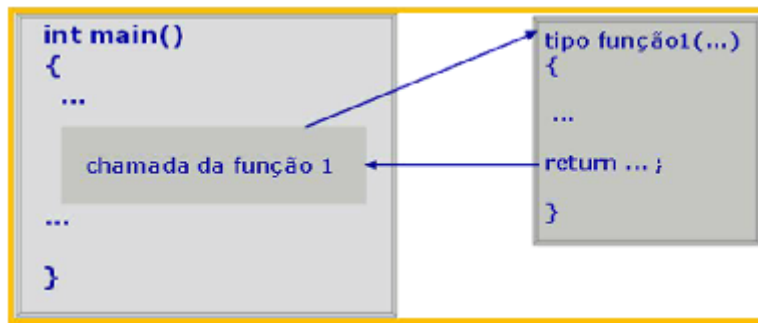
2 - A prototipagem é uma técnica que declara funções através dos seus protótipos antes da *main*, indicando assim que as funções existem, mas se encontram em outro lugar.

Chamada da função e o seu retorno

Quando uma função é chamada, o fluxo de controle é desviado para essa função e, a partir desse momento, os comandos da função são executados e, ao finalizar, o fluxo retorna ao comando seguinte daquele onde ela foi ativada (se for void) ou ao ponto de onde ela foi chamada, para funções com retorno. Para que tudo isso seja possível, não poderemos desconsiderar o tipo de retorno e os parâmetros, caso existam. As figuras abaixo poderão esclarecer melhor o que foi dito, mas, para de fato você entender, precisará conhecer uma estrutura de dados chamada pilha que estudaremos em outra aula.

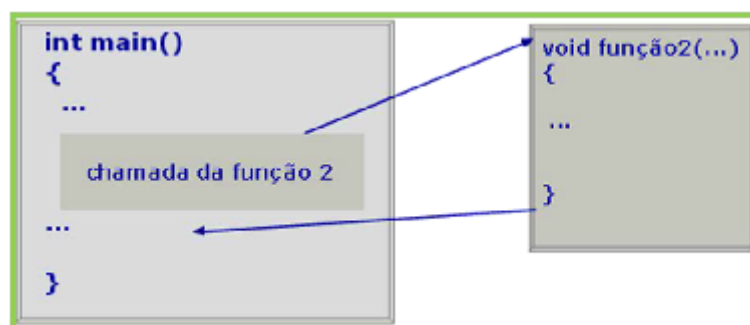
Exemplo de uma chamada de função com retorno

Observe que o retorno é para o ponto de chamada, isto é, para o nome da função e tipo poderá ser int, double, float, mas não, void.



Exemplo de uma chamada de função do tipo void

Observe que o retorno é para a instrução seguinte.



Tipos de Funções

Função sem parâmetros

Na linguagem C++ não é obrigatório o uso de parâmetros em todas as funções. Sendo assim, nesse primeiro momento, vamos analisar as funções sem parâmetros, pois não necessitam do domínio dos conceitos de passagem por valor ou por referência e sem retorno por serem de mais fácil entendimento. Uma função que não retorna nada para a função chamadora é do tipo **void** e esse tipo de função pode ser considerado, hierarquicamente, abaixo da main, tendo em vista que ela "faz tudo" mesmo sem receber qualquer argumento.

Vejamos alguns protótipos:

- void asteriscos();
- void menu();
- void calculaReajuste();

Atenção

Você deve ter percebido que nada foi colocado entre os parênteses, mas poderia, embora seja dispensável, colocar a palavra void.

Sendo assim, o primeiro protótipo poderia ser escrito da seguinte forma: *void asteriscos(void)*.

Como é feita a chamada de uma função sem retorno?

Uma função do tipo void, tendo ou não parâmetros, é chamada pelo nome, isto é, não precisará que um comando lhe anteceda. Se tiver parâmetros, eles estarão presentes entre os parênteses. Vejamos a sintaxe abaixo.

nomeDafunção(...);

Quando lemos o protótipo de algumas funções sem retorno, não conseguimos visualizar todas as operações que serão realizadas, mas isso é natural porque é somente a definição de uma função que apresenta o conjunto de operações que deverão ser realizadas.

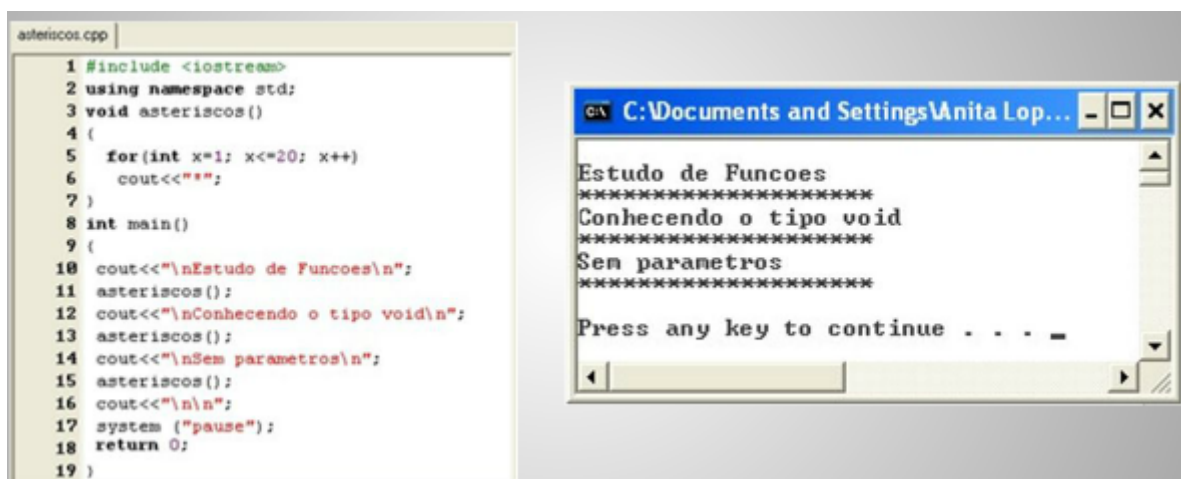
Agora estaremos escrevendo as definições dos três protótipos.

Exemplo 1

Construa uma função que exiba 20 asteriscos.

```
void asteriscos()
{
for(int x=1; x<=20; x++)
cout<<"*";
}
```

Aplicação

The image shows a code editor window on the left and a console window on the right. The code editor, titled 'asteriscos.cpp', contains the following C++ code:

```
1 #include <iostream>
2 using namespace std;
3 void asteriscos()
4 {
5     for(int x=1; x<=20; x++)
6         cout<<"*";
7 }
8 int main()
9 {
10  cout<<"\nEstudo de Funcoes\n";
11  asteriscos();
12  cout<<"\nConhecendo o tipo void\n";
13  asteriscos();
14  cout<<"\nSem parametros\n";
15  asteriscos();
16  cout<<"\n\n";
17  system ("pause");
18  return 0;
19 }
```

The console window, titled 'C:\Documents and Settings\Unita Lop...', displays the output of the program:

```
Estudo de Funcoes
*****
Conhecendo o tipo void
*****
Sem parametros
*****
Press any key to continue . . .
```

