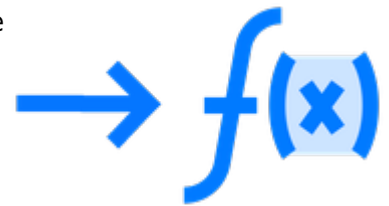


Ponteiros para funções

Video desta aula

Em C, uma função é vista como uma referência (ou endereço) para uma área de memória onde se encontra seu código. Por isso, o identificador de uma função pode ser visto como um ponteiro. Funções podem ser acessadas usando ponteiros, de forma similar às variáveis.



Declaração e uso

Declarar um ponteiro para uma função é relativamente simples, apesar da sintaxe assustar à primeira vista. O código abaixo declara um ponteiro `fp` para uma função que tem um parâmetro inteiro e não retorna nada:

```
// uma função com protótipo "void name (int)"
void f (int)
{
    ...
}

// um ponteiro para funções com protótipo "void name (int)"
void (*fp) (int) ;
```

Observe que os parênteses envolvendo `*fp` são necessários. Caso sejam omitidos, a declaração acima muda completamente de sentido:

```
void * fp (int) ; // protótipo de função retornando 'void*'
```

O uso de ponteiros para funções é simples:

[funcptr.c](#)

```
#include <stdio.h>

void inc (int *n)
{
    (*n)++ ;
}

int main ()
{
    void (*fp) (int *) ;    // function pointer

    fp = inc ;              // fp points to inc

    int a = 0 ;
    printf ("a vale %d\n", a) ;

    inc(&a) ;                // normal call
    printf ("a vale %d\n", a) ;
```

```
fp(&a) ;           // call using the function pointer
printf ("a vale %d\n", a) ;
}
```

Sua execução resulta em:

```
a vale 0
a vale 1
a vale 2
```



Obviamente, um ponteiro de função só pode apontar para funções que tenham o mesmo protótipo (assinatura) com o qual o ponteiro foi declarado.

Caso necessário, pode-se fazer *type casting* de ponteiros para funções.

Funções como parâmetros

Como uma variável pode apontar para uma função, então funções podem ser usadas como parâmetros de outras funções. O código a seguir ilustra como fazer isso:

`funcparam.c`

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

// aplica a função "func" aos caracteres de str
void aplica (int (* func) (int), char *str)
{
    for (int i = 0; str[i]; i++)
        str[i] = func (str[i]) ;
}

// se c for uma vogal, devolve '-'
int tira_vogal (int c)
{
    switch (c)
    {
        case 'a':
        case 'e':
        case 'i':
        case 'o':
        case 'u':
        case 'A':
        case 'E':
        case 'I':
        case 'O':
        case 'U': return ('-') ;
        default : return (c) ;
    }
}
```

```
}  
}  
  
int main ()  
{  
    char frase[128] ;  
  
    strcpy (frase, "Uma frase com MAIUSCULAS e minusculas") ;  
    printf ("Frase: %s\n", frase) ;  
  
    aplica (toupper, frase) ;  
    printf ("Frase: %s\n", frase) ;  
  
    aplica (tolower, frase) ;  
    printf ("Frase: %s\n", frase) ;  
  
    aplica (tira_vogal, frase) ;  
    printf ("Frase: %s\n", frase) ;  
}
```

Resultado da execução:

```
Frase: Uma frase com MAIUSCULAS e minusculas  
Frase: UMA FRASE COM MAIUSCULAS E MINUSCULAS  
Frase: uma frase com maiusculas e minusculas  
Frase: -m- fr-s- c-m m---sc-l-s - m-n-sc-l-s
```



Uma função também pode retornar um ponteiro de função. Neste caso, a sintaxe da declaração da função pode ficar complexa, sobretudo se a função tiver parâmetros e a função retornada também.

Exercícios

1. A função `qsort` (*man 3 qsort*) aplica o algoritmo *QuickSort* a um vetor de dados de um tipo definido pelo usuário (`int`, `float`, `struct`, ...). Para ser genérica, essa função depende de uma função externa para comparar os elementos do vetor. Escreva um programa que:
 - a) crie um vetor de 100 inteiros aleatórios e ordene esse vetor usando a função `qsort`.
 - b) Idem, para um vetor de tipo `double`.
 - c) Idem, para um vetor de *structs* (ordenar por um dos campos do *struct*).

From:

<https://wiki.inf.ufpr.br/maziero/> - Prof. Carlos Maziero

Permanent link:

https://wiki.inf.ufpr.br/maziero/doku.php?id=c:ponteiros_para_funcoes

Last update: **2023/08/15 11:56**

