Professor
Edson Pedro FERLIN

Programação em C

Funções

Prof. Edson Pedro Ferlin

Professor
Edson Pedro FERLIN

Programação em C

Definições

- Definição: São blocos de construção em que ocorrem todas as atividades dos programas;

```
especificador_tipo  nome_função (lista_parâmetros)
declarações de parâmetros
{
   corpo do programa
}
```

- O especificador de tipo especifica o tipo do valor que a função devolverá através do uso de return.
- O valor pode ser qualquer tipo válido;

Retorno de uma Função
Parte 1 - Executa o último comando

```
Impr_inverso(char *s)
{
   int t;
   for (t=strlen(s)-1;t>=0;t--) printf ("%c", s[t]);
}
```


Retorno de uma Função
Parte 2 - Comando Return

```
power (int base, int exp)
{
   int i;
   if (exp<0) return (0);          /* retorna se exp negativa */
   i = 1;
   for (;exp;exp--) i=base*i;
   printf ("A resposta eh: %d", i);
}
```

Valores de Retorno

```
main()
{
   int x,y,z;
   x=10;
   y=20;
   z = mult(x,y);
   printf ("%d", mult (x,y));
   mult (x,y);
}
mult (int a, int b)
{
   return (a*b);
}
```

- Todas as funções, exceto aquelas que você declara como sendo do tipo **void**, devolvem um valor;
- Utiliza-se a palavra reservada (return).

Escopo das Funções
Parte 1 - Visão Geral

- Governam se um trecho de código conhece ou não ou tem acesso ou não a um outro trecho de código ou dados;
- Três tipos:
  - Variáveis locais;
  - Variáveis globais;
  - Parâmetros formais.

Escopo das Funções
Parte 2 - Variáveis Locais

- Variáveis que são declaradas dentro de uma função;
- Podem ser referenciadas apenas pelos comandos que estão dentro do bloco no qual estão declaradas;
- Existem apenas durante a execução do bloco de código no qual estão declaradas; i.e., uma variável local é criadas quando se entra em seu bloco e destruída na saída;
- Vantagem: O armazenamento para as variáveis locais está na pilha (região dinâmica da memória);

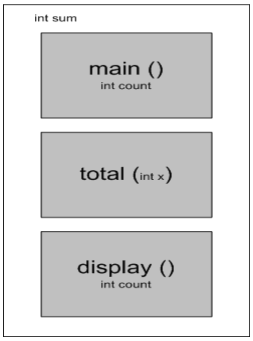
Escopo das Funções
Parte 3 - Parâmetros Formais

- Variáveis que aceitarão os valores dos argumentos;
- Se comportam como qualquer outra variável local dentro da função;

Escopo das Funções
Parte 4 - Variáveis Globais

- São conhecidas por todo o programa e podem ser usadas por qualquer parte o código;
- Retêm seus valores durante toda a execução do programa;
- O armazenamento das variáveis globais fica em uma região fixa da memória;
- Evitar usar variáveis globais desnecessariamente:
  - Ocupam memória durante toda a execução do programa;
  - Menor generalização;
  - Podem ocorrer erros.

Escopo das Funções
Parte 5 – Exemplo de Programa




```
int sum;                /* Variavel Global */
main ()
{
    int count;          /* Variavel Local */
    sum = 0;
    for (count = 0; count < 10; count ++)
    {
        total (count);
        display ();
    }
}
total (int x)                       /* Parametro formal */
{
   sum = x + sum;
}
display ()
{
   int count;
   for (count = 0; count < 10; count++)
   {
            printf (".");
   }
   printf ("A soma corrente eh %d\n",sum);
}
```

Argumentos das Funções
Parte 1 – Visão Geral

- Parâmetros formais das funções tende ser do mesmo tipo dos argumentos usados para chamá-las;
- Duas chamadas:
  - Por Valor → copia o **valor** de um argumento para o parâmetros formal da função;
  - Por Referência → copia o **endereço** de um argumento para o parâmetro.
- O C usa o método de chamada por Valor para passar argumentos.

Argumentos das Funções
Parte 2 – Exemplos

Por Valor

```
main ()
{
   int t=10;            /* Variavel Local */
   printf ("%d %d", pot2(t), t);
}

pot2 (int x)            /* Parametro formal */
{
   x=x*x;
   return (x);
}
```

Por Referência

```
main ()
{
   int x=10,y=20;    /* Variavel Local */
   swap (&x,&y);
}

swap (int *x, int *y) /* Parametro formal */
{
   int temo;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

Argumentos argc e argv

Parte 1 – Visão Geral

- Pode ser útil passar informações para um programa ao executá-lo;
- Argumentos de linha de comando;
- É a informação que segue o nome do programa na linha de comando do S.O;
- São os únicos argumentos que main() pode ter;

- **argc** → contém o número de argumentos na linha de comando e é um inteiro. No mínimo 1 → nome programa;
- **argv** → é um ponteiro para uma matriz de ponteiros para caracteres. Cada elemento nessa matriz aponta para um argumento da linha de comando.

Argumentos argc e argv

Parte 2 – Exemplo

```
main (int argc, char *argv[])
{
   If (argc != 2)
   {
      printf (“Voce nao digitou o nome \n”);
      exit (0);
   }
   printf (“Alo %s”, argv[1]);
}
```

No S.O:

```
C:\> nome Teste

Alo Teste

C:\>
```

Argumentos argc e argv

Parte 3 – Limites

- O separador de argumentos é um <espaço> ou um TAB;
- A maneira mais comum de declarar argv é: **char *argv[]**, em que os colchetes ([]) vazios indicam que argv é uma matriz de comprimento indeterminado;
- O número de argumentos é limitado pelo S.O., no caso do MS-DOS 128 caracteres por linha.

Funções Retornam Valores

Retornam valores não inteiros

```
float sum(float, float);   /* identifica a funcao */

main ()
{
   float prim, segun;
   prim = 123.13;
   segun = 99.09;
   printf ("%f",sum (prim, segun));
}

float sum (float a, float b)
{
   return (a+b);
}
```

- O tipo de uma função, por default, é int;
- Para tipo de dado diferente:
  - dar um especificador de tipo;
  - identificar antes de chamá-la;

Funções Retornam Valores

Funções do tipo *void*

```
void imp_vertical(char *);   /* identifica a funcao */

main ()
{
   imp_vertical ("Alo minha gente");
}

void imp_vertical(char *str)
{
   while (*str) printf("%c \n", *str++);
}
```

- Funções que não devolvem valores;
- Uso do tipo ***void***.

Funções Retornam Valores

Protótipos de Funções

```
float func (int, float);   /* prototipo */

main ()
{
   int x=10;
   float y=1.5;
   func(x,y);
}

float func (int a, float b)
{
   printf("%f", b/(float)a);
}
```

- Declarar o número e os tipos dos argumentos da função;
- Permite que o compilador emita erros se uma função com argumentos for chamada com parâmetros com tipos diferentes.

Funções Retornam Valores

Recursão

```
long factorial (int);

main()
{
    int n;
    printf("Digite o numero: ");
    scanf("%d", &n);
    printf("\nO fatorial de %d eh %ld", n, factorial(n));
}

long  factorial (int n)
{
     long resp;
     If (n==1) return (1);
     resp = factorial (n-1)*n;
     return (resp);
 }
```

- As funções podem chamar a si próprias;
- Uma função é recursiva se um comando no corpo da função chamar a si mesmo.
- O computador aloca memória na pilha e executa o código da função com essas novas variáveis.

Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.webnode.com.br

(YOUTUBE) ProfEdsonPedroFerlin