Estruturas de Dados

1. Subrotinas (Procedimentos e funções)

Uma subrotina é um conjunto de comandos agrupados para realizar alguma tarefa específica, em um módulo, e recebe um nome, que pode ser chamado em diferentes partes do programa.
Algumas linguagens separam conceitualmente as subrotinas em dois tipos:
a) Procedimentos, que não retornam nenhum valor para o ponto onde foi chamado;
b) Funções, que retornam um valor. Esse valor deve ser armazenado em uma variável do mesmo tipo no ponto onde foi chamado.

A linguagem C trata todas as subrotinas como funções.

A função é referenciada pelo programa principal através do nome atribuído a ela.

A utilização de funções visa modularizar um programa, o que é muito comum em programação estruturada.

Desta forma podemos dividir um programa em várias partes, no qual cada função realiza uma tarefa bem definida.

Esqueleto de uma função

```
tipo_de_retorno nome_da_função (listagem de parâmetros)
{
   instruções;
   retorno_da_função;
}
```

1.1 Parâmetros de uma função

Os parâmetros são as variáveis declaradas diretamente no cabeçalho da função.

A finalidade dos parâmetros é fazer a comunicação entre as funções e a função principal.

Chamamos de passagem de parâmetros a passagem de valores entre as funções.

Exemplo de código com uma função.

```
#include <stdio.h>

int duplica(int x) // Funcao que duplica o valor de x, inteiro
{
     x=x*2;
     printf("Mensagem dentro da função: x = %d\n",x);
     return(x);
}

int main()
{
     int a;
```

```
    int resultado;

    a = 50;
    resultado = duplica(a);
    // Imprime o valor de a e o valor retornado de duplica
    printf("Resultado = %d, a = %d\n",resultado, a);
    a = 60;
    resultado = duplica(a);
    // Imprime o valor de a e o valor retornado de duplica
    printf("\na = %d, resultado = %d \n",a, resultado);
    a=80;
    resultado = duplica(a);
    // Imprime o valor de a e o valor retornado de duplica
    printf("\na = %d, resultado = %d \n",a, resultado);
    return(0);
}
```

Exemplo 1: Código de programa com chamada de uma função que duplica um número informado.

Observe que a função *duplica* recebe um parâmetro inteiro *x*, e retorna outro valor inteiro.
No programa principal, o valor de retorno deve ser atribuído a uma variável do mesmo tipo da função, neste caso, um valor inteiro, atribuído à variável *resultado*. Por isso, a variável resultado deve ser do tipo *int*.
Neste programa, a função é chamada três vezes. O programa inicia atribuindo o valor 50 para a variável *a* (que deve ser do mesmo tipo do parâmetro da função, no caso, *int*). Em seguida, chama a função *duplica*, passando como parâmetro a variável *a* (nesse caso, com o valor 50). A função *duplica* recebe esse valor no parâmetro *x* (que tem o mesmo comportamento de uma variável interna da função). Na primeira chamada, o valor associado a *x* é 50. Na segunda, 60 e na terceira, 80.

1.2 Protótipo

Nessa apresentação, a função precisa ser definida antes de ser usada (chamada).
Mas a linguagem C permite que seja definido um protótipo da função, definindo somente o tipo, o nome e a listagem de parâmetros, deixando para definir os comandos da função mais tarde.
Para esse exemplo, o uso com protótipo segue abaixo:

```
#include <stdio.h>

int duplica(int x); // Funcao que duplica o valor de x, inteiro

int main()
{
    int a;
    int resultado;

    a = 50;
    resultado = duplica(a);
    // Imprime o valor de a e o valor retornado de duplica
    printf("Resultado = %d, a = %d\n",resultado, a);
    a = 60;
```

```
     resultado = duplica(a);
     // Imprime o valor de a e o valor retornado de duplica
     printf("\na = %d, resultado = %d \n",a, resultado);
     a=80;
     resultado = duplica(a);
     // Imprime o valor de a e o valor retornado de duplica
     printf("\na = %d, resultado = %d \n",a, resultado);
     return(0);
}

int duplica(int x) // Funcao que duplica o valor de x, inteiro
{
     x=x*2;
     printf("Mensagem dentro da função: x = %d\n",x);
     return(x);
}
```

Exemplo 2: Código de programa com chamada de uma função que duplica um número informado, usando o protótipo de função.

1.3 Função sem retorno

Em C, é possível criar funções que não retornam nenhum valor.

Normalmente, isto é feito quando queremos executar um bloco de comandos, mas estes comandos não precisam retornar nada.

Neste caso, devemos usar `void` no tipo de retorno do cabeçalho da função.

Se a função não recebe nenhum parâmetro, também colocamos `void` no local da listagem dos parâmetros, ou deixamos simplesmente sem nada entre parêntesis “`()`”.
Veja o exemplo, extraído de (http://linguagemc.com.br/funcoes-em-c/):

```
void imprime_cabec(void)
{
  printf("*****************************\n");
  printf("*       LINGUAGEM C          *\n");
  printf("*****************************\n");

  return; /* retorno de uma função void  */
}
```

Exemplo 3: Código de função sem parâmetros

Para funções do tipo void, (ou seja sem retorno) devemos usar o comando return;

Note bem que este return não contém nenhum parâmetro.

