Programação Estruturada Aula 13 - Funções

Prof. Luis Carlos Retondaro

Técnico em Telecomunicações
2^o Ano

CEFET/RJ - Centro Federal de Educação Tecnológica Celso Suckow da Fonseca

Campus Petrópolis

2017

Sumário

Introdução à funções

- É absolutamente indispensável que um bom programador domine completamente a escrita de programas de forma modular;

Definição

Conjunto de comandos agrupados em um bloco que recebe um nome associado e através deste nome pode ser realizada a execução desses comandos.

```
scanf("%d",&x);

x = sqrt(4);
```

Introdução à funções

Porque usar funções?

- Para permitir o reaproveitamento de código já construído (por você ou por outros programadores);
- Para evitar que um trecho de código que seja repetido várias vezes dentro de um mesmo programa;
- Para permitir a alteração de um trecho de código de uma forma mais rápida;

Introdução à funções

Porque usar funções?

- Para que os blocos do programa não fiquem grandes demais e, por consequência, mais difíceis de entender;
- Para facilitar a leitura do programa-fonte;
- Para separar o programa em blocos que possam ser logicamente compreendidos de forma isolada.

Introdução à funções

- **Exemplo:** Escrever um programa que coloque na tela a seguinte saída:

```
********************
Numeros entre 1 e 5
********************
1
2
3
4
5
********************
```

Introdução à funções

```c
#include<stdio.h>

main(){
  int i;

  //escrita do cabecalho
  for(i=0; i< 20; i++)
    putchar('*');
  putchar('\n');

  puts("Numeros entre 1 e 5");

  for(i=0; i< 20; i++)
    putchar('*');
  putchar('\n');

  //escrita dos numeros
  for(i=1; i<=5; i++)
    printf("%d\n",i);

  for(i=0; i< 20; i++)
    putchar('*');
  putchar('\n');
}
```

Introdução à funções

- O conjunto de código utilizado para escrever uma linha de asteríscos na tela aparece repetido três vezes;
- O ideal seria escrever um trecho de código apenas uma única vez e invocá-lo sempre que necessário;
- O objetivo da programação modular é dividir o programa em pequenos trechos de código, sendo que cada um é responsável por uma tarefa;
- A utilização de funções não isenta o programa de ter a função main(), independente da variedade e do número de funções implementadas.

Introdução à funções

- **Exemplo:** Escrever o programa anterior usando função.

```
#include<stdio.h>

linha(){
  int i;

  for(i=0; i< 20; i++)
    putchar('*');
  putchar('\n');
}

main(){
  int i;

  linha();  //escreve uma linha de asteríscos
  puts("Numeros entre 1 e 5");
  linha();

  for(i=1; i<=5; i++)
    printf("%d\n",i);

  linha();
}
```

Introdução à funções

- No código apresentado o programa possui duas funções no mesmo arquivo;
- A Função main() é responsável por iniciar o programa e executar todas as instruções presentes em seu interior;
- A função linha() é responsável por escrever uma linha de asteríscos na tela, assim sempre que pretende-se escrever uma linha basta invocar a função linha(), evitando escrever sempre todo o código que esta executa;
- A principal vantagem dessa abordagem é que se precisarmos modificar a linha a ser impressa (* -> –) será necessário alterar apenas uma vez no código (dentro da função).

Introdução à funções

- Declarando uma função:

```
tipo nome_da_funcao(lista_de_parametros){
   instrucoes
 }
```

- O tipo especifica o tipo de dado que a função retorna;
- A lista de parâmetros é uma lista de tipos e nomes de variáveis separadas por vírgula. Esses parâmetros recebem o valor do argumento quando a função é chamada;

Introdução à funções

- Declarando uma função:

```
tipo nome_da_funcao(lista_de_parametros){
   instrucoes
 }
```

- Quando uma função não exige parâmetros então a lista é vazia, porém os parênteses ainda são necessários;

- Na declaração de variáveis do mesmo tipo, todas podem ser colocadas na mesma linha e separadas por vírgulas, em contrapartida todos os parâmetros de função devem incluir o tipo e o nome da variável.

Introdução à funções

- O fato de uma função existir em um programa não quer dizer que ela será executada;
- É através da função main() (ou qualquer outra função invocada pela main()) que os serviços de determinada função devem ser solicitados;
- A chamada da função ocorre escrevendo o nome da função com os respectivos parênteses (e parâmetros se a função exigir);
- As variáveis declaradas dentro de um bloco, são locais a esse bloco, não sendo conhecidas fora dele.

