Algoritmos e Programação B

Modularização

- parte 2 -

Funções

- Já trabalhamos...
  - Declaração
  - Chamada

- Vamos formalizar:
  - Retorno de valores de funções
  - Passagem de valores às funções

- Vamos aprender:
  - Variável Local e Variável Global
  - Protótipo de funções

Retorno de valores de funções

- Uma função retorna à função chamadora quando termina sua execução.
- Neste retorno, a função pode devolver à função chamadora **um valor:**
  - int, float, double, char ou algum tipo de dado heterogêneo.

Retorno de valores de funções

- O retorno de um valor de uma função pode acontecer antes de se chegar ao finalizador da função: }
  - Isso acontece se for usado o `return` em qualquer ponto da função.

- Quando o `return` é usado, a função é terminada e o controle volta à função chamadora, independente da existência de outros comandos não executados.

Retorno de valores de funções

```
1  void funcao1() {
2     printf(“1”);
3     return ;
4  }
5
6  void funcao2() {
7     return ;
8     printf("%d", 100);
9  }
10
11 void main(void) {
12    printf ("10");
13    funcao1();
14    printf ("20");
15    funcao2();
16    printf ("30");
17    return ;
18 }
```

- O valor a ser retornado deve ser colocado junto ao **return**.
- O tipo do dado retornado deve ser identificado antes do nome da função.

Retorno de valores de funções

```
int funcao1() {
    return 1;
}
int funcao2() {
   return 2;
}
void main(void) {
   int a;
   printf ("10");
   a=funcao1();
   printf(“%d”, a);
   printf ("20");
   printf(“%d”,funcao2());
   printf ("30");
}
```

Retorno de valores de funções

- Funções que não retornam valores não podem ser utilizadas em expressões ou atribuições.

- A declaração `return` pode ocorrer mais de uma vez em uma mesma função.
  - Isto significa que uma função pode ter mais de um ponto de retorno. Como no exemplo, da função que retorna 0 quando o número é par ou 1, quando o número é ímpar.

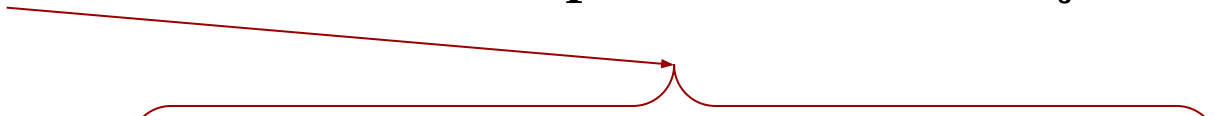
Exemplo 1

- Escreva uma função para verificar se um determinado ano é bissexto. Os anos bissextos são anos com um dia a mais, tendo, portanto 366 dias. O dia extra é introduzido como o dia 29 de fevereiro, ocorrendo a cada quatro anos. No final do século XVI foi introduzido o calendário Gregoriano, usado até hoje na maioria dos países, adotando as seguintes regras:
    - Todo ano divisível por 4 é bissexto
    - Todo ano divisível por 100 não é bissexto
    - Mas se o ano for também divisível por 400 é bissexto

- As últimas regras prevalecem sobre as primeiras.

Passagem de valores às funções (lista de parâmetros)

- No momento da chamada de uma função, é possível enviar valores para a função, para serem utilizados no processamento.
- Estes valores são chamados de argumentos.
- A especificação da quantidade e do tipo dos argumentos é feita na **lista de parâmetros** colocada após o nome da função, entre parênteses:



```
tipo nome_função(tipo1 nome1, tipo2 nome2, ..., tipoN nomeN){
    //Corpo da função
    return ;
}
```

Passagem de valores às funções (lista de parâmetros)

- Os **parâmetros da função** são variáveis criadas quando a função é chamada.
  - Quando a função termina, essas variáveis são eliminadas. Por isso, são chamadas de **variáveis locais**.
  - São variáveis “visíveis” apenas na função e existem enquanto a função estiver sendo executada.
  - Essas variáveis são inicializadas com os valores passados como argumentos (os argumentos são informados na chamada da função)

Passagem de valores às funções (lista de parâmetros)

- Os **parâmetros da função** são variáveis criadas quando a função é chamada.
  - Quando a função termina, essas variáveis são eliminadas. Por isso, são chamadas de **variáveis locais**.
  - São variáveis "visíveis" apenas na função e existem enquanto a função estiver sendo executada.
  - Essas variáveis são inicializadas com os valores passados como argumentos (os argumentos são informados na chamada da função)

Argumento: valor passado para a função

Parâmetro: variável que recebe este valor

Passagem de valores às funções (lista de parâmetros)

