Funções

Uma função é um agrupamento de linhas de código que juntas realizam uma tarefa. Todo programa em C tem pelo menos uma função que se chama `main()`.

A declaração de uma função informa o compilador do nome da função, de seu tipo de retorno e de seus parâmetros. A definição de uma função é o trecho do código que possui corpo da função propriamente dito.

A Biblioteca Padrão de C (*C Standard Library*) possui várias funções pré-definidas que seu programa pode chamar. Por exemplo, `strcat()` para concatenar duas cadeias de caracteres, `memcpy()` para copiar uma região da memória para outro lugar, e muitas outras.

Definindo uma função

De maneira geral, a deifinição de uma função em C é feita da seguinte maneira.

```
tipo_de_retorno  nome_da_função(  lista de parâmetros  ) {
   corpo da função
}
```

A definição de uma função em C conciste do cabeçalho e do corpo. O corpo da função é o bloco que contém as linhas de código que definem o comportamento da função (o que ela faz). A seguir, veja uma lista dos elementos que compõem o cabeçalho de uma função.

- Tipo de retorno – Uma função pode retornar um valor. O tipo_de_retorno é o tipo de dado do valor que a função retorna. Funções que realizam tarefas mas não retornam nenhum valor são chamadas de procedimentos e seu tipo_de_retorno deve ser a palavra-chave void.
- Nome da função – Deve seguir as mesmas regras que o nome de uma variável: consiste de uma sequência de letras maiúsculas ou minúsculas ou underscore _ ou dígitos (de 0 a 9) sendo que não pode começar com um dígito.
- Lista de parâmetros − Define o número e o tipo dos dados que a função admite receber como argumento. Quando uma função é chamada, um valor é associado a cada um dos parâmetros. Como parâmetros são opcionais, a lista de parâmetros pode ser vazia.

Exemplo

Abaixo está o código-fonte de uma função chamada max que recebe dois parâmetros num1 e num2 e que retorna o máximo dos dois.

```
/* função que devolve o máximo de dois números */
int max(int num1, int num2) {

   /* declaração de variável local */
   int resultado;

   if (num1 > num2)
      resultado = num1;
   else
      resultado = num2;

   return resultado;
}
```

Declaração de funções

A declaração de uma função informa o compilador do nome da função e de como ela deve ser chamada. O corpo da função pode ser definido separadamente.

A declaração de uma função tem a seguinte forma.

```
tipo_de_retorno  nome_da_funcao(  lista de parametros  );
```

A função `max()`, tem a seguinte declaração.

```
int max(int num1, int num2);
```

Os nomes dos parâmetros não são importantes (para o compilador) quando a função está sendo apenas declarada. Por isso, a seguinte linha de código também é uma declaração válida para a função max().

```
int max(int, int);
```

Às vezes é necessário declarar uma função em um lugar diferente de sua definição. Por exemplo, quando definimos uma função num arquivo `.c` e desejamos chamar essa função de um ou mais arquivos `.c` diferentes. Nesse caso, em cada arquivo `.c` que deseja chamar a função devemos colocar a declaração dessa função antes de chamá-la.

Chamando uma função

Quando criamos uma função, devemos definir o que a função faz. Quando usamos uma função, é preciso chamá-la para que ela execute a tarefa programada.

Quando um programa chama uma função, o controle da execução é transferido à função chamada. Ela então executa tarefas determinadas até que o comando `return` é executado ou a chave no fim da função `}` é atingida. Nesse ponto, o controle da execução é devolvido ao programa principal.

Para chamar uma função, você precisa passar os parâmetros necessários junto com nome da função e, se a função retorna um valor, você pode armazená-lo. Por exemplo:

```
#include <stdio.h>

/* declaração de função */
int max(int num1, int num2);

int main () {

   /* definição de variáveis locais */
   int a = 100;
   int b = 200;
   int ret;

   /* chamando uma função para obter o valor máximo */
   ret = max(a, b);

   printf( "Max value is : %d\n", ret );

   return 0;
}

/* função que devolve o máximo de dois números */
int max(int num1, int num2) {

   /* declaração de variável local */
   int result;

   if (num1 > num2)
      result = num1;
   else
      result = num2;

   return result;
}
```

```
Max value is: 200
```

Argumentos de uma função

Se uma função aceita argumentos, deve haver declarações de variáveis que irão armazenar os valores passados como argumentos. Essas variáveis são chamadas de os parâmetros formais da função.

Parâmetros formais se comportam como qualquer outra variável local dentro da função. Elas são criadas assim que o controle de execução passa para a função e são destruídas quando o controle de execução sai da função.

	Tipo de chamada e descrição
1	**Chamada por valor** Esse método copia o valor corrente de um argumento dentro do parâmetro formal. Neste caso, mudanças feitas de dentro da função ao parâmetro não afetam o argumento.
2	**Chamada por referência** Este método copia o endereço de memória de um argumento dentro dos parâmetros formais. De dentro da função o endereço é usado para acessar o argumento que, de fato, foi usado na chamada da função.