Exemplo 2

Construa uma função que exiba um menu com os seguintes itens: pilha, fila, lista, arvore, grafo.

```
void menu()
{
system("cls");

cout<<"\nMenu\n";

cout<<"\n1-Pilha";

cout<<"\n2-Fila";

cout<<"\n3-Lista";

cout<<"\n4-Arvore";

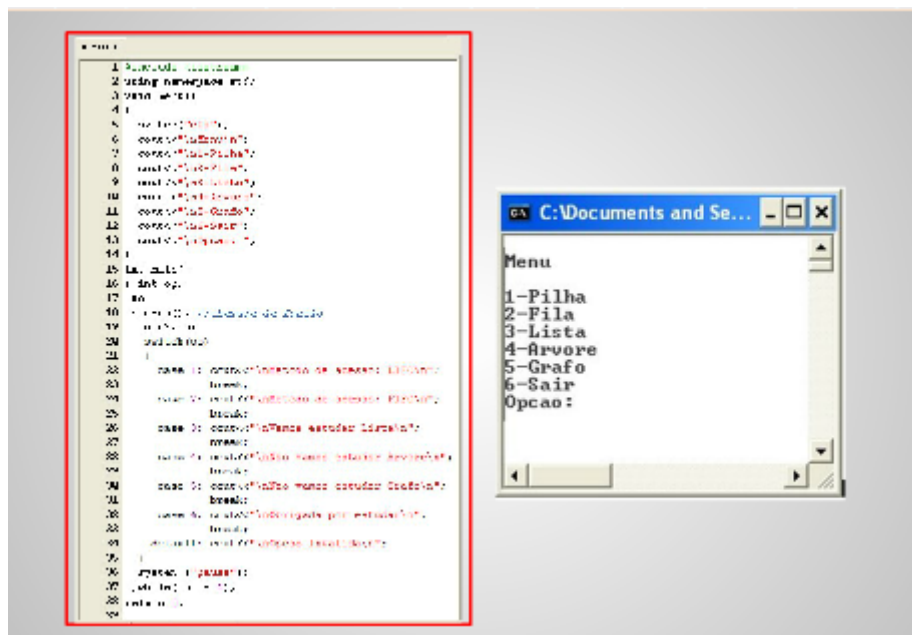
cout<<"\n5-Grafo";

cout<<"\n6-Sair";

cout<<"\nOpcao: ";

}
```

Aplicação



Exemplo 3

Construa uma função que calcule o reajuste salarial de uma pessoa, tendo em vista um valor de resjute.

```
void reajuste()
{
float valor, percentual, reajustado;
```



```

cout<<"\nDigite o valor que devera ser reajustado R$ ";

cin>>valor;

cout<<"\nDigite o valor do percentual de reajuste de 0 a 100: ";

cin>>percentual;

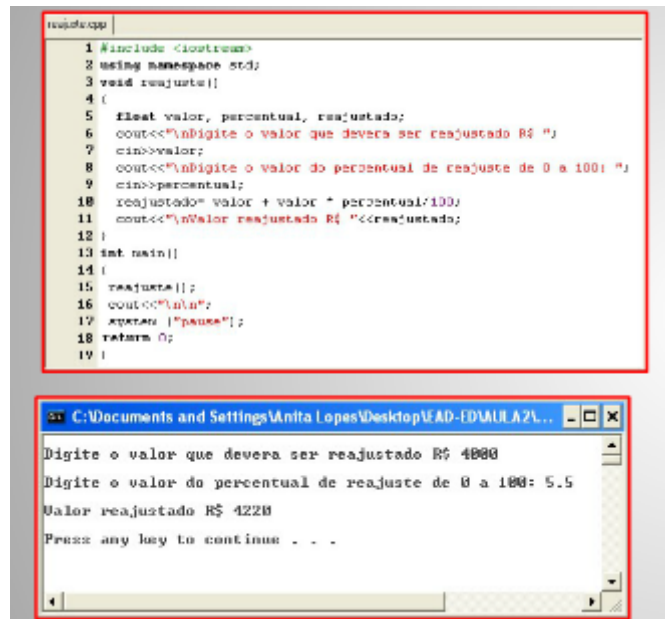
reajustado= valor + valor * percentual/100;

cout<<"\nValor reajustado R$ "<<reajustado;

}

```

Aplicação



Atenção

Optamos por posicionar as funções antes da main porque nos três exemplos só construímos uma função, mas daqui para frente, todos os exemplos apresentarão os protótipos das funções antes da main e as definições depois.

Função com parâmetros

Para que possamos trabalhar com funções com parâmetros, primeiro precisamos saber como a passagem para esses parâmetros é feita. Duas são as formas básicas de passagem dos parâmetros:

Passagem por valor

Na passagem por valor, uma cópia do valor, ou valores, é passada para os parâmetros formais da função chamada através dos parâmetros reais da função chamadora. Sendo que os parâmetros reais poderão estar representados por variáveis ou constantes.

A função chamada poderá operar esses valores, mas não alterará os valores da função chamadora.

Concordamos que para alguns poderá parecer confuso e, por essa razão vamos exemplificar com algo do cotidiano.

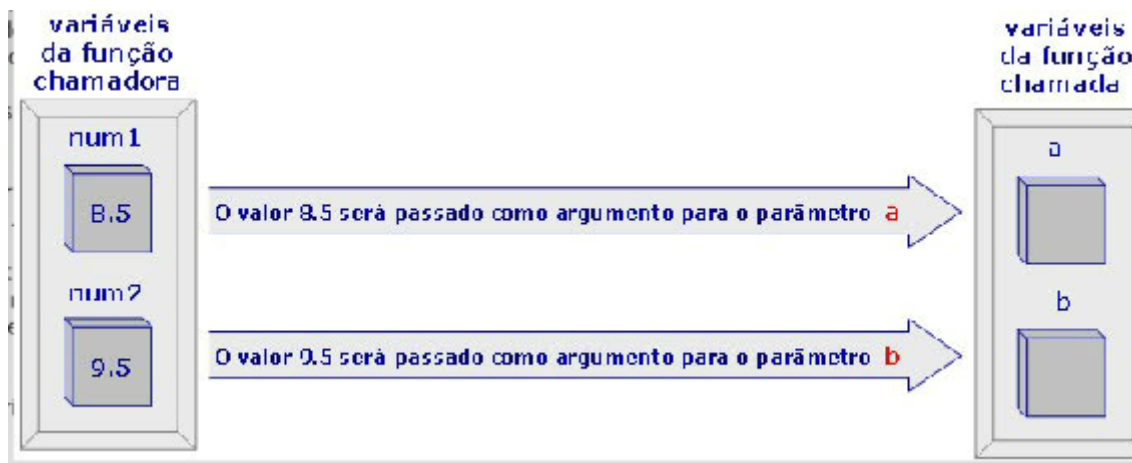
Suponha um autor de livro e um leitor. Ao tentar entender a solução de um problema, o leitor percebe que tem um erro. Ele corrige o erro, mas não corrige o original que está na editora, visto que ele recebeu uma cópia do livro.

As funções pré-definidas que você viu em Algoritmos eram todas com passagem por valor.

Funções com passagem por valor podem, ou não ter retorno.

Se a função tiver retorno, lembre-se de que uma função só poderá retornar um valor para a função chamadora.

Na figura abaixo, você poderá observar como os valores são passados para os parâmetros da função chamada:



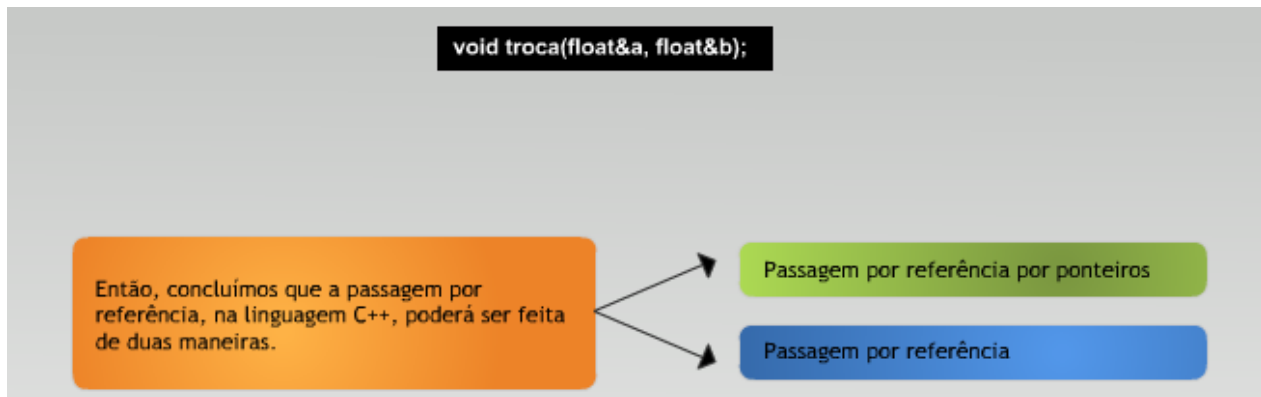
Passagem por referência

Na passagem por referência, o endereço da variável da função chamadora é passado para a função chamada e dessa forma, o valor poderá ser alterado. Na linguagem C, essa passagem tinha que ser feita através do uso da variável ponteiro e acarretava, quando não operado corretamente, em muitos problemas. Tendo em vista os problemas causados pela manipulação dos ponteiros, a linguagem C++ criou um tipo novo de dado, chamado referência que nada mais é do que uma forma de atribuir um nome alternativo (apelido ou alias) para um objeto. Esse conceito aplicado aos parâmetros formais, parâmetros da função chamada, possibilitará que a função chamada receba o endereço do valor da função chamadora, alterando-o.

