Estácio

ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

# ESTRUTURA DE DADOS

## AULA 2

RICARDO EIJI KONDO, Me.

# TÓPICOS

**UNIDADE II - Modularização**

2.1 - Definição (Funções e Procedimentos)

2.2 - Passagem de parâmetros

2.2.1 - Passagem de parâmetros por valor

2.2.2 - Passagem de parâmetros por referência

2.3 - Escopo de variáveis e Retorno de valores

# FUNÇÕES

- A **função** pode ser definida como um bloco de código com uma tarefa específica. Ou seja, em vez de copiar o bloco de código em várias partes do programa, colocamos o bloco dentro de uma função, e ao longo do programa, quando necessário, chamamos a função que executará o código automaticamente.
- Os **procedimentos** diferem das funções apenas por não retornarem resultado

```
Tipo_do_Dado_de_Retorno Nome_da_Função(Listas de parâmetros...)
{
    Corpo da função
}
```

- Exemplo: Cálculo do perímetro de uma circunferência

```
float CalcPerimetroCircunferencia(float raio)
{
    float Perimetro;
    Perimetro = 2 * 3.14 * raio ;
    return Perimetro;
}
```

# FUNÇÕES

- **Protótipo da função:** é uma instrução colocada no início do programa, indicando o tipo da função e os parâmetros que recebem. O objetivo do protótipo é fornecer informações, para checagem de erro, ao compilador.

```
float CalcPerimetroCircunferencia(float raio); // Protótipo da função
```

- No protótipo, o nome do parâmetro não é obrigatório, poderia ser escrito como:

```
float CalcPerimetroCircunferencia(float); // Protótipo da função
```

- Exemplo: Cálculo do perímetro de uma circunferência

```
1   #include <iostream>
2   using namespace std;
3
4   //prototipo da função
5   float CalcPerimetroCircunferencia(float);
6
7   // programa principal
8   int main (void){
9       cout<<CalcPerimetroCircunferencia(3);
10  }
11
12  //função
13  float CalcPerimetroCircunferencia(float raio)
14  {
15      float Perimetro;
16      Perimetro = 2 * 3.14 * raio ;
17      return Perimetro;
18  }
```

```
18.84
--------------------
```

- O protótipo pode ser removido, caso a função seja declarada antes de sua chamada. No entanto, recomenda-se a utilização do protótipo.

```
1   #include <iostream>
2   using namespace std;
3
4   //função
5   float CalcPerimetroCircunferencia(float raio)
6   {
7       float Perimetro;
8       Perimetro = 2 * 3.14 * raio ;
9       return Perimetro;
10  }
11
12  // programa principal
13  int main (void){
14      cout<<CalcPerimetroCircunferencia(3);
15  }
```

# TIPOS DE FUNÇÕES

- O tipo de função é definido de acordo com o tipo de valor que ela retorna. No exemplo, como a função retorna o valor do tipo float, ela é dita do tipo float.
- Poderia ser do tipo bool, int, string, etc. de acordo com o tipo de retorno.

```
float CalcPerimetroCircunferencia(...){
...
}
```

# TIPOS DE FUNÇÕES

- Caso não haja nenhum valor de retorno, a função pode ser declarada como sendo do tipo void.
- O void também pode ser utilizado na declaração de parâmetros.

```
void Teste(void){
    cout<<"ola"<<endl;
}
```

# RETURN

- O comando return finaliza a execução da função e volta o controle para a instrução após ao da chamada da função. O comando return pode ser utilizado de três formas:

```
return;
return expressão;
return (expressão);
```

# RETURN

- **Return sem a expressão:** somente pode ser utilizado em função do tipo void

```
1   #include <iostream>
2   using namespace std;
3
4   //prototipo da função
5   void Teste(void);
6
7   // programa principal
8   int main (void){
9       Teste();
10  }
11
12  //função
13  void Teste(void){
14      if (true){ //testar com false
15          return;
16      }
17      cout<<"ola"<<endl;
18  }
```

# RETURN

- **Return com a expressão:** somente pode ser utilizado em função do tipo void

```
float CalcPerimetroCircunferencia(float raio){
    float Perimetro;
    Perimetro = 2 * 3.14 * raio ;
    return Perimetro;
}
```

```
float CalcPerimetroCircunferencia(float raio){
    return (2 * 3.14 * raio);
}
```

```
float CalcPerimetroCircunferencia(float raio){
    return 2 * 3.14 * raio;
}
```

# ESCOPO DE VARIÁVEIS

- **Variáveis locais:** são aquelas nas quais apenas a função onde ela está pode usá-la.
  - Declarada dentro da função.
  - Ao chamamos uma função, também chamamos as variáveis dentro da função. Se finalizamos uma função, também finalizamos a variável.
- **Variáveis globais:** podem ser utilizadas por qualquer função. E qualquer função pode alterar o valor, utilizá-la em um processo ou até mesmo atribuir o valor que quiser.
  - Declarada fora de qualquer função.
  - A variável global está constantemente na memória.

```
1   #include <iostream>
2   using namespace std;
3
4   //prototipo da função
5   void Teste(void);
6   //variaveis globais
7   int _global1=100;
8
9   // programa principal
10  int main (void){
11      int local1=10;
12      Teste();
13      cout<<_global1<<endl;
14      cout<<local1<<endl;
15  }
16  //funções
17  void Teste(void){
18      int local2=20;
19      cout<<_global1<<endl;
20      cout<<local2<<endl;
21  }
```

