Aula 15

MC 102 - Algoritmos e Programação de Computadores

Procedimentos e Funções.

Procedimentos

Algumas vezes, utilizamos comandos que realizam operações mais complexas, mas que para nós representa apenas um único comando. Por exemplo:

```
scanf("%d", &n);
printf("Voce digitou %d\n", n);
```

Estes são exemplos de procedimentos.

Procedimentos são estruturas que agrupam um conjunto de comandos a serem executados quando o procedimento for chamado.

Funções

Outras vezes, alguns comandos mais complexos possuem uma função específica, retornando um determinado resultado. Por exemplo:

```
x = sqrt(n);
t = pow(p, x);
n = strlen(str);
```

Estes são exemplos de funções.

Funções são procedimentos que retornam um único valor ao final de sua execução.

Vantagens de Uso

- Evitar que blocos de programa fiquem grande demais e, conseqüentemente, mais difíceis de entender;
- Separar o programa em partes que possam ser logicamente compreendidas de forma isolada;
- Permitir reaproveitamento de código, evitando "copy / paste" de trechos bem similares;
- Evitar que trechos idênticos sejam repetidos inúmeras vezes, minimizando os erros e facilitando alterações.

Mas como posso definir essas estruturas e como utilizá-las em meus programas?

Definindo uma função (1)

A definição de uma função consiste de duas partes: o cabeçalho e o corpo.

Cabeçalho:

```
tipo_dado identificador(tipo parâmetro, ...) {
    ...
}
```

tipo_dado determina qual será o tipo do resultado a ser retornado. Toda função tem um.

parâmetros são os "valores" a serem passados para a execução da função

Exemplo:

```
int soma(int a, int b) {
    ...
}
```


Definindo uma função (2)

Corpo da função - Comandos:

```
int soma(int a, int b) {
    int res;

    res = a + b;
    ...
}
```

No corpo da função definimos o conjunto de comandos que serão executados, quando a função for chamada, e também suas variáveis internas (aquelas que serão usadas apenas pela função).

Definindo uma função (3)

Corpo da função - Resultado:

```
int soma(int a, int b) {
    int res;

    res = a + b;
    return res;
}
```

Após a execução dos comandos da função, algum dado será retornado. Usa-se o comando **return** para retornar o dado e finalizar a função. Sempre que um return for encontrado, a função termina e nenhum outro comando após ele será executado.

Uma função especial: a **main**

A função **main** é a função principal do programa, onde definimos o corpo do programa. Ela é definida por meio de um cabeçalho já conhecido:

```
int main(int argc, char **argv) {
    ...
}
```

Finalmente vamos entender o que representa o cabeçalho:

`int main(int argc, char **argv)`
Define o tipo de dado que será retornado

`int main(int argc, char **argv)`
Nome ou identificador para a função

`int main(int argc, char **argv)`
São os parâmetros (e seus tipos de dado) que serão fornecidos para a execução da função; ou seja, os valores que a função poderá utilizar quando necessário

Uma função especial: a **main**

Quando utilizado, o comando **`return`** informa ao sistema operacional se o programa funcionou corretamente ou não. O padrão é que um programa retorne zero caso tenha funcionado corretamente ou qualquer outro valor caso contrário.

Exemplo:

```
int main() {
    printf("Hello, World!\n");
    return 0;
}
```


Definindo uma função (4)

- Uma função pode não ter parâmetros, basta não informá-los.
- As funções só podem ser declaradas fora de outras funções.
- Toda função deve ser declarada antes de seu uso, i.e., de sua chamada. Logo, todas as funções devem ser definidas antes da função main.
- Ela poderá ser chamada em qualquer trecho seguinte do seu programa, por exemplo, dentro da função main.

Usando uma função

Uma forma clássica de realizarmos a invocação (ou chamada) de uma função é atribuindo o seu valor a uma variável:

```
x = soma(4, 2);
```

Na verdade, o resultado da chamada de uma função é uma expressão e pode ser usada em qualquer lugar que aceite uma expressão:

```
printf("Soma de a e b: %d\n", soma(a, b));
```

Em especial, a função main() é invocada automaticamente pelo sistema operacional quando este inicia a execução do programa.

Exemplo

Programa:

```
#include <stdio.h>

int soma(int a, int b) {
    int res;

    res = a + b;
    return res;
}

int main(int argc, char **argv) {
    int x, y, s;
    printf("Informe 2 numeros inteiros: ");
    scanf("%d %d", &x, &y);
    s = soma(x, y);
    printf("A soma %d + %d e %d\n", x, y, s);
}
```

Função x Procedimento

A principal diferença entre função e procedimento é que este último **não retorna valor**. Portanto, o cabeçalho de um procedimento difere do cabeçalho de uma função apenas na definição do tipo de dado retornado:

```
void identificador(tipo parâmetro, ...) {
    ...
}
```

O tipo de dado **void** é um tipo especial utilizado para definir procedimentos na linguagem C, pois representa o nada. Como nenhum dado será retornado, o procedimento pode terminar com um "**return;**" a qualquer momento.