Esse tipo de passagem ficou mais simples do que o uso de ponteiros porque o compilador se responsabiliza por tudo. Como indicar uma referência?

O operador & símbolo de endereço na linguagem C é usado com outra conotação na linguagem C++ como afirma Saade, Joel: " ... assume o papel de declarador de referências"(pág. 112).

Vamos usá-lo nos parâmetros formais da função. Observe o protótipo abaixo.



Passagem por referência por ponteiros

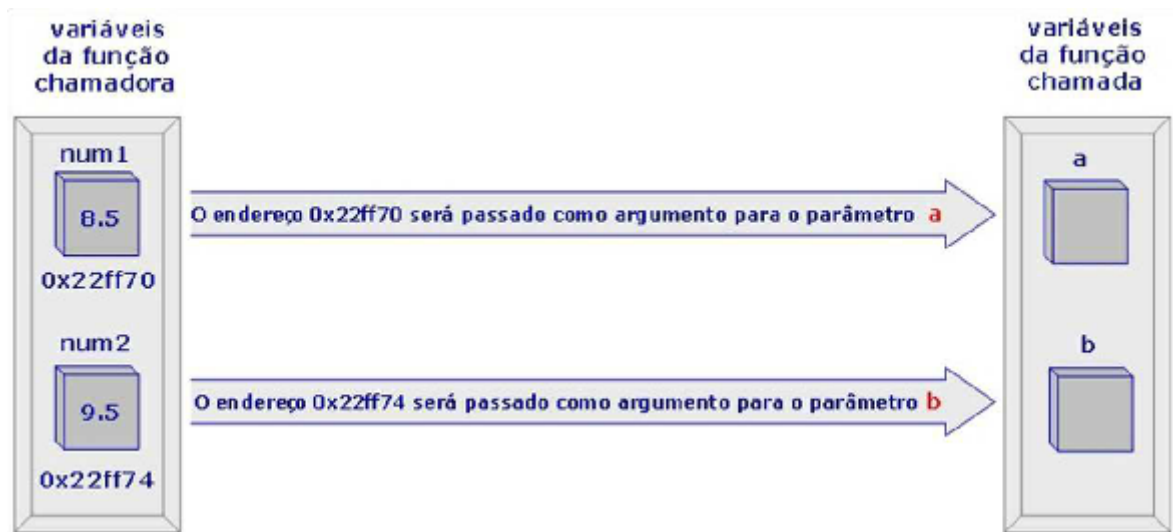
Não se preocupe com os nomes usados por alguns autores para se referenciar a esse tipo: passagem por nome ou passagem por endereço. O importante é que o uso de uma variável do tipo ponteiro é obrigatório. Embora a variável do tipo ponteiro não seja objeto de estudo aqui, vale a pena dizer que uma variável do tipo ponteiro não guarda dado, mas o endereço de uma variável.

Suponha uma variável inteira de nome `a` e uma variável inteira do tipo ponteiro de nome `pa`. Assuma que a variável `pa` aponta para a variável `a`, significando que o conteúdo de `pa` é o endereço de `a`. Trabalhar com variável do tipo ponteiro implica no uso de dois operadores: `&` (endereço) e `*` (conteúdo do endereço apontado por `pa`). Observe a figura ao lado.



Passagem por referência

É a passagem que usa o tipo referência, já definida nesse tópico.



Atenção

Referência não é ponteiro, mas ambos manipulam endereço. Filosofia pura.

Vamos agora analisar alguns protótipos de funções, tendo em vista os novos conceitos estudados.

FUNÇÃO	SIGNIFICADO
<code>void linha(char c, int n);</code>	Função que recebe dois argumentos por passagem de valor. Um do tipo char e outro do tipo int, mas não retorna nada para função chamadora.
<code>int descobrelidade(int anoAtual, int anoNas);</code>	Função que recebe dois argumentos por passagem de valor. Os dois do tipo inteiro e retorna, para a função chamadora, um valor inteiro.
<code>float areaRetangulo(float, float);</code>	Função que recebe dois argumentos reais e retorna, para a função chamada, um valor real.
<code>void troca(float& , float&)</code>	Função que recebe dois argumentos que são endereços que armazenam números reais, por passagem por referência. A função não retorna nada para a função chamadora.

Atenção

Embora possamos simplificar o protótipo de uma função não escrevendo os nomes dos parâmetros como foi feito nos dois últimos exemplos, talvez seja um bom hábito sempre escrevê-los.

Como é feita a chamada de uma função que retorna valor?

Uma função que retorna valor deverá ser chamada através de um comando, não importando se ela tem, ou não, parâmetros.

<code>variavel = nomeDaFunção(...);</code>
<code>cout<<nomeDaFunção(...);</code>
<code>if(nomeDaFunção(...) ...)</code>

Atenção

Fique atento. Uma função só poderá retornar um valor!

O comando return

Uma função que retorna valor terá, obrigatoriamente esse comando no corpo da função. Isso não quer dizer que não poderá ter mais de um comando return no corpo da função.

Quando existir múltiplas respostas, poderemos ter vários tipos de retorno.

return...;

Pode estar presente uma variável, uma constante, uma expressão.

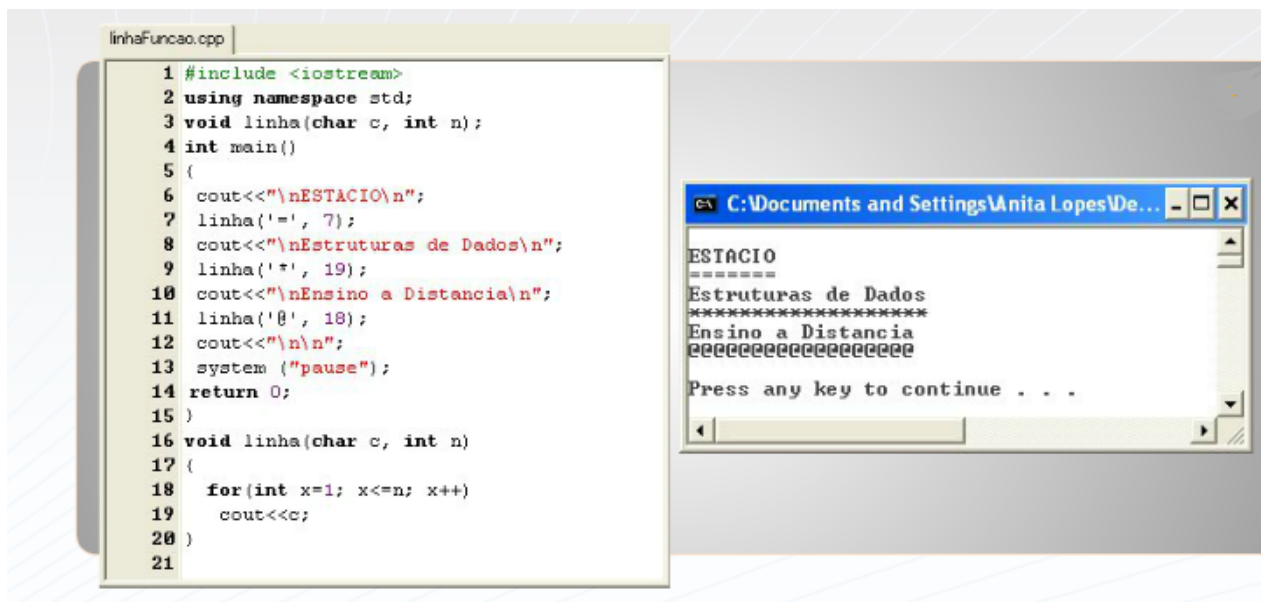
Agora estaremos escrevendo as definições dos 4 protótipos.

