Estruturas de Dados I

Introdução

Prof. Bruno Azevedo

Instituto Federal de São Paulo

INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
SÃO PAULO
Campus Catanduva

Funções

- Funções são blocos de código que realizam tarefas específicas.
- Permitem que você divida um programa em partes menores e mais gerenciáveis.
- Promovem a reutilização de código e a organização, facilitando a sua manutenção.

Funções

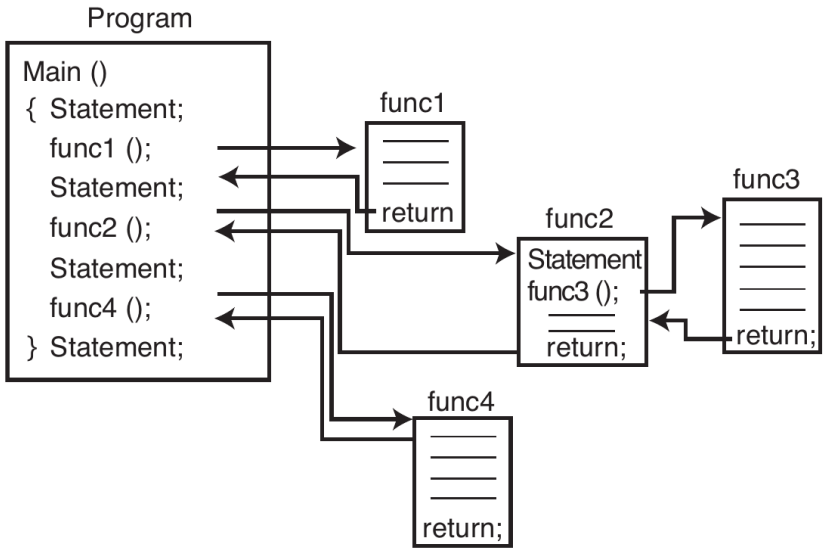
- Funções são úteis para modularizarmos o código, tornando ele mais organizado, e portanto de mais fácil manutenção.
- E possuem mais benefícios, como a possibilidade de reutilização de código.
- Por exemplo, suponha que precisamos fazer algum cálculo complexo, e extenso, durante a execução de nosso programa.
- Podemos simplesmente adicionar este código em nossa função main().
- Entretanto, suponha que precisemos fazer o mesmo cálculo mais 12 vezes em nosso código.
- Teremos que copiar e colocar o mesmo código, idêntico, nos trechos necessários. Isso pode tornar nosso código bem maior, e de mais difícil manutenção.
- Se usarmos funções, podemos simplesmente definir uma função contendo o código que faz esse cálculo, e chamar a função cada vez que precisarmos dele.
- Estes são apenas alguns dos benefícios, existem outros, como por exemplo, o ganho em testabilidade, já que funções independentes podem ser testadas individualmente.

Funções

- Funções bem projetadas executam uma única tarefa específica e facilmente compreensível.
- Tarefas complicadas devem ser divididas em várias funções e, em seguida, cada uma pode ser chamada sequencialmente.
- Temos três tipos principais de funções em C++: as funções definidas pelo programador, as funções fornecidas pela linguagem, e as funções externas, fornecidas por empresas ou terceiros.

Funções

- Todo programa em C++ possui pelo menos uma função, a função main(). Esta é chamada automaticamente quando seu programa inicia.
- A função main() pode chamar outras funções, algumas das quais podem chamar outras funções.
- Cada função tem seu próprio nome e, quando esse nome é encontrado, a execução do programa desvia para o corpo dessa função.
- Isso é conhecido como **chamada da função**.
- Quando a função termina, a execução é retomada na próxima linha da função chamadora.



Funções

- Vamos aprender a criar nossas próprias funções.
- A sintaxe da declaração de função em C++ é:

```
tipo_de_retorno nome_da_funcao(tipoParametro nomeParametro) {
    // corpo da função
    return valorDeRetorno; // opcional
}
```

- Por exemplo:

```
int soma(int a, int b) {
    return a + b;
}
```

Funções

- Uma função pode não possuir retorno. Neste caso, usamos a palavra-chave **void**.
- Por exemplo:

```
void saudacao(string nome) {
    cout << "Olá, seja bem-vindo " << nome << "!"<< endl;
}
```

- Uma função pode não possuir parâmetros.
- Por exemplo:

```
void saudacao() {
    cout << "Olá, seja bem-vindo!" << endl;
}
```

Funções

- Já sabemos criar funções, então vamos aprender a chamar funções em nosso código.
- A sintaxe da chamada de função é a seguinte:

```
nomeDaFuncao(argumentos);
```

- Por exemplo:

```
#include <iostream>
using namespace std;
void saudacao() {
    cout << "Olá, seja bem-vindo!" << endl;
}
int main() {
    saudacao(); // Chamada da função
    return 0;
}
```

Funções

- Variáveis definidas dentro de uma função têm escopo local.
- Ou seja, não podem ser acessadas fora da função.
- Além disso, o ciclo de vida de uma variável local está vinculado ao escopo da função em que foi declarada.
- Considere o seguinte código:

```
int soma(int a) {
    int b = 12;
    return a + b;
}
```

- A variável b existe apenas durante a execução da função. Assim que a função retorna, a variável é desalocada.
- Vamos conversar um pouco sobre este assunto.

Variáveis Locais e Globais

- Variáveis podem ser classificadas como locais ou globais, dependendo de onde são declaradas e onde podem ser acessadas.
- Variáveis Locais:
  - São declaradas dentro de uma função ou bloco.
  - Têm escopo limitado ao bloco ou função em que foram declaradas.
  - São desalocadas ao final do bloco ou função.
  - Não podem ser acessadas de fora do bloco ou função onde foram definidas.

