Laboratório de Algoritmos I
Funções e Pré-processador C

Função - conceito

- Uma função é um conjunto de instruções desenhadas para cumprir uma tarefa particular e agrupadas numa entidade com um nome para referenciá-la.

Chamando uma função

- Chamar uma função é o meio pelo qual solicitamos ao programa para desviar o controle e passar a executar as instruções da função, e que ao término desta volte o controle para a posição seguinte à da chamada.

Funções pré-definidas

- Em nossos programas já utilizamos diversas funções, tais como printf, scanf, system, getch, strcpy, strcmp, gets, puts, etc.

- A declaração “int main()” é justamente uma função. Esta função é, por padrão, a primeira função do programa a ser executada quando chamamos o executável do programa. A partir dela é que as demais funções (pré-definidas e definidas pelo programador) são chamadas.

Definição de uma função

```
float celsius(float fahr) {
    float c;
    c = (fahr - 32.0) * 5.0/9.0;
    return c;
}
```

Definição de uma função

- Esta função de nome `celsius` é uma função do tipo `float`, o que significa que ela retorna um valor do tipo `float`.
- Ela tem apenas um parâmetro, também do tipo float.
- O comando `return` serve para sair da função, retornando o valor para o local de chamada da função.

Funções - parâmetros

- Os parâmetros são dados que podem ser passados para a função. Dentro da função, os parâmetros são usados da mesma forma que usamos variáveis.

- Parâmetros podem ser de qualquer tipo:
  - int func(int x) – parâmetro tipo int
  - int func(float x) - parâmetro tipo float
  - int func(char c) - parâmetro tipo char
  - int func(char c[]) ou int func(char c[10]) - parâmetro tipo string

- Para passar matrizes como parâmetros:
  - void func(int matriz[10][10]);
  - void func(int matriz[][]) **é errado**!

Protótipo de uma função

- Deve ser declarado antes da definição e antes das chamadas à função. Serve para informar ao compilador a existência da função, seu tipo e seus parâmetros. Exemplo:

```
float celsius(float);
```

Protótipo de uma função

- O protótipo não é obrigatório, desde que a função seja definida antes da função que a chama.

Chamando a função

- Dentro da função main, por exemplo, poderia ser colocado o código abaixo, que atribui à variável c o valor retornado pela função:

```
c = celsius(f);
```

Tipo void - procedimento

- Quando a função não retorna nenhum valor (dizemos que é um procedimento), seu tipo é `void`. Abaixo um exemplo de protótipo de um procedimento que recebe um `float` como parâmetro:

```
void procedimento(float);
```

Função sem parâmetros

- Quando a função não tem parâmetros, seu protótipo deve incluir a palavra void no lugar dos parâmetros:

```
int funcaoteste(void);
```

Função com mais de um comando return

- Uma função pode ter mais de um comando return. Quando o controle alcança um comando return, ele sai da função, retornando para o local de chamada o valor passado ao comando return.

Chamada a Função sendo usada como parâmetro para outra função

- Uma chamada a uma função pode ser usada como parâmetro para outra função, desde que o tipo da função (tipo do retorno) seja igual ao tipo do parâmetro. Exemplo:

```
printf("Celsius = %.2f\n", celsius(f));
```

Funções recursivas

- Uma função pode chamar a si mesma.
- São chamadas de funções recursivas.

- Exemplo: Calcular a soma de 1 até n, usando recursão:

```
int sum(int n) {
  if (n == 1) {
    return 1;
  } else {
    return n + sum(n-1);
  }
}
```

O Pré-Processador C

- É um programa que examina o programa-fonte em C e executa certas modificações nele antes da compilação, com base em instruções chamadas *diretivas*.
- O pré-processador faz parte do compilador e é executado automaticamente antes da compilação.

O Pré-Processador C

- As diretivas são geralmente usadas para tornar o programa-fonte mais claro e fácil de manter.
- Elas podem ser escritas em qualquer lugar do programa (geralmente no início) e se aplicam somente do ponto onde são escritas até o final do código.

A diretiva #include

- Causa a inclusão de outro arquivo em nosso programa-fonte. Na verdade, o compilador substitui a diretiva #include de nosso programa pelo conteúdo do arquivo indicado, antes de compilar o programa.

A diretiva #include

- Quando usamos a diretiva #include com os sinais < e >, o arquivo é procurado somente na pasta include.
- Quando usamos aspas duplas, o arquivo é procurado primeiro na pasta atual e depois na pasta include.

A diretiva #define

- Na sua forma mais simples, é usada para definir constantes com nomes apropriados. Exemplo:

```
#define PI 3.14
```

- Onde Pi é o **identificador** e 3.14 é o **texto**.

A diretiva #define

- Por convenção, o identificador é sempre escrito em letras maiúsculas.
- Não há substituição dentro de cadeias de caracteres. Ou seja, se o código tiver, por exemplo, a palavra PIANO, ela continuará inalterada.
- Observe que não há ponto-e-vírgula após nenhuma diretiva do pré-processador.

Macros

- É quando utilizamos a diretiva #define com parâmetros. Exemplo:

```
#define PRN(n) printf("%.2lf\n", n)
```

- Obs.: nunca deve haver espaço em branco no identificador da macro. Exemplo:

```
define PRN (n) errado!
```

Macros

- Chamando a macro:

```
double n1 = 10;
PRN(n1);
```

Cuidados com Macros

- Considere a seguinte macro e sua chamada:

```
#define SOMA(x,y) x + y

z = 10 * SOMA(3,4);
```

- Qual é o valor atribuído a z ?

Cuidados com Macros

- Após a expansão da macro teremos:

```
z = 10 * 3 + 4;
```

- Como o operador * tem maior prioridade que o operador +, o resultado será 30 + 4 = 34.

Cuidados com Macros - solução

- Colocar o texto da macro entre parênteses:

```
#define SOMA(x,y) (x + y)

z = 10 * SOMA(3,4);
```

- Neste caso, qual é o valor atribuído a z ?

Cuidados com Macros - solução

- Após a expansão da macro teremos:

```
z = 10 * (3 + 4);
```

- Neste caso a operação entre parênteses será realizada primeiro, então teremos z = 10 * 7 = 70.

Cuidados com Macros - solução

- Para evitar problemas deste tipo em qualquer situação, o ideal é sempre envolver cada parâmetro com parênteses. O nosso exemplo ficaria desta forma:

```
#define SOMA(x,y) ((x) + (y))
```

Definindo macros a partir de outras macros

- Exemplo:

```
#define AREACIRCULO(r) (PI)*((r)*(r))
```

Funções versus Macros

- Macros são mais rápidas na execução e não é necessário definir os tipos dos parâmetros.
- No entanto o código do programa será maior, visto que o código da macro será duplicado a cada chamada da mesma.