Exemplo 1

Construa uma função que receba valores que correspondem ao caracter e à quantidade de vezes que se deseja exibi-lo.

```
void linha(char c, int n)
{
    for(int x=1; x<=n; x++)
        cout<<c;
}
```

Aplicação

The image shows a code editor window titled 'linhaFuncao.cpp' on the left and a console window on the right. The code in the editor defines a function 'linha' that prints a character 'c' 'n' times, and a 'main' function that calls it with various characters and counts. The console window shows the output of the program, which prints 'ESTACIO' followed by a line of 7 dashes, 'Estruturas de Dados' followed by a line of 19 asterisks, 'Ensino a Distancia' followed by a line of 18 empty characters, and then a pause message.

```
1 #include <iostream>
2 using namespace std;
3 void linha(char c, int n);
4 int main()
5 {
6     cout<<"\nESTACIO\n";
7     linha('-', 7);
8     cout<<"\nEstruturas de Dados\n";
9     linha('*', 19);
10    cout<<"\nEnsino a Distancia\n";
11    linha(' ', 18);
12    cout<<"\n\n";
13    system("pause");
14    return 0;
15 }
16 void linha(char c, int n)
17 {
18     for(int x=1; x<=n; x++)
19         cout<<c;
20 }
21
```

C:\Documents and Settings\Anita Lopes\De...
ESTACIO
=====
Estruturas de Dados

Ensino a Distancia

Press any key to continue . . .

Exemplo 2

Construa uma função que receba valores que correspondem ao ano atual e ao ano de nascimento de uma pessoa, retornando a idade que a pessoa terá até 31 de dezembro.

```
int descobreIdade(int anoAtual, int anoNas)
{
    return anoAtual - anoNas;
}
```

Aplicação

```
descobreIdadeFuncao.cpp
1 #include <iostream>
2 using namespace std;
3 int descobreIdade(int anoAtual, int anoNas);
4 int main()
5 {
6     int anoA, anoN;
7     cout<<"\nDigite ano atual: ";
8     cin>>anoA;
9     cout<<"\nDigite ano de nascimento: ";
10    cin>>anoN;
11    cout<<"\nVoce tera ate 31 de dezembro de "<<anoA<<" "<<descobreIdade(anoA, anoN)<<" anos";
12    cout<<"\n\n";
13    system ("pause");
14    return 0;
15 }
16 int descobreIdade(int anoAtual, int anoNas)
17 {
18     return anoAtual - anoNas;
19 }
20
```

C:\Documents and Settings\Anita Lopes\Desktop\LEAD-E...
Digite ano atual: 2010
Digite ano de nascimento: 1989
Voce tera ate 31 de dezembro de 2010 21 anos
Press any key to continue . . .

Exemplo 3

Construa uma função que receba valores que correspondem à base e à altura de um retângulo e retorne a área.

float areaRetangulo(float b, float h)

```
{
return b*h;
}
```

Aplicação

```
areaRetanguloFuncao.cpp
1 #include <iostream>
2 using namespace std;
3 float areaRetangulo(float, float);
4 int main()
5 {
6     int base, altura;
7     cout<<"\nDigite a base de um retangulo: ";
8     cin>>base;
9     cout<<"\nDigite a altura de um retangulo: ";
10    cin>>altura;
11    cout<<"\nArea: "<<areaRetangulo(base, altura)<<" m2";
12    cout<<"\n\n";
13    system ("pause");
14    return 0;
15 }
16 float areaRetangulo(float b, float h)
17 {
18     return b*h;
19 }
20
```

C:\Documents and Settings\Anita Lopes\Desktop...
Digite a base de um retangulo: 3
Digite a altura de um retangulo: 4
Area: 12 m2
Press any key to continue . . .

Exemplo 4

Construa uma função que possa trocar valores de duas variáveis.

```
void troca(float& a, float &b)
```

```
{
```

```
float aux;
```

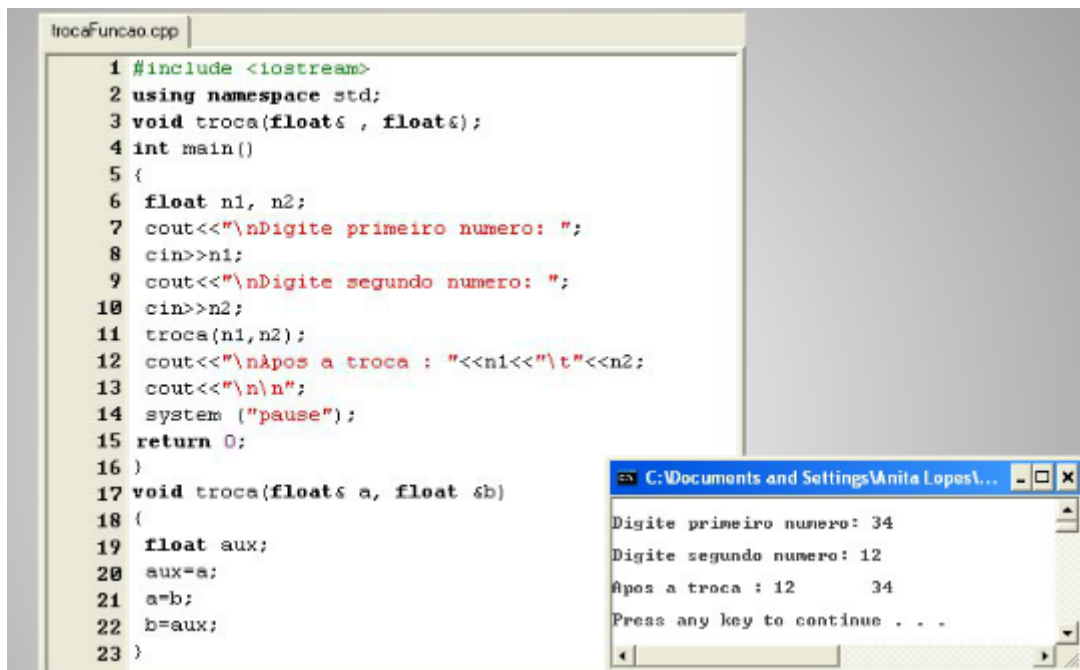
```
aux=a;
```

```
a=b;
```

```
b=aux;
```

```
}
```

Aplicação



The image shows a C++ program in a text editor and its execution in a console window. The program, named 'trocaFuncao.cpp', defines a function 'troca' that swaps two float values using a temporary variable 'aux'. The 'main' function prompts the user to enter two numbers, calls 'troca', and displays the result. The console output shows the user entering 34 and 12, followed by the program outputting 'Após a troca : 12 34' and a pause message.

```
trocaFuncao.cpp
1 #include <iostream>
2 using namespace std;
3 void troca(float& , float&);
4 int main()
5 {
6     float n1, n2;
7     cout<<"\nDigite primeiro numero: ";
8     cin>>n1;
9     cout<<"\nDigite segundo numero: ";
10    cin>>n2;
11    troca(n1,n2);
12    cout<<"\nApós a troca : "<<n1<<"\t"<<n2;
13    cout<<"\n\n";
14    system ("pause");
15    return 0;
16 }
17 void troca(float& a, float &b)
18 {
19     float aux;
20     aux=a;
21     a=b;
22     b=aux;
23 }
```

C:\Documents and Settings\Wania Lopes\...
Digite primeiro numero: 34
Digite segundo numero: 12
Após a troca : 12 34
Press any key to continue . . .