Já que não é preciso retornar nenhum parâmetro e se eu não escrever o comando return?

Se a instrução return for omitida, alguns compiladores vão gerar mensagens de advertência (warning), pois o padrão ANSI C recomenda a utilização da instrução.

```
#include<stdio.h>

/******* Área dos protótipos ******/
void imprime_cabec(void);
int multiplica(int N1, int N2);
/******* fim dos protótipos *******/

/* ******* FUNÇÃO PRINCIPAL *******/
int main(void)
{
    int V1=0, V2=0, resultado=0;
    //Chamada da função imprime_cabec
    imprime_cabec();

    printf("Digite o primeiro valor:");
    scanf("%d", &V1);
    printf("Digite o segundo valor:");
    scanf("%d", &V2);

    //chama a função e recebe o retorno
    resultado = multiplica(V1,V2);
    printf("Resultado = %d\n", resultado);

    return 0;
}
/* ******** FIM DA FUNÇÃO PRINCIPAL *******/

/* --------- Corpo das funções --------- */
/* ***** Função imprime_cabec *****
   Parâmetros de entrada: não tem (void)
   Objetivo: imprimir cabeçalho na tela
   Tipo de retorno: não tem (void)
 */
void imprime_cabec(void)
{
    printf("*****************************\n");
    printf("* LINGUAGEM C *\n");
    printf("*****************************\n\n");

    return;/* retorno de uma função void */
}

/* ***** Função multiplica *****
   Parâmetros de entrada: N1, N2 ambos int
   Objetivo: multiplicar valores
   Tipo de retorno: int (resultado);
 */
//multiplica recebe N1,N2 e retorna um int
int multiplica(int N1, int N2)
{
    int resultado;
    resultado = N1 * N2;
```

```
    //retornando o valor para main
    return(resultado);
}
```

Exemplo 4: Programa chamando uma função sem parâmetros e sem valor de retorno, e outra com parâmetros e valor de retorno, extraído de (http://linguagemc.com.br/funcoes-em-c/).

Neste programa utilizamos duas funções, uma delas com retorno de valor multiplica() e outra delas sem retorno imprime_cabec().

Primeiramente, declaramos todos os protótipos das funções antes da main.

Na função main() são feitas as chamadas para as outras funções.

Após a finalização do código da main(), são implementados os corpos das outras funções.

Quando a função main() chama (invoca) outra função, o programa "salta" para o trecho de código da função que foi chamada, executa esta função e depois retorna novamente para a função principal main().

1.4 Vantagens do uso de funções.

A programação usando funções permite separar trechos de código que atuam em tarefas específicas, organizando melhor o código.

Outra vantagem é a possibilidade de chamar várias vezes a mesma função, sem precisar reescrever todo o código (que pode ser complexo).

Uma terceira vantagem, derivada da segunda, é a facilidade de manutenção do código. Considere um trecho de código que é necessário em 10 pontos diferentes ao longo do programa principal (por exemplo, o cálculo do dígito verificador do CPF). Caso em algum momento haja uma alteração na forma de calcular o dígito verificador do CPF, se o código para isso estiver copiado nos 10 locais necessários, essa alteração deverá ser feita em todos os 10 trechos diferentes. Caso algum deles seja esquecido, em algum momento o programa poderá calcular o dígito verificador do CPF de maneiras diferentes, ocasionando um erro de programação. Se o código estiver separado em uma função, e chamado 10 vezes ao longo do código, no caso de uma alteração do código, basta alterar a função, em um único ponto.

1.5 Escopo de variáveis (locais e globais)

Variáveis locais.

Toda função permite a criação de variáveis dentro da função. Essas variáveis são chamadas de variáveis locais, e só são conhecidas dentro da função. No caso da função `multiplica`, do exemplo 4, a variável `resultado` é uma **variável local**, e só é conhecida dentro da função. Se houver outra variável chamada `resultado`, no programa principal, elas são variáveis diferentes, pois estão definidas em escopos diferentes, apesar de serem chamadas pelo mesmo nome. Nesse exemplo, há uma variável chamada `resultado` no programa principal também.

Vimos que uma função pode receber **parâmetros**. Esses parâmetros tem o mesmo comportamento de variáveis que só são conhecidas dentro da função, e portanto **tem o mesmo comportamento de variáveis locais**, com a diferença que recebem um valor informado pelo programa principal.

Quando uma função é encerrada, o conteúdo de todas as variáveis locais é perdido.

Variáveis globais.

É possível também definir variáveis cujo valor é conhecido (e pode ser alterado) a partir de qualquer parte do programa. Essas variáveis são chamadas de variáveis globais.

