Ponteiros

Em um programa, cada variável tem um **endereço**, que indica sua localização na memória do computador, e um **conteúdo**, que é o valor armazenado naquela posição de memória. Geralmente os valores armazenados são escalares (inteiros, reais ou caracteres) ou não-escalares (vetores, matrizes e estruturas).

Variáveis do tipo **ponteiro** armazenam **endereços** de memória de outras variáveis; em outras palavras, ponteiros são variáveis que **referenciam** outras variáveis. Ponteiros podem ser muito úteis em determinadas situações, por isso são usados extensivamente na programação em C.

Declaração

Um ponteiro é declarado através do modificador "*", seguindo esta definição:

```
  <tipo_base> * <nome_do_ponteiro> ;
```

Exemplo:

```
int   *pi ;  // ponteiro para inteiro
float *pf ;  // ponteiro para float
char  *pc ;  // ponteiro para caractere
```

No exemplo acima, "pi" é um ponteiro para inteiros que pode referenciar ("conter o endereço de" ou "apontar para") um local da memória que contém um valor do tipo "int".

Uso

Ponteiros são usados de três formas:

- **Atribuição**: pode-se atribuir um valor ao ponteiro
- **Leitura**: pode-se ler o valor armazenado no ponteiro
- **Desreferenciar**: pode-se ler conteúdo (valor) da posição de memória indicada ("apontada") pelo ponteiro.

Vejamos um exemplo mais elaborado dessas operações.

```
#include <stdio.h>

int main ()
{
  int *p ;                      // ponteiro para inteiros
  int a = 231 ;
  int b = 7680 ;

  printf ("&a vale %p\n", &a) ;  // endereço de a
  printf ("&b vale %p\n", &b) ;  // endereço de b
  printf ("&p vale %p\n", &p) ;  // endereço de p

  printf ("p vale %p\n", p) ;    // valor de p (leitura)
                                 // este é o endereço para o qual p aponta

  p = &a ;                       // atribuir valor a p
  printf ("p vale %p\n", p) ;    // ler valor de p
  printf ("*p vale %d\n", *p) ;  // desreferenciar p

  p = &b ;
  printf ("p vale %p\n", p) ;
```

```
  printf ("*p vale %d\n", *p) ;

  *p = 500 ;                    // desreferenciar p (a posição apontada por p)
  printf ("b vale %d\n", b) ;

  return 0 ;
}
```

A execução do código acima gera a seguinte saída:

```
  &a vale 0x7ffe2b852890
  &b vale 0x7ffe2b852894
  &p vale 0x7ffe2b852898

  p vale (nil)

  p vale 0x7ffe2b852890
  *p vale 231

  p vale 0x7ffe2b852894
  *p vale 7680

  b vale 500
```

A tabela a seguir mostra o conteúdo da memória em diversos momentos da execução do código:

```
momento         a=0x7ffe2b852890    b=0x7ffe2b852894   p=0x7ffe2b852898
início            231                 7680               nil
após p = &a       231                 7680               0x7ffe2b852890
após p = &b       231                 7680               0x7ffe2b852894
após *p = 500     231                  500               0x7ffe2b852894
```

Observe que os endereços das variáveis não mudam, apenas seus valores.

Ponteiros nulos

Um ponteiro nulo é aquele que aponta para nada, para nenhum endereço válido. A macro "NULL" define o valor de ponteiros nulos, que equivalem a zero (0) no C ANSI.

A tentativa de desreferenciar um ponteiro nulo resulta em erro de acesso à memória, que geralmente leva à interrupção da execução com uma mensagem de *Segmentation Fault* ou similar. Um exemplo de código contendo esse tipo de erro:

```
#include <stdio.h>

int main ()
{
  int *p ;

  // na linha abaixo, qual o valor apontado por p?
  printf ("p vale %p e *p vale %d\n", p, *p) ;

  return 0 ;
}
```

Ponteiros não-inicializados (que contém lixo) também podem levar ao mesmo tipo de erro, pois podem apontar para áreas de memória que não estão acessíveis ao programa.

Ponteiros void

Um ponteiro de tipo "void" é considerado um *ponteiro genérico*, que pode referenciar qualquer endereço de memória independente de seu tipo. Ponteiros "void" são muito usados para trasferir parâmetros

genéricos para funções, ou para construir estruturas de dados genéricas, que podem armazenar/referenciar dados de diversos tipos.

```
void *ptr ;
```

Por não terem um tipo predefinido, ponteiros "void" não podem ser desreferenciados. As operações aritméticas sobre ponteiros "void" consideram como tamanho básico 1 byte.

