Funções: Tal como Fábricas

Mário Leite

...

De um modo bem simples, uma função pode ser comparada a uma máquina que é alimentada por um ou mais *insumos*, para realizar um *trabalho* na criação de algum produto. Deste modo, esses insumos são chamados de ***parâmetros*** e o ***resultado*** do processamento pode ser devolvido como ***retorno***. Esta comparação com uma máquina é bem razoável e bem objetiva para um entendimento preliminar sobre este assunto, que é fundamental para todos os programadores na modularização de um programa; e é assim que muitas literaturas definem uma função. Entretanto, eu prefiro fazer analogia com fábricas, porque oferece uma interpretação bem mais lógica para um iniciante em programação. E, particularmente, para os programas codificados em **C**, a analogia com fábricas é bem mais apropriada, embora este raciocínio possa ser aplicado a outros tipos de linguagens e a todos os tipos de paradigmas de programação. Mas, para não estender muito, este texto será apresentado de modo bem objetivo e prático, analisando quatro situações diferentes.

1 - Função com retorno e com parâmetros (figura 1)

Fábrica: Recebe a matéria prima externa (*parâmetros*), cria o produto (*resultado*) e exporta o produto (*retorna*) para seus clientes

Função: Recebe os parâmetros (*matéria prima externa*), calcula o resultado (*produto*) e retorna (*exporta*) para a rotina que a chamou.

Este é o tipo “normal” de uma fábrica, equivalente ao tipo mais comum de função; não só em **C**, mas, na maioria das linguagens de programação. A **figura 1.1** mostra um exemplo de código para esse tipo de função.

2 - Função com parâmetros e sem retorno (figura 2)

Fábrica: Recebe a matéria prima externa (*parâmetros*), cria o produto (*resultado*), mas, não o exporta (*retorna*) para seus clientes; o produto é consumido internamente.

Função: Recebe os parâmetros **(***matéria prima externa*), calcula o resultado (*produto*), mas, não exporta (*retorna*) para a rotina que a chamou; o resultado é utilizado dentro da própria função.

Esse tipo de função recebe parâmetros, produz o resultado desejado pela programação, porém, esse resultado não é devolvido para a rotina que a chamou (não tem *retorno*); por isto seu tipo é *void*. Então, se uma função desse tipo for chamada num programa, o resultado deve ser exibido “dentro” dela mesmo, pois, esse resultado não é conhecido em nenhum outro local do programa (é como se comportam as *procedures* em algumas linguagens de programação). A **figura 2.1** exemplifica como esse tipo de função pode ser utilizado em **C**.

3 - Função sem parâmetros e com retorno (figura 3)

Fábrica: Utiliza sua própria matéria prima (*dados internos*), cria o produto (*resultado*) e o exporta para seus clientes.

Função: Obtém os dados dentro nela mesmo (*matéria prima interna*), calcula o resultado (*produto*) e o retorna para a rotina que a chamou.

Nesse tipo de função ela mesma “gera” seus dados (*matéria prima interna*) e exporta (*retorna*) o resultado (*produto*) para a rotina que a chamou. Deste modo, não existem parâmetros externos a receber; o processamento ocorre com os dados obtidos pela própria função para realizar a tarefa. Um tipo de função assim poderia ser utilizado no caso em que se deseja, por exemplo, calcula a soma dos ***n*** primeiros números pares, com o valor de ***n*** solicitado pela própria função. O código da **figura 3.1** é um exemplo dessa situação.

4 - Função sem retorno e sem parâmetros (figura 4)

Fábrica: Utiliza sua própria matéria prima (*dados internos*), cria o produto (*resultado*), mas, não exporta para seus clientes; a produção é para consumo interno.