A definição de uma variável global é feita fora do escopo das funções, e ela é conhecida em todo o código que está abaixo dela.

O exemplo 5 ilustra o mesmo programa do exemplo 4, mas definindo a variável resultado como global.

```
#include<stdio.h>

/******* Área dos protótipos ******/
void imprime_cabec(void);
int multiplica(int N1, int N2);
/******* fim dos protótipos *******/

int resultado;

/* ******* FUNÇÃO PRINCIPAL *******/
int main(void)
{
    int V1=0, V2=0;
    //Chamada da função imprime_cabec
    imprime_cabec();

    printf("Digite o primeiro valor:");
    scanf("%d", &V1);
    printf("Digite o segundo valor:");
    scanf("%d", &V2);

    //chama a função e recebe o retorno
    resultado = multiplica(V1,V2);
    printf("Resultado = %d\n", resultado);

    return 0;
}
/* ******** FIM DA FUNÇÃO PRINCIPAL *******/

/* --------- Corpo das funções --------- */
/* ***** Função imprime_cabec *****
   Parâmetros de entrada: não tem (void)
   Objetivo: imprimir cabeçalho na tela
   Tipo de retorno: não tem (void)
 */
void imprime_cabec(void)
{
    printf("*****************************\n");
    printf("* LINGUAGEM C *\n");
    printf("*****************************\n\n");

    return;/* retorno de uma função void */
}
```

```
/* ***** Função multiplica *****
   Parâmetros de entrada: N1, N2 ambos int
   Objetivo: multiplicar valores
   Tipo de retorno: int (resultado);
 */
//multiplica recebe N1,N2 e retorna um int
int multiplica(int N1, int N2)
{
    resultado = N1 * N2;

    //retornando o valor para main
    return(resultado);
}
```

Exemplo 5: Adaptação do exemplo 4 mudando a variável resultado de local para global.

1.6 Passagem de Parâmetros (por valor e por referência)

Existem dois métodos de passagem de parâmetros para funções:

Passagem por valor – permite usar dentro de uma função uma cópia do valor de uma variável, porém não permite alterar o valor da variável original (somente a cópia pode ser alterada).

Passagem por referência – É passada para a função uma referência da variável, sendo possível alterar o conteúdo da variável original usando-se esta referência.

Na linguagem C a passagem por referência é implementada com o uso de ponteiros.

Usando ponteiros (também chamados de apontadores) é possível alterar os valores das variáveis passadas como argumentos para uma função.

A função utilizada dessa forma é denominada função com **passagem por referência**. Nesse tipo de função, os argumentos passam os endereços de memória para os parâmetros declarados na função. Sendo assim, os parâmetros que recebem os valores passados obrigatoriamente tem que ser ponteiros já que irão receber um endereço de memória.

Quando usamos o comando `scanf("%d", &numero);` estamos informando que será feita uma leitura de um número inteiro, em formato decimal, e o resultado será colocado na posição de memória indicada pelo endereço `&numero`. E o sinal `&` antes do nome de uma variável indica que no lugar de usar o valor indicado por essa variável, estamos nos referindo ao endereço de memória onde o conteúdo da variável é armazenado. Resumindo, "`&`" indica o endereço de uma variável.

Quando queremos indicar o valor que está em uma certa posição de memória e temos somente esse endereço, usamos o caracter "`*`" (asterisco). Por exemplo, se uma variável `x` contém um endereço de memória, e queremos saber o valor armazenado na posição de memória indicada por `x`, usamos `*x`.

O exemplo 6 apresenta o uso de parâmetros por valor, na função soma10, e por referência, na função soma10p. Observe que após a execução da função soma10, o valor da variável número do programa principal continua inalterado. Já após a execução da função soma10p, o valor da variável

número do programa principal (cujo endereço é recebido pelo parâmetro x da função soma10p) é alterado.

```
#include <stdio.h>

//função que soma 10 ao valor recebido
void soma10(int x)
{
  x = x + 10;
  printf("Valor de x apos a soma = %d \n",x);
  return;
}

void soma10p(int *x)
{
  *x = *x + 10;
  printf("Valor de x apos a soma = %d \n",*x);
  return;
}

int main(void)
{
  int numero;
  printf("Digite um numero: ");
  scanf("%d", &numero);

  printf("O numero digitado foi: %d \n",numero);

  soma10(numero);  //chamada da função
  printf("Agora o numero vale: %d \n",numero);

  soma10p(&numero);  //chamada da função com ponteiro como
parâmetro
  printf("Agora o numero vale: %d \n",numero);
  return 0;
}
```

Exemplo 5: Chamadas de funções com passagem de parâmetros por valor e referência (extraído de http://linguagemc.com.br/funcao-com-passagem-por-referencia/)

Resultado de uma execução do exemplo 6.

```
Digite um numero: 45
O numero digitado foi: 45
Valor de x apos a soma = 55
Agora o numero vale: 45
Valor de x apos a soma = 55
Agora o numero vale: 55
```