Por padrão, C usa passagem de parâmetros por valor (*call by value*) para passar seus argumentos. Isso quer dizer que o código dentro (do corpo) de uma função não pode alterar os argumentos usados para chamar a função.

Fonte: https://www.tutorialspoint.com/cprogramming/c_functions.htm (https://www.tutorialspoint.com/cprogramming/c_functions.htm)

Uso de funções

Funções são utilizadas para organizar melhor o código. O nome de cada função deve ser escolhido de modo a representar o que ela faz. Vejamos um exemplo de definição de função em C.

```
double distance(double x1, double y1, double x2, double y2) {
    double dx = x1 - x2;
    double dy = y1 - y2;

    double dist = sqrt(dx * dx + dy * dy);

    return dist;
}
```

Demos o nome de `distance` à função acima porque ela calcula a distância do ponto (x1, y1) ao ponto (x2, y2). O tipo de dado que precede o nome da função indica o tipo de dado que a função retorna. Nesse caso, ela retorna um `double`.

Duas situações são indicativas de que você deveria estar usando uma função e não está:

1. seu programa possui vários trechos de código muito semelhantes, com pequenas diferenças nos dados;
2. seu programa possui uym trecho de código muito grande e complexo, com vários blocos `for` ou blocos `if` encaixados.

Trechos de código repetitivos devem ser "fatorados" e a parte comum deve ser transformada no corpo de uma função. Portanto, para eliminar repetição de código você deve criar uma função que encapsule a parte comum dos vários trechos e deve usar os parâmetros da função (quantos forem necessários) para codificar a os dados que variam de um trecho para o outro.

No segundo caso, tente quebrar seu programa em unidades que completam alguma tarefa. Cada tarefa deve ser relegada para uma função e o código que antes fazia aquela tarefa deve ser transportado (com as necessárias alterações) para o corpo da função.

Escopo de variáveis

A região de um programa em que uma variável pode ser usada é chamada de escopo da variável. Em C, o escopo de uma variável se inicia assim que ela é declarada e termina quando se fecha a chave `}` do bloco de código onde a variável foi declarada ou até o fim do programa se sua declaração está fora de todos os blocos de código. Neste último caso, diz-se que a variável é global. Veja um exemplo.

```
#include <stdio.h>

int z = -3;

int main()
{
    int x = 10;

    printf("Hello, World! z = %d\n", z);
    z = 20;
    printf("Hello, World! z = %d\n", z);

    printf("Hello, World! x = %d\n", x);

    return 0;
}
```

```
-3
20
10
```

No exemplo acima, `z` é uma variável global e ainda poderá ser usada (para leitura ou escrita) dentro de qualquer outra função que venha a ser definida depois da função `main()`. Por outro lado, `x` é uma variável local.

Podemos definir blocos de código em qualquer lugar do programa. Variáveis declaradas num bloco mais interno têm seu escopo reduzido. Se houver duas declarações de variáveis com o mesmo nome, uma num bloco mais interno e outra num bloco mais externo, será considerada a declaração mais interna possível. Veja o exemplo abaixo.

```
#include <stdio.h>

int main()
{
    int x = 10;
    printf("Hello, World! x = %d\n", x);

    {
        int x = 0;
        printf("Hello, World! x = %d\n", x);
    }

    printf("Hello, World! x = %d\n", x);

    return 0;
}
```

```
10
0
10
```

C89 x C99

No padrão ANSI C ou ISO 90 ou C89 da linguagem C, que será referido como padrão C89 daqui para frente, as declarações de variáveis feitas dentro de um bloco de código devem vir logo no início do bloco. Contudo, a comunidade de programadores considera uma boa prática de programação declarar e definir cada variável o mais próximo possível de seu primeiro uso.

O padrão C89 também proíbe declaração de variáveis dentro do for. Com C99 isso já é possível (e preferível). Veja:

```
for (int i = 0; i < n; i ++) {
    printf("i = %d; ", i);
    printf("i quadrado = %d\n", i * i);
}
```

Voltar
(https://tidia4.ufabc.edu.br/portal/tool/9d8ce88f-f802-4b32-b5e4-b39c5d9aaece/ShowPage?errorMessage=&clearAttr=&newTopLevel=false&recheck=&itemId=6909&id=&addTool=-1&title=&source=&studentItemId=0&backPath=pop&path=0&sendingPage

Próxima
(https://tidia4.ufabc.edu.br/portal/tool/9d8ce88f-f802-4b32-b5e4-b39c5d9aaece/ShowPage?false&recheck=&itemId=6909&id=&addTool=-1&title=&source=&studentItemId=0&backPath=&path=0&sendingPage=2133&postedComment=false&returnView=)