# PASSAGEM DE PARÂMETROS

- Parâmetros são informações passadas para a função. Uma função pode ter vários valores passados como parâmetros a ser utilizado na função.
- Parâmetros podem ser passados por valor e por referência.

```
bool Empilhar(DADOS_ALUNO Pilha[], int CodAluno, char Nome[],
    int Turma, int &PosTopo) {
    ...
}
```

# PASSAGEM DE PARÂMETROS POR VALOR

- A função cria uma cópia dos dados, a cópia é armazenada em variáveis, que são criadas quando a função é chamada e destruídas quando a função é finalizada.

```
1   #include <iostream>
2   using namespace std;
3
4   void Beep(int); // Protótipo da função
5
6   int main(void){
7       Beep(10); // Chama a função a ser executada
8   }
9
10  //função
11  void Beep(int NroVezes){
12      for(int i = 0; i < NroVezes; i++) {
13          cout << '\x07';
14      }
15  }
```

# PASSAGEM DE PARÂMETROS POR REFERÊNCIA

- A função tem acesso direto às variáveis enviadas, ou seja, altera diretamente a variável.
- Observe a utilização do &.

```
1   #include <iostream>
2   using namespace std;
3
4   // Protótipo da função
5   void Alterar(int&);
6
7   int main(void){
8       int Idade=20;
9       Alterar(Idade);
10      cout << Idade;
11  }
12
13  //função
14  void Alterar(int &paramIdade){
15      paramIdade = 100;
16  }
```

# PASSAGEM DE PARÂMETROS POR REFERÊNCIA

- O que será mostrado na tela nos seguintes códigos?

```cpp
#include <iostream>
using namespace std;

// Protótipo da função
int Alterar(int);

int main(void){
    int Idade=20;
    int IdadeAlterada=Alterar(Idade);
    cout << Idade <<endl;
    cout << IdadeAlterada <<endl;
}

//função
int Alterar(int paramIdade){
    paramIdade = 100;
    return paramIdade;
}
```

```cpp
#include <iostream>
using namespace std;

// Protótipo da função
int Alterar(int&);

int main(void){
    int Idade=20;
    int IdadeAlterada=Alterar(Idade);
    cout << Idade <<endl;
    cout << IdadeAlterada <<endl;
}

//função
int Alterar(int &paramIdade){
    paramIdade = 100;
    return paramIdade;
}
```

# PASSAGEM DE PARÂMETROS POR VALOR E REFERÊNCIA

- Os parâmetros de uma função podem conter, ao mesmo tempo, tanto parâmetros passados por valor como por referência.

```
1   #include <iostream>
2   using namespace std;
3   void CalcularPreco(float, float&);
4
5   int main(void){
6       float VlrProdReal, VlrProdDolar;
7       cout << "Digite o valor em Reais: ";
8       cin >> VlrProdReal;
9       // Chamada da função
10      CalcularPreco(VlrProdReal, VlrProdDolar);
11      cout << "Valor em Dolar: " << VlrProdDolar << endl;
12  }
13
14  void CalcularPreco(float VlrProdutoReal, float &VlrProdutoDolar){
15      VlrProdutoDolar = VlrProdutoReal / 3.8;
16  }
```

# PASSAGEM DE VETOR PARA FUNÇÕES

- Um vetor ou matriz, sempre é passado como referência para uma função, ou seja, a função pode acessar diretamente os seus elementos

```
1   #include <iostream>
2   using namespace std;
3   void GerarValores(float[], float);
4
5   int main(void){
6       float PagtoMes[12];
7       GerarValores(PagtoMes, 150.80);
8       for(int i = 0; i < 12; i++){
9           cout << "Pagamento Mes " << i << ": " << PagtoMes[i] << endl;
10      }
11  }
12
13  void GerarValores(float PagtoM[], float VlrPagto){
14      for(int i = 0; i < 12; i++){
15          PagtoM[i] = VlrPagto;
16      }
17  }
```

# PASSAGEM DE STRUCT PARA FUNÇÕES

```cpp
#include <iostream>
using namespace std;

struct Pag{
    int codigo;
    float valor;
};

void GerarValores(Pag PagtoM){
    PagtoM.codigo = 100;
    PagtoM.valor = 200.2;
    cout<<PagtoM.codigo<< " : "<<PagtoM.valor<<endl;
}

int main(void){
    struct Pag PagtoMes = {10,20.1};
    GerarValores(PagtoMes);
    cout<<PagtoMes.codigo<<" : "<<PagtoMes.valor<<endl;
}
```

```cpp
#include <iostream>
using namespace std;

struct Pag{
    int codigo;
    float valor;
};

void GerarValores(Pag &PagtoM){
    PagtoM.codigo = 100;
    PagtoM.valor = 200.2;
    cout<<PagtoM.codigo<<" : "<<PagtoM.valor<<endl;
}

int main(void){
    struct Pag PagtoMes = {10,20.1};
    GerarValores(PagtoMes);
    cout<<PagtoMes.codigo<<" : "<<PagtoMes.valor<<endl;
}
```

# PASSAGEM DE STRUCT PARA FUNÇÕES

```
1   #include <iostream>
2   using namespace std;
3
4   struct Pag{
5       int codigo;
6       float valor;
7   };
8
9   void GerarValores(struct Pag PagtoM[], float VlrPagto){
10      for(int i = 0; i < 12; i++){
11          PagtoM[i].codigo = i;
12          PagtoM[i].valor = VlrPagto+i;
13      }
14  }
15
16  int main(void){
17      struct Pag PagtoMes[12];
18      GerarValores(PagtoMes, 150.80);
19      for(int i = 0; i < 12; i++){
20          cout << PagtoMes[i].codigo << ": " << PagtoMes[i].valor << endl;
21      }
22  }
```