Exemplo:

```
#include <stdio.h>

int main ()
{
  int a = 34 ;
  int b ;
  void *p ;

  p = &a ;
  b = *p ;  // erro de compilação!

  printf ("p vale %p\n", p) ;
  p++ ;
  printf ("p vale %p\n", p) ;

  return (0) ;
}
```

Ponteiros para ponteiros

Como visto acima, um ponteiro é uma variável que pode conter o endereço ("apontar") de outras variáveis. Nada impede um ponteiro de conter o endereço de outro ponteiro, o que chamados de *referencia indireta*, *ponteiro duplo* ou *ponteiro para ponteiro*.

A declaração e uso de ponteiros indiretos é simples:

```
#include <stdio.h>

int main ()
{
  int a = 231 ;
  int *pd ;  // ponteiro direto
  int **pi ; // ponteiro indireto, equivale a int *(*p)

  pd = &a ;  // pd recebe o endereço de um int
  pi = &pd ; // pi recebe o endereço de um ponteiro para int

  printf ("a  está em %p e vale %d\n", &a, a) ;
  printf ("pd está em %p e vale %p\n", &pd, pd) ;
  printf ("pi está em %p e vale %p\n", &pi, pi) ;

  printf ("*pd  vale %d\n", *pd) ;
  printf ("*pi  vale %p\n", *pi) ;
  printf ("**pi vale %d\n", **pi) ;

  return 0 ;
}
```

O resultado da execução do código acima é:

```
     a  está em 0x7ffda4afa4ac e vale 231
    pd  está em 0x7ffda4afa4b0 e vale 0x7ffda4afa4ac
    pi  está em 0x7ffda4afa4b8 e vale 0x7ffda4afa4b0
```

```
   *pd  vale 231
   *pi  vale 0x7ffda4afa4ac
  **pi  vale 231
```

Representação gráfica

FIXME

Aritmética de ponteiros

Ponteiros são valores numéricos e portanto podem sofrer algumas operações aritméticas simples. Considerando *N* o tamanho do tipo apontado por um ponteiro (em bytes), temos:

- ++ : o valor do ponteiro é incrementado de N.
- -- : o valor do ponteiro é decrementado de N.
- ○ : somando V ao ponteiro, seu valor é incrementado de V*N.
- ○ : subtraindo V do ponteiro, seu valor é decrementado de V*N.

Além disso, ponteiros podem ser comparados (<, >, >=, <=, ==, !=, etc).

Exemplo:

```
int nota[5] = { 45, 78, 92, 73, 87 } ;
int *p ;

p = nota ; // p aponta para nota[0]
printf ("p: %p, *p: %d\n", p, *p) ;

p++ ;      // p aponta para nota[1]
printf ("p: %p, *p: %d\n", p, *p) ;

p += 3 ;   // p aponta para nota[4]
printf ("p: %p, *p: %d\n", p, *p) ;
```

Resultado da execução:

```
  p: 0x7ffd28f319a0, *p: 45
  p: 0x7ffd28f319a4, *p: 78
  p: 0x7ffd28f319b0, *p: 87
```

Ponteiros e vetores II

O uso de ponteiros pode facilitar muito a escrita de código envolvendo vetores. Um exemplo clássico é a implementação da função "strcpy (s,t)", que copia a string "t" para a string "s" (extraído e adaptado do livro *C Programming ANSI* - Kernighan & Ritchie).

A implementação vetorial simples é

```
void strcpy (char *s, char *t)
{
   int i ;
   i = 0 ;

   while (t[i] != '\0')
   {
      s[i] = t[i] ;
      i++ ;
   }
   s[i] = t[i] ;
}
```

A implementação vetorial melhorada é

```
void strcpy (char *s, char *t)
{
   int i ;
   i = 0 ;
   while ((s[i] = t[i]) != '\0')
      i++;
}
```

A versão que faz uso de ponteiros é

```
void strcpy (char *s, char *t)
{
   while ((*s = *t) != '\0')  // o valor da atribuição é o valor atribuído
   {
      s++;
      t++;
   }
}
```

Podemos mudar os operadores de incremento para dentro da condição:

```
void strcpy (char *s, char *t)
{
   while ((*s++ = *t++) != '\0') ;
}
```

Podemos retirar a comparação redundante "!= '\0'":

```
void strcpy (char *s, char *t)
{
   while (*s++ = *t++) ;   // esta versão não é recomendada por ser ilegível
}
```

Exercícios

- Assista ao vídeo Binky Pointer Fun, da Stanford University (existe uma versão com legendas em português).
- Escreva um programa que leia 10 inteiros da entrada padrão, armazene-os em um vetor e os escreva na saída padrão na ordem contrária a de leitura; todos os acessos ao vetor devem ser feitos usando somente ponteiros, sem usar índices de vetor (vet[i], etc).
- Mude o programa anterior para ordenar o vetor usando o algoritmo da bolha.
- Escreva um programa para calcular o tamanho de uma string usando somente ponteiros.
- Escreva um programa para concatenar duas strings usando somente ponteiros.