Uma função, que não seja a main, chamando outra função poderá causar algum problema?

Qualquer função poderá chamar outra função desde que a função chamada já esteja declarada, ou definida. Para responder a essa pergunta, observe o exemplo abaixo onde três funções foram colocadas antes da função main e, no entanto, não compilou.


```
localizandoFuncoes1.cpp
1 #include <iostream>
2 using namespace std;
3
4 void facil()
5 {
6     cout<<"\nAcho que estou entendendo\n";
7 }
8
9 void juntando()
10 {
11     int x;
12     for(x=1; x<=3; x++)
13     {
14         facil();
15         legivel();
16     }
17 }
18
19 void legivel()
20 {
21     cout<<"\nFica muito mais legivel\n";
22 }
23
24 int main()
25 {
26     juntando();
27     cout<<"\n\n";
28     system ("pause");
29     return 0;
30 }
31
```

Compilador	Recursos	Log do Compilador	Debug	Resultados da Busca	Fechar
Linker	Unidade	Mensagem			
15	C:\Documents and Settings\Aula\Lo...	legivel undeclared (first use this function) (Each undeclared identifier is reported only once for each			

Explicando:

- 1- A função *main* chamou a função *juntando* e como essa estava definida antes da *main* não teve problema.
- 2- A função *juntando* chamou a função *fácil* e como essa estava definida antes dela, não teve problema.
- 3- A função *juntando* também chamou a função *legível* e como essa não estava definida antes dela, apareceu a mensagem na hora da compilação, informando que ela não estava declarada.



Eu pensei que, se colocássemos antes da *main*, nunca teria problemas. E agora?



A primeira solução, para esse problema, seria colocar a função *legivel()* antes da função *juntando()*.

```
1 #include <iostream>
2 using namespace std;
3
4 void facil();
5
6 cout<<"\nAcho que estou entendendo\n";
7
8 void legivel()
9 {
10     cout<<"\nFica muito mais legivel\n";
11 }
12
13 void juntando()
14 {
15     int x;
16     for(x=1; x<=3; x++)
17     {
18         facil();
19         legivel();
20     }
21 }
22
23 int main()
24 {
25     juntando();
26     cout<<"\n\n";
27     system("pause");
28     return 0;
29 }
```

```
1 #include <iostream>
2 using namespace std;
3
4 void facil();
5 void juntando();
6 void legivel();
7
8 void facil()
9 {
10     cout<<"\nAcho que estou entendendo\n";
11 }
12
13 void juntando()
14 {
15     int x;
16     for(x=1; x<=3; x++)
17     {
18         facil();
19         legivel();
20     }
21 }
22
23 void legivel()
24 {
25     cout<<"\nFica muito mais legivel\n";
26 }
27
28 int main()
29 {
30     juntando();
31     cout<<"\n\n";
32     system("pause");
33     return 0;
34 }
```


```
Acho que estou entendendo
Fica muito mais legivel
Acho que estou entendendo
Fica muito mais legivel
Press any key to continue - - -
```

Atenção

Embora, a princípio, possa parecer mais simples, é sempre bom declarar as funções porque você não fica limitado à sequência das definições. Para dar maior legibilidade, seria melhor definir depois da main e, para isso, teria que declarar todas as funções. Já vimos que para declarar, é só colocar o protótipo antes da main.



Entendi, mas, se eu desejar continuar colocando as funções antes da *main* e declarar, não ficarei limitado à sequência das definições, não é verdade?



Sim, é verdade, mas estou lhe falando que é melhor posicionar depois da *main*, mas se você declarar, não terá problemas.

Antes de prosseguirmos, apresentaremos dois conceitos que permeiam este estudo.

Variáveis locais

A maioria dos exemplos apresentados até aqui teve variáveis declaradas dentro das funções. A esse tipo de declaração, onde as variáveis só são visualizadas nas funções onde foram declaradas, chamamos de variáveis locais.

Variáveis globais

Variáveis declaradas fora do escopo de todas as funções são chamadas de variáveis globais. Esse tipo de variável poderá ser manipulado por qualquer função.

Veja agora um exemplo de programa usando variável global.

```

global1.cpp
1 #include <iostream>
2 using namespace std;
3
4 float num1, num2; //variáveis globais
5
6 void troca();
7 void dobra();
8
9 int main()
10 {
11     cout<<"\nNúmero 1: ";
12     cin>>num1;
13     cout<<"\nNúmero 2: ";
14     cin>>num2;
15     cout<<"\nInicialmente: "<<num1<<"\t"<<num2;
16     troca();
17     cout<<"\nApós a chamada de troca(): "<<num1<<"\t"<<num2;
18     dobra();
19     cout<<"\nApós a chamada de dobra(): "<<num1<<"\t"<<num2;
20     cout<<"\n\n";
21     system("pause");
22     return 0;
23 }
24 void troca()
25 {
26     dobra();
27     num1+=num2;
28     num2=num1-num2;
29     num1=num1-num2;
30 }
31
32 void dobra()
33 {
34     num1*=2;
35     num2*=2;
36 }

```

```

C:\Documents and Settings\Unita Lopes\Des...
Número 1: 23
Número 2: 13
Inicialmente: 23      13
Após a chamada de troca(): 26    46
Após a chamada de dobra(): 52    92
Press any key to continue . . .

```



Gostei porque não preciso usar parâmetros nas funções e nem precisa de retorno.



Não se anime, não. Usar variáveis globais poderá lhe trazer problemas na hora de debugar seu programa, pois fica complicado descobrir qual função que está alterando indevidamente. Observe outro exemplo.

```

global1.cpp
1 #include <iostream>
2 using namespace std;
3
4 float num1, num2; //variáveis globais
5
6 void troca();
7 void dobra();
8
9 int main()
10 {
11     cout<<"\nNúmero 1: ";
12     cin>>num1;
13     cout<<"\nNúmero 2: ";
14     cin>>num2;
15     cout<<"\nInicialmente: "<<num1<<"\t"<<num2;
16     troca();
17     cout<<"\nApós a chamada de troca(): "<<num1<<"\t"<<num2;
18     dobra();
19     cout<<"\nApós a chamada de dobra(): "<<num1<<"\t"<<num2;
20     cout<<"\n\n";
21     system("pause");
22     return 0;
23 }
24 void troca()
25 {
26     dobra();
27     num1+=num2;
28     num2=num1-num2;
29     num1=num1-num2;
30 }
31
32 void dobra()
33 {
34     num1*=2;
35     num2*=2;
36 }

```

```

C:\Documents and Settings\Unita Lopes\Des...
Número 1: 23
Número 2: 13
Inicialmente: 23      13
Após a chamada de troca(): 26    46
Após a chamada de dobra(): 52    92
Press any key to continue . . .

```

Explicando

- 1- Como a função `troca()` chama a função `dobra()`, primeiro os números são multiplicados por 2, ficando 46 e 26 e, depois, trocados.
- 2- Quando a função `dobra` foi chamada pela função `main()`, ela já encontrou os valores dobrados e, então, dobrou de novo.
- 3- Talvez o programador tenha errado e não desejava chamar a função `dobra()` novamente, mas até descobrir um erro desse em um programa muito extenso, poderia demorar muito porque, normalmente, os valores não estariam sendo impressos.

Atenção

Variáveis globais deverão ser usadas com muita cautela.

E como fica a análise da função `main()`?