Função: Obtém os parâmetros dentro dela mesmo (*matéria prima interna*), calcula o resultado (*produto*), mas, não *retorna* para a rotina que a chamou. Um exemplo de aplicação desse tipo de função seria o caso de ler uma série de números e exibir, dentro dela mesmo, apenas os ímpares; o fim das leituras poderia ocorrer quando fosse digitado o número **0**. Entretanto, embora possa ser utilizado, esse tipo não está de acordo com o objetivo principal de uma função, que é: **”*receber dados*, *processar* e *devolver um resultado*”**. Na verdade, funções com esse tipo se comportamento funciona como um programa autônomo (como uma *procedure* que não recebe parâmetros). Um exemplo de uso desse tipo de função é mostrado na **figura 4.1**.

```
tipo Função1(tipo1 par1, tipo2 par2, tipo3 par3...);
{
  /* Declaração de variáveis locais */
  ...
  ...
  return (valor);
 }
```

Figura 1 - Função com retorno e com parâmetros

```
void Funcao2(tipo1 par1, tipo2 par2, tipo3 par3,...);
{
  /* Declaração de variáveis locais */
  ...
  ...
  ...
 }
```

Figura 2 - Função com parâmetros e sem com retorno

```
tipo Funcao3();
{
  /* Declaração de variáveis locais */
  //Obtém dados para processar
  ...
  ...
  return (valor);
}
```

Figura 3 - Função com retorno e sem parâmetros

```
void Funcao4();
{
  /* Declaração de variáveis locais */
  //Obtém dados para processar
  ...
  ...
  ...
}
```

Figura 4 - Função sem retorno e sem parâmetros

```
long int Fatorial(int N){
  int i;
  long int fat=1;
  for(i=1; i<=N; i++)
     fat = fat*i;
  return fat;
}
```

Figura 1.1 - Calculando o fatorial

```
void MostraRaiz(float i, float N){
  float raiz;
  raiz = pow(N,(1/i));
  printf("%3.2f \n", raiz);
  getch();
}
```

Figura 2.1 - Calculando a raiz de um número

```
int SomaPares(){
  int n, j, Par, Soma=0;
  Par = 2;
  printf("Digite a quantidade de pares: ");
  scanf("%d", &n);
  for(j=1; j<=n; j++){
     Soma = Soma + Par;
     Par = Par + 2;
  }
  return Soma;
}
```

Figura 3.1 - Calculando a soma dos *n* primeiros pares

```
void ExibeImpares(){
  int num=1;
  while(num != 0){
     printf("Digite um numero: ");
     scanf("%d", &num);
     if(num % 2 != 0)
        printf("%d \n", num);
  }
  getch();
}
```

Figura 4.1 - Mostra apenas os ímpares digitados

```
long int Fatorial(int N){
  int i;
  long int fat=1;
  for(i=1; i<=N; i++)
     fat = fat*i;
  return fat;
}
```

Figura 1.1 - Calculando o fatorial

```
void MostraRaiz(float i, float N){
  float raiz;
  raiz = pow(N,(1/i));
  printf("%3.2f \n", raiz);
  getch();
}
```

Figura 2.1 - Calculando a raiz de um número

```
int SomaPares(){
  int n, j, Par, Soma=0;
  Par = 2;
  printf("Digite a quantidade de pares: ");
  scanf("%d", &n);
  for(j=1; j<=n; j++){
     Soma = Soma + Par;
     Par = Par + 2;
  }
  return Soma;
}
```

Figura 3.1 - Calculando a soma dos *n* primeiros pares

```
void ExibeImpares(){
  int num=1;
  while(num != 0){
     printf("Digite um numero: ");
     scanf("%d", &num);
     if(num % 2 != 0)
        printf("%d \n", num);
  }
  getch();
}
```

Figura 4.1 - Mostra apenas os ímpares digitados

```
tipo Função1(tipo1 par1, tipo2 par2, tipo3 par3...);
{
  /* Declaração de variáveis locais */
  ...
  ...
  return (valor);
 }
```

Figura 1 - Função com retorno e com parâmetros

```
void Funcao2(tipo1 par1, tipo2 par2, tipo3 par3,...);
{
  /* Declaração de variáveis locais */
  ...
  ...
  ...
}
```

Figura 2 - Função com parâmetros e sem com retorno

```
tipo Funcao3();
{
  /* Declaração de variáveis locais */
  //Obtém dados para processar
  ...
  ...
  return (valor);
}
```

Figura 3 - Função com retorno e sem parâmetros

```
void Funcao4();
{
  /* Declaração de variáveis locais */
  //Obtém dados para processar
  ...
  ...
  ...
}
```

Figura 4 - Função sem retorno e sem parâmetros