Características de uma função

- Cada função tem que ter um nome único, pelo qual será invocada;
- Pode ser invocada a partir de outras funções;
- Uma função (como o próprio nome indica) deve realizar uma única tarefa bem definida;
- Deve se comportar como uma caixa preta, não interessa como funciona, o que interessa é que o resultado final seja o esperado.

Características de uma função

- Seu código deve ser o mais independente possível do resto do programa;
- O código deve ser tão genérico quanto possível para poder ser usado em outros projetos;
- Pode retornar para a entidade que a invocou um valor como resultado de seu trabalho.

Nome de uma função

- As regras para escolha de nomes de funções são as mesmas para nomes de variáveis (já vistas);
- O nome deve ser único, não pode ser igual ao nome de outra função ou de outra variável;
- O nome deve especificar o que a função faz na realidade, de fácil leitura e interpretação.

Funcionamento de função

- O código da função só é executado a partir do momento em que ela é chamada;
- Uma função é chamada através de seu nome;
- Sempre que uma função é invocada, o programa que invoca é suspenso, temporariamente, e as instruções do corpo da função chamada é executado.

Funcionamento de função

- Quando a execução da função termina, o controle da execução do programa retorna para o local onde a função foi envocada;
- O programa que invoca a função pode enviar argumentos, que são recebidos pela função, armazenados em variáveis locais da função (parâmetros);
- Ao terminar a execução uma função pode devolver um valor (ou não) para o programa que a invocou.

Funcionamento de função

- **Escreva um programa com a seguinte saída**;

```
***
*****
*******
*****
***
```

Funcionamento de funções

- **Exemplo:** Escreva um programa utilizando funções para gerar a seguinte saída:

```
***
*****
*******
*****
***
```

- **Solução 1:**

```
#include <stdio.h>
void linha3x(){
  int i;
  for(i=0;i<3; i++)
    putchar('*');
  putchar('\n');
}

//continua...
```

Funcionamento de funções

- **Solução 1:**

```
//continuacao...

void linha5x(){
  int i;
  for(i=0;i<5; i++)
    putchar('*');
  putchar('\n');
}

void linha7x(){
  int i;
  for(i=0;i<7; i++)
    putchar('*');
  putchar('\n');
}
```

Funcionamento de funções

- **Solução 1:**

```
//continuacao...

void main(){
  linha3x();
  linha5x();
  linha7x();
  linha5x();
  linha3x();
}
```

Funcionamento de funções

Solução 1: comentários

- Utilizamos quatro funções: linha3x(), linha5x(), linha7x() e main();
- Os números nos nomes da funções não tem nenhum significado especial, só auxiliam na leitura do programa;
- O caracter * (asterisco) não é permitido para identificadores, por isso foi usado o caracter x;
- As três funções linhas são iguais exceto as linhas do for que controlam a quantidade de * serão impressos;
- O ideal seria tornar a função mais genérica e ter somente uma função e não três para a impressão dos *.

Funcionamento de funções

- **Solução 2:**

```
#include <stdio.h>
void linha(int n){
  int i;
  for(i=0;i<n; i++)
    putchar('*');
  putchar('\n');
}
void main(){
  linha(3);
  linha(5);
  linha(7);
  linha(5);
  linha(3);
}
```

- Com o programa generalizado pode-se imprimir quantos asteríscos quiser, basta passar como parâmetro a quantidade desejada.

Funcionamento de funções

- **Exemplo com retorno:** Soma dois valores passados como parâmetro.

```
#include <stdio.h>
float soma1(float n1, float n2){
   float soma;
   soma = n1+n2;
   return(soma);
}

//alternativa 2
float soma2(float n1, float n2){
   return(n1+n2);
}

void main(){
   float a, b, soma;

   printf("entre com dois valores\n");
   scanf("%f %f",&a, &b);
   soma = soma1(a, b);
   printf("Resultados \nsoma1 %f \t soma2 %f \n", soma, soma2(7, 2.3));
}
```

Funcionamento de funções

Perguntas que deve-se responder ao criar uma função

- Qual tarefa a função vai realizar?
- Qual o melhor nome para identificar a tarefa?
- Quantos parâmetros vai receber e quais os tipos?