Sua preocupação tem justificativa, visto que, ao estudar, você poderá encontrar os mais variados cabeçalhos para a função `main` em livros com grande aceitação no mercado. Vejamos alguns exemplos.

<code>void main(void)</code>	Programando em C/C++ "A Bíblia"(pág.706, ano 1998)
<code>main()</code>	C++ para leigos(pág.8, ano 1999)
<code>int main()</code>	Aprenda em 24 horas C++(pág.73, ano 1998)
<code>int main(int argc, char *argv[])</code>	Programando em C++(pág. 123, ano ???)

Ao longo dos anos, vários ajustes foram feitos para que existisse um padrão. O tipo `int` foi escolhido embora pudesse ser omitido. Mais tarde, aconteceu a remoção do `int` implícito. Sendo assim, o cabeçalho da `main` passou a ser:

`int main()`

Já vimos que é opcional o uso da palavra `void` entre os parênteses quando a função não tem parâmetros.

Se a `main` é do tipo `int`, então tem retorno. Porque funciona sem `return`?

A `main` é uma função chamada pelo sistema operacional e, ao finalizar, retorna, automaticamente, para o SO. Sendo pontual, o `return` da `main` poderia ser dispensável, visto que ele serve para informar que o programa foi executado com sucesso e isso, podemos constatar. Entretanto seu retorno poderá ser usado pelo sistema operacional, mas isso é uma outra história.

Dica

Um bom hábito é escrever `return 0;` ou `return EXIT_SUCCESS;` em projetos maiores.

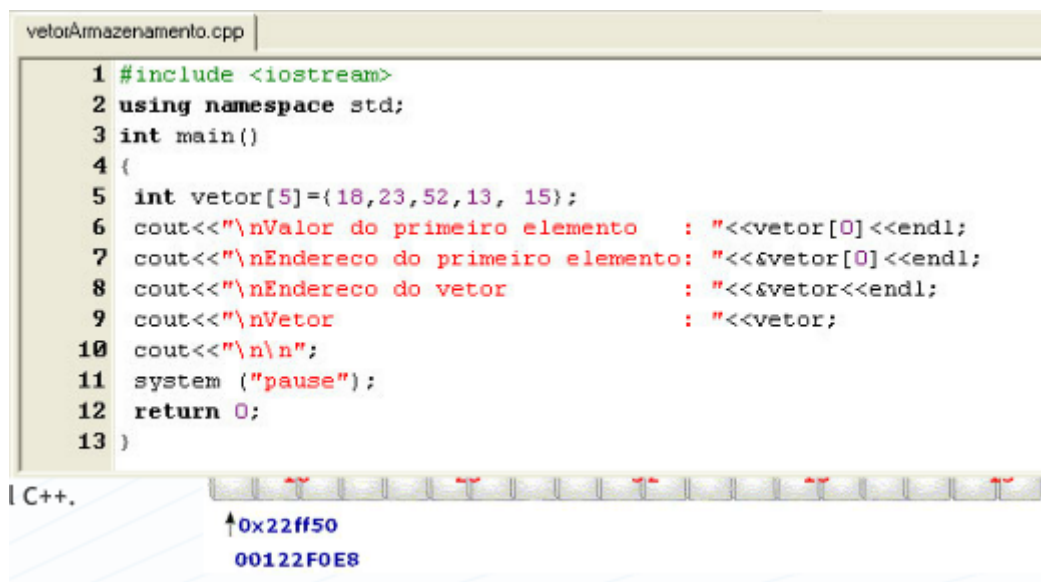
Passando uma estrutura de dados homogênea para uma função

Nossos exemplos só abordaram passagens de valores, ou de endereços, de variáveis numéricas, mas agora vamos aprender como passamos para uma função uma estrutura de dados homogênea. Em particular, uma matriz unidimensional (vetor).

Atenção

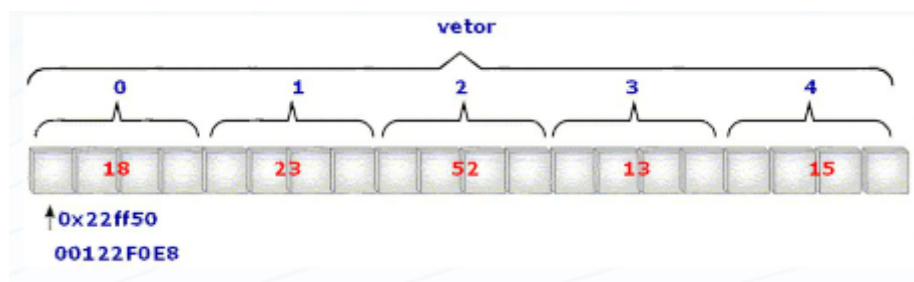
Na disciplina de Algoritmos, na Aula 8, estudamos vários conceitos que serão importantes agora. Releia seus apontamentos ou, assista ao ppt da referida aula, se você tiver feito o download.

Um dos conceitos dizia respeito ao armazenamento na MP de um vetor. Falamos que só era armazenado o primeiro endereço do primeiro elemento do vetor. Vejamos as figuras do programa, do armazenamento na MP e das saídas do programa compilado no Dev-CPP e no Visual C++.

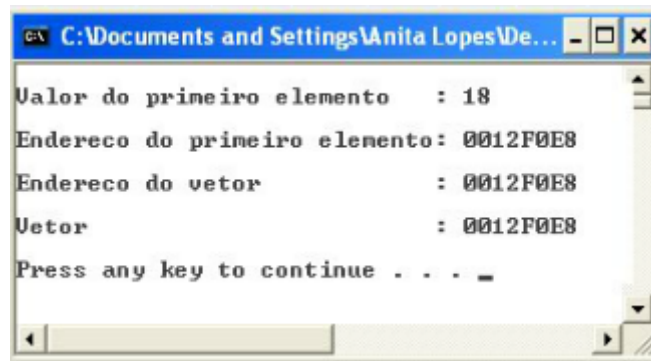


```
vetorArmazenamento.cpp
1 #include <iostream>
2 using namespace std;
3 int main()
4 {
5     int vetor[5]={18,23,52,13, 15};
6     cout<<"\nValor do primeiro elemento  : "<<vetor[0]<<endl;
7     cout<<"\nEndereco do primeiro elemento: "<<&vetor[0]<<endl;
8     cout<<"\nEndereco do vetor          : "<<&vetor<<endl;
9     cout<<"\nVetor                      : "<<vetor;
10    cout<<"\n\n";
11    system ("pause");
12    return 0;
13 }
```

↑ 0x22ff50
00122F0E8

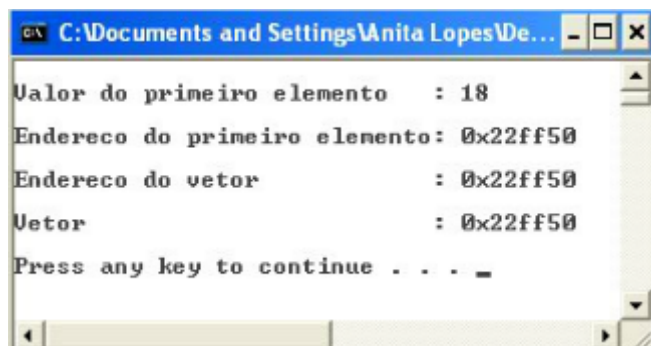


Saída (compilado no Dev-cpp)



```
C:\Documents and Settings\Anita Lopes\De...  
Valor do primeiro elemento : 18  
Endereco do primeiro elemento: 0012F0E8  
Endereco do vetor : 0012F0E8  
Vetor : 0012F0E8  
Press any key to continue . . . _
```

Saída (compilado no Visual C++)



```
C:\Documents and Settings\Anita Lopes\De...  
Valor do primeiro elemento : 18  
Endereco do primeiro elemento: 0x22ff50  
Endereco do vetor : 0x22ff50  
Vetor : 0x22ff50  
Press any key to continue . . . _
```

Considerações:

- 1- O endereço do 1º elemento do vetor é igual ao endereço do vetor.
- 2- Obtém-se o mesmo valor quando se pede para exibir o endereço do vetor e exibir o valor do vetor.

