Endereços e ponteiros

Prof. Me. Hélio Esperidião.

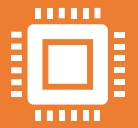
introdução

Os conceitos de *endereço* e *ponteiro* são fundamentais em qualquer linguagem de programação.

Em C, esses conceitos são explícitos;

Algumas outras linguagens eles são ocultos (e representados pelo conceito mais abstrato de *referência*).

Endereços



A memória RAM (random access memory) de qualquer computador é uma sequência de bytes.

A posição (0, 1, 2, 3, etc.) que um byte ocupa na sequência é o endereço (address) do byte.

É como o endereço de uma casa em uma longa rua que tem casas de um lado só.

Se e é o endereço de um byte então e+1 é o endereço do byte seguinte.

variável

Cada variável de um programa ocupa um certo número de bytes consecutivos na memória do computador.

Uma variável do tipo char ocupa 1 byte.

Uma variável do tipo int ocupa 4 bytes

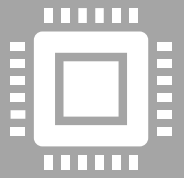
Um double ocupa 8 bytes em muitos computadores.

O número exato de bytes de uma variável é dado pelo operador sizeof.

A expressão sizeof (char), por exemplo, vale 1 em todos os computadores e a expressão sizeof (int) vale 4 em muitos computadores.

variável

Cada variável (em particular, cada registro e cada vetor) na memória tem um endereço.



Na maioria dos computadores, o endereço de uma variável é o endereço do seu primeiro byte.

memoria

Variável	posição
c	1
I	2
Ponto	3
v[0]	4
v[1]	8
v[2]	12
v[3]	16

Operadores

O endereço de uma variável é dado pelo operador & . Assim, se i é uma variável então &i é o seu endereço.

Não confunda esse uso de & com o operador lógico and, que se escreve && em C.

No exemplo acima, &i vale 2 e &v[3] vale 16.

Outro exemplo: o segundo argumento da função de biblioteca scanf é o endereço da variável que deve receber o valor lido do teclado:

```
int i;
```

```
scanf ("%d", &i);
```

Ponteiros

Um *ponteiro* (= apontador = *pointer*) é um tipo especial de variável que armazena um endereço. Um ponteiro pode ter o valor

Se um ponteiro p armazena o endereço de uma variável i, podemos dizer p aponta para i ou p é o endereço de i.

Em termos um pouco mais abstratos, diz-se que p é uma referência à variável i.

Não confunda esse operador * com o operador de multiplicação!

Tipos de ponteiros.

```
int *p;
```

```
int *p;  // p é um ponteiro para um inteiro
```

```
int *q;
```

```
p = &a;  // o valor de p é o endereço de a
```

```
q = &b;  // q aponta para b
```

```
c = *p + *q;
```

Aritmética de endereços

- Os elementos de qualquer vetor são armazenados em bytes consecutivos na memória do computador.

Exemplo

```c
#include <stdio.h>
#include <stdlib.h> //necessário para usar as funções malloc() e free()
#include <conio.h>
```

```c
int main(void)
{
    int *v;
    int i, num_componentes;
    printf("Informe o numero de componentes do vetor\n");
    scanf("%d", &num_componentes);
    v = (int *) malloc(num_componentes * sizeof(int));
```

```c
//Armazenando os dados em um vetor
for (i = 0; i < num_componentes; i++)
{
    printf("\nDigite o valor para a posicao %d do vetor: ", i+1);
    scanf("%d",&v[i]);
}
```

```
    }

    // ------ Percorrendo o vetor e imprimindo os valores ----------
    printf("\n************ Valores do vetor dinamico ************\n\n");

    for (i = 0;i < num_componentes; i++)
    {
        int *temp = &v[i];
        printf("%d -- %d \n",v[i],temp);
    }

    //liberando o espaço de memória alocado
    free(v);

    getch();
    return 0;
}
```

Exer

Compile e execute o seguinte programa:

```
int main (void) {
   typedef struct {
      int dia, mes, ano;
   } data;
   printf ("sizeof (data) = %d\n",
            sizeof (data));
   return 0;
}
```

Compile e execute o seguinte programa. (O cast (long int) é necessário para que &i possa ser impresso com especificação de formato %ld. É mais comum imprimir endereços em notação hexadecimal, usando a especificação de formato %p, que exige o cast (void *).)

```
int main (void) {
   int i = 1234;
   printf (" i = %d\n", i);
   printf ("&i = %ld\n", (long int) &i);
   printf ("&i = %p\n", (void *) &i);
   return 0;
}
```

Compile e execute o seguinte programa.

```
int main (void) {
   int i; int *p;
   i = 1234; p = &i;
   printf ("*p = %d\n", *p);
   printf (" p = %ld\n", (long int) p);
   printf (" p = %p\n", (void *) p);
   printf ("&p = %p\n", (void *) &p);
   return 0;
}
```

Analise o código abaixo

```
void troca (int *p, int *q)
{
   int temp;
   temp = *p; *p = *q; *q = temp;
}
```