Regras de escopo de funções

- São as regras que governam se uma porção de código conhece ou tem acesso a outra porção de código ou dados;
- O código de uma função é privativo àquela função e não pode ser acessado por nenhum comando em uma outra função, exceto por meio da chamada à função;
- **Exemplo:** Não pode-se usar o comando goto para saltar para o meio de outra função;
- O código situado no corpo de uma função é escondido do resto do programa, menos que se use variáveis globais, não pode ser afetado ou afetar outras partes do programa.

Regras de escopo de funções

- O código e os dados que são definidos internamente em uma função não podem interagir com o código ou dados definidos em outra função, pois as duas tem escopos diferentes;
- As variáveis definidas internamente a uma função são chamadas de variáveis locais;
- Uma variável local de uma função começa sua existência quando a instrução de sua declaração é executada e é destruída quando a função é encerrada;
- Variáveis locais não mantêm seus valores entre uma chamada e outra da função.

Variáveis Locais X variáveis Globais

Var. locais

Uma variável é chamada local se ela foi declarada dentro de uma função. Nesse caso, ela existe somente dentro daquela função e após o término de sua execução a variável deixa de existir.

Var. globais

Uma variável é chamada global se ela for declarada fora de qualquer função. Essa variável existe dentro de todas as funções e qualquer função pode ter acesso a mesma inclusive alterá-la.

Regras de escopo de funções

Exemplo:

```
#include <stdio.h>
//variável global
int num;

int func(int a, int b){
  a = (a+b)/2;
  num -= a;
  return(a);
}

main(){
  int first = 0, sec = 50;
  num = 10;
  num += func(first, sec);
  printf("\n\nConfira! num = %d\tfirst = %d\tsec = %d",num, first, sec);
}
```

Parâmetros e argumentos de funções

- A comunicação com uma função é feita através dos argumentos que lhe são enviados e dos parâmetros presentes no cabeçalho da função;
- Os parâmetros são as variáveis locais definidas no cabeçalho da função, tais variáveis recebem os valores enviados pela função que a chamou;
- Os argumentos são os valores passados para a função no momento de sua chamada;
- O número de parâmetros de uma função depende apenas da necessidade de informação para realizar a tarefa para a qual a função foi programada.

Parâmetros e argumentos de funções

- Qualquer tipo de dados da linguagem pode ser enviado como parâmetro para uma função, mesmo os tipos de dados que venham a ser definidos pelo programador;
- Os parâmetros de uma função são separados por vírgula e é absolutamente necessário que para cada um deles seja indicado seu tipo.

```
funcao(int x, char y, float k, int h){
...
}
```

Parâmetros e argumentos de funções

- A passagem de argumentos para uma função deve ser realizada colocando-os dentro de parenteses, separados por vírgulas, imediatamente após o nome da função;
- Quando se faz a chamada de uma função, o número e o tipo dos argumentos enviados devem ser coincidentes com os parâmetros presentes no cabeçalho da função;

```
main(){
...
funcao(3, 'A',123.45, 0);
}
```

- Qualquer expressão válida em C pode ser envida como argumento para uma função.

Parâmetros e argumentos de funções

Generalizando ainda mais...

```
#include <stdio.h>
void linha(int n, char ch){
  int i;

  for(i=0;i<n; i++)
    putchar(ch);
  putchar('\n');
}

void main(){
  linha(10, '-');
  printf("Operadores\n");
  linha(10, '-');
  linha(1,'+');
  linha(1,'-');
  linha(1,'*');
  linha(1,'/');
  linha(1,'%');
  linha(10, '-');
}
```

Parâmetros e argumentos de funções

Generalizando ainda mais - Resultado da execução

```
_____
Operadores
_____
+
-
*
/
%
_____
```

Notas

1. O cabeçalho de uma função NUNCA deve ser seguido de ponto-e-vírgula!
2. Em C não se pode definir funções dentro de funções.