Atenção

Existe uma relação direta entre o nome de um vetor e seu endereço inicial na Memória Principal. Sendo assim, se passarmos o nome de um vetor como parâmetro real de uma função, estaremos passando o endereço inicial desse vetor.

Vamos agora analisar alguns protótipos com passagem de vetores para funções.

PROTÓTIPO	SIGNIFICADO
<code>double media(double [], int tam);</code>	Função que recebe dois argumentos. O primeiro é o endereço de um vetor do tipo real de dupla precisão (double). Enquanto que o segundo, por passagem de valor, recebe um número inteiro que corresponde ao tamanho do vetor. A função retorna um valor de dupla precisão.
<code>int contaMaioresQueH(double [], int tam, double altP);</code>	Função que recebe três argumentos. O primeiro é o endereço de um vetor do tipo real de dupla precisão (double). O segundo, por passagem de valor, recebe um número inteiro que corresponde ao tamanho do vetor e o terceiro, recebe, por passagem de valor, um número de dupla precisão (double) que corresponde a uma altura específica. A função retorna um valor inteiro.
<code>void arredonda(double a[], int tam);</code>	Função que recebe dois argumentos. O primeiro é o endereço de um vetor do tipo real de dupla precisão (double). Enquanto que o segundo, por passagem de valor, recebe um número inteiro que corresponde ao tamanho do vetor. A função não retorna valor.
<code>void dobro(double [], double [], int tam);</code>	Função que recebe três argumentos. Os dois primeiros são endereços de dois vetores do tipo real de dupla precisão (double). Enquanto que o terceiro, por passagem de valor, recebe um número inteiro que corresponde ao tamanho do vetor. A função não retorna valor.

Atenção

Não pense que é fácil, baseada no protótipo de uma função, identificar sua finalidade ou até mesmo os objetivos dos parâmetros.

Agora estaremos escrevendo as definições dos quatro protótipos.

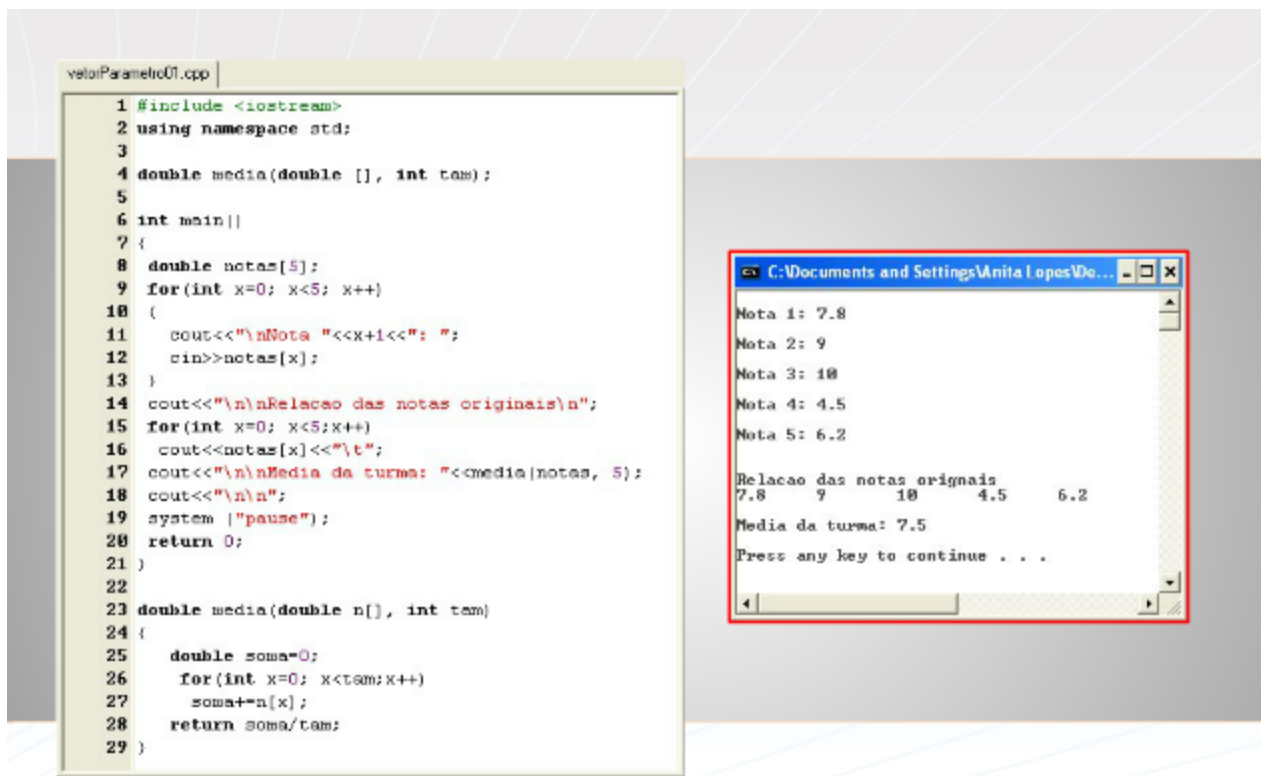
Exemplo 1

Construa uma função que receba o endereço um vetor que armazena notas dos alunos de uma turma e seu tamanho, retornando a média da turma.

```
double media(double n[], int tam)
```

```
{
double soma=0;
for(int x=0; x<tam;x++)
soma+=n[x];
return soma/tam;
}
```

Aplicação



Exemplo 2

Construa uma função que receba o endereço de um vetor que armazena alturas de um grupo de atletas, seu tamanho e a altura mínima de corte, retornando a quantidade de atletas com a altura superior ou igual ao corte.

int contaMaioresQueH(double alts[], int tam, double altP)

```

{
    int conta=0;

    for(int x=0; x<tam; x++)
        if(alts[x]>=altP)
            conta++;

    return conta;
}

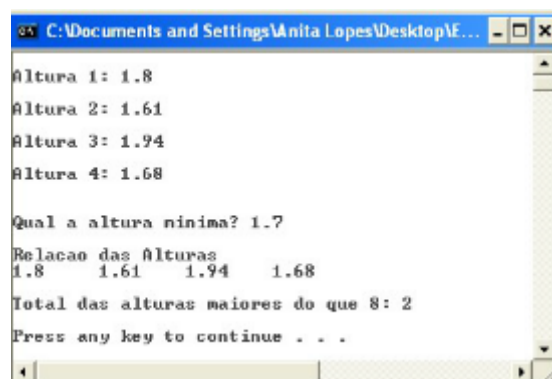
```

Aplicação


```

vetorParametro02.cpp
1 #include <iostream>
2 using namespace std;
3
4 int contaMaioresQueH(double [], int tam, double altP);
5
6 int main()
7 {
8     double alturas[4], procuraAltura;
9     for(int x=0; x<4; x++)
10     {
11         cout<<"\nAltura "<<x+1<<" ";
12         cin>>alturas[x];
13     }
14     cout<<"\nQual a altura minima? ";
15     cin>> procuraAltura;
16     cout<<"\nRelacao das Alturas\n";
17     for(int x=0; x<4;x++)
18         cout<<alturas[x]<<"\t";
19     contaMaioresQueH(alturas, 4,procuraAltura);
20     cout<<"\nTotal das alturas maiores do que 0: "<<contaMaioresQueH(alturas,4, procuraAltura);
21     cout<<"\n\n";
22     system ("pause");
23     return 0;
24 }
25
26 int contaMaioresQueH(double alts[], int tam, double altP)
27 {
28     int conta=0;
29     for(int x=0; x<tam;x++)
30         if(alts[x]>=altP)
31             conta++;
32     return conta;
33 }

```



Exemplo 3

Construa uma função que receba o endereço um vetor que armazena as notas da AV1 de uma turma e seu tamanho.