Invocação de Procedimento

Como não retorna valor, a invocação de um procedimento é semelhante à utilização de qualquer outro comando, ou seja:

procedimento (parametros);

Exemplo:

```
printf("Informe 2 numeros inteiros: ");
scanf("%d %d", &x, &y);
```


Usando um procedimento

Programa:

```
#include <stdio.h>

void soma(int a, int b) {
    int res;

    res = a + b;
    printf("A soma %d + %d e %d\n", a, b, res);
    return;
}

int main(int argc, char **argv) {
    int x, y;
    printf("Informe 2 numeros inteiros: ");
    scanf("%d %d", &x, &y);
    soma(x, y);
}
```


Usando uma função - Passos

- cada parâmetro é testado em relação ao tipo
- o valor passado como parâmetro é atribuído às respectivas variáveis da declaração;
- o corpo da função é executado
- Se o comando return é executado, o controle de execução é devolvido para o trecho de programa que invocou a função
  - Caso o return inclua uma expressão, o valor da expressão é convertido para o tipo específico da função e o valor é retornado
- Se não houver return, o controle de execução é devolvido para o trecho de programa que invocou a função assim que o fim da função for encontrado

Processo

```
#include <stdio.h>

int soma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main(int argc, char **argv) {
    int x, y, s;
    printf("Informe 2 numeros inteiros: ");
    scanf("%d %d", &x, &y);
    s = soma(x, y);  //São int?
    printf("Informe 2 numeros inteiros: %d.", s);
}
```


Processo

```
#include <stdio.h>

int soma(int a, int b) { // a = x; b = y
    int res;
    res = a + b;
    return res;
}

int main(int argc, char **argv) {
    int x, y, s;
    printf("Informe 2 numeros inteiros: ");
    scanf("%d %d", &x, &y);
    s = soma(x, y);
    printf("Informe 2 numeros inteiros: %d.", s);
}
```


Processo

```
#include <stdio.h>

int soma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main(int argc, char **argv) {
    int x, y, s;
    printf("Informe 2 numeros inteiros: ");
    scanf("%d %d", &x, &y);
    s = soma(x, y);
    printf("Informe 2 numeros inteiros: %d.", s);
}
```


Processo

```
#include <stdio.h>

int soma(int a, int b) {
    int res;
    res = a + b;
    return res;
}

int main(int argc, char **argv) {
    int x, y, s;
    printf("Informe 2 numeros inteiros: ");
    scanf("%d %d", &x, &y);
    s = soma(x, y);
    printf("Informe 2 numeros inteiros: %d.", s);
}
```

Escopo de Variáveis

Variáveis podem ser definidas tanto na função **main** quanto em outras **funções** e **procedimentos** definidos no seu programa. Mas como elas se comportam?

Dizemos que as variáveis possuem um escopo e elas só podem ser utilizadas naquele escopo. Isso quer dizer que elas existem apenas durante a execução daquele trecho de programa. Segundo seu escopo elas podem ser:

- Locais: válidas dentro de uma função ou procedimento (otimizam o uso da memória).
- Globais: declarada fora de qualquer função. Neste caso tornam-se visíveis a todo o programa e durante toda sua execução.

Variáveis Locais

```
#include <stdio.h>

int soma(int a, int b) {
   int res;

   res = a + b;
   return res;
}

int main(int argc, char **argv) {
   int x, y, s;
   printf("Informe 2 numeros inteiros: ");
   scanf("%d %d", &x, &y);
   s = soma(x, y);
   printf("A soma %d + %d e %d\n", x, y, s);
}
```

A variável **res** existe apenas na função soma. Os parâmetros **a** e **b** são considerados como variáveis internas da função, e também estão restritos ao escopo local.

Variáveis Locais

```
#include <stdio.h>

int soma(int a, int b) {
   int res;

   res = a + b;
   return res;
}

int main(int argc, char **argv) {
   int x, y, s;
   printf("Informe 2 numeros inteiros: ");
   scanf("%d %d", &x, &y);
   s = soma(x, y);
   printf("A soma %d + %d e %d\n", x, y, s);
}
```

As variáveis **x**, **y** e **s** existem apenas na função main, restritas ao escopo local, e não podem ser referenciadas fora desta função.

Variáveis Globais

```
#include <stdio.h>

int s;

void soma(int a, int b) {
   s = a + b;
}

int main(int argc, char **argv) {
   int x, y;
   printf("Informe 2 numeros inteiros: ");
   scanf("%d %d", &x, &y);
   soma(x, y);
   printf("A soma %d + %d e %d\n", x, y, s);
}
```

A variável **s** está agora definida como global e existe em todo programa. Ela pode ser referenciada em qualquer parte do programa, por qualquer função ou procedimento.

Programa Exemplo

Refaça o programa que calcule o MDC de dois números positivos, para também calcular o MMC desses números

MDC(A, B) – Usar algoritmo de Euclides

MMC(A, B) = A*B/MDC(A, B)

Exemplo

Calculo do mdc(a,b) e mmc(a,b) para a, b>0.

```
#include <stdio.h>

int main(int argc, char **argv) {
  unsigned int a, b, resto, c;

  printf("Informe a e b inteiros positivos: ");
  scanf("%u %u", &a, &b);
  if (a*b == 0) {
    printf("Valores invalidos.\n");
    return 0;
  }
  c = a * b;
  while (b > 0) {
    resto = a % b;
    a = b;
    b = resto;
  }
  printf("MDC = %u\n", a);
  printf("MMC = %u\n", c/a);

  return 0;
}
```


Exemplo – Usando função

Calculo do mdc(a,b) e mmc(a,b) para a, b>0.

```
#include <stdio.h>
unsigned int mdc(unsigned int a, unsigned int b) {
  unsigned int resto;

  while (b > 0) {
    resto = a % b;
    a = b;
    b = resto;
  }
  return a;
}
int main(int argc, char **argv) {
  unsigned int a, b, m;

  printf("Informe a e b inteiros positivos: ");
  scanf("%u %u", &a, &b);
  if (a*b == 0) {
    printf("Valores invalidos.\n");
    return 0;
  }
  m = mdc(a, b);
  printf("MDC = %u\n", m);
  printf("MMC = %u\n", a*b/m);

  return 0;
}
```