Argumentos para a main()

- É possível passar informações para a função main() através da linha de comando no momento da execução do programa;
- Um argumento na linha de comando é a informação que segue o nome do executável.

```
./operacoes.e 5 + 3
```

Argumentos para a main()

- Existem dois parâmetros especiais, que são usados para receber os argumentos na linha de comando: argc e argv;
- O argc armazena o número de argumentos da linha de comando e é um inteiro. O argv é um ponteiro para uma matriz que armazena cada argumento passado na linha de comando;
- Todos os argumentos da linha de comando são strings;

Argumentos para a main()

- **Exemplo:**

```
#include <stdio.h>
#include <stdlib.h

void main(int argc, char *argv[]){

  int num1, num2;
  if(argc != 3){
     printf("Quantidade de argumentos inválidos");
     exit(1);
 }
 num1 = atoi(argv[1]);
 num2 = atoi(argv[2]);
 printf("%d + %d = %d\n",num1, num2, num1+num2);
}
```

```
./argc_argv.e 5 2
5 + 2 = 7
```

Funções que retornam valor

- A instrução return permite terminar a execução de uma função e voltar ao programa que a invocou;
- Em alguns programas uma função pode ser responsável por realizar uma determinada tarefa e no final necessite devolver um resultado;
- A devolução de um resultado é feita através da instrução return, seguida do valor a ser devolvido.

Funções que retornam valor

Exemplo:

```
#include <stdio.h>

int soma(int x, int y){

   int resp;
   resp = x + y;
   return(resp);
}

int dobro(int x){
   return(2*x);
}

void main(){
   int num1, num2, total;

   printf("Entre com dois numeros inteiros\n");
   scanf("%d%d", &num1, &num2);
   total = soma(num1, num2);
   printf("%d + %d = %d\n",num1, num2, total);
   printf("2 * %d = %d\n",dobro(num1));
}
```

Funções que retornam valor

Resultado da execução

```
Entre com dois numeros inteiros
2 5
2 + 5 = 7
2 * 2 = 4
```

Nota

Uma função pode ter várias instruções de return, porém, apenas uma instrução return é executada.

Localização das funções

- Em C as funções podem ser colocadas em qualquer lugar do arquivo, antes ou depois de serem invocadas, antes ou depois da main();

```
#include <stdio.h>
void main(){
  linha(10, '-');
  printf("Operadores\n");
  linha(10, '-');
  linha(1,'+');
  linha(1,'-');
  linha(1,'*');
  linha(1,'/');
  linha(1,'%');
  linha(10, '-');
}
void linha(int n, char ch){
  int i;

  for(i=0;i<n; i++)
    putchar(ch);
  putchar('\n');
}
```

Localização das funções

- Resultado da compilação

```
gcc funcaDepoisMain.c -o funcaDepoisMain.e

funcaDepoisMain.c:17:6: warning: conflicting types for linha [enabled by
    default]

funcaDepoisMain.c:6:3: note: previous implicit declaration of linha was
    here
```

Localização das funções

- Embora o código anterior esteja formalmente correto, ocorrerá um erro de compilação;
- O processo de compilação é em geral um processo sequencial, que percorre todo o arquivo em que o código se encontra, do princípio até o fim, fazendo a verificação sintática, estando tudo correto gera o executável;
- A função linha() está definida após a main() e sua chamada ocorre na main() e no momento que a função main() é compilada o compilador ainda não conhece o cabeçalho da função linha().

Localização das funções

- Sem conhecer o cabeçalho o compilador cria um protótipo para a função linha():

```
int linha();
```

- Quando o compilador encontra a função linha() (com outro cabeçalho) então "*pensa*" que ela foi definida duas vezes, então informa o erro e encerra o processo de compilação.

Localização das funções

- A solução seria indicar ao compilador qual é o cabeçalho da função;
- A declaração de uma função consiste na escrita do seu cabeçalho seguida de um ponto e vírgula e deve ser realizada antes da chamada da função;
- Tradicionalmente, a declaração é realizada imediatamente após os *#includes*.

Localização das funções

```
#include <stdio.h>

void linha(int n, char ch);

void main(){
  linha(10, '-');
  printf("Operadores\n");
  linha(10, '-');
  linha(1,'+');
  linha(1,'-');
  linha(1,'*');
  linha(1,'/');
  linha(1,'%');
  linha(10, '-');
}

void linha(int n, char ch){
  int i;

  for(i=0;i<n; i++)
    putchar(ch);
  putchar('\n');
}
```

Referências

1. C Completo e Total, Herbert Schidt; Pearson Makron Books; 3a. Ed., 1997.
2. Linguagem C, Luís Damas, LTC, 10a. Ed.2014.