A função deverá arredondar as notas para cima.

```
void arredonda(double a[], int tam)
```

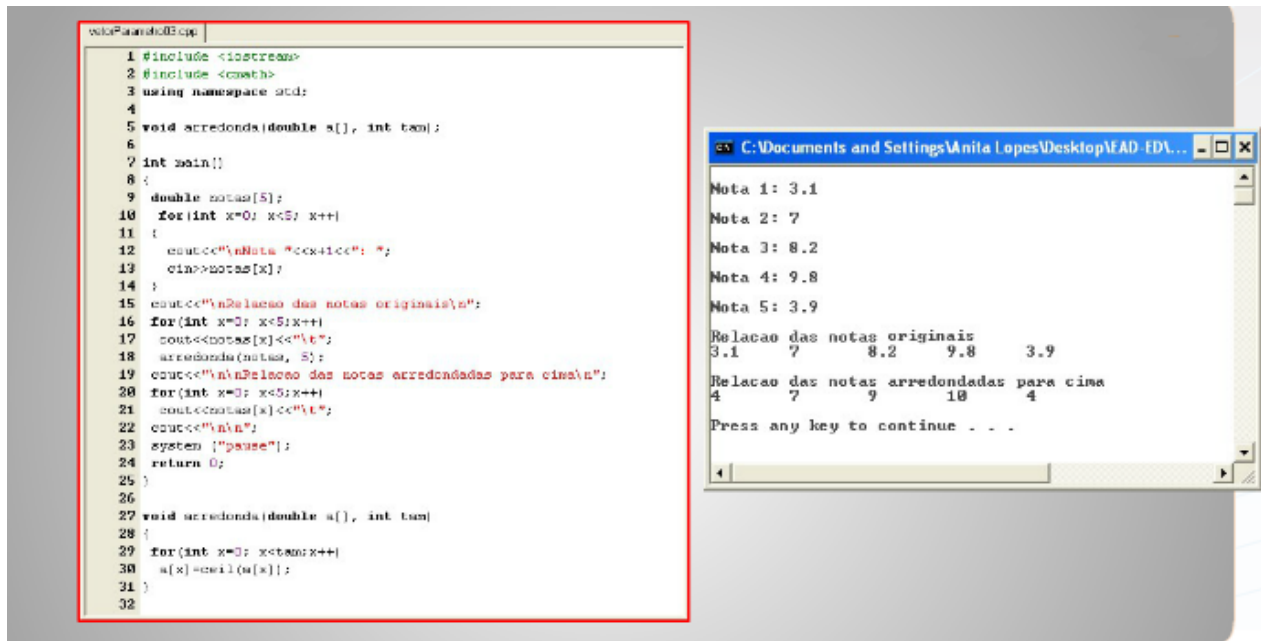
```
{
```

```
for(int x=0; x<tam;x++)
```

```
a[x]=ceil(a[x]);
```

}

Aplicação



Exemplo 4

Construa uma função que receba endereços de dois vetores e o tamanho deles.

A função deverá gerar o vetor dobro.

```
void dobro(double v1[], double v2[], int tam)
```

```
{
```

```
for(int x=0; x<tam;x++)
```

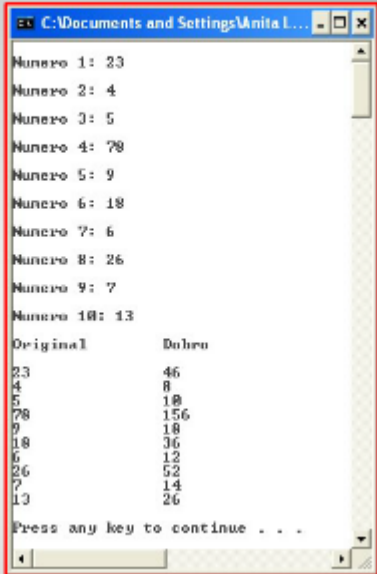
```
v2[x]= v1[x] * 2;
```

```
}
```

Atenção

Um dos vetores estará vazio, mas temos que passar seu endereço para que ele possa ser gerado.

```
vetoParametro04.cpp
1 #include <iostream>
2 #define Tamanho 10
3 using namespace std;
4
5 void dobro(double [], double [], int tam);
6
7 int main()
8 {
9
10     double numeros[Tamanho], numerosDobro[Tamanho];
11     for(int x=0; x<Tamanho; x++)
12     {
13         cout<<"\nNumero " <<x+1<<" : ";
14         cin>>numeros[x];
15     }
16     dobro(numeros,numerosDobro,Tamanho);
17     cout<<"\nOriginal\tDobro\n";
18     for(int x=0; x<Tamanho;x++)
19         cout<<"\n"<<numeros[x]<<"\t\t"<<numerosDobro[x];
20
21     cout<<"\n\n";
22     system ("pause");
23     return 0;
24 }
25
26 void dobro(double v1[], double v2[], int tam)
27 {
28     for(int x=0; x<tam;x++)
29         v2[x]= v1[x] * 2;
30 }
```



Para finalizar, gostaríamos que você desse uma olhada nesse exemplo:

```
vetoParametroBitlicteca.cpp | mirhas h
1 #include <iostream>
2 #include "minhas.h" ←
3 using namespace std;
4 int main()
5 {
6     double notas[5]={6.5,9.2,7.9,8.3, 10};
7     cout<<"\nRelacao das notas originais\n";
8     for(int x=0; x<5;x++)
9         cout<<notas[x]<<"\t";
10     arredonda(notas, 5);
11     cout<<"\n\nRelacao das notas arredondadas para cima\n";
12     for(int x=0; x<5;x++)
13         cout<<notas[x]<<"\t";
14     cout<<"\n\nMedia da turma: " <<media(notas, 5);
15     cout<<"\n\nTotal de notas maiores do que 8: " <<contaMaioresQue8(notas, 5);
16     cout<<"\n\n";
17     system ("pause");
18     return 0;
19 }
```

```
C:\Documents and Settings\Unita Lopes\Deskto...
Relacao das notas originais
6.5    9.2    7.9    8.3    10
Relacao das notas arredondadas para cima
7      10     8      9      10
Media da turma: 8.8
Total de notas maiores do que 8: 3
Press any key to continue . . .
```

Nesta aula, iniciamos o estudo sobre o uso de funções. Esse conteúdo é básico para as aplicações sobre as Estruturas de Dados que serão estudadas nessa disciplina. Reconhecemos que muitas Informações foram estudadas, mas não poderia ser diferente. Por isso, esperamos que você leia, e releia, com cuidado todo esse conteúdo e, se algo não ficar muito claro, peça ajuda ao seu professor.

A disciplina de Estrutura de Dados é de maior importância para seu curso e temos consciência de que não é uma disciplina fácil, visto que o nível de abstração é muito grande, mas, com muita dedicação da sua parte e o apoio dos professores, juntos iremos vencer esse desafio. Até a próxima aula.

Saiba mais



No arquivo fonte, nenhuma função está presente. A inclusão de uma biblioteca possibilitou isso. Veja como fazer clicando aqui!

http://estaciODOcente.webaula.com.br/cursos/gon119/docs/02ED_aula02_doc04.pdf

O que vem na próxima aula

Na próxima aula, você estudará sobre os seguintes assuntos:

- Estruturas de dados heterogêneos;
- Funções e as estruturas de dados heterogêneos.

CONCLUSÃO

Nesta aula, você:

- Compreendeu a importância da modularizar nossos programas;
- Aprendeu as diferenças entre funções por passagem de valor e por passagem de referência com suas variações;
- Analisou protótipos de funções;
- Compreendeu a diferença entre protótipo e cabeçalho;
- Compreendeu a diferença entre declaração e definição de função;
- Implementou funções de todos os tipos;
- Construiu sua biblioteca.