Variáveis Locais e Globais

```
#include <iostream>
using namespace std;
void funcaoExemplo() {
    cout << "Variável a: " << a << endl;
}
int main() {
    int a = 12;
    funcaoExemplo();
    return 0;
}
```

- Esse código gerará um erro de compilação.
- A variável a não está no **escopo** da função funcaoExemplo().
- Ela é uma variável local e está no escopo da função main().

Variáveis Locais e Globais

- Variáveis Globais:
  - São declaradas fora de todas as funções.
  - Têm escopo global, podendo ser acessadas em qualquer lugar do programa.
  - Existem durante toda a execução do programa.
  - Devem ser usadas com cuidado para evitar confusão e problemas de manutenção.
- Para definirmos uma variável como global, ela deve estar fora do escopo de todas as funções, inclusive a main().
- Por exemplo:

```
#include <iostream>
using namespace std;
int variavelGlobal = 10; // Variável global
int main() {
    cout << variavelGlobal << endl;
    return 0;
}
```

- Como ela é global, conseguimos acessar variavelGlobal e imprimir o seu valor dentro do escopo da função main().

Variáveis Locais e Globais

- O código abaixo não gerará um erro de compilação.
- A variável aGlobal é global, portanto pode ser acessado pelo escopo de qualquer função.

```
#include <iostream>
using namespace std;
int aGlobal = 10; // Variável global
void funcaoExemplo() { // Função que utiliza a variável global
    aGlobal +=10;
}
int main() {
    funcaoExemplo();
    cout << aGlobal << endl;
    return 0;
}
```

- Esse programa imprimirá 20, já que a variável aGlobal consegue ser acessada pela funcaoExemplo().

Algumas Razões para Evitar Variáveis Globais

- **Dificuldade de Rastreamento:** Variáveis globais podem ser acessadas de qualquer lugar no programa, tornando difícil rastrear onde e quando elas são modificadas. Isso pode dificultar a depuração de erros e a manutenção do código.
- **Concorrência:** Em programas multithreaded, o uso de variáveis globais pode levar a problemas de concorrência.
- **Acoplamento Forte:** O uso de variáveis globais pode levar a um alto acoplamento (grau de interdependência) entre diferentes partes do código, o que pode tornar o programa menos flexível.
- **Testabilidade:** Código que depende de variáveis globais pode ser mais difícil de testar, pois é difícil isolar unidades individuais de código para testes.

Ponteiros e Funções

- Como não devemos usar variáveis globais, o que podemos fazer para modificar variáveis em diferentes escopos? Para esta finalidade, usaremos ponteiros.
- Vamos estudar o seguinte código:

```
#include <iostream>
using namespace std;
void trocar(int *a, int *b) {
    int temp = *a;
    *a = *b;
    *b = temp;
}
int main() {
    int x = 10, y = 20;
    trocar(&x, &y);//Passa os endereços das variáveis para a função
    cout << "x = " << x << ", y = " << y << endl;
    return 0;
}
```

- Estamos passando os endereços na memória das variáveis locais x e y, que estão no escopo da função main().
- Para receber os endereços, usamos ponteiros como parâmetros da função .
- Deste modo, podemos trabalhar com as variáveis dentro do escopo da função trocar().

Ponteiros e Funções

- Podemos usar ponteiros para passar vetores como parâmetros.

```
#include <iostream>
using namespace std;
void duplicaElementos(int *vetor, int tamanho) {
    for (int i = 0; i < tamanho; i++)
        vetor[i] *= 2;
}
void imprimirVetor(int *vetor, int tamanho) {
    for (int i = 0; i < tamanho; i++)
        cout << vetor[i] << " ";
    cout << endl;
}
int main() {
    const int tamanhoVetor = 5;
    int meuVetor[tamanhoVetor] = {10, 20, 30, 40, 50};
    cout << "Vetor original: ";
    imprimirVetor(meuVetor, tamanhoVetor);
    duplicaElementos(meuVetor, tamanhoVetor);
    cout << "Vetor modificado: ";
    imprimirVetor(meuVetor, tamanhoVetor);
    return 0;
}
```

Referências e Funções

- Também Podemos usar referências para manipularmos variáveis locais em outros escopos.

```
#include <iostream>
using namespace std;
void manipulandoValor(int &rVar) {
    rVar += 10;
}
int main() {
    int var = 10;
    manipulandoValor(var);
    cout << "Var: " << var << endl;
    return 0;
}
```

Exercícios (14)

⇒ Escopo de Variáveis

- Escreva um programa que utilize uma variável global para contar o número de vezes que uma função é chamada.
- Escreva um programa com duas variáveis com o mesmo nome: uma global e uma local. A variável local estará dentro de uma função. Defina um valor para a variável global e modifique o valor da variável local dentro da função. Responda: como o programa se comporta? Explique o seu comportamento.

⇒ Funções

- Escreva uma função que receba dois números inteiros como parâmetros e retorne a soma deles.
- Escreva uma função que calcule e retorne o fatorial de um número inteiro.
- Escreva uma função que receba um vetor de inteiros e retorne o valor mínimo presente no vetor.
- Escreva uma função que receba um vetor como parâmetro e eleve ao quadrado cada um de seus elementos (use ponteiros). Escreva uma função que fará a operação de potenciação e que será chamada pela função anterior.
- Escreva uma função que receba uma matriz como parâmetro e duplique o valor de todos seus elementos (use ponteiros).
- (Desafio) Escreva uma função que receba um vetor como parâmetro e calcule o fatorial cada um de seus elementos. Use a função criada previamente. Esta função deve usar referências em vez de ponteiros para manipulação das variáveis locais.