- Os argumentos podem ser constantes, variáveis, expressões ou outras funções que retornem valores.
- Por exemplo:

```c
#include<stdio.h>

int funcao2(int x) {
   return x + 2;
}

int f(int x, char y) {
   printf("\nOs valores passados foram: %d, %c", x, y);
}

void main(void) {
   int a=50;
   char b='B';
   f(49, 'A'); /* constantes */
   f(a, b); /* variáveis */
   f(a+1, b+1); /* expressões */
   f(funcao2(a), b+2); /* função e expressão */

   return;
}
```

Passagem de valores às funções (lista de parâmetros)

- As funções que não recebem valores devem ter sua lista de parâmetros substituída pela palavra reservada **`void`**, como no caso da função **`main`**, ao lado.

```c
#include<stdio.h>

int funcao2(int x) {
   return x + 2;
}

int f(int x, char y) {
   printf("\nOs valores passados foram: %d, %c", x, y);
}

void main(void) {
   int a=50;
   char b='B';
   f(49, 'A'); /* constantes */
   f(a, b); /* variáveis */
   f(a+1, b+1); /* expressões */
   f(funcao2(a), b+2); /* função e expressão */

   return;
}
```

Variável Local e Variável Global

- As **variáveis locais** podem ser declaradas dentro de qualquer bloco de código
  - Os símbolos abre ‘{‘ e fecha ‘}’ chaves delimitam um bloco de comandos.

- No momento em que o bloco termina, a variável é desalocada da memória (eliminada), não sendo mais possível utilizá-la.

- Assim, a variável só pode ser referenciada dentro do próprio bloco (seu **escopo** é limitado ao **bloco**).

Variável Local e Variável Global

- **Variáveis globais** são aquelas criadas fora de qualquer função, mas antes do seu primeiro uso.

- Seu **escopo** é **todo o programa**, podendo ser acessadas e alteradas por qualquer comando, em qualquer função.

- **Importante:**
  - Se uma variável local receber o mesmo nome de uma variável global, todas as referências ao nome tem relação com a variável local, no bloco de código em que ela foi declarada.

Variável Local e Variável Global

- **Regras de escopo** definem quais dados podem ser acessados por cada parte do programa.
  - Definem quais partes do código podem acessar cada variável.

Há, basicamente, três tipos de variáveis:
Parâmetros: variáveis inicializadas no início da função com os valores dos atributos (também são locais).
Locais: variáveis declaradas dentro de um bloco de comandos.
Globais: variáveis declaradas fora das funções, mas antes do seu primeiro uso (declaradas antes da main()).

Variável Local e Variável Global

- A utilização de variáveis locais ou parâmetros ao invés de variáveis globais é altamente recomendável:

  - Variáveis globais ocupam a memória durante toda a execução do programa, mesmo quando não estão sendo usadas.
  - Variáveis globais aumentam a complexidade do programa.
  - Variáveis locais proporcionam aumentar a generalidade das funções e favorecem o reuso de funções.
  - Variáveis locais possibilitam economia de memória, pois o escopo das variáveis é apenas um bloco de código e não todo o programa.

Protótipo de funções

- O compilador, ao encontrar uma função, conhece as suas características, como o nome, tipo do retorno e a lista de parâmetros, pois essas estão definidas conforme a sintaxe da linguagem.

- Uma forma de garantir isso, é declarar e especificar as funções a serem chamadas, antes das funções que as invocam, ou seja, até o momento, declaramos e especificamos funções antes da função `main()`.

Protótipo de funções

- Com o uso de protótipo, é possível ter a função `main()` como primeira função especificada em um programa.
- O protótipo de uma função deve ser declarado após o `#include<....>` ou `#define...` e tem o mesmo formato da 1ª linha da declaração da função, seguido pelo ponto e vírgula.
- Sintaxe do Protótipo de funções:

```
tipo nome_da_função (tipo nome1, tipo nome2, ... , tipo nomeN);
```

Exemplo de Protótipo de funções

```c
#include<stdio.h>

float volume(float a);
float perimetro(float a);
float area(float a);

void main(void) {
   float aresta;
   printf("Digite a aresta do cubo: ");
   scanf("%f", &aresta);
   printf("Volume do cubo = %f\n", volume(aresta));
   printf("Perimetro da base do cubo = %f\n", perimetro(aresta));
   printf("Area da base do cubo = %f\n", area(aresta));
   return;
}
```

Exemplo de Protótipo de funções

```c
#include<stdio.h>

float volume(float a);
float perimetro(float a);
float area(float a);

void main(void) {
   float aresta;
   printf("Digite a aresta do cubo: ");
   scanf("%f", &aresta);
   printf("Volume do cubo = %f\n", volume(aresta));
   printf("Perimetro da base do cubo = %f\n", perimetro(aresta));
   printf("Area da base do cubo = %f\n", area(aresta));
   return;
}
```

Protótipos de funções

Exemplo de Protótipo de funções - continuação

```
float volume(float a){
        return a*a*a;
}

float perimetro(float a){
        return a+a+a+a;
}

float area(float a){
        return a*a;
}
```